ECE 364 Project: Image Blending & Morphing

Phase II

Due: April 21, 2019

Image Blending & Morphing

Completing this project phase will satisfy course objectives CO2, CO3 and CO4

Instructions

- Work in your Lab12 directory, and you can copy the data files into that folder.
- You must meet all *base requirements* in the syllabus to receive any credit.
- Remember to **commit & push** the required file(s) to GitHub periodically. **We will only grade what's been pushed to GitHub!**
- Do *not* push to GitHub any file that is *not* required. **You will lose points if your repository contains more files than required!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- Name and spell the file, and the functions, exactly as instructed. Your scripts will be graded by an automated process. **You will lose some points, per our discretion, for any function that does not match the expected name.**
- Make sure your output from all functions match the given examples, within acceptable tolerance. **You will not receive points for a question whose output mismatches the expected result.**
- You can use any module from the **Python Standard Library**, but, aside from the libraries described in the document, **you cannot use any external library** unless authorized by the instructor.
- This is the second of two phases for the course project for ECE 364. The first phase is required for this phase to work.
- This is an **individual** project. All submissions will be checked for plagiarism.
- Make sure you are using Python 3.7 for this project.

Introduction

By now, you should have completed the first phase of the project, where you have implemented the process of blending two images with a specific α-value. In addition to the images, you were given two text files that define the correspondences between the two images. In this phase, you are going to implement a simple GUI Application, using PyQt5 and the Qt Framework, that allows for obtaining corresponding points from the two images. These points will then be used to display the result of blending the two images. The purpose of the GUI Application is to help identify the best correspondences for specific images. For this phase, all of the functionality that you will implement must be accessible from the GUI, and not from the command-line.

The GUI Design

Create a UI file called `MorphingGUI.ui`, using the Qt Designer, that contains the layout shown below in Figure 1. Convert it into `MorphingGUI.py` so that you can import it into your Python code, and create a file called `MorphingApp.py` that would consume the GUI file[1]. The figure shown uses many widgets from the Qt GUI[2].

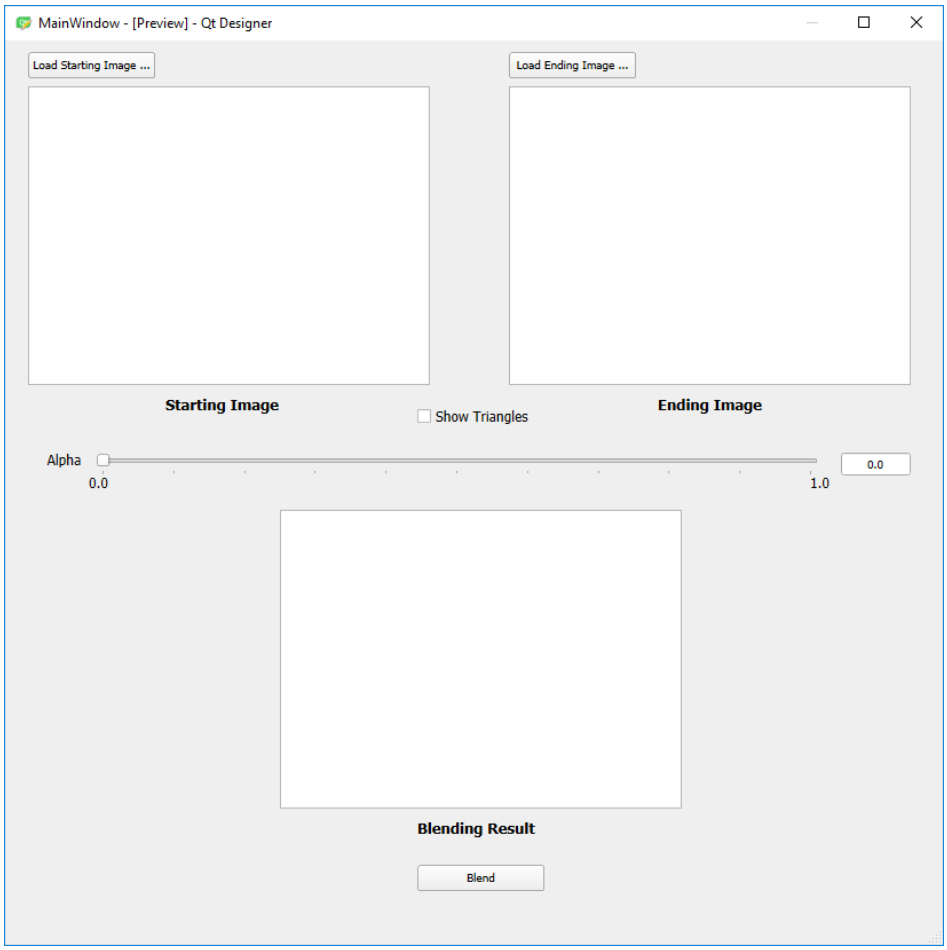

Figure 1: The Morphing GUI.

