Implementation of the "Local Rank Differences" Image Feature Using SIMD Instructions of CPU

Adam Herout, Pavel Zemcik, Roman Juranek, Michal Hradis
Graph@FIT, Brno University of Technology
{ herout, zemcik, ijuranek, ihradis } @fit.vutbr.cz

Abstract

Usage of statistical classifiers, namely AdaBoost and its modifications, in object detection and pattern recognition is a contemporary and popular trend. The computatiponal performance of these classifiers largely depends on low level image features they are using: both from the point of view of the amount of information the feature provides and the executional time of its evaluation. Local Rank Difference is an image feature that is alternative to commonly used Haar features. It is suitable for implementation in programmable (FPGA) or specialized (ASIC) hardware as well as graphics hardware (GPU). Additionally, as shown in this paper, it performs very well on common CPU's. The paper discusses the LRD features and their properties, describes an experimental implementation of LRD using the multimedia instruction set of current general-purpose processors, presents its empirical performance measures compared to alternative approaches, and suggests several notes on practical usage of LRD and proposes directions for future work.

1. Introduction

Statistical classifiers can very well be used for object detection or pattern recognition in raster images. Current algorithms even exhibit real-time performance in detecting complex patterns, such as human faces [11], while achieving precision of detection which is sufficient for practical applications. Recent work of Šochman and Matas [10] suggests that any existing detector can be efficiently emulated by a sequential classifier which is optimal in terms of computational complexity for desired detection precision. In their approach, human effort is invested into designing a set of suitable features which are then automatically combined by the WaldBoost [9] algorithm into an ensemble. This approach may significantly reduce the development time of detectors and it may even lead to more computationally efficient detectors – Šochman and Matas report successfully emulating the Kadir-Brady saliency detector [2], while achieving 70× faster detection times over the original implementation.

In practical applications, the speed of the object detector or other image classifier is crucial. Real-time performance is required in many applications, such as surveillance, where several input image streams need to be precessed. The usage of specialized hardware in image processing and computer vision is nothing new (e.g. [8], [4]). Recent advances in development of graphics processing units (GPUs) attract many researchers and engineers to the idea of using GPU's not for their primary purpose – rendering 3D graphics scenes – but rather for other computationally intensive tasks. Different approaches to so-called GPGPU (General-Purpose computation on GPUs) [1] exist and also the field of image processing and computer vision has seen several successful uses of these techniques (e.g. [8], [4]). These execution environments have been tested by the authors recently with good results and a SIMD CPU implementation has been used as an alternative for comparison. It turned out to be a vital option and the experiments showed that (slightly in contrary to expectations) the LRD outperforms the Haar features also in the general purpose processors – the possibilities to speed up the Haar features by SIMD instructions of current processors are not high. This paper presents an implementation of the LRD using the SIMD instruction set.

Statistical classifiers are built by using low level *weak classifiers* or *image features* and the properties of the classifier largely depend on the quality and performance of the low level features. In face detectors and similar classifiers, Haar-like features [3], [9], [10], [11] are frequently used since they provide good

978-0-7695-3476-3/08 $25.00 © 2008 IEEE
DOI 10.1109/ICVGIP.2008.27

amount of discriminative information and they provide excellent speed performance. Other features, such as the Local Binary Patterns [5], are used in different contexts. Recently, designed especially for being implemented directly in programmable or hard-wired hardware, Local Rank Differences [12] have been presented. These features are described in more detail in section 2 of this paper. The main strengths of this image feature are inherent gray-scale transformation invariance, the ability to capture local patterns and the ability to reflect quantitative changes in lightness of image areas.

The following Section 2 of this paper briefly presents the Local Rank Differences (see [12] for more detail) image feature. In Section 3, the notes on implementation of the LRD using the MMX instruction set of the Intel and compatible CPU's are given. Section 4 presents the experimental results of the implementation carried out and its comparison to other approaches. Conclusions and suggestions for future research in the area are given in Section 5.

