EECS 1710
Programming for Digital Media

Lecture 17 :: Working with Images - 1

YORK U
UNIVERSITÉ
UNIVERSITY

Recall: Audio & Images, as Arrays

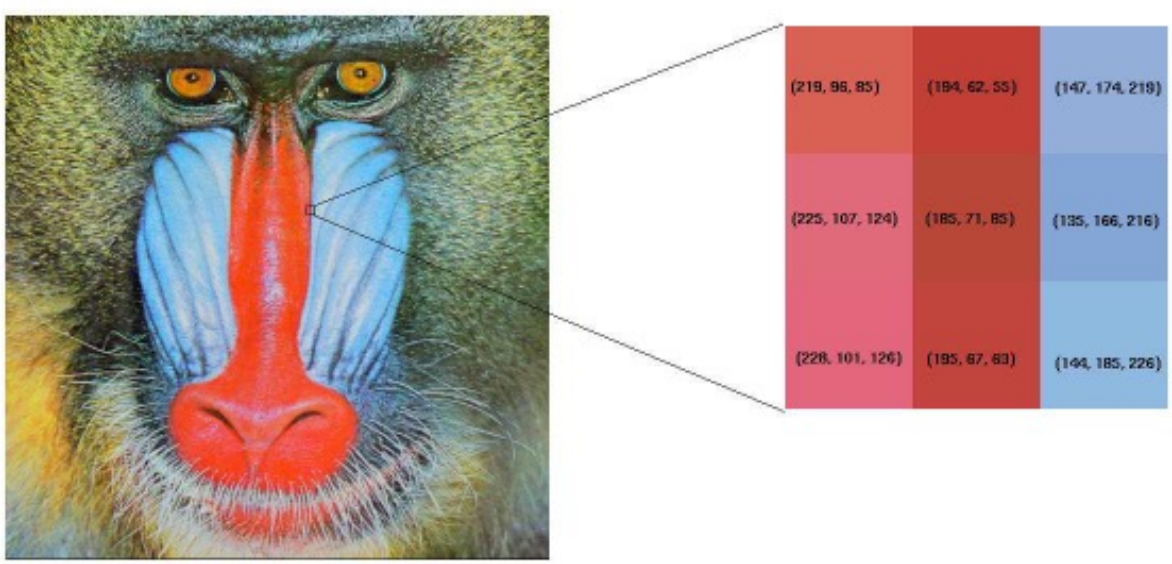
- Audio: 1D (sound samples over time)

1.4	**3.5**	**12**	**4**	**0.6**	**-3.5**	**-10.3**	**…**

1.4	3.5	12	4	0.6	-3.5	-10.3	…

t

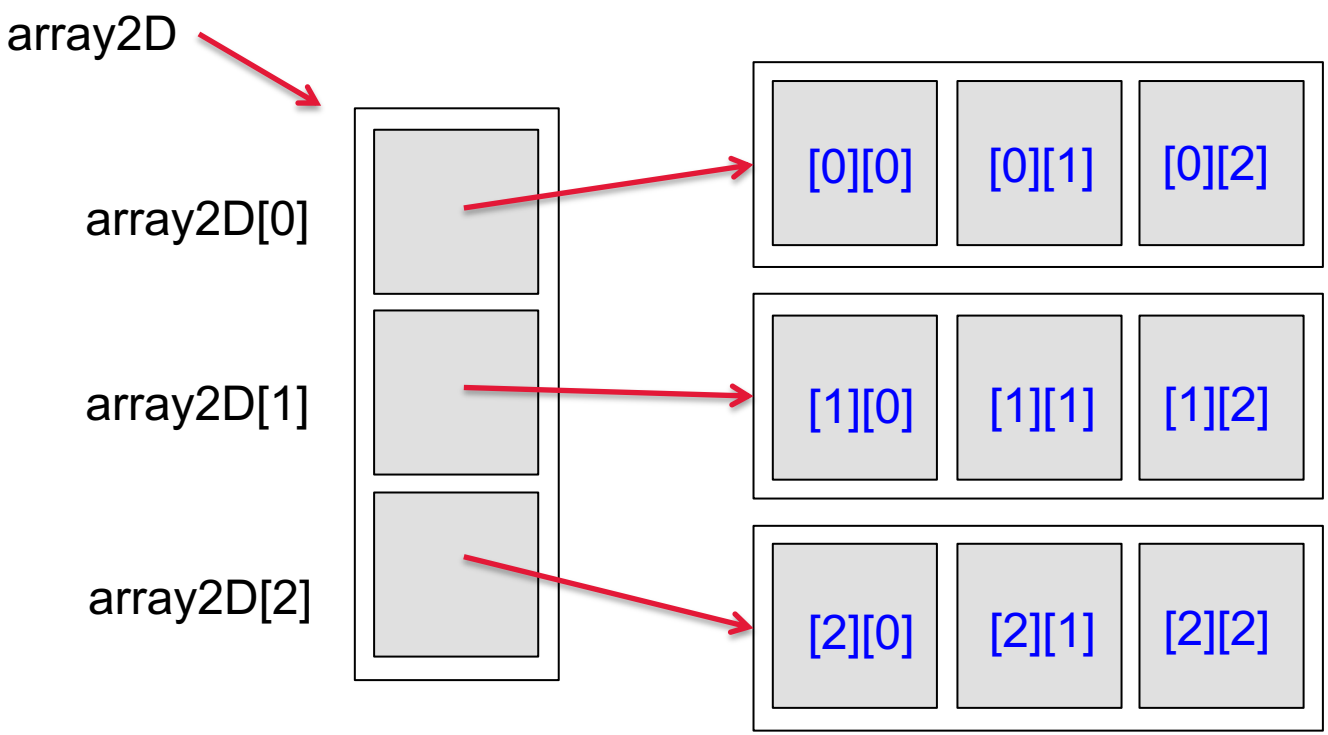
- Images: 2D (pixel/colour samples over space)



