



FALMOUTH
UNIVERSITY

COMP120: Workshop

3: Tinkering Graphics III

Learning Outcomes

- ▶ **Explain how** to duplicate parts of an image
- ▶ **Explain how** to identify the spaces and features of an image
- ▶ **Apply** mathematical knowledge to **write** computer programs that manipulate the spaces and features of an image
- ▶ **Implement** executable code that can 'tinker' graphics

Mirroring

- ▶ Mirroring is averaging across pixels
- ▶ A mirror will:
 - ▶ Define a dimension of inflexion
 - ▶ Duplicate pixels from the one side of the inflexion to the other, in reverse order



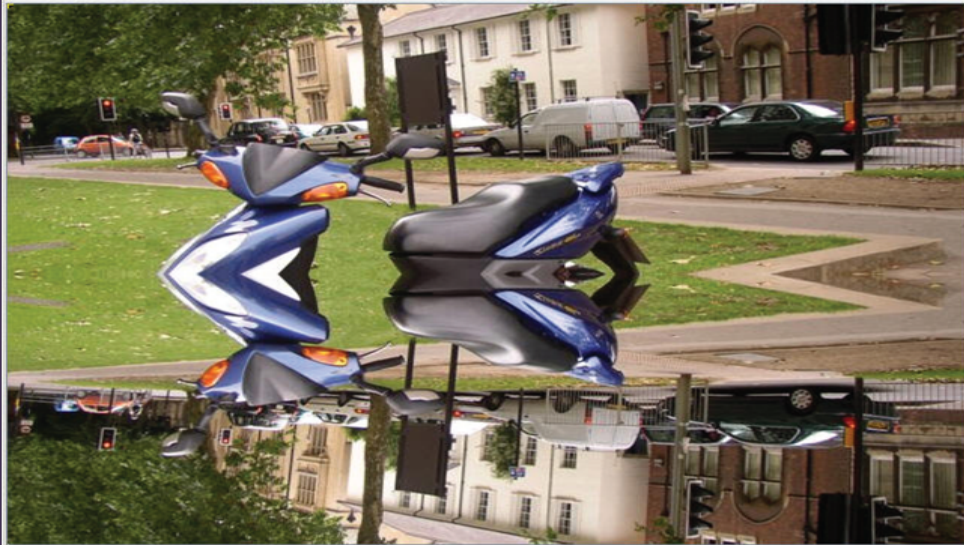
mediasources\blueMotorcycle.jpg



Zoom

X: 0 Y: 0

R: 73 G: 81 B: 32 Color at location:



Source Code: Mirroring (1)

```
def mirror_vertical(picture):  
    width = get_width(picture)  
    height = get_height(picture)  
    mirror_point = width / 2  
    for y in xrange(0, height):  
        for x in xrange(0, mirror_point):  
            left_pixel = get_pixel(picture, x, y)  
            right_pixel = get_pixel(picture, width - x - 1, y)  
            set_colour(right_pixel, get_colour(left_pixel))
```

Source Code: Mirroring (2)

```
def mirror_vertical(picture):  
    width = get_width(picture)  
    height = get_height(picture)  
    mirror_point = height / 2  
    for x in xrange(0, width):  
        for y in xrange(0, mirror_point):  
            top_pixel = get_pixel(picture, x, y)  
            bottom_pixel = get_pixel(picture, x, height - y - 1)  
            set_colour(bottom_pixel, get_colour(top_pixel))
```

Activity #1: Mirroring

In pairs:

- ▶ Use your function to repair the temple
- ▶ Post your repaired temple on Slack!



