Background*:*

In this lab, we will simulate various Photoshop filters by using Processing to manipulate pictures at the pixel level. You will create a new class called Filter that contains all the methods used to make the various filters. This is practicing 1-D array manipulation but using graphics instead

Exercise 1*: setup() and draw() –*

Create a `PImage` object and load it with the "`cherry.jpg`" file. Display it on a screen that is `800x600`. Create a `boolean` called `lightSwitch`. We will only change a filter when the mouse is clicked (rather than repeatedly calling a filter every time draw happens), but we DO want the filters to be activated in `draw()` (more on why later) This variable will initially be set to `false` in `setup()`. However, when the mouse is clicked, it updates to `true`. In `draw()` when the `lightSwitch` is true, we display the original image then set the `lightSwitch` back to `false`. Use the `mouseClicked()` method to adjust the `lightSwitch` variable. Additionally, create a counter variable called `count` to track the number of times the mouse has been clicked. It determines which filter method to call.

Exercise 2*: Filter class –*

Create a new tab called `Filter`. This class contains one instance variable called `filterCounter` that counts the number of times the filters have been called. In the constructor set this variable to `0`. Create a method called `filterCount()` that returns this instance variable. You will have to update the "main" tab by creating a `Filter` variable and instantiating a `Filter` object in `setup()`. All the filters in the following exercises will be methods you write in the `Filter` class, but are then called in the `draw()` method. All filters will increase the `filterCounter` by 1.

Add the following helper methods to your `Filter` class:

```
//converts an index into a (x,y) coordinate
  public PVector indexToPoint(int i)
  {
    float x = i % width;
    float y = i / width;
    return new PVector(x, y);
  }
```

```
//converts an (x,y) coordinate into an index)

public int pointToIndex(PVector p)
  {
    return (int)(p.x + width*p.y);
  }
```

Exercise 3*: dropChannel()*

Sometimes it is important to filter out one of the three channels of color (red, green, or blue). This filter will take a `String` as a parameter if this parameter is "`RED`" then the red channel is removed from the picture, if the parameter is "`GREEN`" then the

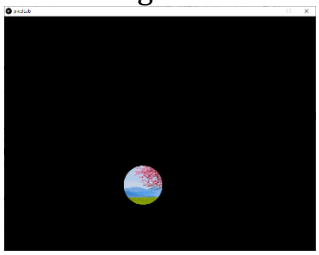
green channel is removed, if the parameter is "BLUE" then the blue channel gets removed. The following examples display the output. Update the draw method so that if the `count` is 1, then we drop the red channel by calling the `dropChannel()` method with "RED" as a parameter, if the `count` is 2 then we drop the blue channel, and if `count` is 3 then we drop the green channel. Here's an example of each filter:

Exercise 4*: solarize()*

This method looks at the `threshold` of a particular pixel to make a decision to keep the original color or invert the pixels color. `threshold` is calculated to be the sum of all three channels of a pixel. One integer parameter to determines how intense the `threshold` has to be in order to be converted to a negative pixel. If the `threshold` is below the parameter, the pixel is inverted by taking subtracting each channel's value from 255 otherwise it stays the same. The following is an example of `solarize(400);` This method activates when the counter is 4.

Exercise 5*: flashlight()*

`flashlight()` goes through the array and determines if an individual pixel is within `50` pixels of the current (`mouseX, mouseY`) coordinate. If it is less than `50` pixels then we keep the color. We turn all other pixels black. Use the helper method `indexToPoint()` to take an index in the array and convert it to a `PVector` with its corresponding `(x,y)` point on the screen. The `dist()` method in Processing would be helpful as well. You MUST create a separate if statement in draw that only checks to see if `count` is 4 and the flashlight method is called.



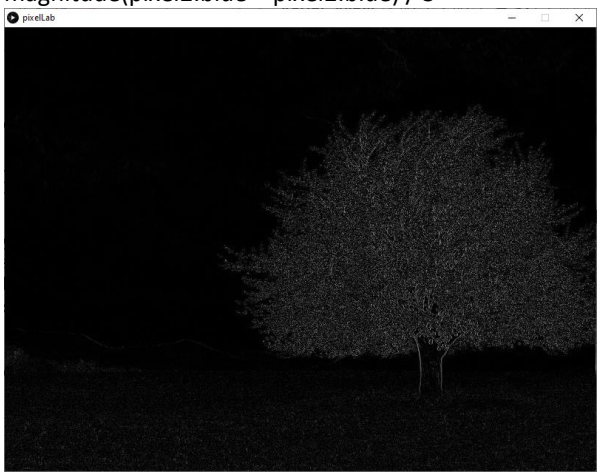
Exercise 5*: invert()*

`invert()` will take the pixel at spot `0` and switch it with the pixel at the end of the array, then it will take the second pixel and switch it with the pixel that is second from the end of the array. It will repeat this until all pixels are flipped. Watch your for loop's stopping condition for this one. Call the method when the counter is 5

Exercise 6*: edgeDetection()*

A simple way to detect edges between objects is to determine if there is a significant switch between the channels of one pixel and the pixel next to it. To do this grab a pixel and its neighbor. Find the magnitude of the difference of each channel and calculate the average of these three channels. The new pixel will have this avg as the value for each of its channels:

Avg = magnitude(pixel1.red – pixel2.red) + magnitude(pixel1.green – pixel2.green) + magnitude(pixel2.blue – pixel2.blue) / 3



Exercise 7*: glass()*

The last effect traverses the array picking a random pixel within that is within a +/- 5 of its position on the screen. Once again you'll need to use the `indexToPoint()`

method to translate an individual index to a (x,y) coordinate. Next, you need to calculate a random point whose `(x +/- 5, y +/- 5)` from the original. Finally, you'll have to turn this new point back into a second index. If this is a valid index, swap the pixels. The picture is on the next page so you can see the effect.