[1]Remember to only work in the consumer file, as you may override the GUI Python file and lose your work.
[2]Please refer to the PyQt5 documentation to understand the functionality of the available widgets.

Application Behavior

Designing a functioning and a stable GUI application is a non-trivial task whose complexity grows with every widget you add to the application. The simplest way to control the application behavior is to model it after a "Finite State Machine." In this phase, we are going to define a set of application states, and your job will be to make sure that the application is always in one of those states.

Once you understand the application states, it is recommended that you draw a state-transition diagram and verify the behavior of the application with your TA before you start coding.

Note: One common mistake that students do is to create member attributes and/or member functions that bear the names of the described states below. This is *not* required, nor is it useful. These states are only a conceptual way to understand the application behavior.

Initial State

When you first run the application, it should start with some widgets disabled (The slider, the slider value text box, and the "Blend" button,) and the two loading buttons enabled, as shown in Figure 2. This will force the user to load the two images to be able to continue. Clicking on these buttons should pop up a file selection dialog box that allows the user to select an image, and once the image is selected it should be displayed on the widget below the button. You may assume that both images will either be grayscale or color images, with extensions `".png"` or `".jpg"`.

In addition to loading the images, you should also load the associated text files containing the correspondences, if they exist at the same images path. For this project, we will use the convention that for image `"<image name>.jpg"`, the associated text file must be `"<image name>.jpg.txt"`.

Loaded State

Once you verify that both images have been selected by the user, and load the points from the text files, you should enable all widgets that were disabled, as shown in Figure 3. The application now is in the "Loaded State", where the user can do one of the following actions:

- Reload a different starting image, via clicking again on the "Load Starting Image ..." button.
- Reload a different ending image, via clicking again on the "Load Ending Image ..." button.
- Display the Delaunay triangles, via checking the checkbox, which would display the result shown in Figure 4.
- Hide the Delaunay triangles, via unchecking the checkbox, which would bring back the application to Figure 3.
- Select an α-value from the slider, then clicking on the "Blend" button to display[3] the result in the "Blending Result" view, as in Figure 5.

Note that the slider does not have a mechanism to display its value by default, hence the addition of the adjacent textbox, which should be updated for every change in the α-value (You may restrict the α increment to 0.05.) Also, note that the user can now repeat any of the actions described above as many times as desired.

While the above behavior is a nice presentation of the blending process, it does not offer much functionality; since we are only invoking the library internally and not utilizing the GUI for anything else. The true utilization of the GUI application is when used to acquire correspondences, and there are two possibilities: either not having any correspondences, or having some correspondences that we want to preserve and add more pairs to. For either of these, the application will be in the "Point-Selection State".

[3] You may want to maintain the blended image with a descriptive name that includes the α-value.