2. Local Rank Differences

Let us consider a scalar image $I(x,y) \rightarrow \mathbf{R}$. On such image, a *sampling function* can be defined

$$S_{xy}^{mn}(u,v) = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} I\big(x+m(u-1)+i,\, y+n(v-1)+j\big) \quad (1)$$

This sampling function is parameterized by the sampling block dimensions *m, n*, and by the origin of the sampling *(x,y)*, which is a pixel in the image. Note that this function "subsamples" the image by a multiple of pixels in each direction. Note please also that this function can be defined in other manners, namely not by *summing* rectangular blocks of the image but by convolving them with a suitable feature, etc.

Based on this sampling function a rectangular *mask* can be defined:

$$M_{xy}^{mnwh} = \begin{bmatrix} S_{xy}^{mn}(1,1) & S_{xy}^{mn}(2,1) & \cdots & S_{xy}^{mn}(w,1) \\ S_{xy}^{mn}(1,2) & S_{xy}^{mn}(2,2) & \cdots & S_{xy}^{mn}(w,2) \\ \vdots & \vdots & \ddots & \vdots \\ S_{xy}^{mn}(1,h) & S_{xy}^{mn}(2,h) & \cdots & S_{xy}^{mn}(w,h) \end{bmatrix} \quad (2)$$

The mask is parameterized by sampling block dimensions *m, n* and sampling origin *(x,y)*, just as the used sampling function *S*. Along with these parameters, the mask has its dimensions *w, h* as well. Experiments (see [12]) show that in the context of AdaBoost and WaldBoost object detection, the masks of dimensions 3×3 (*w=3, h=3)* are sufficient. For different classifiers and applications, different sampling block sizes are necessary: for face detectors operating on image windows with resolution of 24×24 pixels, sampling sizes of 1×1 (*m=1, n=1* etc.), 2×2, 2×4, 4×2 are sufficient.

For each position in the mask, its *rank* can be defined:

$$R_{xy}^{mnwh}(u,v) = \sum_{i=1}^{w}\sum_{j=1}^{h} \begin{cases} 1, if\ S_{xy}^{mn}(i,j) < S_{xy}^{mn}(u,v) \\ 0, otherwise \end{cases}, \quad (3)$$

id est, the rank is the order of the given member of the mask in the sorted progression of all the mask members. Note that this value is independent on the local energy in the image, which is an important property useful for the behavior of the Local Rank Differences image feature, which is defined as:

$$LRD_{xy}^{mnwh}(u,v,k,l) = R_{xy}^{mnwh}(u,v) - R_{xy}^{mnwh}(k,l), \quad (4)$$

The notation can be slightly facilitated by vectorizing the matrix M by stacking its rows (it is just a convention that row rather than column stacking is used):

$$V_{xy}^{mnwh} = \left[S_{xy}^{mn}(1,1) \quad S_{xy}^{mn}(2,1) \quad \cdots \quad S_{xy}^{mn}(w,h)\right]. \quad (5)$$

The rank of a member of the vector then is (note that for clarity $V_{xy}^{mnwh}(i)$ denotes the ith member of the vector):

$$R_{xy}^{mnwh}(a) = \sum_{i=1}^{w\times h} \begin{cases} 1, if\ V_{xy}^{mnwh}(i) < V_{xy}^{mnwh}(a) \\ 0, otherwise \end{cases}. \quad (6)$$

The Local Rank Difference of two positions *a, b* within the vector obviously is:

$$LRD_{xy}^{mnwh}(a,b) = R_{xy}^{mnwh}(a) - R_{xy}^{mnwh}(b). \quad (7)$$

Empirical experiments carried out so far show that one w×h dimension used in a classifier is sufficient (currently we are using 3×3 mask dimension only). LRD features available to the statistical classifier therefore offer varying position *x, y* within the window of interest and varying size *m, n* of the sampling block used.

