Local Binary Pattern Based Texture Analysis in Real Time using a Graphics Processing Unit

Gregor Zolynski, Dipl. Inform. Tim Braun, Prof. Dr. Karsten Berns
TU Kaiserslautern; AG Robotersysteme; Gottlieb-Daimler-Str. 1
67663 Kaiserslautern; Tel. 0631-2052621; braun@informatik.uni-kl.de

Abstract
This paper presents a novel implementation of the “Local Binary Pattern” (LBP) texture analysis operator on a consumer-grade graphical processing unit (GPU), which yields a 14 to 18-fold run time reduction compared to standard CPU implementations. The obtained result allows the application of this powerful texture analysis method for tasks requiring both high resolution and real time performance, such as mobile outdoor robotics or industrial inspection, without the need for specialized and thus costly hardware components.

Keywords / Relevant Areas
Computer Vision, Outdoor Robotics

1. Introduction

The analysis of image texture is well suited for several difficult tasks in computer vision and robotics. In the area of outdoor robotics, texture analysis is often used to estimate terrain traversability [1] or to discriminate between roads and their surroundings [2]. For these outdoor scenarios, the high robustness of textural features against environmental changes such as lighting variations is a key benefit. Another important application is quality inspection, with successful examples encompassing the quality control of paper [3] and ceramics [4].

However, a serious drawback of many sophisticated approaches is their enormous computational complexity. Especially for mobile robots which have limited resources, texture analysis of high resolution images in real time is almost infeasible. In the past, specialized hardware components have been proposed as a solution [5]. Unfortunately, these devices are often costly, not widely available and lack flexibility. In contrast to this, today’s cheap and freely available customer-grade graphics cards have become programmable enough so that they can be used to efficiently solve such problems.

In this paper, we present a reformulation of the powerful ‘Local Binary Pattern’ multi-scale texture feature extractor [8] that can be efficiently executed on a GPU. The new algorithm is integrated into a pipeline framework that handles the low-level data flow between different GPU program elements. This concept can be easily transferred to other GPU-based algorithms.

2. GPU based Computation of Local Binary Patterns

Local Binary Patterns have been introduced by Ojala [7] and have undergone several modifications [8]. Loosely speaking, the *LBP value* (*LBP pattern, LBP code*) of a pixel captures the structure of local brightness variations around it. Algorithmically, the value is computed by sampling circularly around the selected pixel and setting 1-bits in the LBP value for each sample that is brighter than the center (See fig. 1 for an example using 8 samples). A feature vector describing the textural properties of a given area can be computed by calculating a histogram of the LBP values located inside this area and subsequently used for texture classification etc.

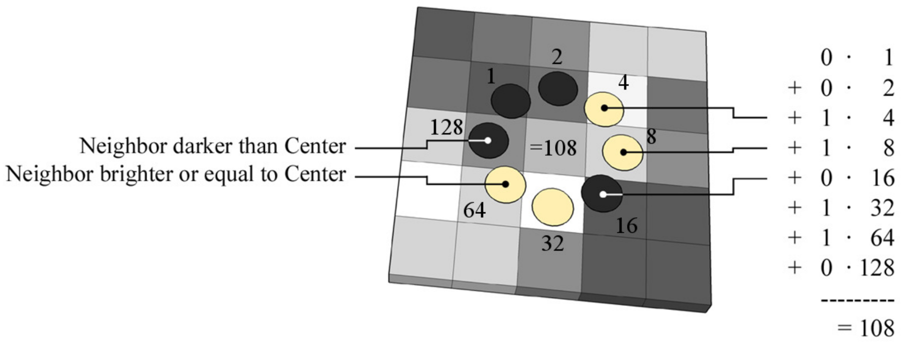

Figure 1: LBP-8 computation for a single pixel.

For the GPU implementation, we focus on the computation of the LBP values themselves. We build a transformation which takes a normal grayscale input image and produces an ‘LBP image’ where each pixel contains the result of the LBP operator applied to the corresponding input pixel. As operator, we use both the multi-scale and uniform extensions of the Local Binary Pattern, which improve results for texture classification tasks [6], [8]. The multi-scale variant computes multiple LBP values per pixel with increasing sample radii, thus capturing textural properties on different scales. In order to avoid aliasing artifacts, the samples need to be gathered from a larger area than a single point. This quickly becomes inefficient, since every single sampling operation is performed by multiple reading operations. Fortunately, there is a solution to this issue. Instead of averaging the gray values of a region every time a sample is taken, the sampling is preceded by appropriate low-pass filtering of the whole input image with a Gaussian filter. The sizes of sampling areas are the same for every sampling point in a ring, therefore for every radius only one blurring filter is needed. Its sigma can be derived from the size of the related sampling areas.

