EECS 1710
Programming for Digital Media

Lecture 18 :: Working with Images - 2

YORK UNIVERSITÉ UNIVERSITY

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)                                   );
```

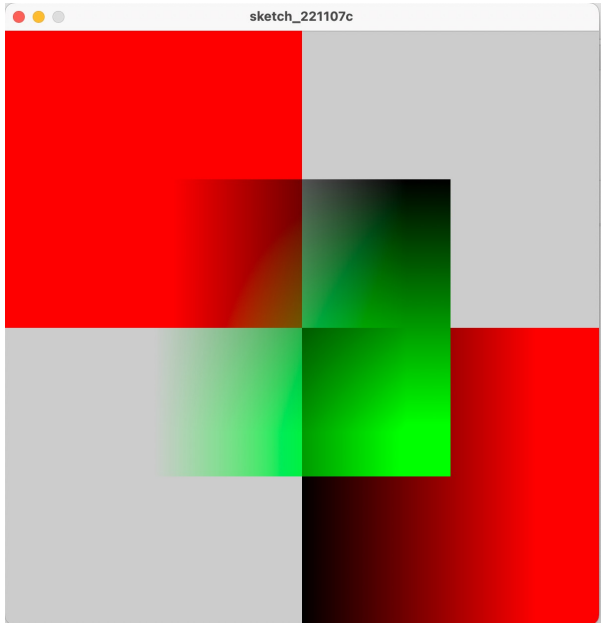
BLACK (before set)

```
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

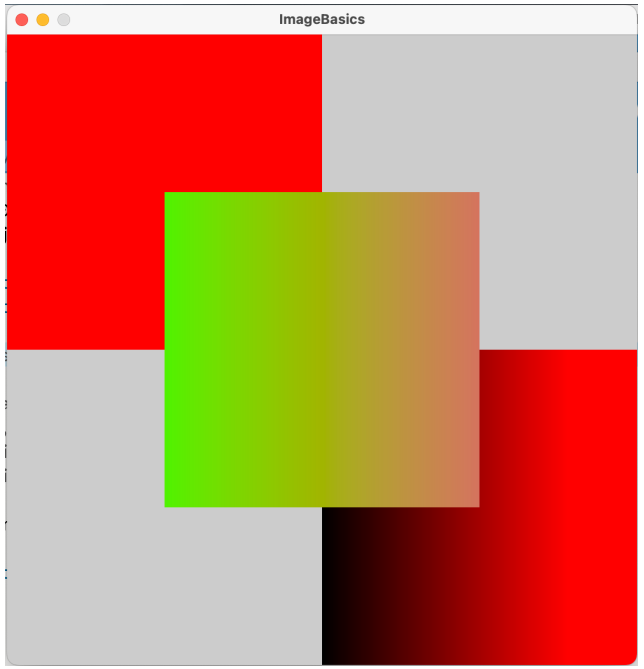



```
void setup() {

  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)

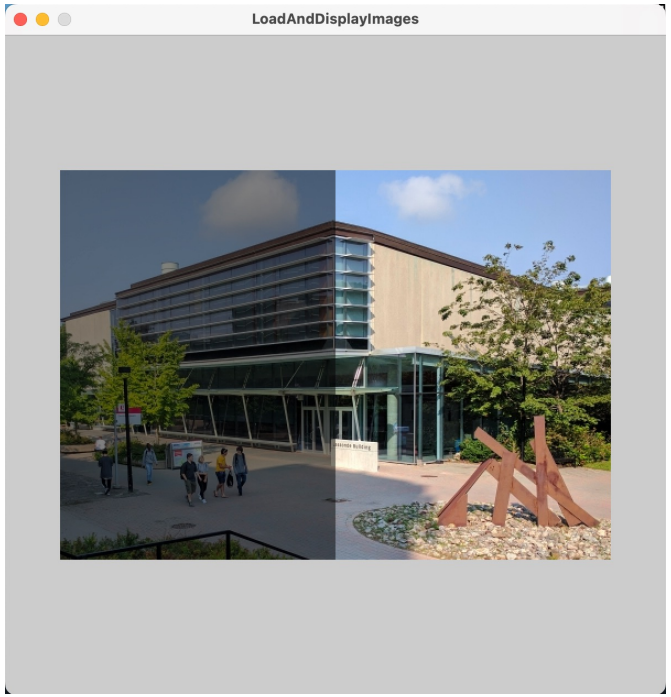
- lerp()
- lerpColor()




```
// 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);

