Project 1: An Image Editing Program

Introduction

In this project you will write an image editing program that allows you to load in one or more images and perform various operations on them. Consider it to be a miniature Photoshop.

UI Operations

The operations are summarized here, with details on implementing them below.

Categories	Methods	DS
Transform	ToGray	5
Quantinzed	Uniform	5
	Populosity	20
Dithering	Naive	3
	Brightness	7
	Random	5
	Cluster	10
	Floyd	15
	Color Floyd	10
Filter	Box	15/3
	Barlette	15/3
	Gaussian	15/3
	Ab G	10
	Edge Detect	15/3
	Edge Enhance	15/3
Resizing	Half Size	8
	Double Size	12
	Arbitrary Size	25/10
	Arbitrary Rotate	25/10
	NPR	15~50

Grading

1. Please print 'Project1-Grading.doc' and mark the implemented items. We will verify the score using your grading sheet during the on-site demonstration.
2. In this project you score points for each image operation you correctly implement.
3. The possible operations and their values will be listed in the operation section.
4. The total number of points up for grabs is greater than 100, but the maximum any individual can get is 100 + 10. Please aim for 110.
5. A reference program will be provided so that you can check your implementation, although you may still get full points for an operation even if your result doesn't match the reference program's. Many of the operations are sensitive to very subtle differences in coding, and it is not worth anyone's time to

try to have everyone implement everything in exactly the same way. The operation descriptions below indicate the extent to which we think you should match the reference solution.

6. We will look for identical programs. If you are found to have duplicated someone else's work you will be treated as a group and given half the earned points. We will also take steps to ensure that you don't submit the reference program and don't manipulate the system in other ways we anticipate.

Submission

Please submit it to the indicated FTP site. TA will send you for further notification

Resources

Please get the resources on FTP. TA will send you for further notification

Basic Program

1. The basic program must be controllable through a scripting language, and you should not change this in any way. The project will be graded by running scripts, so the scripting interpreter must function. The scripting language is simply a sequence of lines, each of which has a command and some arguments, generally a filename. The commands for each operation are listed below along with the arguments. All arguments should be considered strings. A user should be able to enter a script in a window or load one from a file. This feature has been implemented in the skeleton program.
2. The program should maintain the current image, which is displayed. The current image is modified by the various operations as outlined below. The skeleton already contains operations which change the current image.
3. All files will be in the Targa (tga) format. LibTarga supports pre-multiplied RGBA images. To load the alpha bits, tell it that you are loading 32 bit data, and it will fill the alpha channel (with ones if necessary) along with the color information. When you read an RGBA image with LibTarga, it returns pre-multiplied alpha pixel data. You must divide out the alpha channel before display, taking care to avoid dividing by zero. The skeleton already does this for display.

Program Skeleton

We provide a skeleton program: Project1-Skeleton.zip and download lib from FLTK or pre-compiled Lib FLTK_1.3.5.zip (compile by VS 2019 X64).

You can download Project1_Solution_exe.zip to check your result.

This is for practicing the rendering engine which is kept using in my classes. You should modify the skeleton by changing the file TargaImage.cpp and/or TargaImage.h to implement the functions. At the moment all the functions change the current image to black.

1. NOTE: The TargaImage class stores RGBA. Many of the operations only need greyscale, so this is a waste. Ignore it for now by storing grey as RGB with R=G=B, or put separate greyscale image

information inside the TargaImage class.
2. NOTE: Many of the operations you need to implement are very similar. For instance, all the filter operations differ only in the filter mask, not the basic filtering algorithm. Write your program to take advantage of such common operations.
3. IMPORTANT: If you choose not to implement a function, it is essential that the function call ClearToBlack and return false. This is the default behavior, so if you don't change it, you should be OK.
4. ALSO IMPORTANT: For each member in your group, you must alter the function MakeNames in Main.cpp so that the function vsStudentNames.push_back is called with the member's name as the argument. We will be using this information during the grading process.

1. A basic program skeleton in ogre version with the scripting language implemented is available. The program provide the proper user interface to do the operations. In addition to the UI. We also provide a script system for you to run a sequence of operations.
2. A basic program skeleton with the scripting language implemented is available. This program will also load and save images with alpha (if present in the image).

1. As it is currently implemented, the skeleton will execute all the commands in a script file that is given as an argument. To specify arguments in Visual Studio, go to the Debug part of the project settings dialog. The skeleton also provides a single line command entry dialog. To execute a command, type it in and hit the Enter key. Hitting Enter again will run the same command again. You can of course change the command. Try "load test.tga" to load the test image. "save test-save.tga" also works. (Leave out the quotes when you type things in.)
2. The skeleton is slightly modular in design. In particular, the widget for displaying an image is separate from the object for storing the image, and both are separate from the main window itself.
3. There is a Makefile included in the program skeleton, and the skeleton should compile under Linux. You are welcome to make use of this if you like, but it is an unsupported feature of the project. You should not ask the TAs or the instructor questions about how to make your program run under Linux.

Supporting Programs

We provide the following programs:

1. A reference program that implements all the operations.
2. We do not support you doing this project under Linux, however, there is a Linux binary reference program that you can use in this unsupported capacity.
3. A program that shows targa files along with their alpha channel. Use this to test your compositing operations.

