GPU Programming
(in CUDA)

Summer 2020

Designed by Julian Gutierrez, Presented by Nicolas Agostini

Session 9

Outline

- Image Processing
- Final Project

Why Do We Process Images?

- Enhancement and restoration
  - Remove artifacts and scratches from an old photo/movie
  - Improve contrast and correct blurred images
- Transmission and storage
  - Images and Video can be more effectively transmitted and stored
- Information analysis and automated recognition
  - Recognizing terrorists
- Evidence
  - Careful image manipulation can reveal information not present
  - Detect image tampering
- Security and rights protection
  - Encryption and watermarking preventing illegal content manipulation

Examples

Compression

- Color image of 600x800 pixels
  - Without compression
    - 600*800 * 24 bits/pixel
      = 11 520K bits = 1.44M bytes
  - After JPEG compression (popularly used on web)
    - only 89K bytes
    - compression ratio ~ 16:1
- Movie
  - 720x480 per frame, 30 frames/sec, 24 bits/pixel
  - Raw video ~ 243M bits/sec
  - DVD ~ about 5M bits/sec
  - Compression ratio ~ 48:1

"Library of Congress" by M.Wu
(600x800)

Denoising

From X.Li
http://www.ee.princeton.edu/~lixin/denoising.htm

Deblurring

Blurred & noisy image

Restored image

From Mathworks

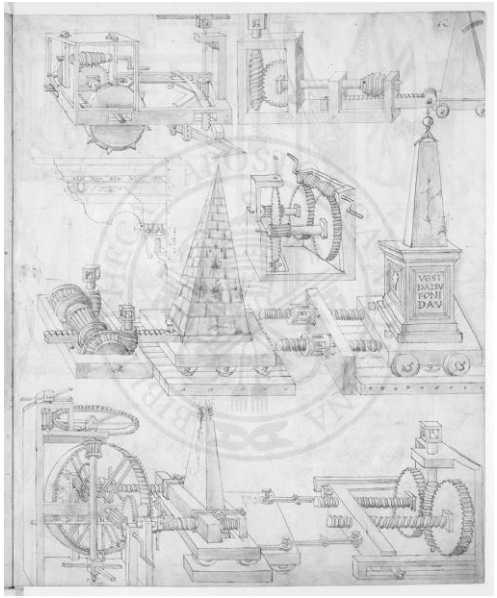
Visual Mosaicing

- Stitch photos together without thread or scotch tape

R.Radke – IEEE PRMI journal paper draft 5/01

Visible Digital Watermarks

- from IBM Watson web page “Vatican Digital Library”

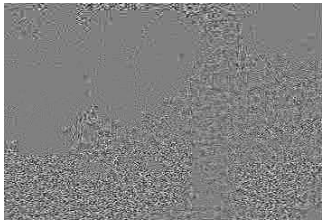
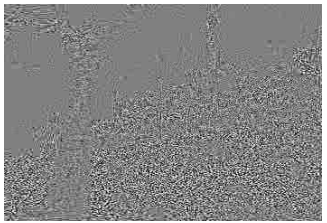


Invisible Watermark

- 1st & 30th Mpeg4.5Mbps frame of original, marked, and their luminance difference
- human visual model for imperceptibility: protect smooth areas and sharp edges



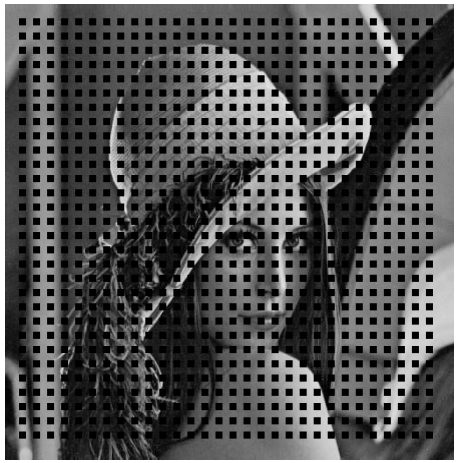


Error Concealment

(a) original Lenna image

(b) corrupted Lenna image



(c) concealed Lenna image

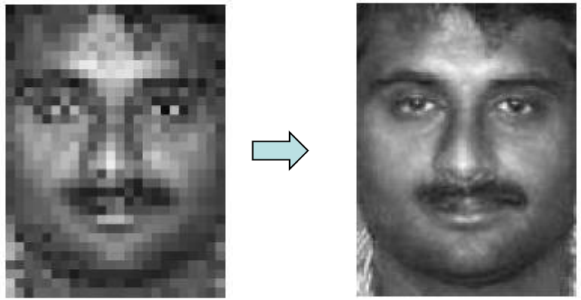
25% blocks in a checkerboard pattern are corrupted

corrupted blocks are concealed via edge-directed interpolation

Examples were generated using the source codes provided by W.Zeng.

