

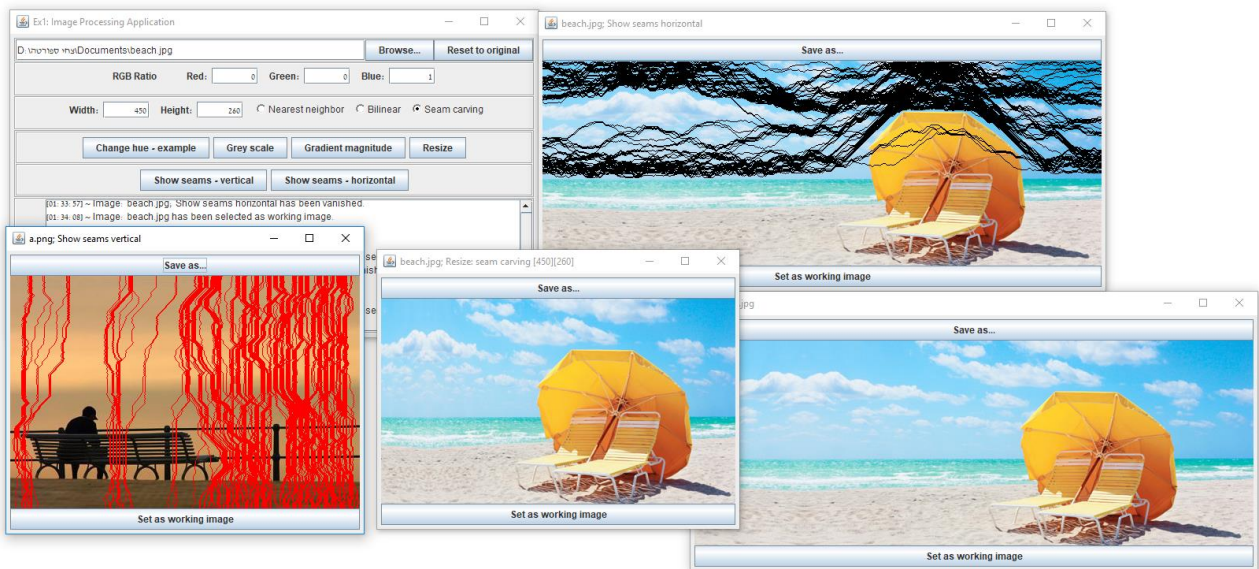
# Computer Graphics ex1

The Interdisciplinary Center 2018

Prof. Ariel Shamir

Submission date: Sunday April 15, 11:55pm

## 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.
- Calculation of gradient magnitude.
- Resizing an image using three methods:
  - Nearest neighbor.
  - Bilinear interpolation.
  - 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.

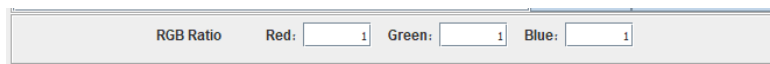
## What you should do

1. Install java ([jdk](#)) 9, if you are using macOS this [video](#) will guide you how to remove the older version of jdk.
2. Run ex1complete.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 ex1complete.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 in 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 ex1complete.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 was entered in the application main window:



RGB Ratio    Red:     Green:     Blue:

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).

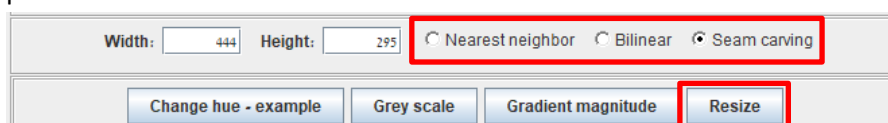
Use the next formula to calculate the grey scaled image:

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

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

set the greyColor as the output image's pixel.

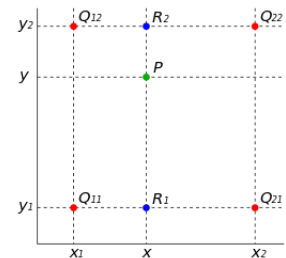
- **Gradient magnitude:** The gradient magnitude should be computed as:  $\sqrt{\frac{dx^2 + dy^2}{2}}$ , where  $dx$  is the difference between the current and next horizontal pixel, and  $dy$  is the difference between the current and next vertical pixel. Pay attention to the image borders conditions. Take the previous pixel instead of the next one. If the image dimensions are too small, throw an appropriate exception.  
**Note:** The gradient magnitude is based on the grey scaled image by using the given RGB weights.
- **Resize:** By clicking the "Resize" button, an invocation of the selected method will be performed.



Width:  Height:     ☐ Nearest neighbor    ☐ Bilinear    ☒ Seam carving

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. Think how to calculate the transformation.
- Bilinear interpolation: Each new sized image pixel's color will be calculated as a bilinear interpolation by using the four closest surrounding pixels of the original image. Each image has different dimensions. Think how to calculate the transformations. You can read about bilinear interpolation [here](#).
- Seam carving: general
  - Use the forward energy formulation.
  - 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.
  - You should implement both image reducing and image increasing.
  - The code shouldn't run much slower than the supplied jar example.



## Seam carving - algorithm

- Assume you need  $k$  seams
- For each seam
  - Calculate the costs matrix. Remember the formula from class:
 
$$M_{y,x} = \text{pixelEnergy}(y, x) + \min \begin{cases} M_{y-1,x-1} + C_L(y, x) \\ M_{y-1,x} + C_V(y, x) \\ M_{y-1,x+1} + C_R(y, x) \end{cases}$$
  - 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.
- In order to increase the image's width by  $k$ , duplicate all the seams (paths) that you have stored in your data structure.

### 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 `.MAX_VALUE+1` is **negative**. 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).

## General Tips

- A gray color  $g$  can be coded as: `new Color(g, g, g).getRGB();`
- 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	

- 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:

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

The case of the rightmost pixels is equivalent.

- At the top row ( $y = 0$ ), the cost would be:

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

- To print numbers to the console with a constant width, you can use: `System.out.format("%3d ", num);`
- Explore the given code, learn how it works.
- Java 9 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.

Zip it to a file called:

`<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.

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

**FunctionalForEachLoops** – 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 **FunctionalForEachLoops** – 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.

`gradientMagnitude()` - Calculates the magnitude of gradient at each pixel. 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.

`bilinear()` – Resize the image by using the bilinear interpolation. You should implement this.

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

`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.

`showSeams(int seamColorRGB)` - Colors the seams pending for removal/duplication in the given argument. You should implement this as a bonus.

## Good Luck!