}
```



YORK UNIVERSITÉ UNIVERSITY

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

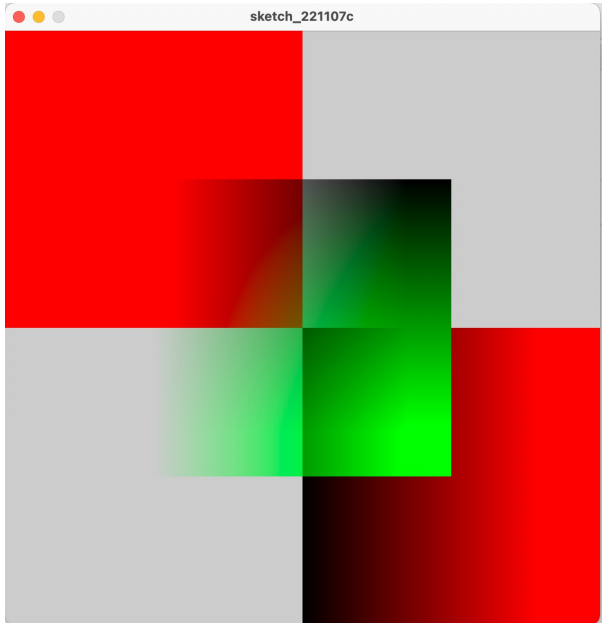
  can also use method on a specific PImage (makes a pixels[] array available in that image)

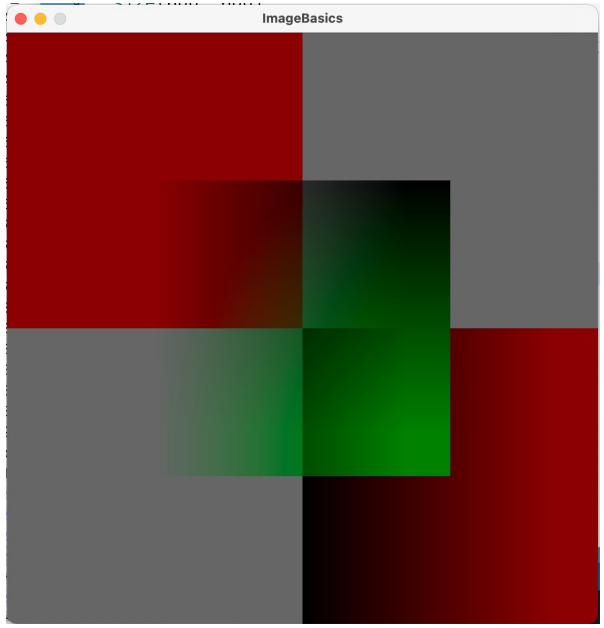
- `updatePixels()`

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

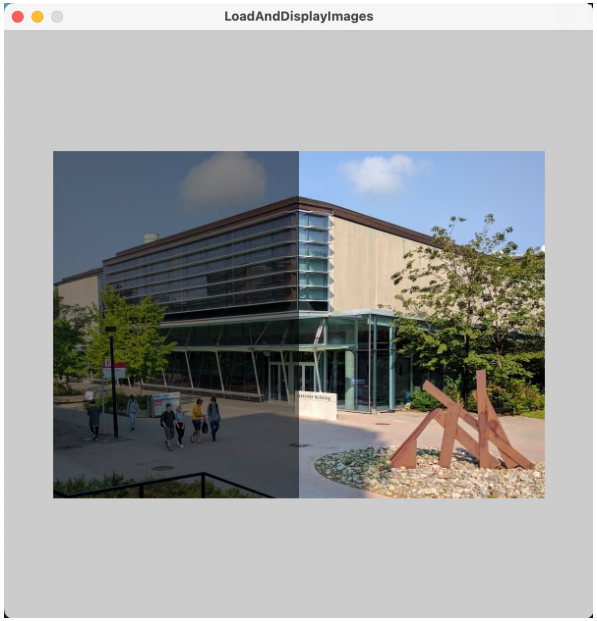
pixels[ ]

```
// 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);
```

YORK UNIVERSITÉ UNIVERSITY

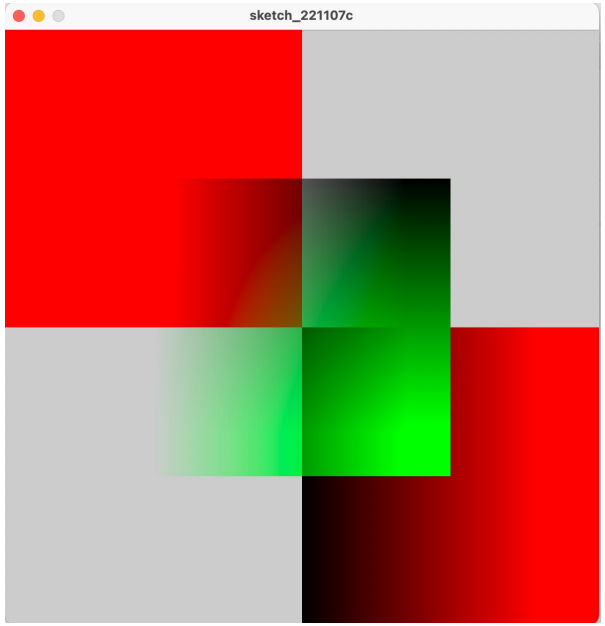
Using app window as compositing canvas

- Painting images overlaid will show mixtures (if transparency exists)
- Similarly, painting images + drawing graphics over the top will composite image data (into app window)

- How then to extract the results?
  - get() → can be used to grab the pixels from the app window when not invoked on a Pimage
  - E.g.

```
// previous statements to paint images and draw
PImage myImg = createImage(width,height, ARGB);
save("mySaveImage.jpg");
myImg = get();
```

Saving




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

// saving & getting images

save("myComposite1.jpg");      // saves from app window
img2.save("myImg2.png");       // saves from PImage object
img3.save("myImg3.png");

PImage myImg = createImage(width,height, ARGB);
myImg = get();
myImg.save("myComposite2.jpg");
```

YORK UNIVERSITÉ UNIVERSITY

Filters

- Built-in methods → filter()
- https://processing.org/reference/PImage_filter_.html

Usage:

```
PImage img = loadImage("filename.jpg");

img.filter( … );

// modifies the pixels of img by applying the filter
e.g.
img.filter(THRESHOLD, 0.5);  // creates black & white image
img.filter(GRAY);            // converts to grayscale
img.filter(BLUR,6);          // blurs with 6x6 mask
img.filter(POSTERIZE,4);     // condenses to 4 colours
```

Masks

- Learn more about this in lab 6
- Create an image with alpha channel (with transparency)

Usage:

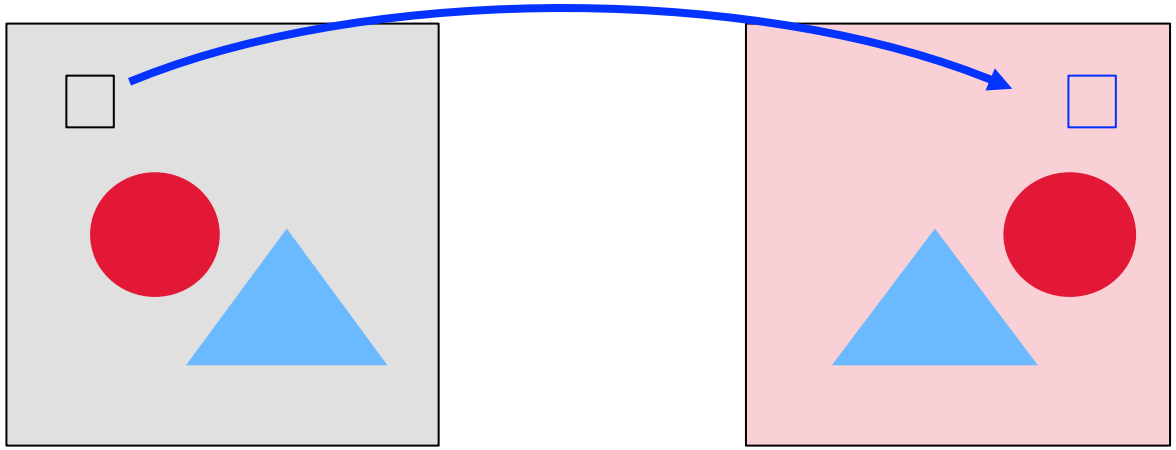
```
PImage img = loadImage(“filename.jpg”);
PImage imgA = loadImage(“alphaImage.jpg”);

img.mask(imgA);      // similar to painting as overlay
                     // to app window
```

