WEEK 4

IMAGE PROCESSING

copy()

- copy(sx, sy, sw, sh, dx, dy, dw, dh)

  Copies a region of pixels from the display window to another area of the display window

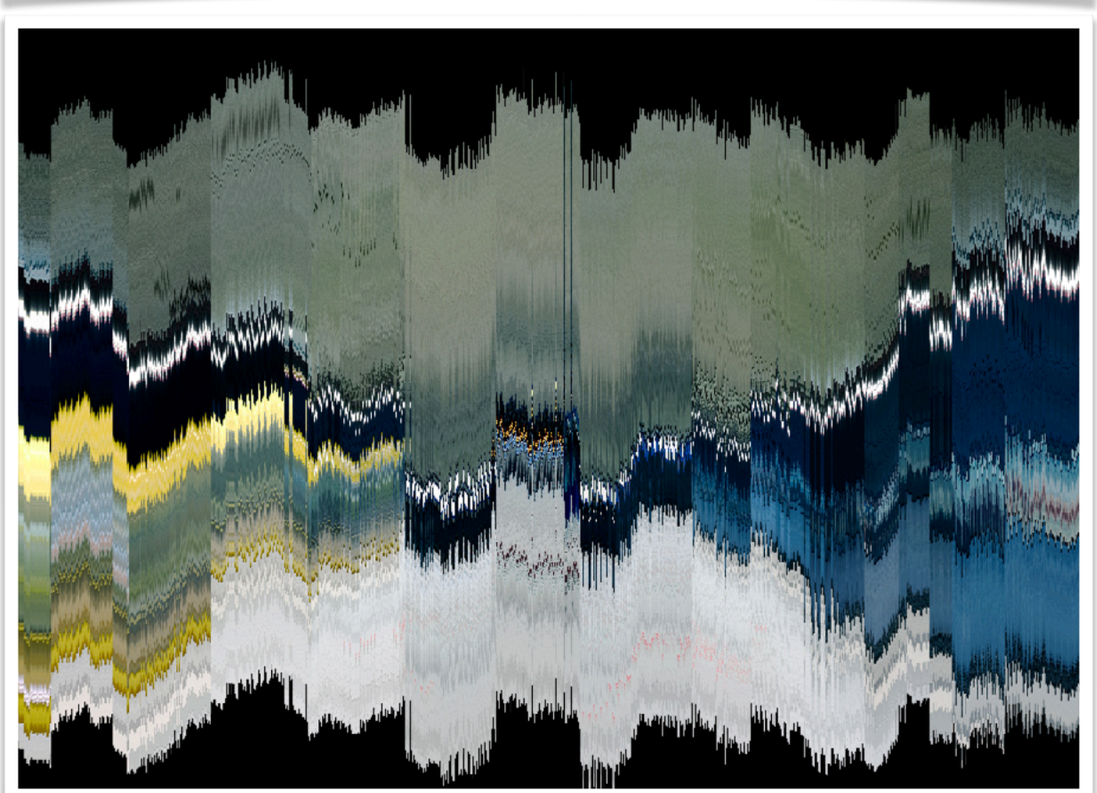
- copy(src, sx, sy, sw, sh, dx, dy, dw, dh)

  copies a region of pixels from an image into the display window

- if the source and destination regions aren't the same size, it will automatically resize the source pixels to fit the specified target region.

EXAMPLE 1

```
PImage img;

void setup() {
  size(1024, 780);
  background(255);
  img = loadImage("station.png");
  image(img, 0, 100);
}

void draw() {
  int x1 = (int) random(0, width);
  int y1 = 0;
  int x2 = round(x1 + random(-7, 7));
  int y2 = round(random(-5, 5));
  int w = (int) random(35, 50);
  int h = height;
  copy(x1, y1, w, h, x2, y2, w, h);
}

void keyReleased() {
  background(255);
  image(img, 0, 100);
}
```