2.1 Input Image Pre-Processing

For increasing the performance of the LRD evaluation, the *function* S_{xy}^{mn} defined on the input image can be pre-calculated. As stated above, low number of combinations of $m \times n$ is sufficient for learning an object classifier – experiments show that 1×1, 2×2, 1×2 and 2×1 combinations are enough (see Figure 2). The input image I can be convolved with

$$h_{2\times2} = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix},$$
$$h_{4\times2} = \begin{bmatrix} 1/8 & 1/8 & 1/8 & 1/8 \\ 1/8 & 1/8 & 1/8 & 1/8 \end{bmatrix}, \quad (8)$$
$$h_{w\times h} = \begin{bmatrix} 1/wh & \cdots & 1/wh \\ \vdots & \ddots & \vdots \\ 1/wh & \cdots & 1/wh \end{bmatrix}$$

and the resulting images at given location (x,y) can contain the values of the sampling function. Such pre-processing of the input images can be done efficiently and the LRD evaluation then only consists of 9 lookups (for the case of 3×3 LRD mask) into appropriate pre-processed image and then evaluation of ranks for two members of the mask. The evaluation then can be done in parallel on platforms supporting vector operations; each of MMX, GPU and FPGA platforms are strong in such kind of parallelism.

2.2. The Role of LRD in an Object-Detecting Classifier

Figure 1 shows the simplified flow for evaluating a single LRD classifier. It begins with the detection window (e.g. 31×31 pixels) being classified where rectangular mask M_{xy}^{mn33} is positioned (considering e.g. 3×3 masks). Each field of the mask spans across several pixels which need to be convolved (see the equation 1 above).

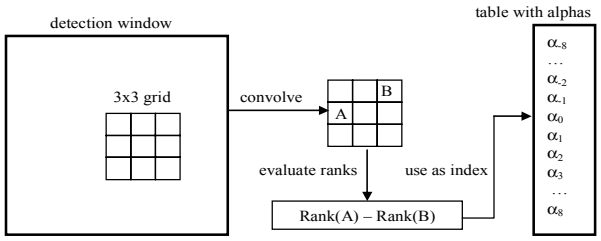

Figure 1. Usage of Local Rank Differences in a classifier

Next, the ranks are evaluated, and finally the rank difference is used as index into the alpha table, selecting the feature's result.

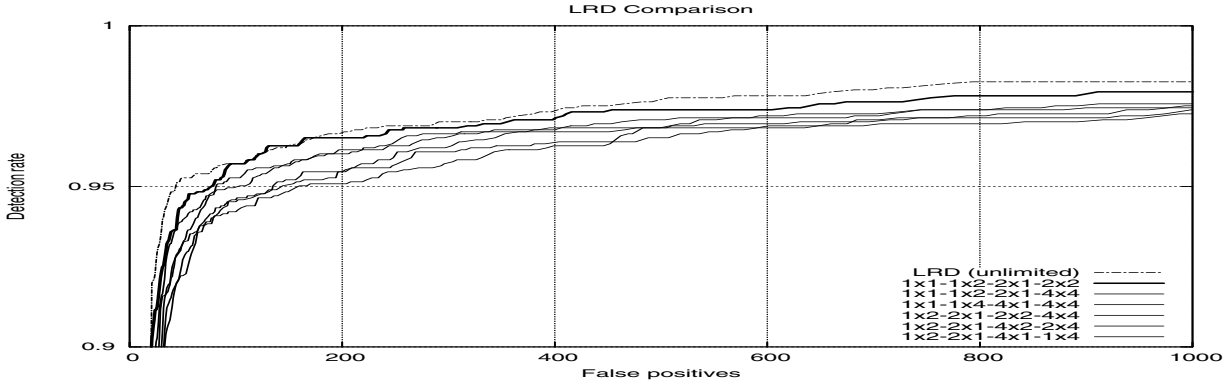

Figure 2. Performance of LRD with different sizes of the sampling function

2.3 Local Rank Differences Compared to Haar Features

Comparing LRD with Haar features is only natural as both of these types of features were first intended to be used in detection classifiers. There are two fundamental aspects in respect to the detection classifier which must be addressed. The first aspect is the computational complexity of evaluating the features and the second aspect is the amount of discriminative information the features provide.

