A modular CUDA-based framework for scale-space feature detection in video streams

M. Kinsner\(^1\), D. Capson\(^1\) and A. Spence\(^2\)

\(^1\) Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ontario, Canada
\(^2\) Department of Mechanical Engineering, McMaster University, Hamilton, Ontario, Canada
E-mail: kinsnemh@mcmaster.ca, capson@mcmaster.ca, adspence@mcmaster.ca

Abstract. Multi-scale image processing techniques enable extraction of features where the size of a feature is either unknown or changing, but the requirement to process image data at multiple scale levels imposes a substantial computational load. This paper describes the architecture and emerging results from the implementation of a GPGPU-accelerated scale-space feature detection framework for video processing. A discrete scale-space representation is generated for image frames within a video stream, and multi-scale feature detection metrics are applied to detect ridges and Gaussian blobs at video frame rates. A modular structure is adopted, in which common feature extraction tasks such as non-maximum suppression and local extrema search may be reused across a variety of feature detectors. Extraction of ridge and blob features is achieved at faster than 15 frames per second on video sequences from a machine vision system, utilizing an NVIDIA GTX 480 graphics card. By design, the framework is easily extended to additional feature classes through the inclusion of feature metrics to be applied to the scale-space representation, and using common post-processing modules to reduce the required CPU workload. The framework is scalable across multiple and more capable GPUs, and enables previously intractable image processing at video frame rates using commodity computational hardware.

1. Introduction

In specific computer and machine vision applications, the size or scale of features to be detected is either unknown in advance or changes significantly during imaging. These changes can be caused by a varying distance between camera and target, or by blurring effects within poorly focused or motion blurred images. Multi-scale image processing techniques enable detection of these features, but the requirement to process data at many scale levels within a single image imposes a substantial computational load. In applications where image processing at video frame rates is required, such as in some machine vision metrology systems, processing using commodity hardware requires computational acceleration. Recent advances in General Purpose Graphics Processing Units (GPGPUs) have enabled new classes of multi-scale algorithms for video frame rate processing within these applications.

This paper describes a modular framework for GPGPU-accelerated multi-scale image processing, with specific application to scale-space feature detection. Recent results are presented from implementation of the framework on an NVIDIA CUDA-enabled GPU, with detectors implemented for two classes of scale-space features.
The remainder of this paper is organized as follows: Section 2 provides a brief introduction to scale-space techniques and the associated detection of Gaussian blob and ridge features; Section 3 describes the modular framework for scale-space feature detection on a GPU; Section 4 describes the GPGPU implementation; Section 5 presents results, and the paper concludes in Section 6.

2. Scale-space Image Processing

2.1. Generation of the scale-space

Scale-space theory provides a framework suitable for feature detection where little is known about the scale (size) of relevant features, and where the scale varies considerably within and between images in a data set. A parameter $t$ is introduced, known as the scale parameter, and is used as a filter variance parameter to progressively filter the original image at successive variance levels [1]. The kernel used to perform the filtering operation must meet specific requirements to generate a linear scale-space representation, and in the continuous case, the unique kernel with these properties is the Gaussian [2]. A basic scale-space is defined on an image $f$ as

$$L(x, y; t) = g(x, y; t) * f$$

where $g$ represents a two-dimensional Gaussian kernel with variance $t$. When discretized in practical systems, the Gaussian kernel does not necessarily produce the correct scale-space properties, so alternative discrete kernels have been proposed [3, 4], the latter of which is used for this work. More specifically, a separable convolution kernel with smoothing parameter step size of $\Delta t = \frac{1}{3}$, is [4]:

$$k = \left[ \begin{array}{ccc} 1 & 2 & 1 \\ 3 & 6 & 3 \\ 1 & 2 & 1 \end{array} \right]$$

Multiple discrete scale levels are chosen and the image is filtered at these chosen variances, collectively forming a scale-space representation of the image. Figure 1 depicts an original image frame, and progressive filtering at two variance levels. As the filter variance increases, finer features in the image are filtered away, providing a progressively smoothed and simplified representation.

