GPU Acceleration of Eff\textsuperscript{2} Descriptors using CUDA

Kristleifur Daðason, Herwig Lejsek, Ársæll P. Jóhannsson
Videntifier Technologies ehf.
Ofanleitri 2
101 Reykjavík, Iceland
{kristleifur, herwig, arsaell}@videntifier.com

Björn Póór Jónsson
Reykjavík University
School of Computer Science
101 Reykjavík, Iceland
bjorn@ru.is

Laurent Amsaleg
CNRS-IRISA
Campus de Beaulieu
35042 Rennes, France
laurent.amsaleg@irisa.fr

ABSTRACT

Video analysis using local descriptors requires a high-throughput descriptor creation process. This speed can be obtained from modern GPUs. In this paper, we adapt the computation of the Eff\textsuperscript{2} descriptors, a SIFT variant, to the GPU. We compare our GPU-Eff\textsuperscript{2} descriptors to SiftGPU and show that while both variants yield similar results, the GPU-Eff\textsuperscript{2} descriptors require significantly less processing time.

Categories and Subject Descriptors
I.4.7 [Image Processing and Computer Vision]: Feature Measurement; D.1.3 [Programming Techniques]: Concurrent Programming—Parallel Programming

General Terms
Algorithms, Experimentation, Measurement, Performance

Keywords
cuda, sift, gpu, gpgpu, local image descriptors, image retrieval

1. INTRODUCTION

Video analysis is a central component in many applications, such as video surveillance, news analysis, and video copyright protection. Recent methods for such analysis are typically based on computing many local descriptors per frame, which are then merged to form the video description. As many video analysis applications require real-time performance, high demands are made on the efficient computation of the local descriptors.

1.1 Scaling Descriptor Creation

The traditional method for achieving high throughput is using a computer cluster and large-grain parallelism where the data collection is split into independent parts. The advantage is that the standard non-parallel description code can be used. However, there are two major disadvantages.

Computing clusters are relatively expensive, and it is difficult in practice to deliver cluster-based software products to end users.

An alternative method, solving both problems, is to use powerful graphics processing units (GPU). The advent of highly scalable and parallel yet inexpensive GPUs has been a minor revolution in the computer industry; many projects have therefore evaluated GPUs in a variety of tasks such as feature tracking [8] and local descriptor computation [9].

However, the disadvantage of GPUs is that the description code does not work unchanged. Data and computations have to be adapted to meet constraints on the access patterns and operations available on-GPU. As a result, some computational processes remain incompletely adapted, e.g., forcing data loadback to the host CPU for completion (e.g., see [9]). Fortunately GPUs have now become much easier to utilize due to the recently released CUDA programming environment from NVIDIA [3]. The CUDA model relaxes memory access patterns, and supports a large set of computing primitives.

1.2 Contributions

In the past, large-scale performance studies of description creation have been next to impossible, due to the computing power required for creating the local descriptors. When varying parameters, the descriptors must be created over and over, making the whole process time-consuming. As a result, most such studies have been performed using small collections (e.g., see [4]). With GPU processing, however, large-scale studies easily become feasible. This paper presents initial steps on the path to a large-scale study of all GPU-based variants.

The “gold standard” in local descriptions has been considered the SIFT descriptors, proposed by Lowe in 2004 [7]. Since then, several variants have been proposed, such as PCA-SIFT [4] and the Eff\textsuperscript{2}-descriptors [5]. Previous work showed the Eff\textsuperscript{2}-descriptors to outperform many of the SIFT variants in the context of very-large-scale descriptor databases where small differences in descriptor schemes can have a large effect on retrieval [5]. We therefore adapt the computation of the Eff\textsuperscript{2} descriptors to the GPU through the CUDA environment. We compare the GPU-Eff\textsuperscript{2} descriptors to SiftGPU [9], another GPU-based variant of SIFT, and show that while both GPU-based variants yield similar results (better than SIFT, and comparable to Eff\textsuperscript{2}), the GPU-Eff\textsuperscript{2} descriptors require significantly less processing time.

Note that since performing this comparison we have become aware of the more recent SURF descriptors [1], which are an even faster variant of SIFT, and a GPU version
First, the dominant gradient angle (direction of greatest accuracy) is determined. In the second step, the descriptor itself is created. The descriptor is created around the keypoint, aligned to the dominant gradient angle, and a gradient histogram of 8 bins created in each cell of the grid, obtaining a $4 \times 4 \times 8 = 128$ dimensional histogram of gradient strengths, which is finally normalized.

