CMPSCI 373 (S20) Homework 2: Image Processing

Overview

In this homework you will implement several image processing algorithms, including pixel operations, convolution, dithering, and mosaicing.

IMPORTANT: Browser Requirement

- Because this homework requires accessing image files from local folders, an operating forbidden by default in most modern browsers, you will need to resolve this restriction first. Specifically:
  - If you use Firefox (**which we highly recommend**), type `about:config` in the url, proceed and search for option `privacy.file_unique_origin`, set it to **false**. Then restart Firefox.
  - If you use Safari, follow the Safari instructions here to enable local file access.
  - If you use other browsers, a generally working approach to enable file access is to launch a web server in your Homework 2 folder (such as by using `python -m SimpleHTTPServer 80` or installing Apache), and then in your browser open url *localhost*. This enables file access through HTTP, bypassing local file security issue. Alternatively, you can upload your code to a remote web server and access the server url.
- We recommend you to keep the browser's **Web Console** open at all times while working on this homework, to facilitate debugging and error checking. In Firefox, this can be turned on by going to *menu -> Web Developer -> Web Console*.

Starter Code

- **Download** the starter code from Piazza, and **unzip** it. The starter code consists of the following:
  - `index.html` is the main HTML file and includes other .js (Javascript) files.
  - `dat.gui.min.js` is a Javascript library for building a simple and stylish user interface (it creates a control panel with sliders, buttons etc.).
  - `imProcess.js` contains the main program code of this homework.
  - There is also a `sample_images` subfolder and several mosaic dataset images.
  - The code uses `p5.js` library for loading, saving, and displaying images. You do NOT need to be familiar with this library as how to access pixels values are evident in the provided examples.

- Open `index.html` in your browser (or your server url if you use a web server), you should see the first sample image, and a control panel on the right side. The starter code has only implemented some features (such as adjusting brightness, Gaussian blur, edge detection), and the rest are yours to complete. First, take a look at the control panel:
  - Select an input image from the **image** dropdown list.
  - In the **Pixel Operations** panel, you can adjust **brightness**, **contrast**, **saturation**. These operations are applied immediately after you move the sliders.
  - You can reset the image to the original input by clicking **Reset**.
  - In the **Image Convolution** panel, you can choose **Box Blur**, **Gaussian Blur**, **Sharpen**, or **Edge Detect**. You can change the relevant parameters, but these operations only apply after you click on the associated buttons (such as Apply Box Blur, Apply Gaussian Blur etc.)
  - In the **Dither** panel, you can choose **uniform**, **random**, or **ordered** dithering.
  - In the Image **Mosaic** panel, you can choose the mosaic dataset, then click **Apply Mosaic** button to compute image mosaic.

CMPSCI 373 (S20) Homework 2: Image Processing

After playing with the user interface, open **`imProcess.js`** in a code editor, and take a look at the code.

- This is the only file you need to modify. In fact, functions that require your work are all marked by the **===YOUR CODE STARTS HERE==** throughout the code. You **should NOT** need to create any new functions, modify any function not marked, or include any other Javascript library.
- In Javascript, you can declare a local variable by using the **`let`** keyword (e.g. **`let i=0;`**) or **`var`** keyword. We strongly prefer the **`let`** keyword as it provides block-level scope (while **`var`** provides function-level scope).
- You can create an array by using square bracket (e.g. **`let array=[];`**). Perhaps somewhat surprisingly, you do not have to specify the size, as the array size is dynamic. You can access an array element directly (e.g. **`array[5]=10;`**) and the array size (**`array.length`**) will update dynamically. You can also initialize array values. For example, you can define a 2-dimensional box kernel (3x3) by: **`let box=[ [1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9] ];`**
- Unlike Java and C/C++, Javascript **does NOT** observe integer division. For example, `1/9` would give a floating point value of `0.11111...` even though both sides of the division are integers. In order to round it to the closest integer, you can either use **`Math.round()`** function, or use **`>>0`** such as **`Math.round(1/9)`** or **`(1/9)>>0.`**
- All image processing functions take an **`input`** image parameter and an **`output`** image parameter. The **`input`** image should be **read-only** and NEVER be modified; the result should be stored in the **`output`** image only and is what will be displayed on the web interface.