![Figure 1. Progressive filtering of a sample image](image)

2.2. Feature detection

Once a scale-space representation of the original image has been computed, feature metrics and associated post-processing are applied. The details of processing depend upon the characteristics of the feature class, with two such classes described following. Selection and demonstration of blob and ridge feature detectors is motivated by a machine vision application where blob-like fiducials must be tracked across frames, and grid lines (ridges) must be extracted in the presence of substantial depth-of-field blur.
2.2.1. Blob Detection  
Blob features can be described through many metrics, but one measure useful for scale-space extraction is a scale-normalized Laplacian (scale-normalized discrete second derivative approximation) operator [5, 6]. The operator is applied to the scale-space image representation at all scale levels, and a spatially local maximum search is then performed (also across scale levels). A global sorting operation can then locate the strongest metric maxima, where the strongest responses are taken to be the best defined blobs. This method of blob detection allows features of varying sizes and spatial extents to be detected, intrinsically adjusting for the effects of changing camera range-to-target and blur.

2.2.2. Ridge Extraction  
Ridge extraction is more complex than blob detection in the sense that ridges are not restricted to local regions of an image, and must be extracted as continuous structures. The metrics defining a ridge are typically more arithmetically intense than primitive feature detectors (such as blob detection), and require the use of multiple metrics from which zero iso-surfaces are interpolated and intersected to define ridge segments [7]. A series of ridge extraction metrics have been proposed in the literature, one of which has been implemented in this work [7]. A primary metric is designed to detect ridge characteristics in the image, but in many cases the same physical ridge is detected at multiple scale levels in the image representation. A second metric is then applied to the scale-space that defines the optimal scale at which to detect the feature, typically based on some quality such as resolution or confidence in spatial localization. The two metrics form zero iso-surfaces, which are intersected to form the ridges. Ridge segments are interpolated within a voxel bounded by neighbouring pixels at neighbouring filter variance levels, and although continuous across multiple voxels, the ridge segments are computed independently of one another. A linking stage is then required to follow connected segments of a ridge, establishing the length and properties, and producing a connected output.

2.3. Other Features  
Many other feature metrics have been proposed in the literature including edge, junction, and corner detectors. Common post-processing tasks and re-use of the scale-space representation lead to the conclusion that a modular architecture is suitable, wherein detectors for additional feature classes can be added with minimal effort.

3. Modular Architecture  
In the context of image feature detection, scale-space techniques involve three primary phases: (a) generation of the scale-space image representation; (b) application of feature detection metrics; and (c) post-processing to extract features from the multi-scale metric data. Generation of the scale-space representation is common across various feature detectors, and as such can be viewed as the initial stage in a feature detection framework.

Detection of features involves application of a feature-specific metric (multiple metrics in some cases), followed by post-processing to reduce the data into a compact form usable by output applications with minimum additional computation. Post-processing tasks including non-maximum suppression, local extrema search, and value-based sorting are common to multiple classes of detectors, and are therefore amenable to reuse in a modular architecture. Figure 2 shows such an architecture, where all feature detectors reuse the output from a common scale-space generation module, and blob and junction detectors use common maxima search and sorting modules. The structure of the metric computation phase is common across all features, with the only difference being the metric equation implemented.

Passing of intermediate data between processing modules can be accomplished in two ways. First, intermediate data output by one module can be stored back to global memory, allowing a following module to explicitly load into shared memory the data that it requires. This method
causes the intermediate data to be buffered in the slower global memory, but does not impose any constraints on the data block size or structure to be used by subsequent modules.

A second method of intermediate data passing is to incorporate the modules together into a single GPU kernel, but separating the modules as code libraries. The isolated code modules are therefore combined at the compile phase, at the cost that the developer must pay more attention to the local data storage structures. This technique is less flexible, but can save data store/load overhead when a subsequent module requires an identical or similar structure of data in the multiprocessor’s shared memory.

4. GPGPU Implementation

4.1. GPGPU Overview
Recent advances in GPGPU technology by major manufacturers have extended the conventional GPU graphics pipeline into a more general purpose computing platform, highly efficient for many arithmetically intense applications. Algorithms must be parallelized for efficient GPU implementation, and further must be structured to execute within the computing resource constraints of a group of Single Instruction Multiple Data (SIMD)-like cores. Arithmetically intense algorithms typically experience the best acceleration on GPGPUs, especially where groups of threads exhibit non-divergent behaviour in the SIMD processing stream.

This work focuses on one NVIDIA GPGPU product line, utilizing the programming interface known as the Compute Unified Device Architecture (CUDA). A commodity GTX 480 graphics card is chosen as a realistic GPU, currently available in standard computing systems used for machine vision metrology.

4.2. Scale-space Algorithms on GPGPU
Scale-space algorithms exhibit properties making them suitable for GPGPU implementation, specifically spatial locality, arithmetic intensity, and operations that can employ parallel reduction techniques. Spatial locality of data exists during the scale-space generation and metric computation phases of feature detection, allowing independent portions of the data set to be assigned to independent GPU multiprocessors. A lack of required communication between the independent data provides for efficient scaling properties across larger or additional GPUs.