EXERCISE 1

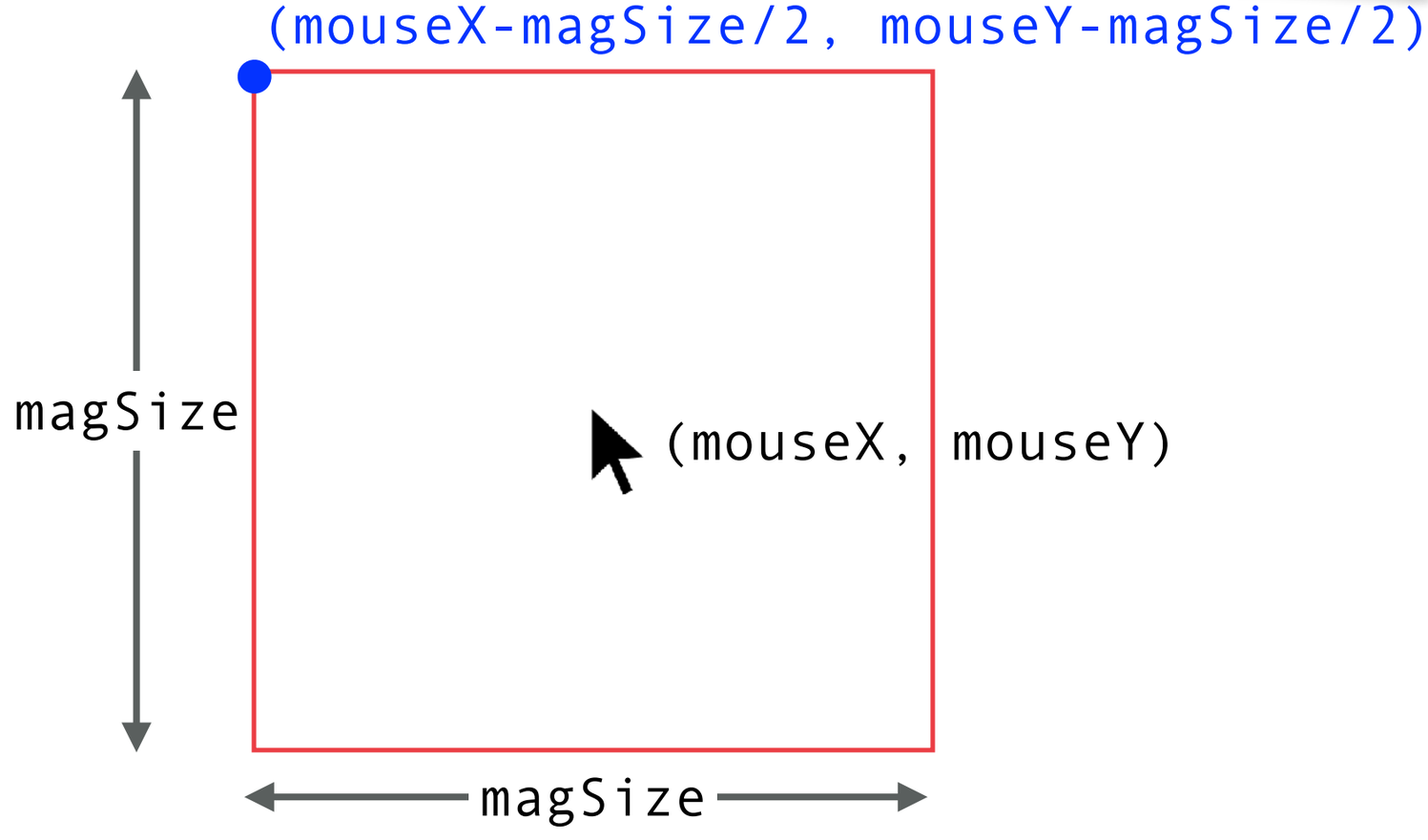
- Base on the given template, use copy() or get() to create a magnifier that will zoom a region selected by the mouse

SUGGESTED SOLUTION TO EXERCISE 1

```
copy(img, mouseX-magSize/2, mouseY-magSize/2, magSize, magSize,          source
     0, height-magSize*magScale, magSize*magScale, magSize*magScale);   destination
```

or

```
//Pick up the subimage at the mouse location
PImage zoom = get(mouseX-magSize/2, mouseY-magSize/2, magSize, magSize);

//Show the subimage at the left-bottom corner of the display window
image(zoom, 0, height-magSize*magScale, magSize*magScale, magSize*magScale);
```



PIXEL-LEVEL IMAGE PROCESSING

- Visit every pixel in the image or display window

```
for (int x = 0; x < img.width; x++ ) {
    for (int y = 0; y < img.height; y++ ) {
        ...
    }
}
```

← Two/nested for-loop structure

- Then change the pixel's appearance e.g. color or position
- Two approaches to get pixel color at (x, y):
  1. Modify the pixel array

     ```
     loadPixels()
     pixels[x+y*img.width]
     updatePixels()
     ```

  2. Use get() and set() (or other functions for rendering shapes and color)

     ```
     c1 = get(x, y);
     set(x, y, c2);
     ```

LOAD AND UPDATE PIXELS

- Always call loadPixels() before reading from or writing to pixels[] (corresponding to a snapshot of the current display window)

- updatePixels() updates the display window with the data in the pixels[] array

```
int halfImage = width*height/2;
PImage myImage = loadImage("apples.jpg");
image(myImage, 0, 0);

loadPixels();
for (int i = 0; i < halfImage; i++) {
  pixels[i+halfImage] = pixels[i];
}
updatePixels();
```

LOCATING PIXELS

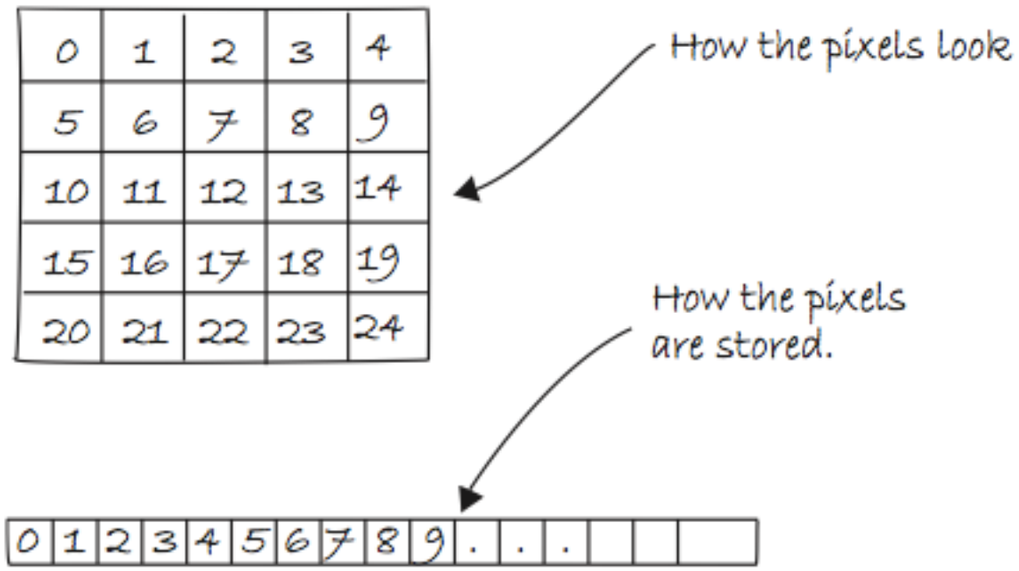


fig. 15.5

Converting (x, y) into an index in pixels[] array:

```
pixel_index = x + y * width
```

