GPU-Based Foreground-Background Segmentation using an Extended Colinearity Criterion

Andreas Griesser[1], Stefaan De Roeck[2], Alexander Neubeck[1], Luc Van Gool[1,2]

[1]Swiss Federal Institute of Technology (ETH), Computer Vision Lab, Zürich, Switzerland
{griesser,aneubeck,vangool}@vision.ee.ethz.ch

[2]Katholieke Universiteit Leuven, VISICS, Leuven, Belgium
{luc.vangool,stefaan.deroeck}@esat.kuleuven.ac.be

Abstract

We present a GPU-based foreground-background segmentation that processes image sequences in less than 4ms per frame. Change detection wrt. the background is based on a color similarity test in a small pixel neighbourhood, and is integrated into a Bayesian estimation framework. An iterative MRF-based model is applied, exploiting parallelism on modern graphics hardware. Resulting segmentation exhibits compactness and smoothness in foreground areas as well as for inter-frame temporal contiguity. Further refinements extend the colinearity criterion with compensation for dark foreground and background areas and thus improving overall performance.

1 Introduction

Robust and accurate foreground-background segmentation is a relatively small but crucial step in several computer vision applications. It is a key element in surveillance, 3D-modelling from silhouettes, motion capture, or gesture analysis for human-computer interaction (HCI). For several of these - surveillance and HCI are cases in point - real-time processing is crucial. Hence, for these applications, foreground-background segmentation should be extremely fast, as the bulk of the computation time on the CPU has to remain available for the subsequent stages of processing and interpretation.

As a result, the type of foreground-background segmentation that can be used on-line has typically been kept as simple as possible, and has led to important constraints on the background. For instance, in their semi on-line user modeling work, Matusik *et al.* [8] had to resort to a rather simple background subtraction. On the other hand, more sophisticated algorithms are available today, like Bayesian pixel classification based on time-adaptive, per-pixel mixture of Gaussians color model [4, 5]. A comprehensive survey of image change detection algorithms is presented in [6]. Recently, Mester *et al.* developed a color similarity criterion [1], which has already performed well in our - offline - gesture recognition setup [7]. However, these sophisticated algorithms lack real-time performance.

Here we propose a GPU-based implementation of Mester's approach, combined with some refinements to further improve performance. Our implementation takes less than 4 milliseconds per frame and frees the CPU from this preprocessing step altogether. Thus, our approach is especially useful for algorithms already using the GPU in the further processing stages.

The paper is organized as follows. Section 2 recapitulates the criterion developed by Mester and colleagues, and describes the modifications that we propose. Section 3 focuses on the GPU-based implementation and section 4 gives results on the proposed algorithm. Finally, we conclude the work in section 5 and discuss some future work.

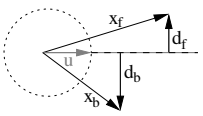
2 Mathematical Model

2.1 The Colinearity Criterion

Mester's method compares the color values at pixels in a reference (background) image, and a given image. In particular, all color values within a small window around a pixel, here always a 3×3 neighbourhood, are stacked into row vectors $\mathbf{x}_b$ resp. $\mathbf{x}_f$ for the background resp. the given image, where the

latter will typically contain some additional foreground objects. In Mester's analysis, change detection amounts to assessing whether $\mathbf{x}_b$ and $\mathbf{x}_f$ are colinear. If they are (the null hypothesis H_0), no change is judged to be present and the background is still visible at that pixel in the given image. If not, the pixels are considered to have different colors, and a foreground pixel has been found.

Rather than testing for perfect colinearity, one has to allow for some noise in the measurement process. A kind of bisector has to exist (u in the figure) to which both $\mathbf{x}_b$ and $\mathbf{x}_f$ lie close. Indeed, when Gaussian noise is assumed, the unkown 'true signal' direction u can be estimated by minimizing the sum $D^2 = |\mathbf{d}_b|^2 + |\mathbf{d}_f|^2$. By defining



$$\mathbf{X} := \begin{bmatrix} \mathbf{x}_f \\ \mathbf{x}_b \end{bmatrix} = \begin{bmatrix} r_f^1 & g_f^1 & b_f^1 \dots r_f^N & g_f^N & b_f^N \\ r_b^1 & g_b^1 & b_b^1 \dots r_b^N & g_b^N & b_b^N \end{bmatrix} \tag{1}$$