x

y

Recall: 2D Array – in memory

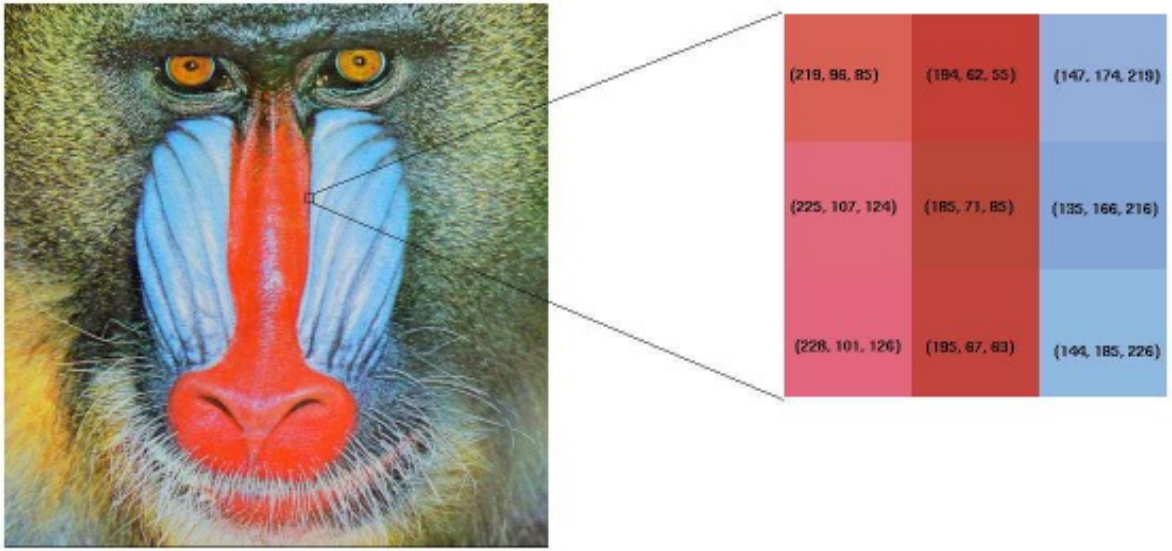


Recall: Iterating over 2D array

```
// assumes array is MxN -> print out


for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {

       out.print("\t" + array2D[i][j]);

   }
   out.print("\n\n\n");

}
```

Image stores 3 values at each location

- r, g, b (red, green & blue value)
- each stored as an int (0-255).



PImage – an object to manage image data

Class Name

PImage

Description

Datatype for storing images. Processing can display `.gif`, `.jpg`, `.tga`, and `.png` images. Images may be displayed in 2D and 3D space. Before an image is used, it must be loaded with the `loadImage()` function. The `PImage` class contains fields for the `width` and `height` of the image, as well as an array called `pixels[]` that contains the values for every pixel in the image. The methods described below allow easy access to the image's pixels and alpha channel and simplify the process of compositing.