The scale-space generation and metric computation tasks exhibit arithmetic intensity, with many arithmetic operations taking place per data load operation from global memory. This characteristic allows memory fetches to be hidden behind computation operations, and leads to an implementation that is not tightly memory bound.

Post-processing tasks such as maxima search and data summary operations can be structured to conform to parallel programming patterns, including parallel reduction and parallel sorting.
4.3. Scale-space Generation
The kernel defined by (2) is used to generate the scale-space image representation, but it produces a filter variance step of $\Delta t = \frac{1}{3}$. Discrete scale levels must be chosen when generating the scale-space, and for this work, eight scale levels are defined at $t = 1, 4, 7, 10, 13, 16, 19, 22$ where $t$ is the filter variance parameter in pixels. Repeated application of (2) is required in each image dimension to achieve the necessary filter variance, so to achieve a flexible implementation, an intermediate “unit step kernel” $k_{\text{unit}}$ is generated which provides an incremental filter operation by a variance of $\Delta t = 1$.

$$k_{\text{unit}} = \begin{bmatrix} 1/216 & 3/54 & 51/216 & 11/27 & 51/216 & 3/54 & 1/216 \end{bmatrix}$$ (3)

The “unit step kernel” $k_{\text{unit}}$ is then repeatedly applied to the image data to achieve the required scale-space filter variances, with each set of repeated applications called a “step kernel” in Figure 3. The convolutions are separable, so separate operations are performed to filter the image data in the horizontal and vertical image axes.

![Figure 3. Scale-space generating convolution work flow](image)

Output from the convolution operations forms the image scale-space representation, which is subsequently used to generate the various feature detection metrics.

4.4. Computation of Feature Metrics
Specific metrics are computed for each type of feature to be extracted, as described in Section 2. The metric equations typically contain many spatial derivative approximations in the scale-space [7], and these are computed using central difference approximations as the core. To maintain uniformity across different orders of differentiation in a given image dimension, higher order derivative kernels are formed through repeated convolution of a first order central difference kernel. Using this approach, the spatial extent of data required to form a derivative approximation expands with the order of differentiation, so overlapping border data is loaded into the GPU multiprocessors to avoid fetching bordering data from global memory in a non-coalesced fashion.

Metric values are computed for each pixel of the image, at each scale level in the scale-space. Output from the metric computation calculations is therefore of the same dimension as the scale-space representation (for each metric computed), and this data forms the raw input to the feature extraction operations.

4.5. Blob Detection
The work flow for blob feature detection is presented in Figure 4. After application of the blob detection metric to the scale-space representation, an extrema search is performed within a
patch of data that is contained in a GPU multiprocessor. This search is structured as a parallel reduction operation using coalesced memory accesses for efficiency, and produces the maximum blob response within a given scale-space image patch. These local maxima can then be searched by the CPU to find a few maximum blob responses, or sorted on the GPU such that the CPU only needs to access the initial items in a sorted list to locate the strongest blob responses. When performed on the GPU, a parallel radix sort [8] is used to order the local maxima by response strength.

![Figure 4. Blob detection work flow](image)

### 4.6. Ridge Detection
Scale-space ridge detection on an NVIDIA GTX 280 has been described in a previous publication [9], so details of the ridge detection process such as iso-surface intersection and ridge summary data are not repeated here. The overall ridge detection process is shown in Figure 5. Arithmetically intense metrics are computed upon the scale-space image representation, and interpolated zero iso-surfaces are intersected to form ridge segments. The segments are then linked and examined within local blocks of the scale-space, and where valid portions of a ridge are found, summary data is produced that can later be used by the CPU to rapidly locate and analyse long ridges. The iso-surface intersection and summary data operations are independently performed on subsets of the scale-space data that is local to a specific GPU multiprocessor. These operations are also common to other scale-space algorithms such as edge detectors (given alternative edge detection metrics), so like blob detection post-processing, these modules can be reused and directly applied to alternative feature detectors.

![Figure 5. Ridge detection work flow](image)
5. Results

Sample output from the GPGPU-accelerated scale-space feature detectors is presented in Figures 6 and 7. Timing results are presented in Table 1 from implementation of the blob and ridge detectors on the most recent generation NVIDIA GTX 480 graphics card, running CUDA 3.0. The image resolution was 1024 by 768 pixels, and eight discrete scale-space scale levels were used, with scale variances of $t = 1, 4, 7, 10, 13, 16, 19, 22$. The total GPU processing time, reported in the final row of Table 1, includes the operations between and the overhead of kernel calls. This aggregate processing rate is better than 15 frames per second, which is the required frame rate for the machine vision camera used in testing.

