# SHAPE MORPHING

Mini Project work

Submitted in partial fulfillment of the academic requirement for the award of the degree of

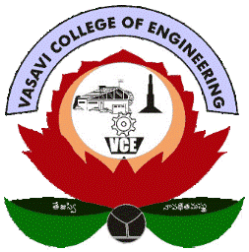
**BACHELOR OF ENGINEERING**

**In**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**By**


| | |
|---|---|
| **K. HARISHRI** | **1602-23-735-013** |
| **SK. SAMEERA** | **1602-23-735-042** |
| **B. SWAPNA** | **1602-23-735-050** |



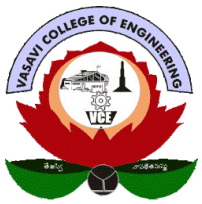
**Department of Electronics and Communication Engineering**

**Vasavi College of Engineering (Autonomous)**

***ACCREDITED BY NAAC WITH 'A++' GRADE***

**IBRAHIMBAGH, HYDERABAD-500031**


**2024-2025**




# Department of Electronics and Communication Engineering

## Vasavi College of Engineering (Autonomous)

*ACCREDITED BY NAAC WITH 'A++' GRADE*

**IBRAHIMBAGH, HYDERABAD-500031**

## CERTIFICATE

This is to certify that the Project work titled **"Shape Morphing"** submitted by

| | |
|---|---|
| **K. HARISHRI** | **1602-23-735-013** |
| **SK. SAMEERA** | **1602-23-735-042** |
| **B. SWAPNA** | **1602-23-735-050** |

students of Electronics and Communication Engineering Department, Vasavi College of Engineering in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Electronics and Communication Engineering is a record of the Bonafide work carried out by them during the academic year 2024-2025. The result embodied in this project report has not been submitted to any other university or institute for the award of any degree.

Head of the Department
**Dr.E. SREENIVASA RAO**
Professor and HOD
E.C.E Department

Internal Guide
**Dr. SRILAKSHMI AOUTHU**
Associate Professor
E.C.E Department

# DECLARATION

This is to state that the work presented in this thesis titled **"Shape Morphing"** is a record of work done by us in the Department of Electronics and Communication Engineering, Vasavi College of Engineering, Hyderabad. No part of the thesis is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source. I hereby declare that the matter embedded in this thesis has not been submitted by me in full or partial thereof for the award of any degree/diploma of any other institution or university previously.

Signature of the students


**K. HARISHRI** **1602-23-735-013**

**SK. SAMEERA** **1602-23-735-042**

**B. SWAPNA** **1602-23-735-050**

# ACKNOWLEDGEMENTS

This satisfaction and euphoria that accompany the successful completion of any task would be incomplete without mentioning the people whose constant guidance and encouragement made it possible. We take pleasure in presenting before you our project, which is the result of studied blend of both research and knowledge.

It is our privilege to express our earnest gratitude and venerable regards to Dr. **E. Sreenivasa Rao**, Head of the Department, E.C.E., for his abounding and able guidance during the preparation and execution of the project work, which helped throughout the project.

We also thank our Project coordinators, **Dr. SriLakshmi Aouthu,** Associate Professor, Dept of ECE and **Mrs.K.R.Deepthi,** Assistant Professor for their continuous guidance and valuable suggestions throughout our project.

Our sincere thanks to the **Principal** and **Management, Vasavi College of Engineering, Hyderabad,** for providing all the facilities in carrying out the project successfully.

Also, we acknowledge with thanks for the support extended by all the staff members and technical staff in shaping up our project. We are thankful to one and all who co-operated with us during the course of our project work.

**K. HARISHRI** **1602-23-735-013**

**SK. SAMEERA** **1602-23-735-042**

**B. SWAPNA** **1602-23-735-050**

# ABSTRACT

**Title of the Project:**

**SHAPE MORPHING**


***ABSTRACT:***

- **Shape Morphing** is a computational technique used to smoothly transform one shape into another by interpolating their geometric structures. It ensures a natural transition between shapes through mathematical transformations such as linear interpolation, Delaunay triangulation, and warping techniques. This algorithm finds applications in **computer graphics, animation, image processing, and AI-based shape recognition**. By preserving structural integrity and maintaining smoothness, shape blending enhances visual realism and is widely used in **morphing effects, medical imaging, and computer vision tasks**.