with N pixels in the neighbourhood of the considered pixel, Mester et al. [1] pointed out that the test statistic D^2 is identical to the smallest non-zero eigenvalue of the 2x2 matrix $\mathbf{X}\mathbf{X}^T$:

$$D^2 = eig\left(\mathbf{X}\mathbf{X}^T\right) = eig\begin{bmatrix} fore & cross \\ cross & back \end{bmatrix} \tag{2}$$

with three image qualifiers defined as

$$\begin{aligned} fore &:= \mathbf{x}_f \cdot \mathbf{x}_f^T \\ cross &:= \mathbf{x}_f \cdot \mathbf{x}_b^T \\ back &:= \mathbf{x}_b \cdot \mathbf{x}_b^T. \end{aligned} \tag{3}$$

This amounts to

$$0 = \begin{vmatrix} fore - D^2 & cross \\ cross & back - D^2 \end{vmatrix}. \tag{4}$$

Mester [1] empirically showed that D^2 follows a χ^2 probability density function with $3(N-1)$ degrees of freedom and a proportionality factor σ_u^2. Based on the knowledge of this distribution the null hypothesis test can be reduced to a significance test, whereby D^2 is compared with a threshold t through $Prob[D^2 > t|H_0]{=}\alpha$ with the significance level α.

2.2 Bayesian estimation

A Bayesian analysis allows the above decision to be made on a principled basis. The result of this analysis will be a foreground or 'change mask' Q, found by maximizing its a-posteriori probability (MAP). The binary change mask consists of pixels with labels $q_i = u$ (unchanged, background) or $q_i = c$ (changed, foreground). Based on the distance measurement D^2, change labels are assigned following the decision rule:

$$\frac{p(Q_c|D^2)}{p(Q_u|D^2)} \underset{u}{\overset{c}{\gtrless}} t.$$

A 'changed' label is assigned to a pixel if the left term is greater than the threshold t, otherwise it gets 'unchanged' assigned to it. Using Bayes' theorem we get

$$\frac{p(D^2|Q_c)}{p(D^2|Q_u)} \underset{u}{\overset{c}{\gtrless}} t \cdot \frac{p(Q_u)}{p(Q_c)}.$$

In order to calculate the fraction on the left side of the above equation, both conditional probability density functions must be estimated. By modeling both pdf's as Gaussian distributions with variances σ_c and σ_u with $\sigma_c^2 \gg \sigma_u^2$, the decision rule can be simplified to (see [2])

$$D^2 \underset{u}{\overset{c}{\gtrless}} T_s + \underbrace{2 \cdot \ln\left(\frac{p(Q_u)}{p(Q_c)}\right)}_{T_{adapt}} \tag{5}$$

with a static threshold T_s and an adaptive threshold T_{adapt}.

2.3 Adaptive Threshold

Without any prior knowledge of the change mask, the adaptive threshold T_{adapt} in equation (5) is 0 because both probabilities $p(Q_u)$ and $p(Q_c)$ would then be equal. This often results in scattered foreground and background segments. To remedy this, one would like to bring spatio-temporal compactness considerations into play. Indeed, foreground objects tend to cover larger and more compact regions. If the object moves slowly compared to the framerate, this adds a temporal smoothness.

Such considerations are now added. A first element, is the spatial compactness. A pixel should have a higher chance of being considered foreground if several of its neighbours have this status.

This is a bit of a chicken-and-egg problem, however, as this assumes we already have a mechanism to decide on the neighbours first. In practice, this deadlock is solved by designing an iterative scheme. This will start with all pixels as background during the first iteration for the first frame. After that the results from the previous iteration for that frame are used, or that of the last iteration of the previous frame in case a next frame is started. Note that the latter choice pushes towards temporal smoothness.

Still following Mester [1] and in order to bring the spatial compactness idea to bear, the change mask is considered to be sampled from a two-dimensional Gibbs/Markov random field (MRF). Hereby the a priori probability is expressed by

$$p(Q) = \frac{1}{Z} \cdot e^{-E(Q)} \tag{6}$$

with a normalization constant Z and an energy-term $E(Q)$. The smoother the boundary of the change mask within the considered window W, the lower the energy-term $E(Q)$ is. Evaluating the smoothness and compactness can be simplified to account for changes between pixel pairs only. A pixel pair consists of two adjacent pixels in either horizontal, vertical or diagonal direction. Within any neighbourhood region, two kinds of pixel pairs can be distinguished: those who comprise the currently considered pixel, denoted as *local* pixel pairs, and all other pairs not comprising the current pixel, denoted as *global* pixel pairs. Hence, the energy-term can also be split into a local and a global term:

$$E(Q) = E_L(Q) + E_G(Q). \tag{7}$$

Based on a squared image grid, the 8 possible local pixel pairs within a 8-neighbourhood can be divided into two groups: 4 pairs of horizontal/vertical pairs (hv-dir.) and 4 diagonal pairs ($diag$-dir.). Designating the number of adjacent pixels in hv direction as ν_B and in $diag$ direction as ν_C wrt. the label q_i, equation (7) can be rewritten as

$$E(Q) = \nu_B(q_i) \cdot B + \nu_C(q_i) \cdot C + E_G(Q), \tag{8}$$

whereby B and C are constant multiplicative factors influencing the level of compactness.

Combining (8) with (6), inserting into (5) and accounting for both labels $q_i = c$ (changed) and $q_i = u$ (unchanged) yields to

$$\begin{aligned} T_{adapt} &= 2 \cdot [(\nu_B(q_i{=}c) - \nu_B(q_i{=}u)) \cdot B \\ &+ (\nu_C(q_i{=}c) - \nu_C(q_i{=}u)) \cdot C]. \end{aligned}$$

Since $\nu_B(q_i{=}c) + \nu_B(q_i{=}u){=}4$ and $\nu_C(q_i{=}c) + \nu_C(q_i{=}u){=}4$, and we choose $B{=}2C$, the equation above is simplified to

$$\begin{aligned} M &:= 2 \cdot \nu_B(q_i{=}c) + \nu_C(q_i{=}c) \\ T_{adapt} &= 12B - 2BM \\ D^2 &\overset{c}{\underset{u}{\gtrless}} T_s + T_{adapt} \end{aligned} \tag{9}$$

Indeed, the lower the amount of foreground pixels in the surrounding change mask is, the lower M and therefore the higher the adaptive threshold gets, increasing the barrier at which a considered pixel may be assigned to foreground. This intuitive behaviour results in smooth and compact regions, even in small neighbourhoods, i.e. 3×3 pixels.

2.4 Darkness Compensation

Though an intensity-invariant method as Mester's does provide robustness against shadows and lighting changes, such invariance also has drawbacks. Often, part of the foreground or background will be dark, i.e. close to black. As black can be seen as a low intensity version of any color, the current approach will never trigger segmentation in those areas.

Our solution consists of adding an additional component to the vectors $\mathbf{x}_b$ and $\mathbf{x}_f$ in eq. (1). This additional component has a fixed value of $\sqrt{O_{dc}}$. (The awkward use of the square-root has been opted for as this simplifies further notation; e.g. *fore, cross, and back* of eq. (3) are now increased by O_{dc}.) This additional component renders the color similarity measure more sensitive to differences, esp. when dark pixels are involved. Indeed, this additional component has the effect of lifting the $3N$-dimensional ground plane in the enlarged $3N + 1$-space, to height $\sqrt{O_{dc}}$, thereby distinguishing vectors that were colinear but had different norms.

When running these new $3N + 1$-dimensional vectors through the criterion, we come to the following observations:

- The resulting distances never decrease (proof is straightforward), thus regions that were previously segmented, remain segmented after the manipulation (for the same threshold T).
- The comparison of equal vectors remains unhampered (D^2 is 0 in both approaches).
- Moreover,when $\|\mathbf{x}_f\| = \|\mathbf{x}_b\|$ (more or less equal intensities), there is no impact whatsoever on the distance measure.

- Vectors that were previously colinear but of different sizes, will not remain colinear. The impact is dependent on the difference in intensity.
- As O_{dc} goes to infinity, the distance measure becomes $(fore + back - 2 \cdot cross)/2$. This equals $\|\mathbf{x}_f - \mathbf{x}_b\|/2$, which still yields a valid distance measure for background segmentation, but totally lacks the illumination invariance property useful to cope with shadows. The choice of O_{dc} determines how illumination sensitive the result is.

The above observations show that the provided manipulation now correctly segments dark coloured areas, compared with bright background. This was previously not the case, as these areas were seen as noise on a 0-vector. Furthermore, normal operation, where foreground looks like background or where foreground was already segmented from the background without the extra compensation, is not impaired.

