Photo processing with `cs1media`

Otfried Cheong

1 First steps with `cs1media`

Suppose you have a photo taken with your digital camera on your computer. You can load this photo using:

```
from cs1media import *
img = load_picture("C:/photos/geowi.jpg")
```

(If you do not remember the filename of the photo, you can instead simply say `load_picture()` and use a file chooser to select your photo.)

You have now created a picture object and assigned the name `img` to it. You can get some interesting information about the picture using the methods `size()` (which returns a tuple consisting of the width and height of the picture in pixels) and `title()` (which typically returns the filename). If you call the method `show()`, then Python will start a program to display the picture:

```
>>> type(img)
<class 'cs1media.Picture'>
>>> img.size()
(500, 375)
>>> img.title()
'geowi.jpg'
>>> img.show()
```

2 Digital images

Digital images are stored as a rectangular matrix of tiny squares called *pixels* (picture elements). If the image has width `w` and height `h`, then there are `h` rows of pixels (numbered `0` to `h-1` from top to bottom) and `w` columns (numbered `0` to `w-1` from left to right).

The `cs1media` module represents each pixel as an *RGB color*, which is a triple of red, green, and blue light intensities. Each intensity is an integer between `0` (no light) and `255` (full light). So red is `(255, 0, 0)`, and yellow is `(255, 255, 0)`.

Many colors are predefined and can be accessed like this:

```
>>> Color.yellow
(255, 255, 0)
>>> Color.chocolate
(210, 105, 30)
```

To get a list of all predefined colors, use `help("cs1media.Color")` or simply `help(Color)` (if you have imported the `cs1media` module).

If you want to mix your own color, you can use:

```
>>> choose_color()
(150, 76, 121)
```

`choose_color()` returns the color triple you have chosen.

`cs1media` also contains a tool that is useful to find coordinates in a photograph (for instance to select a range to crop the photo) or to find the color of pixels you are interest in. You call it as

```
>>> picture_tool("C:/photos/geowi.jpg")
```

(or omit the filename to use a file chooser). If you click in the picture with the left mouse button, the tool displays the pixel's coordinate (X and Y) and the pixel's color (R, G, and B).

3 Color modifications

Looking at our photograph, it is rather dark. Let's make it lighter by increasing the light intensity of each pixel.

```
def make_lighter(img, factor):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      r, g, b = img.get(x, y)
      r = int(factor * r)
      g = int(factor * g)
      b = int(factor * b)
      if r > 255: r = 255
      if g > 255: g = 255
      if b > 255: b = 255
      img.set(x, y, (r, g, b))

img = load_picture("photos/geowi.jpg")
make_lighter(img, 1.5)
img.show()
```

You want to save a copy of the modified image? Just call

```
img.save_as("geowi_lighter.jpg")
```

When called as `make_lighter(img, 0.5)`, the function actually makes the picture *darker*.

With the same principle, we can create various image effects. For instance, we can create the sensation of a sunset by turning an image redder (by reducing the blue and green light intensity). We can create a negative image if we replace `(r,g,b)` by `(255-r,255-g,255-b)`:

To create a black-and-white image, we have to compute the "brightness" or *luminance* of each pixel. The total amount of light of a pixel seems to be `r + g + b`, but in fact this gives an unnatural result. The reason is that the human eye perceives blue to be much darker than green when the intensity of the light is the same. A good conversion to luminance is given by the following function:

```
def luminance(p):
  r, g, b = p
  return int(0.299*r + 0.587*g + 0.114*b)
```

Once we have the luminance `v` of a pixel, we can set it to the graytone pixel `(v,v,v)`.

Even though it's called "black and white", we have actually created graytone images. We can create images really using only black and white if we *threshold* the image: if the luminance is larger than some threshold, we set the pixel to white, otherwise to black:

4 Reflections

When you look in a mirror, you see yourself mirrored. We can achieve this effect:

```
def mirror(img):
  w, h = img.size()
  for y in range(0, h):
    for x in range(w / 2):
      pl = img.get(x, y)
      pr = img.get(w - x - 1, y)
      img.set(x, y, pr)
      img.set(w - x - 1, y, pl)
```

Instead of reflecting the entire photo, we can take the left half, reflect it, and use it as the new right half:

It is interesting to apply this technique to photos of human faces, because faces are symmetric, but not perfectly so, and the symmetrized version looks interestingly different from the real face.

