Combining Cat Images Using Image Morphing

Moses Ananta 13519076
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519076@std.stei.itb.ac.id

Abstract—**Cats are popular animals with over 40 to 70 different kinds of breeds. Although the number of cat breeds are large, sometimes people wanted to have or atleast know a different kind of cat that may not be available in those 40 to 70 listed breeds. This paper explores the use of image morphing using Delaunay triangulation for combining two cat images to produce a unique cat resulting from morphing two of the original cats.**

Keywords—image morphing, delaunay triangulation, cat

I. Introduction

Cats have a long history of domestication, with evidence dating back thousands of years. They are widely considered to be popular household pets due to their independent nature and playful personalities. In fact, there are over 40 to 70 different cat breeds globally [1][2], offering a wide range of options for potential pet owners.

However, it is not uncommon for individuals to desire a cat with specific characteristics that are not present in any existing breed. In these cases, crossbreeding may be employed as a means of creating a unique feline with desired traits. It is important to note that while crossbreeding can offer the possibility of specific characteristics, it may also carry potential risks and drawbacks, such as an increased risk of health issues or breeding problems. It is worth considering that if the goal is just to satisfy the curiosity of the result of combining two different cats, it's possible to use image processing techniques such as image morphing to achieve the result.

Image morphing is a computer graphics technique that allows for the smooth transition of two images. Using this technique,it's possible to create a single image that blends elements from both of the original images. This can result in a totally unique and visually appealing image that would be impossible to generate using traditional approaches.

II. Theoretical Basis

A. Delaunay Triangulation

Delaunay triangulation is a method of dividing a two-dimensional space into a series of interconnected triangles. In the context of image morphing, Delaunay triangulation can be used to divide an image into a series of interconnected triangles which could smooth the morphing transition between two images.

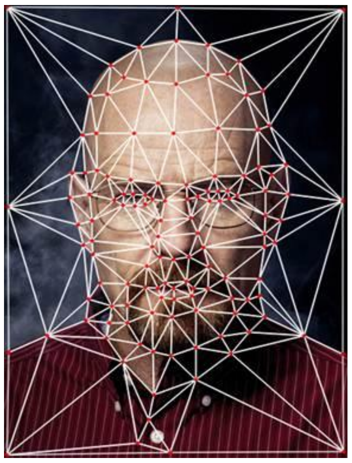

Fig. 1. Example of a Delaunay Triangulation on a person's face[3].

Performing Delaunay triangulation on an image[4] is done by first selecting a set of key points in the image. These key points should be selected in a way that reflects the overall structure and features of the image.

Once the key points have been identified, Delaunay triangulation can be used to divide the image into a series of interconnected triangles. One such approach to compute the Delaunay triangulation of an image is by iterative and incremental process. It works by starting with an empty triangulation and then adding each point to the triangulation one at a time. As each point is added, the algorithm checks for the existence of any triangles that contain the point within their circumcircle. If such triangles are found, they are removed and replaced with new triangles that connect the point to the vertices of the original triangles. This process is repeated until all of the points have been added to the triangulation.

After all points have been triangulated, each resulting triangle is connected to its neighboring triangles along its edges. This creates a mesh of interconnected triangles that covers the entire image. The resulting mesh can then be transformed and deformed to match the corresponding triangles in the other image, creating a smooth transition between the two.

B. Image Morphing

Image morphing is a technique in computer graphics that allows for the smooth transition between two images. The general concept behind image morphing is the idea of creating a series of intermediate images that smoothly transition between the two original images. These intermediate images are created by blending elements from the two original images, with the amount of blending gradually increasing over the course of the transition.

The first step to perform image morphing is to align the two images so that they are properly positioned in relation to each other. This can be done using image warping techniques, which involve transforming the shape of one image to match the other.

Once the images are aligned, the corresponding points in the two images need to be identified. These are the points that will be used to connect the triangles in the two images and create the smooth transition between them. The triangles in one image can then be transformed and deformed to match the corresponding triangles in the other image, creating a smooth transition between the two images.

To create a smooth and seamless transition between the two images, the values of the pixels in the final combined image can be interpolated between the values of the pixels in the two original images. This can be done using image interpolation techniques such as nearest neighbor interpolation, bilinear interpolation, or bicubic interpolation.

III. Experiments

A. General Experiment Process Outline

The general experiment process outline for image morphing two cat images using Delaunay triangulation can be as follows:

1) Select two cat images

The first step is to select the two cat images that will be used for the image morphing. These images should be of high quality and should clearly show the features and characteristics of the cats. The images should also have the same width and height sizes.

