Sobel Edge Detection

Overview

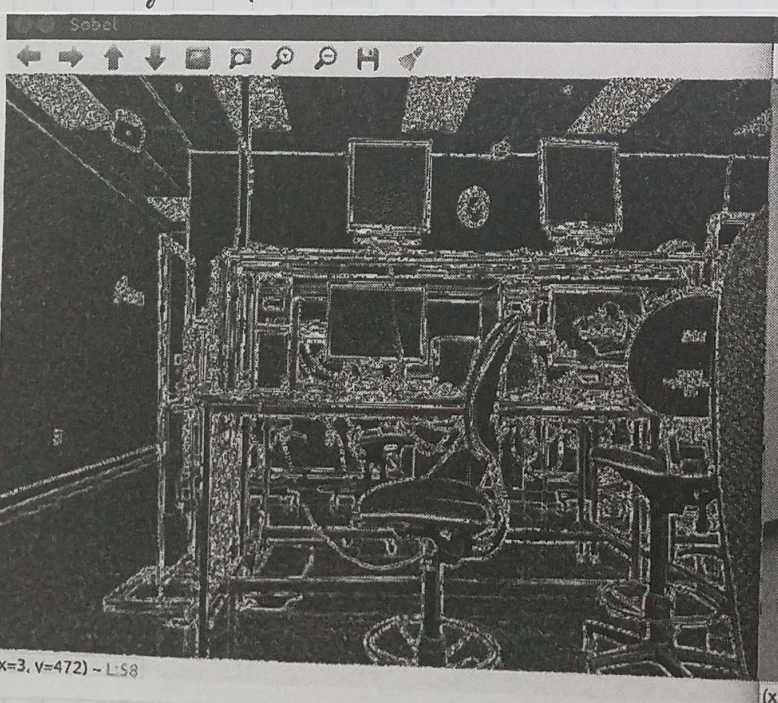
Sobel edge detection is to be computed on the Jetson TK1 GPU. A grayscale image of size 640 x 480 will be converted to edges in less than 1/30 seconds (30 fps).

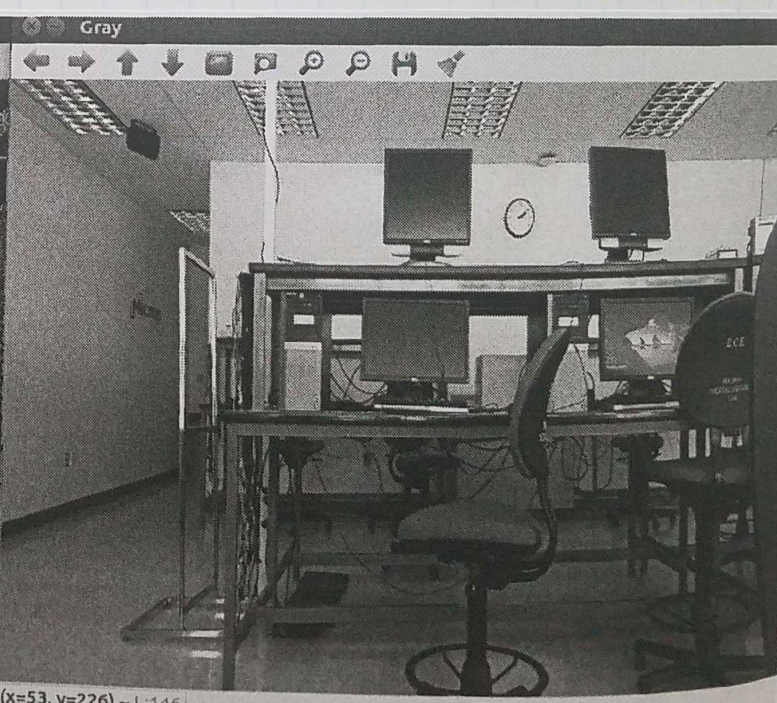
Open_cv needed to be updated to ~~3.1.~~ 3.4.1. Dana compiled the new version and sent it out to the class for use

An additional Compile flag must be used. VideoCapture would throw errors while compiling. -lopencv_videoio was added to the compile ~~comand~~ command to fix this issue.

