Assignment 7: Image Processing (Part 3)

Due Jun 26 by 8:59pm **Points** 0

LIME: Layered Image Manipulation and Enhancement (Part 3)

Things you can do with Swing: **code**

1 Additional Features

In this iteration of this project, you will build a view for your image processing application, featuring a graphical user interface. This will allow a user to interactively load, process and save images. The result of this iteration will be a program that a user can interact with, as well as use batch scripting from the previous iteration.

2 Graphical View

In this iteration you will build an interactive view with a graphical user interface.

2.1 Graphical user interface

Build a graphical user interface for your program. While the choices about layout and behavior are up to you, your graphical user interface should have the following characteristics, and obey the following constraints:

1. You must use Java Swing to build your graphical user interface. Besides the features from lecture, the provided code example illustrates some other features of Swing that you may find useful.
2. The user should see the image that is being processed on the screen. The image seen should be the topmost visible layer of the image. The image may be bigger than the area allocated to it in your graphical user interface. In this case, the user should be able to scroll the image.
3. The user interface must expose all the required features of the program (blurring, sharpening, greyscale, sepia, layer operations, saving and loading images, loading and executing a script from a file). You are not required to support interactive scripting.
4. When saving an image as a PNG/PPM/JPG, it should continue to save the topmost visible layer. This corresponds to what the user is currently seeing.
5. The user should be able to specify suitably the image to be loaded and saved that the user is going to process. That is, the program cannot assume a hardcoded file or folder to load and save.
6. When the user specifies an operation, its result should be visible to the user.

7. The user interface must expose all its features through menus. A subset of the operations may also be exposed in other ways (e.g. buttons, etc.). Proportions and placement of user interface elements must be reasonable (e.g. no oversized buttons, weirdly placed panels and regions, significant mis-alignment, etc.)
8. Each user interaction or user input must be reasonably user-friendly (e.g. making the user type the path to a file is poor UI design). We do not expect snazzy, sophisticated user-friendly programs. Our standard is: can a user unfamiliar with your code and technical documentation operate the program correctly **without reading your code and technical documentation?**

2.2 View and controller

Carefully design the interaction between a view and a controller, and formalize the interactions with view and controller interfaces. You may design a single controller that processes the batch-script input from the user, as well as the graphical user interface. Different controllers for different views are also possible if the views are very different from each other. However be mindful of the MVC principles and separation between the model, view and controller. When designing, always ask: "can I change one part with no/minimal changes to the others?"

3 Extra credit

The following two features are for extra credit. Both features will be equally weighted. You may complete one or both features.

The grading for extra credit will be as follows:

- 25% for extra credit 1 to work (all or none)
- 25% for the design of extra credit 1 (i.e. how well your solution is designed)
- 25% for extra credit 2 to work (all or none)
- 25% for the design of extra credit 2 (i.e. how well your solution is designed)

Please note that in order to qualify for any extra credit, you must score at least 50% on the manual grading of the required parts of the assignment. The threshold is deliberately kept at 50% and not higher: you will know from the quality of your submission if you stand lose to more than half your grade; you do not need to actually see your grade for it.

This is to discourage choosing to do extra credit as a substitute for the required work.

3.1 Image Downscaling

A basic operation on an image is to resize it (change its width, height or both). Upscaling an image (increasing the number of pixels in a row or column) is much more complicated because it requires creating data that is not there. Downscaling, on the other hand, is simpler. There are several ways to downsize an image, ranging from fast (but often more visual artifacts) to slow (but more visually pleasing).

Consider two rectangles $S(w,h)$ and $T(w',h')$. We create a mapping between points inside S and T . Specifically consider a location (x,y) in S . Its mapped counterpart is the location (x',y') in T , so that x is at the same proportion from the left and right edges of S as x' in T (and the same between y and y'). For example, if $x=0.3w$ then $x'=0.3w'$. The relation between x and x' is described by the following equation:

$$\frac{x'}{w'}=\frac{x}{w}$$

Similarly,

$$\frac{y'}{h'}=\frac{y}{h}$$

We can use this to resize an image of the same dimensions as S into an image of the same dimensions as T as follows: map each pixel location from T to the corresponding location in S using the math given above. Then use the color from S at the mapped location to compute the color of the pixel in T .