2.5 Final Decision Rule

We can now convert the final decision rule into a form, which is suitable for computation. First, the determinant in (4) has to be computed involving a square root term, which is often a bottleneck in high-speed implementations on the CPU:

$$D^2 = \frac{fore{+}back{-}\sqrt{(fore{-}back)^2{+}4 \cdot cross^2}}{2}.$$

Fortunately, the square root must not explicitely be calculated, but D^2 directly compared to a threshold $T = T_s + T_{adapt}$ (see formula 9). After some algebraic manipulation we can deduce the following two inequalities:

$$\begin{aligned} (fore{+}back{-}2T)^2 &> (fore{-}back)^2{+}4cross^2 \\ fore{+}back{-}2T &> 0, \end{aligned}$$

which is equivalent to

$$\begin{aligned} (fore - T)(back - T) &> cross^2 \\ fore + back &> 2T. \end{aligned}$$

The first inequality shows that either $fore$ and $back$ are both $> T$, or both $< T$, while additionally considering the latter inequality forces $fore > T$ or $back > T$. As described in section 2.4, darkness compensation can be integrated by adding O_{dc} to each of the three qualifiers $fore$, $back$ and $cross$. Now the final decision rule set can be formulated:

$$\begin{aligned} M &= 2 \cdot \nu_B(q_i{=}c) + \nu_C(q_i{=}c) \\ T_t &= T_s + 12B - 2BM - O_{dc} \\ fore &\overset{c}{>} T_t \\ (fore{-}T_t)(back{-}T_t) &\overset{c}{>} (cross{+}O_{dc})^2 \end{aligned} \tag{10}$$

Influenced by three user-defined parameters, a static threshold T_s, a darkness offset O_{dc} and a compactness value B, a label 'changed' is assigned to a pixel if both inequalities are fulfilled. Otherwise the pixel's label is set to 'unchanged'. Notice that the $back$ qualifier only depends on the background image, which practically is the average over several background images and has to be computed only once, while $fore$ and $cross$ need to be updated every frame.

2.6 Iterative, randomized MRF computation

Up to now we assumed that a change mask is already known and the decision rule set updates this mask allowing for smooth and compact segments. Starting with a pixel in an input image requires the knowledge of all 8 adjacent pixel labels, which themselves are mutually dependent on the current pixel label. Additional information is available from the change mask of the previous frame. Though sliding a 3×3 window through the image plane once, starting from top left, whereby 4 labels are already set and the other 4 are taken from the prior frame (see [2, 3, 1]), does deliver good segmentations, we designed a more general algorithm, applicable to parallel computation hardware.

Under consideration of a 3×3 neighbourhood, we can observe that pixels with a chessboard distance ≥ 2 do not directly affect each other and can therefore be handled in parallel. This processing step, denoted as substep, operates on a subset of the input data, whereby pixels within this subset are mutually independent wrt. the MRF computation. To cover all the input data, several substeps have to be executed in sequential manner. The smallest number of substeps is given by the local partition shown in the above

m	n	m	n	m	n	
k	l	k	l	k	l	
m	n	m	n	m	n	
k	l	k	l	k	l	

image. Four substeps k, l, m and n are sequentially executed, whereby the execution order is chosen randomly in order to guarantee uniform distribution over time. The whole process is repeated several times until convergence in the segmentation wrt. compactness is reached. From the implementation point of view at each iteration j we select the execution sequence by randomly picking one of the 24 possible permutations of (k, l, m, n).

$$S_j = rand\left(perm(k, l, m, n)\right). \qquad (11)$$

It is important to mention that the result of each substep is written back into the change mask, which serves as input for the next substep. A priori knowledge from the previous frame's change mask is integrated by initializing the change mask with the prior change mask and then start the MRF-iterations. If no prior frame is given, we set the prior change mask to full background, i.e. black color.

The program flow, as depicted in figure 1, starts with the computation of $fore$, $back$ and $cross$ based on the average background image BG and the current foreground image FG. This is followed by one iteration comprising four substeps on the previous change mask as input and a parameter set 1 (static threshold T_s, darkness offset O_{dc} and compactness value B_1). Finally we run the iteration j times with a parameter set 2, where the static threshold and the darkness offset both remain the same while the compactness value differs between between both parameter sets.

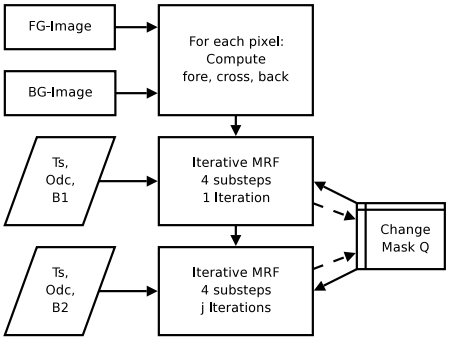


Figure 1: Program flow for the iterative MRF computation.