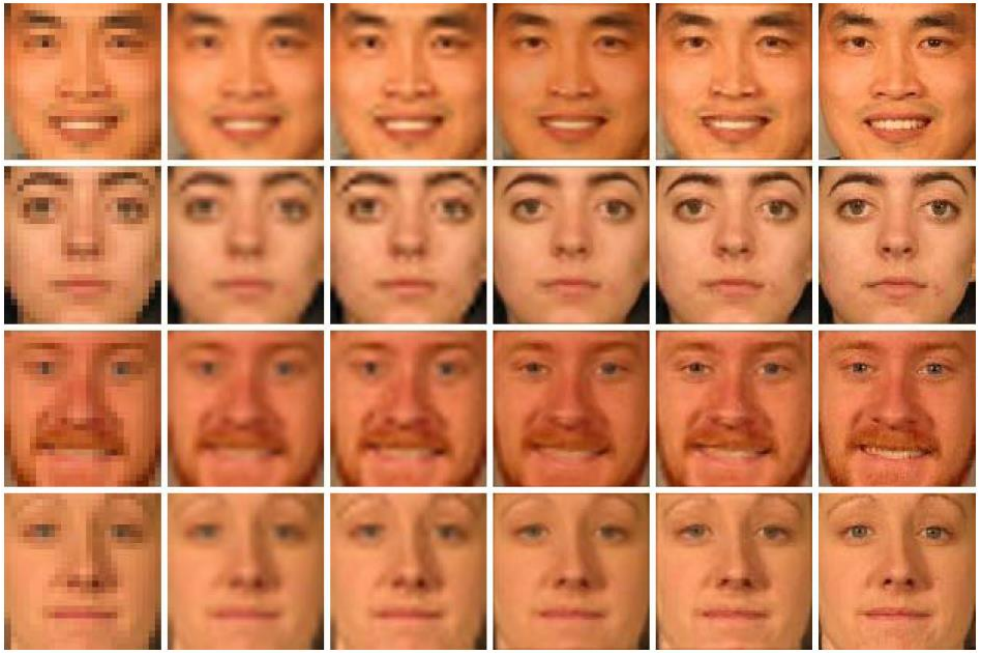
Image Super-Resolution

- Super-Resolution(SR)
  - Low resolution images ⇒ High resolution images
- Face Hallucination
  - Super-resolution on human faces



Courtesy of Jianchao Yang

Face Hallucination



Input image | Bicubic interpolation | Back projection | NMF | Sparse Coding | Original

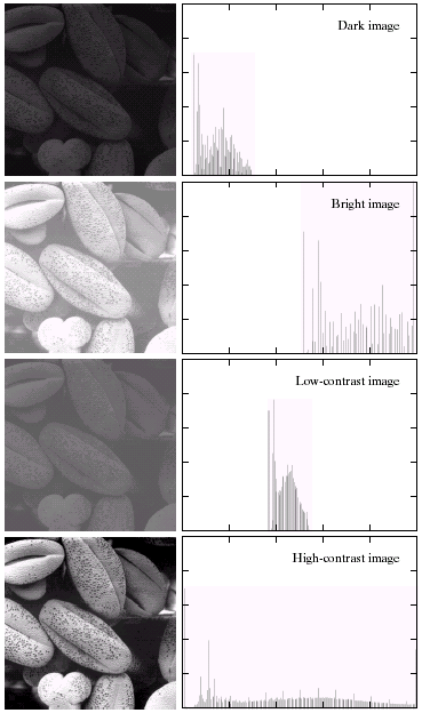
Courtesy of Jianchao Yang

Image Enhancement

a b
c d

FIGURE 3.9
(a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0$, and 5.0, respectively. (Original image for this example courtesy of NASA.)