2) Identify key points

Once the images are aligned, the key points in the images need to be identified. These key points will be used as the vertices of the triangles in the Delaunay triangulation. Key points should be selected in a way that reflects the overall structure and features of the images, such as the tips of the ears, the corners of the eyes, and the end of the nose.

3) Compute the Delaunay triangulation

Using the incremental algorithm that has been mentioned before, the Delaunay triangulation of the key points can be computed to create a series of interconnected triangles.

The resulting triangles can then be connected to form a mesh, with each triangle connected to its neighboring triangles along its edges. This creates a series of interconnected triangles that cover the entire image

4) Transform the triangles:

The triangles in one image can then be transformed and deformed to match the corresponding triangles in the other image, creating a smooth transition between the two images.

5) Interpolate the values of the pixels in the final combined image

To complete the experiment process, the values of the pixels in the final combined image can be interpolated between the values of the pixels in the two original images. This can be done using image interpolation techniques such as nearest neighbor interpolation, bilinear interpolation, or bicubic interpolation.

6) Evaluate the resulting combined image and make any necessary adjustments

Evaluate the resulting combined image and make any necessary adjustments: Finally, the resulting combined image can be evaluated and any necessary adjustments can be made to optimize the quality of the image and ensure a smooth and seamless transition between the two original images

B. Limitation

The limitation of the current experiment is that the process of acquiring the key points in the cat images is done using a machine learning model from the module pycatfd[5] that has been pre-trained to identify the key points in the cat images. The reason why the machine learning model is involved in these experiments is that it enables for automatically identifying the important key points in cat image without manually doing so with human intervention. This means that that there could be an error in which the machine learning model failed or incorrectly identify the pixel coordinates which are not the key points of the cat and because the limitation of the model the model could only identify a few parts of the cat which is the nose, the chin, the left and right eye, the left and right parts of the each of the cat ears.

C. Experiments and Codes

For better understanding, below are the codes that are used on the experiments.

1) Selecting two cat images

```
img_source_path = 'charteux.jpg'
img_target_path = 'cat_41.jpg'

im_src = plt.imread(img_source_path)
im_tgt = plt.imread(img_target_path)
```

The codes in this process make use of the python library Matplotlib[], that is commonly used for graphics. The code reads the two cat images that are 'charteux.jpg' and 'cat_41.jpg' using the function `plt.imread`.

2) Identifying key points in cat images

```
!rm source_anno.json
!rm target_anno.json
!rm clone https://github.com/marando/pycatfd.git
%cd pycatfd
os.system(f'python catfd.py -i ../{img_source_path} -j >>
../source_anno.json')
os.system(f'python catfd.py -i ../{img_target_path} -j >>
../target_anno.json')
%cd ..
```

The codes in this process make use of the python module pycatfd from github that has been mentioned on the limitation to identify the key points on the cat images.

3) Computing the Delaunay triangulation

```
def Delaunay(vertices, img_size):
  # Convert the vertices to a numpy float32 array
and clip them to the image bounds
  vertices = np.float32(vertices)
  vertices = np.clip(vertices, [0, 0],
[img_size[1]-1, img_size[0]-1])

  # Map the vertices to their indices
  vertex_to_index = {tuple(v): i for i, v in
enumerate(vertices)}

  # Compute the Delaunay triangulation and get the
list of triangles
  image_rect = (0, 0, img_size[1], img_size[0])
  triangulation = cv2.Subdiv2D(image_rect)
  for v in vertices:
      triangulation.insert(v)
  triangles = triangulation.getTriangleList()

  # Filter out triangles that are outside the
image bounds
  triangles = [[int(x) for x in t] for t in
triangles if is_point_in_rect(t, image_rect)]

  # Convert the triangle coordinates to indices
using the vertex-to-index mapping
  triangles = [[vertex_to_index[tuple(t[i:i+2])]
for i in range(0,len(t)-1,2)] for t in triangles]

  # Return the list of simplices (triangles)
  return triangles

def is_point_in_rect(point, rect):
    if point[0] < rect[0] or point[1] < rect[1] or
point[2] > rect[2] or point[3] > rect[3]:
        return False
    return True
```

The code in this process defines the function Delaunay for computing Delaunay Triangulation on sets of key points in an image. The Delaunay function takes in two arguments: vertices and img_size.

The vertices argument is a list of key points that have been identified before, and the img_size argument is a tuple representing the size of an image which.

