CS 355 Lab #6: Image Processing

Overview

For this lab, you will implement the following basic image processing operations:

- Brightness adjustment
- Contrast adjustment
- Noise removal / blurring (uniform averaging)
- Noise removal (median filtering)
- Sharpening (unsharp masking)
- Edge detection (gradient magnitude)

You will again use the shell you've used for Labs #2-3 and #5. This is solely so that when you're done you can have all of the parts you did in one program. *You will not be graded on whether all the parts from prior labs work, so long as what you have integrates with the new image processing operations.*

The drawing of the image should be done *behind* anything drawn with the 2D drawing tools or 3D rendering (if any). Think of the image as forming a backdrop that always stays behind the other drawing.

User Interface

In addition to the GUI elements from Labs #1-3 and #5, there is a new button that toggles display of the background image layer, much like the toggle button for the 3D graphic rendering layer in Lab #5. There are also new menu commands for the following operations:

- Open Image
- Save Image
- Save Picture
- Brightness
- Contrast
- Grayscale
- Blur (Uniform)
- Blur (Median)
- Sharpen
- Detect Edges

These menu commands, including any resulting dialog boxes, are handled by the shell. Your controller's interface will include methods for handling these operations based on parameters provided through these dialog boxes.

The "`Open Image`" menu command opens a file selection dialog box, allowing a user to select which file to open.

The "`Save Image`" menu command opens the same dialog, but it will save the background image to disk.

The "`Save Picture`" menu command saves whatever you have rendered to the drawing area. This doesn't require you to do anything to implement—whatever you render to the `Graphics2D` object will be saved as an image to disk. *You will not be graded on whether this works.*

The "`Brightness`" and "`Contrast`" menu commands open a dialog box allowing the user to enter the amount for adjustment. The contrast adjustment parameter will be in the range $[-100, 100]$, with 0 (no adjustment) as the default.

The remaining commands will not take any parameters and thus do not involve dialog boxes.

General Implementation

As mentioned already, you will be adding your code for this lab to the code from Lab #5. The model, view, and controller should be separate, as you did in Labs #1–3 and #5.

Model

For this lab, the shell provides the `CS355Image` abstract class. You may *not* edit this class, nor any other classes in its package; instead, you should create a subclass.

The `CS355Image` class has abstract methods for each image operation. You can choose whether to implement them in the subclass or in helper classes.

Be aware that some of these require creating other images as temporary results. For each operation, the new image should replace the stored image so that subsequent operations use the result of the current one.

You should implement the seven operations for this lab as follows. In addition, you should provide a method to convert your image to a Java `BufferedImage`, which your viewer will use to draw the image.

Brightness Adjustment

The brightness adjustment will be in the range [-100,100] and is applied additively. Make sure to convert the adjustment parameter to the range [-1.0,1.0.].

You should first convert each pixel to the HSB color space. To do that, you need to first get the pixel color through the `CS355Image.getPixel` method. Then call the `Color.RGBtoHSB` static method to convert the color.

If b is the the brightness adjustment parameter, then the level operation applied is

$$s = r + b$$

where as used in class, r denotes the input brightness and s denotes the output brightness.

Do the above operation solely on the brightness channel.

After applying this operation, convert pixels back to RGB. To do this, you should convert the HSB color by first calling `Color.getHSBColor`. Then get each channel from the `Color` and pass them to `CS355Image.setPixel`.

In general, your conversion code should look like this:

```
// Preallocate the arrays.
int[] rgb = new int[3];
float[] hsb = new float[3];

// Do the following for each pixel:

// Get the color from the image.
getPixel(x, y, rgb);

// Convert to HSB.
Color.RGBtoHSB(rgb[0], rgb[1], rgb[2], hsb);

// Do whatever operation you're supposed to do...

// Convert back to RGB.
Color c = Color.getHSBColor(hsb[0], hsb[1], hsb[2]);
rgb[0] = c.getRed();
rgb[1] = c.getGreen();
rgb[2] = c.getBlue();

// Set the pixel.
setPixel(x, y, rgb);
```

Contrast Adjustment

Rather than a straight linear operation, we will use a mapping similar to what Photoshop does. In particular, the contrast will be in the range [-100,100] where 0 denotes no change, -100 denotes complete loss of contrast, and 100 denotes maximum enhancement (8x multiplier). If c is the contrast parameter, then the level operation applied is

$$s = \left(\frac{c+100}{100}\right)^4 (r-128) + 128$$

For this operation, you should also convert to HSB, apply the operation to the brightness channel, and convert back to RGB.

Make sure you work in floating point, not integers. Integer division would not be very accurate.

Grayscale

Convert the image to HSB, zero the saturation channel, and convert back to RGB.

Blurring

This operation should blur the image using a 3×3 uniform averaging kernel.

The averaging should happen on all three channels in the RGB color space.