Haar features can be computed very rapidly on general purpose CPUs by using the integral image representation [11] which can be created in a single pass through the original image. The simple Haar features of any size can be computed using only six accesses into the integral image, six additions/subtractions, and two bit-shifts. When

scanning the image in multiple scales, this approach gives the possibility to scale the classifier instead of down-sampling the image. The Haar features are usually normalized by the size of the feature and the standard deviation of pixel values in the classified sub-window. Computation of the standard deviation requires additional integral image of squared pixel values and uses square root operation.

While the Haar features can be computed relatively efficiently on general purpose CPUs, it may not be the same on other platforms. On FPGAs, the six random accesses into memory would significantly limit performance (in most cases only single feature evaluated per every six clock cycles due to the random access in the memory) and the high bit-precision needed for representing the integral images would make the design highly demanding. On the other hand, the nine values needed to compute LRD with grid size 3×3 can be obtained on FPGAs with only single memory access as the memory access pattern is fixed [12], in the case of SSE using two memory accesses (see section 3 for details).

Some detection classifiers evaluate in average very low number of features (even less than 2). In such cases, computing the normalizing standard deviation poses significant computational overhead. Furthermore, the square root, which is needed, cannot be easily computed on FPGAs. The LRD inherently provide normalized results, which are, in fact, very similar to local histogram equalization.

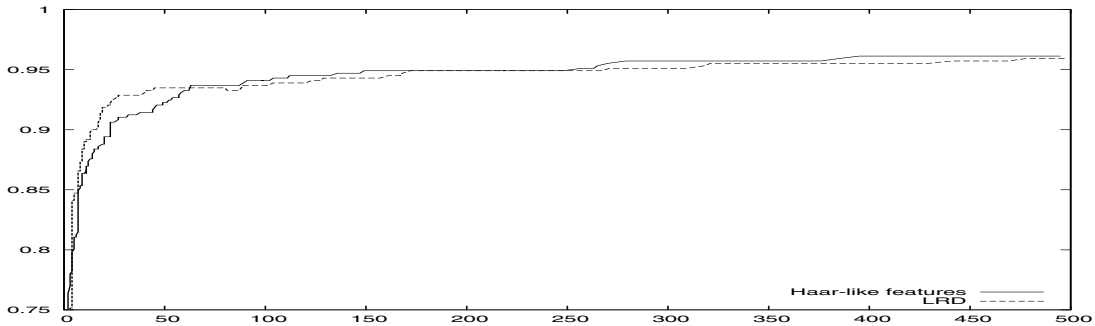

Figure 3. Receiver operating characteristic of two WaldBoost classifiers on the MIT+CMU face database. The classifiers differ only in type of features witch they use (Haar- features, LRD).

The detection performance of classifiers with the LRD has been evaluated on the frontal face detection task and it has been compared to the performance of classifiers with standard Haar features [11]. The results suggest that the two types of features provide similar classification precision and similar average classification speed. This fact can be clearly seen in Figure 3 (which presents receiver operating characteristic (ROC) of two classifiers which have similar classification speed – 5 weak classifiers per scanning position). Both are WaldBoost [9] classifiers, but one uses the classic Haar features and the other uses the LRD. These results were measured on the MIT+CMU frontal face database which is commonly used to evaluate performance of the face detectors [9], [10], [12]. Although only limited set of the sampling block sizes is used for the LRD (see section 2.1), the information the LRD provide is similar to the Haar features. This is probably due to the localized normalization of the results of the LRD which provides information about local image patterns that goes beyond simple difference of intensity of image patches.

3. Implementation of LRD on Intel CPU

The goal of the described work is an efficient implementation of the Local Rank Difference image features on the Intel CPU (compatible) using the SIMD instruction set. Different generations of the SIMD instructions were considered, the SSE 4.1 was selected, because it is covered in all current processors and supports operations with multiple integers.

The desired implementation must address two crucial issues: memory accesses performed by the algorithm (minimizing the number of memory accesses and ensuring their speed by aligning the operands) and the algorithmic evaluation of the local ranks and their differences.