The function first converts the vertices list to a numpy float32 array using np.float32 function, and then clips the values to the bounds of the image using np.clip so that there would be no out-of-bound points. Next, the function creates a mapping of each vertex to its index in the vertices list, using a dictionary. The reason the mapping is needed is so the triangulation result that would be returned is not in point coordinates format but point index.

The function then creates a Subdiv2D object representing the image bounds and inserts each vertex into the object. It then retrieves the list of triangles from the Subdiv2D object. The function then filters out any triangles that are outside the bounds of the image, and converts the coordinates of the remaining triangles to indices using the vertex-to-index mapping. Finally, the function returns the list of triangles.

The is_point_in_rect function takes in a point and a rectangle, represented as lists of coordinates, and returns True if the point is inside the rectangle and False otherwise. This function is used to filter out triangles that are outside the bounds of the image. It is defined as a helper function.

4) Transform and interpolate the images

```
def morph_images(im1, im2, points1, points2,
alpha):
  # Convert the images to floating point arrays
  im1 = np.float32(im1)
  im2 = np.float32(im2)

  points1 =  np.float32(points1)
  points2 =  np.float32(points2)
  # Compute the Delaunay triangulation of the
points
  tri = Delaunay(points2,im1.shape)

  # Initialize the intermediate image with the
same size and type as the input images
  im_intermediate = np.empty_like(im1)
  im_intermediate.fill(0)

  # Iterate over the simplices (triangles) of the
Delaunay triangulation
  for s in tri:
    # Compute the affine transformation matrix
that maps the triangle in the first image to the
corresponding triangle in the second image
    affine_matrix = cv2.getAffineTransform(
points1[s],  points2[s])

    # Warp the triangle from the first image to
the intermediate image using the affine
```

```
transformation matrix
    cv2.fillConvexPoly(im_intermediate,
np.array(points1[s], 'int32'), (1, 1, 1),
cv2.LINE_AA, 0)
    cv2.warpAffine(im1 ,affine_matrix,
im_intermediate.shape[:2],im_intermediate,
cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT_101)

    # Warp the triangle from the second image to
the intermediate image using the inverse of the
affine transformation matrix
    cv2.fillConvexPoly(im_intermediate,
np.array(points2[s], 'int32'), (1, 1, 1),
cv2.LINE_AA, 0)
    cv2.warpAffine(im2, affine_matrix,
im_intermediate.shape[:2],
im_intermediate,cv2.WARP_INVERSE_MAP+cv2.INTER_LIN
EAR,borderMode=cv2.BORDER_REFLECT_101)


  # Blend the intermediate image with the first
and second images using the specified alpha value
  im_output = add_weighted(im1, 1-alpha,
im_intermediate, alpha, 0)


  return im_output
```

The code in this process defines the function morph_images for transforming the triangle from the Delaunay triangulation process and interpolating the images based on the given alpha value . that takes in five arguments: two images (im1 and im2), two lists of points (points1 and points2), and a float value alpha

The function begins by converting the images and points to floating point arrays using the np.float32 function. It then computes the Delaunay triangulation of the points in points2 using the Delaunay function, passing in points2 and the shape of im1 as arguments.

Next, the function initializes an empty image called im_intermediate with the same size and type as im1 which functions as a placeholder for keeping the transform result. It then iterates over the triangles in the Delaunay triangulation, and for each triangle it computes an affine transformation matrix that maps the triangle in points1 to the corresponding triangle in points2, warps the triangle from im1 to im_intermediate using the affine transformation matrix, and warps the triangle from im2 to im_intermediate using the inverse of the affine transformation matrix.

Finally, the function blends the intermediate image im_intermediate with the first and second images using the specified alpha value, and returns the resulting image.

IV. Results

Following the aforementioned processes, this is the result.

