# ECE1779

# final\_report.txt

# Vipin Bakshi and Andre Lo

# 20170412

**INTRODUCTION**

Feature and Object detection in videos and images is an extremely common and important task in today's computational world. From applications in biometrics, Deep Neural Networks for Image/Video processing, surveillance to Real-Time Systems for Autonomous Car behaviors, various algorithms have been developed to efficiently analyze and detect objects in video / image streams. One of the oldest and most commonly used Algorithms is the Hough Transform which was patented by Peter Hough in 1962 and formalized by Richard Duda and Peter Hart in 1972.

The Hough Transform is an edge detection algorithm that can be extended to then detect shapes such as lines and circles in objects. The Hough Transform is able to behave on a pixel by pixed basis for its computation and uses a bin based voting system to determine pixels that fall into a particular category. This algorithm is quite computationally expensive with a complexity of O(A ^ (x-2)) where A is the size of the image space and (x) is the number of parameters required to represent 2D or 3D shapes. Therefore, the algoritm is ideally suited to detect 2D lines and cirles rather than 3D cylinders or spheres. However, it is possible to use the algorithm for all sorts of complex object detections.

This algorithm is of particular interest to us as autonomous vehicles use computationally intensive vision analysis to detect objects such as STOP signs, lane lines and pedestrians. These computations need to happen in a real-time environment with minimal latency due to computation. Therefore, for our project, we will analyze the computational benefits and speedups possible due to the usage of Hough Transform to perform computation on a high-resolution HD image. The following applications and experiments were performed:

1. Single Threaded CPU computation

2. Multi-Threaded CPU computation

3. 1D-Kernel GPU computation with different grid and block sizes.

4. Determination of optimal kernel arrangement for most optimal computaitons

4. Optimizations to position computation heavy cos, sin, divide operations outside of the main thread blocks

5. Optimizations to use floats instead of doubles to reduce accuracy for performance

6. intrinsic division and sinch functions used to improve performance

**HARDWARE ARCHITECTURE**

<<<< TODO FILL GPU AND CPU HW DETAILS >>

**SOFTWARE ARCHITECTURE**

A popular open-source implementation of the Hough's Transform was used as a starting benchmark for the analysis. The application is organized as follows:

>> cairo : Contains graphing libraries and binaries to load data from images.

>> hough\_cpu: Hough algorithm and applications.

>> time : Time module to time algorithm performance.

>> readme.txt: Application build and usage documentation.

>> Compilation: cuda-8.0 NVCC compiler is used for the computation of the CPU and GPU based CUDA applications.

>> Computation performed on car.png which is a 3700 \* 1900 image as an 8-bit/color RGBA

**SINGLE THREADED APPLICATION**

The following steps are taken to perform the computation in a single-threaded sequential manner:

1. User CAIRO to load .png image and extract data to uint8\_t buffer.

2. Determine width, height, stride and bits per pixel of image.

3. Invoke HoughTransform on data. Basic Algorithm:

a. Convert planar to polar coordinates with rho and theta representation. rho =

xcos(theta) + ysin(theta)

b. For rho between 0 to max(rho), perform below computation.

c. For theta between 0 to 360 degrees, perform the following computation.

d. For y between 0 to height of image, find differential dx and dy.

Use dx and dy to read corressponding R, G, B values of input data.

e. Save into hough transform array, the corresponding R,G,B mixtureas a bin.

f.Save new height, width and stride of Hough Transform. Return Hough Transform array.

4. Use CAIRO to write hough array as a new .png image.

**MULTI-THREADED APPLICATION**

Before the GPU is used, a CPU Multi-Threaded application will be used to analyze the speedups that can be used as well. This application was multi-threaded using the pthreads library in a coarse grained manner by using the rho value as a work distributor:

1. User CAIRO to load .png image and extract data to uint8\_t buffer.

2. Determine width, height, stride and bits per pixel of image.

3. Invoke HoughTransform on data. Basic Algorithm:

a. Convert planar to polar coordinates with rho and theta representation. rho =

xcos(theta) + ysin(theta)

b. Dispatch 8 threads to perform c: Each thread works to compute a subsection of

the rho computation loop, with the last thread computing the residue as well,

if any.

c. For rho between 0 to max(rho), perform below computation.

d. For theta between 0 to 360 degrees, perform the following computation.

e. For y between 0 to height of image, find differential dx and dy.

