SIFT implementation and optimization using OpenCL

1. Background and Motivation

OpenCL is relatively new standard from the Khronos group. The standard describes a platform for parallel heterogeneous computing. It essentially allows collaboration between different type of a processor such as CPUs, GPUs and DSPs. On modern personal computers, OpenCL allows any application to get access to the GPU to perform non graphics processing.

SIFT is the most popular algorithm in computer vision. It describes local features in images. These features can be used to match similar images in object recognitions.

The scope of this final project is mostly educational. The purpose is to learn about the new OpenCL platform while exploring a new adaptation of the SIFT algorithm to another kind of parallel programming architecture. In the process we can hope to achieve a speed up of the algorithm.

2. Overview of SIFT

The SIFT algorithm published by D.Lowe[1] takes an image as input and outputs a set of distinct local feature vectors. The algorithm is partitioned in four stages. Due to time constraints, orientation assignment which guarantees the rotational invariance of the feature descriptor generator which actually builds the feature vector have not been implemented and will not be discussed in this section.

- Scale Space Extrema Detection

This first step is where the SIFT keypoints are detected. This is accomplished by convolving the input image,$I(x,y)$ with Gaussian filters of varying widths,$G(x, y, k_i \sigma)$ and taking the difference of Gaussian-blurred images,$L(x, y, k_i \sigma)$. This creates a Difference of Gaussians “scale space” function defined as follows:

$$D(x, y, k_i) = (G(x, y, k_{i+1}) - G(x, y, k_i)) * I(x, y) = L(x, y, k_{i+1}) - L(x, y, k_{i+1})$$
Keypoints are the local maxima/minima of the DoG function. This is done by comparing each pixel to its 26 immediate neighbors (8 on the same scale and 9 corresponding neighbors on each of the 2 neighboring scales). A pixel is selected as a candidate pixel if it is strictly greater or strictly smaller than all its 26 neighbors.

- **Keypoint Refinement:**

  The scale-pace extrema detection generates too many keypoints that are potentially unstable. First, this stage refines the location of the keypoints by improving the localization to a sub-pixel accuracy using a Taylor series expansion of the DoG function. The new location of the keypoint is given as

  \[
  z = -\left( \frac{\partial^2 D}{\partial x^2} \right)^{-1} \frac{\partial D}{\partial x}
  \]

  Second, this stage rejects the keypoints that have low contrast (hence sensitive to noise). The value of keypoint at the refined location is given as

  \[
  D(z) = D + \frac{1}{2} \frac{\partial D^{-1}}{\partial x} z
  \]

  If \(|D(z)|\) is smaller than a certain threshold the keypoint is discarded. Finally, the keypoints that are poorly located on edges are excluded. In these cases the principal of curvature across the edge would be significantly larger than the curvature in the perpendicular direction. The curvature is retrieved from the eigenvalues of the second-order Hessian Matrix $H$. The ratio of principal curvature is directly related to the ratio of the trace and determinant. If the latter is above a certain threshold the keypoint is deemed poorly located and rejected.

  \[
  H = \begin{bmatrix}
  D_{xx} & D_{xy} \\
  D_{xy} & D_{yy}
  \end{bmatrix}
  \]
3. Overview of OpenCL

3.1 Hardware

OpenCL introduces a new hardware hierarchical organization. At the highest level of hierarchy seats the host machine. The host machine can have multiple devices supporting OpenCL. Each of the devices has multiple compute units. Finally, each of the compute units has processing elements.

3.2 NDRange, work-groups and work-items

Kernels are the basic functions that are executed by the openCL devices. Kernels can be written in high-level programming language similar to C. For the execution of a kernel, OpenCL introduces the concepts of NDRange, work-groups and work-items. NDRange specifies the dimensionality and size of the problem which also corresponds to the number of work-items. Work-items are lightweight equivalent to a software thread on the CPU. They are executed by the processing units. Work items are grouped into work-groups which share very fast local memory.
3.3 Memory architecture

The memory architecture defined by OpenCL also follows the hierarchies mentioned earlier. Each OpenCL device has global memory. This is a large amount of slow read/write memory located off-chip. Each compute unit has shared memory. This is a small amount of fast read/write memory located on-chip. Work-items in a workgroup share this memory. Each processing element has private memory consisting of registers.
4. Hardware resources