1) *Selecting two cat images*

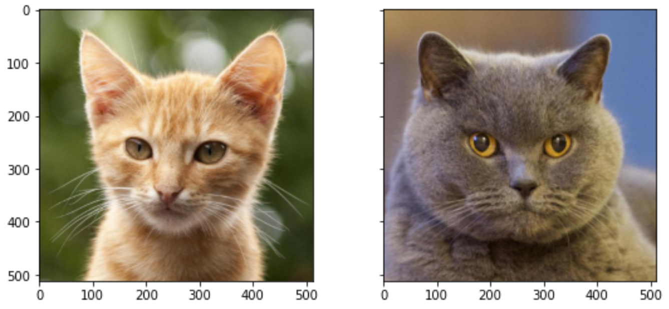

Fig. 2. Two selected cat images

2) *Identifying the key points*

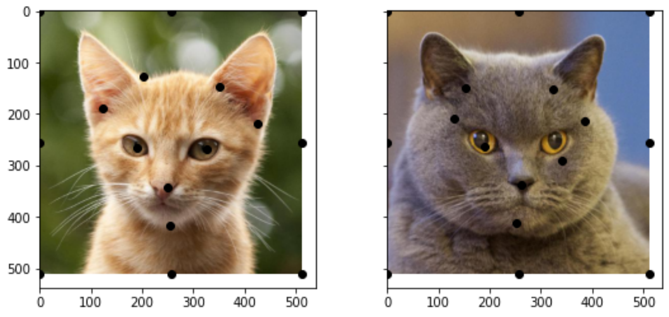

Fig. 3. Key points as black dots in cat images

3) *Triangulate the key points*

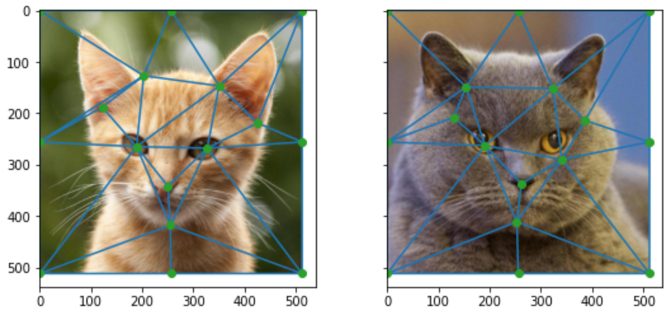

Fig. 4. Triangulation Result

4) *Morph the image*

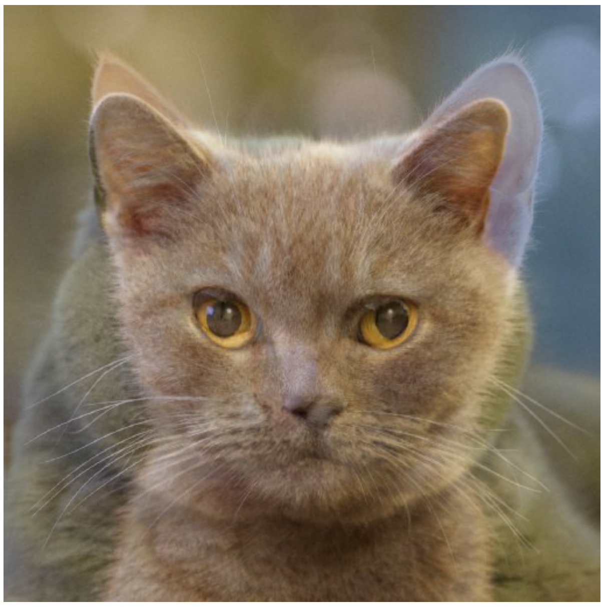

Fig. 5. Image morphing result with alpha value 0.62

Based on the result that has been shown, the resulting image could blend two cat images well although there is some of the image artifact that could be seen as well.

Although the above experiment blended the cat photos well. Sometimes not all images can be blended well such as the example below.

