IMAGE PROCESSING

OVERVIEW

During this section of the workshop, participants will have an opportunity to learn the fundamentals of image processing. The concepts explored in this section have familiar applications such as Instagram and Photoshop.

Coding is performed utilizing the accessible software called Processing, which allows the participants to learn how to code within the context of the visual arts. Processing is free and open source and can be downloaded for GNU/Linux, Mac OS X, and Windows.

The four coding exercises in this section are:

- Build your own Instagram-like filter

- Flip an image vertically

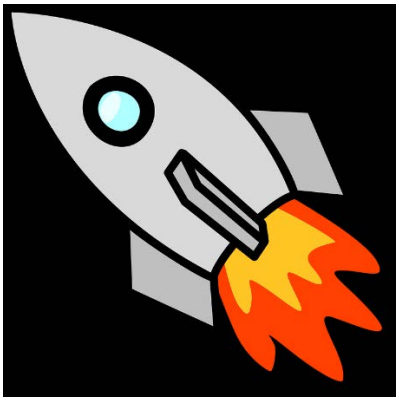
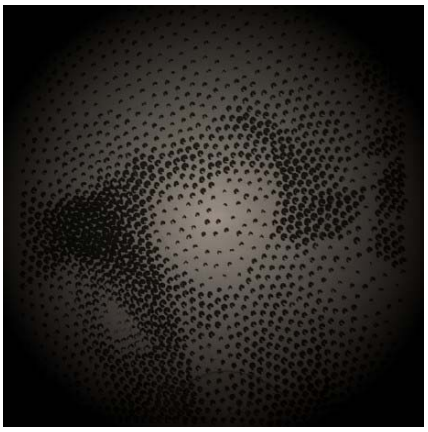


Illustration via OpenClipArt. This image is in the public domain.

- Convert an image to a single color

- Add a vignette



INTENDED OUTCOMES

- Expose participants to image processing concepts that have familiar, real-world applications
- Demystify computer science concepts and make them more accessible through fun, simple coding exercises
- Introduce the basics of programming such as if statements and variable declaration
- Encourage critical thinking and persistence through problem solving and debugging
- Foster collaboration through the creation of supportive teams
- Challenge negative stereotypes about working in technology

MATERIALS

- Laptop
- Processing software (https://processing.org/)

FOUR PRIMARY COMPONENTS OF A PROCESSING PROGRAM

The four primary components of a Processing program are variable declaration, setup(), draw(), and mouseClicked().

VARIABLE DECLARATION

Processing has the ability to declare various variable types. In these exercises, the variable types used are PImage, boolean, and float.

PImage is a class for storing images. This class contains fields for width and height, as well as pixels[], which is an array of values for every pixel in the image. The methods (functions that “belong to” a particular class) utilized in these exercises include loadPixels, updatePixels, get, set, and mask.

The declaration boolean is a datatype for storing Boolean values of true or false. It is common to use Booleans with control statements such as if or while.

Other datatypes that you may encounter when utilizing Processing for these exercises and other applications include integers (int), 32-bit floating point numbers (float), 64-bit doubles (double), and character strings (char).

Additional information on PImage, boolean, and other datatypes can be found in the Processing documentation: https://processing.org/reference/

SETUP() FUNCTION

The setup() function is run once when a program starts. This function can be used to initialize the environment properties such as screen size and the image to load.

Notes on Usage:

- There can only be one call to setup() for each program.
- Variables declared within setup() are not accessible within other functions.

DRAW() FUNCTION

Called directly after setup(), the draw() function continuously executes the lines of code contained inside until the program is stopped or noLoop() is called. Its behavior can be controlled with noLoop(), redraw(), and loop(). If noLoop() is called, redraw() will cause the code within draw() to run once. Calling loop() will cause the code in draw() to resume continuous execution.

Notes on Usage:

- There can only be one draw() function for each sketch.
- draw() must exist if you want the code to run continuously or to process events such as mouse clicks.
- The number of times draw() executes in each second can be controlled with the frameRate() function.
- It is recommended to call background() near the beginning of the draw() loop to clear the contents of the window. Pixels drawn to the window are cumulative, so omitting background() may have unintended results.

MOUSECLICKED() FUNCTION

The mouseClicked() function processes the event of a mouse click.