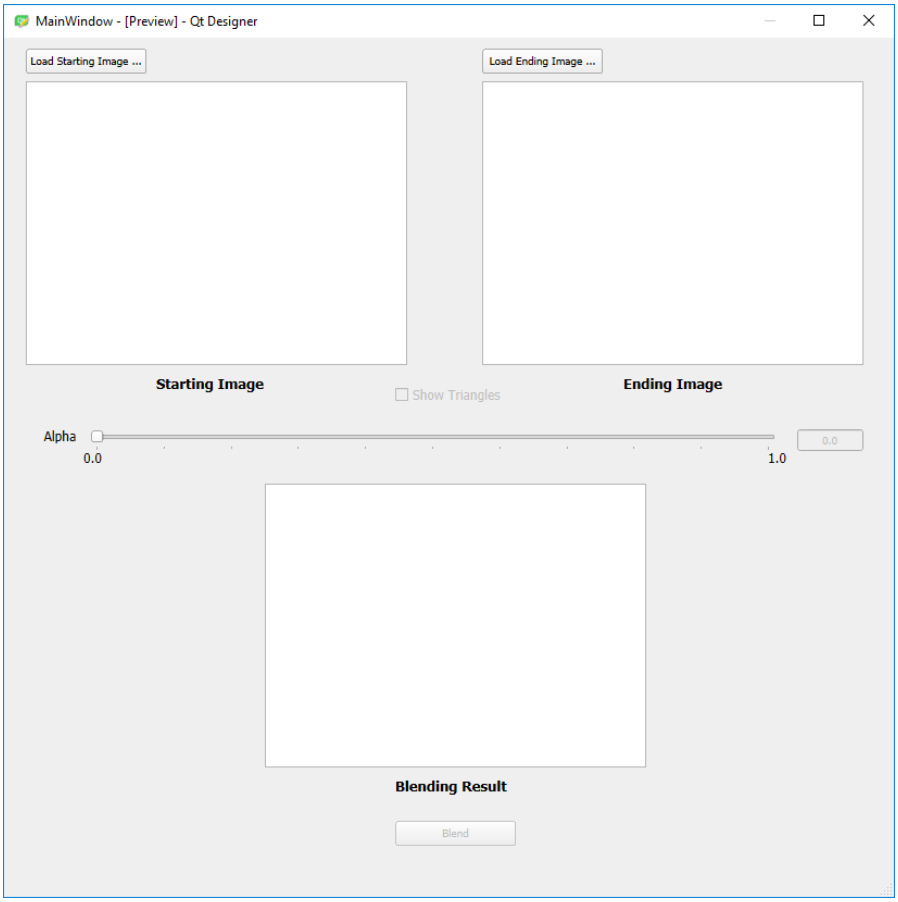

Figure 2: The Initial State of the Application.

Point-Selection State with no Prior Correspondences

If there were no correspondences, the application should create the text files, and update them with the correspondences as the user acquires them. The behavior for selecting the points should be as follows:

- Using the mouse, the user will hover over the starting image and click to select the first point.
- Once the point is selected, a "Green dot" at the selected location should be displayed, to provide a feedback to the user of the selected location.
- The (x, y) coordinates of the selected point should be captured temporarily.
- If the user is not satisfied with the point selection, he/she can press the backspace key on the keyboard once, and the point will be discarded and the dot must be removed.
- If the user is OK with the point, he/she cannot select another point on the same image, even if they click the mouse again on the image many times.
- Given that we chose a point from one image, the user must now select the corresponding point from the second image, and a "Green dot" should then be displayed at that location as well.
- Similarly, the user can accept or reject the point, and they cannot select another point from the same image.
- If the user is satisfied with the point-pair, they can persist it via doing one of the following:

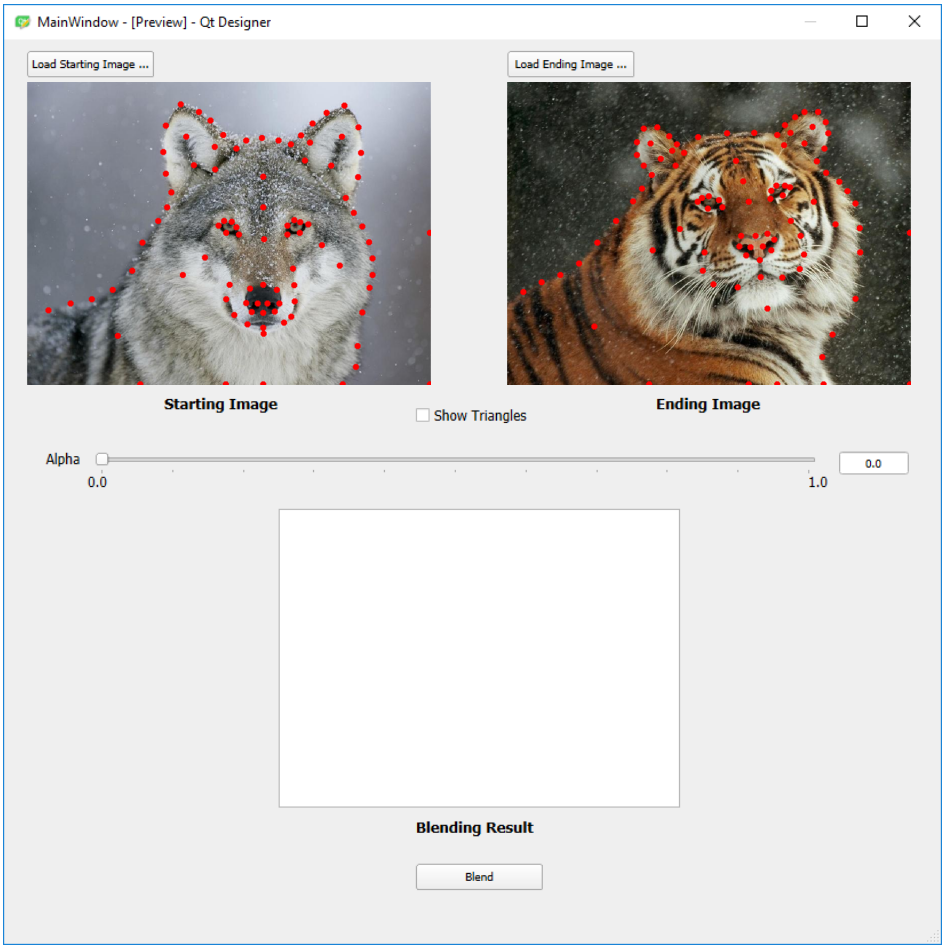

Figure 3: The application window with the images and points shown.