In this paper, we focus on the Eif$^2$ descriptors, proposed by Lejsek et al. [5], which are a more scalable variant of SIFT. There are three key differences to SIFT. First, relatively more descriptors are created at higher scales, as more scales are considered and gamma correction is applied to the increasingly coarse blurs. Second, a $3 \times 3$ grid is used around the point, resulting in $3 \times 3 \times 8 = 72$ dimensions in total. Third, advanced filters remove descriptors for lines and bright spots, that are ubiquitous and non-discriminative (similar to common words in information retrieval). The Eif$^2$ descriptors have been shown to perform significantly better than SIFT for most image transformations [5].

2.1 SIFT and Eif$^2$ Descriptors

The SIFT local descriptors developed by Lowe [7] have been considered the state of the art in image description computer vision applications. The creation process consists of two steps, and can be roughly outlined as follows. In the first step, keypoint detection, small regions of interest are detected where descriptors may potentially be created. This is done by building a sequence of gaussian blurs at different scales, taking their differences, and then detecting local minima and maxima in the differences across the scales. The keypoint is then localized to a sub-pixel accuracy. In the second step, the descriptor itself is created. First, the dominant gradient angle (direction of greatest contrast) around the keypoint is found. Then a $4 \times 4$ grid is created around the keypoint, aligned to the dominant gradient angle, and a gradient histogram of 8 bins created in each cell of the grid, obtaining a $4 \times 4 \times 8 = 128$ dimensional histogram of gradient strengths, which is finally normalized.

In this section we describe the implementation of Eif$^2$ descriptors in CUDA. We first present the keypoint extraction process and then our major contribution, the parallelization of the descriptor extraction process to fully utilize the GPU.

3. IMPLEMENTATION

In this section we describe the implementation of Eif$^2$ descriptors in CUDA. We first present the keypoint extraction process and then our major contribution, the parallelization of the descriptor extraction process to fully utilize the GPU.

3.1 Keypoint Extraction

The scale space creation process is easily parallelized, as the gaussian blurring of octaves and the subtraction of adjacent blurrings is a prime example of where each thread can be applied to produce pixels independent of other threads.

Once the rough keypoint location is determined, it must be localized as precisely as possible. In the CPU based implementation, this is done by shifting the points by fractional amounts in the two spatial dimensions, and sometimes by moving between scales in the scale space. In the CUDA implementation, however, the two-dimensional shift is implemented, but not the scale-space jumps.

The reason for this change is that on GPUs it is efficient to assign a certain segment of problem space to each thread. In our implementation, one thread’s processing is done in one scale, in one octave. Allowing jumps out of the scale being processed requires either a synchronisation step or significantly more complex interest point detection processing. We determined, through experimentation, that scale jumps increased execution time significantly with no measurable accuracy benefits.

3.2 Descriptor Extraction

While building up a scale-space of an image has been discussed in several papers before, descriptor extraction on the GPU has been too complex for the previously available interfaces. The first of two processing steps in descriptor extraction is determining the dominant gradient angle around the interest point. Next, the descriptor histogram is computed using that angle. The description is aided by Figure 1.

3.2.1 Dominant Angle Determination

To identify the dominant gradient angle around the interest point, we must compute a gradient orientation histogram within a circular window around the point (Fig-
Figure 1: Details of the Descriptor Extraction

3.2.2 Descriptor Histogram Computation

Once the strongest gradient direction has been found, the shape signal must be encoded into a 72-dimensional descriptor, which is created by computing a gradient histogram of 8 buckets for each block of the 3 × 3 grid (Figure 1(b)). As each subhistogram is calculated from a specific area around the point, all 9 areas can be processed in parallel by separate sets of threads.

For each block, we must first calculate the border vertices of those areas with respect to the strongest gradient direction computed above. Each of the 9 threads rotates the four bounding vertices and determines the smallest pixel-aligned bounding box of the rotated square; this area is highlighted in Figure 1(b) and shown in detail in Figure 1(c). Within this bounding box, seven independent parallel threads process the gradient strength with respect to the distance to the interest point and the center of the cell. The computation of the gradient histogram is then the same as when determining the strongest gradient. The pixels are thus read in the order shown in Figure 1(c).