Program speed

Required Computation speed for 30 fps: $3.\overline{33}$ ms
Average Computation over 100 Frames: 2.73681 ms





I originally tried to use arrays to compute the sobel value. Either I need to declare the arrays somewhere else, or I need to store it in Shared Memory. As the filter was short, I instead ~~used a lot of~~ unrolled the for loop and used conditional flags to see if I was on one of the image edges.

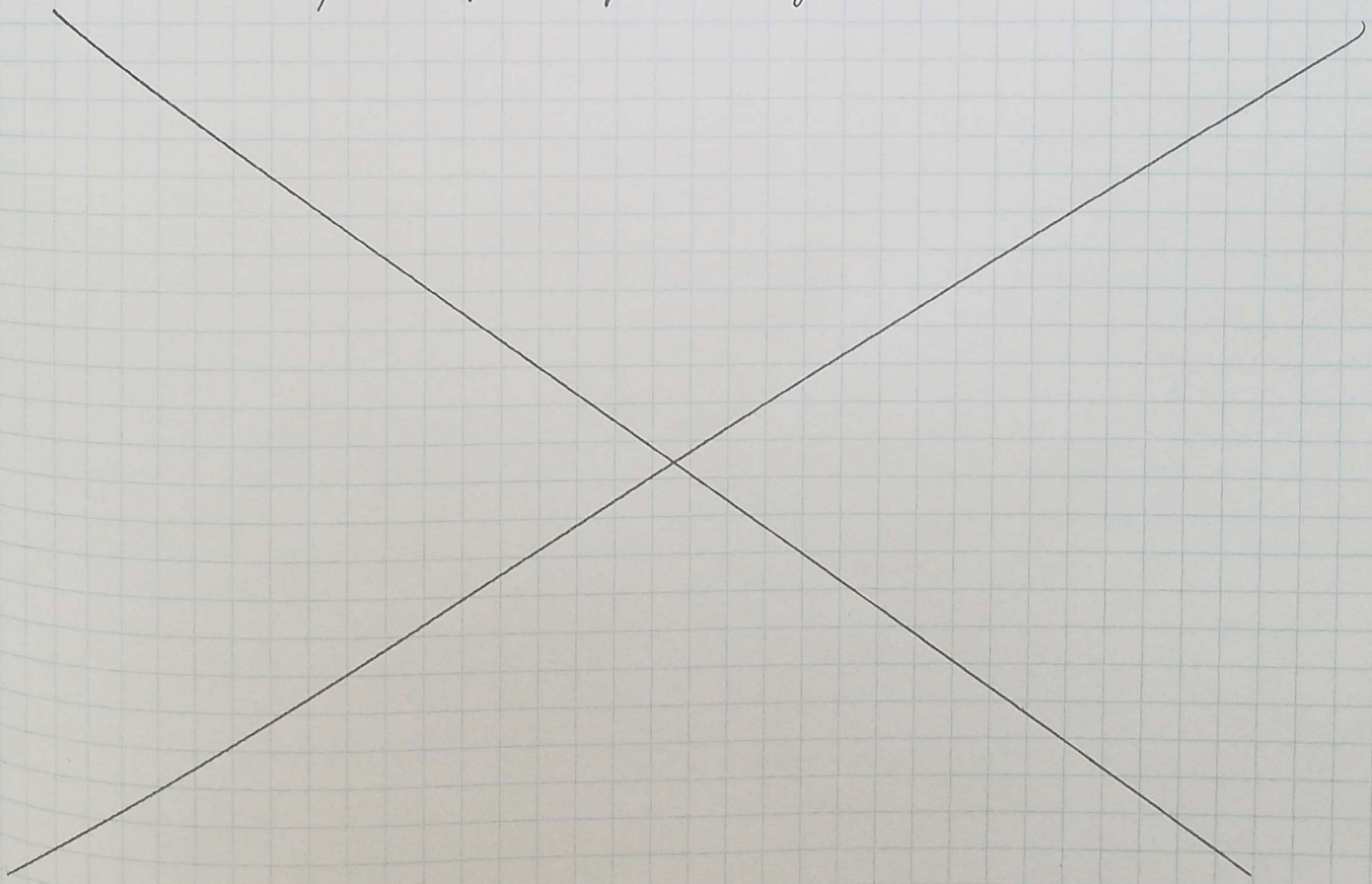
Block and Gridsize are both 16x16 giving a total thread Dimension of 256x256 or 65,536 threads.
This isn't enough to cover the picture so certain threads increment their index and compute multiple pixels