  - Clicking on the application form anywhere other than the two images.
  - Clicking on the starting image again to start selecting a new pair.

- Once a corresponding-pair is persisted, both points must be saved in the files, and the color of the dots must change from "Green" to "Blue", giving the user another feedback that the pair is now persisted.
- The selection behavior is repeated until the user deems the number of paired selected sufficient.

Figure 6 shows the application in the "Point-Selection State" with some points selected and one pair still not persisted. Note that if the user clicks on checkbox, the Delaunay triangulation must be calculated on the fly, and the result displayed using the persisted points only, as in Figure 7. Additionally, if the triangles are already displayed, the user can select and persist another pair, at which point the triangulation should be recalculated and the display re-updated.

Note: The point selection process should always start with the left image.

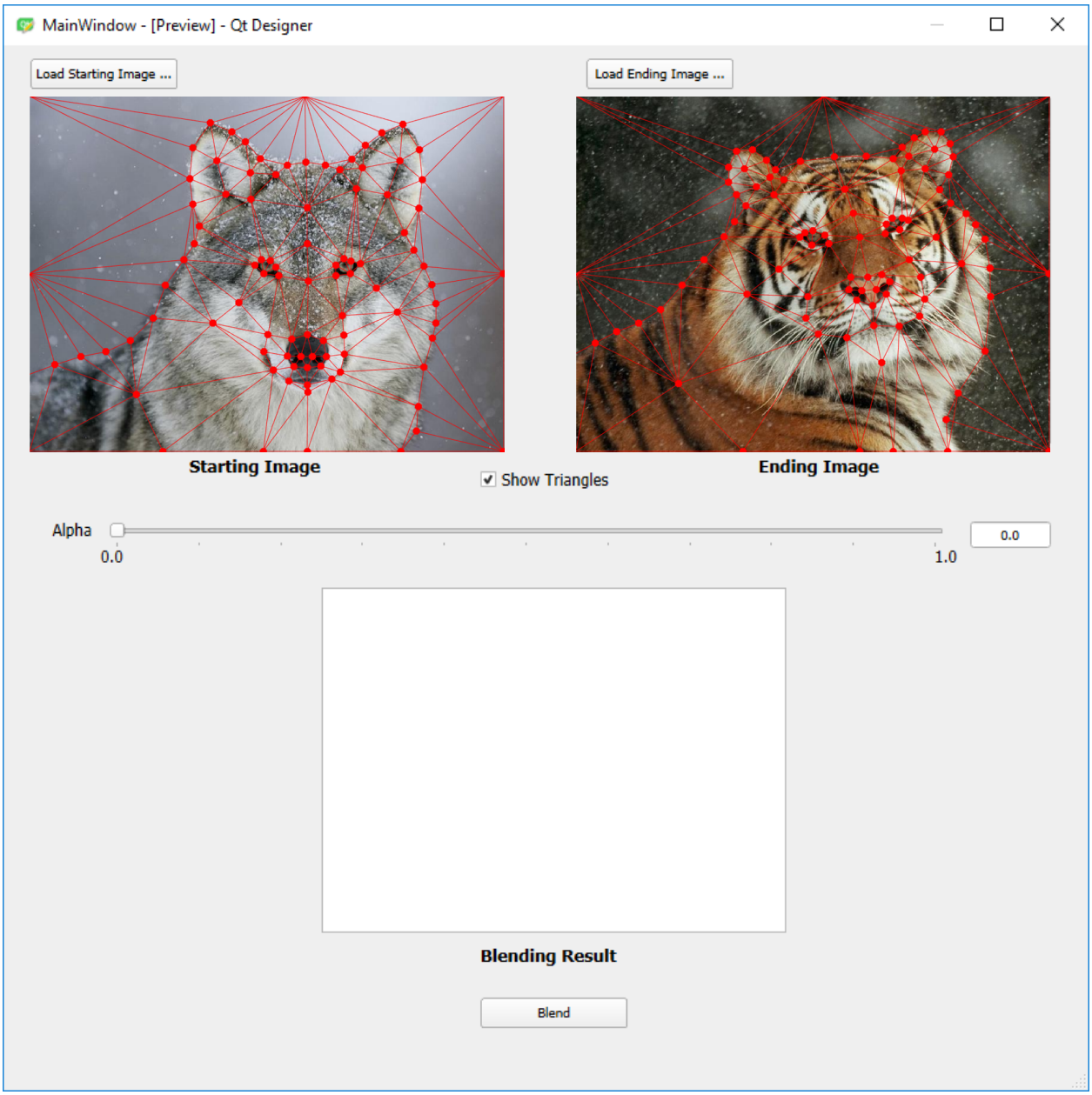

Figure 4: The application window with the Delaunay triangles shown.