Spatial manipulations

- Flip (vertical or horizontally flip image)
- Crop (select subimage from within image)
- Tile Mosaic (duplicate image MxN times)

Flip

- Horizontally



New position (i,j) takes value of (width-1-i , j)

- Vertically?

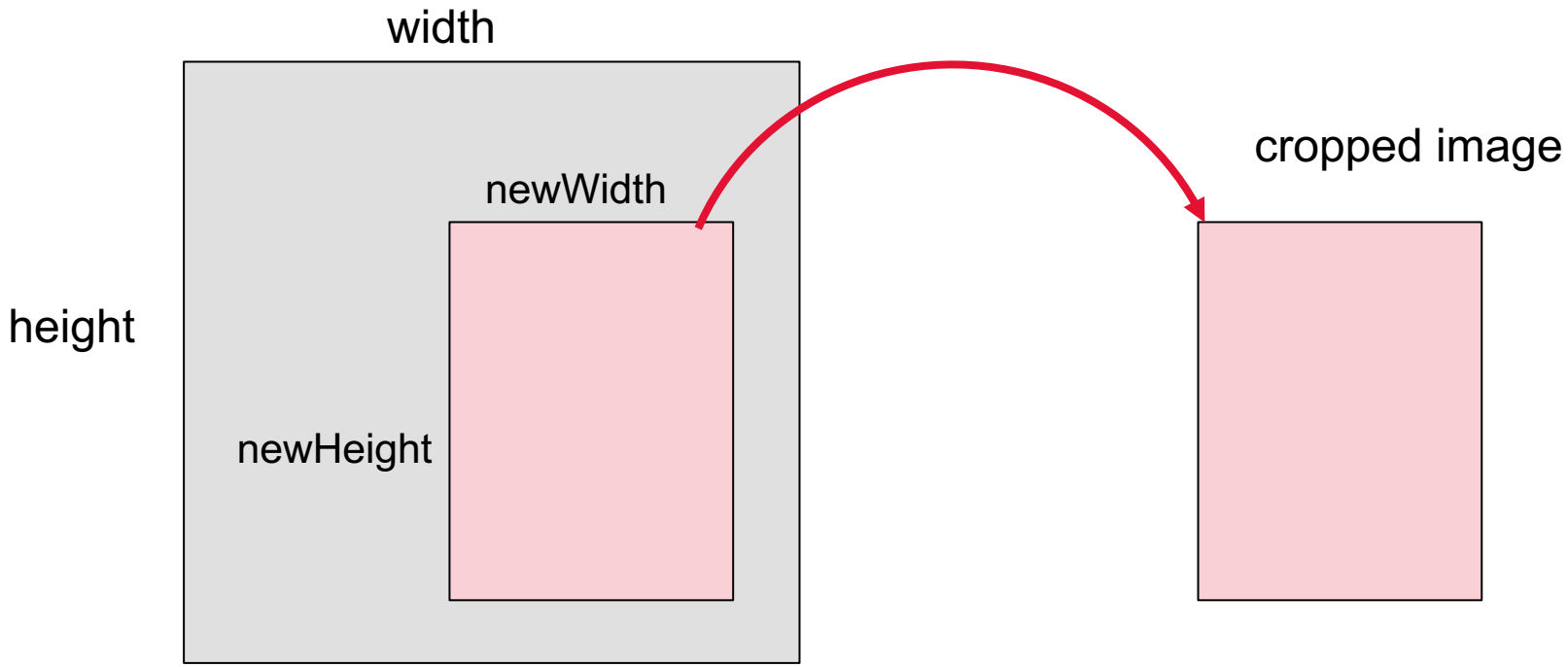
New position (i,j) takes value of (i , height-1-j)

Flip

```
PImage flipImage(PImage original, String direction) {

  PImage result = original.copy();


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

      if (direction.toLowerCase().equals("horizontal")) {
        result.set(i,j,original.get(original.width-1-i,j));

      }
      else if (direction.toLowerCase().equals("vertical")) {
        result.set(i,j,original.get(i,original.height-1-j));

      }
    }
  }

  return result;

}
```

Cropping a Picture

- Idea: extract a sub region and create a picture from it



Cropping with get()