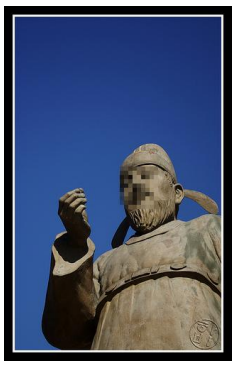
5 Pixelation

So far we modified every pixel of the photo independently. Let's see what we can do if we look at several pixels together. For instance, we can take the average color for a group of pixels:

```
def blocks(img, step):
  w, h = img.size()
  norm = step * step
  for y in range(0, h - step, step):
    for x in range(0, w - step, step):
      # compute average
      r, g, b = 0, 0, 0
      for x0 in range(step):
        for y0 in range(step):
          r0,g0,b0 = img.get(x+x0, y+y0)
          r, g, b = r+r0, g+g0, b+b0
      r, g, b = r/norm, g/norm, b/norm
      # set block
      for x0 in range(step):
        for y0 in range(step):
          img.set(x+x0, y+y0, (r, g, b))
```

This technique is often used to make part of an image unrecognizable, for instance car licence plates or faces on Google street view. To do that, we have to restrict the pixelation to a smaller rectangle of the photo:

```
def blocks_rect(img, step, x1,y1, x2,y2):
  w, h = img.size()
  for y in range(x1, x2 - step, step):
    for x in range(y1, y2 - step, step):
      # same as before
```



6 Blurring

If instead we compute a new color for each pixel by taking an average of a block around it, we can *blur* pictures. This is useful on pictures where pixels are visible because they have been magnified too much, etc. To implement this, we need two picture objects to work with, because we need the original pixel values to compute each new pixel:

```
def blur(img, radius):
  w, h = img.size()
  result = create_picture(w,h,Color.white)
  norm = (2 * radius + 1)**2
  for y in range(radius, h-radius):
    for x in range(radius, w-radius):
      # compute average
      r, g, b = 0, 0, 0
      for x0 in range(-radius,radius+1):
        for y0 in range(-radius,radius+1):
          r0, g0, b0 = img.get(x+x0, y+y0)
          r, g, b = r+r0, g+g0, b+b0
      r, g, b = r/norm, g/norm, b/norm
      result.set(x, y, (r, g, b))
  return result
```

7 Edge detection

The human visual system works by detecting *boundaries*. Even though natural objects like human faces have no sharp boundaries, we can easily recognize a child's line drawing of a face. When teaching a computer to recognize objects—an area called *computer vision*—one of the first steps is to recognize *edges* in the image.

A simple edge detection algorithm is the following: we compare the *luminances* of a pixel and the pixels to its left and above. If there is a big enough difference in both directions, then we have found an "edge" pixel. We can visualize this easily by drawing edge pixels black, and all other pixels white:

```
def edge_detect(img, threshold):
  w, h = img.size()
  r = create_picture(w, h)
  for y in range(1, h-1):
    for x in range(1, w-1):
      pl = luminance(img.get(x-1,y))
      pu = luminance(img.get(x,y-1))
      p = luminance(img.get(x,y))
      if (abs(pl - p) > threshold and
          abs(pu - p) > threshold):
        r.set(x, y, Color.black)
      else:
        r.set(x, y, Color.white)
  return r
```

8 Artificial images

So far we have modified pixel values based on the value of the pixels. We can also create entirely artificial images by setting pixels to whatever value we like. Let's first put our geese behind prison bars:

```
def prison(img, step):
  w, h = img.size()
  for y in range(0, h, step):
    for x in range(w):
      img.set(x, y, (0, 0, 0))
```

And now an entirely artificial image:

```
def interpolate(t, c1, c2):
  r1, g1, b1 = c1
  r2, g2, b2 = c2
  r = int((1-t) * r1 + t * r2)
  g = int((1-t) * g1 + t * g2)
  b = int((1-t) * b1 + t * b2)
  return (r, g, b)

def gradient(img, c1, c2):
  w, h = img.size()
  for y in range(h):
    t = float(y) / (h-1) # 0 .. 1
    p = interpolate(t, c1, c2)
    for x in range(w):
      img.set(x, y, p)

img = create_picture(100, 100)
gradient(img, Color.yellow, Color.red)
```

9 Cropping

We often need to transform photos in various ways. Cropping, for instance, means to cut out the part of the photo we are really interested in:

```
def crop(img, x1, y1, x2, y2):
  w1 = x2 - x1
  h1 = y2 - y1
  r = create_picture(w1, h1)
  for y in range(h1):
    for x in range(w1):
      r.set(x, y, img.get(x1+x, y1+y))
  return r

img = load_picture("photos/geowi.jpg")
r = crop(img, 67, 160, 237, 305)
```

We can use the `picture_tool` to find the coordinates for cropping.

Sometimes we are given a photo with a frame (like our statue above). We can leave it to the computer to find the right coordinates for cropping—all we need to do is remove the part of the photo that is similar in color to the frame color.