Point-Selection State with Prior Correspondences Present

If the images already had correspondences saved in the relevant text files, then they should be displayed using the "Red dots", as in the "Loaded State", and we can then start adding points using the behavior and colors described in the previous section. The different colors of the dots would inform the user of which ones were obtained from the files, which ones were selected and persisted, and what pair has been selected last. Figure 8 shows an example with all three types of dots displayed along with the triangles. Note that the triangulation should now include both, the original and the newly persisted pairs.

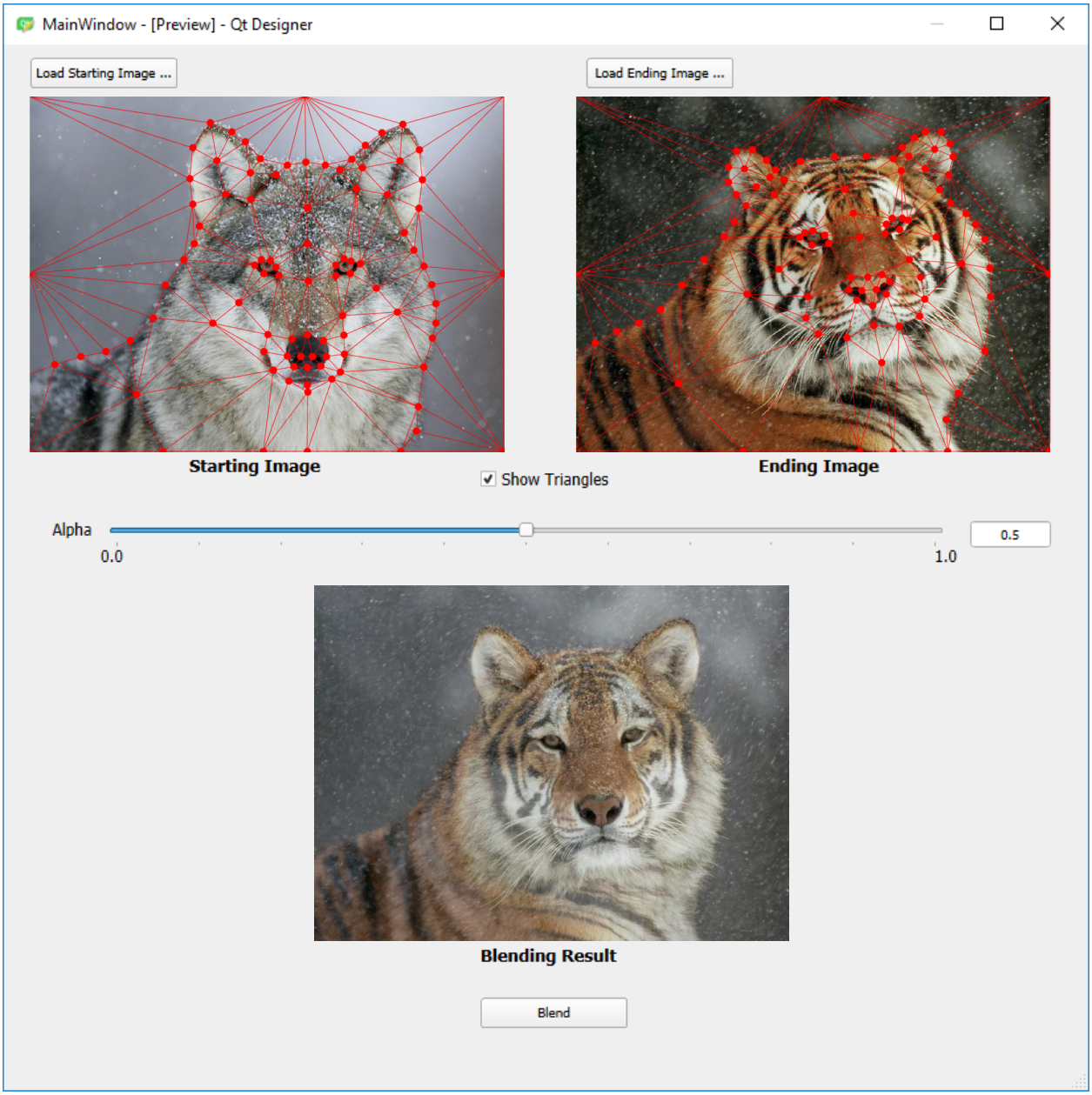

Figure 5: Displaying the result of the blending operation.