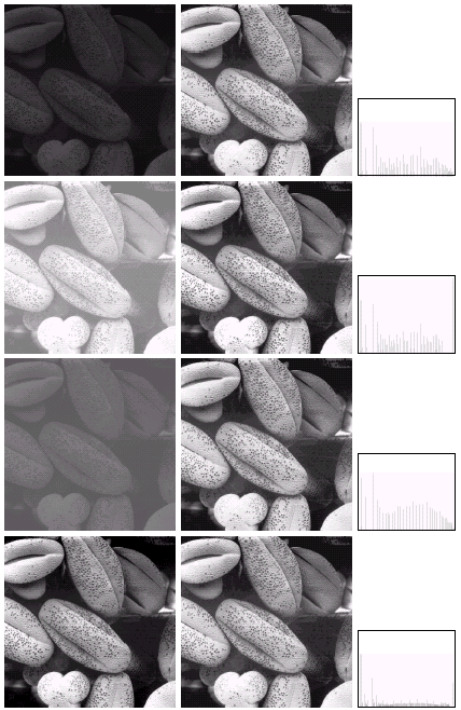
Image Histograms



a b

FIGURE 3.15 Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Histogram Equalization



a b c

FIGURE 3.17 (a) Images from Fig. 3.15. (b) Results of histogram equalization. (c) Corresponding histograms.

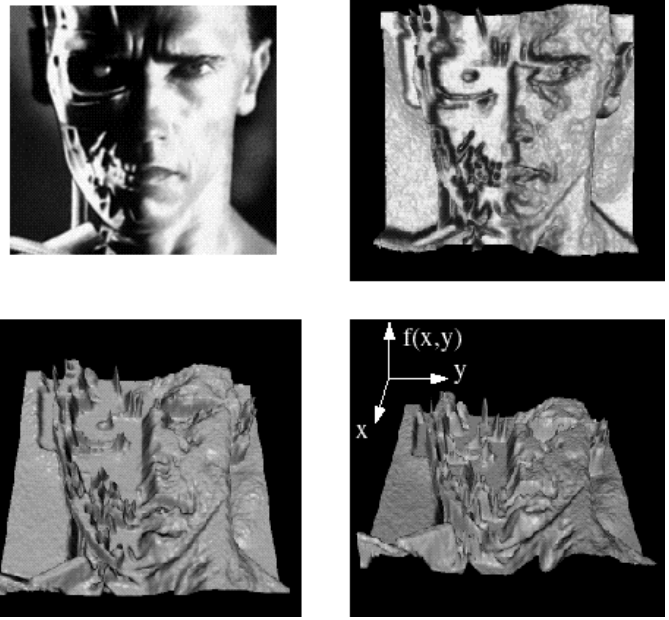
What is an Image?

What is an image?

- We can think of an image as a function, f:
  - f(x, y) gives the intensity at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - f: [a,b]x[c,d] → [0,1]
- A color image is just three functions pasted together. We can write this as a "vector-val

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Images as functions



Digital Image

- In digital imaging, a pixel (picture element) is the smallest piece of information in an image.
- The word pixel is based on a contraction of pix (for "pictures") and el (for "element")
- Each pixel is a sample of an original image, where more samples typically provide a more accurate representation of the original.
- The intensity of each pixel is variable; in color systems, each pixel has typically three or four components such as red, green, and blue, or cyan, magenta, yellow, and black.

http://en.wikipedia.org/wiki/Pixel
Google.com image search

What is a digital image?

- We usually operate on digital (discrete) images:
  - Sample the 2D space on a regular grid
  - Quantize each sample (round to nearest integer)
- The image can now be represented as a matrix of integer values

$j \longrightarrow$

$i \downarrow$

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

How can we read images?

Image file Formats

- Image file formats are standardized means of organizing and storing digital images.
- Types of files
  - JPEG
  - PNG
  - BMP
  - GIF
  - PPM
  - PGM
  - Etc

How can we read these formats

- So many different formats, which one is the best one?
- Which one is the easiest to interpret?
- The answer to these questions is not easy.
- Best way to read these types of files is through libraries such as Cimg, libjpeg, OpenCV.

Simplest version

- portable pixmap format (PPM), the portable graymap format (PGM)
  - These formats don't use any sort of compression so they are big files but easy to read the data.

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Simplest version

- In this PGM example
  - P2 means PGM
  - The next two numbers give the width and the height.
  - The next number represents the maximum value (numbers of grey between black and white)
    - Black is 0 and max value is white.
  - Then follows the matrix with the pixel values.
  - The PGM and PPM formats (both ASCII and binary versions)

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Code Examples

Simple algorithm

- Brightness
  - Increase/Decrease the intensity of the pixels by a specific value
  - Lets look at the code

What is the Makefile?

