Project 4 **CSC 470-03** **Spring 2019**

Description

You are the JavaScript engineer on a project to develop a web application that lets the user apply a variety of filter effects to images loaded in a browser. You have demonstrated that intensity transformation filters limited to pixel point operations (e.g. grayscale, brighten, sepia, …) have run times that are well within design specifications. Unfortunately, more complex filters involving the processing of a region of pixels (e.g. median, sharpen, blur, …) take too long to complete and make the browser window unresponsive. To improve the experience of users applying these filters you have decided to split the image horizontally into roughly equal strips and divide the work amongst multiple Web Workers. The goal of this project is to demonstrate the viability of this idea using the median filter, which is commonly the most computationally intensive of all pixel region processing filters.

Requirements

1. Select a large image to use to test your program (several example provided).
2. Create an HTML file that loads your image, resizes a `<canvas>` element, and draws the image on the canvas.
3. Include an `<input type="range" …>` element on your page that allows the user to select 1 to 8 Workers to process the image. (See `project4_start.html`)
4. Include a `<button>` element on your page that creates and sets up the selected number of Worker objects when clicked. Each Worker must load your processing script. Manage all Worker objects in an Array.
5. For each Worker created:
   - Get an ImageData object from the canvas 2D context that holds a separate horizontal strip of pixel data to be processed. Use the 2D context's `getImageData(…)` method to get each ImageData object.
   - Create a message object containing at a minimum this ImageData object as well as the (x, y) coordinates of the original canvas where the image data should be redrawn after it is processed.
   - Post a message to the new Worker with this three-part object as its message data.
6. Each Worker object's "message" event should be handled by a function that does the following.
   - Receive processed data from each Worker as a modified ImageData object
   - Use this processed ImageData and the returned (x, y) coordinates to redraw the processed image data back to the canvas at the proper location.
   - Terminate the Worker.
7. Write a separate script file to be loaded by your Worker objects. (See `process_start.js`)
8. Your Worker script must handle the "message" event (`self.onmessage…`) so that image data is processed and returned.
9. Each Worker script should extract the received ImageData's Uint8ClampedArray object to be processed using the median filter. (See `median` function in `process_start.js`)
10. When a Worker's processing is complete, replace the ImageData object's Uint8ClampedArray (the `.data` property) with the new Uint8ClampedArray that holds processed pixel data. Use the Uint8ClampedArray object's `.set(…)` method to replace the data. (i.e. `myImageData.data.set( myNewArray );`)
11. You must have at least one main HTML file and one JavaScript file that is used by your Worker objects.
12. You MUST enter header comments into your JavaScript file including (1) File name, (2) Your name, (3) Description and or purpose of the assignment.
13. You MUST comment your code, explaining what you did in each section.
14. Submit your HTML and/or JavaScript files using Canvas under the appropriate assignment.

Hints

To split each image into horizontal strips, calculate the height of each strip of pixels using a formula like the one below.

```
let strip_height = Math.ceil( image_height / num_workers );
```

When extracting the ImageData object that holds the strip of pixel data from the canvas to send to each Worker, make sure to increase the strip height by 2 to account for the unprocessed pixel border. The extra pixel data is required by the median processing filter and provides an overlap of pixels to ensure the final reassembled image is complete.

Be careful with the last strip of pixel data. Because image heights likely will not divide evenly into the number of Workers, adjust your parameters as necessary when calculating the height of the last strip of pixels to process.

In addition to the ImageData object holding a strip of pixel data, the parameters in the message object posted to a Worker should also include at least the (x, y) coordinates indicating from where the ImageData object was extracted as well as the strip height. When the main script receives a message from a Worker after processing is complete, the main script needs to know where to redraw processed pixels and one way to solve that problem is to pass these data to the Worker so that the Worker can pass them back as part of its response message data. The object you pass to a Worker may be formatted like the following.

```
{ imdata: myImageData, x: x, y: y, height: strip_height }
```

Remember that an ImageData object contains a `.data` property holding a Uint8ClmpedArray object with pixel data, as well as `.width` and `.height` properties that indicate the dimensions of the ImageData object.

Note that an outer border of pixels is not modified as part of the median filter. When the processed ImageData object is returned to your main script, draw only modified pixels back to the canvas, excluding the outer border. The 2D context object's `.putImageData(…)` method has a 7-parameter overload that allows specific pixel ranges to be transferred.

When the main script receives a message in response from the Worker, after putting the processed ImageData on the canvas, make sure to terminate the Worker using its `.terminate()` method.