What You Need to Do (30 points)

The following is a list of features that you must implement and the number of points they are worth:

- [0 pts] **brighten**, **blur**, and **edgeDetect** are already provided to you as starting examples.
- [4] **adjustContrast**: change image contrast using the **`contrast`** parameter. You must implement the algorithm we covered in lecture, otherwise you will get 0.
- [4] **adjustSaturation**: change image saturation using the **`saturation`** parameter. You must implement the algorithm we covered in lecture, otherwise you will get 0.
- [2] **sharpen**: sharpen an image using the **`sharpness`** parameter. The **`filterImage`** function is already provided to you and it implements 2D image convolution. So practically you just need to define the sharpen kernel, and then call **`filterImage`** to compute the result. Again, you must use the sharpen kernel we covered in lecture, otherwise you will get 0.
- [2] **uniformQuantization**: implement uniform quantization to convert an image to binary (black-white). As the input is a color image, use each pixel's grayscale (i.e. luminance) value. The output should have each pixel either black **`[0,0,0]`** or white **`[255,255,255]`**.
- [4] **randomDither**: implement random dithering.
- [4] **orderedDither**: implement ordered dither, using the 4x4 ordered matrix covered in lecture.
- [10] **imageMosaic**: compute image mosaic. You must implement the **final version** of image mosaic algorithm we discussed in class (which includes the scaling of mosaic images and also randomization). Since the algorithm involves several nested loops and can take a while to run, this function will be implemented using non-blocking loop so that the browser will not complain about its running speed. Details are in the comments embedded in the code.

CMPSCI 373 (S20) Homework 2: Image Processing

Helpful Hints

- Start with the simple tasks (those worth less points). Look at the the provided examples (**`brighten`** and **`blur`**) which serve as excellent starting points for the required tasks.
- Review Image Processing lectures as most tasks have pseudo-code in lecture slides.
- Keep your browser's **Web Console** window open at all times so you can check error messages in case your program has a syntax error. You can also insert **`console.log()`** to print variable values for debugging purpose.
- The starter code has already implemented all of image loading, update, display, and user interface functions. You just need to complete the required functions, using the parameters of each function.
- In **imageMosaic**, since the width and height of the input image are generally not divisible by the mosaic image width and height, the result will have a border of partial blocks at the right and bottom sides. You can simply ignore those partial blocks and do not need to process them.
- As **imageMosaic** can take a while to finish, particularly for large images, you should optimize the code to increase the computation speed. For example, the denominator of the distance metric only depends on each mosaic image and not on the input image blocks, so you can pre-compute the denominator values, store them in an array, and use them in the main computation loops. This will reduce the overhead of computing these values over and over again.
- Try your code on a number of different input images -- it is possible that the code may run fine for a particular input image but may fail or present bugs on other input images. It's a good idea to test different inputs.