Only 1 kernal was used for the sobel function. 2 total were used for the final image.
Color → Gray → Sobel

Shared Memory wasn't very well utilized as the computation time was just faster than the required time. One way to further optimize the code would be to have ~~8 thre~~ 9 threads per pixel, each computing a part of the sobel matrix. the values could then be stored in shared memory, then added up.

$$S_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad G_x = S_1 * A \quad G_y = S_2 * A$$

$G = \sqrt{G_x^2 + G_y^2}$ ← sqrt causing issues, using magnitud instread $G = abs(G_x) + abs(G_y)$



4/25/19

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <cuda_runtime.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// BUILD: nvcc sobelEdge.cu -o sobelEdge -lopencv_highgui -lopencv_core -lopencv_videoio

__global__ void rgb2gray(unsigned char *rgb, unsigned char *gray, int num) {
    // Get thread and block indexes
    int i = ((blockIdx.y * gridDim.x + blockIdx.x) * blockDim.y + threadIdx.y) * blockDim.x + threadIdx.x;
    int step = blockDim.x * blockDim.y * gridDim.x * gridDim.y;
    int j;
    int g;
    while(i < num) {
        j=3*i;
        g = 114*rgb[j] + 587*rgb[j+1] + 299*rgb[j+2];
        gray[i] = g/1000;
        i += step;
    }
}

#define Gray(u,v) gray[(u)*cols+(v)]

__global__ void sobelEdge(unsigned char *gray, unsigned char *edge, int rows, int cols) {
    int xpixel = threadIdx.x + blockDim.x * blockIdx.x;
    int ypixel = threadIdx.y + blockDim.y * blockIdx.y;
    int xstep = blockDim.x * gridDim.x;
    int ystep = blockDim.y * gridDim.y;
    float gx,gy;
    int nleftEdge,nrightEdge,ntopEdge,nbotEdge;
    unsigned char pixelVal;
/*  float s1[]={            //reversed h form
        1, 0, -1,
        2, 0, -2,
        1, 0, -1
    };
    float s2[]={
        1, 2, 1,
        0, 0, 0,
        -1, -2, -1
    };*/

    while(xpixel<cols){
        while(ypixel<rows){
            int k,j;
            gx=gy=0.0f;
            //determind if the pixel is on any of the edges
            nleftEdge=(xpixel>0);
            nrightEdge=(xpixel<cols);
            ntopEdge=(ypixel>0);
            nbotEdge=(ypixel<rows);

            //calculate sobel derivative with edge cases
            if(nleftEdge) gx+=2*Gray(ypixel,xpixel-1);
            if(nleftEdge&&ntopEdge){
                pixelVal=Gray(ypixel-1,xpixel-1);
                gx+=pixelVal;
                gy+=pixelVal;
            }
            if(ntopEdge) gy+=2*Gray(ypixel-1,xpixel);
            if(nrightEdge&&ntopEdge){
                pixelVal=Gray(ypixel-1,xpixel+1);
                gx-=pixelVal;
                gy+=pixelVal;
            }
```

4/25/19
AP

```
        if(nrightEdge) gx-=2*Gray(ypixel,xpixel+1);
        if(nrightEdge&&nbotEdge){
            pixelVal=Gray(ypixel+1,xpixel+1);
            gx-=pixelVal;
            gy-=pixelVal;

        }
        if(nbotEdge) gy-=2*Gray(ypixel+1,xpixel);
        if(nleftEdge&&nbotEdge){
        pixelVal=Gray(ypixel+1,xpixel-1);
            gx+=pixelVal;
            gy-=pixelVal;

        }
        //take the magnitude, sqrtf causing unknown issues.
        edge[ypixel*cols+xpixel]=(unsigned char) abs(gx)+abs(gy);
        ypixel+=ystep;


    }
    ypixel=threadIdx.y+blockDim.y*blockIdx.y;
    xpixel+=xstep;