Intel CPUs offer some features that are interesting for efficient LRD evaluation. The SSE instruction set has extensive support of instructions working with eight 8bit values in a single 128bit register. It is also possible to perform arithmetic and logic operations with sixteen of these values at once instead of using a loop.

3.1 Storage of the Image Convolutions

To simplify feature evaluation as much as possible, the convolutions of image are pre-computed and stored in the memory in such manner that all the results of the LRD grid can be fetched into the CPU registers through two 64-bit loads.

The convolved (pre-calculated) image is divided into separate blocks which correspond to different modulo shifts of the convolution kernel – e.g. for 4×2 convolution eight possible shifts exist: 0,0; 0,1; … 0,3; 1,0; … 1,3. Figure 4 shows the situation where the 2×2 kernel is used and four blocks are formed in the convolved image.

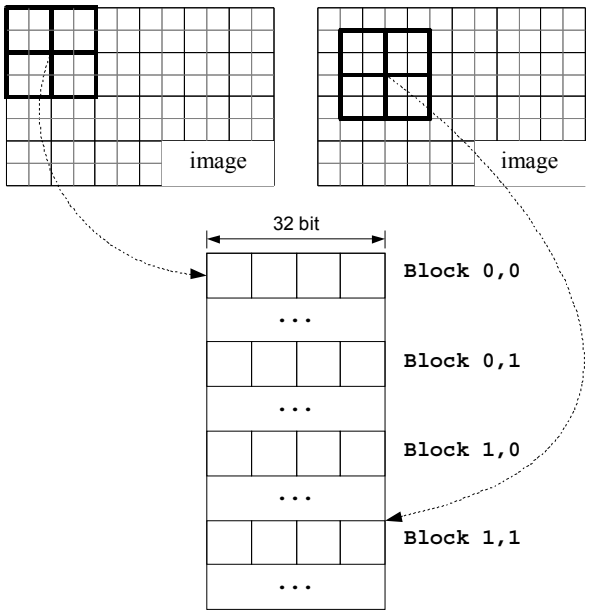

Figure 4. Memory storage of 2×2 convolved image.

Each byte in the convolved image corresponds to a pixel in the convolution of original image with a convolution kernel (see Section 2.1) and each 32-bit aligned 4-byte block corresponds to four adjacent convolutions.

$$n = \frac{2W}{M} \quad k = \frac{Hn}{2N} \qquad (9)$$

For the LRD evaluation, important parameter of the convolution image is spacing of adjacent convolution rows n and blocks k - Eq.X, where W, H is width and height of convolved image and M, N width and height of convolution kernel.

In above example (Fig 3) when image size is e.g. 320x240px, size of one convolution row would be 320B and size of one convolution block 19200B and size of the convolved image 76800B, which is same size as the source image.

3.2 LRD evaluation

The code of the evaluation using SIMD instructions (by using Intel's intrinsic functions in C) is shown in Figure 5 and the block diagram of the evaluation is in Figure 6. The LRD are parameterized by the feature's position within the classified sample (x, y), the block size (w, h) which determines the convolution image to use and indices of rank pixels (A, B).

```
const union {
  int i[4];
  m128i q;
} masks[4] = { ... }
  ...
data0 = ...
data1 = ...
maskId = ...
valA = (idxA < 8) ?
           data0[idxA] : data1[idxA-8];
valB = (idxB < 8) ?
           data0[idxB] : data1[idxB-8];
  ...
__m128i data = _mm_set_epi64(
                     *(__m64*)(data0),
                     *(__m64*)(data1));
__m128i zero = _mm_setzero_si128();
__m128i A = _mm_set1_epi8(valA);
__m128i B = _mm_set1_epi8(valB);
union {
  __m128i q;
  signed short ss[8];
} diff = { _mm_sub_epi16(
             _mm_sad_epu8( // countA
               _mm_and_si128(
                 _mm_cmpgt_epi8(A, data),
                 masks[maskId].q),
               zero),
             _mm_sad_epu8( // countB
               _mm_and_si128(
                 _mm_cmpgt_epi8(B, data),
               masks[maskId].q),
               zero) )
}.
```

Figure 5. C code of the SIMD implementation of the LRD

The convolution image representation ensures that the data for one feature can be obtained by two 64-bit load operations. But only 4×4 grid from even positions can be obtained in this way. The data, therefore, must be masked to exclude pixels (of the convolved image) that do not belong to the feature grid.

