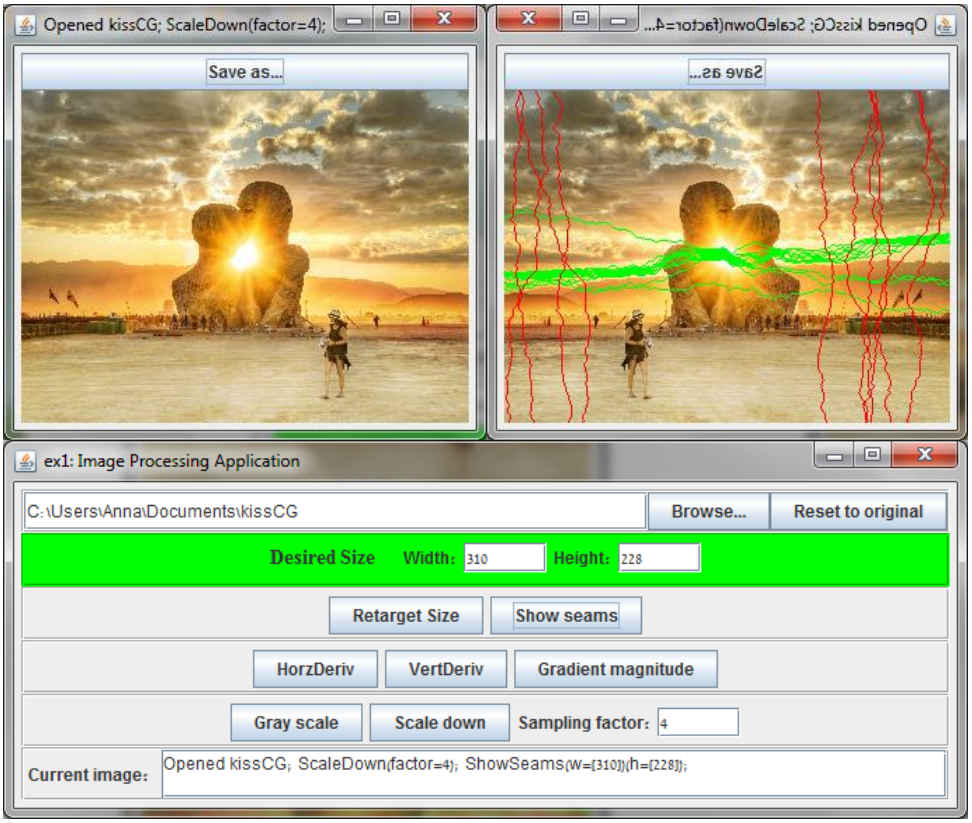
Computer Graphics ex1

The Interdisciplinary center 2015

Prof. Ariel Shamir

Submission date: Sunday March 15, 11:59pm

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:

- Seam carving, horizontal and vertical.
- Show Seams
- Conversion into grayscale.
- Calculation of magnitude of the gradient and directional derivatives (horizontal and vertical).
- Scaling the image down.

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

What you should do

Programming Section

1. In the bin directory, you can find ex1complete.jar, which is a binary similar to what you are going to implement. First, run it (you can execute `java -jar ex1complete.jar`), and play around.
2. Read the partial code given in the src directory for this exercise. Understand what each class does and how.
3. 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.
4. Submit your implementation in a jar file according to the submission guidelines below.

Note: The application always works on the current image which is the results of the last operation (except for the show seams operation, which doesn't change the image). This means you can do successive operations on an image. In some cases you need to reload the image to return to the original image. To do this you push the 'Reset to original' button. This has already been implemented in the given design.

Writing Section

Answer the following questions and submit a pdf- **do it after finishing the programming part, as you can use your application to check your answers**. Specifically, use show seams operation. You can also save the images and explore on your computer's images viewer.

1. Does the order of the retargeting operations make a difference on the output? Answer regarding the following scenarios:
    a. Keeping the orientation of the retargeting constant (either horizontal or vertical), compare a reduction by n1 seams and then a reduction by n2 vs. a reduction by n1+n2 seams at once.
    b. Keeping the orientation of the retargeting constant (either horizontal or vertical), compare enlargement by n1 seams and then by n2 vs. enlargement by n1+n2 seams at once.
    c. Compare reduction by n1 vertical seams, then by n2 horizontal seams vs. reduction by n2 horizontal seams first and then reduction by n1 vertical seams.
2. In your implementation for retargeting you used first one orientation and then the other. Show the seams for **reduction** by n1 seams in the first orientation **and** reduction by n2 in the second orientation (i.e. show the seams in both dimensions). Notice that one orientation has continuous seams, while the other has "holes". Why is that?
3. Similar to question 2, show the seams for retargeting both dimensions. This time perform **enlargement** by n1 seams in the first orientation **and** reduction by n2 in the second. Which seams have holes in this case? Why is it different from the case in question 2?