Notes on Usage:

- This will work only when a program has draw(). Without draw(), the code is only run once, and then any additional events will not cause anything to happen.

OTHER IMPORTANT ELEMENTS

Other important elements of Processing code are comments and statements. Statements are elements that make up the program. These statements end in a ";" (semi-colon). The semi-colon is the statement terminator.

Comments, which are explanatory notes that are embedded in the code, are used to document the code and provide information as to the purpose and function of a program. Comments are ignored by the compiler. There are two ways in which one can create comments. A single line comment is designated as follows:

```
// Here is my single line comment
```

A multi-line comment is created as follows:

```
/* Here is my
multi-line
comment*/
```

For the multi-line comment, everything between the "/*" and "*/" is ignored by the compiler.

IMPLEMENTATION AND SUGGESTIONS

To accommodate varying levels of experience and also to get the most of the allotted time (2 hours), the coding exercises are set up in a fill-in-the-blank style with accompanying informative descriptions. The four coding exercises in this section are:

- Build your own Instagram-like filter
- Flip an image vertically
- Convert an image to a single color
- Add a vignette

It is recommended that the first exercise, create an Instagram filter, is done together before breaking up into groups. During this time, the four primary components of a Processing program can be introduced. In this way, the playing field is more leveled to accommodate individuals with varying levels of coding experience.

After working through the first exercise and discussing the primary components of a Processing program, it is recommended to split up into small teams. Girls Who Build had 1 volunteer per group of 4 girls. This was done to encourage collaboration and sharing of ideas between girls, while allowing volunteers to be more attentive and involved.

If participants finish early, encourage participants to help other groups or have them delve deeper into the mechanics of the code by doing the bonus exercises.

OTHER THOUGHTS

- During this workshop, participants generally had their own computer. That said, 1 laptop for 1 individual is not necessary. Coding involves logical thinking and problem solving, so working in pairs is not an issue. Furthermore, working in pairs will allow participants to take on different roles and develop effective communication and collaboration skills.
- While there are pictures that accompany the exercises, it was found that the girls enjoyed selecting their own pictures to process. For instance, the participants can opt to use the pictures they take with the Raspberry Pi cameras.
- Some advice for volunteers and instructors: don't underestimate the participants! Let the participants drive. Be present to answer questions, but do not feel as though you need to direct everything. Even those without much coding experience can surprise you!
- Although it was not utilized during the Girls Who Build workshop, there are free, online resouces that allow several users to edit a file using different computers in real-time. Examples include Collabedit and TitanPad. This may further increase collaboration but has yet to be tested.

SUPPORTING PARTICIPANTS WITH DIFFERENT LEVELS OF CODING EXPERIENCE

The fill-in-the-blank style of the four coding exercises was utilized to meet the needs of participants with diverse coding experience. To further accommodate participants with little-to-no experience:

- Introduce computer science concepts with a fun, interactive demo by making a peanut butter and jelly sandwich. Link: http://static.zerorobotics.mit.edu/docs/team-activities/ProgrammingPeanutButterAndJelly.pdf
- Present coding concepts in an informative presentation.
- Walk-through first example together before splitting into teams
- Randomize composition of groups or have more experienced participants work with those who may have less experience. The less experienced may be inclined to just sit back and take a passive role. Take steps to foster collaboration.

BONUS EXERCISES

Time permitting, additional activities within the context of the four coding exercises can be performed.

BUILD YOUR OWN INSTAGRAM-LIKE FILTER

BONUS: MODIFY THE RGB AND ALPHA VALUES

Change RGB and alpha values of the filter.
RGB values of common colors: http://www.workwithcolor.com/color-chart-full-01.htm
Difficulty: Low

BONUS: INVESTIGATE BUILT-IN PROCESSING FILTERS

Within the PImage Processing class exists various built-in filters, which alter the picture according to the mode selected. The modes are:

- Threshold
- Gray
- Opaque
- Invert
- Posterize
- Blur
- Erode
- Dilate

Experiment with the filters and see what happens!

