Accelerating Robust Normal Estimation
A comparison of CPU vs. GPU performance

Rohit Gupta & Ian Endres
HERE Technologies
rohit.gupta@here.com — +31 (60) 74 13 688

Abstract
We present an implementation for calculating surface normals on the CPU and GPU. The GPU implementation can be up to an order of magnitude faster than the CPU implementation for the setup on a Desktop workstation with a 6-core (12 thread) Intel Xeon processor and a Quadro M6000 from NVIDIA with 8GB memory on board.

Introduction
In order to understand the contents of a scene from point clouds (Figure 1), it is often required to calculate normals of the surface the point cloud represents. Furthermore, this calculation is highly time consuming and recurring so optimizing it can have a significant impact on the total runtime.

Algorithm
We use the RANSAC approach for calculating surface normals on our point clouds. In Algorithm 1 we list the computational steps in our implementation.

Algorithm 1 Robust L1 normals estimation

Inputs
1. Point cloud: \( X = x_j, \) \( N \) number of points
2. No. of neighbors: \( K_{\text{neigh}} \)
3. No. of normal hypotheses/point: \( K_{\text{samp}} \)

Outputs
1. Surface normals/point: \( y_j \)
2. Goodness of fit score/point: \( s_j \)

Division into Kernels
The code was broken down into kernels. Brief description of the kernels used follows:
- FLANN NN search – This kernel is a call to the nearest neighbor search as implemented in FLANN. Line 2 in Algorithm 1.
- Populate_y – This kernel generates a matrix whose rows contain relative vectors of the current query point to all its neighbors. So, the matrix size is \( N \times K_{\text{neigh}} \). It also normalizes each of these vectors. Lines 3 through 6 in Algorithm 1.
- Calculate_normals – This kernel calculates the cross product of two randomly selected vectors from the \( y \) matrix for all points. The number of generated cross products is \( K_{\text{samp}} \). They are also normalized. Line 9 in Algorithm 1.
- Dot_products – This kernel calculates dot products of each query point with each of its candidate cross products to generate \( K_{\text{samp}} \) scores per point in the point cloud. Line 10 in Algorithm 1.
- Best_normal_search – The normal vector (cross product from calculate normals kernel) with the minimum of these scores is chosen as the normal for a query point in this kernel. Lines 11 through 13 in Algorithm 1.

Results
In the adjoining plots we show how the execution times vary for both robust normal calculation and the preceding calculation of nearest neighbors on the GPU. In Figure 4 we show the time taken by the FLANN implementation to calculate nearest neighbors. This is common to both CPU and GPU implementations. In Figures 5 and 6 we have the plots for the execution times for CPU and GPU implementations of normal estimation.