Project 1: An Image Editing Program

# Command and Detail

Things in bold are category headings. The comments associated with each category apply to all the sub- operations. For instance, the comments associated with Filtering apply to all of the filtering operations.

Operation Keyword Arguments Details Points

Load load filename Load the specified image file and make it the current image. 0,

provided

Save save filename Save the current image to the specified file. 0,

provided Difference diff filename Subtract the given image file from the current image and put 0,

the result in the current image.

Run run filename Executes the script named filename. The script should

contain a sequence of other commands for the program, one per line. The script must end with a newline.

provided

0,

provided

Color to Grayscale

24 to 8 bit Color

gray Use the formula I = 0.299r + 0.587g + 0.114b to convert 5

color images to grayscale. This will be a key pre-requisite for many other operations. This operation should not affect alpha in any way.

All of these operations assume that the current image has 24 bits of color information. They should still produce 24 bit images, but there should only be 256 different colors in the resulting image (so the image could be stored as an 8 bit indexed color image). Don't be concerned with what happens if you run these operations on something that is already quantized. These operations should not affect alpha

- we will only test them on images with alpha = 1 (fully opaque images).

Uniform

quant-

Use the uniform quantization algorithm to convert the 5

Quantization unif

Populosity quant-

pop

current image from a 24 bit color image to an 8 bit color image. Use 4 shades of blue, 8 shades of red, and 8 shades of green in the quantized image.

Before building the color usage histogram, do a uniform 20

quantization step down to 32 shades of each primary. This gives 32 x 32 x 32 = 32768 possible colors. Then find the 256 most popular colors, then map the original colors onto their closest chosen color. To find the closest color, use the euclidean (L2) distance in RGB space. If (r1,g1,b1) and (r2,g2,b2) are the colors, use sqrt((r1-r2)^2 + (g1-g2)^2 + (b1-b2)^2) suitably converted into C++ code.

Dithering All of these operations should convert the current image into an image that only contains black and white pixels, with the exception of dither-color. If the current image is color, you should first convert it to grayscale in the range 0 - 1 (in fact, you could convert all images to grayscale - it won't hurt already gray images). We will only test these operations on images with alpha = 1.

Naive Threshold Dithering

Brightness Preserving

dither- thresh

dither- bright

Dither an image to black and white using threshold dithering 3 with a threshold of 0.5.

Dither an image to black and white using threshold dithering 7 with a threshold chosen to keep the average brightness constant

Threshold Dithering

Random Dithering

Clustered Dithering

Floyd- Steinberg Dithering

dither- rand

dither- cluster

dither- fs

Dither an image to black and white using random dithering. 5 Add random values chosen uniformly from the range

[-0.2,0.2], assuming that the input image intensity runs from 0 to 1 (scale appropriately). There is no easy way to match the reference program with this method, so do not try. Use either a threshold of 0.5 or the brightness preserving threshold - your choice.

Dither an image to black and white using cluster dithering 10 with the matrix shown below. The image pixels should be compared to a threshold that depends on the dither matrix below. The pixel should be drawn white if: I[x][y] >= mask[x%4][y%4]. The matrix is:

1. 0.7059 0.3529 0.5882 0.2353

2. 0.0588 0.9412 0.8235 0.4118

3. 0.4706 0.7647 0.8824 0.1176

4. 0.1765 0.5294 0.2941 0.6471

Dither an image to black and white using Floyd-Steinberg 15 dithering as described in class. (Distribution of error to four neighbors and zig-zag ordering).

Color Floyd- dither- Dither an image to 8 bit color using Floyd-Steinberg 10

Steinberg Dithering

color

dithering as described in class. You should use the color table corresponding to uniform quantization. That is, the table containing all colors with a red value of 0, 36, 73, 109,

146, 182, 219 or 255, green in the same range, and blue in the set 0, 85, 170, 255. If you do this, but not the grayscale version of Floyd-Steinberg, then you get 15 points.

Filtering All of these operations should modify the current image, and 15 for the

Box Filter filter- box

assume color images. The alpha channel should NOT be filtered. The alpha channel for all the test images will be 1 for all pixels, so you do not need to worry about the differences between filtering regular pixels or pre-multiplied pixels. Implement whichever approach you prefer.

