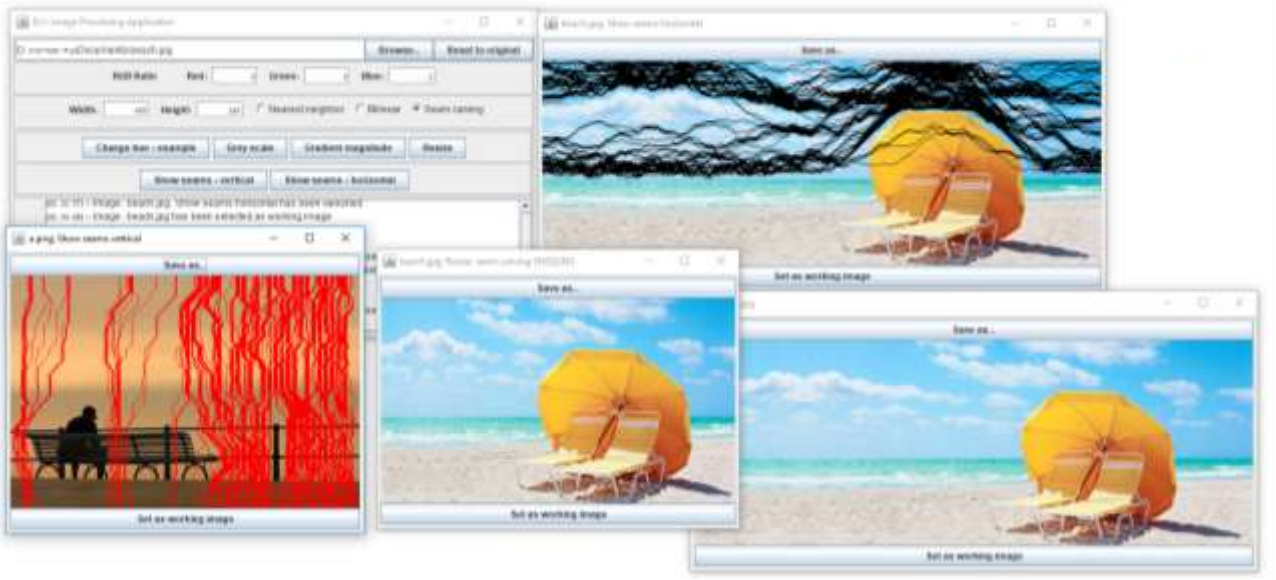
Computer Graphics ex1

The Interdisciplinary Center - spring 2020

Prof. Ariel Shamir

Submission date: **19th of April, 2020 (11:55 PM)**

Image Processing and Seam Carving



In this exercise you will implement and explore image resizing using the seam carving algorithm, along with other basic image processing operations.

To illustrate the different operations you are provided with an example working java application that illustrates these operations:

- Changing the image's hue – already implemented as an example.
- Conversion into grayscale.
- Resizing an image using two methods:
    - Nearest neighbor.
    - Seam carving.
- Object removal using seam carving

You are also provided with partial code, which you will need to complete to achieve an application that is similar to the given one.

What you should do

1. Install java (jdk) 11, see installation instruction in the following link.
2. Run the provided jar, which is a binary similar to what you are going to implement, and play around.
   If double click doesn't work, you can run the next command line:
   `java -jar SeamCarving.jar`
3. Read the partial code given in the src directory for this exercise. Understand what each class does and how.
4. You can use any kind of java IDE, but we will check your solution with eclipse. Make sure it works on eclipse before you submit.
5. Your responsibility is to fill in the missing pieces according to the functionality described below. **Implement** all TODO's in the code (search for the string "TODO" and replace it with your code). The final result should behave the same as the provided jar.
6. Submit your implementation in a zip file according to the submission guidelines below.

More specifically

- **Change hue – example:** This function is already implemented, explore its code and learn how to use the framework.
- **Greyscale:** Convert the image into grayscale, using the RGB weights that were entered in the application main window:

You can change those values to get different results. (Each value should be an integer $\in [0,100]$ and the total amount should be greater than zero).
Use the next formula to calculate the grey scaled image:

for each pixel p $\in$ input image do:

$$\text{greyColor} = \frac{p.red \times weights.red + p.green \times weights.green + p.blue \times weights.blue}{weights.red + weights.green + weights.blue}$$

set the greyColor as the output image's pixel.

- **Resize:** By clicking the "Resize" button, an invocation of the selected method will be performed.

