Curvature Estimation Algorithm's Implementation

NGUYEN Quang Anh, REBMANN Guillaume

Monday 27th April, 2015

Abstract

Several curvature estimators along digital contours were proposed. In this paper, we show the implementation of *Osculating Circles Estimator* and *Binomial Convolution Curvature Estimator*. After that is an approach to parallel the calculations on GPU.

1 INTRODUCTION

The main target of this article is to implement different algorithms to calculate Curvature. To reach our goal, we studied differents scientific articles [1] and then implement the algorithm for an parallel use.
This article is composed of two parts: Osculating Circles Estimator & Binomial Convolution Curvature Estimator.
The purpose of the comparison is to find the best algorithm to use to find curvature to classify different insect species.

2 DGTal Library

In this project, we used DGtal for image and contour's realization. DGtal library is a project aimed at developing generic, efficient and reliable digital geometry data structures, algorithms and tools. This project is made by the cooperation between LIRIS, LAMA, LORIA, GREYC and IRCCyN.

3 Osculating Circles Estimator

Osculating Circles Estimator (CC) is an algorithm that calculate the curvature of a contour, by pointing out the tangent circle at each point of the contour.

For implementing this part, we used the definition of digital straight segment and digital straight line.

3.1 Digital Straight Line

Digital Straight Line (DSL) is defined by 4 values : $D(a, b, \mu, \omega)$, with $a, b, c, d \in \mathbb{Z}$ and $gcd(a, b) = 1$. a/b is called a sloped of D, μ is an intercept and ω is the thickness of D.

Every points that belong to $D(a, b, \mu, \omega)$ must satisfy :

$$\mu \leq ax - by < \mu + \omega$$

In DGtal, there are 2 types of DSL :

- Naive Digital Straight Line
- Standard Digital Straight Line

These type of DSL is made by specifying the value *thickness*.

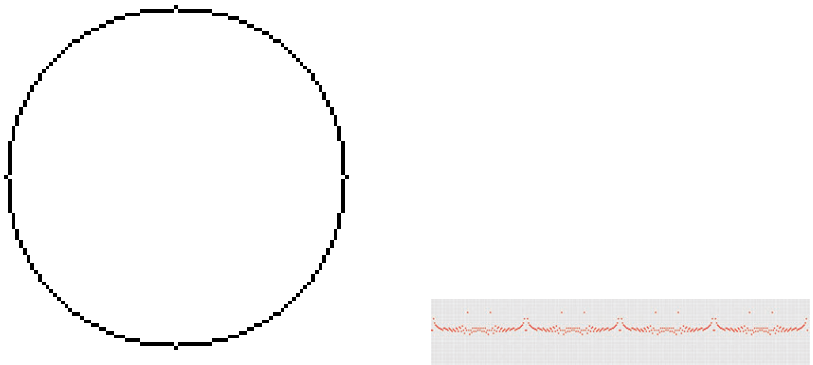
- Naive DSL : $\omega = max(|a|, |b|)$
- Standard DSL : $\omega = |a| + |b|$

3.2 Digital Straight Segment

Digital Straight Segment (DSS) is the set of points that belong to a digital straight line. Just like DSL, in DGtal there are also two specifics DSS : Naive DSS and Standard DSS.

3.3 Implementation

In this project, we used the standard DSL and DSS to implement this algorithm. A class **ArithmeticalDSSComputer** will be used to analyze the points of the contour.



(a) Source File

(b) Result after applying the algorithm

Figure 1: Source and Result

StandardDSS4Computer is the sub class of **ArithmeticalDSSComputer**, which is specialized which type of DSS that use to make the calculation. After extract all the points of the contour into a vector, we need to adjust this vector to be able to iterate through all the points of the contour.

The idea of CC algorithm is that at each point of the contour, called K point, we will try to extend the DSS, which is started at the point we are at, to the utmost left (resp. utmost right). The last point that can be added to the DSS called L point (resp. R point). The three points K, L, R form a triangle. Call the rayon of the circle that go through these three points R. Then the curvature's value of point K will be :

$$C(K) = \frac{sign(det(\overrightarrow{KR}, \overrightarrow{KL}))}{R}$$

. In case that these three points is alignment, the curvature's value is set to zero.

So the most important work in this algorithm, is to be able to locate the two points L and R with each point K that we iterate through. We need two DSS, one for searching the point R, another for searching the point L. In class **StandardDSS4Computer** there are two methods that could help with the searching : *extendFront* and *extendBack*. These two method will return true if they can still add point to the DSS, and false if not. So now the problem of searching for the two points L and R is solved. But we still have one more problem.

For calculating the curvature of point K, we have to have the two points L and R corresponded to the point K. So now what will happen if we are at the begin or the end of our vector ? Only one point, either L or R, could be found. The vector is the set of all the points of the contour, that means that the begining and the end of this vector, is at the same point of the contour. So the set of points obtained by *StandardDSS4Computer::extendFront* at the begin of the vector, and the set of points obtained by *StandardDSS4Computer::extendBack* at the end of the vector will be added to end and the head of the vector. Now, we can easily iterate through the set of points of the contour.