***Keywords:***

- Shape Morphing
- Shape Blending
- Interpolation
- Geometric Transformation
- Delaunay Triangulation
- Image Processing
- Computer Graphics
- Morphing Algorithm
- Warping
- Feature Mapping


***Methodology***:

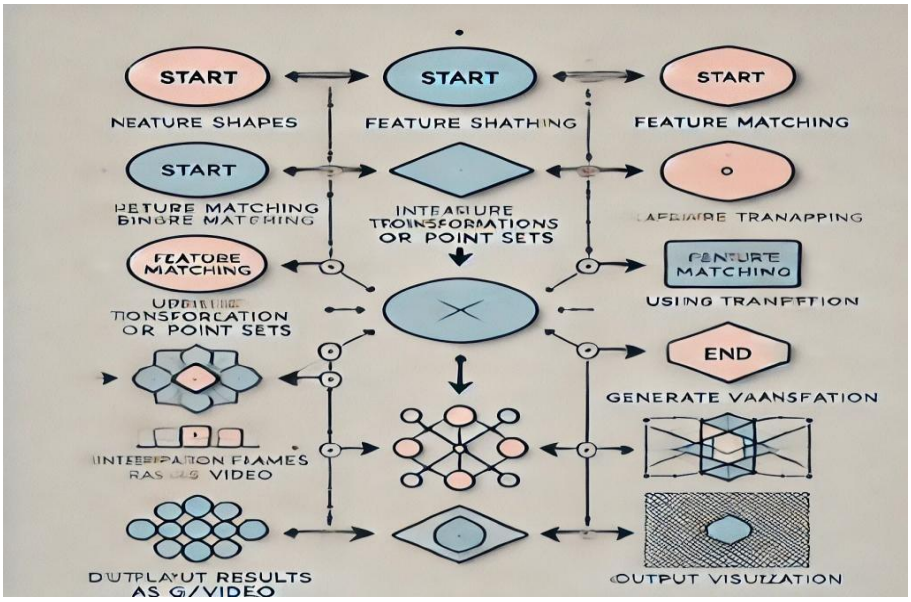
- **Input Shape Representation** – Define initial and target shapes as binary images or point sets.
- **Feature Matching –** Identify corresponding points using landmark mapping or Delaunay triangulation.
- **Interpolation & Transformation –** Apply linear interpolation, affine transformations, and warping for smooth blending.

- **Generating Intermediate Frames –** Compute transition frames using weighted averaging for animation.
- **Visualization & Output –** Display or save the morphing sequence as images or video.
- **Optimization (Optional) –** Use smoothing techniques or AI-based feature detection for better results.

***Tools***:

- **Programming Language –** Python
- **Libraries & Frameworks:**
  a) **OpenCV –** Image processing and transformation
  b) **NumPy –** Numerical computations and matrix operation
  c) **Matplotlib –** Visualization of intermediate shape transitions
- **Software & IDEs:**
  a) **Jupyter Notebook / Google Colab –** Interactive development
  b) **PyCharm / VS Code –** For scripting and debugging
- **Optional Tools:**
  a) **SciPy –** Advanced interpolation techniques
  b) **Pillow –** Handling image formats
  c) **FFmpeg –** Exporting morphing sequences as video/GIF

***Block diagram***:




***References:***


1. **"Computer Graphics: Principles and Practice"** – John F. Hughes, Andries van Dam
2. **"Digital Image Processing"** – Rafael C. Gonzalez, Richard E. Wood
3. **"Morphing: Smooth Shape Transitions Using Computer Graphics"** – T. Beier and S. N
4. **"Computer Vision: Algorithms and Applications"** – Richard Szeliski