Syntax

```
pimg.get(x, y)
pimg.get(x, y, w, h)
pimg.get()
```

Parameters

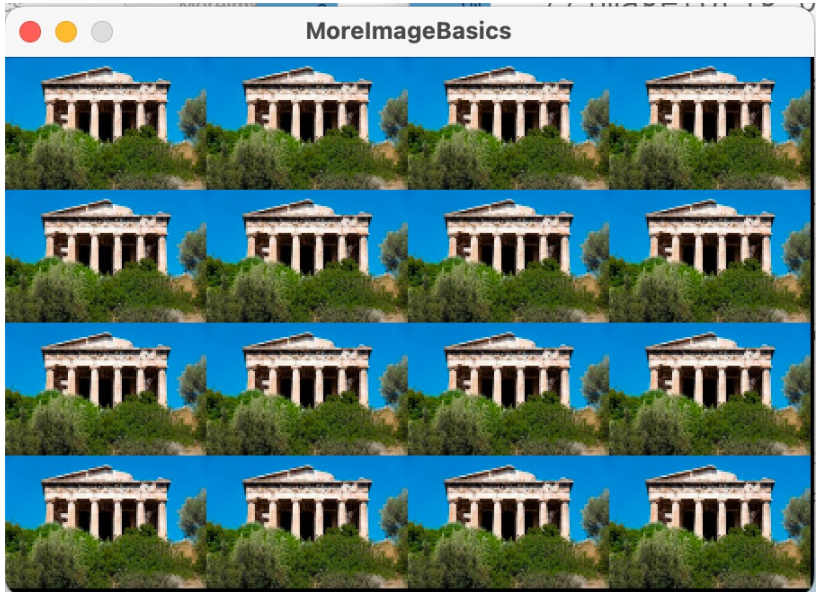
pimg	(PImage)	any object of type PImage
x	(int)	x-coordinate of the pixel
y	(int)	y-coordinate of the pixel
w	(int)	width of pixel rectangle to get
h	(int)	height of pixel rectangle to get

Crop

```
PImage cropImage(PImage orig, PVector maskStart, PVector maskFinish) {


  PImage result = orig.get( int(maskStart.x),
                            int(maskStart.y),
                            int(maskFinish.x-maskStart.x),
                            int(maskFinish.y-maskStart.y)   );

  return result;

}
```

Tiling (create a mosaic)

```
PImage tileMosaic(PImage original, int numSteps) {

  PImage result = createImage(original.width, original.height, RGB);
  int tileW = original.width/numSteps;
  int tileH = original.height/numSteps;

  PImage tile = original.copy();
  tile.resize(tileW,tileH);

  for (int i=0; i<numSteps; i++) {
   for (int j=0; j<numSteps; j++) {
     result.set(i*tileW,
                j*tileH, tile);

    }
  }

  return result;
}
```



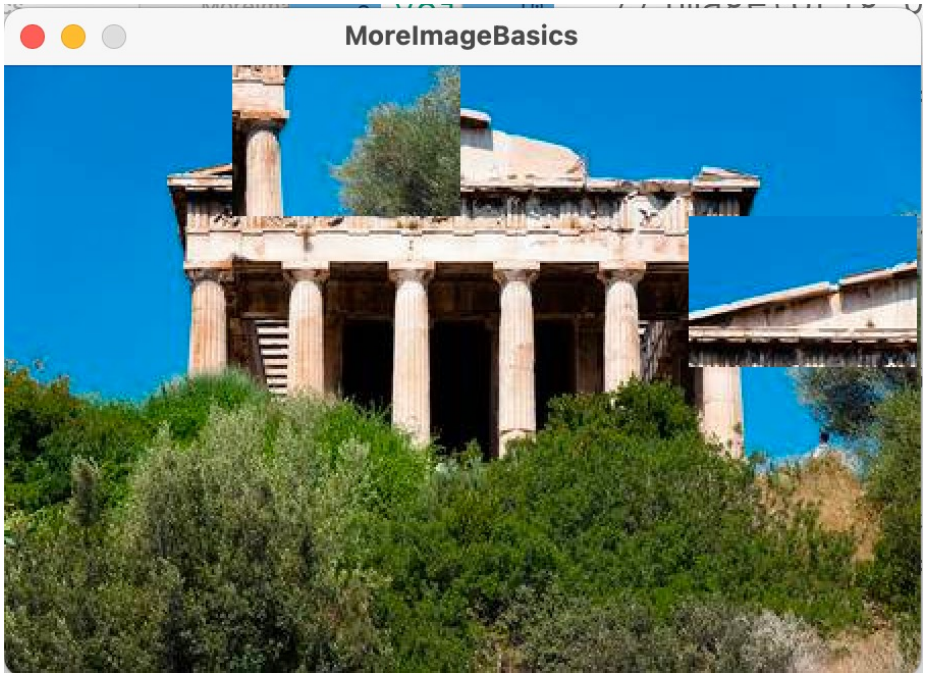
Crop and Swap! (swap two sub-tiles)

```
PImage swapTiles(PImage original, int numSteps) {

  int tileW = original.width/numSteps;
  int tileH = original.height/numSteps;

  PImage result = original.copy();

  // pick random index on numSteps grid
  // (both horizontal and vertical)

  int iSrcTile = floor(random(numSteps));
  int jSrcTile = floor(random(numSteps));

  int iDstTile   = floor(random(numSteps));
  int jDstTile   = floor(random(numSteps));

  println("moving tile(" + iSrcTile + "," + jSrcTile + ")
       -> tile(" + iDstTile + "," + jDstTile + ")");

  // …
```

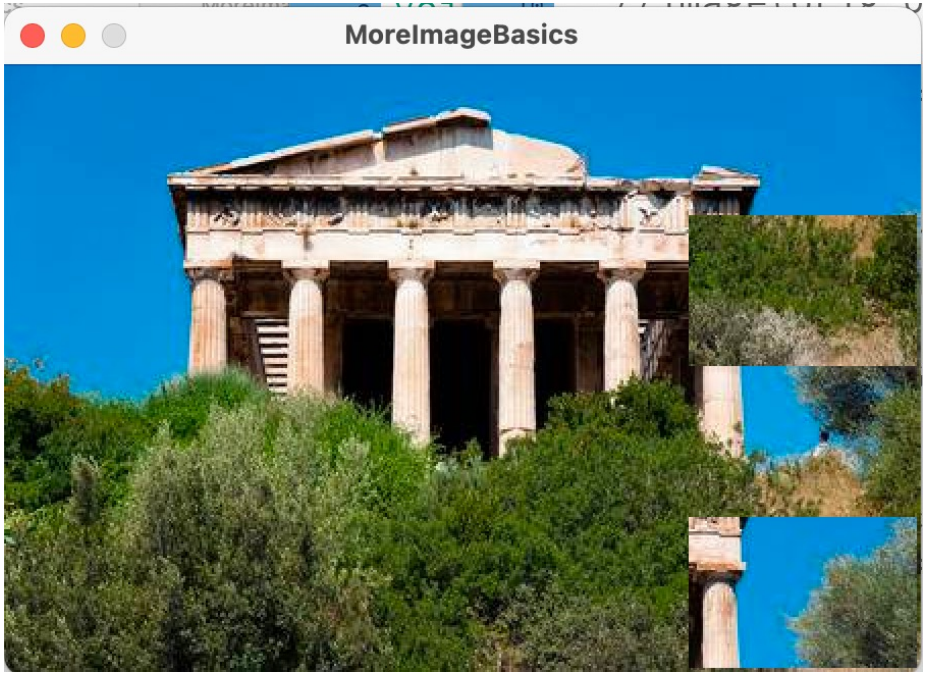
YORK UNIVERSITÉ UNIVERSITY

```
// … continued

// save dstTile in temp image
  PImage tempTile = createImage(tileW,tileH,RGB);
  tempTile.set( 0, 0, result.get(iDstTile*tileW,
                        jDstTile*tileH, tileW, tileH) );

  //image(original,width/2,height/2);


  // copy srcTile into dstTile location
  result.set(iDstTile*tileW, jDstTile*tileH,
                original.get(iSrcTile*tileW,
                 jSrcTile*tileH, tileW, tileH) );

  // copy temp image (i.e. original dstTile)
  // into srcTile location
  result.set(iSrcTile*tileW, jSrcTile*tileH, tempTile );


  return result;

}
```

```
moving tile(1,0) -> tile(3,1)
```