The methods are:

- Nearest neighbor: Each new sized image pixel's color will be taken from the nearest appropriate pixel in the original image. Each image has different dimensions.

- Seam carving: *general*
  - Use the forward energy formulation.
  - You need to make sure you take into account the input mask. Meaning, ***you need to encourage seams to pass through regions that were masked by the user*** (more info about the masking panel will be given shortly).
  - You should implement only vertical seam carving (changing the image width). Horizontal seam carving will be implemented automatically by rotating the image first, then using the vertical seam carving, and finally, rotating the image back (these steps are already implemented in the partial code).
  - You should implement both image reducing and image increasing.
  - The code shouldn't run much slower than the supplied jar example.

- **Mask Image:** the user may specify regions where the seams should pass through. Pressing the "Mask Image" button in the main window will open the following window:

The user can control the brush size (see blue-highlight above), and may simply the desired region by painting it on the image (red-highlight). If the user presses the "Set Mask" button (green-highlight), this will automatically save the drawn mask to be later used in seam carving algorithm.
If the user presses the "Remove Object" button (yellow-highlight), then the highlighted region will be removed from the image while keeping the image dimension intact (see output below). This means that the output will have the same dimension as the original image, but all pixels from the highlighted region will be removed. More information about the object removal algorithm will be given shortly.

Seam carving

Algorithm

- Assume you need to handle k seams
- For each seam
  - Calculate the costs matrix. Remember the formula from class:

$$M_{y,x} = pixelEnergy(y,x) + \min \begin{cases} M_{y-1,x-1} + C_L(y,x) \\ M_{y-1,x} + C_V(y,x) \\ M_{y-1,x+1} + C_R(y,x) \end{cases}$$

  - The pixel energy should be the gradient magnitude. Use forward difference for calculating the derivative at a certain pixel. If this is not possible, that is, the pixel is at the right/bottom borders of the image, then use backward difference for calculating the derivative in the corresponding direction. **DON'T FORGET:** THE MAGNITUTED IS AN ABSOLUTE VALUE!
  - The pixel energy should be very small if the pixel is part of the "masked" region as specified by the input mask. You must do so by adding `Integer.MIN_VALUE` to the energy matrix at **the marked pixels only (pixels in which the mask entry is TRUE)**. This will encourage seams to pass through these regions.
  - Use dynamic programming to find the optimal seam. First find the smallest cost in the bottom row, then, travel back to the top along the path of minimal costs (invert the formula above).
  - Store the path of each seam in an appropriate data structure.
  - Remove the seam.
- In order to reduce the image's width by k, remove all the seams (paths) that you have stored in your data structure. It is important to make the appropriate modification to the image mask as well. This means that you need to also remove these seams from the input mask (see getMaskAfterSeamCarving() in the source code SeamCarver.java).
- In order to increase the image's width by k, duplicate all the seams (paths) that you have stored in your data structure (do so to the image mask as well).

Notes:

- Note the coordinates of the paths; you should make an appropriate transformation. Think how.
- The costs matrix may contain very large values.
  Remember that: `Integer.MIN_VALUE-1` is **positive**. You're advised to use matrix of `long` values.
- We will check your work manually; your results should look similar to ours (not identical, but close enough).

Object Removal:

In order to support object removal, you need to follow these steps:

1. Given an input image, of width $\boldsymbol{W}$, where we would like to remove pixels highlighted in the input mask.
2. While the current mask has true entries:
   2.1. Let $\Delta = \min\left(\frac{\boldsymbol{W}}{\boldsymbol{3}} - 1, \#Maximum\ number\ of\ true\ values\ in\ a\ row\right)$
   2.2. Remove Δ vertical seams using seam carving.
   2.3. Increase the image size to the original image's size.

Notes:

1. You only need to handle vertical seams. Therefore, you may simply remove vertical seams until the mask has all false values.
2. **[Important]** The provided SeamCarver class requires the output width in advance. This means that if you want to use SeamCarver, then you need to know in advance how much seams you want to remove. However, since we don't know the input mask in advance thus we don't know how much seams we should remove (why?). Instead, you can do seam carving in steps, where in each step you remove specific amount of seams until no further 'highlighted regions' are present in the modified mask (the mask after each seam carving invocation).
3. In order to speed the process, at each SeamCarver invocation, remove the maximum amount of true values that each row has in the current input mask. This is the minimum number of seams you will have to remove anyway (but not the actual amount). This value is denoted by Δ in step (2.1). Note: in some cases, the number of true values a row has may be large, thus we need o bound the number of seams by $\Delta < \frac{W}{3}$ (why?).
4. Make sure you always work with the modified mask.