The reason that seven threads are applied to each histogram, is that there are nine areas processed in parallel, and 7 × 9 = 63, which is almost a full set of 64 threads. We experimented with assigning 14 threads to each histogram (for a total of 14 × 9 = 126 threads) but the process was slower due to less efficient register usage, as each thread must store some local information. This trade-off may change with hardware development. Finally, after all 9 × 8 = 72 orientation buckets have been calculated, a single thread normalizes the descriptor.

4. EXPERIMENTAL EVALUATION

In this section we present the results of our experimental evaluation of the four descriptor variants: SIFT, SiftGPU, Eff², and GPU-Eff². We used the default parameter settings for each variant. All experiments were performed using an Intel Q6600 processor and an NVIDIA GTX 280 GPU. First, we compare and analyze the time spent on descriptor extraction, and then we study the result quality of each variant. We follow the experimental methodology of [5].

4.1 Extraction Time

In this experiment, we applied each of the variants to a collection of 29,277 high-quality news images [5]. Table 1 shows the number of descriptors created and the running time. To avoid measuring initialization effects, the descriptors of each image are computed three times, and the average time of the second two runs is used.

As Table 1 shows, descriptor creation on the GPU is almost an order of magnitude faster than Eff² on the CPU and nearly another order of magnitude faster than SIFT. Furthermore, GPU-Eff² performs descriptor creation significantly faster than SiftGPU.

Table 2 shows the execution time for each of the GPU-based variants in more detail. The first two lines show the time required to create the scale-space and to localize the keypoints. As the table shows, this is significantly faster for SiftGPU. The primary reason is that SiftGPU uses fewer
octaves (4 or 5 for our experimental collection, as opposed to 7 for GPU-Eff2) and scales (3 vs. 7 for GPU-Eff2), resulting in significantly less processing (at the expense of lower recognition for some image modifications, as shown below). Another reason is that this part is the earliest code of GPU-Eff2, and SiftGPU is using some well-known optimizations that we have yet to apply.

The third line indicates the time required to gather the keypoints for descriptor extraction. This part is partially implemented using the CPU with SiftGPU and is therefore slower. The final three lines indicate the cost of the descriptor creation itself. The time required for feature orientation, 0.7 ms, is very low as gradient calculation has been performed within the "Build pyramid" step. As mentioned above, the GPU-Eff2 descriptors only extract a single descriptor per keypoint, and hence the multi-orientation is not needed. Finally, the descriptor creation itself also benefits from our efficient histogram calculation, and is significantly faster for GPU-Eff2.

### 4.2 Detection Capability

In order to study the detection capability of the descriptors, we loaded each descriptor collection into an NV-tree index [6]. We then used the same 108 original query images and 26 StirMark modifications used in [5] to evaluate the matching capabilities of the four descriptor variants. As reported in [5], about 41.2% of the SIFT query descriptors find a match from the original image in the top 30 neighbors, while about 57.1% of the Eff2 query descriptors find a match. Both SiftGPU and GPU-Eff2, however, perform similarly to Eff2, finding 57.6% and 58.1% respectively. For individual modifications, however, the tradeoffs are slightly different and Figure 2 presents a representative set of six modifications that we now discuss.

The first two modifications, AFFINE 3 (two-axis affine transformation) and RESC 75 (75% rescaling) represent the majority of the modifications (18 in total). For these modifications SiftGPU, Eff2 and GPU-Eff2 descriptors yield very similar rates, while the SIFT descriptors yield a lower score. Most of these modifications are easy to detect; the lower ratio for SIFT is due to the abundance of descriptors.

The next two modifications, JPEG 15 (compression) and SS 1 (conversion to HSV color format), show an advantage for the Eff2 variants compared to the SIFT variants. The reason is that these modifications remove some of the finer detail in the images, showing the advantage of the Eff2 variants’ emphasis on higher-level descriptors. This effect is seen in a total of six modifications.

The last two modifications in Figure 2, CONV 1 (low brightness) and CROP 75 (75% central crop), show a better detection rate with the SIFT variants than with the Eff2 variants. This phenomenon was already described in [5] and is a result of the emphasis of the SIFT variants on low-scale, low-octave descriptors. Note that these were the only two modifications where the SIFT variants performed better.

### 5. CONCLUSION

This paper has given an overview of the porting of the Eff2 descriptors onto a GPU using the CUDA programming model. The experimental evaluation shows a significant speed advantage, not only over CPU implementations, but also over SiftGPU. We have also shown, however, that GPU-Eff2 still has room for improvement in scale-space computation and keypoint detection efficiency, which we aim to address in our future work.

### 6. REFERENCES