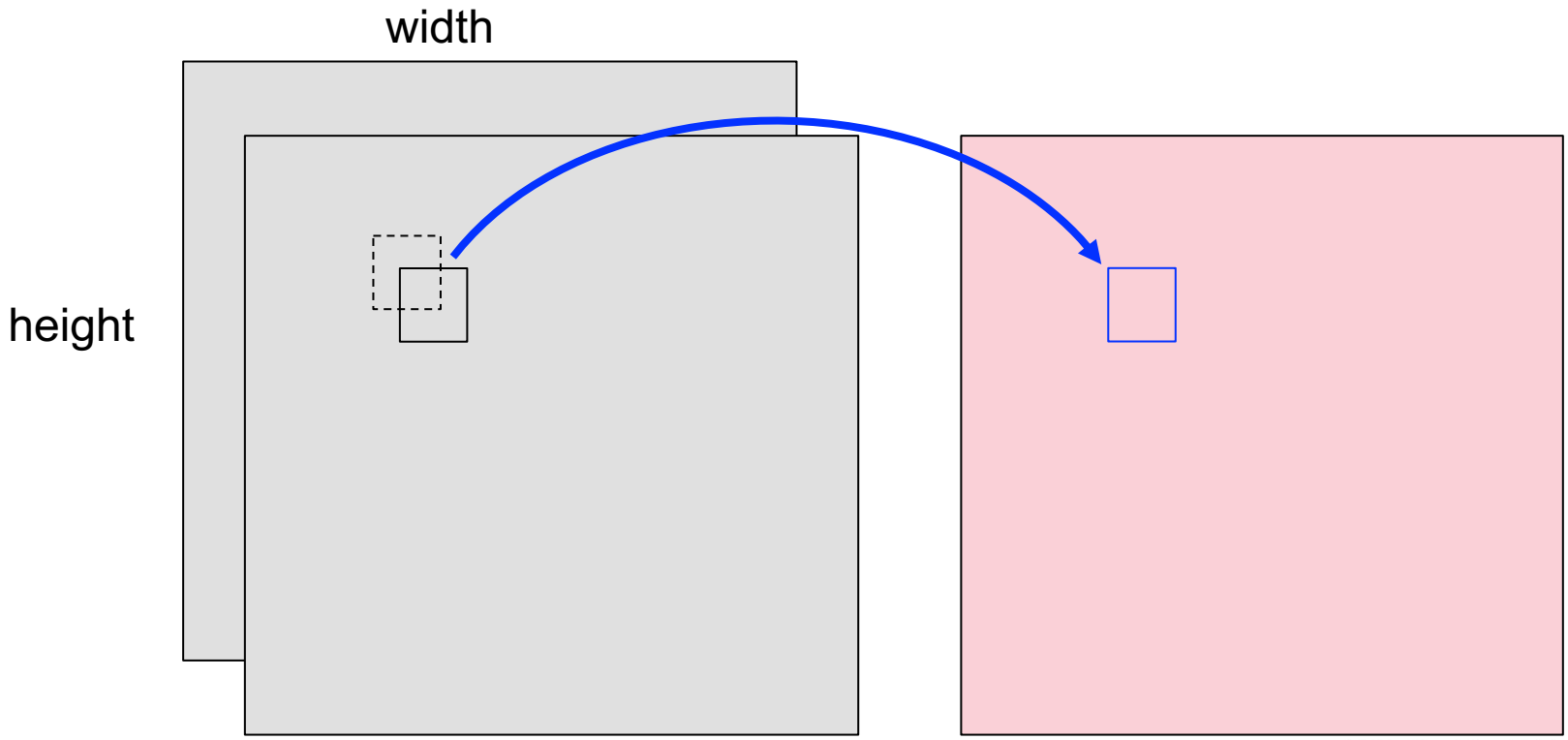



```
moving tile(3,1) -> tile(3,3)
```