The code above assumes that *data0* and *data1* point to the top and bottom half of the 4×4 grid in the

convolution image corresponding to the evaluated feature, *masks* is a pre-filled array of four masks for different positions of the feature within the 4×4 grid. The *valA* and *valB* are values of the rank pixels. a

```
shift = (fx mod M, fy mod N)
pos = ((fx div M) >> 1, (fy div N) >> 1)   (10)
mask = ((fx div M) & 1, (fy div N) & 1)
```

The data pointers are calculated from the absolute feature position in image *(fx, fy)* and the feature block size *(M, N)*. Eq. 9 shows calculation of modulo shift of the convolution kernel i.e. the block of convolution image to use *(shift)*, feature position in the convolution image *(pos)* and mask type *(mask)*. Note that the calculations can be reduced to logical operations and bit shifts for features with block size of power of 2.

The evaluation of LRD itself is very simple and could be expressed by Equation 10, where all variables are vectors and each operation processes corresponding components in given operands separately. This is the property of the SSE instruction set – all instructions work with 128-bit registers, where the register can hold e.g. 16 signed char values. The sum function returns sum of all items in the given register.

```
diff = sum((A > data) & mask) -
       sum((B > data) & mask)          (11)
```

The basic step of the evaluation is comparison of all data pixels to the values in the rank positions. The comparison of two registers results in -1 (0xFF) when the condition is true in SSE. The masking zeros values that are not in the feature grid and also converts 0xFF values to 0x01 so the sum of all items of the register corresponds to the number of positive comparisons. The SSE instruction set has no instruction for sum of differences of two vectors which is needed for LRD evaluation. However, there is an instruction calculating the *sum of absolute differences* (SAD) of two registers, which can be used as SAD to zero vector - sum of all the items. The instruction operates over high and low 64 bits separately, whose results need to be summed together to obtain the actual sum.

The LRD evaluation has therefore these steps (Figure 6). First, the data are compared to *A* and *B* vectors and masked (temporary results *cmpA*, *cmpB*). Sums of absolute differences of *cmpA* and *cmpB* are subtracted and the results for high and low parts are summed together producing the LRD value.

The evaluation is much more efficient compared to CPU code without SSE since all the values are processed in parallel. The slowest step of the evaluation is the expansion of the 8-bit value to full 128-bit SSE register. Since the instruction set lacks a single instruction to do this, the expansion must be done by a sequence of *shift-left* and *or* instructions.

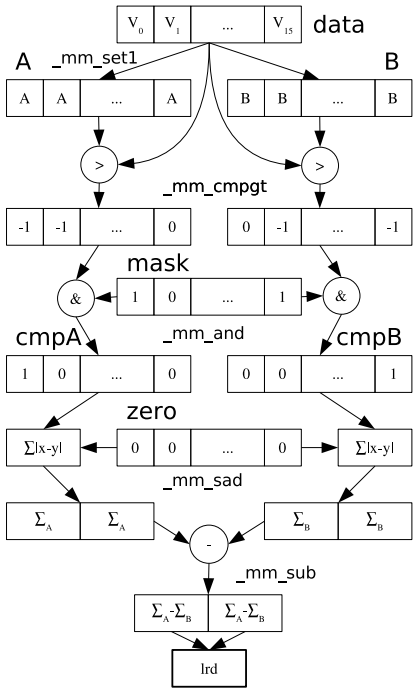

Figure 6. Schematic of LRD evaluation by SSE

4. Performance Evaluation and Analysis

To evaluate the efficiency of the presented SSE implementation of the LRD, these implementations were compared:

- LRD on CPU/SSE (section 3 above)
- LRD on CPU – integral image
- LRD on CPU – no preprocessed image
- Haar on CPU
- LRD on GPU

Variants of the CPU implementation of the LRD feature include: the described SSE implementation, straightforward C implementation using integral image to evaluate the speed-up achieved by using SIMD instructions, straightforward C implementation without the pre-processed convolution images, but with the convolutions performed on-the-fly.