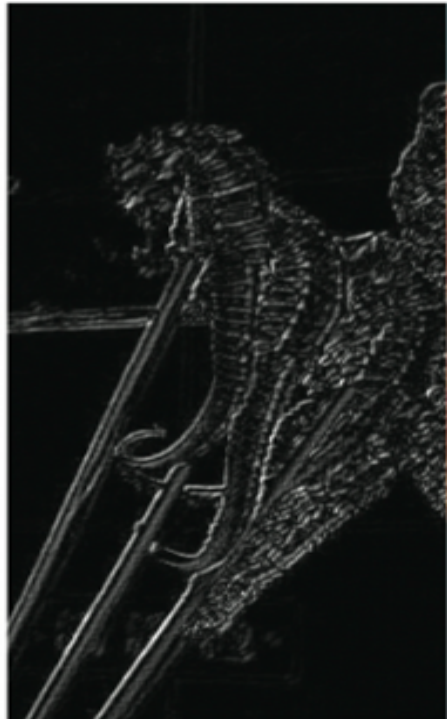
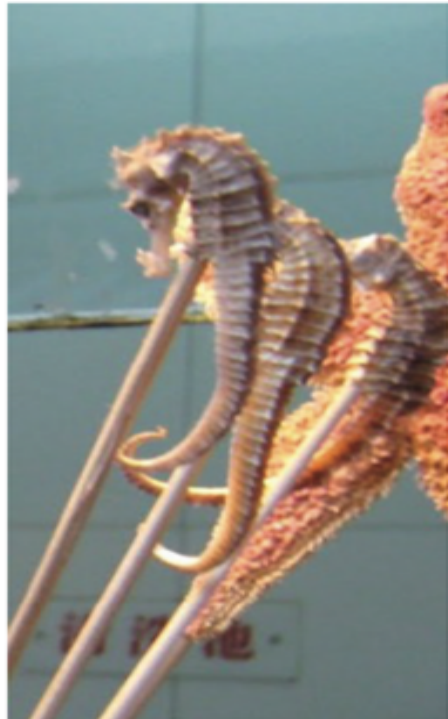
Edge Detection

- ▶ Blurring is averaging across pixels
- ▶ Edge detection is looking for *differences* between pixels:
 - ▶ We draw lines that our eyes see — where the luminance changes
- ▶ If the pixel changes left-to-right, or up-and-down, make a pixel black. Else, white.

Source Code: Edge Detection

```
def drawEdges (picture):  
    for x in xrange(0, picture.length - 1):  
        for y in xrange(0, picture.height - 1):  
            pixelSum = getRed(pixel) + getGreen(pixel) + ↵  
                        getBlue(pixel)  
            nextPixel = getPixel(picture, x+1, y+1)  
            nextPixelSum = getRed(nextPixel)+ getGreen( ↵  
                        nextPixel) + getBlue(nextPixel)  
            diff = abs(nextPixelSum-pixelSum)  
            newColor = makeColor(diff,diff,diff)  
            setColor(pixel, newColor)
```

Note: This source code excerpt will not work in PyGame.



Source Code: Edge Detection (1)

```
def drawBetterEdges (picture):  
    orig = makePicture (filename)  
    makeBw = makePicture (filename)  
  
    for x in xrange (0,getWidth (picture)-1):  
        for y in xrange (0,getHeight (picture)-1):  
            here=getPixel (makeBw,x,y)  
            down=getPixel (orig,x,y+1)  
            right=getPixel (orig,x+1,y)  
  
            hereL=(getRed (here)+getGreen (here)+getBlue (here) ←  
                ) /3  
            downL=(getRed (down)+getGreen (down)+getBlue (down) ←  
                ) /3  
            rightL=(getRed (right)+getGreen (right)+getBlue ( ←  
                right)) /3  
  
    ...
```

Note: This source code excerpt will not work in PyGame.

Source Code: Edge Detection (2)

```
...  
    if abs (hereL-downL)>10 and abs (hereL-rightL)>10:  
        setColor (here,black)  
    if abs (hereL-downL)<=10 and abs (hereL-rightL) <=10: ←  
        setColor (here,white)  
return makeBw
```

Note: This source code excerpt will not work in PyGame.

C:\ip-book\mediasources\...



C:\ip-book\mediasources\...



Activity #2: Edge Detection

In pairs:

- ▶ Setup a basic project in PyGame
- ▶ Refer to the following documentation
- ▶ Refactor the function: `drawBetterEdges (picture)` to use better variable names
- ▶ Test your solution

Background Subtraction

- ▶ Let's say that you have a picture of someone, and a picture of the same place (same background) without the someone there, could you subtract out the background and leave the picture of the person?
- ▶ Maybe even change the background?
- ▶ What we most need to do is to figure out whether the pixel in the Person shot is the same as the in the Background shot.
- ▶ Will they be the EXACT same colour? Probably not.

Source Code: Background Subtraction

```
def swapBackground(picture, back, newBack, tolerance):  
    for pixel in getPixels(picture):  
        x = getX(pixel)  
        y = getY(pixel)  
        backPixel = getPixel(back, x, y)  
        pixelColour = getColour(pixel)  
        backColour = getColour(backPixel)  
        if (distance(pixelColour, backColour) < tolerance):  
            newColour = getColour(getPixel(newBack, x, y))  
            setColor(pixel, newColour)
```

Note: This source code excerpt will not work in PyGame.