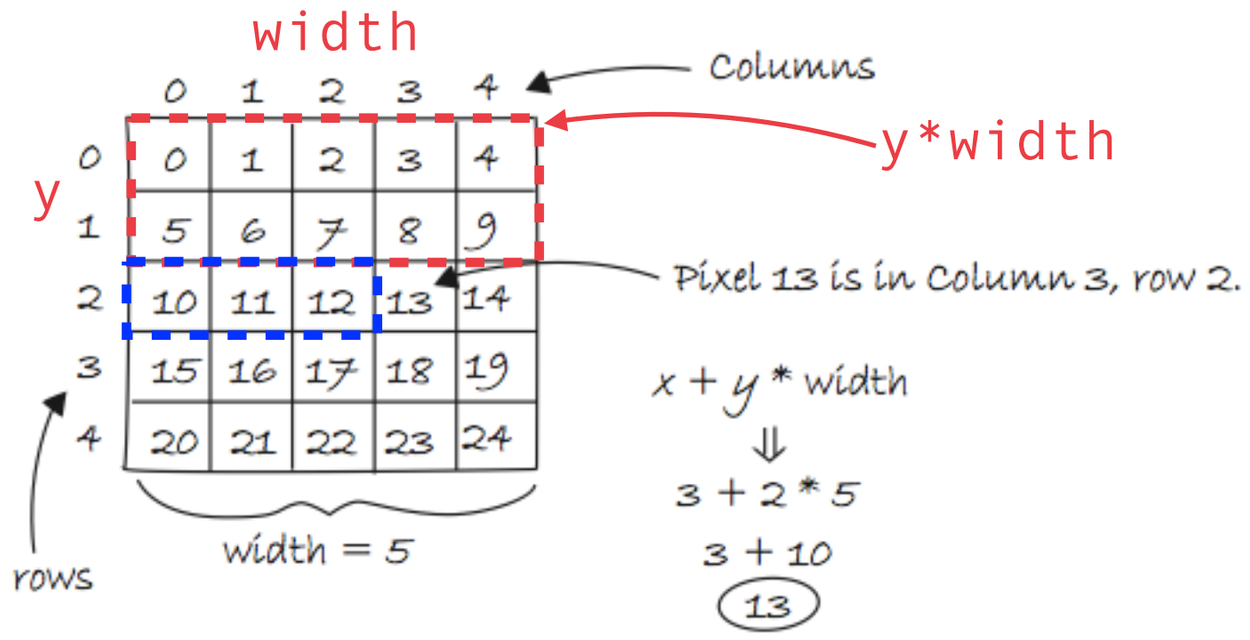


fig. 15.7

Load and Update Pixels of PImage

- PImage has its own loadPixels(), updatePixels(), and pixels[]

```
PImage myImage;
int halfImage;

void setup() {
  size(100, 100);
  halfImage = width * height/2;
  myImage = loadImage("apples.jpg");
  myImage.loadPixels();
  for (int i = 0; i < halfImage; i++) {
    myImage.pixels[i+halfImage] = myImage.pixels[i];
  }
  myImage.updatePixels();
}

void draw() {
  image(myImage, 0, 0);
}
```

The image pixels are changed permanently!

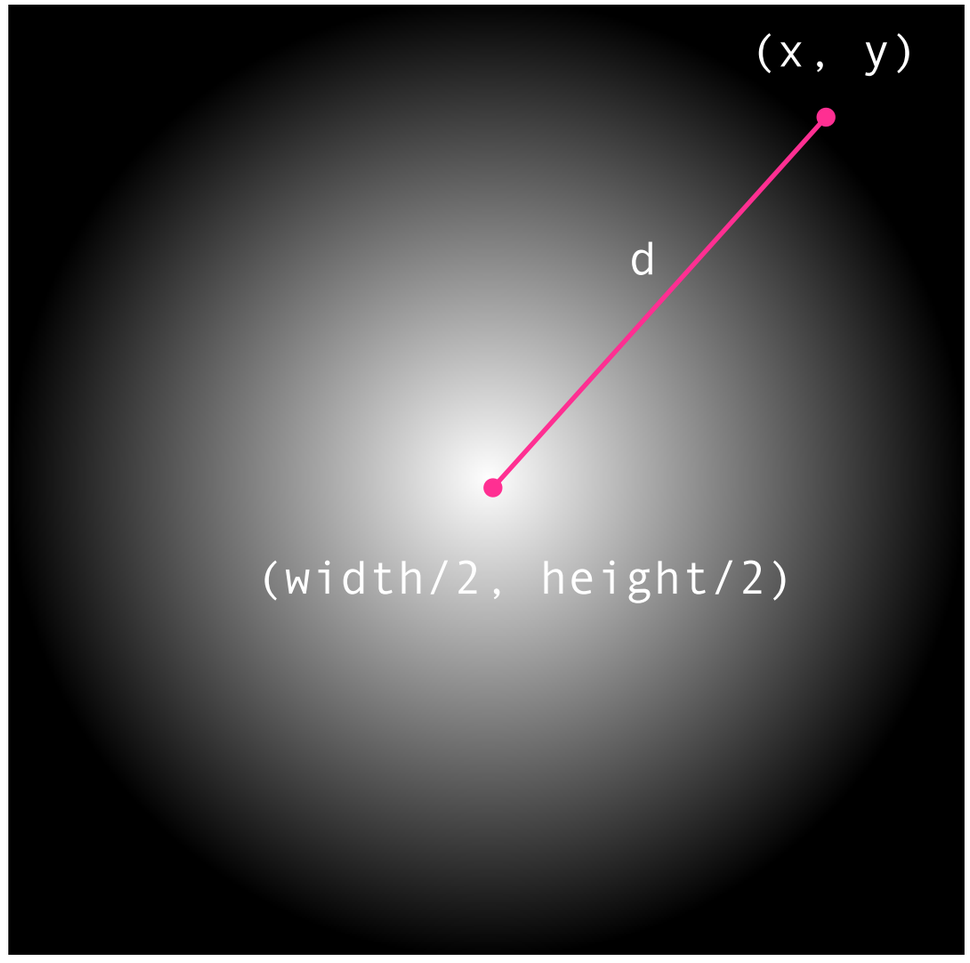
pixels[] vs img.pixels[]