One practical problem is that an integer pixel location in T may be mapped to a floating-point location in S , but colors are available only at integer locations. One solution is to round the values to the nearest integer and copy the color over. This solution, although fast, may result in visual artifacts of some images. A better solution is to consider the four integer pixel locations around a floating-point location (by rounding up and down x and y values). The colors at these four pixels is used to compute the color at the required location.

Specifically, let the pixel location be $P(x,y)$ where both are floating-point values. Consider the pixels $A(\lfloor x\rfloor,\lfloor y\rfloor)$, $B(\lceil x\rceil,\lfloor y\rfloor)$, $C(\lfloor x\rfloor,\lceil y\rceil)$ and $D(\lceil x\rceil,\lceil y\rceil)$. Given values c_a , c_b , c_c , c_d at A , B , C , D respectively, we compute the value c_p at P can be computed as follows:

$$m=c_b(x-\lfloor x\rfloor)+c_a(\lceil x\rceil-x)$$

then,

$$n=c_d(x-\lfloor x\rfloor)+c_c(\lceil x\rceil-x)$$

and finally,

$$c_p=n(y-\lfloor y\rfloor)+m(\lceil y\rceil-y)$$

These computations are applied three times for the red, green and blue components of the colors, to get the complete color.

Implement the image downsizing operation for your image. To maintain the invariant that all layers of an image have the same dimensions, downsizing an image should downsize all its layers.

A complete successful attempt for this extra credit should include:

- An original image (PNG or JPG), and two downsized versions of that image (PNG or JPG). At least one of these versions should have different ratio of width:height than the original image. As before, you are not allowed to use any images provided to you as part of these assignments. All images must be cited in your README file.
- Image downsizing is exposed suitably in the graphical user interface. Supporting it through the text view from the previous assignment is not required.
- The design for this feature is well incorporated into your design of the overall application.
- A section in your README file briefly explaining the different parts of your program that changed to implement this feature.

Note: even this algorithm may produce some visual artifacts, if the downsizing is drastic. Practically, image downsizing is implemented by considering each pixel in the destination image to be a rectangle, and mapping it to a larger rectangular portion in the source image and combining colors of all pixels within that rectangle. You are not expected to implement this, but you may to see how much better it looks!

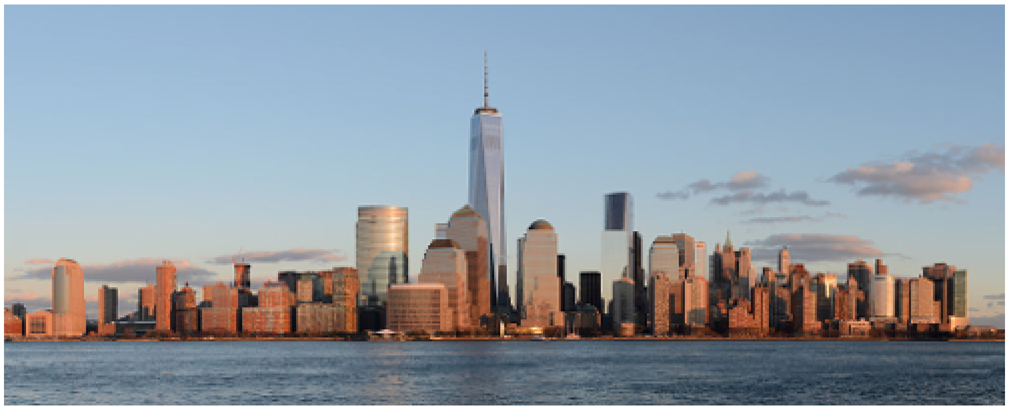
Clarification: In the above math, floor and ceiling are the previous and next integers respectively. So, if the pixel location's x-value is 10, then $\lfloor x \rfloor$ is 10, and $\lceil x \rceil$ is 11.