The evaluation was performed for different resolutions of the image, for different sizes of the classified window and for different amount of the weak hypotheses calculated for each classified window. Note that this evaluation is to determine the evaluation speed of the weak classifiers only, not the overall performance of the boosted classifier. However the performance of the final classifier will be defined by the times spent in low-level features it is composed of.

resol	num wc	frame-time [mili sec] lrdSSE	LRDinteg	LRDsmpl	haarCPU	lrdGPU	time-per-wc [nano sec] lrdSSE	LRDinteg	LRDsmpl	haarCPU	lrdGPU
320x240	5	7.909	16.882	30.570	12.339	0.244	**25.267**	52.974	103.336	**38.816**	**0.872**
320x240	10	16.457	33.113	60.060	23.902	0.527	**25.104**	51.861	95.407	**38.577**	**0.942**
320x240	50	77.958	160.236	274.760	129.223	2.524	**24.731**	50.588	86.786	**40.929**	**0.902**
640x480	5	36.186	73.860	143.153	51.852	1.173	**25.317**	52.466	102.656	**38.694**	**0.810**
640x480	10	67.815	131.385	262.625	98.376	2.232	**24.277**	48.163	94.118	**36.008**	**0.771**
640x480	50	348.138	641.946	1179.700	511.766	11.066	**24.934**	45.998	85.120	**39.057**	**0.764**

Table 1. Performance table for LRDonSSE, LRD using integral image, LRD-simple implementation and LRDonGPU; the table contains the times of sole evaluation of the classifier, the pre-calculation stage is not included

In the following table, a comparison of the performance of the pre-processing stage is given.

resol	LRDconvol	integral	convGPU
320x200	2.523	1.223	0.728
640x480	9.134	10.298	1.226
800x600	13.805	16.413	3.515
1024x768	24.808	27.948	3.750
1280x1024	37.458	45.169	4.530

Table 2. Evaluation of the pre-processing stage (convolutions for the LRD using SSE; integral image both for Haar features and for the variant of LRD; convolutions for LRD using GPU). Times are given in milliseconds.

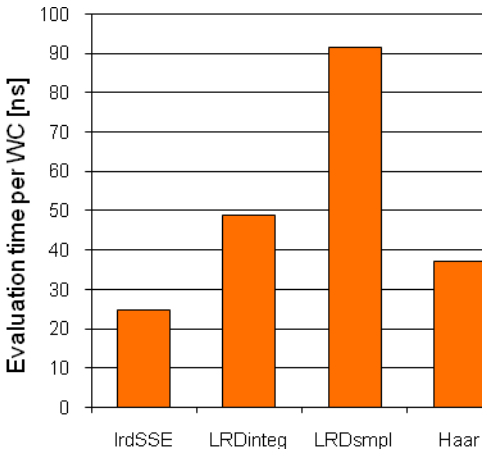

Figure 7. Evaluation times for different kinds of weak classifiers on CPU

The graph in figure 7 visualizes the information from Table 1, comparing the evaluation times for each of the weak classifier variant.

The measurements show that the presented SSE implementation of the LRD compared to the commonly used Haar features exhibits speedup of more than 1.5. The "simple" implementation of LRD that evaluates the *sampling function* on-the-fly is 2.46 times slower than the commonly used Haar features calculated using integral image. The advantage of such LRD's is that no pre-calculated data is required, but the weak classifier is evaluated directly from the input grayscale image. Haar features can also be computed without using the integral image; however, typically, the slowdown will not be less than 10×. This is beneficial for "prototype" implementations not optimized for speed, since only the LRD evaluating function needs to be supplied (no extra data structures), to perform object detection or similar tasks. Embedded devices with limited amount of memory will also benefit from this advantage.