Before using the `pixels[]` array, be sure to use the `loadPixels()` method on the image to make sure that the pixel data is properly loaded.

To create a new image, use the `createImage()` function. Do not use the syntax `new PImage()`.

Examples

Copy

```
PImage photo;

void setup() {
  size(400, 400);
  photo = loadImage("Toyokawa-city.jpg");
}

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

PImage

Constructors

`PImage(width, height, format, factor)`

`PImage(width, height, pixels, requiresCheckAlpha, parent)`

`PImage(width, height, pixels, requiresCheckAlpha, parent, format, factor)`

`PImage(img)`

Fields

`pixels[]`	Array containing the color of every pixel in the image
`width`	The width of the image in units of pixels
`height`	The height of the image in units of pixels

Methods

`loadPixels()`	Loads the pixel data for the image into its `pixels[]` array
`updatePixels()`	Updates the image with the data in its `pixels[]` array
`resize()`	Resize the image to a new width and height
`get()`	Reads the color of any pixel or grabs a rectangle of pixels
`set()`	Writes a color to any pixel or writes an image into another
`mask()`	Masks part of an image with another image as an alpha channel
`filter()`	Converts the image to grayscale or black and white
`copy()`	Copies the entire image
`blendColor()`	Blends two color values together based on the blending mode given as the `MODE` parameter
`blend()`	Copies a pixel or rectangle of pixels using different blending modes
`save()`	Saves the image to a TIFF, TARGA, PNG, or JPEG file

Related

`loadImage()`

`imageMode()`

`createImage()`

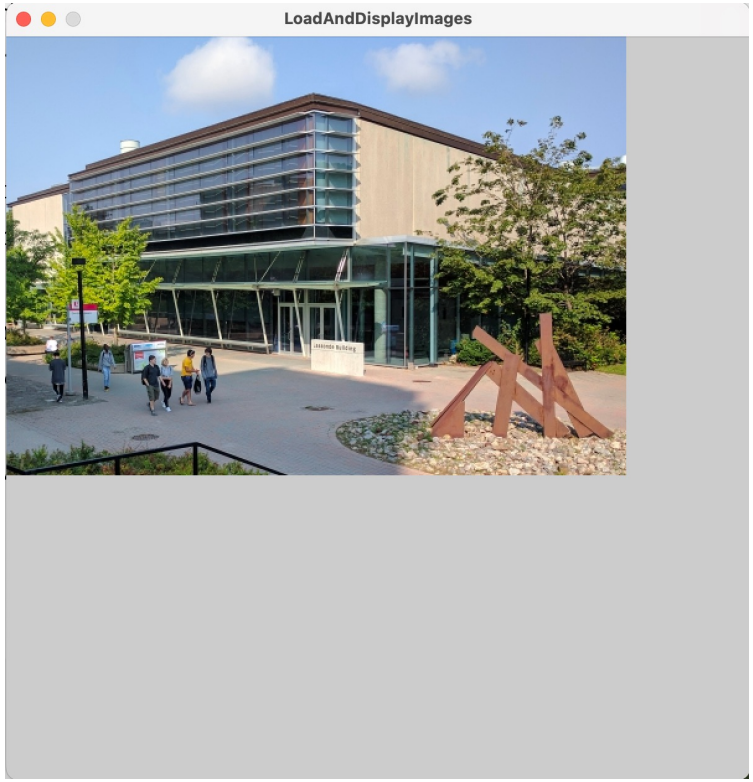
In Processing…

- **`PImage`** object used to store and work with images
  - **`loadImage()`** → to load from file:
    - Recognizes *.gif, *.jpg, *.tga, *.png files (must be in sketch folder)
  - **`createImage()`** → creates a new PImage object:
    - to store/manipulate image data

- To display image:
  - **`imageMode()`** → determines how image() params interpreted
    - CORNER, CORNERS, or CENTER
  - **`image()`** → displays at position (& size) in app window

loadImage(), imageMode() and image():

```
PImage myImage1;

void setup() {
  size(600,600);
  myImage1 = loadImage("lassonde.jpg");

}


void draw() {
  imageMode(CORNER);
  image(myImage1,0,0);

