Exercises

For all sketches, be sure to **include a comment with the names of your group at the beginning of the sketch.**

Provided images –

- If you are working on your own computer, use the link on the schedule page – right-click on the filename and choose "Save Link As…" to save images on your computer.
- If you are working in Linux (either the virtual desktop or on an actual machine), you can find the same images in **/classes/cs120/images**.

Don't forget to also add the image(s) to each sketch you create with Sketch → Add File…

1. Create a new sketch called **sketch_241125a** which displays a version of the original image where the green and blue components of the color are swapped. Write a filter function (with a parameter for the source image) for the swap colors operation.

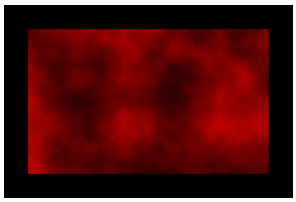
2. Create a sketch called **sketch_241125b** which displays an image with a black-to-red color gradient. Write a generator function (with parameters for the size of the image to generate) for the gradient. Scale the gradient so you get the full black to red range for any size image.

   Hint: to figure out the red color value, use `map` to scale the current column coordinate to a color value 0-255. Look `map()` up in the Processing API: `https://processing.org/reference/map_.html`)

If you have time –

- Create a sketch **sketch_241125c** which displays a flipped upside down version of an image. Write a filter function (with a parameter for the source image) for the flip operation.

  Hint: the pixel at (row,col) in the flipped image comes from what position (orig_row,orig_col) in the original image? Figure out the pixel array location for (orig_row,orig_col) for retrieving a color from the original image and for (row,col) for setting the color in the flipped image.



- Create a sketch called **sketch_241125d** which displays an image where the red component of the pixel's color comes from Perlin noise. (This sort of image is known as a 2D texture.) Write a generator function with parameters for the size of the image and the smoothness of the texture.

  Use the two-parameter version of `noise` for this – you need one noise value which varies in both the horizontal and vertical directions rather than two separate noise values. See the Processing API for more on `noise`: `https://processing.org/reference/noise_.html`

  Hint: Make the noise parameters into loop variables – one with the row loop and one with the col loop. (Since the amount a noise parameter is updated by is typically very small, the noise parameters will need to be `floats` – which means that while they can be updated in the update part of the for loop, they will need to be declared and initialized immediately before each loop rather than in the initialization part.) The smoothness is the amount to update the noise parameters by.

  Historical note: Generating textures like this is what Perlin noise was invented for.

At the End of Class

- Copy the entire directory for each sketch (not only the .pde file) into your handin directory (**/classes/cs120/handin/*username***). You only need to hand in one copy for the group.

Example – Defining a Filter Function

```
// brighten - make each color component brighter (closer to white)
//  src - image to brighten
// amt - amount to brighten
PImage brighten ( PImage src, int amt ) {
  // create a new blank image
  PImage dst = createImage(src.width, src.height, RGB);

  // load src pixels
  src.loadPixels();

  // compute pixel colors
  for ( int row = 0; row < dst.height; row = row+1 ) {
    for ( int col = 0; col < dst.width; col = col+1 ) {
      // compute index in pixel array for this p
      int loc = row*dst.width+col;
      // compute r, g, b
      float r = red(src.pixels[loc])+amt;
      float g = green(src.pixels[loc])+amt;
      float b = blue(src.pixels[loc])+amt;
      // set the pixel's color
      dst.pixels[loc] = color(r, g, b);
    }
  }

  // save the pixel colors
  dst.updatePixels();

  return dst;
}
```

PImage instead of void

source image as parameter

modify the blue outline parts (and optionally add parameters) to customize; the rest is a set template

return the generated image

Image Filters

"filter" refers to a process for modifying an image's appearance in some way – generate a new image whose pixels colors are derived from those in the source image

