Generating Images

- create a new blank image

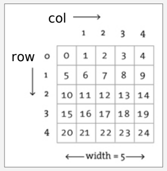
```
img = createImage(w,h,RGB);
```

  - 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);
  }
}
```



		1	2	3	4
row	0	0	1	2	3	4
	1	5	6	7	8	9
	2	10	11	12	13	14
	3	15	16	17	18	19
	4	20	21	22	23	24

- save the pixel colors

```
img.updatePixels();
```

Generating Images

Where to write this?

- create a generator function to do the three steps and return the created image
  - width, height of image to generate as parameters
- call the generator function
  - in `setup()` if generation of the image is the same for every frame
  - in `draw()` if generation of the image can be different in different frames

Example – Defining a Generator Function

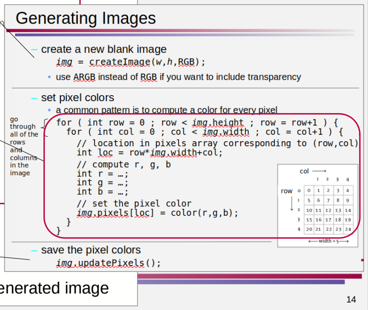
```
// create a random image
//  w, h - dimensions of image to generate
PImage generateRandom ( int w, int h ) {

  // 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);
  }
}
```

col
row
width = 5

		1	2	3	4
row	0	0	1	2	3	4
	1	5	6	7	8	9
	2	10	11	12	13	14
	3	15	16	17	18	19
	4	20	21	22	23	24

- save the pixel colors

```
img.updatePixels();
```

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()`

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

void draw () {
  background(0);

  PImage img = generateRandom(width,height);
  image(img,0,0);
}
```

if the generation of the image can be different in different frames –

declare a local variable for the image, call the generator function to initialize the image variable, and draw the image all in `draw()`

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();
  ```

Image Filters

Where to write this?

- create a filter function to do the four steps and return the created image
  - take the source image as a parameter
- call the function
  - in `setup()` if the filter is the same for every frame
  - in `draw()` if the filter can be different in different frames

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

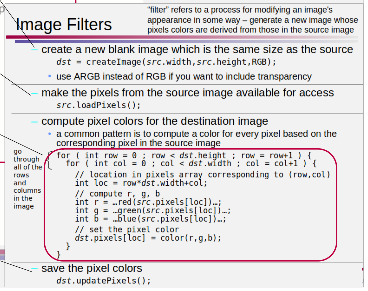


Image Filters

"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 – 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);
}
```

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()`

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

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

  img = loadImage("pelican.jpg");
  brightamt = 0;
}

void draw () {
  background(0);

  image(img, 0, 0, 420, 640);

  PImage filtered = brighten(img, (int)brightamt);
  image(filtered, width/2, 0, 420, 640);

  brightamt = brightamt+0.3;
}
```

if the filter can be different in different frames –

declare a local variable for the image, call the filter function to initialize the image variable, and draw the image all in `draw()`