Implementing Haralick Features on GPU for Pavement Distress Detection

Kristina Doycheva¹, Christian Koch², and Markus König³

¹) Dr.-Ing. Candidate, Department of Civil and Environmental Engineering, Ruhr-University Bochum, Bochum, Germany. Email: kristina.doycheva@rub.de
²) Dr.-Ing., Assoc. Prof., Department of Civil Engineering, The University of Nottingham, Nottingham, UK. Email: christian.koch@nottingham.ac.uk
³) Dr.-Ing., Prof., Department of Civil and Environmental Engineering, Ruhr-University Bochum, Bochum, Germany. Email: koenig@inf.bi.rub.de

Abstract:

The condition of municipal roads has deteriorated in recent years, leading to pavement distress such as cracks or potholes. In order to enable road maintenance, pavement distress should be timely detected. However, manual investigation, which is still the most widely applied approach towards pavement assessment, puts maintenance personnel at risk and is time-consuming. During the last decade several efforts have been made to automatically assess the condition of the municipal roads without any human intervention. Vehicles are equipped with sensors or cameras in order to collect data related to pavement distress or record videos of the pavement surface.

Yet, this data is usually not processed while driving, but it is instead saved persistently and later analyzed offline. As a result, a vast amount of memory is required to store the data and the available memory may not be sufficient. To reduce the amount of saved data, the authors have previously proposed a GPU-enabled pavement assessment approach based on the wavelet transform of pavement images. The GPU implementation enables pavement distress detection in real time. Although the method used in the approach provides very good results, the method can still be improved by incorporating pavement surface texture characteristics.

This paper presents an implementation of Haralick features on GPU for pavement distress detection. Haralick features are based on gray tone spatial dependencies in an image and characterize the image texture. To evaluate the computational efficiency of the GPU implementation, performance tests are carried out. In addition, classification results obtained by applying the approach on 1000 pavement images are compared to the results without integrating Haralick features. The results of both tests show that reliable detection of pavement distress is achieved in real-time.

Keywords: pavement distress detection, texture features, Haralick features, graphics processing units

1. INTRODUCTION

Evaluating the pavement condition is a mandatory process which enables road maintenance. However, the evaluation of the pavement condition is still mostly performed manually by trained personnel. As a consequence, the time required to inspect roads and analyze pavement data often does not comply with the requirement for detection of pavement deterioration at the early stages. Therefore, efforts to automatically assess the condition of the roads using cameras and sensors installed on vehicles have been made in the latest years. Various methods for pavement assessment and evaluation have been proposed, namely sensor-based and visual-based methods (Koch et al., 2015).

Yet, most of these methods do not detect key indicators of deteriorating pavement condition in real-time, since state-of-the-art CPU processors which are used for pavement image processing are not able to execute the corresponding algorithms fast enough.

Nevertheless, current technology provides us with an opportunity to implement these methods on various architectures in order to accelerate the execution and achieve efficiency. On one hand, Field Programmable Gate Arrays (FPGA) are increasingly used for high-performance computing. On the other hand, Graphics Processing Units (GPU) have emerged as a powerful tool for massively parallel computations in recent years (Owens et al., 2005).

We already have proposed an implementation of the wavelet transform for pavement distress detection on GPU (Georgieva, 2015). Although promising results have been achieved, the adopted method for pavement distress detection needs enhancement due to the variety of different textures roads are characterized with, which makes it impossible to develop universal methods capable of detecting distress on various types of roads. Therefore, in state-of-the-art pavement assessment studies, the methods for distress detection have been optimized for a certain type of pavement. However, if fully automated pavement distress detection is pursued, the methods should be
able to detect distress on all types of roads independently from the texture.

Figure 1 presents some examples of images of pavement with different texture. The wavelet-based descriptors for these images were calculated as described in (Georgieva, 2015; Zhou et al., 2006) after pre-processing the images to correct the non-uniform background illumination. In Figure 1, the values of the wavelet descriptors for different images are presented. As can be seen, the high-amplitude wavelet coefficient percentage (HAWCP) value of the image in the middle which contains cracks is lower than the HAWCP value of the good pavement image on the right.

![Figure 1. Pavement images with different texture and corresponding wavelet descriptor values](image)

To address this issue and compensate the limitation of the method to distinguish between different types of pavement, this paper presents a method capable of detecting distress on pavement images with various textures by incorporating texture features. The method was implemented for Graphics Processing Units using the Open Computing Language (OpenCL).

The method is based on texture features which, in turn, are calculated depending on gray-tone spatial dependencies in the images.