}
```



loadImage(), imageMode() and image():

```
PImage myImage1;

void setup() {
  size(600,600);
  myImage1 = loadImage("lassonde.jpg");

}


void draw() {
  imageMode(CENTER);
  image(myImage1,width/2,height/2);

}
```

loadImage(), imageMode() and image():

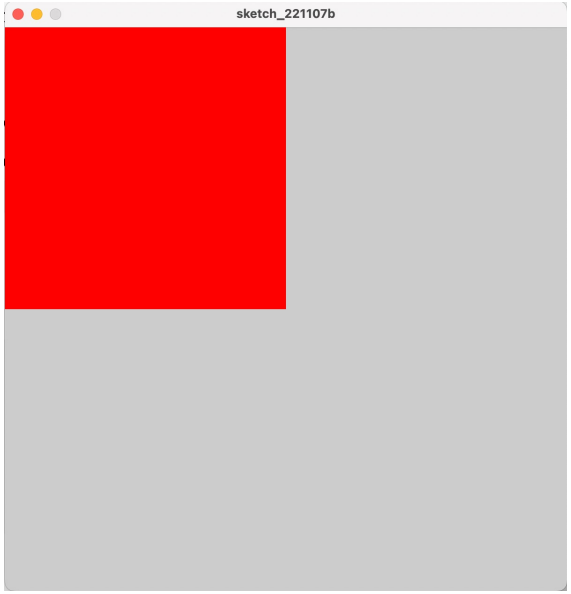
```
PImage myImage1;

void setup() {
  size(600,600);
  myImage1 = loadImage("lassonde.jpg");

}


void draw() {
  imageMode(CENTER);
  image(myImage1,width/2,height/2,
                 myImage1.width/2,
                 myImage1.height/2);

}
```

Creating a blank image

Syntax	`createImage(w, h, format)`		
Parameters	**w**	(int)	width in pixels
	h	(int)	height in pixels
	format	(int)	Either RGB, ARGB, ALPHA (grayscale alpha channel)
Return	`PImage`		
Related	`PImage`		
	`PGraphics`		

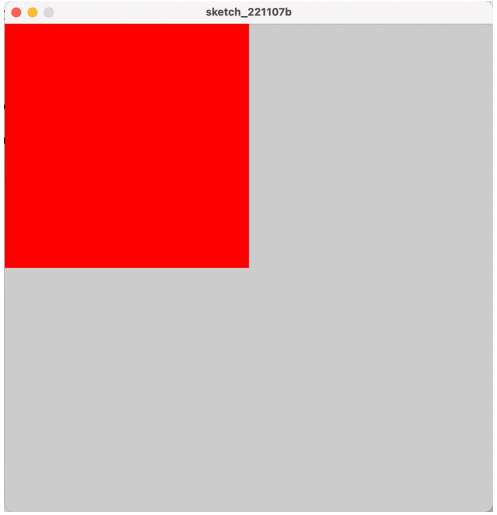
Traversing, accessing and changing pixels:

```
final int BLACK = color(0,0,0);
final int RED = color(255,0,0);

void setup() {

  size(600,600);
  PImage img1 = createImage(300,300, RGB);

  // access and set pixels in img1 (method 1):

  for (int i=0; i<img1.width; i++) {
    for (int j=0; j<img1.height; j++) {

      // access a pixel at i,j location (in img1)
      int pixel = img1.get(i,j);

      // modify a pixel at i,j location (in img1)
      img1.set(i, j, RED);

    }
  }

  image(img1,0,0);  // show image
}
```



Traversing, accessing and changing pixels:

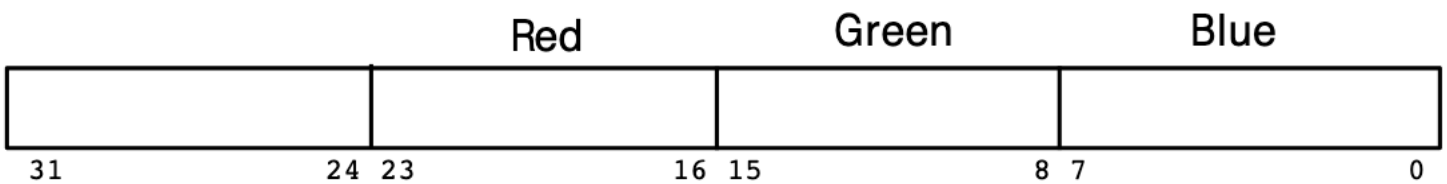
```
final int BLACK = color(0,0,0);
final int RED = color(255,0,0);

void setup() {

  size(600,600);
  PImage img1 = createImage(300,300, RGB);