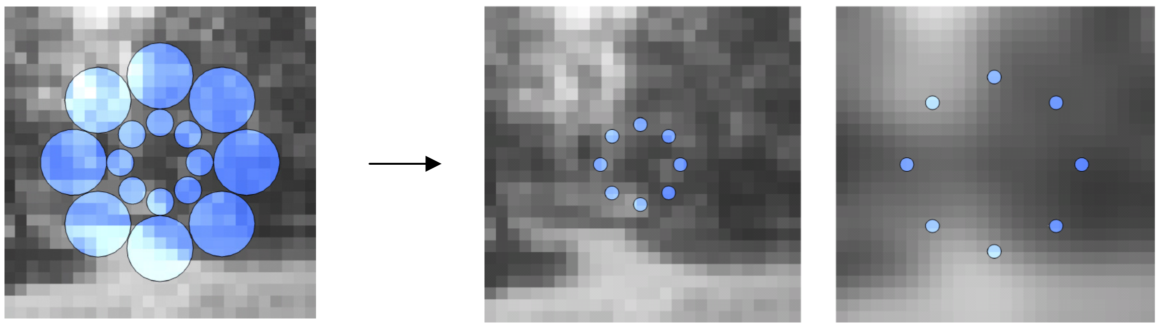
Figure 2: An exemplary two-scale-LBP with big sampling areas (left hand side). Separated rings with prior Gaussian filtering (right hand side) improve sampling performance.

For uniform LBPs, an additional mapping stage projects several rare LBP values representing irregular structures into just one bin, which shrinks the histogram and increases numerical stability [6].

The concrete algorithm is implemented as a sequence of GPU fragment shader programs using NVIDIA's Cg programming language. Each shader program takes an image and transforms it into an output image, which serves as basis for the next program. We propose to employ the fragment shader's capability of processing four floating point 'channels' in parallel (usually calculating RGBA color values) to compute a multi-scaled LBP operator with four scales, with the sampling points arranged as indicated by fig. 3. Each of the four LBP values is based on 8 samples.

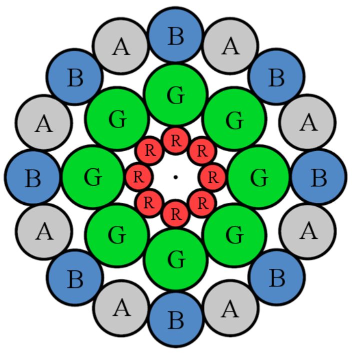

Figure 3: LBP-8-8-16 constructed from four independent rings.

We utilize the Cg Framework by Microsoft/NVIDIA to implement the GPU programs. This framework incorporates a compiler and a special runtime engine allowing quick development of GPU shader programs. But being designed as a system for visual effects, its interface is rather cumbersome when 'misused' for general purpose computing. Thus we created another layer of abstraction by wrapping the Cg Framework into a much simpler, pipelined form. This pipeline consists of a chain of Cg Programs, which each read an input texture, perform some computation and write the results onto an output texture. The pipeline framework hereby allocates and manages both the program control flow and the textures used for in- and output. In order to save memory on the GPU and to avoid costly memory allocation operations, a ping-pong texture technique is employed, where two textures A and B are alternately used for input or output (fig. 4).

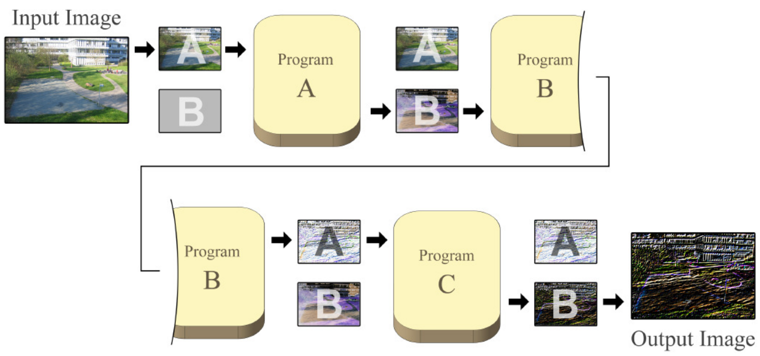

Figure 4: Schematic of the GPU pipeline using ping-pong textures for data transport.