3.2 Image mosaicing

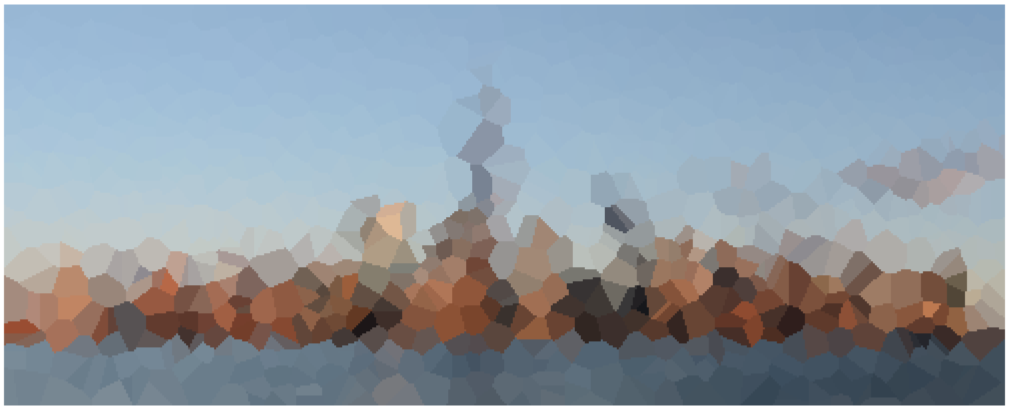
Many image processing applications support an effect that gives an image a "stained glass window" effect. Stained glass windows are popular in many ancient architectures, especially older churches. They create pictures by joining smaller irregularly-shaped pieces of stained glass.

An image can be "broken down" into such stained glass pieces, by choosing a set of points in the image (called seeds). Each pixel in the image is then paired to the seed that is closest to it (by distance). This creates a cluster of pixels for each seed. Then the color of each pixel in the image is replaced with the average color of its cluster. This may be thought of as an "easier cousin" of k-means clustering, a popular clustering algorithm from machine learning.

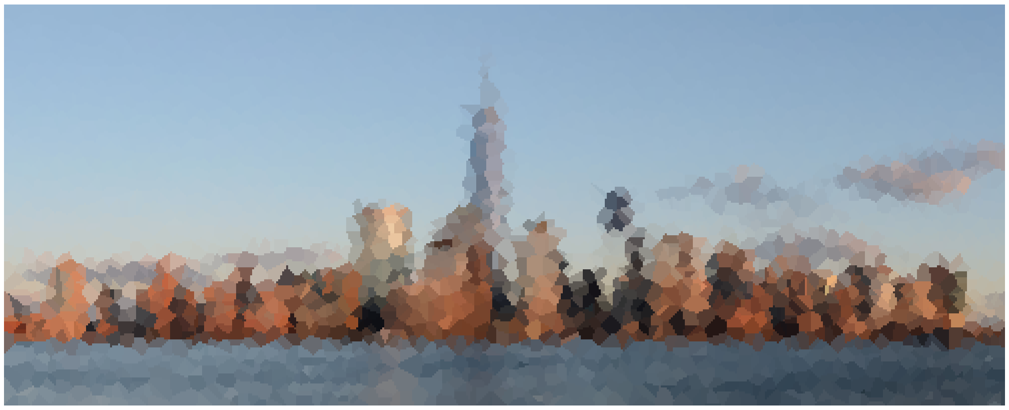
The seeds can be chosen in a number of ways. The simplest method (that you will implement) is to choose the seeds randomly (i.e. choose random pixel locations from the image). The picture below illustrates the effect of mosaicing.



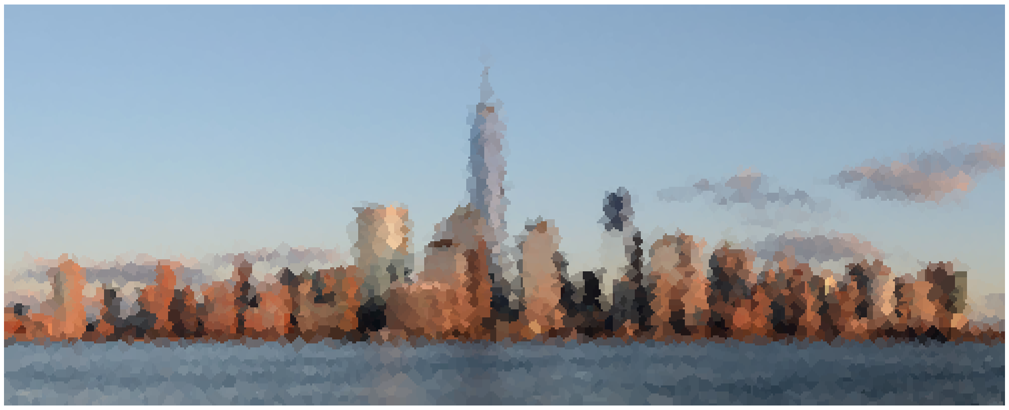
"Original image"