  // access and set pixels in img1 (method 1):
  for (int i=0; i<img1.width; i++) {
    for (int j=0; j<img1.height; j++) {

      int pixel = img1.get(i,j);
      if (i<=10 && j==0)  // print some pixel values before set
        print("pixel (" + i + ", " + j + ")=" + pixel + " ");

      img1.set(i, j, RED);

      pixel = img1.get(i,j);
      if (i<=10 && j==0) // print some pixel values before set
        print(" -> pixel (" + i + ", " + j + ")=" + pixel + " ");
    }
    if (i<=10)
      println();
  }

  image(img1,0,0);        // show image
}
```



??

```
pixel (0, 0)=-16777216  -> pixel (0, 0)=-65536
pixel (1, 0)=-16777216  -> pixel (1, 0)=-65536
pixel (2, 0)=-16777216  -> pixel (2, 0)=-65536
pixel (3, 0)=-16777216  -> pixel (3, 0)=-65536
pixel (4, 0)=-16777216  -> pixel (4, 0)=-65536
pixel (5, 0)=-16777216  -> pixel (5, 0)=-65536
pixel (6, 0)=-16777216  -> pixel (6, 0)=-65536
pixel (7, 0)=-16777216  -> pixel (7, 0)=-65536
pixel (8, 0)=-16777216  -> pixel (8, 0)=-65536
pixel (9, 0)=-16777216  -> pixel (9, 0)=-65536
pixel (10, 0)=-16777216  -> pixel (10, 0)=-65536
```

- Each pixel holds a color as a 24-bit RGB value (within a 32-bit integer):



- Each value is 8 bits in the range 0_{10} to 255_{10}
  - 0 → absence of R, G, or B
  - 255 → full intensity R, G, or B

Note:
$255_{10} = 11111111_2$

RGB Color Model

Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Cyan	0	255	255
Magenta	255	0	255
White	255	255	255
Black	0	0	0

24 bits

Hexadecimal representations of colour??

Color	Red	Green	Blue
"red"	0 0 0 0 0 0 0 0 (31–24)	1 1 1 1 1 1 1 1 (23–16)	0 0 0 0 0 0 0 0 (15–8)	0 0 0 0 0 0 0 0 (7–0)
"grey"	0 0 0 0 0 0 0 0 (31–24)	1 0 1 0 0 0 1 1 (23–16)	1 0 1 0 0 0 1 1 (15–8)	1 0 1 0 0 0 1 1 (7–0)
?	0 0 0 0 0 0 0 0 (31–24)	1 1 0 0 1 0 1 0 (23–16) 0xca	1 0 1 0 1 0 0 0 (15–8) 0xa8	0 0 1 0 1 0 1 1 (7–0) 0x2b

```
int color = 0xcaa82b; // some color (see above)
```

Hexadecimal

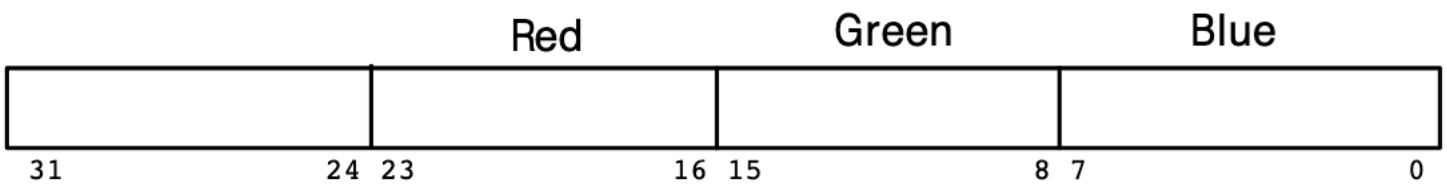
Hexadecimal?

- recall:
  - **decimal** is a 10-digit number system (digits 0-9)
    - "123" = $1*10^2 + 2*10^1 + 3*10^0$
  - **binary** is a 2-digit number system (digits 0,1)
    - "01011" = $0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

- **Hexadecimal** = 16-digit number system (digits: 0-9,a-f)
  - 0x**ca** = $\mathbf{c}*16^1 + \mathbf{a}*16^0$
    = (**12**)*16 + (**10**)*1
    = 202

a=10, b=11, c=12,
d=13, e=14, f=15

- Each pixel holds a color as a 24-bit RGB value (within a 32-bit integer):



alpha channel

(indicates transparency)

Unpacking a colour value:

```
// show image

  image(img1,0,0);

  int pixel = img1.get(0,0);
  println();
  println("pixel: (" + pixel + ") -> ");

  println(" (r) = " + red(pixel)                                    );
  println(" (g) = " + green(pixel)                                  );
  println(" (b) = " + blue(pixel)                                   );
```

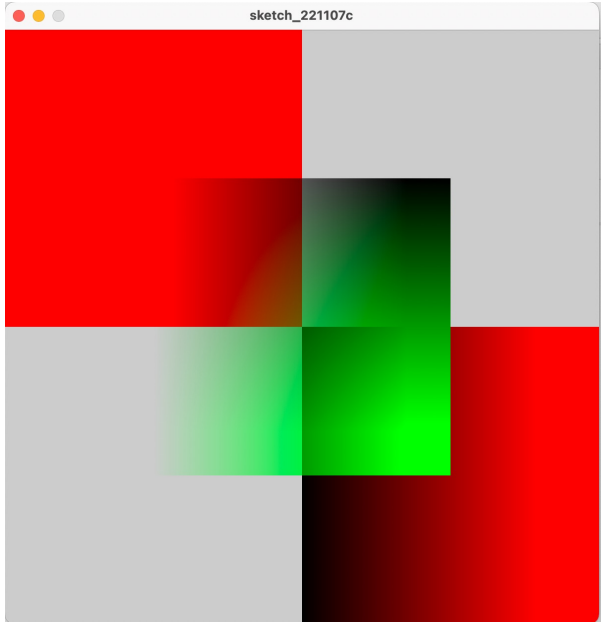
BLACK (before set)

```
pixel: (-16777216) ->
 (r) = 0.0
 (g) = 0.0
 (b) = 0.0
```

RED (after set)

```
pixel: (-65536) ->
 (r) = 255.0
 (g) = 0.0
 (b) = 0.0
```

Gradients

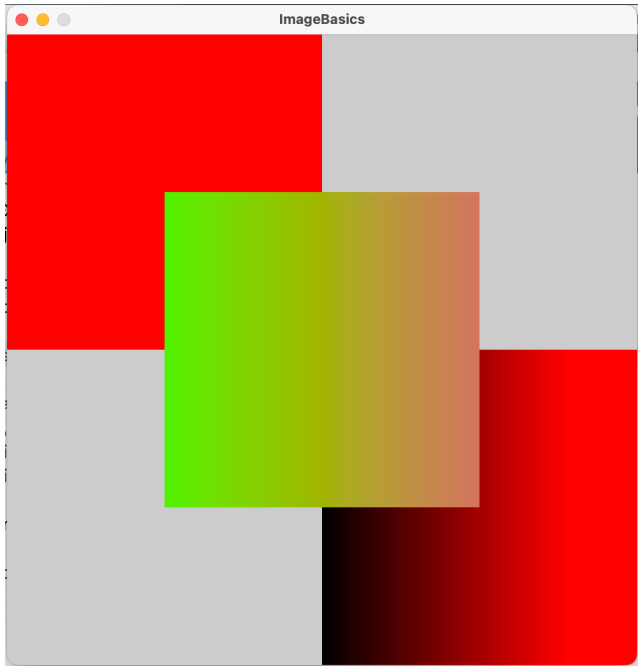



```
void setup() {

  size(600,600);
  PImage img1 = createImage(300,300, RGB);
  PImage img2 = createImage(300,300, RGB);
  PImage img3 = createImage(300,300, ARGB);

  // access and set pixels in img1 directly:
  for (int i=0; i<img1.width; i++) {
    for (int j=0; j<img1.height; j++) {

      img1.set(i, j, RED);
      img2.set(i, j, color(i,0,0));     // gradient
      img3.set(i, j, color(0,j,0,i));   // gradient (colour + transparency)
    }
  }

  // show image
  image(img1,0,0);
  image(img2,300,300);
  image(img3,150,150);