The Dell M1530 laptop is the hardware of choice for this final project. The laptop has an NVidia GPU that supports the most recent OpenCl driver. The hardware specification is listed below:

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core 2 Duo T8300</td>
<td>NVidia GeForce 8600M GT</td>
</tr>
<tr>
<td>2.4GHz</td>
<td>950 MHZ</td>
</tr>
<tr>
<td>4 GB RAM</td>
<td>512 MB</td>
</tr>
<tr>
<td></td>
<td>32 processing elements</td>
</tr>
</tbody>
</table>

5. Implementation

5.1 OpenCL-SIFT Framework

Writing an OpenCL application involves a considerable amount of set up code to prepare the device for computation, to compile and launch kernels, to transfer data to and from the device. This is a repetitive task reduce the programmers efficiency and expose him to errors. A framework that abstracts out the counter-productive OpenCL setup code developed to tackle this problem. The framework provides singleton classes to setup the device, to launch kernels and handle host-device data transfers. For instance one could a 2D separable convolution on the GPU using just a few lines.

```cpp
oclSingleton<oclConvolution2D>::getSingleton()->initialize();
oclImage imIn (imageW, imageH, CL_MEM_READ_ONLY, "./graftiti.csv");
oclImage imOut (imageW, imageH, CL_MEM_WRITE_ONLY);
oclArray arrK = sift.buildKernel(4.1);
oclSingleton<oclConvolution2D>::getSingleton()->runProgram(imOut, imIn, arrk);
imOut.save("./filtered.csv");
```

5.2 Gaussian Filter
The Gaussian filter is separable, meaning that the 2D dimensional filtering can be broken up into 2 1D sequential filtering. The `convolutionSeperable()` method developed by NVIDIA for efficient separable convolution on the GPU is used.

### 5.3 Extrema Detection

The `siftExtremaDetection` singleton class encapsulates the OpenCL kernel used to determine which pixel location correspond to a minimum or maximum in the DOG scale-space. The kernel takes as input 3 successive images in the DoG Pyramid and outputs a 2D map with the values +1 to indicate a maximum, -1 to indicate a minimum and 0 to indicate absence of a keypoint. `siftExtremaDetection` launches a work-item for each pixels of the middle input image. The work-item compares the pixel value at the stored location to its 26 neighbors.

### 5.4 Keypoint Refinement

The 2D map of extrema location is read back to the CPU where it is collapsed into an data structure storing the (x,y) location and scale of the extrema. The map is then passed to a keypoint refinement kernel which is essentially a slightly modified version of the equivalent Matlab C module by Andrea Vedaldi. The `siftKeypointRefine` singleton class launches one work-item for each element in the collapsed data structure mentioned above. Each work-item determines the sub-pixel location of the keypoint and returns the refined location.

### 6. Results

#### 6.1 Time

All the measurements are performed on an input image of size 512x512.

Gaussian Filtering (sigma 4.1):

- Vedaldi Matlab CPU = 0.19s – 100%
- Naïve C++ CPU = 0.33s – 57%
- GPU = 0.0094s – 2000%
- GPU with data transfer = 0.0133s – 1400%
Extrema Detection (octave 0 of pyramid):

- Vedaldi Matlab CPU = 0.179 s – 100 %
- GPU = 0.035725s – 500 %

Keypoint Refinement (octave 0 of pyramid):

- Vedaldi Matlab CPU = 0.004 s – 100 %
- GPU = 0.0689s – 6 %

The keypoint refinement does not perform as well on the GPU because the small size of the input. In the test cases that generate the data above less than approximately 1100 keypoints are refined. To amortize the upfront cost of transferring data to the GPU and setting up the execution environment the size of the problem needs to be considerably larger.

### 6.2 Performance

The image below shows the location of the keypoints detected in the first octave of the DoG function. The red crosses represent the keypoints that are unique to OpenCL implementation; the blue ones are unique to the Matlab Implementation while the green ones are common to both. 85 % of the keypoints locations match.
As you may notice, the OpenCL implementation detects a significant number superfluous keypoint along the borders. This is due to the fact that the current implementation of `convolutionSeperable()` does not pad the image for filter continuity. The difference in the border behavior between the smoothing functions used by the different implementation is very apparent in the image below which shows the difference between filtered images generated by the two methods.
If we disregard all the keypoints detected within a border region of 33 pixels i.e. longest kernel filter applied during the pyramid building the matching goes up to 94%
7. References