  }


}
int main(int argc, char *argv[]) {

   // OpenCV stuff: camera, window, frame, etc.

   cv::VideoCapture cam("nvcamerasrc ! video/x-raw(memory:NVMM), width=(int)640, height=(int)480,format=
(string)I420, framerate=(fraction)30/1 ! nvvidconv ! video/x-raw, format=(string)BGRx ! videoconvert !
video/x-raw, format=(string)BGR ! appsink");
   if(cam.isOpened()) { printf("camera opened\n"); }
   else { printf("camera not opened\n"); return 1; }
   int width  = cam.get(CV_CAP_PROP_FRAME_WIDTH );
   int height = cam.get(CV_CAP_PROP_FRAME_HEIGHT);
   printf("width = %d, height = %d\n",width,height);
   int frame_size = width*height*3;
   int gray_size  = width*height;
   unsigned char *h_frame_buff = (unsigned char*)calloc(frame_size,sizeof(unsigned char));
   unsigned char  *h_gray_buff = (unsigned char*)calloc( gray_size,sizeof(unsigned char));
   unsigned char  *h_sobel_buff = (unsigned char*)calloc( gray_size,sizeof(unsigned char));
   cv::Mat frame(height,width,CV_8UC3,h_frame_buff);
   cv::Mat  gray(height,width,CV_8UC1, h_gray_buff);
   cv::Mat  sobel(height,width,CV_8UC1, h_sobel_buff);
   cv::namedWindow( "Frame", 0 );
   cv::namedWindow( "Gray" , 1 );
   cv::namedWindow( "Sobel", 1 );

   // Allocate global memory on device
   unsigned char *d_frame_buff;
   unsigned char *d_gray_buff;
   unsigned char *d_sobel_buff;
   cudaMalloc((void**)&d_frame_buff,frame_size*sizeof(unsigned char));
   cudaMalloc((void**)&d_gray_buff , gray_size*sizeof(unsigned char));
   cudaMalloc((void**)&d_sobel_buff, gray_size*sizeof(unsigned char));

   // Define block and thread dimensions
   dim3 grid_size_rgb(16,16);
   dim3 block_size_rgb(16,16);

   cudaEvent_t start,stop;
   cudaEventCreate(&start);
   cudaEventCreate(&stop);
   float averageTime=0;
   // Processing loop
   int i;
   for(i=0;i<100;i++) { // For profiling
```

4/25/19

```
    //for(;;) { // Run in an infinite loop
        // Read in a camera frame
        cam >> frame;
        if( frame.empty() ) { printf("frame empty\n"); break; }

    // Memcopy frame to device
    cudaMemcpy(d_frame_buff,h_frame_buff,frame_size*sizeof(unsigned char),cudaMemcpyHostToDevice);


    cudaEventRecord(start);
    // Launch kernels
    rgb2gray<<<grid_size_rgb,block_size_rgb>>>(d_frame_buff,d_gray_buff,gray_size);
    sobelEdge<<<grid_size_rgb,block_size_rgb>>>(d_gray_buff,d_sobel_buff,height,width);
    cudaEventRecord(stop);

    // Memcopy result to host
    cudaMemcpy(h_gray_buff,d_gray_buff,gray_size*sizeof(unsigned char),cudaMemcpyDeviceToHost);
    cudaMemcpy(h_sobel_buff,d_sobel_buff,gray_size*sizeof(unsigned char),cudaMemcpyDeviceToHost);

    cudaEventSynchronize(stop);
    float milliseconds=0;
    cudaEventElapsedTime(&milliseconds, start, stop);
    averageTime+=milliseconds;
    //printf("Elapsed time %f ms\n", milliseconds);

    // Show the results
    cv::imshow( "Frame", frame );
    cv::imshow( "Gray" , gray );
    cv::imshow( "Sobel", sobel );
        cv::waitKey(1);
    }
    averageTime/=100;
    printf("Average Elapsed time %f ms\n", averageTime);
    printf("Required time for 30 fps: %f ms\n",100.0f/30.0f);

    cudaFree(d_frame_buff);
    cudaFree(d_gray_buff);
    cudaFree(d_sobel_buff);

    // For profiling
    cudaDeviceReset();

    return 0;
}
```

4/25/19