Table 2 presents resource utilization for the various GPU kernels, and represents a significant increase in resource availability from previous generations of NVIDIA hardware. Registers are the floating point memory locations local in scope to a single thread, while shared memory is shared between groups of threads executing within a single multiprocessor (single thread block). Occupancy is the ratio of active threads to the maximum number of threads supported by a multiprocessor, as constrained by the hardware resources available and the individual utilization of each thread. For further description of the GPU architecture, the reader is referred to [10, 11].

Memory copies between the host system and GPU mass memory are not a bottleneck to performance. Data transfer times are variable, but small compared with the aggregate GPU processing time per frame. Current hardware further provides the capability to asynchronously transfer data between host and device, allowing data transfer operations to occur in parallel with kernel execution.

![Sample images with strongest scale-space blob metric response locations shown by red markers](image)

Figure 6. Sample images with strongest scale-space blob metric response locations shown by red markers

6. Conclusions

This paper has described a framework for GPGPU-accelerated scale-space feature detection intended to operate at video frame rates. The architecture is modular, allowing additional feature detectors to re-use both the image scale-space representation, and also common post-processing and analysis modules. By design the framework can be easily extended to new feature classes, and is scalable across multiple and more capable GPUs. Extraction of ridge
Figure 7. Scale-space ridge detector output line segments from a single sample image, with unfiltered noise detected between the straight grid lines.

Table 1. GPU Timing Results

| GPU Kernel                        | GPU Time/Frame [ms] |
|-----------------------------------|---------------------|
| Horizontal Convolution            | 0.6                 |
| Vertical Convolution              | 3.75                |
| Ridge Metric Computation          | 15.56               |
| Ridge Iso-surface Intersection    | 21.58               |
| Ridge Linking                     | 1.5                 |
| Blob Metric and Local Maximum     | 1.3                 |
| Blob Strength Radix Sort          | 3.6                 |
| **Total processing time (with overhead)** | **48.76**          |

and blob features is achieved at faster than 15 frames per second on video sequences from a real machine vision system, utilizing an NVIDIA GTX 480 graphics card for acceleration. This processing speed enables a set of algorithms previously intractable at video frame rates, providing simultaneous extraction of multiple feature classes from a non-trivial scale-space representation. The proposed framework can further be used to implement other multi-scale feature detection algorithms, and arithmetically intense image processing techniques in general.
Table 2. GPU Resource Utilization

| GPU Kernel                     | Registers | Shared Memory [bytes] | Occupancy [%] |
|--------------------------------|-----------|-----------------------|---------------|
| Horizontal Convolution         | 7         | 4224                  | 100           |
| Vertical Convolution           | 17        | 3208                  | 100           |
| Ridge Metric Computation       | 54        | 4928                  | 33            |
| Ridge Iso-surface Intersection | 25        | 8128                  | 58            |
| Ridge Linking                  | 15        | 2240                  | 88            |
| Blob Metric and Local Maximum  | 10        | 6976                  | 100           |

Acknowledgments
Funding from SHARCNET and NSERC is gratefully acknowledged.

References
[1] Witkin A P 1983 Proceedings of the 8th International Joint Conference on Artificial Intelligence pp 1019–1022
[2] Babaud J, Witkin A P, Baudin M and Duda R O 1986 IEEE Trans. Pattern Anal. Mach. Intell. 8 26–33 ISSN 0162-8828
[3] Lindeberg T 1993 Journal of Mathematical Imaging and Vision 3 349–376
[4] Lindeberg T 1993 International Journal of Computer Vision 30 79–116
[5] Lindeberg T 1996 Proceedings of Computer Vision and Pattern Recognition pp 465–470
[6] Lindeberg T 1998 International Journal of Computer Vision 11 283–318
[7] Lindeberg T 1996 Proceedings of Computer Vision and Pattern Recognition pp 465–470
[8] Lindeberg T 1998 International Journal of Computer Vision 30 79–116
[9] Lindeberg T 1996 Proceedings of Computer Vision and Pattern Recognition pp 465–470
[10] NVIDIA 2010 NVIDIA CUDA Programming Guide URL http://developer.nvidia.com/object/cuda.html
[11] NVIDIA 2010 Tuning CUDA Applications for Fermi URL http://developer.nvidia.com/object/cuda.html