- Makefile's are a simple way to organize code compilation.
- A Makefile is a file containing a set of directives used with the make build automation tool from GNU.
- Incredibly useful when your project becomes bigger and bigger.

```
# environment
SM := 35

GCC := g++
NVCC := nvcc

# Remove function
RM = rm -f

# Specify opencv Installation
opencvLIB= `pkg-config opencv --libs`
opencvINC= `pkg-config opencv --cflags`

GENCODE_FLAGS := -gencode arch=compute_$(SM),code=sm_$(SM)
LIB_FLAGS := -lcudadevrt -lcudart

NVCCFLAGS :=
GccFLAGS = -fopenmp -O3

debug: GccFLAGS += -DDEBUG -g -Wall
debug: NVCCFLAGS += -g -G
debug: all

# The build target executable:
TARGET = brightness

all: build

build: $(TARGET)

$(TARGET): src/dlink.o src/main.o src/$(TARGET).o
    $(NVCC) $(NVCCFLAGS) $(opencvINC) $^ -o $@ $(GENCODE_FLAGS) $(opencvLIB) -link

src/dlink.o: src/$(TARGET).o
```

```
int main( int argc, const char** argv ) {
    double start_cpu, finish_cpu;
    double start_gpu, finish_gpu;

    // Read input image from argument
    Mat input_image = imread(argv[1], IMREAD_COLOR);

    int inc = atoi(argv[2]);

    // Convert the color image to grayscale image
    cvtColor(input_image, input_image, COLOR_BGR2GRAY);

    unsigned int height = input_image.rows;
    unsigned int  width = input_image.cols;

    // Construct padded image
    Mat padded;

    int gridXSize = 1 + (( width - 1) / TILE_SIZE);
    int gridYSize = 1 + ((height - 1) / TILE_SIZE);

    int XSize = gridXSize*TILE_SIZE;
    int YSize = gridYSize*TILE_SIZE;

    padded.create(YSize, XSize, input_image.type());
    padded.setTo(cv::Scalar::all(0));

    input_image.copyTo(padded(Rect(0, 0, input_image.cols,
input_image.rows)));

    // START CPU Processing
    start_cpu = CLOCK();

    Mat cpu_output;
    padded.convertTo(cpu_output, -1, 1, inc);

    finish_cpu = CLOCK();

    // START GPU Processing
    start_gpu = CLOCK();
    Mat gpu_output = padded.clone();

    gpu_function((unsigned char *) gpu_output.data,
                                   height,
                                   width,
                                   inc);
    finish_gpu = CLOCK();

    // Calculate % difference between GPU and CPU
    unsigned int wrong_pixels = 0;
    for(unsigned int j = 0; j < height; j++){
        for(unsigned int i = 0; i < width; i++){
            unsigned char cpu_pixel = cpu_output.data[width * j + i ];
            unsigned char gpu_pixel = gpu_output.data[width * j + i ];
            if (abs(cpu_pixel - gpu_pixel) > 0)
                wrong_pixels++;
        }
    }

    cout << "CPU execution time: " << finish_cpu - start_cpu << " ms"
<< endl;
    cout << "GPU execution time: " << finish_gpu - start_gpu << " ms"
<< endl;
    cout << "Percentage difference: " <<
wrong_pixels*100.0/(height*width) << " %\n";

    imwrite ("input.jpg", input_image);
    cv::Mat subImg1 = cpu_output(cv::Range(0, input_image.rows),
cv::Range(0, input_image.cols));
    imwrite ("output_cpu.jpg", subImg1);
    cv::Mat subImg2 = gpu_output(cv::Range(0, input_image.rows),
cv::Range(0, input_image.cols));
    imwrite ("output_gpu.jpg", subImg2);

    return 0;
}
```

```
void gpu_function (unsigned char *data,
                   unsigned int height,
                   unsigned int width,
                   int inc ){

    int gridXSize = 1 + (( width - 1) / TILE_SIZE);
    int gridYSize = 1 + ((height - 1) / TILE_SIZE);

    int XSize = gridXSize*TILE_SIZE;
    int YSize = gridYSize*TILE_SIZE;

    // Both are the same size (CPU/GPU).
    int size = XSize*YSize;

    // Allocate arrays in GPU memory
    checkCuda(cudaMalloc((void**)&input_gpu    ,
size*sizeof(unsigned char)));
    checkCuda(cudaMalloc((void**)&output_gpu   ,
size*sizeof(unsigned char)));

    checkCuda(cudaMemset(output_gpu , 0 , size*sizeof(unsigned
char)));

    // Copy data to GPU
    checkCuda(cudaMemcpy(input_gpu,
        data,
        size*sizeof(char),
        cudaMemcpyHostToDevice));

    checkCuda(cudaDeviceSynchronize());

    // Execute algorithm

    dim3 dimGrid(gridXSize, gridYSize);
    dim3 dimBlock(TILE_SIZE, TILE_SIZE);

    // Kernel Call
    #if defined(CUDA_TIMING)
        float Ktime;
        TIMER_CREATE(Ktime);
        TIMER_START(Ktime);
    #endif

        kernel<<<dimGrid, dimBlock>>>(input_gpu,
                                      output_gpu,
                                      inc);

        checkCuda(cudaPeekAtLastError());
        checkCuda(cudaDeviceSynchronize());

    #if defined(CUDA_TIMING)
        TIMER_END(Ktime);
        printf("Kernel Execution Time: %f ms\n", Ktime);
    #endif

    // Retrieve results from the GPU
    checkCuda(cudaMemcpy(data,
            output_gpu,
            size*sizeof(unsigned char),
            cudaMemcpyDeviceToHost));

    // Free resources and end the program
    checkCuda(cudaFree(output_gpu));
    checkCuda(cudaFree(input_gpu));

}
```