Submission

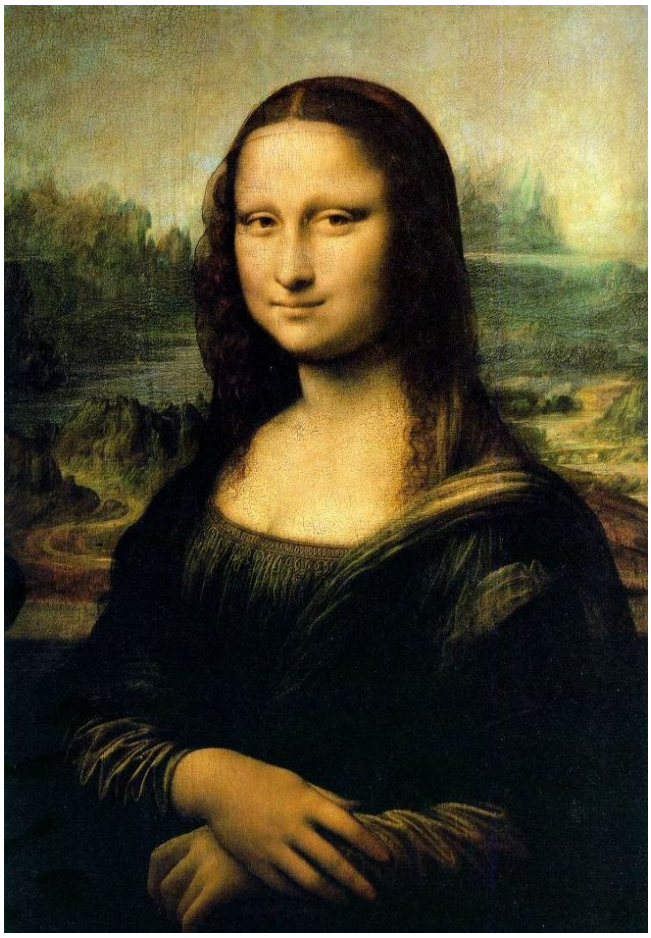
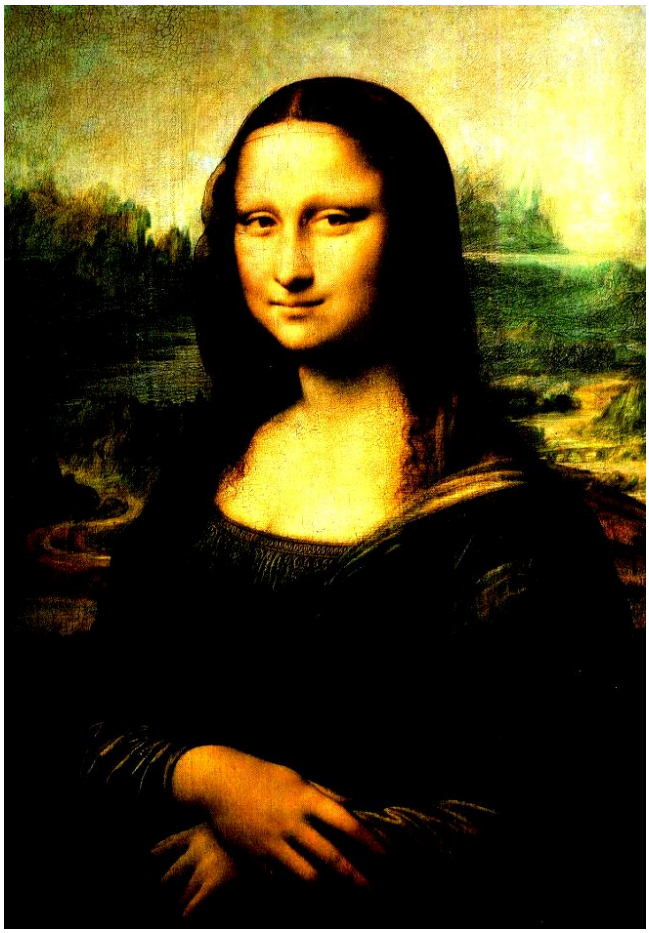
Zip your homework 2 folder (including all files) and submit the zip file to Gradescope.