The remainder of this paper is organized as follows. Background information about the texture features is provided in the next section, while the methodology is presented in section 3. The implementation is described in section 4. To evaluate the efficiency of the implemented method both in terms of performance and distress detection, several tests were carried out. The tests are presented, including the results, in sections 5 and 6. The paper concludes with a summary and suggestions for future research.

### 2. HARALICK FEATURES

Texture is an important characteristic in image analysis and contains information about the spatial distribution of tonal variations within an image. An exact definition of texture does not exist. Several scientists define texture depending on the purpose they need texture for or on their applications. Jain (Jain et al., 1995), for example, defines texture as “repeating patterns of local variations in image intensity which are too fine to be distinguished as separable objects at the observed resolution”. Texture analysis comprises texture classification, texture segmentation, and shape recovery from texture. We are particularly interested in texture classification, whereby the problem is identifying the given textured region from a given set of texture classes such as for example agricultural land, forest region or urban area (Jain et al., 1995).

With the aim of enabling texture classification, Haralick (Haralick et al., 1973) has proposed a set of meaningful texture features for classification of pictorial data. By conducting several experiments, he has concluded that texture features have applicability for a wide variety of image-classification applications. The set consists of 14 textural features which can be extracted from a so-called gray-tone spatial-dependence matrix (or gray-level co-occurrence matrix).

Haralick assumes that an image is a set of resolution cells (i.e. pixels) and the classification of pictorial data can be performed either on a resolution cell basis or on a block of resolution cells. If a large block of resolution cells is observed, a procedure for extracting textural properties of these blocks is required. The procedure proposed by Haralick first computes a set of gray-level co-occurrence matrices (GLCMs) and then calculates textural features based on these matrices.

The GLCM expresses the relationship between adjacent or nearest-neighbor pixels in an image. According to Haralick’s notion, a pixel has eight nearest neighbors, as shown in Figure 2, where 1 and 5 are 0 degree (horizontal) nearest neighbors to the pixel in the middle (•). Pixels 2 and 6 are 135 degree nearest neighbors.
Pixels 4 and 8 are 45 degree nearest neighbors, and pixels 3 and 7 are 90 degree (vertical) nearest neighbors to *. To calculate the GLCM, the numbers of all possible nearest-neighbor gray-tone pairs in an image are counted. In the gray level co-occurrence matrix each cell represents the number of occurrences of pixels with intensity $a$, which are 0 degree nearest neighbors to pixels with intensity $b$.

$$f_1 = \sum_i \sum_j (p(i,j))^2$$  \hspace{1cm} (1)

Contrast:

$$f_2 = \sum_{n=0}^{N_g-1} n^2 \left( \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \text{ with } |i - j| = n \right)$$  \hspace{1cm} (2)

Inverse difference moment (IDM):

$$f_3 = \sum_i \sum_j \frac{1}{1 + (i - j)^2} p(i,j)$$  \hspace{1cm} (3)

Entropy:

$$f_4 = -\sum_i \sum_j p(i,j) \log(p(i,j))$$  \hspace{1cm} (4)

3. METHODOLOGY

With the aim of improving the current pavement evaluation framework, the texture features are incorporated into an existing pavement distress detection methodology. An overview on the methodology is presented in Figure 3. First, the pavement images are pre-processed by applying a median filter in order to remove noise. Then, another pre-processing step is performed, namely the top-hat transform is applied, to correct the non-uniform illumination in the images. As a result of the top-hat transform, shadow regions such as the one present on the right image in Figure 1 are corrected and misleading results are avoided. Afterwards, the wavelet transform of the images is computed and the high-amplitude wavelet coefficient percentage (HAWCP) is calculated.

The contribution of this paper is the blue part on Figure 3. In addition to the HAWCP value, texture features are calculated to enhance the classification performance. The texture features are used together with the HAWCP descriptor in order to generate a classification model. The classification model is created based on training images which are manually labeled by experts. Then, the classification model is used to evaluate the presence of distress on new images in real-time.
Figure 3. Overview on the pavement distress detection methodology

4. IMPLEMENTATION

4.1 Gray Level Co-occurrence Matrix

The idea of the implementation of the GLCM computation is to use multiple work-groups to compute different parts of the GLCM. The approach is shown in Figure 4. The implementation of the GLCM computation is based on local memory. Since local memory access is much faster than global memory access, local memory is used to store partial GLCMs computed by different work-groups. Depending on the amount of available memory, the GLCM is split into tiles. The more local memory is available, the smaller is the number of the work-groups, because one work-group can cover a bigger part of the GLCM. The values belonging to the different tiles are calculated by the different work-groups. Each work-group processes the whole image, but uses only one part of the GLCM. This means that different work-groups update different parts of the GLCM, so there are no racing conditions between the work-groups.