General Tips

1. A gray color *g* can be coded as: `new Color(g, g, g).getRGB();`
2. **Make sure** you work properly when accessing 2D Arrays and Images. The indexing may differ between data structures.
3. When you remove a seam, all pixels to the right of it are shifted left by one. You will have to remember their original position. Use a helper array for that, e.g. an array with the size of the image, with the original column indices in each row:

```
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
becomes
0 1 3 4 5 6
0 1 2 4 5 6
0 1 2 4 5 6
```

4. At the sides (left/right), you don't have all three options for the costs. What you should do, is using the options you can.

   For example, at a leftmost pixel ($x = 0$), the cost would be:

$$M_{y,0} = pixelEnergy(y,x) + \min\begin{cases} M_{y-1,0} + C_V(y,0) \\ M_{y-1,1} + C_R(y,0) \end{cases}$$

   The case of the rightmost pixels is equivalent.
5. At the top row ($y = 0$), the cost would be:

$$M_{0,x} = pixelEnergy(0,x)$$

6. To print numbers to the console with a constant width, you can use:
   `System.out.format("%3d ", num);`
7. Explore the given code, learn how it works.
8. Java 11 supports functional programming paradigm (lambda expressions, method references, etc.). You're encouraged to use it in your code.

Bonus

These things can give you extra points, but you're on your own here. We won't answer questions regarding the bonus points.

1. (5pt) Implement the functionality of "Show seams" buttons.

Note that the implementation of bonus items has to be documented in an accompanying `readme.txt` file. **Undocumented items won't be graded.**

Submission Guidelines

Submit the "src" folder with all the java source files, **including** the files you didn't change.
REMEMBER: If you want the bonus, then also add the readme.txt file.
Your zip file should have the following name:
`<Ex##> <FirstName1> <FamilyName1> <ID1> <FirstName2> <FamilyName2> <ID2>`
For example: `Ex01 Bart Cohen-Simpson 34567890 Darth Vader-Levi 12345678`
Upload the file to the Moodle site.

Appendix I – Code

The source consists of the following classes:

`Main` – The entry point to the program. It is fully implemented.

`MenuWindow` – A class that represents the app's main window. It is fully implemented.

`ImageWindow` – A window that displays an image and allows you to save it as a PNG file. It is fully implemented.

MaskPainterWindow - An implementation of the mask painter window. This window allows to draw a mask on the input image.

`Logger` – An interface providing the log method(s), to print messages to the log field in the app's main window.

`FunctioalForEachLoops` – An abstract class that provides a convenient way of iterating over 2D arrays (also 1D arrays). It is fully implemented.

`RGBWeights` – A class that represents the weight of each color channel.

`ImageProcessor` extends **`FunctioalForEachLoops`** – A class that has some image processing methods:

- `logger` – A Logger type field that prints messages to the log field in the app's main window. You can use it the way you want.
- `changeHue()` – This is already implemented as an example. Test the implementation.
- `greyscale()` - Makes an image greyscale. You should implement this. The rgbWeights field should affect the result.
- `nearestNeighbor()` – Resize the image by using the nearest neighbor interpolation. You should implement this.

`SeamsCarver` extends **`ImageProcessor`** – A class that does seam carving. The rgbWeights field should affect the results., a

- `constructor` – Initializes some necessary fields. Not fully implemented. You are required to continue its implementation; initialize some additional fields, run preliminary computations such as: finding the k most minimal seams, storing their paths etc.
- `increaseImageWidth()` - Increases the image's width. You should implement this.
- `reduceImageWidth()` - Reduces the image's width. You should implement this.
- `getMaskAfterSeamCarving()` - Return the image mask after seam carving is applied on the image. The new returned mask, should contain the original values of the remaining pixels in case seam removal was applied. If pixels were replicated due to seam insertion, then you need to replicate the mask values of the replicated pixels as well.
- `showSeams(int seamColorRGB)` - Colors the seams pending for removal/duplication in the given argument. You should implement this as a bonus.

`MenuWindow - This class is almost implemented. You need to implement the following method in order to support object removal.`

- `removeObjectFromImage(boolean[][] srcMask) -` the method receives the input mask, and removes the object highlighted in this mask from the current working image. You need to implement this method in order to support object removal.

Good Luck!