3 GPU-based Implementation

Modern graphics hardwares provide programmable vertex- and pixel-shaders, interfacable by shading languages like Cg (C for graphics), developed by NVIDIA Corporation and Microsoft Corporation. The graphics hardware is optimized for 32bit-RGBA buffers, storing 8bits of data in each of the four color channels red, green, blue and alpha. Vector operations are implemented directly in hardware, which increases performance over standard CPU's. Moreover the internal pipeline structure includes several parallel processing units, i.e. 16 shader units with 2 processing modules each on NVIDIA's GeForce 6 series.

Figure 2 gives an overview of the program flow of the GPU implementation, whereby a fixed neighbourhood of 3×3 is used. The round-shaped boxes in the left column symbolize the different Shader-programs, which are described in the following sections. Each Shader gathers information from one or more inputs and writes into an offscreen PBuffer object. A copy-command afterwards transfers the currently written data into the target texture, as depicted in the right column. Thereby not the full texture object has to be written but just the affected memory areas (grey-shaded in the right column of the figure).

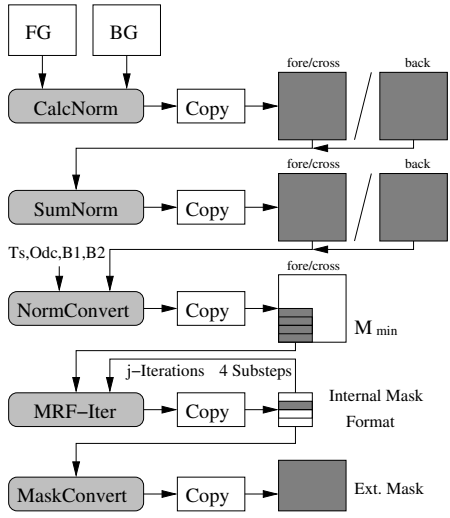


Figure 2: Program flow for GPU implementation.

3.1 The *CalcNorm*-Shader

This first Shader steps through the input image, calculating for each pixel $\mathbf{y}_i=[r_i\ g_i\ b_i]$ at location i the three qualifiers f_i, b_i and c_i based on an input image FG and the background average image BG by computing the following three dot products:

$$f_i=\mathbf{y}_{i,f}\cdot\mathbf{y}_{i,f}^T, b_i=\mathbf{y}_{i,b}\cdot\mathbf{y}_{i,b}^T, c_i=\mathbf{y}_{i,f}\cdot\mathbf{y}_{i,b}^T$$

The indices f and b denote foreground and background pixels. The dot product computation on the GPU is rather trivial, it is executed within one machine operation and thus much faster than a CPU variant. It turns out that decision rule (10) is only applicable when all three qualifiers have at least 16bit resolution. This could be reached by using the floating-point extensions on modern GPU's, which -still- results in slow texture lookup times. Instead, we decided for packing each 16bit qualifier into two bytes of a standard 8bit-buffer.

Only when a background image has changed, the b-buffer has to be updated, while remaining constant during normal operation. Hence, we store the values f_i and c_i in one RGBA texture and b_i in a separate RGBA texture.

3.2 The *SumNorm*-Shader

After the dot products are computed per pixel, the three qualifiers $fore$, $back$, and $cross$ can now be derived by following eq. (3). Accounting for f_i, b_i and c_i, the equation is simplified to summing up all dot products in the neighbourhood W_i around the pixel location i:

$$fore=\sum_{j\epsilon W_i} f_j, back=\sum_{j\epsilon W_i} b_j, cross=\sum_{j\epsilon W_i} c_j$$

Again $fore$ and $cross$ are packed and stored in an RGBA texture while $back$ remains in a separate RGBA buffer.