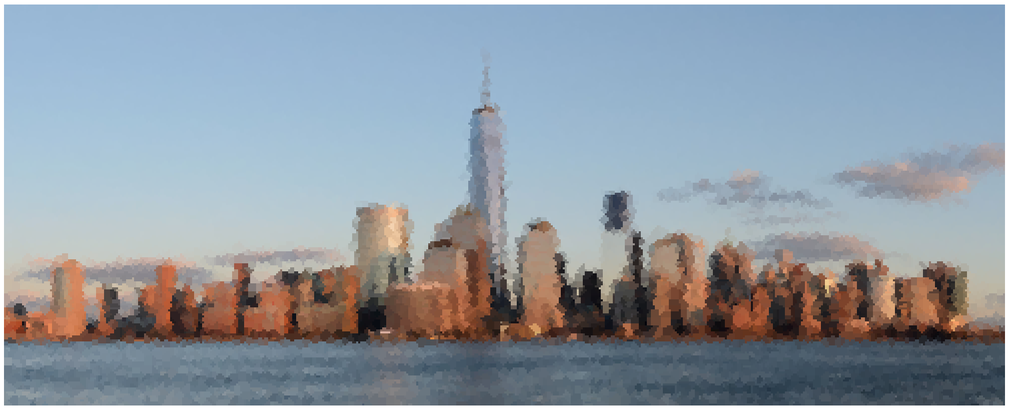
"Mosaic with 1000 seeds"



"Mosaic with 4000 seeds"



"Mosaic with 8000 seeds"


"Mosaic with 15000 seeds"

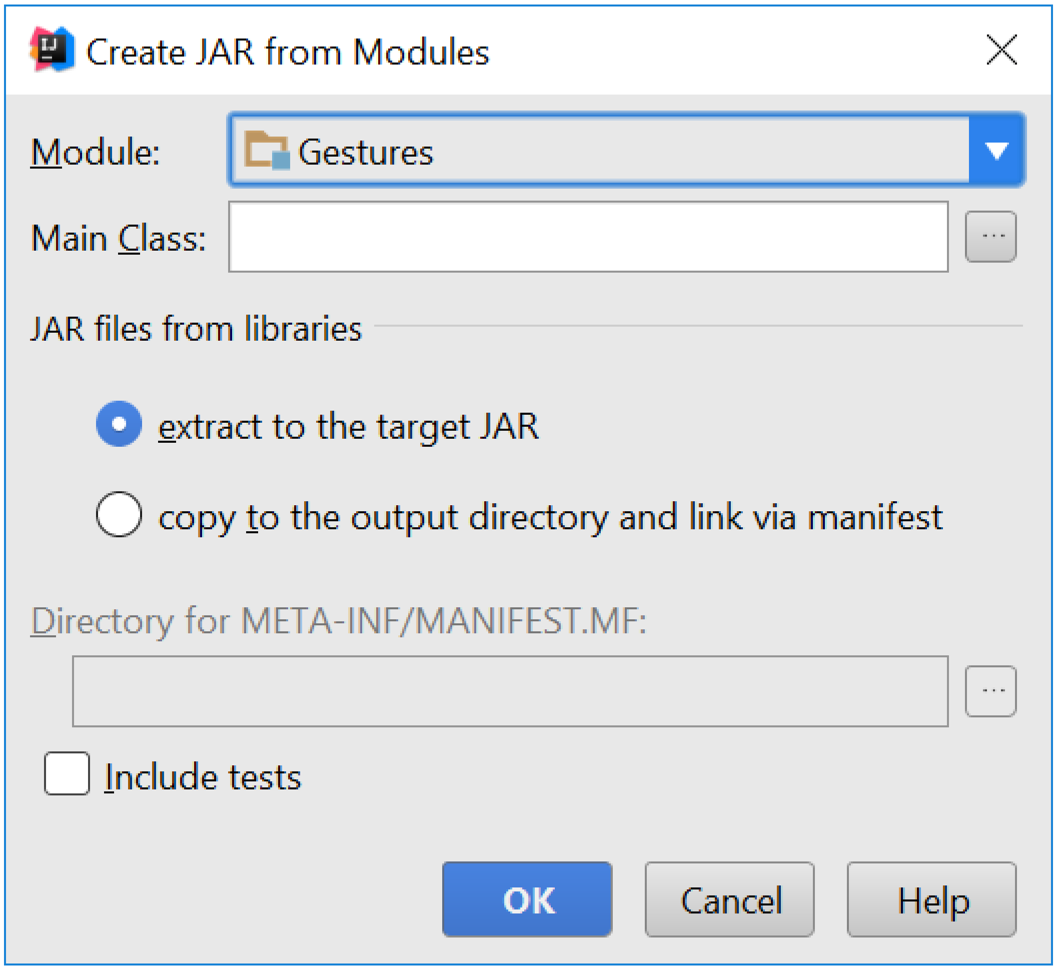
A complete successful attempt for this extra credit should include:

- An original image (PNG or JPG). As before, you are not allowed to use any images provided to you as part of these assignments. All images must be cited in your README file.
- 3 mosaicked versions of that image, using different number of seeds.
- Image mosaicking is exposed suitably in the graphical user interface. Supporting it through the text view from the previous assignment is not required.
- The design for this feature is well incorporated into your design of the overall application.
- A section in your README file briefly explaining the different parts of your program that changed to implement this feature.

Create a JAR file of your program

To create a JAR file, do the following:

- Go to File -> Project Structure -> Project Settings -> Artifacts
- Click on the plus sign
- Choose JAR -> From Modules with dependencies. You should now see the module in your project that you are working on (may be different than what is shown in the image below)


Create JAR from Modules

Module: Gestures

Main Class:

JAR files from libraries

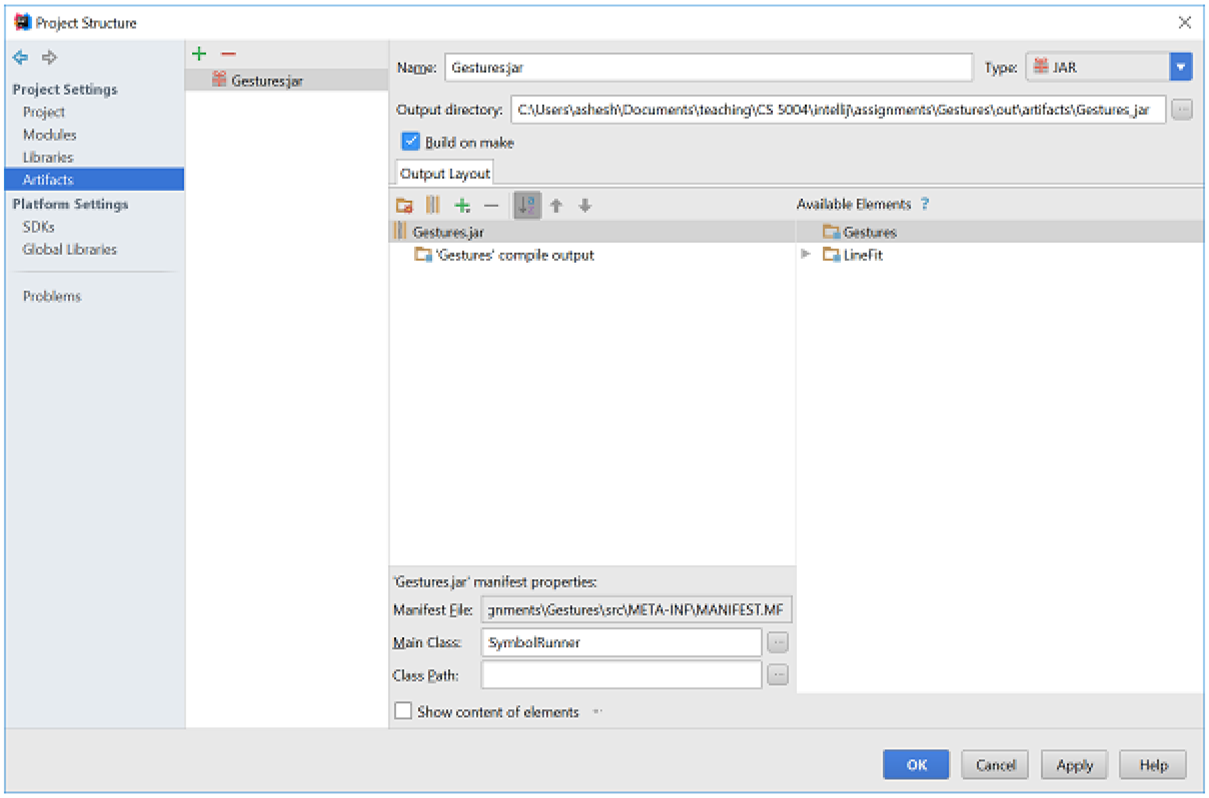
● extract to the target JAR

○ copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

☐ Include tests

- Select the main class of your program (where you defined the `main(String[] args)` method)
- If you see a checkbox labelled "Build on make", check it.
- Hit ok
- You should now see something like



Gestures.jar

Name: Gestures.jar

Type: JAR

Output directory: C:\Users\ashesh\Documents\teaching\CS 5004\intellij\assignments\Gestures\out\artifacts\Gestures_jar

[✓] Build on make

Gestures.jar
'Gestures' compile output

Gestures
LineFit

'Gestures.jar' manifest properties:

Manifest File: gnments\Gestures\src\META-INF\MANIFEST.MF

Main Class: SymbolRunner

Class Path:

[ ] Show content of elements

If now you see a checkbox labelled "Build on make", check it now.

- Make your project (the button to the left of the run configurations dropdown, with the ones and zeros and a down-arrow on it). Your .jar file should now be in /out/artifacts/.
- **Verify that your jar file works** . To do this, copy the jar file to another folder. Now open a command-prompt/terminal and navigate to that folder. Now type java -jar NameOfJARFile.jar and press ENTER. The program should behave accordingly. If instead you get errors review the above procedure to create the JAR file correctly. **You can also run the jar file by double-clicking on it.**