Sample Results

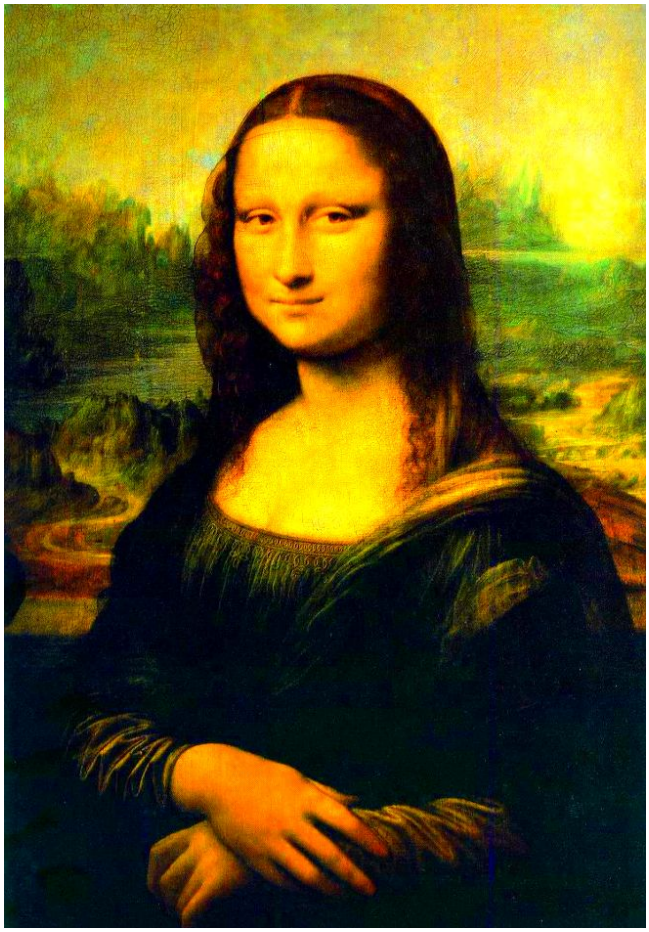
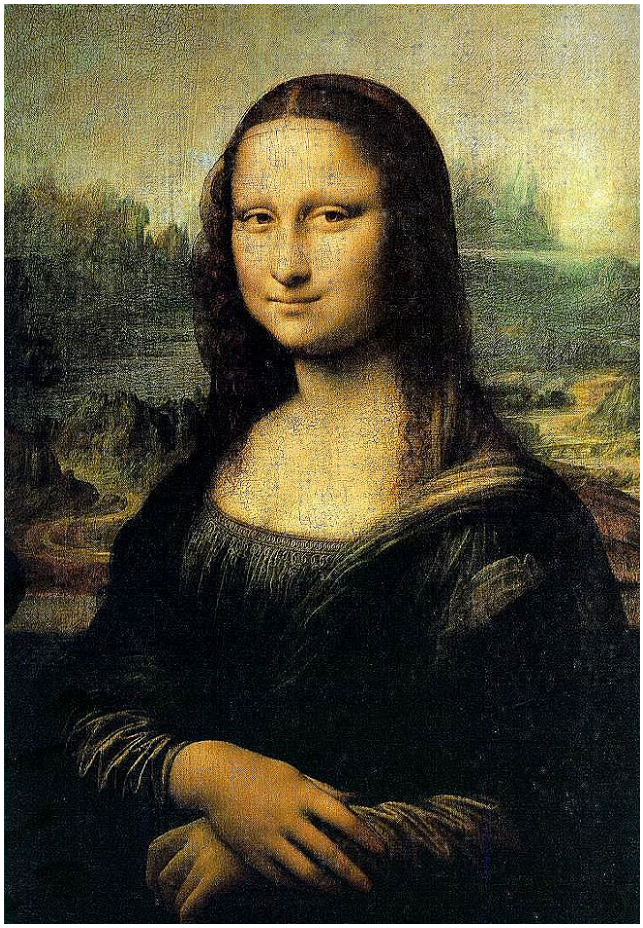
To help you visually verify the correctness of your implementations, see sample results in the next few pages.