- create a new blank image which is the same size as the source
  `dst = createImage(src.width,src.height,RGB);`
  - use ARGB instead of RGB if you want to include transparency
- make the pixels from the source image available for access
  `src.loadPixels();`
- compute pixel colors for the destination image
  - a common pattern is to compute a color for every pixel based on the corresponding pixel in the source image

go through all of the rows and columns in the image

```
for ( int row = 0 ; row < dst.height ; row = row+1 ) {
  for ( int col = 0 ; col < dst.width ; col = col+1 ) {
    // location in pixels array corresponding to (row,col)
    int loc = row*dst.width+col;
    // compute r, g, b
    int r = …red(src.pixels[loc])…;
    int g = …green(src.pixels[loc])…;
    int b = …blue(src.pixels[loc])…;
    // set the pixel color
    dst.pixels[loc] = color(r,g,b);
  }
}
```

- save the pixel colors
  `dst.updatePixels();`

Example – Defining a Generator Function

```
// create a random image
//  w, h - dimensions of image to generate
PImage generateRandom ( int w, int h ) {

  // create a new blank image
  PImage dst = createImage(w,h,RGB);

  // compute pixel colors
  for ( int row = 0 ; row < dst.height ; row = row+1 ) {
    for ( int col = 0 ; col < dst.width ; col = col+1 ) {

      // location in the pixels array correspon
      int loc = row*dst.width+col;

      // compute r, g, b
      float r = random(0,255);
      float g = random(0,255);
      float b = random(0,255);

      // set the pixel's color
      dst.pixels[loc] = color(r,g,b);
    }
  }

  // save the pixel colors
  dst.updatePixels();

  // return the generated image
  return dst;
}
```

PImage instead of void

width, height of image to generate as parameters

modify the blue outline parts (and optionally add parameters) to customize; the rest is a set template

return the generated image

Generating Images

- create a new blank image
  `img = createImage(w,h,RGB);`
  - use ARGB instead of RGB if you want to include transparency
- set pixel colors
  - a common pattern is to compute a color for every pixel

go through all of the rows and columns in the image

```
for ( int row = 0 ; row < img.height ; row = row+1 ) {
  for ( int col = 0 ; col < img.width ; col = col+1 ) {
    // location in pixels array corresponding to (row,col)
    int loc = row*img.width+col;
    // compute r, g, b
    int r = …;
    int g = …;
    int b = …;
    // set the pixel color
    img.pixels[loc] = color(r,g,b);
  }
}
```

- save the pixel colors
  `img.updatePixels();`

Images Recap

- declare a variable to hold an image

  `PImage img;`
- load an image (image must also be added to sketch with Sketch->Add File…)

  `img = loadImage("myimage.jpg");`
- create a new image

  `img = createImage(w,h,RGB);`
  - use ARGB instead of RGB to control transparency
- displaying an image

  `image(img,x,y);`
  `image(img,x,y,w,h);`

  (use `imageMode(CORNER)` or `imageMode(CENTER)` to specify whether (x,y) is the upper left corner or center of the image)
- size of an image

  `img.width`
  `img.height`
- loading pixels (in order to access)

  `img.loadPixels();`
- accessing the color of a pixel (once loaded)

  `img.pixels[loc]`

  (loc = row*img.width+col)
- accessing color components for a pixel (once loaded)

  `red(img.pixels[loc])`
  `green(img.pixels[loc])`
  `blue(img.pixels[loc])`

  (loc = row*img.width+col)
- saving pixels (after modification)

  `img.updatePixels();`

Example – Calling a Filter Function

```
PImage img, filtered;

void setup () {
  size(840,640);

  img = loadImage("pelican.jpg");
  filtered = brighten(img,60);
}

void draw () {
  background(0);

  image(img,0,0,420,640);
  image(filtered,width/2,0,420,640);
}
```

```
PImage img;
float brightamt;   // amount to brighten the image
```

if the filter always does the same thing, or if it does different things but you want the same image in every frame –

declare variables for the source and filtered images

call the filter function to initialize the filtered variable in `setup()`

draw the filtered image in `draw()`

Example – Calling a Generator Function

```
PImage img;

void setup () {
  size(400,400);

  img = generateRandom(width,height);
}

void draw () {
  background(0);

  image(img,0,0);
}
```

if the generator always does the same thing, or if it does different things but you want the same image in every frame –

declare a variable for the image

call the generator function to initialize the image variable in `setup()`

draw the image in `draw()`