The calculation times for the GPU implementation stick out from all the other measurements by excellent performance. The GPU implementation has been studied by the authors also, and detailed results have been accepted for publication elsewhere [6]. The original purpose of the LRD [12] was to efficiently operate in strongly parallel environments such as FPGA's, but other computational platforms with parallelism (GPU, SSE evaluated here) seem to prefer this feature as well. An important advantage of LRD on different hardware platforms, compared to the commonly used Haar features, is also feasibility of acceleration of the pre-calculation stage. The convolutions that pre-calculate the sampling function are operating locally and can easily be parallelized, but creating the integral image in parallel environments is not efficient or virtually impossible (as in the case of GPU).

5. Conclusions and Future Work

This paper presents an experimental implementation of the Local Rank Differences image feature using the SSE instruction set on the CPU and its comparison to other approaches, specifically to the Haar features on

the CPU and LRD on the GPU.

The LRD features seem very well suitable for pattern recognition by image classifiers. They exhibit inherent grayscale transformation invariance, ability to capture local patterns, and the ability to reflect quantitative changes in lightness of image areas.

The authors of this paper are currently working on an efficient implementation of the whole WaldBoost engine utilizing the LRD features either using SSE.

Although the implementation of the LRD on GPU, which is used in the comparison (section 4) is efficient (by using Cg programming language at the moment), better implementations and variation of the LRD will also be looked for on the GPU platform and different possible GPGPU environments (such as CUDA).

In any case, the results of the presented work definitely lead in a conclusion that the Local Rank Differences features represent a vital low level image feature set, which outperforms the commonly used Haar features in several important measures. Fast implementations of object detectors and other image classifiers should consider the LRD as an important alternative.

6. Acknowledgments

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics), by the research project "Security-Oriented Research in Informational Technology" CEZMŠMT, MSM0021630528, and by Czech Grant Agency, project GA201/06/1821 "Image Recognition Algorithms".

10. References

[1] General-Purpose Computation on GPUs, (available as of 2008-07-02 at http://www.gpgpu.org)

[2] Kadir, T., Brady, M.: Saliency, Scale and Image Description. International Journal of Computer Vision, Volume 45, Number 2 / November 2001.

[3] Lienhart, R., Maydt, J.: An extended set of Haar-like features for rapid object detection, ICIP02(I: 900-903).

[4] Michel, P. et al: GPU-accelerated Real-Time 3D Tracking for Humanoid Locomotion and Stair Climbing, Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007

[5] Ojala, T., Pietikäinen, M., Mäenpää, T.: Gray scale and rotation invariant texture classification with local binary patterns. In: Computer Vision, ECCV 2000 Proceedings, Lecture Notes in Computer Science 1842, Springer (2000) 404-420.

[6] Polok, L., Herout, A., Zemčík, P., Hradiš, M., Juránek, R., Jošth, R.: "Local Rank Differences" Image Feature Implemented on GPU, accepted for publication at ACIVS 2008, France

[7] Schapire, R., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. In: Machine Learning, 37(3):297-336, 1999

[8] Sinha, S.N., Frahm, J.M., Pollefeys, M., Genc, Y.: GPU-based Video Feature Tracking And Matching, Technical Report TR 06-012, Department of Computer Science, UNC Chapel Hill, May 2006

[9] Šochman, J., Matas, J.: WaldBoost — Learning for Time Constrained Sequential Detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2

[10] Šochman, J., Matas, J.: Learning A Fast Emulator of a Binary Decision Process. In ACCV 2007.

[11] Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR, 2001

[12] Zemčík, P., Hradiš, M., Herout, A.: Local Rank Differences - Novel Features for Image, In: Proceedings of SCCG 2007, Budmerice, SK, 2007, s. 1-12

[13] Intel, SSE instruction set, (avail. as of 2008-07-02 at: http://download.intel.com/design/processor/manuals/253665.pdf