}
```

Syntax	color(gray)		
	color(gray, alpha)		
	color(v1, v2, v3)		
	color(v1, v2, v3, alpha)		
Parameters	**gray**	(int)	number specifying value between white and black
	alpha	(int, float)	relative to current color range
	v1	(int, float)	red or hue values relative to the current color range
	v2	(int, float)	green or saturation values relative to the current color range
	v3	(int, float)	blue or brightness values relative to the current color range

Linear Interpolation (useful methods)

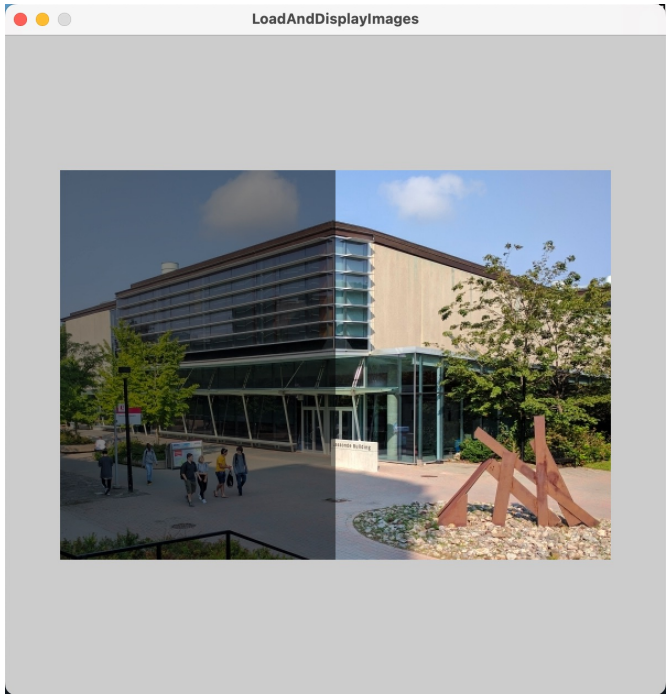
- lerp()
- lerpColor()




```
// Linear Interpolate between 2 colours
// Here, i/img1.width used as a percentage change
// As (i/img1.width) closer to 0, more of "from" colour
// As (i/img1.width) closer to 1, more of "to" colour

int from = color(130,240,11);
int to   = color(200,120,100);

img3.set(i, j, lerpColor(from, to, float(i)/img1.width));
```

Can access and mod pixels from a preloaded image in the same way:

```
PImage myImage1;

void setup() {
  size(600,600);
  myImage1 = loadImage("lassonde.jpg");

  for (int i=0; i<myImage1.width; i++) {
   for (int j=0; j<myImage1.height; j++) {

     if (i<myImage1.width/2) {
       float red = red(myImage1.get(i,j));
       float green = green(myImage1.get(i,j));
       float blue = blue(myImage1.get(i,j));

       myImage1.set(i,j,
          color(red/2,green/2,blue/2,i) );
     }
   }
  }
}

void draw() {
  imageMode(CENTER);
  image(myImage1,width/2,height/2);