Going from the abstract view of the pipeline framework to the concrete implementation, the shader program sequence executed in order to compute the LBP image is depicted in fig. 5.

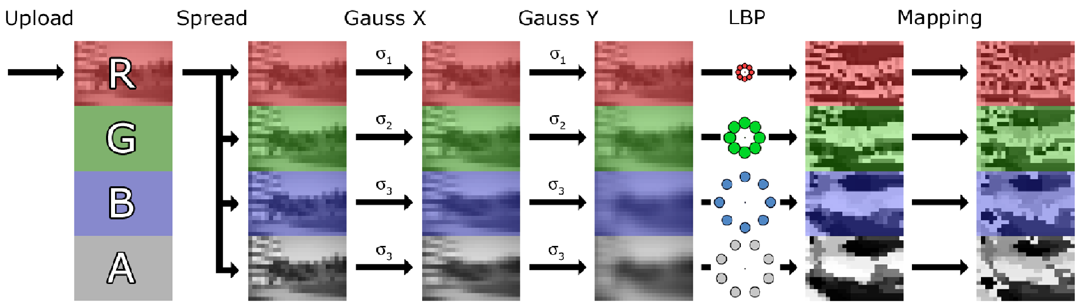

Figure 5: Sequence of GPU Shader Programs for LBP Image Computation

After uploading the grayscale input image, it is first spread (copied) over all color channels. Afterwards, each channel is blurred in two steps using separated Gaussian filter kernels. The kernel sigma is chosen according to the sampling radius of the corresponding LBP ring. Note that since the two largest LBP rings (stored in the blue and alpha channels) have equal radii, their sigma values are also equal. The core step of the algorithm is the computation of the LBP values from the blurred images. For this, a branch-free and vectorized reformulation of the standard algorithm has been developed (table 1).

Table 1: a) Pseudo-Code for naïve CPU LBP Implementation b) Vectorized GPU Code

```
int pattern = 0;
int centerVal = takeSample(center);

for (int i=0; i<numSamples; i++) {
  int neighborValue =
        takeSample(neighbor[i]);
  if (neighborValue > centerVal) {
     setBit(pattern, i, 1);
  }
}
```

```
int4 patterns = int4(0, 0, 0, 0);
int4 centerVals = take4Samples(center);
int mult = 1;
for (int i=0; i<numSamples; i++) {
  int4 neighborVals =
         take4Samples(neighbor[i]);
  bool4 predicate =
          (neighborVals>centerVals);
  patterns += (int4(predicate) * mult);
  mult *= 2;
}
```

An important feature of the GPU algorithm is the explicitly performed 4-fold predication using the variable 'predicate'. This removes the computationally expensive branching required in the naïve implementation and allows the shader program to run at full speed. As a further advantage, whenever the 'take4Samples' method needs to sample points that are not exactly aligned with actual pixels, the automatic interpolation offered by the graphics hardware can be exploited at no extra speed penalty. The final step of the processing pipeline consists of the value mapping required for the *uniform LBP* extension and is realized as a simple texture lookup.

3. Experiments and Results

Experiments were carried out using GeForce 7600 GT, 8600 GT and 8800 GTS type graphics cards measuring the speed and the accuracy of the implemented operator. As evaluation basis, 43 self-made, typical input images with dimensions of 1024x1024 pixels were used as well as 33 test images[1] published by the Signal & Image Processing Institute (University of South California) with dimensions 512x512. In the following, the results obtained with the second data set will be presented in order to allow future comparisons.

Accuracy

The first experiment compared the accuracy of the proposed GPU operator with a reference implementation based upon a C++ port of the code published by the original LBP authors.

[1] http://sipi.usc.edu/database/database.cgi?volume=textures

Since the reference implementation is based on 32 bit integer computation and the GPU implementation uses 16-bit floating point accuracy (‘half’ values in Cg) to achieve faster processing speeds and enable hardware-accelerated linear interpolation, slight divergences of the computed LBP values were expected.