Median Filter

This operation applies a median filter to the image using a 3×3 neighborhood.

In this case, you should find the median of all three channels individually. Put them together to make the ”median color.”

At that point, treat the colors as vectors. Compute the squared distance from the median color to all nine colors from all nine pixels. The color you select should be the color with the least squared distance from the median color.

Sharpening

This operation sharpens the image using an unsharp masking kernel with $A = 2$ (i.e., a 6 in the middle and -1s for the four-connected neighbors, then divide by 2).

This operation should be done in the RGB color space.

Edge Detection

This operation computes the gradient magnitude for the image by first applying Sobel kernels to the image then combining the results into a gradient magnitude image. For the Sobel kernels, make sure to divide the result of the convolution by 8 before using them for the gradient computations.

You will want to convert to HSB and apply the operation to the brightness channel. At that point, you should convert the resulting value from a float in the range [0.0, 1.0] to an int in the range [0, 255] and use the result as the value for all three channels of RGB. The reason for this is because the resulting image will be grayscale.

Notes

The `CS355Image.getPixel` method allows you to pass in a pre-allocated array. If it is valid, the method will fill and return that array. Because this method will be called often in tight inner loops, it is *highly* recommended that you always provide a pre-allocated array that you can reuse on every iteration.

The `Color.RGBtoHSB` does the same thing, and again, it is *highly* recommended that you pre-allocate and reuse a float array for that. Unfortunately, there is no good alternative for `Color.getHSBColor`, which will allocate a new `Color` every time.

The `CS355Image` class provides a convenience method for switching the image to use a new buffer: `CS355Image.setPixels(CS355Image)`. Be aware that it does a *shallow* copy.

As always, *read the Javadocs*!

Controller

In addition to the event handlers your controller provided in Labs #1–3 and #5, you have 10 new methods to implement in the controller interface. Since these operations do not involve user interaction (other than invoking them through menus), there is no additional state information for the controller to store.

`public void toggleBackgroundDisplay()`

This should toggle off and on the display of the image layer, much like you did for the 3D rendering layer in Lab #5.

`public void openImage(File file)`

The shell will pass the image file to your controller. It should call `CS355Image.openImage(File)`, which has been provided for you. That method will open the file and populate the image with the data. (This is to save you from having to implement the I/O and image handling.)

`public void saveImage(File file)`

The shell will pass the image file to your controller. It should call `CS355Image.saveImage(File)`, which has been provided for you. That method will save the file out to disk. (This is to save you from having to implement the I/O and image handling.)

`public void doChangeBrightness(int brightnessAmountNum)`

Invokes the model's code for applying brightness adjustment to the image stored in the model.

`public void doChangeContrast(int contrastAmountNum)`

Invokes the model's code for applying contrast adjustment to the image stored in the model.

`public void doGrayscale()`

Invokes the model's code for turning the image to grayscale.

`public void doUniformBlur()`

Invokes the model's code for blurring the image stored in the model.

`public void doMedianBlur()`

Invokes the model's code for median filtering the image stored in the model.

`public void doSharpen()`

Invokes the model's code for sharpening the image stored in the model.

`public void doEdgeDetection()`

Invokes the model's code for applying edge detection to the image stored in the model.

View

When asked to draw the screen, your viewer should first draw the background image (if applicable), then draw the 2D shapes, and do the 3D rendering (again, if applicable). Each of these should be drawn to the same `Graphics2D` object passed to your view refresher.

To render the image to the `Graphics2D` object, the viewer should first query the model for the `BufferedImage`. Then, draw the `BufferedImage` to the 2048×2048 drawing area using the graphic object's `drawImage` method so that the center of the image corresponds to the center of the drawing area.

Make sure to set the `Graphics2D` object's `AffineTransform` to be the same as you use for the other layers (the viewing transformation you added in Lab #4) so that the scrolling and zooming works as in previous labs.

Restrictions

In addition to the restrictions on all previous labs, the code in this lab has the following restriction:

- You *cannot* modify any code in the `cs355.model.image` package.

Submitting Your Lab

To submit this lab, zip up your src directory to create a single new `src.zip` file, then submit that through Learning Suite. We will then drop that into a NetBeans project, copy in the originals for any files you are not allowed to modify (see Restrictions above and the Restrictions sections of the Lab 1 and Lab 5 specifications), do a clean build, and run it there.

If there are compile errors in your project because you modified files that you were not allowed to modify, we will grade your lab accordingly.

If you need to add any special instructions, please do so in the notes when you submit it. If you use any external source files, please make sure to include those as well.

Rubric

- Loading and displaying image (10 points)
- Brightness adjustment (10 points)
- Contrast adjustment (10 points)
- Greyscale (10 points)
- Uniform blurring (10 points)
- Median filtering (10 points)

- Sharpening (15 points)
- Edge detection (15 points)
- Otherwise correct behavior (10 points)

TOTAL: 100 points