We also provide a whole range of example images, some with non trivial alpha channels.

Details on Things to Implement

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 the result in the current image.	0, provided
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.	0, provided
Color to Grayscale	gray		Use the formula I = 0.299r + 0.587g + 0.114b to convert color images to grayscale. This will be a key pre-requisite for many other operations. This operation should not affect alpha in any way.	5
24 to 8 bit Color			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 Quantization	quant-unif		Use the uniform quantization algorithm to convert the 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.	5
Populosity	quant-pop		Before building the color usage histogram, do a uniform 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.	20
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	dither-thresh		Dither an image to black and white using threshold dithering with a threshold of 0.5.	3
Brightness Preserving	dither-bright		Dither an image to black and white using threshold dithering with a threshold chosen to keep the average brightness constant.	7
Threshold Dithering				
Random Dithering	dither-rand		Dither an image to black and white using random dithering. 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.	5
Clustered Dithering	dither-cluster		Dither an image to black and white using cluster dithering 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	10

Floyd-Steinberg Dithering	dither-fs		Dither an image to black and white using Floyd-Steinberg dithering as described in class. (Distribution of error to four neighbors and zig-zag ordering).	15
Color Floyd-Steinberg Dithering	dither-color		Dither an image to 8 bit color using Floyd-Steinberg 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.	10
Filtering			All of these operations should modify the current image, and 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.	15 for the first 3 for any additiona l
Box Filter	filter-box		Apply a 5x5 box filter.	
Bartlett Filter	filter-bartlett		Apply a 5x5 Bartlett filter.	
Gaussian Filter	filter-gauss		Apply a 5x5 Gaussian filter.	
Arbitrary-Size Gaussian Filter	Filter-gauss-n	N (size)	Apply an NxN Gaussian filter. Use the binomial method 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.	10
Image Resizing			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	8, or nothing if you do scale
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 (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 If the output pixel (i,j) has i odd and j even, you do something very similar to above.	12, or nothing if you do scale
Arbitrary Uniform Scale	scale	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	25 or 10

			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.	
Arbitrary Rotation	rotatex	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
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 (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 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. 2. Advance (15 ~ 50) You can add stroke or other effects into NPR rendering and your score depends on how impressive your work.	15 ~ 50

Sample Results

Sample Results

You can use the reference program to generate sample images, and then use the difference operation to compare your results with the sample. The table below summarizes ways in which your results could reasonably differ from the reference program's. You can find these results in Sample Results.zip

Source image:wiz.tga

Operation	Test Images(s)	Notes
gray	gray.tga	You should be able to reproduce this exactly.
quant-unif	quant-unif.tga	You probably cannot re-produce this exactly. Your result should, however, show the same poor quality and color banding effects.
quant-popul	quant-pop.tga	You probably cannot re-produce this exactly. A populosity algorithm should do a reasonable job on the gray floor, and not too bad on the browns. It should, however, draw the blue ball as gray, because there are not enough blue pixels to be popular.
dither-thresh	dither-thresh.tga (threshold = 0.5)	You should be able to reproduce this almost exactly. Some pixels may be different around the boundaries between white and black.
dither-bright	dither-bright.tga	You should be able to reproduce this almost exactly. Some pixels may be different around the boundaries between white and black.
dither-rand	dither-rand.tga	You have no chance of reproducing this exactly. Instead, you should get an image that is similar in style but not identical.
dither-order	dither-order.tga	You should be able to reproduce this almost exactly. A few borderline pixels (those close to the threshold) may be different.
dither-cluster	dither-cluster.tga	You should be able to reproduce this almost exactly. A few borderline pixels (those close to the threshold) may be different.
dither-fs	dither-fs.tga	There's a good chance you can re-produce this exactly, but it is not essential. The character of your result should be similar.
dither-color	dither-color.tga	There's a good chance you can re-produce this exactly, but it is not essential. The character of your result should be similar.
filter-box	filter-box.tga	You may get different results around the boundary, but interior pixels should be identical. The reference program extended the size of the input image by reflecting it about its edges.
filter-bartlett	filter-bartlett.tga	You may get different results around the boundary, but interior pixels should be identical. The reference program extended the size of the input image by reflecting it about its edges.
filter-gauss	filter-gauss.tga (5*5 kernal)	You may get different results around the boundary, but interior pixels should be identical. The reference program extended the size of the input image by reflecting it about its edges.
filter gauss-n		The differences are the same for the 5x5 version of the gaussian filter.
filter-edge	filter-edge.tga	Edge detection
filter-enhance	filter-enhance.tga	Edge Enhance
half	half.tga	You may get slightly different results, particularly around the boundary.
double	double.tga	You may get slightly different results, particularly around the boundary
scale		You may get different results, but they should be qualitatively similar (no banding).
rotate	rotate_30.tga	You may get slightly different results, particularly around the boundary.
npr-paint	npr-paint.tga	This is a randomized algorithm, so it is very unlikely that your results will match the reference solution exactly (the reference solution operating twice on the same image is unlikely to match itself exactly). Your results should be

		qualitatively similar to the output of the reference solution, but need not be pixel-wise identical.