ImTransistor

Bhavana Jonnalagadda & Anusha Sinha

CS 11 Project - Spring 2014

1. **Introduction**

   The purpose of this program is to create a smooth transition between two different images that looks natural. We define the "natural" look to mean that pixels are usually moved together in the groups they originated in, where possible. The program converts the two input images to black and white, and then it generates a number of frames (specified by the user) between the images and encodes them into a video with the QuickTime File Format (`.mov`). The video currently outputs at 30fps. So far this encoding only works on machines running the Mac OS. We used the OpenCV and Boost C++ libraries to complete this project. Throughout the program, we make use of the OpenCV `Mat` and `Point` objects.

2. **The Main Loop**

   i. **Input Data**

      We start by creating two `Mat` objects to store our initial and final images, `src1` and `src2`. We then call the `get_images` function on these two `Mat` objects, and we also pass the function `argc` and `argv` (the command line arguments).

      The `get_images` function first checks that the proper number of command line arguments were received (i.e., 3 arguments), and then stores the data for each image in `src1` and `src2` using the OpenCV `imread` function. The `get_images` function then checks to make sure the images were successfully loaded (i.e., if either `src1` or `src2` is empty), and then checks if the images are the same size using the `size` members of the `Mat` image objects. Next, the OpenCV `threshold` function is used to make inverted binary images so that operations can be performed on the black pixels, since there are less black pixels than white pixels. This inversion is necessary because a black pixel has a value of 0 without inverting, which would cause the transition to be performed on the white pixels. The `get_images` function is of the type `void`, so it does not return anything.

      Finally, we input the number of frames desired by the user using the stardard `cout` and `cin` procedures. We check that the number of frames (`num`) is valid, before creating a `vector<Mat>` object called `frames` by passing the `move_transition` function `src1`, `src2`, and `num`.

ii. **Creating the Frames**
The frames are stored in a `vector<Mat>` object called `frames` that is created by passing the `move_transition` function `src1`, `src2`, and `num`. The `move_transition` function creates a `vector<Mat>` called `frames` which is of length num, along with three `Point` vectors, called `finalv`, `initv`, and`pixels` (which store indices to the initial and final images, along with an arbitrary intermediary). It also creates a `Mat` image called `same` that stores the pixels that will not change from the initial to final image. The function performs a bitwise `AND` on `src1` and `src2` and stores the result in `same`. It then uses the OpenCV function `findNonZero` to get the non-zero indices for `src1` and `src2` and stores them in `initv` and `finalv` (basically the number of black pixels that do need to move). It then splits down two branches depending on whether the initial image has more black pixels that move, or if the final image has more black pixels that move.

(a) If the initial image has more black pixels that do not stay the same, the program starts by making `pixels` the proper size (according to `finalv`). It then iterates through every element in finalv and finds the point in the initial image that has the least distance to a point in the final image. It then adds the point to the `pixels` vector and deletes it from `initv`. It then calculates the distances each pixel needs to travel by using the OpenCV `subtract` function on `finalv` and `pixels`, storing the result in `finalv`. It then creates a `Point` vector to use in the loop that will create the frames called `move`. Finally, it creates the frames using the values stored in `pixels` and using the OpenCV `scaleAdd` function to move each pixel gradually along the line from its initial and determined final positions. At each iteration of this `for` loop, it reduces the number of pixels that are left to be drawn. At the end of each iteration, it adds in pixels that stay constant (the values stored in `same`).

(b) If the final image has more black pixels that do not stay the same, the program goes through the same general process described above, but it scales everything using the size of `initv` (i.e., the number of original black pixels that need to move).

Finally, the original image is inserted in the beginning of the `frames` vector, the original inversion of the pixel values is reversed in each of the frames and the frames are converted to color. The function then outputs `frames`.

iii. **Creating a Directory to Store the Frames**
The program then creates a directory to store the frames using the Boost `create_directory` function and performs error checking to make sure the output folder was created. It then iterates through the `frames` and writes each frame to the newly created `Output` directory using the OpenCV `imwrite` function. If, for some reason, `frames` was not created correctly, it returns an error message.

iv. **Creating the Video (if Applicaple)**
The final step is to create the output video. The program first asks the user whether an output video is desired (because this step of the process takes the longest). If

desired by the user, the video is encoded using the OpenCV `VideoWriter` and `outputVideo` functions. The program first checks that the video writing process has been started properly (returning an erro message if it has not been started properly) before iterating through `frames` and writing the image in `frames` to the appropriate frame of the video. Finally, the program uses the `cout` function to print the location of the files and then returns 0 by convention.

3. **Running the Program**
   This program requires three command line arguments: the program's name, the original image, and the final image. The image argument must be of the same size and can be mostly any format (except `.gif`). Before running the program, make sure there is no folder named "Output" in the same directory.