**CSCD 439/539 GPU Computing Homework 2 (Total 100 points)**

**Simple Image Processing On CUDA Device**

**Description**

In this project, we manipulate PGM images (Portable Gray Map). PGM files could be pure ASCII text files, with header information and intensity (brightness) value for each pixel in an image file. If you have an image named ballon.pgm, you can use command ***less ballon.pgm*** to explore its format. Of course, you can install an image viewer to visualize the image with your naked eye. On windows, you can use **Irfanviw**, download is available at <http://www.irfanview.com>

On a Mac machine, you can download **ToyViwer** in your **apple store for free**.

**The detailed format description for PGM file can be found here.** (Also you can download more sample PGM files, besides one included in this project package.)

[**http://people.sc.fsu.edu/~jburkardt/data/pgma/pgma.html**](http://people.sc.fsu.edu/~jburkardt/data/pgma/pgma.html)

**The following is an example PGM file named smallFile.pgm**

P2

# feep.ascii.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

**The image header consists of the first 4 lines.**

1. The first line **P2**, is a magic number to tell the image viewer that the file is a ASCII pgm file.
2. The second line starts with #, is a line of comment.
3. The numbers in the third line means this image has 24 **COLUMNS**, and 7 ROWs of pixels in it. **Note: column goes first!**
4. The fourth line means the maximum intensity value in the image is 15. Each pixel has intensity value. **The bigger intensity value is, the brighter or whiter at that location is in the image. The intensity value 0 means a total black color in the image.**

Intensity values of all pixels are listed following the file header, starting with 5th line. These intensity values could be considered as a 2D array, with the **row index** and **column index** specifying each point (or pixel) at a particular location.

**What you should do?**

**Please carefully read the comments on top of each function declared in pgmUtility.h file.**

1. In pgmUtility.c file, implement the function that reads in the image **(actually a text file)** using file I/O. The function has been declared already in pgmUtility.h file.

int \*\* pgmRead( char \*\*header, int \*numRows, int \*numCols, FILE \*in );

1. In pgmUtility.c file, implement the function that paints a black dot (circle) in the image, you have to parameterize the center pointer and the radius of the circle you will draw. The function has been declared already in pgmUtility.h file. Inside this function, you have to invoke a CUDA kernel function that performs all image processing on GPU device. Please **also** design and implement that kernel function that can draw a circle on the image using CUDA.

int pgmDrawCircle( int \*\*pixels, int numRows, int numCols, int centerRow,

int centerCol, int radius, char \*\*header );

1. In pgmUtility.c file, implement the function that paints a black edge frame in the image, you have to parameterize the width of the edge you will paint. The function has been declared already in pgmUtility.h file. Inside this function, you have to invoke a CUDA kernel function that performs all image processing on GPU device. Please **also** design and implement that kernel function that can draw an edge frame on the image using CUDA.

int pgmDrawEdge( int \*\*pixels, int numRows, int numCols, int edgeWidth, char \*\*header );

1. Note that after you paint the edge or the black dot in an image, you may have to update the header to reflect the new maximum intensity value in the header (specifically the last line of image header). But in this homework, you are NOT required to update the maximum intensity value in image header.
2. In pgmUtility.c file, implement the function that writes back to a new image file that contains your painting using file I/O. The function has been declared already in pgmUtility.h file.

int pgmWrite( const char \*\*header, const int \*\*pixels, int numRows, int numCols, FILE \*out );

1. You are required to define the functions that are provided in the \*.h file, you CANNOT change the signature (parameter list and return type) of these **existing** functions, though you are allowed to add other functions or definitions. You have to implement them in its corresponding \*.c file and use them in main or other functions if necessary. **Please carefully read the comments on top of each function declared in pgmUtility.h file.**
2. In addition, please design and implement two of your CUDA kernels to do the image processing, and call these kernels in relevant functions defined above. You are required to place all CUDA functions into a separate pgmProcess.cu file, and declare them in corresponding pgmProcess.h file.
3. You are required to include a Makefile in your folder that compiles all your code and link them into a target named **myPaint**.
4. You are required to deallocate all memories you dynamically allocated in program (either on host or device). Please make sure you do it in a proper timing and at proper location in your program, where you know these memories are no longer useful.
5. You have to write your own main() function, in which if necessary, you call some of these function(s) specified above in order to **output a new image file on disk**. Depending on the command-line arguments passed in, the new image should look like what is shown in the **test cases section** below.
6. In your main function, you have to write code to parse the command-line arguments. (FYI: a sample command-line arguments parsing code is provided in package). When you run you program, you pass in the Circle Center, Radius or Edge Width as command line arguments. For example,

If the number of command line argument is not expected, your program are required to show a message:

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

You have to run your program using command with this synopsis:

***./programName –e edgeWidth originalImage newImageFile***

to paint an edge of width of ***edgeWidth*** in the image of ***originalImage***

***./programName –c circleCenterRow circleCenterCol radius originalImage newImageFile***

to paint an big round dot on the image with center at (***circleCenterRow, circleCenterCol***) and radius of ***radius***.

When user inputs wrong number of argument in command line or wrong input format, your program **should not crash**, instead showing the Usage message above.

If your program could handle drawing a circle and an edge at the same time in one command in your terminal window you get another **5** bones points on top of 100.

In this case, your command line should look like the following; otherwise you cannot get the bonus.

***./programName -ce circleCenterRow circleCenterCol radius edgeWidth originalImage newImageFile***

***AND***

***./programName -c -e circleCenterRow circleCenterCol radius edgeWidth originalImage newImageFile***

**Test cases with the provided image**

**./myPaint -c 470 355 100 ./balloons.ascii.pgm balloons\_c100\_4.pgm**

**Your program yields an image looks like:**

****

**./myPaint -c 228 285 75 ./balloons.ascii.pgm balloons\_c75\_5.pgm**

**The command above yields an image,**

****

**./myPaint -e 50 ./balloons.ascii.pgm balloons\_e50\_2.pgm**

**The command above produces an image that looks like,**

****

**If input wrong command line parameters:**

./myPaint -e 50 300 ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

./myPaint -e ab ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

./myPaint -e ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

./myPaint -c 470 355 ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

./myPaint -c 470 355 50 60 ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

./myPaint -c 470 90bc ./balloons.ascii.pgm

***Usage:***

***-e edgeWidth oldImageFile newImageFile***

***-c circleCenterRow circleCenterCol radius oldImageFile newImageFile***

**The original image before your processing looks like:**



**To turn in:**

Please wrap up all your source code and all included test images into a single zip file, name it as lastname + firstinitial + hw2.zip. If you are Will Smith, your zip file is named as smithwhw2.zip.

**If you did the bonus points part, please clearly output a message on the stdout.**

Turn in your single zip file on Canvas CSCD439 -> Assignments->Hw2->submit

If your code shows a compile error, you will get zero credit. If your file is corrupted, you get a zero credit.