The kernel comprises several steps. First, a local memory array is created for each work-group. This local array stores the values of the corresponding part of the GLCM. For example, if the pixel values are between 0 and 255, the complete GLCM would have 65,536 elements. If we use four work-groups, the local GLCM arrays would have a size of $65,536/4 = 16,384$ elements. To initialize the local array, the values of the array elements are set to
zero in parallel. For this purpose, each work-item in the workgroup iterates over work-group-sized pieces of the local GLCM array and sets the corresponding value to zero.

In the second step, the work-items iterate over their parts of the whole image and compute the GLCM. Here it is necessary to check if the current work-group is responsible for the GLCM cell. For this purpose, we check whether the position of the GLCM cell which we want to increment is within a certain range. For instance, in our example with the four work-groups the range of the first work-group would be 0 – 16,383, the range of the second one would be 16,384 – 32,767. To increments the GLCM values, the OpenCL built-in function atomic_inc is used.

After that, following a synchronization barrier, the GLCM parts are copied to global memory.

![Figure 4. Schematic of the GLCM implementation](image)

4.2 Haralick Features

As can be seen from the equations above, the calculation of the texture features is based on summation of the GLCM values. As a result, one and the same approach can be applied for the OpenCL implementation of the four features. Then, the implementation is modified according to the corresponding feature (e.g. we take the square of the values for the angular second moment).

Summations are examples of the so-called reduction operations. They are called reduction operations because a vector of data is reduced to a single element (e.g. the sum of the elements). Another example of a reduction operation is finding the maximum element of a vector.

AMD and NVIDIA have proposed several approaches to implement reductions on GPUs (AMD, 2014; Harris, 2007). The most intuitive approach is the tree-based approach, where the sums of pairs of elements are calculated. In this case, the vector which initially contained N elements is reduced to N/2 elements in the first step. The process continues recursively until the vector is reduced to a single element. However, the number of active work-items gets smaller with each reduction step, which results in poor SIMD efficiency.

AMD and NVIDIA have observed better performance by applying two-stage reduction. The idea behind the two-stage reduction is that instead of parallelizing the reduction as much as possible, we combine sequential and parallel reduction.

The input array (which in our case contains all the GLCM values) is split into multiple chunks. Each work-item loops over its parts of the chunks and performs sequential reductions. In Figure 5, the dark blue work-item computes sequentially the sum of all dark blue elements. Assuming that we have two work-groups (a blue workgroup and a red workgroup), all the work-items within a group perform the same operation as the dark blue work-item and they write their temporary results into an array located in local memory. In the case with two workgroups, there exist two local arrays. Then, the local arrays are reduced to a single element with parallel reductions. At the end of these parallel reductions, each work-group writes its partial result into a global memory array. When more than one workgroup is required (i.e. when we have so many elements that they don’t fit in a single work-group), it is necessary to perform one more reduction on the result to merge the partial results. For this purpose, a summation reduction kernel is invoked with the result array as input. The latter array is reduced by the kernel to a single element which contains the sum of all elements.
At this point, we should note that programmers usually tend to implement algorithms in such a way, that logically independent pieces of code are defined in different code fragments (e.g. in functions). If this programming paradigm is applied, each Haralick feature should be implemented in its own kernel. However, since more than one feature is required to accurately classify a texture, it is sensible to combine the computations of a set of features in one kernel. Thus, the performance of the implementation is further enhanced, because some instructions are performed once instead of four times.

5. PERFORMANCE EVALUATION

To evaluate the performance of the proposed implementation, performance tests are carried out. The tests are performed on an Intel(R) Core(TM) i7-4600U CPU @2.10GHz and an Intel(R) HD Graphics 4400 GPU. To guarantee (ensure) for accurate measurements, the computational routine for the calculation of the four features is invoked 10 000 times. Then, the average execution time is used to deduct a conclusion on the performance enhancement.