3.3 The *NormConvert*-Shader

Now that all parameters for testing the decision rule (10) are known, the iterative MRF computation begins. However, for each substep in an iteration the required inequalities have to be recalculated. As this would slow down the overall process the ruleset is reformulated to a simpler test. The only variable during a MRF iteration is M and therefore we can rewrite the decision rule based on M

$$S := \frac{T_s+12B-O_{dc}}{2B}$$

$$D := \frac{\sqrt{(fore-back)^2+4(cross+O_{dc})^2}}{4B}$$

$$M \overset{c}{>} S-\frac{fore}{2B}$$

$$M \overset{c}{>} S-\frac{fore+back}{4B}+D$$

The conjunction of both inequalities yields

$$M_{min}=S-\frac{fore}{2B}+max\left(0, D+\frac{fore-back}{4B}\right) \quad (12)$$

and turns the decision rule into

$$M \overset{c}{\underset{u}{\gtrless}} M_{min} \quad (13)$$

Due to the fact that M can only vary between 0 and 12 and therefore consumes only 4 bits, both $M_{min,1}$ for the first parameter set and $M_{min,2}$ for the second parameter set are packed into one 8bit color value by $((M_{min,1} \ll 4) \mid M_{min,2})$.

In order to gain maximum performance of the GPU, all 4 color channels of the RGBA PBuffer should be used at once, leading to a special internal data format, which is shown in figure 3.

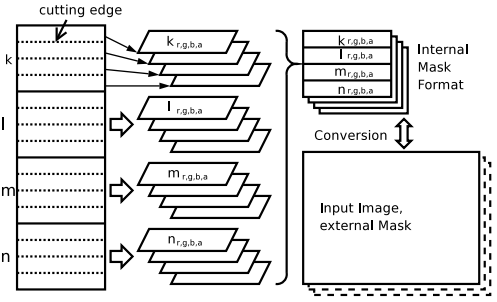


Figure 3: Data formats used on the GPU

As described in section 2.6, each substep within one iteration operates on a subset (k, l, m, n) of the input data with size half of the image height and half of the image width. Those subsets (left column in figure 3) contain change/unchange-labels for each considered pixel. We slice subsets into 4 equally sized pieces by cutting along a horizontal *cutting edge* and folding them to the 4 color channels R, G, B, and A. Clearly, the topmost quarter of subset k is now stored in the red channel of the internal structure, that is k_r. The bottommost quarter of the same subset k is stored in the alpha channel (k_a), and so forth. This brings us to the internal data format, where 4 pixels are processed in parallel. It is

the task of the *NormConvert*-Shader to perform this conversion while maintaining the $M_{min,1+2}$ values for both parameter sets (T_s, O_{dc}, B_1, B_2).

3.4 The *MRF-Iter*-Shader

As already mentioned in the section above, the iterative MRF-computation consists of 4 substeps, each of which solves the MRF for compactness and smoothness. This is done by calculating M per pixel and comparing it with the stored value M_{min}. The different subsets are selected based on eq. (11). As calculation of M_{min} for a pixel within a subset depends on its neighbours, special care has been taken on the cutting edges in figure 3. For example, M_{min} has to be computed for a pixel in the border row of the subset k_g (green channel of k), which, amongst others, requires access to pixels one line above. Indeed, these pixels do not reside in the same subset but in k_r (red channel of k).

Moreover, optimization is done wrt. the amount of necessary texture lookups by utilizing the bilinear interpolation functionality of the graphics hardware. For example, computing the sum of 4 values in a 2x2 grid would require 4 texture lookups and 3 summations. By placing the texture coordinates in the middle of the 4 pixels and activating bilinear texture filtering, only one texture lookup delivers the same result without loss of time.

3.5 The *MaskConvert*-Shader

After the iterative MRF-computation, the internal change mask has to be backtransformed into an external format, suitable for further processing. Hence, this last shader converts the internal change mask into an RGB buffer, having the same dimensions as the input image, with white foreground and black background regions.

4 Results

The impact of the number of iterations during the MRF computation to the segmentation result is shown in figure 4, whereby the prior frame is completely black. Without iterating at all, some outliers are detected, depending on the threshold level. With increasing number of iterations the segmented regions become more smooth and compact and isolated small foreground pixels are turned into background. The parameters used are $T_s = 310$, $O_{dc} = 5800$, $B_1 = 2$, and $B_2 = 200$. We observed that robustness of segmentation wrt. the static threshold is increased with the MRF iterations.

Figure 4: Segmentation result with different number of iterations j=0,2,4,6 (from left to right) during the MRF-computation.

As the algorithm is mainly designed for indoor applications, we recorded some image sequences in our 3D-scanning setup (see [10]). Accounting for motion blur elimination and thus forcing low exposure times, the grabbed images suffer from very low contrast and high image noise. Therefore darkness compensation is essential for acceptable segmentations even the foreground object and the background are nearly similar in luminance. Figure 4 demonstrates the importance of darkness compensation. All three segmentations use the same parameter set. The fourth image is the result of an optimized CPU-version of the segmentation without MRF-iterations, that is darkness compensation added to the algorithm of Mester. Additionally we tested the algorithm in outdoor environments (see figures on the separate colorplate 5) and still get good results.