CMPSCI 373 (S20) Homework 2: Image Processing

Sample Results

	
monalisa.png	contrast=2

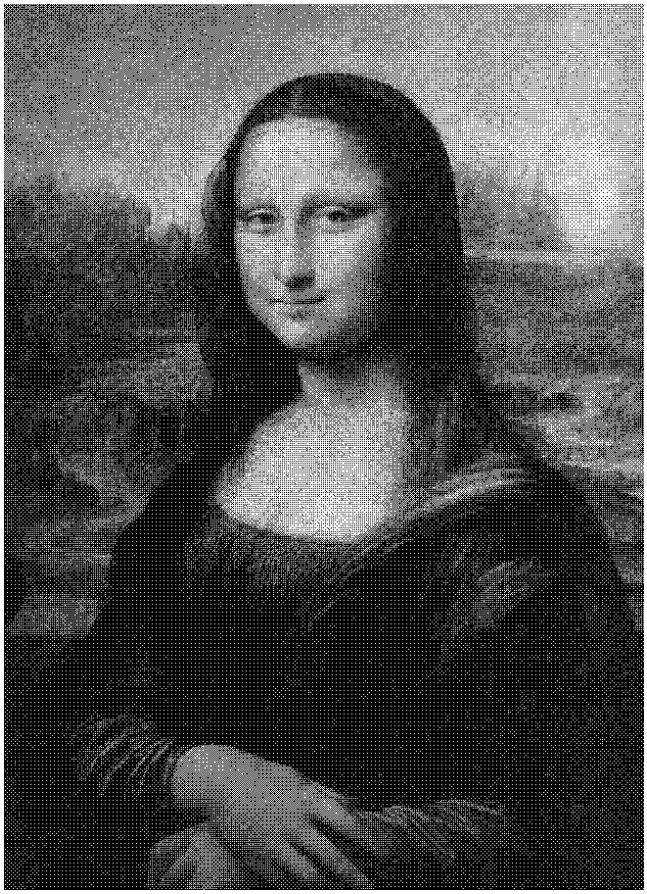
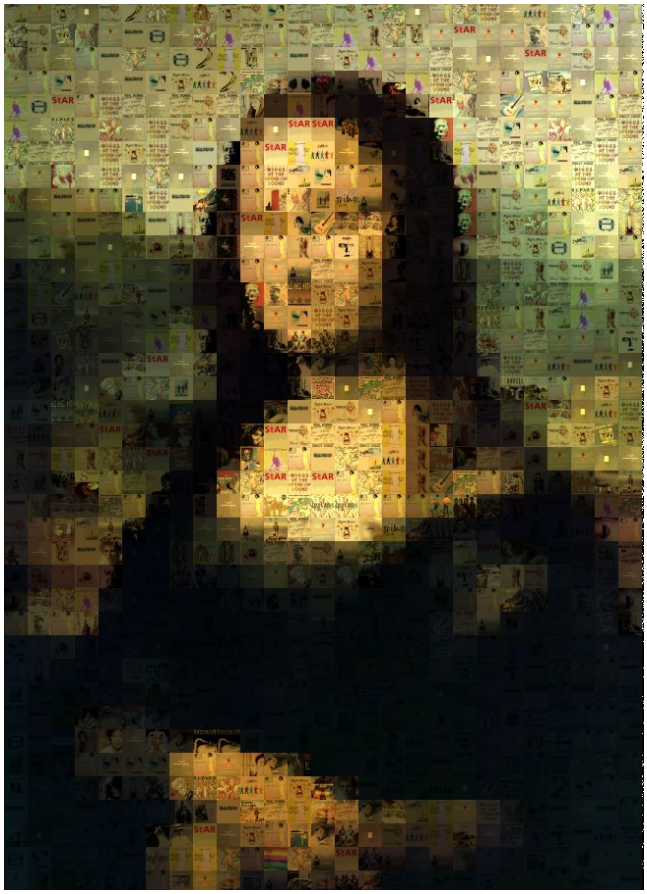
CMPSCI 373 (S20) Homework 2: Image Processing

	
saturation=3	sharpness=1

CMPSCI 373 (S20) Homework 2: Image Processing

	
Uniform quantization	Random dither

CMPSCI 373 (S20) Homework 2: Image Processing

	
Ordered dither	mosaic