CS180 Project 3: Face Morphing

Part 1: Defining Correspondences

In this part, we first need to manually mark the key points on two images to control the morphing. By defining correspondences (e.g., mapping eyes to eyes, mouth to mouth, chin to chin, etc.), we can ensure a smooth transformation between the two images.

Our Approach:

1. Selecting Corresponding Points

- `select_points()` (commented out in the current version) allows users to manually select key points on both images, such as eyes, mouth, or facial features, that will be used for triangulation and alignment. These points ensure that the transformation (morphing) between images is meaningful and smooth.

2. Delaunay Triangulation (`compute_delaunay()`)

- Delaunay triangulation is computed based on the selected points for each image. This algorithm divides the set of points into triangles such that no point is inside the circumcircle of any triangle. This produces a mesh of triangles over each image.
- **Purpose**: Triangulation helps in defining smaller regions of the image that can be warped individually during the morphing process, ensuring that the transformation happens smoothly and consistently across corresponding regions in both images.

3. Visualizing the Delaunay Triangulation

- `plot_delaunay_side_by_side()`: This step visualizes the computed Delaunay triangulation for both images, allowing the user to see how the images are segmented into triangles based on the corresponding points.
- Each triangle in Image A will correspond to a triangle in Image B based on the point correspondences, which is crucial for the morphing operation that follows.

4. Affine Transformation and Morphing

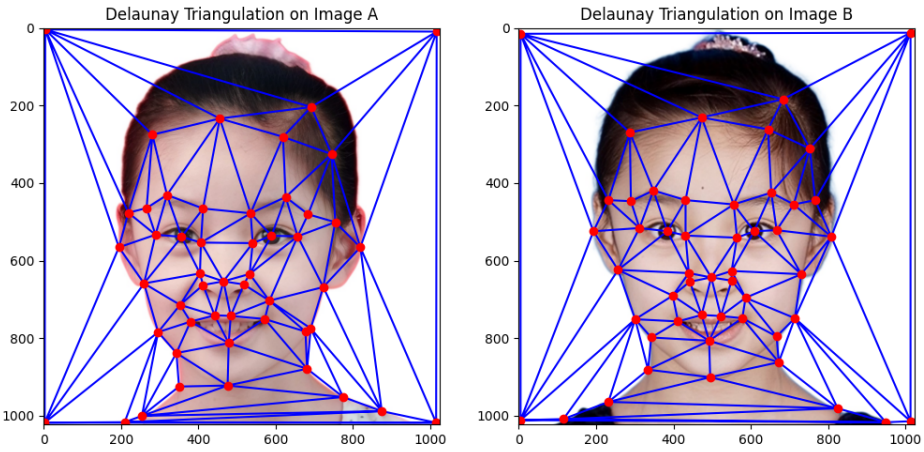
- The affine transformation is the core part of the morphing algorithm. For each triangle:
  - **Affine Transformation**: Each triangle from Image A is warped to its corresponding triangle in Image B using an affine transformation. This transformation linearly maps the points in one triangle to another, preserving the shape and ensuring smooth transitions.
  - **Morphing**: The triangles from both images are blended together using a weighted average (controlled by an `alpha` parameter) to create intermediate frames in the morph sequence. For example, an `alpha` of 0.5 creates an equal blend of both images, while an `alpha` of 0 or 1 retains one of the original images.

5. Saving the Results

- Once the triangulation is visualized, it is saved as a PNG image. Similarly, the triangulation data (which triangles correspond between the two images) is saved as text files for later use in image morphing or further processing.

Result

Image 1	Image 2
	

The Result of Triangulation


Part 2: Computing the "Mid-way Face"

In this part, we computed the "mid-way face" between two images, which represents a blend of both shape and color. To generate the mid-way face, we:

1. Calculated the average position of the key points between the two images to get the average shape.
2. Warped both images to this average shape.
3. Blended the colors of both images to create the mid-way face.

Our Approach:

1. We computed the average shape by taking a weighted average of the key points from both images. The `warp_frac` controls how much the shape transitions between the two images.
2. For each triangle, we applied an affine transformation to warp the shapes of both images to the average shape. The function `apply_affine_transform` computes the affine

transformation matrix for each triangle, and we used OpenCV's `warpAffine` function to perform the actual image warping.
3. The `morph_triangle` function handled the color blending between the two images based on the `dissolve_frac`.

Result

Image 1	Midface	Image 2
		

Part 3: The Morph Sequence

In this part, the goal is to generate an animation consisting of transition frames between the two images. Each frame represents a gradual morph from image A to image B, involving changes in both shape and color.

Our Approach:

1. The `warp_frac` and `dissolve_frac` parameters are gradually increased from 0 to 1, meaning the shape and color transition from image A to image B over time.
2. For each frame, we calculated intermediate key points between the two images, which determine the shape of the image at that point in time. For each triangle, we applied the affine transformation to warp both images to the intermediate shape and blended the colors to create the final morphing frame.
3. The frames were saved and assembled into an MP4 video using OpenCV's `videoWriter`.

Result

Part 4: The "Mean Face" of a Population

In this part, we computed the mean face of a group of images. The mean face represents the average shape and appearance of a population of faces. We can also warp each face in the dataset to the average shape and compare individual faces with the mean face.

Our Approach:

1. We averaged the key points across all individuals in the dataset to compute the mean shape.
2. We warped each face to the mean shape using affine transformations and displayed the results to compare individual faces to the average geometry.
3. We computed the mean face by blending all the faces in the population and displayed the final result.

Result

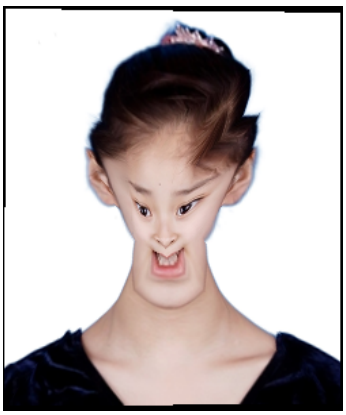
Part 5: Caricatures: Extrapolating from the Mean

In this part, we generated caricatures by exaggerating the differences between our own face and the mean face. By moving the key points of our face further away from the average face's key points, we can create exaggerated facial features.

Our Approach:

1. We computed the difference between our face's key points and the mean face's key points.
2. We exaggerated this difference by amplifying the deviation, thus creating a caricature with exaggerated features (Finally I choose alpha = 1.5 to make the caricature).
3. We applied an affine transformation to warp our face into this exaggerated shape and produced the final caricature image.

Result

Mean Face	Caricatures	My Face
		

Part 6 Bells and Whistles

Using face morphine, I created a picture of my changing process from kindergarten to now.

Our Approach:

1. **Loading Images**: The `load_image` function loads PNG images, converting 4-channel images (RGBA) into RGB format.
2. **Delaunay Triangulation**: The key points from each image are triangulated using Delaunay triangulation, which splits the image into triangles for smooth warping between corresponding areas of different images.
3. **Affine Transformation & Morphing**: Each triangle from the first image is warped and blended with the corresponding triangle from the second image using an `alpha` value. This creates intermediate frames that gradually transition between images.
4. **Generating Morph Sequence**: The intermediate frames are generated by morphing between the images in sequence, and the frames are saved as a video file.
5. **Adding Audio**: The algorithm uses `moviepy` to add audio to the generated morphing video, creating a final MP4 video with sound.

Result

Here is the link of youtube.

https://youtube.com/shorts/WarX4Df_IEM?feature=share

Plus

I write every code for each part seperately. So to submit the project, I copy all my seperate code into the `main.ipynb`.