Use dx and dy to read corressponding R, G, B values of input data.

f. Save into hough transform array, the corresponding R,G,B mixtureas a bin.

g. Barrier to Join threads.

h.Save new height, width and stride of Hough Transform. Return Hough Transform array.

4. Use CAIRO to write hough array as a new .png image.

**GPU APPLICATION 1**

1. User CAIRO to load .png image and extract data to uint8\_t buffer.

2. Determine width, height, stride and bits per pixel of image.

3. Invoke Hough Transform on data:

a. Convert planar to polar coordinates with rho and theta representation. rho = xcos(theta) + ysin(theta)

b. Allocate resources for GPU and CPU. Copy input data from CPU to GPU.

c. For rho between 0 to max(rho), perform below computation.

d. Dispatch (360 / grid) GPU threads For theta between 0 to 360 degrees,

perform the following computation.

e. For y between 0 to height of image, find differential dx and dy.

Use dx and dy to read corressponding R, G, B values of input data.

f. Save into hough transform array, the corresponding R,G,B mixtureas a bin.

g. Copy resources from GPU to CPU with resultant array.

4. Use CAIRO to write hough array as a new .png image.

**GPU APPLICATION 2**

Same as GPU Application 1 with the following changes:

2 warps:

WARP1: theta << 45, theta between 135 and 225, theta > 315

WARP2: theta between 45 and 135, theta between 225 and 315

**GPU APPLICATION 3**

Same as GPU Application 1 with the following changes:

Factoring of sin() / cos() outside of loop

**GPU APPLICATION 4**

Same as GPU Application 1 with the following changes:

Factoring of divisions outside of the loop.

**GPU APPLICATION 5**

Same as GPU APPLICATION 3 with the following changes:

Factoring of sin() and cos() with sincos()

**GPU APPLICATION 6**

Same as GPU APPLICATION 1 with varying sizes of grid and block granularities:

|  |
| --- |
| CUDA v6: <<<1,360>>> |
| CUDA v6: <<<2,180>>> |
| CUDA v6: <<<3,120>>> |
| CUDA v6: <<<4,90>>> |
| CUDA v6: <<<5,72>>> |
| CUDA v6: <<<6,60>>> |
| CUDA v6: <<<8,45>>> |
| CUDA v6: <<<9,40>>> |
| CUDA v6: <<<10,36>>> |
| CUDA v6: <<<12,30>>> |
| CUDA v6: <<<15,24>>> |
| CUDA v6: <<<18,20>>> |
| CUDA v6: <<<20,18>>> |
| CUDA v6: <<<24,15>>> |
| CUDA v6: <<<30,12>>> |
| CUDA v6: <<<36,10>>> |
| CUDA v6: <<<40,9>>> |
| CUDA v6: <<<45,8>>> |
| CUDA v6: <<<60,6>>> |
| CUDA v6: <<<72,5>>> |
| CUDA v6: <<<90,4>>> |
| CUDA v6: <<<120,3>>> |
| CUDA v6: <<<180,2>>> |
| CUDA v6: <<<360,1>>> |

**GPU APPLICATION 7**

Same as GPU APPLICATION 1 with the following changes:

Lower precision float usage instead of double usage.

**GPU APPLICATION 8**

Same as GPU APPLICATION 1 with the following changes:

sin() and cos() replaced with single precision floating point sinf() and cosf()

**GPU APPLICATION 9**

Replaced division with intrinsic division \_\_ddiv\_rz() in kernel

**GPU APPLICATION 10**

Mixture of CUDA APPLICATION 6 and CUDA APPLICATION 3 which serves as best performance on 1D arrangement.