As can be seen in fig. 6, between 0.2% and 1% of the computed values differed from the C++ implementation, depending on the input image. Also, one can observe an overall accuracy improvement between NVIDIA’s GeForce 7 and GeForce 8 chipset families, as the GeForce 8 cards typically exhibit about 20-40% fewer errors than the GeForce 7 cards. Overall, the comparisons showed that on average, less than **0.5%** of the computed values are inaccurate, which was seen as acceptable.

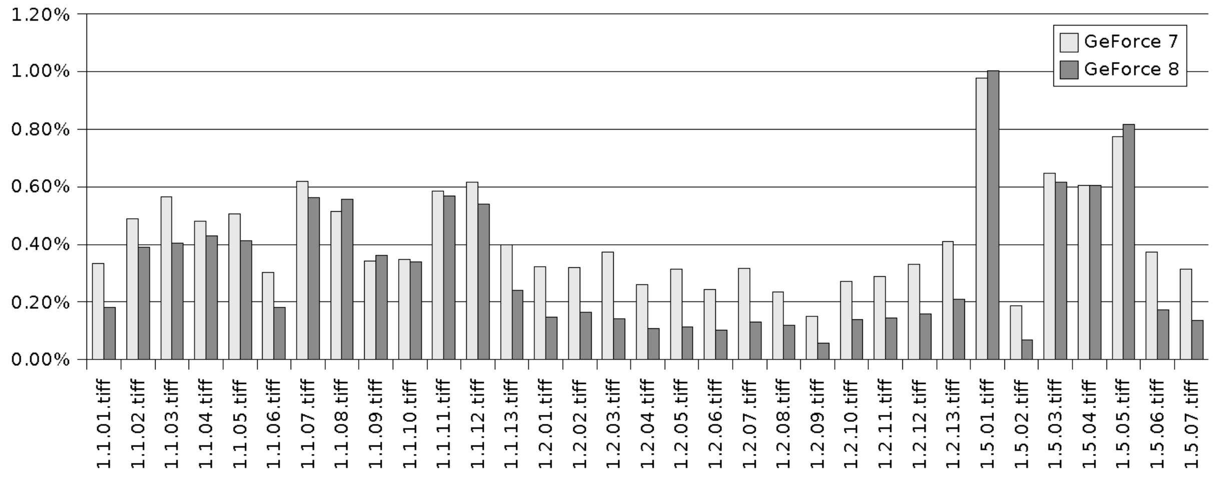

Figure 6: Percentage of LBP values differing between GPU and reference implementation for the South California SIPI Dataset

Speed

The project’s foremost aim was to achieve real time capability on available and comparably cheap hardware. Being massively parallel by design, the Local Binary Pattern operator is very well suited to be implemented on the likewise parallel architecture of a GPU processor. Since every LBP pattern is independent of other patterns, any amount of patterns can be computed simultaneously. The limit to the number of concurrent executions is only set by the amount of shading pipelines within a graphics processor, which is as much as 96 for the GeForce 8800 GTS.

Running time tests were performed on images with dimensions of 1024x1024 pixels. The complete program was run 10,000 times and the measurements were averaged. The results are presented in table 2.

Table 2 shows the execution speed of the three graphics adapters compared with the CPU reference implementation. Even the slowest GeForce 7600 GT adapter beats the CPU by a factor of almost 14. On the other hand, differences between cards are small, since most of the time is spent in the data up and download phase.
Note that the CPU implementation was running on a single core only. Assuming that the bottleneck of CPU computation is the processor and not the memory (we consider this very unlikely, as the core did not reach peak performance during the tests), its execution time could be lowered to a quarter, which is still about 3.5 to 4.0 times slower than our GPU implementation, and blocks the entire CPU, preventing any other (navigation-related) computation.