In this example, img serves as a buffer only.

```
loadPixels();
img.loadPixels();
// We must also call loadPixels() on the PImage since we are going to read its pixels.
for (int x = 0; x < img.width; x++ ) {
  for (int y = 0; y < img.height; y++ ) {
    // Calculate the 1D pixel location
    int loc = x + y*img.width;
    // Get the R,G,B values from image
    float r = red  (img.pixels[loc]);
    float g = green(img.pixels[loc]);
    float b = blue (img.pixels[loc]);
    // We calculate a multiplier ranging from 0.0 to 8.0 based on mouseX position.
    // That multiplier changes the RGB value of each pixel.
    float adjustBrightness = map(mouseX, 0, width, 0, 8);
    r *= adjustBrightness;
    g *= adjustBrightness;
    b *= adjustBrightness;
    // The RGB values are constrained between 0 and 255 before being set as a new color.
    r = constrain(r, 0, 255);
    g = constrain(g, 0, 255);
    b = constrain(b, 0, 255);
    // Make a new color and set pixel in the window
    color c = color(r, g, b);
    pixels[loc] = c;
  }
}
updatePixels();
```

EXAMPLE 2 RADIAL GRADIENT

```
size(600, 600);
for (int x = 0; x < width; x++) {
  for (int y = 0; y < height; y++ ) {
    float d = dist(x, y, width/2, height/2);
    float maxdist = 300;
    float b = 255*(maxdist-d)/maxdist;
    // equivalent to
    //float b = map(d, 0, maxdist, 255, 0);
    color c = color(b);
    set(x, y, c);
  }
}
```



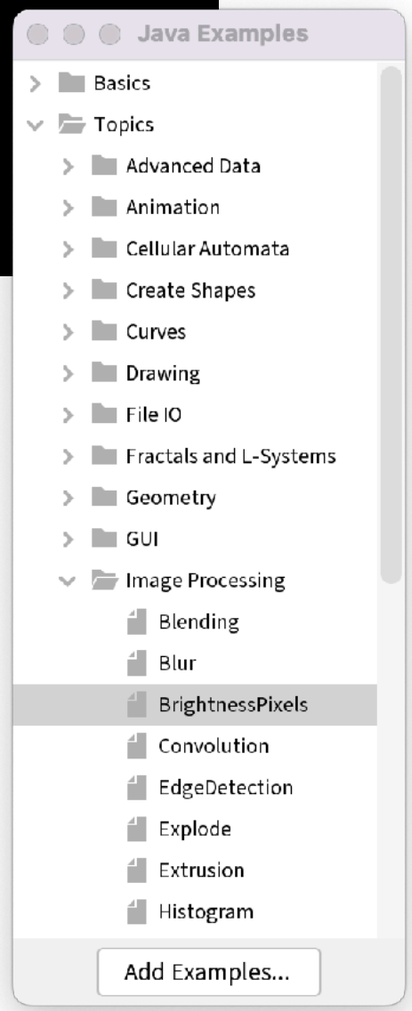
EXAMPLES/TOPICS/IMAGE PROCESSING/BRIGHTNESS PIXELS

```
PImage img;

void setup() {
  size(640, 360);
  frameRate(30);
  img = loadImage("moon-wide.jpg");
  img.loadPixels();
  // Only need to load the pixels[] array once, because we're only
  // manipulating pixels[] inside draw(), not drawing shapes.
  loadPixels();
}

void draw() {
  for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++ ) {
      // Calculate the 1D location from a 2D grid
      int loc = x + y*img.width;
      // Get the R,G,B values from image
      float r,g,b;
      r = red (img.pixels[loc]);
      //g = green (img.pixels[loc]);
      //b = blue (img.pixels[loc]);
      // Calculate an amount to change brightness based on proximity to the mouse
      float maxdist = 50;//dist(0,0,width,height);
      float d = dist(x, y, mouseX, mouseY);
      float adjustbrightness = 255*(maxdist-d)/maxdist;
      r += adjustbrightness;
      //g += adjustbrightness;
      //b += adjustbrightness;
      // Constrain RGB to make sure they are within 0-255 color range
      r = constrain(r, 0, 255);
      //g = constrain(g, 0, 255);
      //b = constrain(b, 0, 255);
      // Make a new color and set pixel in the window
      //color c = color(r, g, b);
      color c = color(r);
      pixels[y*width + x] = c;
    }
  }
  updatePixels();
}
```

loads a snapshot of the current display window into the pixels[] array

sets current pixel to color c
can be replaced with
set(x, y, c);