Problems

- ▶ We've got places where we got pixels swapped that we didn't want to swap
- ▶ We've got places where we want pixels swapped, but didn't get them swapped
 - ▶ Shirt stripes
 - ▶ Shadows
 - ▶ etc.

Activity #3: Background Subtraction

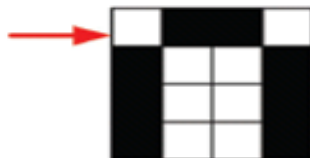
In pairs:

- ▶ Setup a basic project in PyGame
- ▶ Refer to the following documentation:
 - ▶ <http://www.cs.utah.edu/~michael/chroma/>
- ▶ Implement chroma key as a form of background subtraction
- ▶ Test your solution

Source Code: Collage

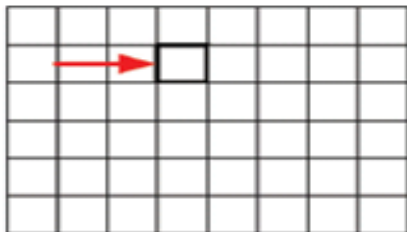
```
def copyBarb() :  
    # Set up the source and target pictures  
    barbf=getMediaPath("barbara.jpg")  
    barb = makePicture(barbf)  
    canvasf = getMediaPath("7inX95in.jpg")  
    canvas = makePicture(canvasf)  
    # Now, do the actual copying  
    targetX = 0  
    for sourceX in range(0,getWidth(barb)) :  
        targetY = 0  
        for sourceY in range(0,getHeight(barb)) :  
            color = getColor(getPixel(barb,sourceX,sourceY))  
            setColor(getPixel(canvas,targetX,targetY), color ←  
                )  
            targetY = targetY + 1  
        targetX = targetX + 1  
    show(barb)  
    show(canvas)  
    return canvas
```

source



sourceX=0
sourceY=0

canvas



targetX=3
targetY=1

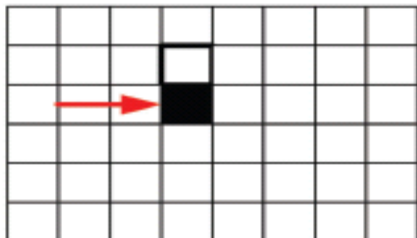
source



sourceX=0
sourceY=1

targetX=3
targetY=2

canvas



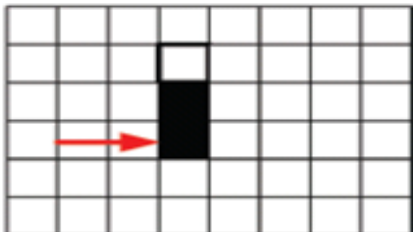
source



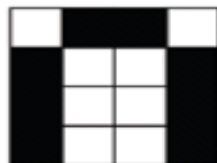
sourceX=0
sourceY=2

targetX=3
targetY=3

canvas



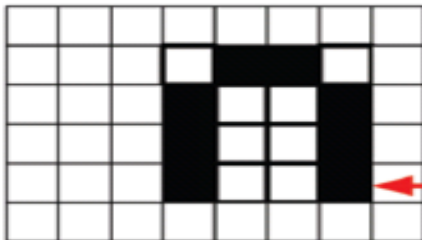
source



sourceX=3
sourceY=3

targetX=6
targetY=4

canvas





Activity #4: Collage

In pairs:

- ▶ Find some smaller images online
- ▶ Integrate the copy algorithm into your tinkering graphics project
- ▶ Create a collage of the images you found

Sprite Sheets and Animations

Review Al Swigart's pyganim python module:

<http://inventwithpython.com/pyganim/>

Activity #5: Sprite Sheets

In pairs:

- ▶ Find a sprite sheet online
- ▶ Integrate pyganim into your tinkering graphics project
- ▶ Animate something

What Next?

If you have implemented all of these algorithms, then use the rest of the workshop to:

- ▶ You now have a range of image manipulation algorithms at your disposal
- ▶ You only need to successfully implement and repurpose a set of these to do well on your Tinkering Graphics assignment!
- ▶ So, finish implementing the algorithms needed to complete your coursework
- ▶ Tidy up your code, ready for next session's peer-review activity
- ▶ Extend the code beyond the brief as appropriate to your particular game