3.4 Usage

For testing this algorithm, go to the folder **TestDSL**, execute **cmake**, then **make**. Then run :
./TestDSL path-to-file
path-to-file is a path to an image file, with **pgm** extention (Portable Graymap)

4 Binomial Convolution Curvature Estimator

Binomial Convolution Curvature Estimator (BCC) is an algorithm that calculate the curvature of a contour, by using a discrete convolution product.
The algorithm of Malgouyres et al. use derivative Kernel & smoothing Kernel [2]

4.1 Smoothing Kernel

The author give a smoothing kernel defined like that:

$$h_n(x) = \begin{cases} n & \\ a + \frac{n}{2} & \text{if } n \text{ is even} \\ n & \\ a + \frac{n+1}{2} & \text{if } n \text{ is odd} \\ 0 & otherwise \end{cases}$$

4.2 Derivative Kernel

The derivative kernel is defined by the function:

$$B_n(a) = \delta * H_n(a)$$

where δ is defined by the function:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ -1 & \text{if } x = 1 \\ 0 & otherwise \end{cases}$$

4.3 Calcul of the curvature

To calculate the curvature, we compute

$$Curvature = \frac{D_n^2(x) * D_n(y) - D_n^2(y) * D_n(x)}{D_n^2(x) + D_n^2(y)}$$

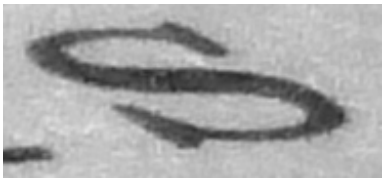
Where

$$n = h^{2(\alpha-3)/3}$$

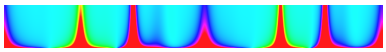
h Represent the grid size. When we compute the algorithm, we increase the grid size on every step.

4.4 Implementation

To implement this algorithm, we used the libraryr DGtal to get access to differents objects and functions. We tried to implement the same algorithm to be computed on the GPU (Parallel) but we had some difficulties to work with the source code of the DGtal library that is fully written in a generic style. We tried to use CUDA and openCL but we didn't get the algorithm working in parallel. In fact we could put some calculs in parallel, but it's not really usefull to put only one addition or multiplication in parallel. That's why we focused on the full algorithm in parallel.



(a) Source File



(b) Result after applying the algorithm

Figure 2: Source and Result

4.5 Usage

For testing this algorithm, go to the folder **TestBCC**, execute **cmake**, then **make**. Then run :
./main -f ../images/contourS.fc –gridStepInit 0.001 –gridStepIncrement 0.0005 –gridStepFinal 0.05 -o result.ppm

5 Parallel Programing

We have tried to use OpenCL for paralleling algorithm on GPU. Although it's still not perfect, but some archievements have been made. The target of parelling is CC algorithm.

There are some main points about OpenCL that we need to point out. First is a **kernel**. **Kernel** is a function that will be run on the GPU, by many threads at the same time. Each thread on the GPU will handle a part of data which is sent to GPU by the host. Second, the kernel compilation happens at run time, means that after the **make** command for compiling the program, **kernel** still wasn't compiled yet. At the time the command to run the program is executed, that is when the **kernel** would be compiled. So it's a convenient way to re-write the **kernel** without re-compiling the whole program each time.

Because OpenCL don't support class call in the **kernel**, so the part that searching for the points L and R can't be paralleled. So this is the way we parallel algorithm on GPU. First, the program will run normally, find all the points L and R corresponding to the point K, and stock all of these datas in a vector. These vector will be passed to the **kernel** to perform paralleling the calculation of curvature on GPU.

This processing was successful for the small size of vector. When the vector's size grows above 91, the problem appears. It seemed that on our machine, only 91 threads could be run on GPU. This is the problem we still didn't figure it out yet, whether the problem lies in the GPU's ability, or in the program itself.

6 Difficulties

Thorough this project, we have encountered many difficulties, but in contrast, we have obtained many knowledges. First of all is the compilation process of this project. We have to use **CMAKE** system, which is new to us.

Secondly, also in the compilation project, is linking the library and the project by **CMAKE**. There are two kinds of library that we could classify : one that supports **CMAKE** and the other don't. But then, once we found the solution for this problem, we could see that **CMAKE** is a really powerful tool.

Thirdly, the **DGtal** library. This is the first time that we use a big project as **DGtal** as a tool in our project. And what trouble us the most, is the enormous usage of template and generic items. But then, this could be a reference for us, to work in a big project in the future.

Finally, the paralleling algorithm. There were a long time that we have to decide which should be used : **CUDA** or **OpenCL**. **CUDA** is in fact, easier to write than **OpenCL**, as **NVIDIA** provides a lot of template and library, that make the programing easier a lot more than **OpenCL**. But because of the installation on one of our machines is unsuccessful, we chose **OpenCL**. And furthermore, **OpenCL** is available on MAC OSX, and it isn't a **CMAKE** supported library.

7 CONCLUSION

We implemented the 2 differents algorithms and tried without succes to implement them in parallel. We also got some diffculties to use the algorithm on real picture because they are only working well on *fake* contour (created in the code). We implemented some calculs in parallel but they weren't efficient to improve the speed of the computing (Only on simple calcul but not in a global way). It was maybe an error from us to use the DGtal library to try to implement them in openCL and CUDA.

References

[1] Bertrand Kerautret, J-O Lachaud, and Benoît Naegel. Comparison of discrete curvature estimators and application to corner detection. In *Advances in Visual Computing*, pages 710–719. Springer, 2008.

[2] Rémy Malgouyres, Florent Brunet, and Sébastien Fourey. Binomial convolutions and derivatives estimation from noisy discretizations. In *Discrete Geometry for Computer Imagery*, pages 370–379. Springer, 2008.