# Parallelization of the Canny Edge Detection Algorithm using CUDA

**Team Members:**

Alex Varvel: varvel@pdx.edu

Daniel Collins: daniel.e.collins1@gmail.com

Anil Reddy-Peddolla: anil2@pdx.edu

**Topic Description:**

Edge detection is a mathematical method of determining the "edges" within an image. An "edge" is defined as a point which the brightness of the image has discontinuities. Sharp changes in brightness allow the edge detection algorithm to map an outline of the image being observed, and these edges can correspond to changes in depth, orientation, materials, or lighting. Mapping the edges of an image is a non-trivial task, as many false edges can be mapped, or segments of a real edge can be left off if the change is not drastic enough for the algorithm to pick up.

The algorithm we chose to implement is the Canny Edge Detection algorithm. This algorithm is a first-order image processing algorithm that was developed by John F. Canny in 1986 and follows a 5-step detection algorithm: Apply a Gaussian filter, find the intensity gradient, apply non-maximum suppression, apply a double threshold, track the edges using hysteresis. The size of the Gaussian filter and the double threshold can be manipulated in order to affect both the computation time and the effectiveness of the algorithm. Based on the image being processed, different combinations of the two may yield more accurate results.

We plan to implement this algorithm using the parallel computing platform CUDA. We chose CUDA as our platform because it has been designed/optimized for graphics and image processing falls into that category. Additionally, this is a very common platform in industry, so we feel having experience in it will benefit us greatly.

**Importance of Topic:**

This topic is of particular importance for machine/computer vision and, in general, image processing. In image processing, an edge is of particular importance because it allows distinct characteristics of 3-D space to be mapped. Mapping edges allows the device processing the image to significantly reduce the amount of data being processed/stored which becomes particularly important for computer vision.

The Canny Edge Detection algorithm is an important algorithm within the edge detection field because it drastically reduces the amount of data that needs to be processed

when compared to other first-order image processing algorithms. This algorithm can also be parallelized using CUDA, which falls in line with our project interests.

**Method of Evaluation**

Validation of an edge detector is an admittedly difficult problem in academia due to the subjectivity of the results. It requires establishment of what is called the *ground truth*, or a baseline of results. Our validation methodology will rely running two classes of images through our implementation as well as through a high-quality reference implementation (TBD). The classes of images will be 1) synthetic hand-crafted images with varying types of edges and 2) a corpus of stock images commonly used in literature. The output of each edge detector will be evaluated both by statistical analysis of the binaries as well as qualitative human analysis.

**Project Status Report (Feb. 27)**

Since the original progress proposal, we have made significant progress on our edge detection algorithm. There were a number of factors we had to consider when implementing the project and they are detailed in chronological order below:

1) **Determined a uniform environment:**

CUDA can only be run on nVidia GPUs, so naturally this makes finding a uniform environment inherently difficult. Since each team member has a difference version of nVidia graphics cards, we wanted to make sure that our code could be run the same way by each of us, and more importantly, by anyone so that it would be easy to grade, evaluate speedup, and update. In order to do so, we decided to use the FAB computers because they have nVidia GPUs accessible to them. By using a unified resource that everyone has access to, we were able to allow for everyone to edit the project regardless of what platform they were on.

2) **Ability to read and write images in any popular encoding (.jpg/.bmp/.png, etc.):**

The first step to an image processing algorithm is to have an image to process. We chose to use the magick++ image processing library. This library is portable and flexible across many systems, so like in step 1, we wanted to have code that is accessible and easy to maintain. Magick++ is used to generate a flat buffer of pixel structures, each of which has a red, green, blue, and alpha field. These pixel structures are then modified by the different filters, and a final image is generated (which detects the edges).

3) **Infrastructure has been built for consistent environment:**

By using CUDA, C++, and the Magick++ library, we were able to create an environment that is consistent across users. This allows us to debug and share code easily. Github is currently being used for version control and to facilitate the sharing of code amongst group members as well.

4) **Initial serial code for algorithm (still needs testing though):**

In order to quantify the speedup of our image processing algorithm, we wanted to write a serial implementation of our code. This implementation is run on a single core, with a single thread. In theory, this is the slowest way this code can be executed so this will provide a baseline time measurement for our CUDA based implantation. We have implemented this step by writing the five filters that make up a Canny Edge Detection algorithm to execute on a single CPU. This allowed us to debug our algorithms thoroughly on a platform we were familiar with, which hopefully, will allow us to implement the CUDA implementation more easily.

5) **CUDA tutorials and integration of CUDA modules into our build flow:**

In conjunction with implanting our algorithms for a single CPU, we have been watching/reading tutorials on CUDA to help us understand how we are going to implement our functions using CUDA. We have CUDA code that is executing on our test machines as of now, however, we have yet to implement our algorithms using CUDA.

**Work to be done:**

Although there are only about two weeks until the project is due, we are confident we can complete the tasks discussed in the proposal. We plan to have the CUDA implementation done by the end of this week, and then the final week of the project will be devoted to testing and creation of the final presentation. Once we give the final presentation and take the final exam, the last week will be devoted to writing the final project report.

**References:**

The original Canny edge detection paper:

Canny, John. "A computational approach to edge detection." *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986): 679-698.

CUDA implementations of Canny edge detection:

Ogawa, Kohei, Yasuaki Ito, and Koji Nakano. "Efficient Canny edge detection using a GPU." *Networking and Computing (ICNC), 2010 First International Conference on*. IEEE, 2010.

Luo, Yuancheng, and Ramani Duraiswami. "Canny edge detection on NVIDIA CUDA." *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 2008.

L. H. A. Lourenço, D. Weingaertner and E. Todt, "Efficient Implementation of Canny Edge Detection Filter for ITK Using CUDA," *2012 13th Symposium on Computer Systems*, Petropolis, 2012, pp. 33-40.
doi: 10.1109/WSCAD-SSC.2012.21