4 Command-line arguments

Your program (from IntelliJ or the JAR file) should accept command-line inputs. Three command-line inputs are valid:

- `java -jar Program.jar -script path-of-script-file`: when invoked in this manner the program should open the script file, execute it and then shut down.
- `java -jar Program.jar -text`: when invoked in this manner the program should open in an interactive text mode, allowing the user to type the script and execute it one line at a time.
- `java -jar Program.jar -interactive`: when invoked in this manner the program should open the graphical user interface.

Any other command-line arguments are invalid: in these cases the program should display an error message suitably and quit.

Criteria for grading

You will be graded on:

1. The completeness, layout and behavior of your graphical user interface.
2. Your design (interfaces, classes, method signatures in them, etc.)
3. Whether your code looks well-structured and clean (i.e. not unnecessarily complicating things, or using unwieldy logic).
4. Correctness of your implementations, evidenced in part by the images you submit.
5. Whether you have written enough comments for your classes and methods, and whether they are in proper Javadoc style.
6. Whether you have used access modifiers correctly: `public` and `private`.
7. Whether your code is formatted correctly (according to the style grader).

What to submit

Your zip file should contain three folders: src, test and res (even if empty).

1. All your code should be in src/.
2. All your tests should be test/.
3. Submit a correct JAR file in the res/ folder. We should be able to run your program from this jar file.
4. **Submit a screen shot of your program with an image loaded.**
5. All required components of any extra credit you attempted.
6. Submit a README file that documents how to use your program, which parts of the program are complete, design changes and justifications, and citations for all images.
7. A USEME file that contains a bullet-point list of how to use your GUI to use each operation supported by your program. Screenshots would be helpful, but not necessary.
8. At least one script file that should work. Include any images that the script file uses.