**SPEEDUPS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **code** | **run 1** | **run 2** | **run 3** | **avg of 3 runs** | **speedup vs CPU st** |
| CPU st | 41325072749 | 41325072749 | 41325072749 | 41325072749 |  |
| CPU mt | 10892418676 | 10892418676 | 10892418676 | 10892418676 | 3.793929886 |
| CUDA v1 | 9258325200 | 9257530460 | 9261512272 | 9259122644 | 4.463173709 |
| CUDA v2 | 18674428165 | 18681797219 | 18685667301 | 18680630895 | 2.212188281 |
| CUDA v3 | 9046384584 | 9046635923 | 9047386354 | 9046802287 | 4.567920403 |
| CUDA v4 | 9260967252 | 9272570653 | 9256013197 | 9263183701 | 4.461217016 |
| CUDA v5 | 9398973257 | 9402151610 | 9400024354 | 9400383074 | 4.396105183 |
| CUDA v6: <<<1,360>>> | 11503817616 | 11653368154 | 11507654126 | 11554946632 | 3.576396678 |
| CUDA v6: <<<2,180>>> | 10281457985 | 10282890531 | 10279062070 | 10281136862 | 4.019504195 |
| CUDA v6: <<<3,120>>> | 9520027372 | 9523913143 | 9523015539 | 9522318685 | 4.339811985 |
| CUDA v6: <<<4,90>>> | 9524113813 | 9521658232 | 9526836179 | 9524202741 | 4.338953493 |
| CUDA v6: <<<5,72>>> | 9519561995 | 9522202685 | 9529804974 | 9523856551 | 4.339111213 |
| CUDA v6: <<<6,60>>> | 9430581465 | 9434379868 | 9432527437 | 9432496257 | 4.381138526 |
| CUDA v6: <<<8,45>>> | 9128860584 | 9125751755 | 9125951363 | 9126854567 | 4.527854853 |
| CUDA v6: <<<9,40>>> | 9479761872 | 9482997078 | 9497488999 | 9486749316 | 4.356083562 |
| CUDA v6: <<<10,36>>> | 9484706868 | 9469243428 | 9463040744 | 9472330347 | 4.362714479 |
| CUDA v6: <<<12,30>>> | 9109598818 | 9257530460 | 9261512272 | 9209547183 | 4.4871992 |
| CUDA v6: <<<15,24>>> | 9106076710 | 9110995483 | 9101838090 | 9106303428 | 4.538073333 |
| CUDA v6: <<<18,20>>> | 9002682626 | 9003482710 | 9002652482 | 9002939273 | 4.590175663 |
| CUDA v6: <<<20,18>>> | 9021131266 | 9020119769 | 9020580825 | 9020610620 | 4.581183524 |
| CUDA v6: <<<24,15>>> | 8798276243 | 8802045263 | 8801546966 | 8800622824 | 4.695698654 |
| CUDA v6: <<<30,12>>> | 8816894148 | 8823746028 | 8821459408 | 8820699861 | 4.685010645 |
| CUDA v6: <<<36,10>>> | 8765548952 | 8768236650 | 8768318088 | 8767367897 | 4.7135096 |
| CUDA v6: <<<40,9>>> | 8562477629 | 8564735688 | 8565072992 | 8564095436 | 4.825386762 |
| CUDA v6: <<<45,8>>> | 8658712783 | 8655270489 | 8659820294 | 8657934522 | 4.773086773 |
| CUDA v6: <<<60,6>>> | 8568831434 | 8565424499 | 8567000246 | 8567085393 | 4.823702678 |
| CUDA v6: <<<72,5>>> | 8468714558 | 8468703304 | 8472532441 | 8469983434 | 4.879002783 |
| CUDA v6: <<<90,4>>> | 8752876117 | 8754625455 | 8756988404 | 8754829992 | 4.720259878 |
| CUDA v6: <<<120,3>>> | 8673861845 | 8678141816 | 8676731945 | 8676245202 | 4.763013468 |
| CUDA v6: <<<180,2>>> | 9877076141 | 9879835315 | 9879160422 | 9878690626 | 4.183254068 |
| CUDA v6: <<<360,1>>> | 11998202232 | 11997469720 | 11998056773 | 11997909575 | 3.444356076 |
| CUDA v7 | 9374023165 | 9380535660 | 9371356452 | 9375305092 | 4.407864314 |
| CUDA v8 | 9365491212 | 9369583555 | 9381068778 | 9372047848 | 4.409396262 |
| CUDA v9 | 9460930793 | 9444447390 | 9443251152 | 9449543112 | 4.373235008 |
| CUDA v11 | 8276090646 | 8272686646 | 8271431494 | 8273402929 | 4.994930515 |