To our knowledge, our GPU-based implementation outperforms any other existing image segmentation wrt. runtime. The following table summarizes the measured timings dependent on the number of MRF-iterations, whereby we used images of size 640×480 pixels on an NVIDIA GeForce 6800GT graphics card with AGP 8x. The time needed for uploading the input images to the GPU and downloading the final segmentation is not included here.

iter.	time [ms]
0	2.51
2	2.97
4	3.44

iter.	time [ms]
6	3.89
8	4.38
10	4.84

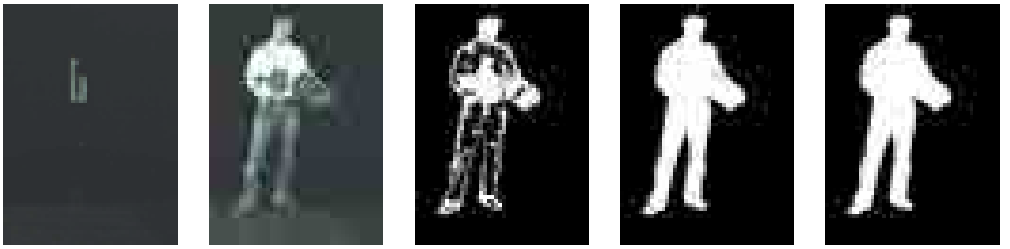


Figure 5: From left to right: BG-image, FG-image, segmentation without darkness compensation (d.c.), CPU-based segmentation with MRF and d.c., GPU-based iterative segmentation with MRF, d.c. and 6 iterations.

5 Conclusion and Future Work

We have provided a high-speed foreground-background segmentation algorithm based on an extended colinearity criterion. Darkness compensation helps the algorithm to correctly detect foreground regions even if the object's color is nearly black. The colinearity criterion is integrated in a MRF framework, which is solved in an iterative manner. A priori knowledge is integrated through the change mask of the prior frame as well as the compact property of connected foreground regions. The resulting segmentations are smoothly shaped and the number of outliers caused by varying illumination or inappropriate thresholds is reduced. Our GPU-based implementation runs in less than 4ms and thus offers performance improvement in applications, which operate on silhouette images, i.e. visual hull reconstructions.

In the near future we plan to integrate our segmentation algorithm into a 3D-scanning setup, which operates on a GPU-based plane-sweep method (see [11]). We are currently developing a hybrid 3D-scanner consisting of several cameras and projectors around the scanned object, i.e. human body, combining multi-view stereo, structured light and shape from silhouette.

References

[1] R. Mester, T. Aach, L. Dümbgen, "Illumination-Invariant Change Detection Using a Statistical Colinearity Criterion", *Proc. 23rd DAGM Symp.*, 2001.

[2] T. Aach, A. Kaup, "Bayesian algorithms for adaptive change detection in image sequences using Markov random fields', *Signal Processing: Image Communication 7(2)*, 1995.

[3] T. Aach, A. Kaup, R. Mester, "Change detection in image sequences using Gibbs random fields", *IEEE Int. Workshop Intell. Signal Processing Com. Sys.*, 1993.

[4] C. Stauffer, W.E.L. Grimson, "Adaptive Background Mixture Models for Real-Time Tracking", *Proc. CVPR*, 1999.

[5] N. Friedman, S. Russell, "Image Segmentation in Video Sequences: a Probabilistic Approach", *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, 1997.

[6] R. J. Radke *et al.*, "Image Change Detection Algorithms: A Systematic Survey", *Image Processing 14 (3)*, 2005.

[7] R. Kehl, L. Van Gool, "Real-time Pointing Gesture Recognition for an Immersive Environment", *Proc. 6th IEEE Intl. Conf. on Aut. Face and Gesture Recog.*, 2004.

[8] W. Matusik, C. Bueler, L. McMillan, "Polyhedral visual hulls for real-time rendering", *Proc. EGRW*, 2001.

[9] M. Li, M. Magnor, H.-P. Seidel, "Hardware-Accelerated Visual Hull Reconstruction and Rendering", *Graphics Interface*, 2003.