Apply a 5x5 box filter.

first 3 for any additional

Bartlett Filter

Gaussian Filter

Arbitrary- Size Gaussian Filter

Image Resizing

filter- bartlett

filter- gauss

filter- gauss- n

Apply a 5x5 Bartlett filter. Apply a 5x5 Gaussian filter.

N (size) Apply an NxN Gaussian filter. Use the binomial method 10

presented in lecture to derive the filter values. Note that this is the same Gaussian you will use if you do the NPR paint task.

All of these functions should change the size of the current image by the appropriate amount. They should also operate on the alpha channel.

Half Size half Halve the image size. Use a Bartlett filter to do the

reconstruction. That means that for each output pixel (i,j) you place a 3x3 discrete filter at input pixel (2i,2j) and the filter is:

1. 1/16 1/8 1/16

2. 1/8 1/4 1/8

3. 1/16 1/8 1/16

Double Size double Double the image size. Use a Bartlett filter to compute the

reconstructed pixel values. There are four specific cases, depending on whether the desired output pixel is odd or even in x or y. Three of the cases are given here, the other can be derived from the last one given. If the output pixel

8, or nothing if you do scale

12, or nothing if you do scale

(i,j) has i even and j even, you apply the following filter at input location (i/2,j/2):

## 1. 1/16 1/8 1/16

2. 1/8 1/4 1/8

## 3. 1/16 1/8 1/16

If the output pixel (i,j) has i odd and j odd, you apply the following filter covering input locations (i/2-1,j/2-1) through (i/2+2,j/2+2) (integer division):

1. 1/64 3/64 3/64 1/64

2. 3/64 9/64 9/64 3/64

3. 3/64 9/64 9/64 3/64

4. 1/64 3/64 3/64 1/64

If the output pixel (i,j) has i even and j odd, you apply the following filter covering input locations (i/2-1,j/2-1) through (i/2+1,j/2+2) (integer division):

1. 1/32 2/32 1/32

2. 3/32 6/32 3/32

3. 3/32 6/32 3/32

4. 1/32 2/32 1/32

Arbitrary Uniform Scale scale

Arbitrary Rotation

If the output pixel (i,j) has i odd and j even, you do something very similar to above.

amount Scale the image up or down by the given multiplicative factor. By uniform scaling I mean scale the x and y axes by the same amount, so the aspect ratio does not change. Use Bartlett filters for the reconstruction. The reconstruction filter should be a Bartlett filter of width 4 pixels, so it always picks up 4x4 values in the input image (although some of these values may be multiplied by 0). Note this is the same filter size used for double and half size operations above. You can get 25 points for this if you did not do Arbitrary Rotation, but at most 35 points for the combination of this and Arbitrary Rotation. And if you do this you get no points for double and half, because they can be done in one line if you have this implemented.

rotate amount Rotate the image clockwise by the given amount, specified in degrees. The output image should be the same size as the imput image, with black pixels where there is no input image data. Use a 4x4 Bartlett filter for the reconstruction, as per the resizing operations above. You should note that the reconstruction process for this operation and scale is identical. You can get 25 points for this if you did not do Arbitrary Scale, but at most 35 points for the combination of this and Arbitrary Scale.

25 or 10

25 or 10

NPR Paint npr-

paint

1. Fundamental (15)

Apply a simplified version of Aaron Hertzmann's painterly rendering algorithm from the 1998 SIGGRAPH

[Paper Painterly Rendering with Curved Brush Strokes of Multiple Sizes. You need only implement the multiple](http://mrl.nyu.edu/publications/painterly98/hertzmann-siggraph98.pdf) (circular) brush size version from section 2.1 of this paper. A function to do the actual drawing of the circular strokes (TargaImage::Paint\_Stroke) has been provided for you.

To match the reference solution (which is what you're graded on), your implementation should use the brush size radii of 7, 3 and 1. When calling the Gaussian-blur function, use the filter constructed using the binomial coefficients with a filter size of

15 ~ 50

2 \* radius + 1

The fg parameter should be set to 1, and the threshold parameter T should be set to 25.

The difference function in Hertzmann's pseudo-code is simply Euclidean distance (as specified in the text below the paintLayer figure), so you'll need to compute and store these distances on a per-pixel basis.

* 1. Advance (15 ~ 50)

You can add stroke or other effects into NPR rendering and your score depends on how impressive your work.