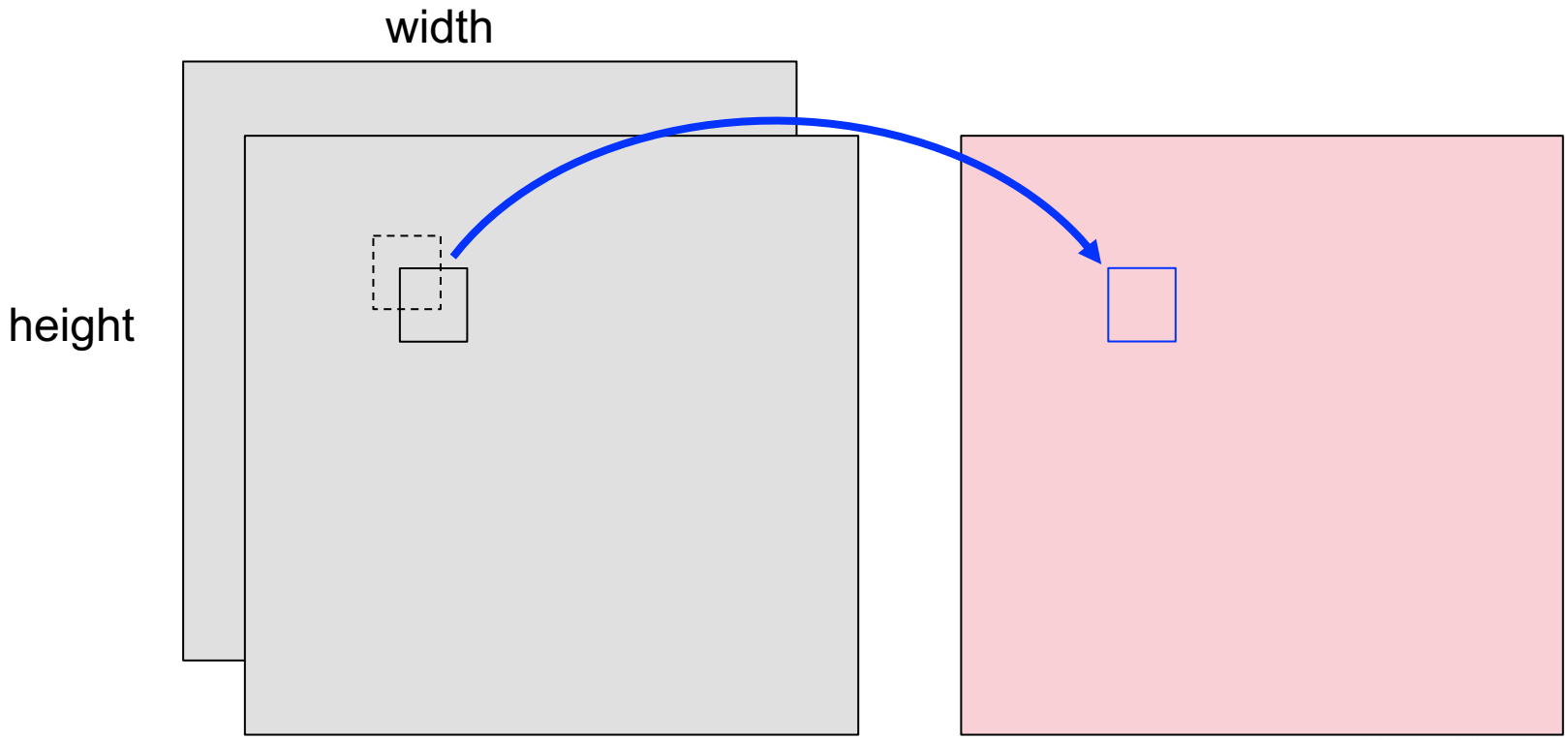
Manually Blending a Picture

- Idea: combine adjacent pixels from 2 images to create pixel in resulting image (e.g. 50%*pixel1 + 50%*pixel2)



Manually Blending a Picture

- Idea: combine adjacent pixels from 2 images to create pixel in resulting image (e.g. 50%*pixel1 + 50%*pixel2)



Blending

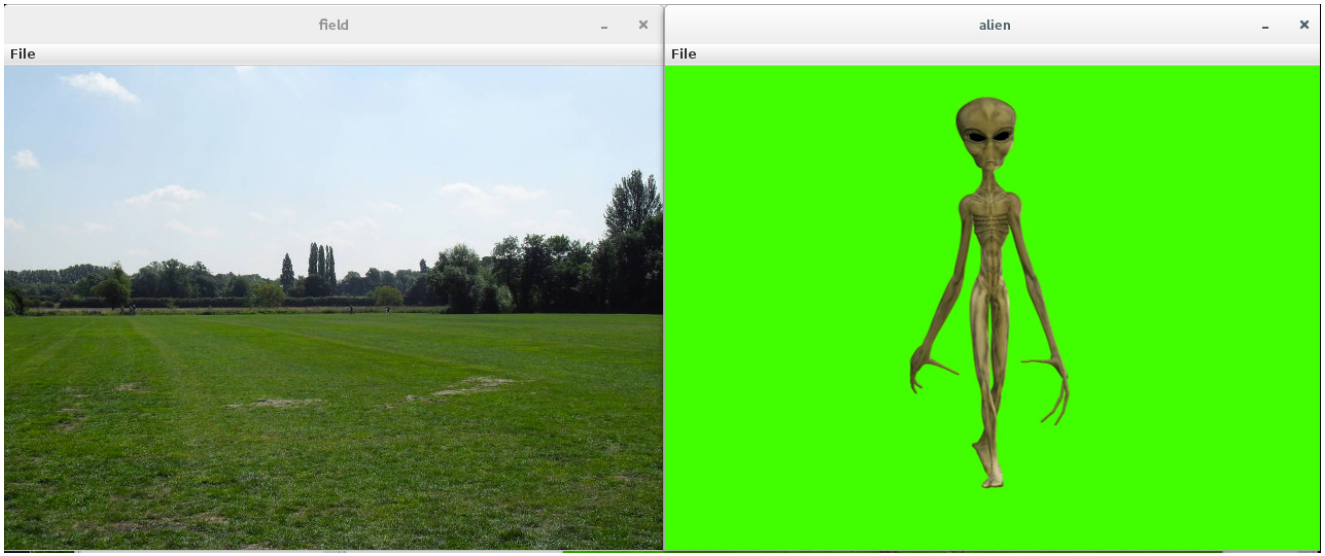
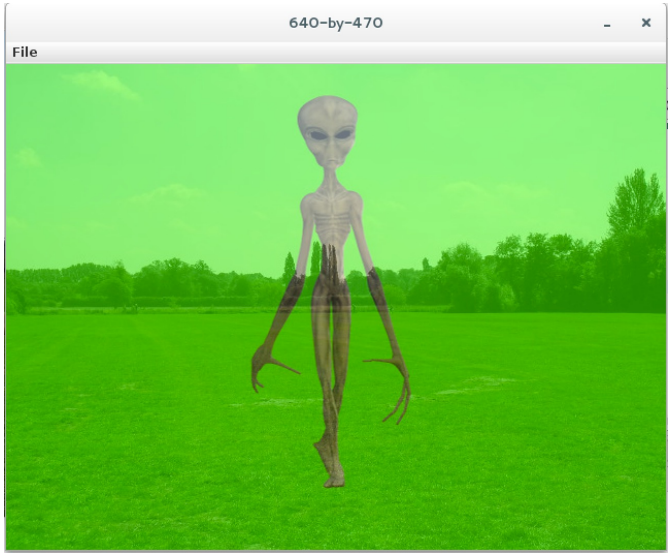