1) Selected cat images

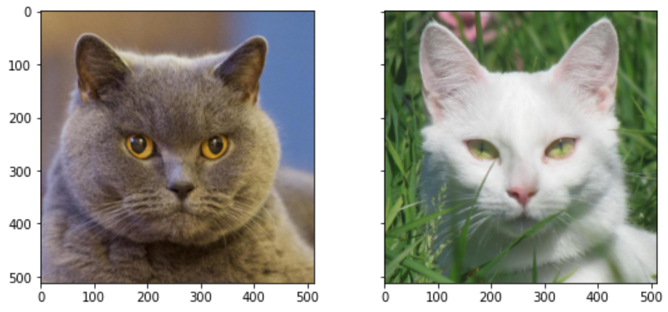

Fig. 6. Selected cat for the second experiment

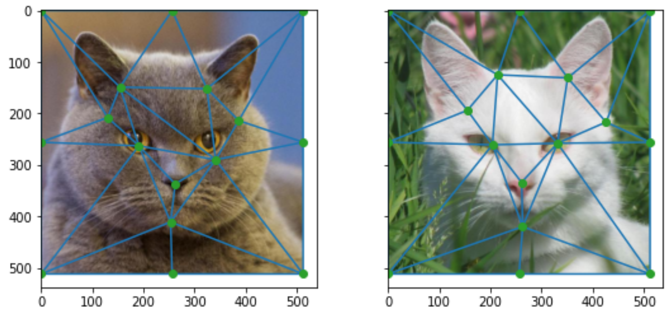

Fig. 7. Triangulation result

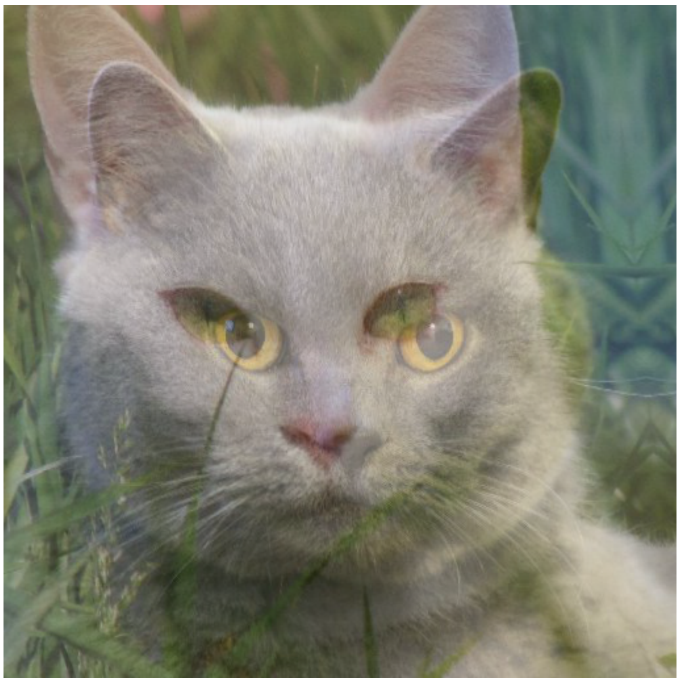

Fig. 8. Image morphing result with alpha value 0.49

From the result that has been shown on Fig. 8., the morphing results poorly and the images are miss aligned. When comparing this image which use delaunay triangulation morphing and the image without the triangulation below,

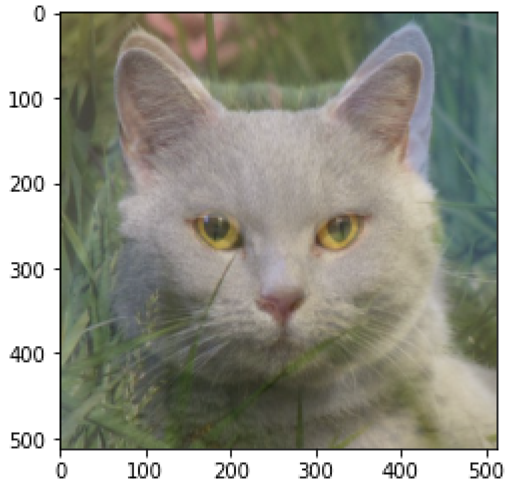

Fig. 9. Image morphing without using delaunay triangulation

It can be seen that the resulting image is more natural. One of the factors why the resulting image from image morphing with delaunay triangulation is miss aligned is because of the incorrect warping of the delaunay triangulation which is caused by the poorly placed key points on the image. As a result, the warping algorithm tried to warp the image to the poorly placed key points which resulted in a poorly aligned morphed image.

V. Conclusion

Based on the experiments, image morphing using Delaunay triangulation method can combine images from 2 different cats. However the resulting image sometimes is miss-aligned due to the key points identifier model assigning the key points in the wrong coordinates.

To further improve the morphing result, key points may have to be manually assigned to guarantee that the key points coordinated are rightly assigned or use a better model. Also increasing the number of key points may further smoothen the interpolation process. And finally, to reduce the artifact from combining 2 images, some algorithm like gradient-based blending may be integrated.

References

[1]]"Breeds – The Cat Fanciers' Association, Inc," cfa.org. https://cfa.org/breeds/

[2] S. User, "Browse All Breeds," tica.org. https://tica.org/breeds/browse-all-breeds

[3] "Face Morphing," devendrapratapyadav.github.io. https://devendrapratapyadav.github.io/FaceMorphing/.

[4] "Morphing in Two Dimensions: Image Morphing Magdil Delport," 2007. [Online]. Available: https://core.ac.uk/download/pdf/37321395.pdf

[5] ashley, "pycatfd," GitHub, Dec. 14, 2022. https://github.com/marando/pycatfd.

[6] Matplotlib, "Matplotlib: Python plotting — Matplotlib 3.1.1 documentation," Matplotlib.org, 2012. https://matplotlib.org/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2022

[illegible]

Moses Ananta 13519076