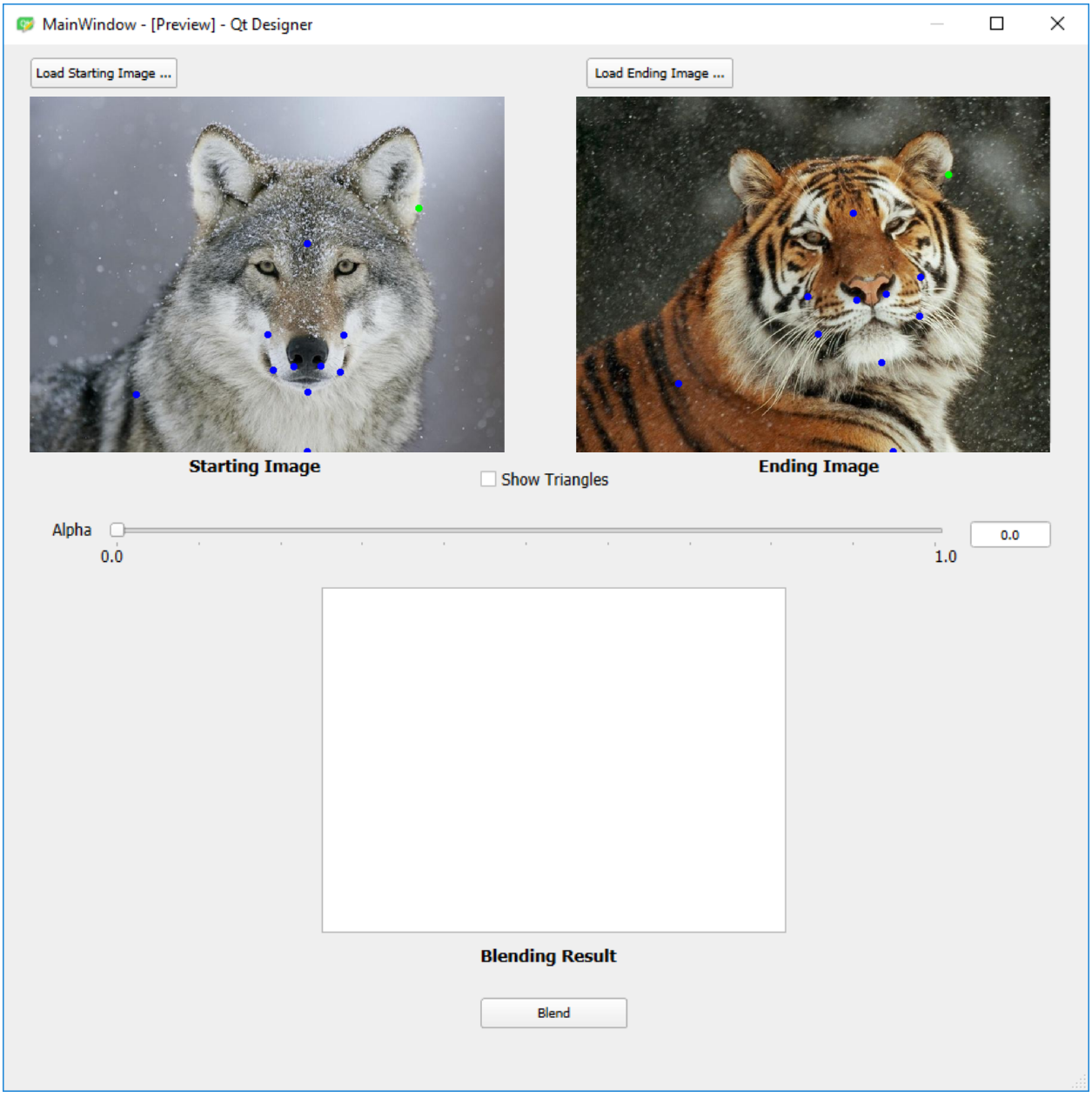

Figure 6: The images with some points selected, and one pair still not persisted.

