RISHI RAJ SHARMA
M.Sc CS (2402519)
CSM051

Additional Assignment – Part B

Title: Image Morphing Using Delaunay Triangulation and Barycentric Interpolation

Introduction

Image morphing is a technique used to gradually transform one image into another through a sequence of intermediate images. This process is widely used in animation, computer vision, and special effects. The project focuses on implementing image morphing using **Delaunay Triangulation** for feature correspondence and **Barycentric Interpolation** for smooth blending of pixel values.

Methodology

The image morphing process involves these following key steps:

Feature Point Selection

- User manually select the corresponding feature points on the **source (S)** and **destination (D)** images.
- Feature points should include important areas such as eyes, nose, mouth (for face morphing), or other significant structures.
- Four corner points of the image are also included to ensure full coverage.

Delaunay Triangulation

- Delaunay triangulation is computed for the **source image (S)**.
- The same triangulation structure is transferred to the **destination image (D)** by mapping corresponding points.
- The triangulation is used to maintain geometric consistency during interpolation.

Intermediate Feature Point Computation

- For each intermediate frame **i**, the interpolation parameter **t** is calculated as:
- The feature points for the intermediate image are computed as:
- This ensures smooth transition of feature points between source and destination.

Rendering Intermediate Images

- Each triangle in the intermediate image is rendered using **Barycentric Interpolation**:
  - The color of each pixel inside a triangle is computed as a weighted sum of its barycentric coordinates.
  - The final pixel color is determined by blending corresponding pixel values from the source and destination images.

Generating the Morph Sequence

- A series of **n** intermediate images is generated.
- The entire sequence, including the source and destination images, is saved.
- The images can also be converted into an animated GIF for visualization.