Image 1 Image 2



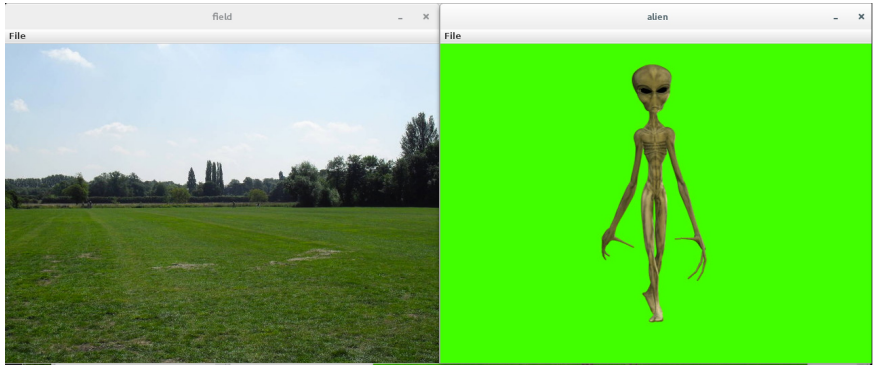
50% blend

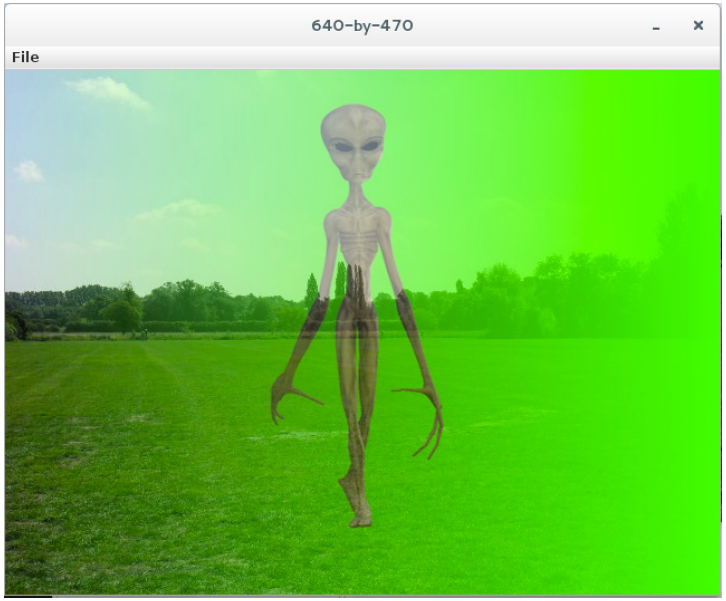
YORK UNIVERSITÉ UNIVERSITY

```
PImage blendImages(PImage img1, float p1, PImage img2, float p2) {

  PImage result = img1.copy();

  if (img1.width!=img2.width || img1.height!=img2.height)
    return null;

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

      int img1P = img1.get(i,j);
      int img2P = img2.get(i,j);

      float redP =   red(img1P)*p1 +   red(img2P)*p2;
      float greenP = green(img1P)*p1 + green(img2P)*p2;
      float blueP =  blue(img1P)*p1 +  blue(img2P)*p2;

      result.set(i,j,color(redP,greenP,blueP));

    }
  }

  return result;

}
```

Graded blend?

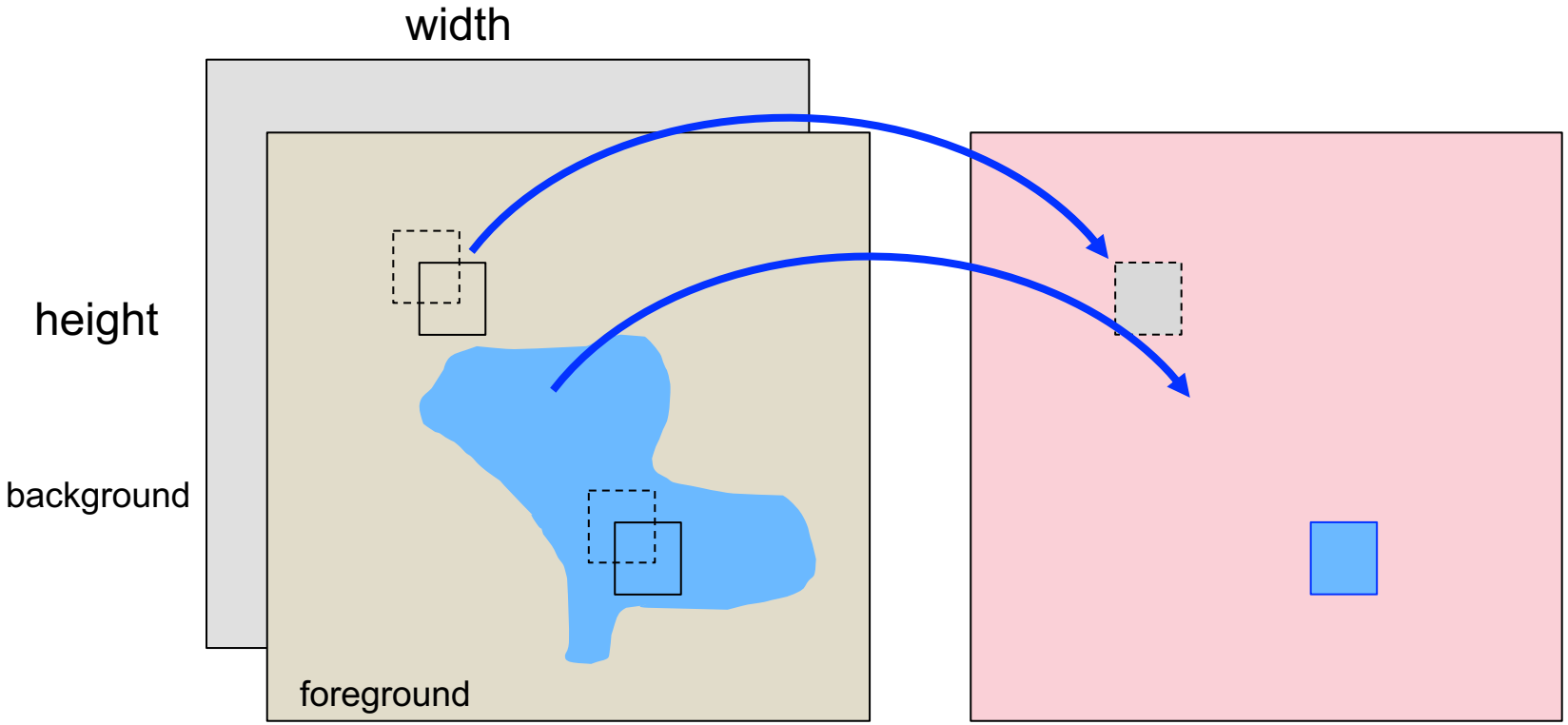
- How would you combine images to achieve this?

