3460:677 LAB 3: IMAGE PROCESSING

Objective

The purpose of this lab is to experiment with simple image processing in two ways. First, you will convert an RGB image into a grayscale equivalent. Second, you will implement an efficient image blurring algorithm for an input image. In both cases, the image is represented as RGB `float` values.

Prerequisites

Before starting this lab, make sure that you have completed all of Online Module 3.

Local Setup Instructions and Overview

The most recent version of source code for this lab can be found on the class web page. Download the files to a folder in your Windows account on a lab machine. (Because we need graphics display capabilities for this lab we must use a local machine and not remote access to `tesla`.) The files include:

1. *runviewer.bat* — Special Windows batch file for compiling and executing viewer.
2. *viewer.cu* — CUDA program for displaying a bitmap file.
3. *bitmap_help.h* — Additional definitions needed by the viewer program.
4. *gl_helper.h* — Header file for setting up GLUT in your system.
5. *cpu_bitmap.h* — Header file for creating bitmaps with GLUT.
6. *glut32.lib*, *freeglut.dll* and *freeglut32.lib* — The OpenGL Utility Toolkit (GLUT), a simple windowing API.
7. *pic1.bmp* — Sample image to test program.

Executing the Base Program

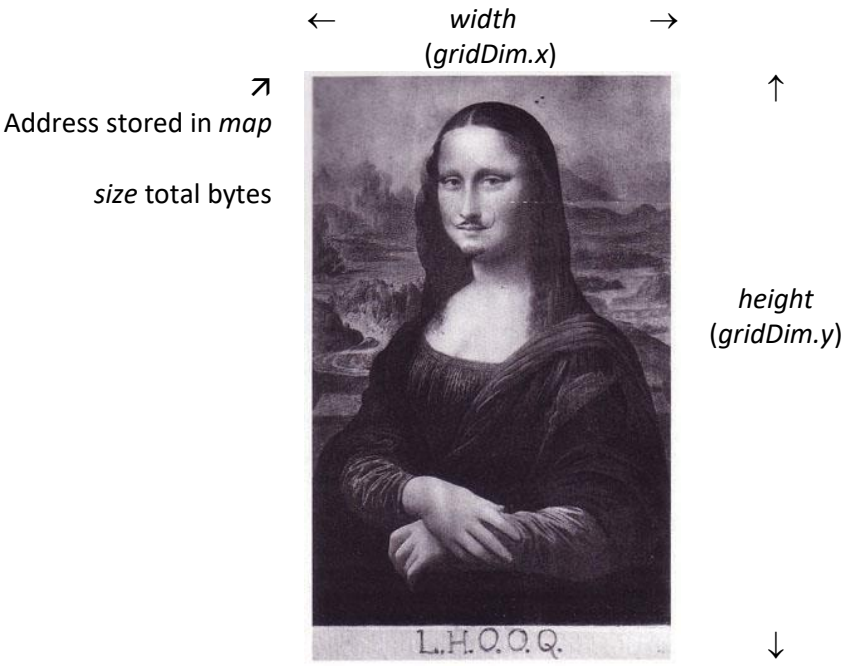
This program simply reads a bitmap (*.bmp*) file and displays it. On one of the Windows machines in CAS 241 or 254, launch a MS Visual Studio command prompt window, navigate to your local copy of the viewer files, and type *runviewer*. After the program finishes compiling and begins execution, type *pic1* for the name of the file to display. (The program adds on the *.bmp* extension.) When you decide you've enjoyed the view long enough, close the viewer window and type [CTRL][C]. Type "y" at the "Terminate batch job (Y/N)?" query to return a command prompt.

Examining the Program

As you read through the program as-is, you will note the unused `__host__` function *imgProc* atop the code. There are four arguments passed to the function:

1. The starting address of the pixels in host memory is passed in the variable *map*.
2. The total number of bytes used to store the image is passed as *size*.
3. The dimensions of the image are passed in the variables *width* and *height*.

These arguments are summarized in the drawing below. Each pixel has ***four*** components to it: red, green and blue components (in that order), followed by transparency information (called *alpha*, typically zero). Thus, when accessing individual pixels, you should jump around the image using offsets of 4.



General Instructions

Edit *imgProc()* to perform the following:

- Allocate device memory.
- Copy host memory (the bitmap pixel data) to device.
- Create a *width*-by-*height* grid of 1-by-1 blocks. Each block corresponds to an individual pixel, whose coordinates are given as *blockIdx.x + blockIdx.y * gridDim.x*. (Remember that access to global memory is only in the form of 1-D arrays.)
- Invoke a CUDA kernel which you will write. Insert this kernel code prior to *imgProc()*.
- Copy results from device to host.
- Deallocate device memory.

Because there are two different transformations to implement you will write two different versions of the program with two different kernels. Once you're sure they work, rename them `grayscale.cu` and `blur.cu` before submitting them for grade.

Instructions, Part 1 (Color to Grayscale program)

As outlined above, each thread determines its i.d. and thus its pixel. Adapt the code from the lecture notes for the actual grayscale calculation. For your program simply reset the red, green and blue color

components of the original pixel to this common value. Remember to synchronize the threads so you get a complete grayscale image when finished. In your lab report answer these questions:

(1) How many floating operations are being performed in your color conversion kernel? EXPLAIN.
(2) How many global memory reads are being performed by your kernel? EXPLAIN.
(3) How many global memory writes are being performed by your kernel? EXPLAIN.
(4) Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.
(5) Name three applications for color conversion.

Instructions, Part 2 (Image Blur program)

Operate directly on the RGB float values and use a 3-by-3 box filter to blur the original image. As before adapt the code from the lecture notes. In your lab report answer these questions:

(1) How many floating operations are being performed in your color conversion kernel? EXPLAIN.
(2) How many global memory reads are being performed by your kernel? EXPLAIN.
(3) How many global memory writes are being performed by your kernel? EXPLAIN.
(4) Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

Lab Report and Submission

When finished write a one-to-two page lab report (single-spaced, 1-inch margins, 12- to 14-point font size) summarizing your work and findings. In your report answer the questions listed above. Submit the two edited `.cu` files and your lab report by the due date for grade.