5.1 Performance evaluation GLCM

Figure 6 presents the results for the GLCM computation for images with a resolution of 256x256, 512x512, 1023x1024, and 2048x2048 pixels. To evaluate the performance results, the speed-up achieved by executing the computations on GPU was calculated as defined in Equation 5.

\[
\text{Speed-up} = \frac{\text{Sequential execution time}}{\text{OpenCL execution time}}
\]  

The speed-up increases with the resolution of the images. In case of images of a size of 2048x2048 pixels, the speed-up is approximately 39. However, in all cases the OpenCL implementation is much faster than the sequential one.
5.2 Performance evaluation Haralick features

Three different implementations are compared for performance evaluation: a sequential C++ implementation, a GPU implementation where four separate kernels are defined for the individual features, and a GPU implementation where a single kernel is used to compute all the features as described in section… Costly operations which do not belong to the set of mandatory operations required to calculate the features are omitted when measuring the execution time in order to make sure that exactly the same set of operations is wrapped in all cases. For precise measurements of the OpenCL execution time, the OpenCL profiling functionality is utilized. When evaluating the performance of OpenCL it is recommended (Intel Corporation, 2013) to not only measure the kernel execution time, but to also take into account the time required to transfer data between the host and the device, i.e. the time to write the input values into the buffer and the time to read the result. Hence, the OpenCL execution time is computed as shown in Equation 6:

\[
\text{Execution time} = \text{data transfer time from the host to the device} + \text{kernel execution time} + \text{data transfer time from the device to the host}
\]

(6)

The results of the test are presented in Table 1. The GPU execution of the implementation is approximately 126 times faster than the sequential execution which in average took 6.85 milliseconds.

Table 1. Time required to calculate the Haralick features

<table>
<thead>
<tr>
<th>Execution type</th>
<th>Average time in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential C++</td>
<td>6.85200</td>
</tr>
<tr>
<td>OpenCL</td>
<td>0.05424</td>
</tr>
</tbody>
</table>

5.3 Overall performance

The execution times presented above include also the time required to transfer the GLCM data between the host and the GPU device. As a result, the performance enhancement would be even greater if we do not transfer the GLCM to the host after computing it with the GPU. The GLCM stays in the device memory buffer and is used directly to compute the Haralick features. In this way, two data transfers are spared, because we also do not need to transfer GLCM data to the device for the Haralick features kernel.

6. CASE STUDY

To validate the pavement assessment approach, a passenger vehicle equipped with a camera and a GPS receiver was used to collect pavement images. A road segment in Germany was chosen for the case study. Two different types of pavement surface as shown in Figure 1 were present on the road. In total, 1549 images were acquired using the camera. In contrast to the general idea of the approach to discard images where no distress is detected, here all images were saved for validation purposes. 66 % of the images were manually labeled as images of distress or of healthy/intact pavement and were used as training data for a machine learning algorithm. In particular, Rotation Forest (Rodriguez & Kuncheva, 2006) was used to generate a classification model based on these training images and classify the remaining 34 % of the images.

In order to evaluate the improvement of the approach, the results of the classification before and after incorporating texture features were compared. The results are presented in Table 2 in terms of the percentage of correctly classified images, precision, recall, and false positive rate.

Table 2. Classification results

<table>
<thead>
<tr>
<th></th>
<th>Correctly classified images in %</th>
<th>Precision</th>
<th>Recall</th>
<th>False positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without texture features</td>
<td>79.1271</td>
<td>0.777</td>
<td>0.791</td>
<td>0.098</td>
</tr>
<tr>
<td>With texture features</td>
<td>98.8615</td>
<td>0.989</td>
<td>0.989</td>
<td>0.003</td>
</tr>
</tbody>
</table>

The results show that a significant improvement of the classification is achieved by incorporating texture features. The percentage of the correctly classified images when using texture features is approximately 19% higher than the one without using texture features, while the false positive rate is lower.
7. CONCLUSIONS

Pavement condition assessment is a key component of road maintenance programs. In current practice, pavement images and data are collected by vehicles equipped with cameras and sensors in order to enable autonomous pavement distress detection. However, the collected data and images are usually processed offline.

The authors have previously proposed a methodology and implementation of a pavement distress detection system, which is capable of processing pavement images in real time. Yet, although good results have been achieved, the methodology could be improved by incorporating texture features.

In this paper, an implementation of Haralick texture features was presented. The implementation is based on the Open Computing Language (OpenCL). To evaluate its performance, the OpenCL implementation was compared against a sequential C++ implementation of the texture features. The results show that a significant improvement in terms of speed-up was achieved. Moreover, by incorporating texture features, better classification results were obtained than without using them, as proved by conducting a case study.

The achieved execution time enables real-time processing and analysis of various pavement images. As a result, the amount of data stored for offline processing is reduced as well as the costs for pavement condition assessment.

Future research and development may include considering other issues related to pavement condition assessment. For example, methods for calculating more accurate distress positions than the ones provided by single GPS receivers will be investigated.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this ongoing project by the German Research Foundation (DFG) under grant KO4311/2-1.

REFERENCES