updates the display window with the data in the pixels[] array

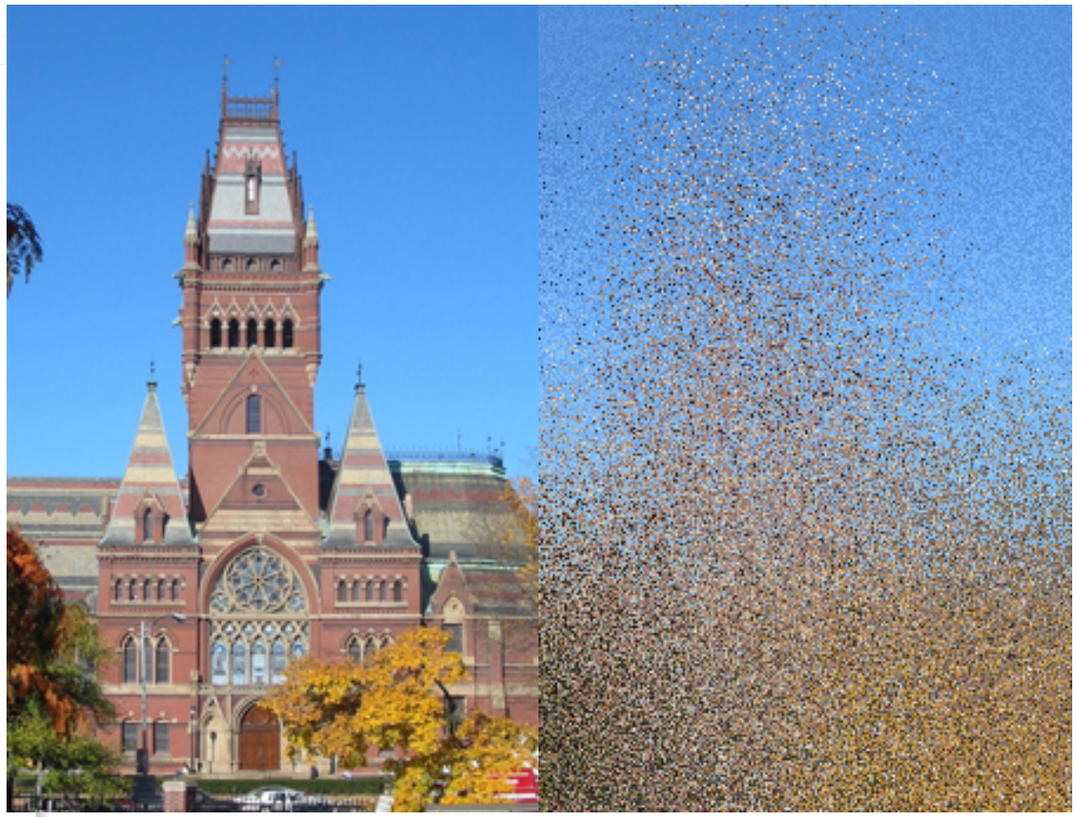


EXAMPLE 3 DISPLACEMENT OPERATION

```
// Based on Algorithms for Visual Design
// Using the Processing Language P.114-115

PImage img = loadImage("memorial.jpg");
size(226, 342);
int step = 20;
noStroke();
rectMode(CENTER);
for (int x=2; x<width-2; x++) //for all rows
  for (int y=2; y<height-2; y++) { //for all columns
    color c = img.get(x, y);
    int xx = x+int(random(-step, step));
    int yy = y+int(random(-step, step));
    //set(xx, yy, c); //color the pixel
    fill(c);
    rect(xx-2, yy-2, 5, 5);
  }
```