Table 2: Speed Comparison CPU / GPUs

CPU / GPU	Total Processing Time	Upload	Computation	Download
Core2Quad 2.4GHz	**1150 ms** (1 core used)	-	1150 ms	-
GeForce 7600 GT	83 ms	17 ms	20 ms	46 ms
GeForce 8600 GT	72 ms	17 ms	11 ms	44 ms
GeForce 8800 GTS	65 ms	17 ms	5 ms	43 ms

The relative execution durations for each single Cg programs is presented in fig. 7. The percentages of run time are independent of the input image's dimensions; they scale linearly with the amount of input data. The two separated Gaussian blurring operations and the mapping to uniform LBP each take up about 10% - 11% of the entire execution time. Spreading the data from one texture channel into the other three channels could have been done for no extra cost in one of the Gaussian blurring programs, but was implemented as standalone program in behalf of modularity. It takes up around 7% of the running time. The remaining 70% - 75% are spent by the LBP pattern computation.

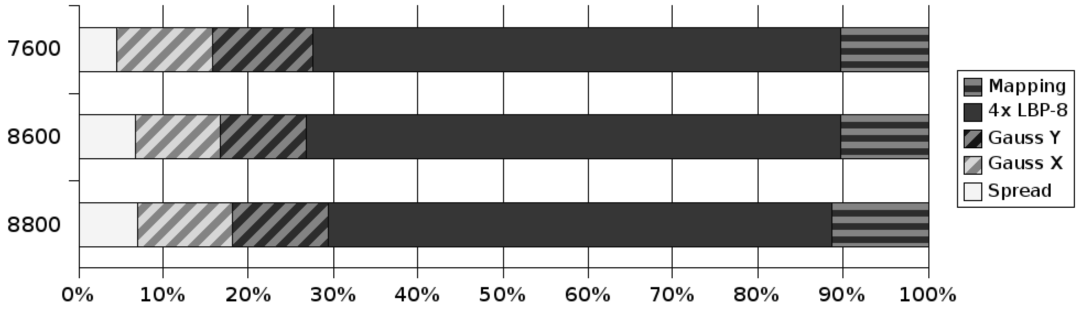

Figure 7: Running time fractions for the GPU LBP pipeline. Data transfer times are excluded.

4. Conclusion and Future Work

In this paper, we presented a reformulation of the multi-scaled Local Binary Pattern Texture Analysis Operator on customer-grade GPUs. Results show that even using older graphics cards, 10 or more fps can be attained, while the CPU remains idle except for the data transfer. Thus, texture analysis becomes possible even for systems with limited resources, such as mobile outdoor robots.

Future work includes the integration of the presented approach on the mobile outdoor robot RAVON developed by the AG Robotersysteme in Kaiserslautern into a texture-based terrain traversability estimation system that is currently being researched. Also, the developed GPU computation framework shall be further extended and consolidated as a preparation for a public release under the GPL.

[1] Halatci, I. Brooks, Christopher A.; Iagnemma, K.
Terrain Classification and Classifier Fusion for Planetary Exploration Rovers
2007 IEEE Aerospace Conference, March 3-10 2007 Page(s):1 - 11

[2] Rasmussen C.: Combining Laser Range, Color, and Texture Cues for Autonomous Road Following, Proceedings of the ICRA 2002

[3] Turtinen, M.; Pietikäinen, M.; Silvén, O.; Mäenpää, T; Niskanen, M: Texture-based Paper Characterization Using Non-Supervised Clustering. Proc. 6th International Conference on Quality Control by Artificial Vision (SPIE vol. 5132), May 19-23, Gatlinburg, Tennessee, 2003, 350-358.

[4] López, F.: Real-Time Surface Inspection of Ceramic Tiles. PhD Thesis, Department of Systems Data Processing and Computers, Technical University of Valencia, Spain. 2005

[5] Laiho, M.; Lahdenoja, O.; Paasio, A.: Dedicated Hardware for Parallel Extraction of Local Binary Pattern Feature Vectors.

[6] Mäenpää, T.; Ojala, T.; Pietikäinen, M.; Maricor, S.: Robust Texture Classification by Subsets of Local Binary Patterns. 15th International Conference on Pattern Recognition, Barcelona, Spain, 2000. 3:947-950

[7] Ojala, T.; Pietikäinen, M; Harwood, D: A Comparative Study of Texture Measures with Classification Based on Feature Distributions. Pattern Recognition 29(1) 1996, 51–59

[8] Mäenpää, T: The Local Binary Pattern Approach to Texture Analysis – Extensions and Applications. PhD Thesis. University of Oulu, 2003