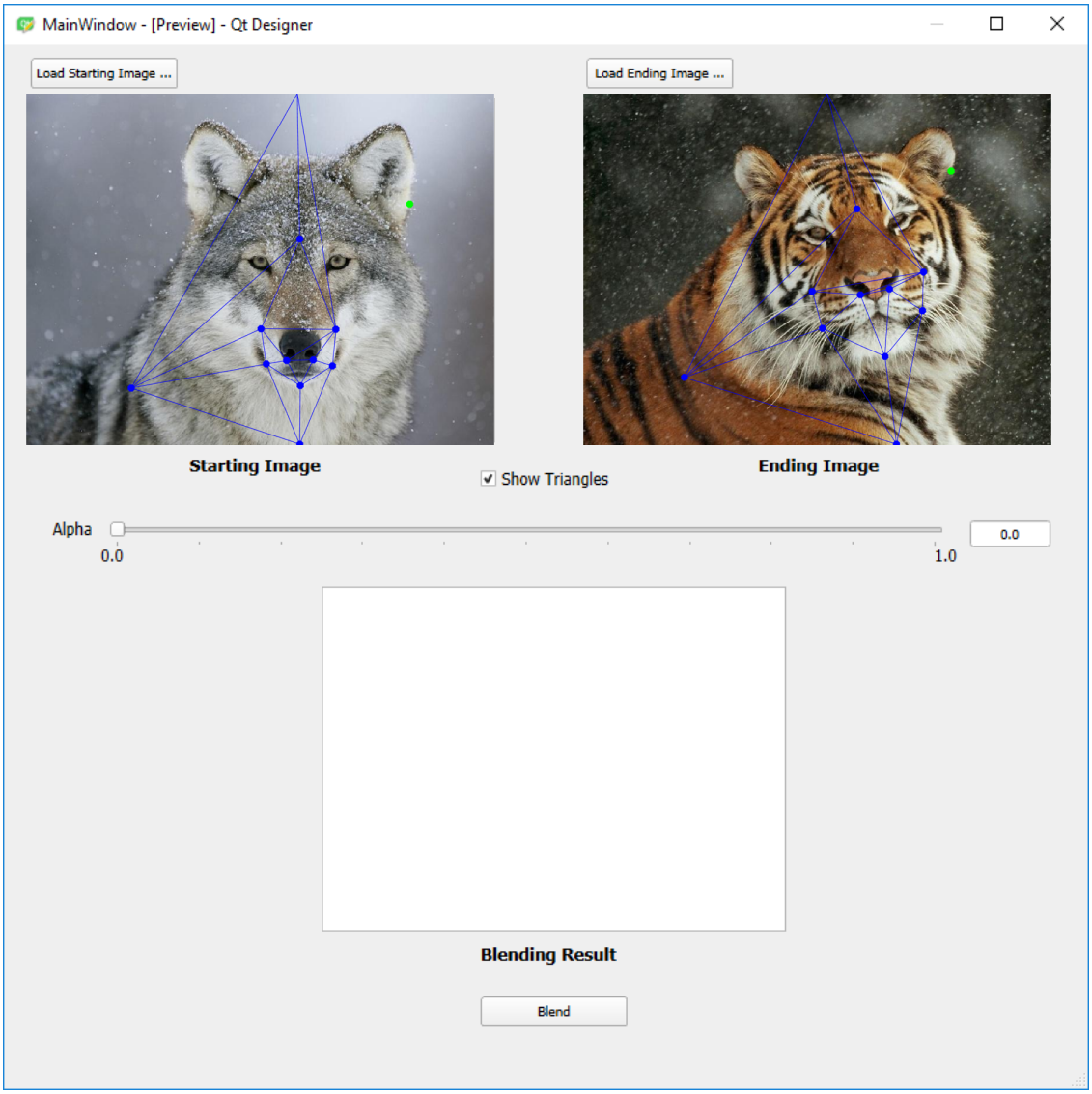

Figure 7: The images with some points selected, and Delaunay triangles displayed.