We first need a function to compare two colors:

```
def dist(c1, c2):
  r1, g1, b1 = c1
  r2, g2, b2 = c2
  return math.sqrt((r1-r2)**2 +
                   (g1-g2)**2 +
                   (b1-b2)**2)
```

The following function will then find the right `x1` coordinate for cropping:

```
def auto_left(img, key, tr):
  w, h = img.size()
  x = 0
  while x < w:
    for y in range(h):
      if dist(img.get(x, y), key) > tr:
        return x
    x += 1
  # image is empty!
  return None
```

We do exactly the same for the top, bottom, and right side of the photo, and can finally crop like this:

```
def autocrop(img, threshold):
  # use top-left color as key
  key = img.get(0, 0)
  x1 = auto_left(img, key, threshold)
  x2 = auto_right(img, key, threshold)
  y1 = auto_up(img, key, threshold)
  y2 = auto_down(img, key, threshold)
  return crop(img, x1, y1, x2, y2)
```

It turns out our statue actually has two frames, a black outer frame and a whitish inner frame:

```
r1 = load_picture("photos/statue.jpg")
r2 = autocrop(r1, 60)
r3 = autocrop(r2, 200)
```

10 Rotations

If we have taken a photo while holding our camera sideways, then we need to rotate the final photo:

```
def rotate(img):
  # swap w and h:
  h, w = img.size()
  r = create_picture(w, h)
  for y in range(h):
    for x in range(w):
      r.set(x, y, img.get(h - y - 1, x))
  return r
```

11 Scaling

Very often we want to have a photo in a different resultion than what we have taken. We create a new picture with the right size, and then need to compute the color of each pixel of the scaled image by looking at the pixels of the original image.

Here is a version that is similar to the pixelation idea we used above: we take the average color for a block of pixels from the original image.

```
def scale_down(img, factor):
  w, h = img.size()
  w /= factor
  h /= factor
  result = create_picture(w, h)
  norm = factor ** 2
  for y in range(h):
    for x in range(w):
      # compute average
      r, g, b = 0, 0, 0
      x1 = x * factor
      y1 = y * factor
      for x0 in range(factor):
        for y0 in range(factor):
          r0,g0,b0 = img.get(x1+x0,y1+y0)
          r, g, b = r+r0, g+g0, b+b0
      r, g, b = r/norm, g/norm, b/norm
      result.set(x, y, (r, g, b))
  return result
```

This function can only scale down by an integer factor. To scale down by other factors or to scale up needs a different way to decide the color of each pixel of the scaled image, possibly using interpolation to create a pleasing result.

12 Collage

A collage is a picture created by using (pieces of) existing pictures. For instance, let's start with this photo:

We'll take our statue above and paste it in at a suitable location:

```
def paste(canvas, img, x1, y1):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      canvas.set(x1+x,y1+y, img.get(x,y))
```

```
def collage1():
  trees = load_picture("trees1.jpg")
  r1 = load_picture("statue.jpg")
  r2 = scale_down(r1, 2)
  mirror(r2)
  paste(trees, r2, 100, 20)
  trees.show()
```

This works, but the result is not too exciting:

13 Chromakey

The background of the statue doesn't look natural. Can we get rid of this background? In this case, this is actually very easy, as the color of the background is very different from the color of the statue. We use the *chromakey* technique, used in the entertainment industry (`http://en.wikipedia.org/wiki/Chromakey`). Let's first try it by marking the statue's background in yellow:

```
def chroma(img, key, threshold):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      p = img.get(x, y)
      if dist(p, key) < threshold:
        img.set(x, y, Color.yellow)
```

Using the `picture_tool`, we can find a suitable key color `(41,75,146)` and a threshold of `70`.

Now all we need is an improved version of the `paste` function that will skip background pixels when copying an image onto the canvas. We need to pass the *color key* which we used for the background to the function:

```
def chroma_paste(canvas,img,x1,y1,key):
  w, h = img.size()
  for y in range(h):
    for x in range(w):
      p = img.get(x, y)
      if p != key:
        canvas.set(x1 + x, y1 + y, p)
```

Let's try this on our example:

```
def collage2():
  trees = load_picture("trees1.jpg")
  r1 = load_picture("statue.jpg")
  r2 = scale_down(r1, 2)
  mirror(r2)
  chroma_paste(trees, r2, 100, 20,
               Color.yellow)
  trees.show()
```

14 Concatenating photos

Let's make a comic strip out of a list of given pictures. We first have to compute the size of the result—its width is the sum of the width of the given pictures, its height is the maximum height of any picture. We then create a new picture of the correct size, and and copy the images at the right location one by one:

```
def concat(imglist):
  # compute height and width
  w = 0
  h = 0
  for img in imglist:
    w1, h1 = img.size()
    w += w1
    h = max(h, h1)
  r = create_picture(w, h, Color.white)
  x0 = 0
  for img in imglist:
    w1, h1 = img.size()
    for y in range(h1):
      for x in range(w1):
        r.set(x0 + x, y, img.get(x, y))
    x0 += w1
  return r
```

Here is how we can use this:

```
def make_strip():
  r1 = load_picture("geowi.jpg")
  r2 = load_picture("statue.jpg")
  r3 = load_picture("yuna.jpg")
  return concat([r1, r2, r3])
```