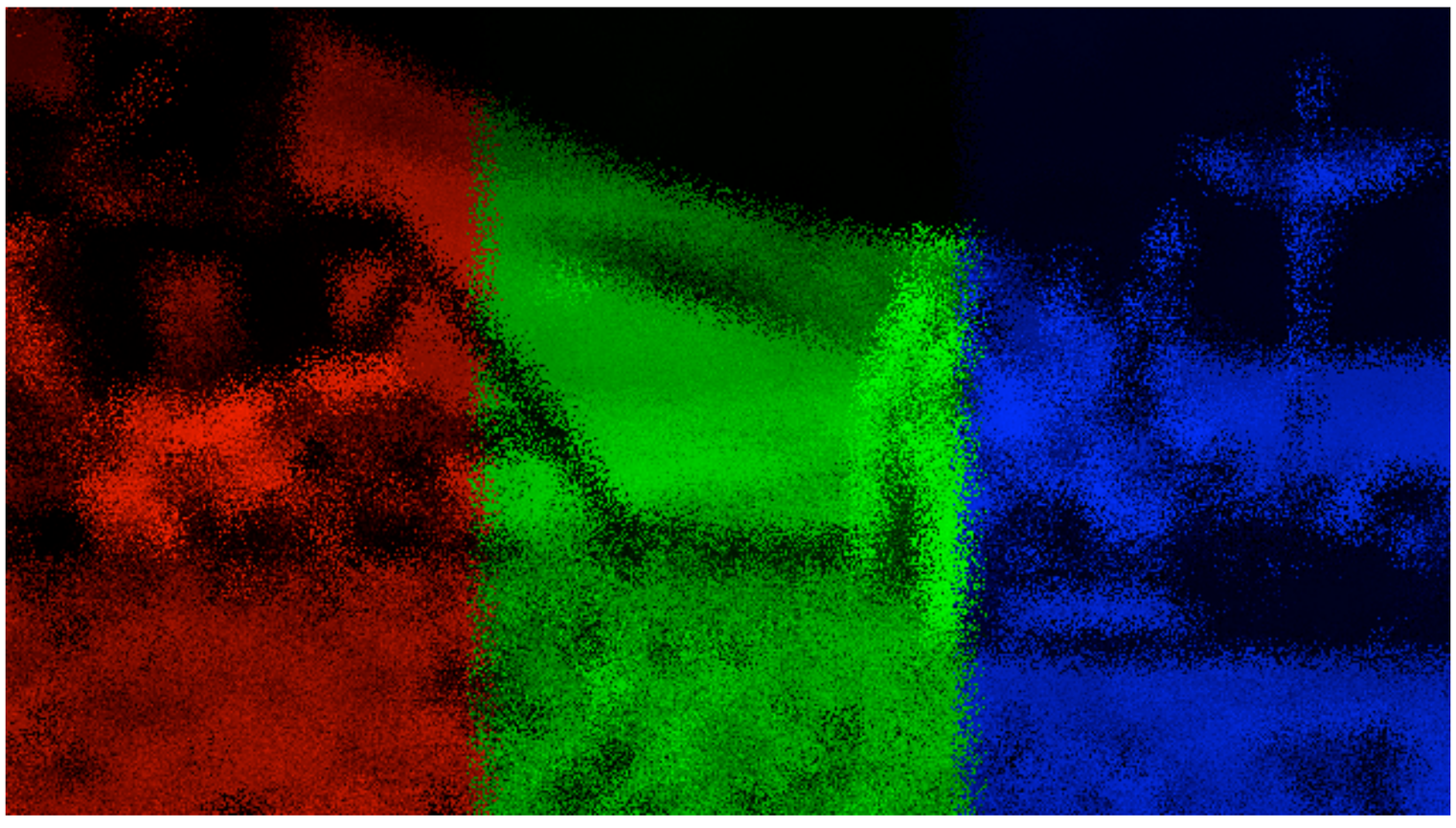
EXAMPLE 4 DISPLACEMENT OPERATION

```
// Based on Algorithms for Visual Design Using the
// Processing Language P.114-115
PImage img;
int step;
void setup() {
  img = loadImage("memorial.jpg");
  size(226, 342);
  noStroke();
}
void draw() {
  if (keyPressed)
    image(img, 0, 0);
  else {
    float d = dist(mouseX, mouseY, width/2, height/2);
    step = int(map(d, 0, 360, 0, 100));
    for (int x=2; x<width-2; x++) //for all rows
      for (int y=2; y<height-2; y++) { //for all columns
        color c = img.get(x, y);
        int xx = x+int(random(-step, step));
        int yy = y+int(random(-step, step));
        set(xx, yy, c); //color the pixel
        //fill(c);
        //rect(xx-2, yy-2, 5, 5);
      }
  }
}
```