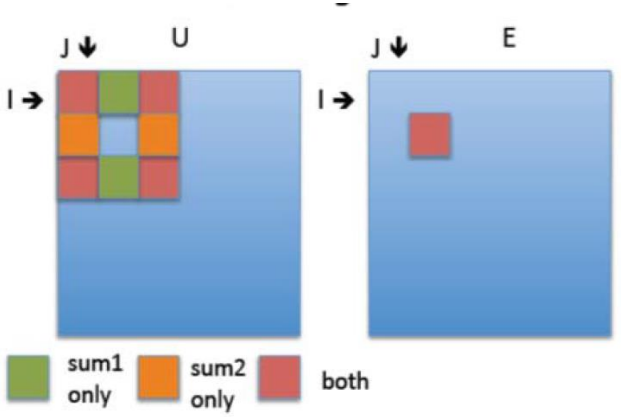
```
__global__ void kernel(unsigned char *input,
                       unsigned char *output,
                       int inc){

    // Read Input Data
    ////////////////////////////////////////////////////

    int x = blockIdx.x*TILE_SIZE+threadIdx.x;
    int y = blockIdx.y*TILE_SIZE+threadIdx.y;

    int location =  y*(gridDim.x*TILE_SIZE)+x;

    unsigned char value = input[location];

    // Algorithm
    ////////////////////////////////////////////////////

    if ((int) value + inc > 255) value = 255;
    else if ((int) value + inc < 0) value = 0;
    else value = value + inc;

    output[location] = value;

}
```

How can we improve brightness?

- Shared memory?
- Constant Memory?
- Texture memory?
- More processing per thread
  - Adding more work to each thread
- Bring in more data on each read

A more complex example

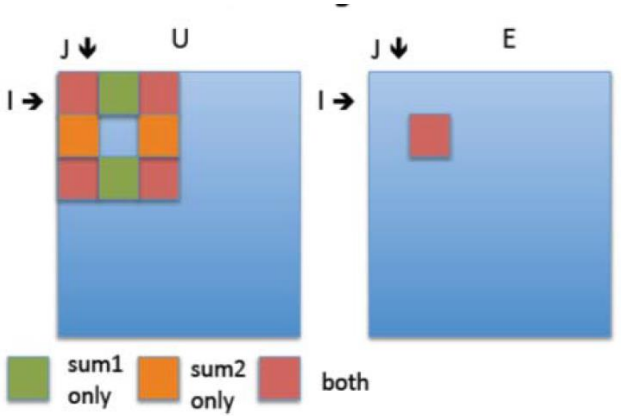
- Sobel algorithm
  - Sobel edge detection
    - Find the boundaries of the image where there is significant difference as compared to neighboring “pixels” and replace values to find edges.