CODE

```
//SET UP
PImage img; //Declare variable of type PImage

void setup() {
  //LOAD IMAGE
  size(200,200); //Change size of image.
  img = loadImage("IMG_3294.JPG"); //Image in our library

   //INVERT
  img.filter(INVERT);
}

// CREATE IMAGE
void draw(){
  image(img,0,0,width,height); //Resize the image image(file name, x_origin, y_orgin, size x, size
y)

  // CREATE FILTER
```

```
  fill(255,255,0,60); //Fill shape with semi-transparent filter over image (R value,G value,B
value, alpha/transparency)
  noStroke(); //no border
  rect(0,0,width,height); //Define shape of filter
}
```

RESULT

Additional information on the built-in filters can be found in the Processing documentation:
https://processing.org/reference/PImage_filter_.html
Difficulty: Low

FLIP AN IMAGE VERTICALLY

BONUS: FLIP AN IMAGE HORIZONTALLY

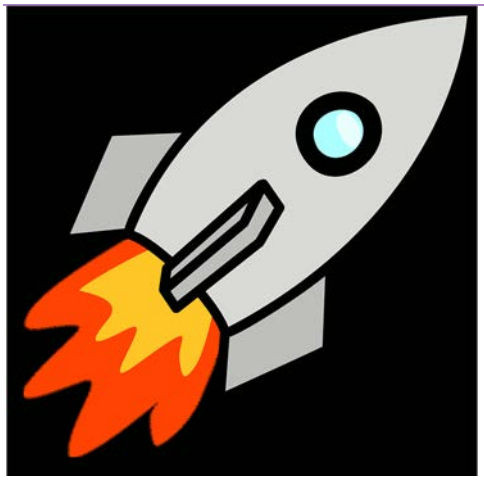
The original exercise flips an image vertically. Have the participants modify the code to flip the image horizontally.

CODE

```
//SET UP
PImage img, img_flip;
boolean flip;

//SETUP
void setup() {
  //LOAD IMAGE AND INITIALIZE IMAGE FLIP
  size(750, 750);
  img = loadImage("spaceship.png");
  img_flip = createImage(750, 750, RGB);

  //LOAD PIXEL DATA
  /*  Loads the pixel data for the image into its pixels[] array. This function must always be
  called before reading from or writing to pixels. */
  img.loadPixels();
  img_flip.loadPixels();

  //CREATE THE FLIPPED IMAGE
  for (int i = 0; i < img.width; i++) { //i++ is iterating through the pixels horizontally
    for (int j = 0; j < img.height; j++) {
      img_flip.set(img_flip.width-1-i, j, img.get(i, j)); //Reads the color of the specified pixel
    }
  }
  img_flip.updatePixels();
  flip = false;
```

```
}

//DISPLAY IMAGE
void draw() {
  background(0);
  if (flip) { // FLIP
    image(img_flip,0,0);
  }
  else { // DO NOT FLIP
    image(img,0,0);
  }
}

//CONDITION FOR MOUSE CLICK (USER INPUT)
void mouseClicked() {
  flip = !flip;
}
```

RESULT



Difficulty: Medium

CONVERT AN IMAGE TO A SINGLE COLOR

BONUS: MANUALLY CONVERT TO GRAY-SCALE

This is an activity that is a great addition to the “Convert to a Single Color” exercise. Manually changing a color photo to grayscale involves the manipulation of the color of all pixels and knowledge of luminance.

The most common strategy to convert a color image to grayscale is to match the luminance (light intensity) of the grayscale image to the luminance of the original color image. To convert a color from an RGB colorspace to a grayscale representation, a weighted sum of the three linear intensity values is calculated. According to CIE 1931, the linear luminance Y is given by

$$Y = 0.2126R + 0.7152G + 0.0722B,$$

where R, G, and B correspond to the values of red, green, and blue, respectively. Note that green light contributes the most intensity perceived by humans, but blue contributes the least. After the luminance Y is calculated, one can convert an image to grayscale by setting the RGB values to YYY.