[10] A. Griesser, T.P. Koninckx, L. Van Gool, "Adaptive Real-Time 3D Acquisition and Contour Tracking within a Multiple Structured Light System", *Proc. 12th Pacific Graphics*, 2004.

[11] N. Cornelis, L. Van Gool, "Real-Time Connectivity Constrained Depth Map Computation Using Programmable Graphics Hardware", *Proc. CVPR*, 2005.

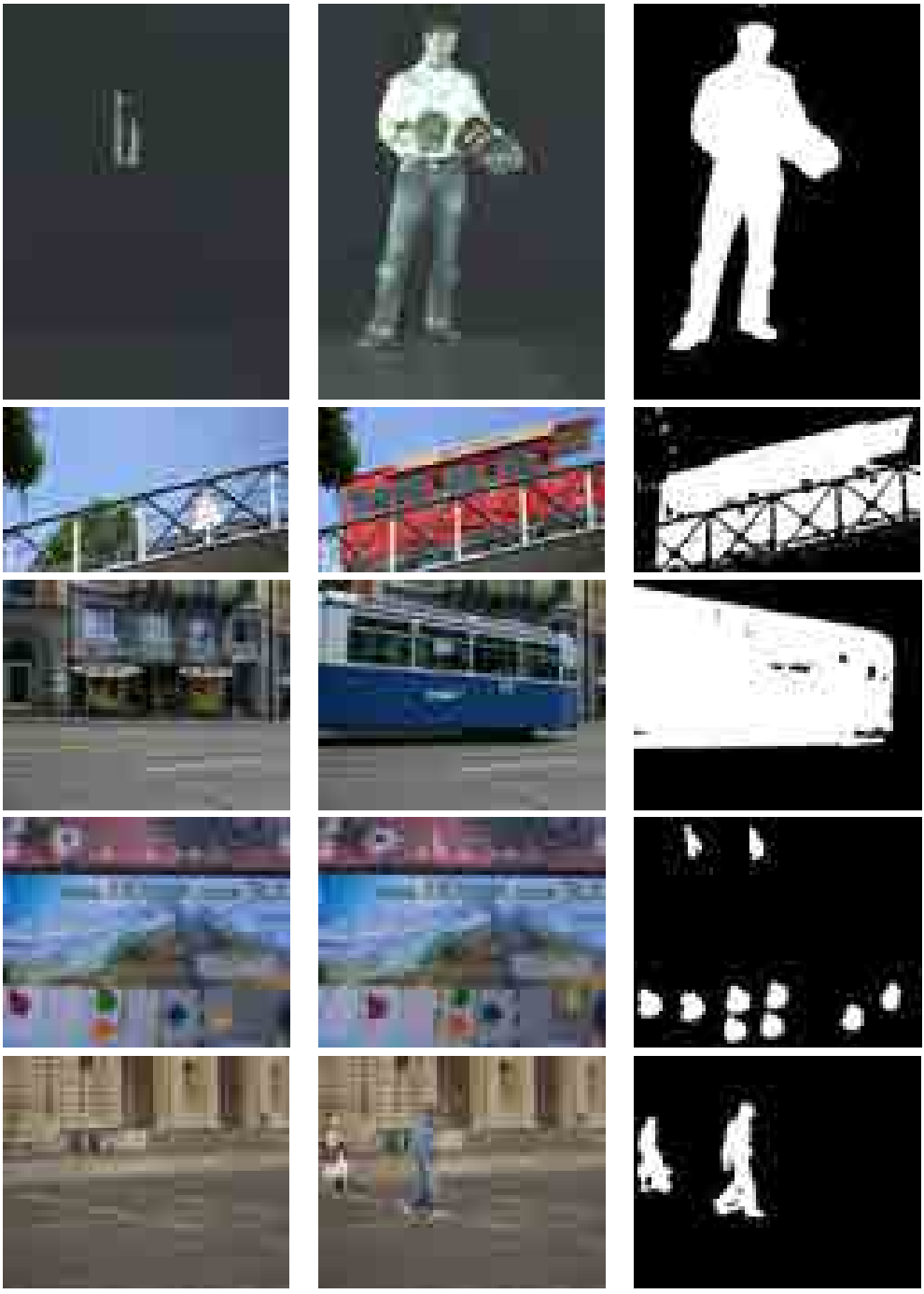


Figure 6: Segmentation results in indoor and outdoor environments. From left to right: BG-image, FG-image, Segmentation usind darkness compensation and 6 MRF iterations on the GPU. Processing time for each image is less than 4ms