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 #1-3 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:

- Load
- Save
- Brightness
- Contrast
- 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" 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 pass to your view refresher will be saved as an image to disk. *You will not be graded on whether this works.*

The "Brightness" menu command opens a dialog box allowing the user to enter the amount for the adjustment. The brightness adjustment parameter will be a number in the range $[-255, 255]$, with 0 (no adjustment) as the default.

The "Contrast" menu command opens 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 add 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 model should store the current image (if any) in addition to your 2D shape model code from Lab #3 and the `HouseModel` from Lab #5. The model should store the current image as a new class that contains the following information:

- The height and width of the image
- The pixel data for the image as a 2D array of `int` values
- Any other information you may need in order to perform the supported operations

All of the image-processing operations should be done in the model. You may choose to implement them as methods of your image class (but be aware that some of these require creating other images as temporary results). Another option that is often done is to create separate classes with methods that implement operations on the image. For each operation, the new image should replace the stored image so that subsequent operations use the result of the current one.

All of the images we will use will be grayscale. (If you load a color image, the shell will automatically convert it to gray.) Your intermediate values may be floating point or go outside the [0,255] range of the image, but you should make sure to store only integer values clipped to this range.

You should implement the seven operations for this lab as follows. In addition, you should provide methods for the viewer to ask the model for image's height, width, and 2D array of `int` pixel values.

New Image

Replace the data stored in the model with that for the newly opened image.

Brightness Adjustment

The brightness adjustment will be in the range [-255,255] and is applied additively. 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.

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

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

Blurring

Blurs the image using a 3×3 uniform averaging kernel.

Median Filter

Applies a median filter to the image using a 3×3 neighborhood.

Sharpening

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).

Edge Detection

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.

Controller

In addition to the event handlers your controller provided in Labs #1–4 and #6, you have seven 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 #6.

```
public void doLoadImage(BufferedImage openImage)
```

The shell will open the file and verify that it is an image, then read it in and pass it to you as a `BufferedImage`. (This is to save you from having to implement the I/O and image handling.)

The `BufferedImage` passed to your controller will be a `TYPE_INT_RGB` one, but it will be "gray" in the sense that all of the three channels will be the same. You need to read from only one of the channels.

You should then use `getRaster` to get the `WritableRaster` from the `BufferedImage`, then iterate over the pixels in the image and use `getPixel` or `getSample` to populate a 2D array of `int` values for the pixels. Then pass this array to your model to store and use.

(This is meant only as a way to show you how it can be done. You are welcome to do it otherwise, but this is the general idea.)

`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 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 size of the image and a 2D array of `int` values for its pixels. The viewer should then create a new `BufferedImage` of the correct size, making sure to set the `ImageType` to `TYPE_BYTE_GRAY`. Then use `getRaster` to get this image's `WritableRaster`. Then use `setPixel` or `setSample` to set the pixels in the `WritableRaster` for the `BufferedImage`. Finally, draw the `BufferImage` 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.

Submitting Your Lab

To submit this lab, again zip up your src directory to create a single new `src.zip` file, then submit that through Learning Suite. If you need to add any special instructions, you can add them there in the notes when you submit it.

Rubric

This is tentative and may be adjusted up to or during grading, but it should give you a rough breakdown for partial credit.

- Loading and displaying image (10 points)
- Brightness adjustment (10 points)
- Contrast adjustment (10 points)
- Uniform blurring (15 points)
- Median filtering (15 points)
- Sharpening (15 points)
- Edge detection (15 points)
- Otherwise correct behavior (10 points)

TOTAL: 100 points

Change Log

- June 4: initial version for Spring 2015