}
```



Alternative to using get(), set()

- `pixels[]`

  1D array that holds all pixels in an image/display window
  Made available using the loadPixels() method

- `loadPixels()`

  makes pixels[] available

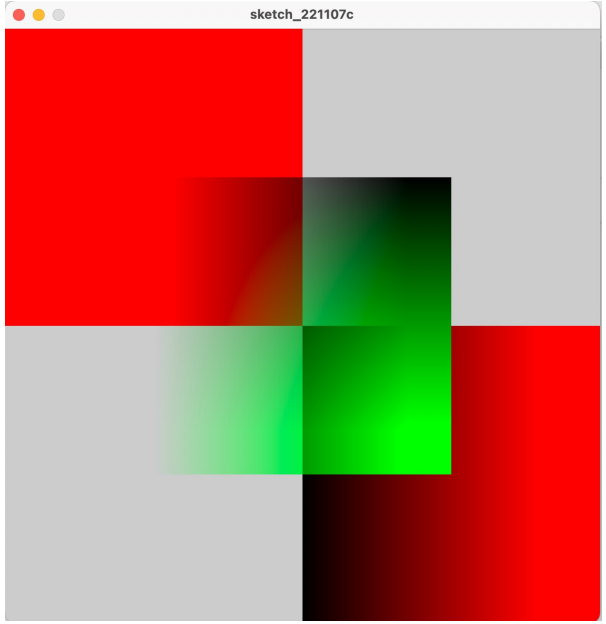
  can also use method on a specific PImage (makes a pixels[] array available in that image)

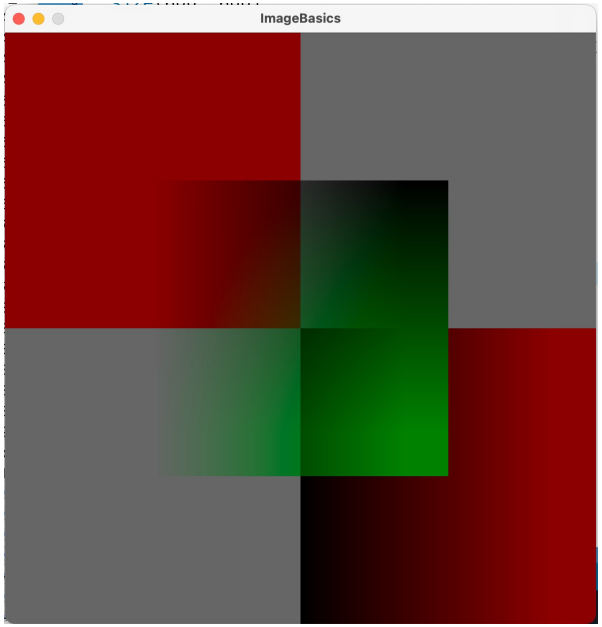
- `updatePixels()`

  updates the pixels in the image by writing the values from the pixels[] array back into the image

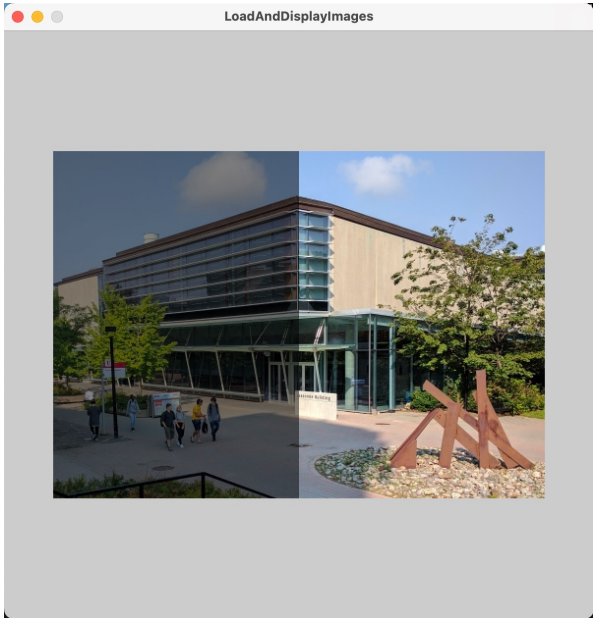
pixels[ ]

```
// code to create gradients (not shown)

// show images (paints 3 images into app window)
  image(img1,0,0);
  image(img2,300,300);
  image(img3,150,150);

  // get pixels array for entire app window
  loadPixels();

  for (int i=0; i<width; i++) {
    for (int j=0; j<height; j++) {

      int location = i + j*width;

      int pixelColor = pixels[location];

      float r = red(pixelColor);
      float g = green(pixelColor);
      float b = blue(pixelColor);

      pixels[location] = color(r/2,g/2,b/2);
    }
  }
  updatePixels();
```





pixels[ ]




```
myImage1 = loadImage("lassonde.jpg");
myImage1.loadPixels();

for (int i=0; i<myImage1.width; i++) {
    for (int j=0; j<myImage1.height; j++) {

      int location = i + j*myImage1.width;

      if (i<myImage1.width/2) {
        float red = red(myImage1.pixels[location]);
        float green = green(myImage1.pixels[location]);
        float blue = blue(myImage1.pixels[location]);

        myImage1.pixels[location] = color(red/2, green/2, blue/2, i) ;
      }
    }
}

myImage1.updatePixels();

imageMode(CENTER);
image(myImage1, width/2, height/2);
```

Homework - consider how you might:

- Flip an image horizontally?
- Flip an image vertically?
- Flip a region in an image ??

- Next lecture.. (above) +
- Querying & merging pixels from two images
  - e.g. common application = *chroma-keying* (working with green/blue screens)