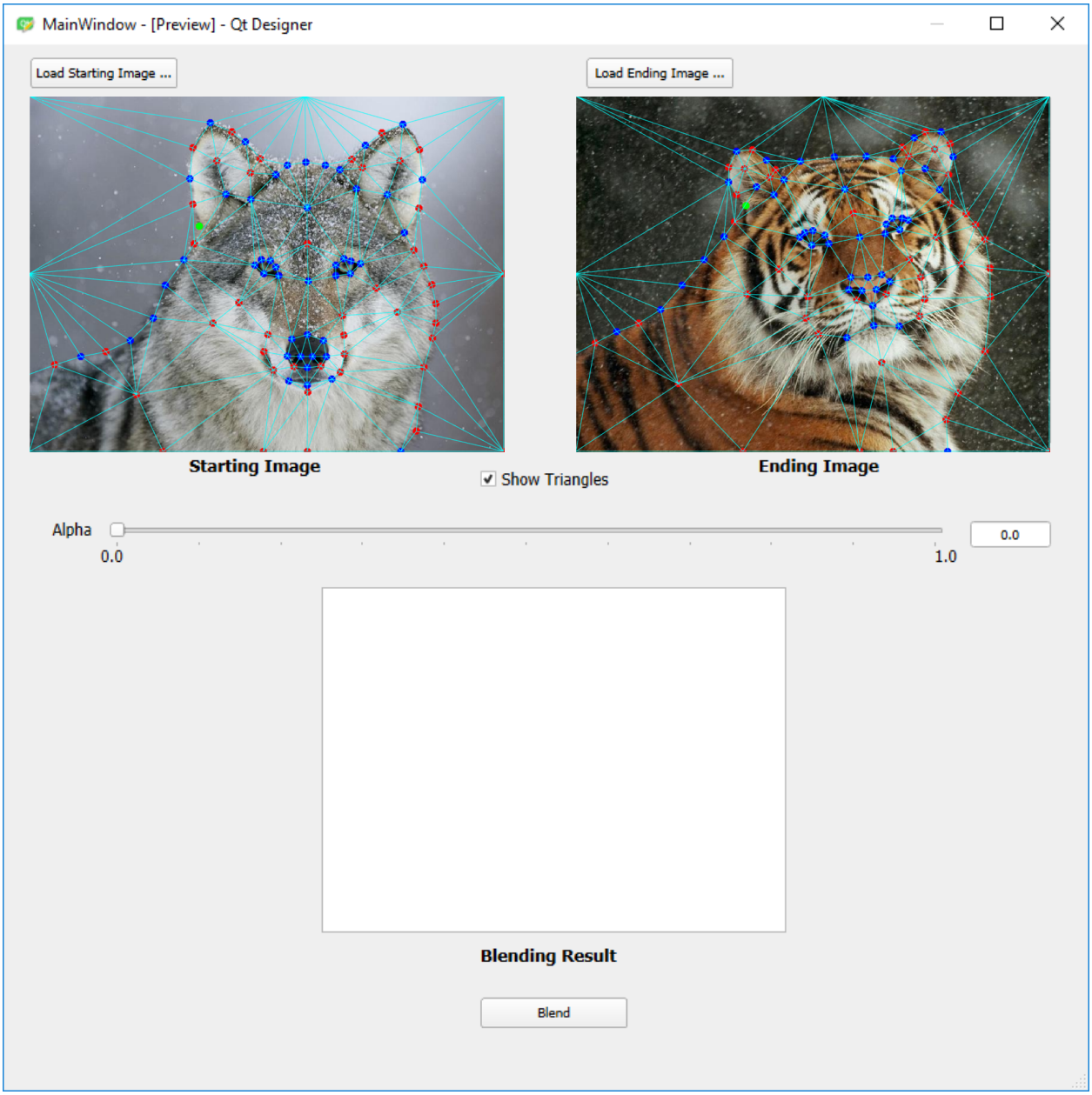

Figure 8: The images with some points using different colors.

Comments

- The project description shows three types of colors to use for the dots: red, blue and green; for priors from files, newly acquired, and last point/pair. You may choose other colors, as long as you have three of them.
- The color of the triangulation lines may be similar to the new acquired points, or you can select a fourth color.
- When you display the image in the widget, please note the following:
  - It is acceptable for the image to be smaller and not fill the whole widget.
  - It is *not* acceptable for the image to be larger than the widget, which will cause scrollbars to be visible in the widget.
  - It is *not* acceptable for the image to be deformed. In other words, the aspect ratio of the image must be maintained.
- Your grade for this phase is based on your demo to the TA. **Failing to demonstrate your work to your TA will result in a zero credit for this phase**. Moreover, please note that the final judgment for passing the use-cases rest with the TAs. If you have any doubt about the application behavior, it pays to ask first before you find out at demo time.
- Your submission must consist of the files:
  1. `MorphingGUI.ui`.
  2. `MorphingGUI.py`.
  3. `MorphingApp.py`.

Extra Credit

Point Selection History

It has become a common expectation of GUI applications, that accept multiple user actions, to provide a method for undoing and redoing several actions. For this task, modify the behavior of your application to react to undoing the point selection via pressing (CTRL + Z) and redoing the point selection via pressing (CTRL + Y). Pressing the "Undo" combination should remove the selected points (not pairs) one at a time, covering selected points until you reach the beginning of the new points selection. Similarly, pressing the "Redo" combination should add the selected points (not pairs) one at a time, until the last point of selection where you stopped.

Note that the conditions for selecting points after "undoing" any number of them must still conform to preventing the selection of two consecutive points from the same image. Also, note that the color of the last point displayed after undoing any number of points should always be indicated using a different color from the rest. Finally, note that the user can reset the "Redo" points by selecting a new point before we reach the last selection. For example, if the user selects 31 points, undoes until point 10, and selects a new point via clicking, then points 11-31 would be discarded.

Form Resizing Behavior

All real-world Rich GUI applications are expected to react smoothly to window resizing. For this task, you must add some logic to handle window resizing, such that the aspect ratio of all widgets (and any displayed images) is maintained. You may choose a minimum and a maximum window size.

Morphing Sequence Auto-Generation

If your code performs the blending for a single α-value relatively fast, then, once point selection is complete and the user clicks on the "Blend" button, you should generate images for **all** possible α-values (a total of 21 images corresponding to $[0.0 \leq \alpha \leq 1.0]$ at an increment of 0.05,) and display the one which matches the slider selection. After that point, the user can simply change the α-value from the slider, and the respective blended image can be displayed instantaneously following any change in the α-value, which makes changing the slider, from one end to the other, display the whole sequence smoothly.

Note: Do not save any of the created images to disk. Also, for this task to be accepted, the whole sequence must be generated in less than two minutes.