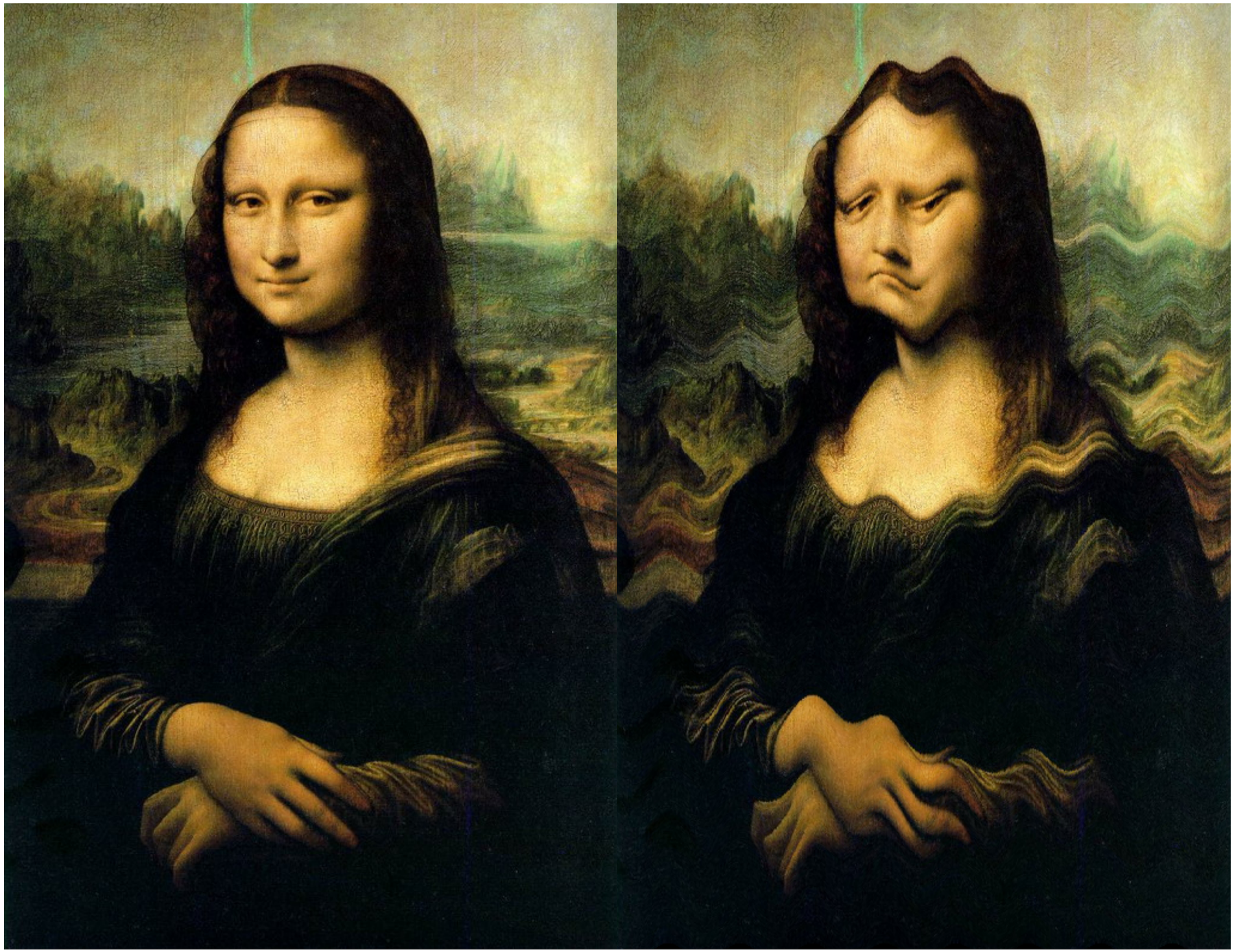


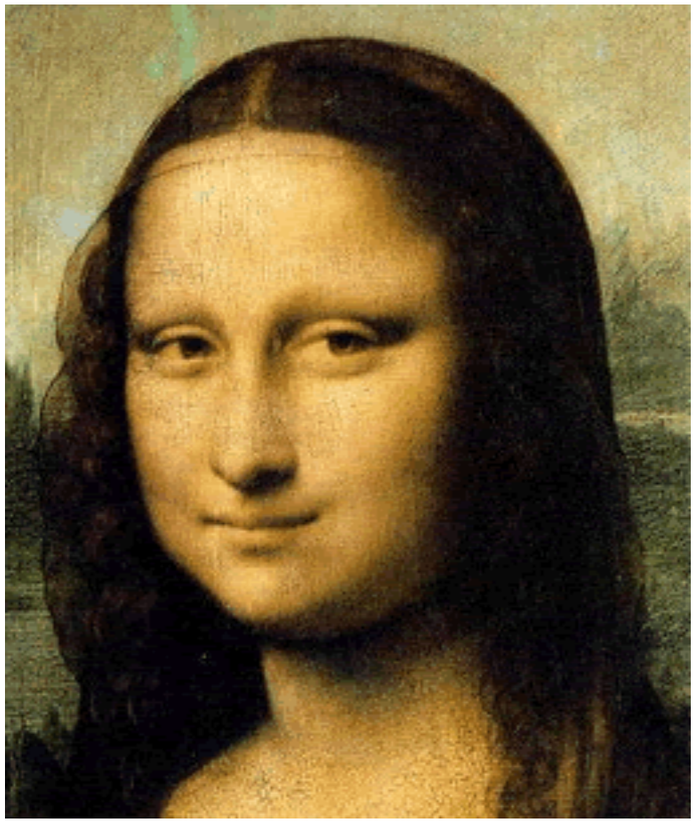
EXERCISE 2

- Base on the given template, create a sketch that manipulate the source image in pixel levels:
  - Display only one channel of color each region
  - Apply a random displacement to each pixel