CODE

```
// INITIALIZATION
PImage img, imgGray; // Initialize images - one gray, one color
```

```
boolean color_image; // Initialize Boolean for mouse click
float Y; // Initialize float for storage of luminance value

//SET UP
void setup() {
  // LOAD COLOR IMAGE
  size(512,384);
  img = loadImage("boats.JPG");

  // INITIALIZE GRAYSCALE IMAGE
  imgGray =  createImage(512,384,RGB);
  imgGray = img.get();

  // MAKE GRAYSCALE
  color_image = true;
  for (int i = 0; i < imgGray.width; i++) {
    for (int j = 0; j < imgGray.height; j++) {
      color c = imgGray.get(i,j);
      Y = red(c)*0.2126+green(c)*0.7152+blue(c)*0.0722; // Calculate luminance
      imgGray.set(i, j, color(Y,Y,Y)); // Apply luminance
    }
  }
}


// CREATE IMAGE
void draw(){
  if (color_image){
    background(0);
    image(img,0,0,width,height); //Resize the image, image(file name, x_origin, y_orgin, size x,
size y)
  }
  else{
    background(0);
    image(imgGray,0,0,width,height); //Resize the image, image(file name, x_origin, y_orgin, size
x, size y)
  }
}


// CHANGE TO COLOR WHEN MOUSE IS CLICKED
void mouseClicked() {
  color_image = !color_image;
}
```

RESULT

Difficulty: Medium

ADD A VIGNETTE

BONUS: VARY THE SIZE AND COLOR OF THE VIGNETTE

Make the border smaller or larger. Vary the RGB values of the vignette.
Difficulty: Low

KITCHEN-SINK

Lastly, participants can combine elements from all of the original coding exercises into one. For example, request that the students write code that adds a vignette to a loaded image. When clicked, the vignette will remain, but the image will flip vertically, the image will be converted to a single color, and an Instagram-like overlay will be added.

CODE

```
PImage img, img_single_color, img_flip, img_click, msk;
boolean mouse_event;
float maxDist;

//SET UP
void setup() {
  //LOAD IMAGE
  size(750, 500);
  img = loadImage("flowers.jpg");
  colorMode(HSB);

  //CREATE VIGNETTE
  msk = createImage(750, 500, HSB);
  msk.loadPixels();
  maxDist = dist(0, 0, width/2, height/2);
  for (int i = 0; i < msk.width; i++) {
    for (int j = 0; j < msk.height; j++) {
        msk.set(i, j, color(hue(get(i,j)), saturation(get(i,j)), maxDist-dist(i, j, width/2,
height/2)));
    }
  }
  msk.updatePixels();
  img.mask(msk);

  //INITIALIZE BOOLEANS
```

```
  mouse_event = false;
}

// DISPLAY IMAGE
void draw() {
  if (mouse_event) {
  background(0);
  image(img_click, 0, 0);
  }
  else {
  background(0);
  image(img, 0, 0);
  }
}
// CHANGE TO SINGLE COLOR WHEN MOUSE IS CLICKED AND FLIP IMAGE
void mouseClicked() {
  mouse_event = !mouse_event;
  if (mouse_event) {
  img_single_color = img;
  img_single_color.loadPixels();
  float h = hue(get(mouseX, mouseY));
  for (int i = 0; i < img_single_color.width; i++) {
    for (int j = 0; j < img_single_color.height; j++) {
      color c = img_single_color.get(i,j);
      float ph = hue(c);
      if (abs(ph - h) > 10.) {
        img_single_color.set(i, j, color(hue(c), 0, brightness(c)));
      }
    }
  }
  img_single_color.updatePixels();

  // CREATE FLIP IMAGE
  img_flip = createImage(750, 500, HSB);
  img_flip.loadPixels();
  for (int i = 0; i < img.width; i++) { //i++ is iterating through the pixels horizontally
    for (int j = 0; j < img.height; j++) {
      img_flip.set(i, img_flip.height-1-j, img.get(i, j));//Reads the color of the specified pixel
    }
  }
  img_flip.updatePixels();

  //ADD VIGNETTE
  img_click = img_flip;
  img_click.mask(msk);
  }
  else {
    size(750, 500);
    img = loadImage("flowers.jpg");

    //ADD VIGNETTE
    img.mask(msk);
  }
}
```

RESULT

Difficulty: High

Resource: Girls Who Build Cameras

Kristen Railey

Resource: Girls Who Build Cameras

Kristen Railey, Bob Schulein, Olivia Glennon, Leslie Watkins, Alex Lorman, Carol Carveth, and Sara James