Sobel Algorithm

- Looks for edges in both horizontal and vertical directions, then combine the information into a single metric.
- The masks are:

$$y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Edge Magnitude = $\sqrt{x^2 + y^2}$

Edge Direction = $\tan^{-1}\left[\frac{y}{x}\right]$

Sobel Algorithm

- Input

- Output

```
int main( int argc, const char** argv ) {

    double start_cpu, finish_cpu;
    double start_gpu, finish_gpu;

    int threshold = atoi(argv[2]);

    // Convert the color image to grayscale image
    cvtColor(input_image, input_image, COLOR_BGR2GRAY);

    unsigned int height = input_image.rows;
    unsigned int  width = input_image.cols;

    // Construct padded image
    Mat padded;

    int gridXSize = 1 + (( width - 1) / TILE_SIZE);
    int gridYSize = 1 + ((height - 1) / TILE_SIZE);

    int XSize = gridXSize*TILE_SIZE;
    int YSize = gridYSize*TILE_SIZE;

    padded.create(YSize, XSize, input_image.type());
    padded.setTo(cv::Scalar::all(0));

    input_image.copyTo(padded(Rect(0, 0, input_image.cols,
input_image.rows)));

    // START CPU Processing
    Mat cpu_output = padded.clone();

    for(unsigned int y = 0 ; y < height ; y++) {
        for(unsigned int x = 0 ; x < width ; x++) {
        if (y >= 1 && y < height - 1 && x >= 1 && x < width - 1){
                // Algorithm
                ////////////////////////////////////////////////

                int sum1 = padded.at<uchar>(y-1, x+1) -
                           padded.at<uchar>(y-1, x-1) +
                       2 * padded.at<uchar>(y  , x+1) -
                       2 * padded.at<uchar>(y  , x-1) +
                           padded.at<uchar>(y+1, x+1) -
                           padded.at<uchar>(y+1, x-1);

                int sum2 = padded.at<uchar>(y-1, x-1) +
                       2 * padded.at<uchar>(y-1, x  ) +
                           padded.at<uchar>(y-1, x+1) -
                           padded.at<uchar>(y+1, x-1) -
                       2 * padded.at<uchar>(y+1, x  ) -
                           padded.at<uchar>(y+1, x+1);

                int magnitude =  sqrt( (float) (sum1*sum1 +
sum2*sum2));

                if (magnitude > threshold)
                    cpu_output.at<uchar>(y, x) = 255;
                else
                    cpu_output.at<uchar>(y, x) = 0;
            } else
                    cpu_output.at<uchar>(y, x) = 0;
        }
    }

    Mat gpu_output = padded.clone();

    gpu_function((unsigned char *) gpu_output.data,
                                   height,
                                   width,
                                   threshold);

    finish_gpu = CLOCK();

    // Calculate % difference between GPU and CPU
    …
    return 0;
}
```

```
void gpu_function (unsigned char *data,
                   unsigned int height,
                   unsigned int width,
                   int threshold ){

    int gridXSize = 1 + (( width - 1) / TILE_SIZE);
    int gridYSize = 1 + ((height - 1) / TILE_SIZE);

    int XSize = gridXSize*TILE_SIZE;
    int YSize = gridYSize*TILE_SIZE;

    // Both are the same size (CPU/GPU).
    int size = XSize*YSize;

    // Allocate arrays in GPU memory
    checkCuda(cudaMalloc((void**)&input_gpu    ,
size*sizeof(unsigned char)));
    checkCuda(cudaMalloc((void**)&output_gpu   ,
size*sizeof(unsigned char)));

    checkCuda(cudaMemset(output_gpu , 0 , size*sizeof(unsigned
char)));

    // Copy data to GPU
    checkCuda(cudaMemcpy(input_gpu,
        data,
        size*sizeof(char),
        cudaMemcpyHostToDevice));

    checkCuda(cudaDeviceSynchronize());

    // Execute algorithm

    dim3 dimGrid(gridXSize, gridYSize);
    dim3 dimBlock(TILE_SIZE, TILE_SIZE);

    // Kernel Call
    #if defined(CUDA_TIMING)
        float Ktime;
        TIMER_CREATE(Ktime);
        TIMER_START(Ktime);
    #endif

        kernel<<<dimGrid, dimBlock>>>(input_gpu,
                                      output_gpu,
                                      width,
                                      height,
                                      threshold);

        checkCuda(cudaPeekAtLastError());
        checkCuda(cudaDeviceSynchronize());

    #if defined(CUDA_TIMING)
        TIMER_END(Ktime);
        printf("Kernel Execution Time: %f ms\n", Ktime);
    #endif

    // Retrieve results from the GPU
    checkCuda(cudaMemcpy(data,
            output_gpu,
            size*sizeof(unsigned char),
            cudaMemcpyDeviceToHost));

    // Free resources and end the program
    checkCuda(cudaFree(output_gpu));
    checkCuda(cudaFree(input_gpu));

}
```