Code:

```
import cv2
import numpy as np
from scipy.spatial import Delaunay
import os
# Global list for click events.
points = []
def get_feature_points(img, window_name="Image"):
   """
   Displays an image window and collects feature points via mouse clicks.
   Press 'q' to finish selecting points.
   Returns the collected feature points.
   """
   points = []

   def click_event(event, x, y, flags, param):
      """ Mouse callback to record feature points on the image. """
      if event == cv2.EVENT_LBUTTONDOWN:
         points.append((x, y))
         # Draw a small circle and label on the image.
         cv2.circle(param, (x, y), 3, (0, 0, 255), -1)
         cv2.putText(param, f"{len(points)}", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1)
         cv2.imshow(window_name, param)

   img_copy = img.copy()  # To preserve the original image
   cv2.imshow(window_name, img_copy)
   cv2.setMouseCallback(window_name, click_event, img_copy)

   print(f"Click on feature points in the {window_name} window. Press 'n' when finished.")
   while True:
      key = cv2.waitKey(1) & 0xFF
      if key == ord("n"):
         break

   cv2.destroyWindow(window_name)
   return points

def add_boundary_points(pts, w, h):
   pts.extend([(0, 0), (w - 1, 0), (w - 1, h - 1), (0, h - 1)])
   return pts

def compute_delaunay(pts):
```

```
    pts_np = np.array(pts)
    tri = Delaunay(pts_np)
    return tri.simplices

def barycentric_coords(pt, tri_pts):
    x, y = pt
    (x1, y1), (x2, y2), (x3, y3) = tri_pts
    denom = ((y2 - y3) * (x1 - x3) + (x3 - x2) * (y1 - y3))
    if denom == 0:
        return (0, 0, 0)
    u = ((y2 - y3)*(x - x3) + (x3 - x2)*(y - y3)) / denom
    v = ((y3 - y1)*(x - x3) + (x1 - x3)*(y - y3)) / denom
    w = 1 - u - v
    return (u, v, w)

def is_inside_triangle(bary):
    u, v, w = bary
    return (u >= -1e-4) and (v >= -1e-4) and (w >= -1e-4)

def morph_triangle(src, dst, out_img, src_tri, dst_tri, inter_tri, t):
    inter_tri_np = np.array(inter_tri, dtype=np.int32)
    r = cv2.boundingRect(inter_tri_np)
    x, y, w, h = r

    for i in range(y, y + h):
        for j in range(x, x + w):
            pt = (j, i)
            bary = barycentric_coords(pt, inter_tri)
            if is_inside_triangle(bary):
                u, v, w_b = bary
                src_pt = np.array(src_tri[0]) * u + np.array(src_tri[1]) * v + np.array(src_tri[2]) * w_b
                dst_pt = np.array(dst_tri[0]) * u + np.array(dst_tri[1]) * v + np.array(dst_tri[2]) * w_b
                src_color = src[int(round(src_pt[1])), int(round(src_pt[0]))]
                dst_color = dst[int(round(dst_pt[1])), int(round(dst_pt[0]))]
                color = (1 - t) * src_color + t * dst_color
                out_img[i, j] = color

def morph_images(src, dst, src_points, dst_points, tri_indices, n):
    src = src.astype(np.float32)
    dst = dst.astype(np.float32)
    morphed_images = []
    h, w = src.shape[:2]

    # Append the source image as the first frame.
    morphed_images.append(src.astype(np.uint8))

    for i in range(1, n + 1):
        t = i / (n + 1)
        inter_points = []
```

```
        for p_src, p_dst in zip(src_points, dst_points):
            x = (1 - t) * p_src[0] + t * p_dst[0]
            y = (1 - t) * p_src[1] + t * p_dst[1]
            inter_points.append((x, y))

        inter_img = np.zeros((h, w, 3), dtype=np.float32)
        for tri in tri_indices:
            idx1, idx2, idx3 = tri
            src_tri = [src_points[idx1], src_points[idx2], src_points[idx3]]
            dst_tri = [dst_points[idx1], dst_points[idx2], dst_points[idx3]]
            inter_tri = [inter_points[idx1], inter_points[idx2], inter_points[idx3]]
            morph_triangle(src, dst, inter_img, src_tri, dst_tri, inter_tri, t)
        inter_img = np.clip(inter_img, 0, 255).astype(np.uint8)
        morphed_images.append(inter_img)

    # Append the destination image as the last frame.
    morphed_images.append(dst.astype(np.uint8))
    return morphed_images

def display_points_table(points_src, points_dst):
    if len(points_src) != len(points_dst):
        print("Mismatch in number of feature points.")
        return
    header = f"{'Index':<6}{'Source (x,y)':<20}{'Destination (x,y)':<20}"
    print(header)
    print("-" * len(header))
    for i, (ps, pd) in enumerate(zip(points_src, points_dst)):
        print(f"{i:<6}{str(ps):<20}{str(pd):<20}")

# Set paths to your images.
src_path = r"Istack.jpg"
dst_path = r"Shutterstock.jpeg"

src_img = cv2.imread(src_path)
dst_img = cv2.imread(dst_path)
if src_img is None or dst_img is None:
    raise FileNotFoundError("One or both input images were not found. Check the paths.")

h, w = src_img.shape[:2]
dst_img = cv2.resize(dst_img, (w, h))

# Collect feature points from the user.
print("Select feature points for the SOURCE image.")
src_points = get_feature_points(src_img.copy(), "Source Image")
print("Select feature points for the DESTINATION image.")
dst_points = get_feature_points(dst_img.copy(), "Destination Image")

# Add boundary points to both.
src_points = add_boundary_points(src_points, w, h)
```

```
dst_points = add_boundary_points(dst_points, w, h)

if len(src_points) != len(dst_points):
    raise ValueError("The number of feature points in source and destination images must be the same.")

# Display the selected points in table format.
print("\nFeature Points Table:")
display_points_table(src_points, dst_points)

# Compute Delaunay triangulation using the source points.
tri_indices = compute_delaunay(src_points)

# Set the number of intermediate frames.
n = 8
morphed_images = morph_images(src_img, dst_img, src_points, dst_points, tri_indices, n)

# Save the morphed sequence as individual images.
output_folder = "morphed_sequence"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

total_frames = len(morphed_images)
for i, img in enumerate(morphed_images):
    output_path = os.path.join(output_folder, f"morph_{i:02d}.jpg")
    cv2.imwrite(output_path, img)
    print(f"Saved {output_path}")

print("Morphing sequence generated successfully.")

# Optionally, display the morphed sequence.
for img in morphed_images:
    cv2.imshow("Morphed Image", img)
    cv2.waitKey(500)
cv2.destroyAllWindows()

# Save the sequence as an animated GIF.
gif_filename = "morphing.gif"
frame_pattern = os.path.join(output_folder, "morph_{:02d}.jpg")
```

Applications

- Facial Morphing: Used in aging effects and face blending.
- Entertainment Industry: Used in special effects.
- Computer Vision: Useful in image alignment and feature extraction.
- Medical Imaging

References

- OpenCV Documentation
- SciPy Library
- Image Morphing Techniques, Research Papers, IEEE Xplore
- Chat-gpt