PROJECT #A: PIXELWISE OPERATOR

ABEL GOMES

This project aims at fostering the interest of students in image processing techniques. In practice, each student must to design and implement the project whose number matches the rightmost digit of his/her student number. For example, the student with the number 20768 must pursue the project labelled as A-8.

Project A-0 and Project A-5

This project aims at implementing the operator

`enlargeImage(s, inputImage)`

which enlarges the input image `inputImage` by some integer factor `s`. Enlarging an image is useful for magnifying small details in an image. There are various ways to enlarge a given image. We will use a simple method: to enlarge an image by a given factor `s`, we must replicate pixels such that each pixel in the input image becomes an $s \times s$ block of identical pixels in the output image. This technique is most easily implemented by iterating over pixels of the output image and computing the coordinates of the corresponding input image pixel.

Project A-1 and Project A-6

This project aims at implementing the operator

`shrinkImage(s, inputImage)`

which shrinks the input image `inputImage` by some integer factor `s`. Shrinking an image is useful, for example, to reduce a large image in size so that it fits on the screen. There are various ways to shrink a given image. Here, we will use a simple method: to shrink an image by a scale factor `s`, we must sample every `s`-th pixel in the horizontal and vertical dimensions and ignore the others. Again, this technique is most easily implemented by

Date: Assigned: November 03, 2013; Due: November 15, 2012 (in class); Visual Computing and Multimedia, Fall 2013-2014.

iterating over pixels of the output image and computing the coordinates of the corresponding input image pixel.

Project A-2 and Project A-7

This project aims at implementing the operator

```
reflectImage(flag, inputImage)
```

which reflects the input image `inputImage` along the horizontal or vertical directions (determined by the boolean `flag`). Reflection along either direction can be performed by simply reversing the order of pixels in the rows or columns of the image.

Project A-3 and Project A-8

This project aims at implementing the operator

```
translateImage(t, inputImage)
```

which translates the input image `inputImage` by some amount `t`. The translation process can be performed with the following equations:

$$r^* = r + t \tag{1}$$

$$c^* = c + t \tag{2}$$

where `t` is an integer. Note that displacement in the horizontal and vertical directions may be different.
Besides, there is a practical difficulty with the direct application of the translation equations (1) and (2). When translating, what is done with the“leftover” space? For example, if we move everything one row down, what do we put in the top row? One solution is to fill the top row with a constant value value (typically black (0) or white (255)). Can you think of a better solution?

Project A-4 and Project A-9

This project aims at implementing the operator

```
rotateImage(theta, inputImage)
```

which rotates the input image `inputImage` by some angle `theta`. The rotation process requires the use of the following equations:.

$$r^* = r.\cos(theta) - c.\sin(theta) \tag{3}$$

$$c^* = r.\sin(theta) + c.\cos(theta) \tag{4}$$

where `theta` is the angle of rotation (positive values correspond to counterclockwise rotation). Although the above formula is the basis of rotation, it only gets you halfway there because it will rotate an image about point (0,0). In most cases, what we really want is to rotate an image about its center. The following equations rotate an image about its center:

$$r^* = r_0 + (r - r_0).\cos(theta) - (c - c_0).\sin(theta) \tag{5}$$

$$c^* = c_0 + (r - r_0).\sin(theta) + (c - c_0).\cos(theta) \tag{6}$$

There are some practical difficulties implementing rotation using (3)-(4) or (5)-(6). Let us consider what happens to pixel (0,100) after a 90 degrees rotation using equations (3)-(4):

$$r^* = r.\cos(90) - c.\sin(90) = -100 \tag{7}$$

$$c^* = r.\sin(90) + c.\cos(90) = 0 \tag{8}$$

In this case, the pixel moves to coordinates (-100,0). This is clearly a problem since pixels cannot have negative coordinates.

Let us now consider what happens to pixel (50,0) after a 35 degrees rotation:

$$r^* = r.\cos(35) - c.\sin(35) = 40.96 \tag{9}$$

$$c^* = r.\sin(35) + c.\cos(35) = 28.68 \tag{10}$$

The coordinates calculated by the transformation equations are not integers, and therefore do not index a pixel in the output image.

The first problem can be resolved by testing coordinates to check that they lie within the bounds of the output image before attempting to copy pixels. A simple solution to the second problem is to find the nearest integers to r? and c? and use these as the coordinates of the transformed pixel.

Also note that the C++ math functions `cos()` and `sin()` require that the angle is given in radians (enter "man cos" or "man sin" from the command line to get a description of these functions). So, you need to convert degrees to radians.