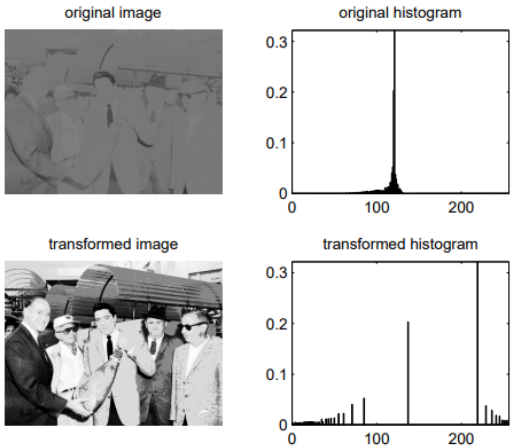
```
__global__ void kernel(unsigned char *input,
                       unsigned char *output,
                       unsigned int size_x,
                       unsigned int size_y,
                       int threshold){
     // Read Input Data
    ////////////////////////////////////////////////////////

    unsigned int x = blockIdx.x*TILE_SIZE+threadIdx.x;
    unsigned int y = blockIdx.y*TILE_SIZE+threadIdx.y;
    unsigned int location =     y*(gridDim.x*TILE_SIZE)+x;

    // If thread out of image range, exit
    if (y >= 1 && y < size_y - 1 && x >= 1 && x < size_x - 1){

        // Algorithm
        ////////////////////////////////////////////////////////
         int sum1 = input[ (gridDim.x*TILE_SIZE) * (y-1) + x+1 ] -
                    input[ (gridDim.x*TILE_SIZE) * (y-1) + x-1 ] +
                2 * input[ (gridDim.x*TILE_SIZE) * (y)   + x+1 ] -
                2 * input[ (gridDim.x*TILE_SIZE) * (y)   + x-1 ] +
                    input[ (gridDim.x*TILE_SIZE) * (y+1) + x+1 ] -
                    input[ (gridDim.x*TILE_SIZE) * (y+1) + x-1 ];
         int sum2 = input[ (gridDim.x*TILE_SIZE) * (y-1) + x-1 ] +
                2 * input[ (gridDim.x*TILE_SIZE) * (y-1) + x   ] +
                    input[ (gridDim.x*TILE_SIZE) * (y-1) + x+1 ] -
                    input[ (gridDim.x*TILE_SIZE) * (y+1) + x-1 ] -
                2 * input[ (gridDim.x*TILE_SIZE) * (y+1) + x   ] -
                    input[ (gridDim.x*TILE_SIZE) * (y+1) + x+1 ];
         int magnitude =  sqrt( (float) (sum1*sum1 + sum2*sum2));

        if (magnitude > threshold)
            output[location] = 255;
        else
            output[location] = 0;
    }

}
```

Final Project

Open File

Histogram Equalization

- Basics
  - Image histogram equalization is an algorithm commonly used to preprocess images to enhance the contrast of the image by trying it's best to "normalize the histogram" of the image.
  - Example:



Histogram Equalization

- Algorithm:
  - Calculate histogram of the image
  - Calculate PMF of image (histogram / total # pixels)
  - Calculate the CDF (cumulative distributive function) (Prefix Sum)
  - Calculate the CDF * 255 to obtain the new gray level values
  - Use the old gray value and map it into the CDF * 255 array to obtain the new gray level value of the pixel

Possible Optimizations

- Use shared memory
- Optimize the histogram using local histograms and then updating globally
- Process multiple pixels per thread
- Use texture memory
- Incorporate pinned memory (covered next week)
- Use streams for more parallelization (covered next week)
- etc…

Histogram Equalization

- For more information on histogram equalization:
  - http://www.tutorialspoint.com/dip/Histogram_Equalization.htm
  - https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf
- Animation explaining Histogram Equalization by Telkon University
  - https://www.youtube.com/watch?v=PD5d7EKYLcA