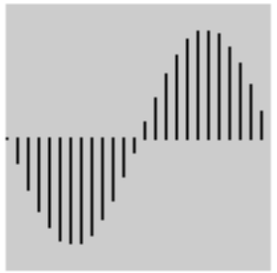


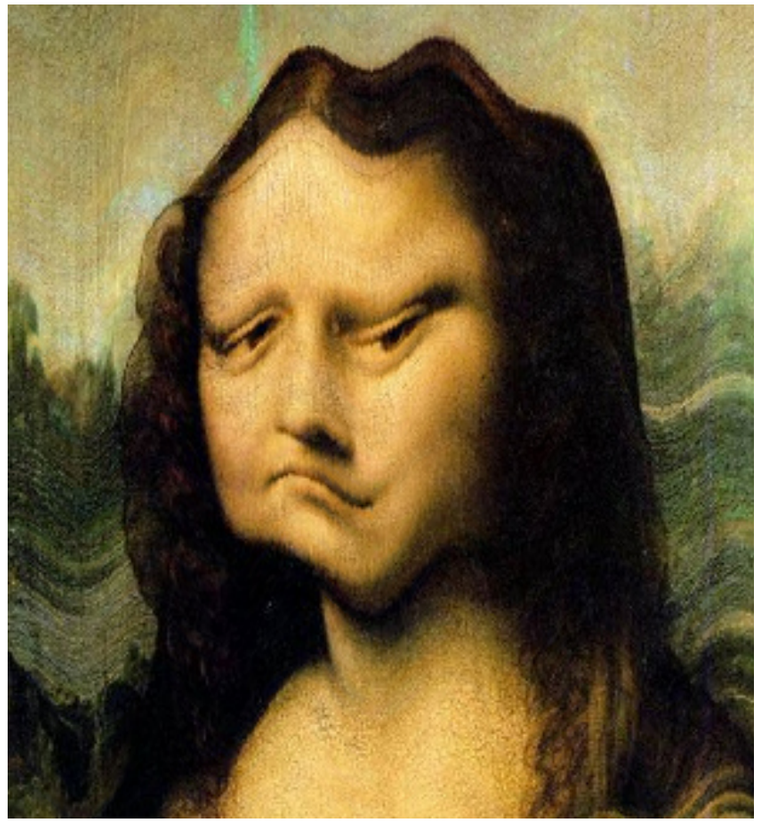
Wave effect



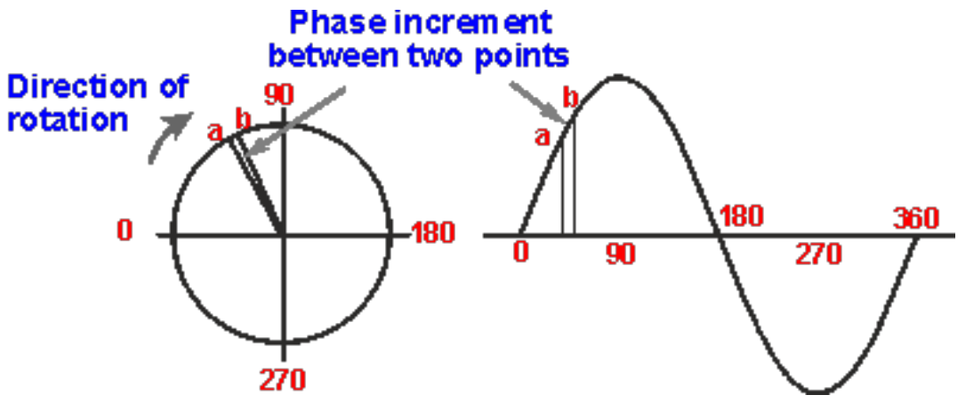


Source Image





Pixels are displaced
along the y axis by using
a sine wave form



Name

sin()

Examples

```
float a = 0.0;
float inc = TWO_PI/25.0;

for(int i=0; i<100; i=i+4) {
  line(i, 50, i, 50+sin(a)*40.0);
  a = a + inc;
}
```

Description

Calculates the sine of an angle. This function expects the values of the **angle** parameter to be provided in radians (values from 0 to 6.28). Values are returned in the range -1 to 1.

Syntax

`sin(rad)`

Parameters

rad	float: an angle in radians

Returns

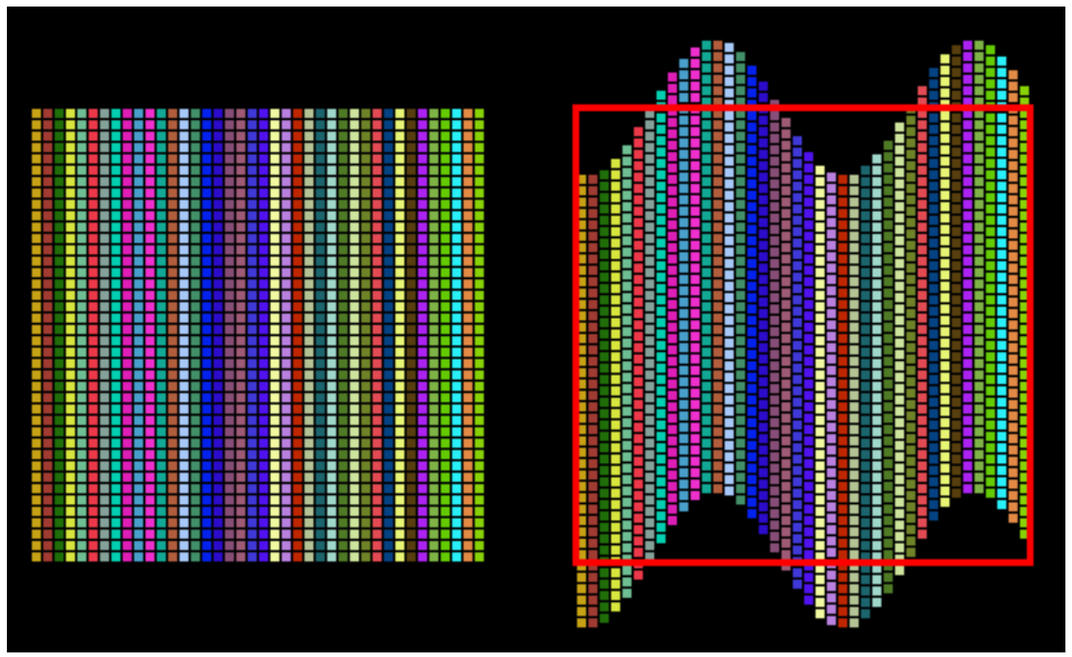
float

Usage

Web & Application

Related

cos()
radians()



source.pixels[loc] destination.pixels[newloc]

```
float a=0;       // Phase
float da=PI/30; // Phase increment: 180/30 = 6 degrees
float amp=8;    // Amplitude for the sin wave

int loc = x + y*w;
int newloc = x + int(y+sin(a)*amp)*w;

newloc = constrain(newloc, 0, w*h-1);
destination.pixels[newloc] = source.pixels[loc];

a=a+da;
```

https://processing.org/reference/sin_.html

EXAMPLE 5 WAVE EFFECT

```
PImage source;      // Source image
PImage destination; // Destination image

float a=0; // Phase
float da=PI/30; // Phase increment: 180/30 = 6 degrees
float amp=8; // Amplitude for the sin wave

int w, h;

void setup() {
  size(515, 800);
  source = loadImage("lisa.jpg");
  destination = createImage(source.width, source.height, RGB);
  // We are going to look at both image's pixels
  source.loadPixels();
  destination.loadPixels();
  noLoop();
  w=source.width;
  h=source.height;
}

void draw() {
  background(0);
  a = 0;
  for (int x = 0; x < w; x++ ) {
    for (int y = 0; y < h; y++ ) {
      int loc = x + y*w;
      int newloc = x + int(y+sin(a)*amp)*w;
      newloc=constrain(newloc, 0, w*h-1); // Image 1D array: from index "0" to "w*h-1"
      destination.pixels[newloc]=source.pixels[loc]; //Copy the pixel from "source" to "destination"
    }
    a=a+da;
  }
  destination.updatePixels(); // We changed the pixels in destination
  image(destination, 0, 0);   // Display the destination
}

void mousePressed()
{
   da=mouseX/1000.0;
   amp= mouseY/15.0;
   println("amp="+amp+" da="+da);
   redraw();
}
```