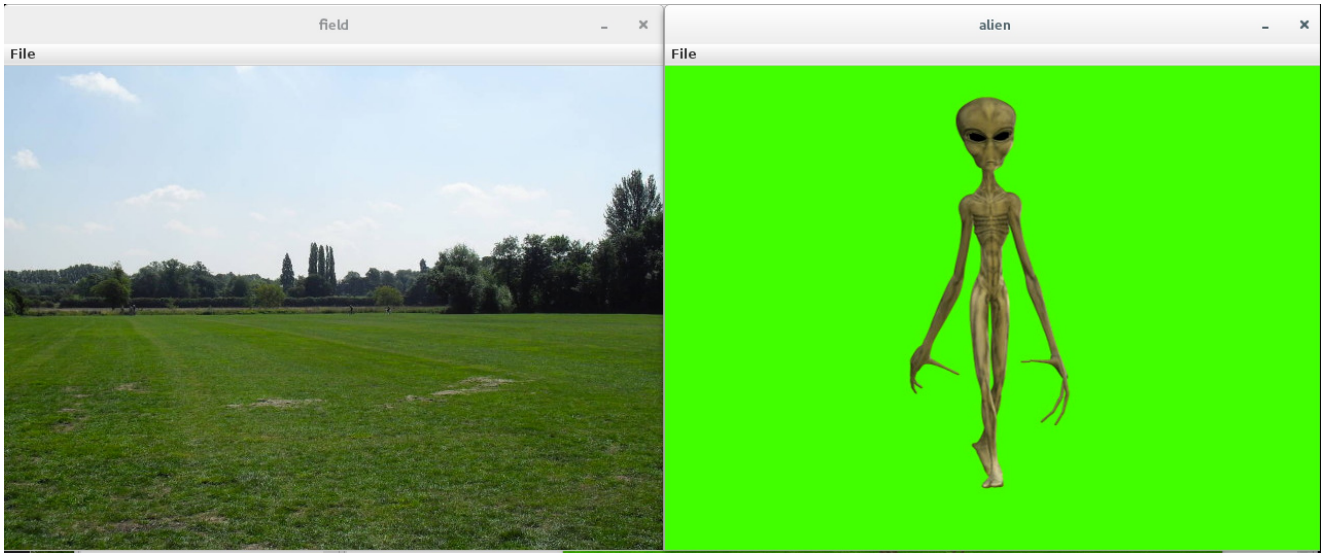




Chromakey a Picture

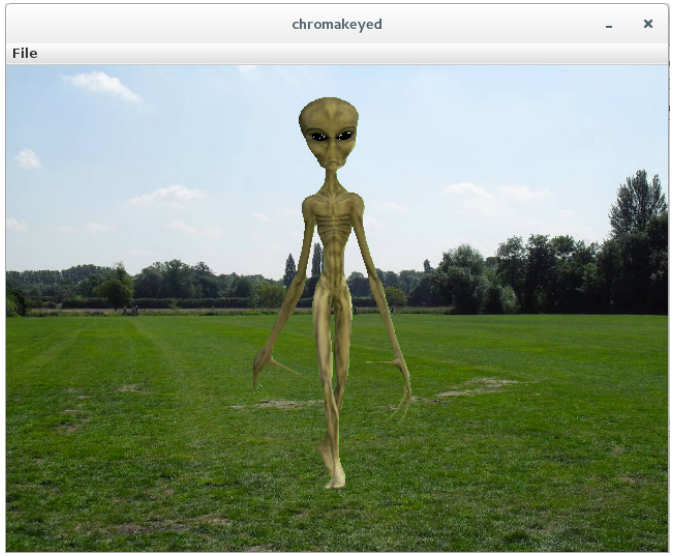
- Idea: for pixels in a region/satisfy a condition (green/blue), copy pixel from foreground image, otherwise copy background





background

foreground



Chromakey merge

YORK UNIVERSITÉ UNIVERSITY

Chromakey a Picture

```
PImage chromaKey(PImage imgB, float p1, PImage imgF, float p2) {

  PImage result = imgB.copy();
  if (imgB.width!=imgF.width || imgB.height!=imgF.height)
    return null;

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

      int imgBpix = imgB.get(i,j);
      int imgFpix = imgF.get(i,j);

      if ( green(imgFpix) > red(imgFpix) + blue(imgFpix) )  {
        result.set(i,j,imgBpix);
      }
      else {
        result.set(i,j,imgFpix);
      }

    }
  }

  return result;

}
```