More specific instructions

- **GrayScale:** Convert the image to grayscale using one of the methods learnt in class of your choice.
- **Horizontal and vertical derivatives and Gradient:** The gradient magnitude can be computed as $\sqrt{dx^2 + dy^2}$, where dx is the difference between the current and next horizontal pixel, which is used for horizontal derivative, and dy is the difference between the current and next vertical pixel, which is used for horizontal derivative. Border conditions can be ignored (just put zero or trim).
- **Seam Carving** - *general*
  - You should implement both horizontal and vertical seam carving. Try to **avoid code duplication!**
  - You should implement both reduction and enlargement. You can assume that enlargement is allowed to be at most twice the original size.
  - You should implement a method to present the horizontal seams and vertical seams colored by different color
  - 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 cost matrix. Remember the formula from class:

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

    - Use dynamic programming to find the optimal seam. First find the smallest cost in the bottom row, then start going up on a path of minimal costs.
    - Remove the seam
  - Store the order and the pixels removed by each of the seams in a matrix.
  - To reduce image size by k pixels, remove all seams with order $\leq k$.
  - To enlarge by k pixels, duplicate all seams with order $\leq k$.

General Tips

- First implement only horizontal and only vertical retargeting
- 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.
- When retargeting both dimensions

    - Use retargeting in one orientation and then the other. There are many possibilities for orientation seam carving order (think how many for n1 x n2 seams?). This option is the easiest to implement.
    - To show the seams of the second orientation on the original image as well, you should use a similar helper array. The array remembers for each pixel in the retargeted image its position in the original image.
- At the image boundaries, you don't have all three options for the cost. What you should do is use the ones you have, and add the cost of the adjacent pixel. For example, at a leftmost pixel (index $x = 0$), the cost would be:

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

  This is done to avoid a bias for removing the border pixels.
- When you handle the sides, make sure that there is already a valid value in $M_{0,y}$ and $M_{w-2,y}$. To do this, you should first fill the cost in the middle, and only then at the borders.
- At the two upper corner pixels, you can't really calculate a cost. Put a value of 1000.
- Make a small test image that would make it easy for you to debug.
- To print numbers to the console with a constant width, you can use

```
System.out.format("%3d ", num);
```

Bonus

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

1. (5pt) Antialiasing – add a smoothing operation (to your choice) before scaling down. **Don't** submit images, but explain how the smoothing affected scaled down images.
2. (5pt) Multi-size image – add two buttons to the GUI in the main window that create a vertical or horizontal multi-size image. Also add a new slider that will control the output image size of these images (it will always control the last one created). When you move the slider the output image size adapts in real time. Note that you will need a preprocessing stage, which is not real-time (but still not much slower than the given implementation). The slider and multi-size image range should be between 50% to150% of the original image size.

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

Submission Guidelines

Submit all the java source files, including the files you didn't change and the pdf of the answer to the writing question.
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 files to the Moodle site!

Appendix I – Code

The source consists of the following classes:

`MainWindow` – The main application window and entry point to the program. It is fully implemented, except for bonus items.

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

`ImageProc` – A static class that has some image processing methods.

`scaleDown()` creates a reduced size image from the original image by taking every n^{th} pixel in x and y direction. n is given by the variable `factor`. This is a fully implemented example.

`grayScale()` converts an image to gray scale. **You should implement this**.

`verticalDerivative()` calculates the magnitude of the vertical derivative each pixel. **You should implement this.**

`horizontalDerivative ()` calculates the magnitude of the horizontal derivative at each pixel. **You should implement this.**

`gradientMagnitude()` calculates the magnitude of gradient at each pixel. **You should implement this.**

`retarget()` runs the seam carving algorithm to resize an image, use the **Retargeter class**. Y**ou should implement this**.

`showSeams()` colors the seams pending for removal/duplication. You can use here the helper array mentioned in the tips and the seam order matrix from the ***Retargeter class***. **You should implement this**.

`Retargeter` – A class that does seam carving.

`C'tor()` does all necessary preprocessing, including the calculation of the seam order matrix. **You should implement this**.

`retarget()` does the actual resizing of the image. **You should implement this**.

You can implement "get" methods for original position of the pixels in the retargeted image and seam order matrix for ***showSeams*** method of ***ImageProc class*** as suggested

Good Luck!