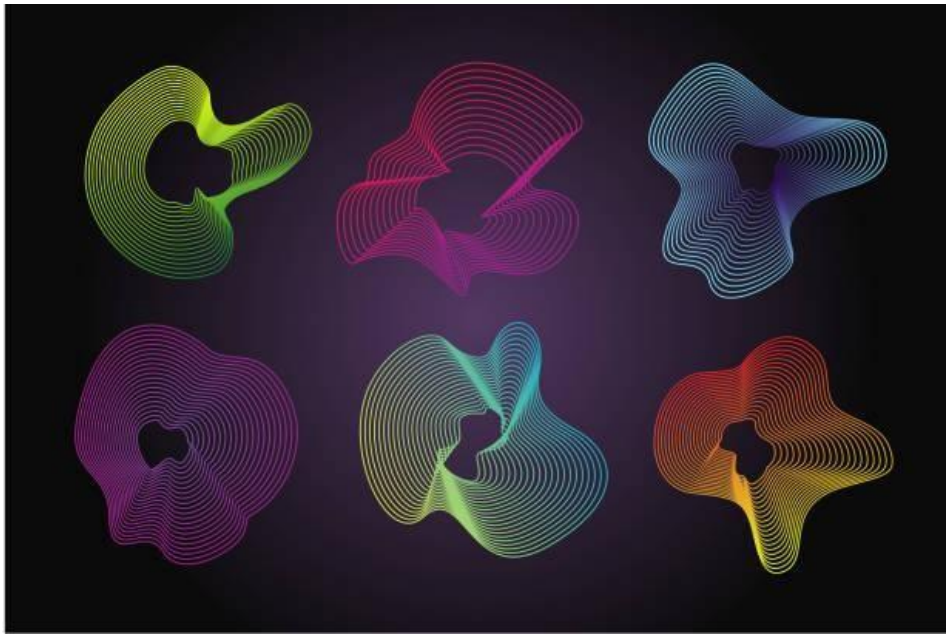
# TABLE OF CONTENTS

# LIST OF FIGURES

# INTRODUCTION

Shape morphing using Delaunay triangulation is a computational technique that transforms one shape into another by leveraging the geometric properties of Delaunay triangulation. This method is particularly effective in preserving the structural integrity of shapes during the morphing process. The introduction of this technique often highlights its ability to ensure smooth transitions between shapes while minimizing distortions, thanks to the connectivity and spatial relationships inherent in Delaunay triangulation.

This approach has found applications in various fields, including computer graphics, animation, and scientific visualization. It is especially useful for tasks requiring high precision and adaptability, such as reconstructing 3D shapes or morphing binary images while maintaining their structural features.



**Shape Morphing**

**Figure (1)**

# DESIGN, METHODOLOGY AND IMPLEMENTATION

## 2.1. SYSTEM ARCHITECHTURE

1. Input Module: Accepts two shapes or sets of points as input. Prepares the data for triangulation by ensuring proper alignment and scaling.

2. Delaunay Triangulation Module: Constructs Delaunay triangulations for the input shapes. Ensures that the triangulation adheres to the geometric properties of Delaunay, such as maximizing the minimum angle of triangles.

3. Correspondence Mapping: Establishes point-to-point correspondences between the two shapes. Ensures smooth transitions by aligning the triangulated meshes.

Input Images (Shape 1, Shape 2)

↓

Preprocessing (Resize, Normalize)

↓

Feature Extraction (Contours, Key points)

↓

Point Matching (Correspondence)

↓

Interpolation (Intermediate Shapes Generation)

↓

Rendering (Draw and Display Frames)

↓

Output (Animation/GIF)

**Block Diagram**

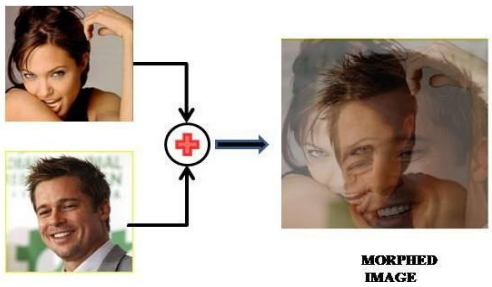



**Figure (2)**

## 2.2. COMPONENTS OVERVIEW

1. **Input Data Preparation:** Collects and preprocesses the input shapes or point sets. Ensures alignment, scaling, and normalization for accurate triangulation.

2. **Delaunay Triangulation Module:** Constructs Delaunay triangulations for the input shapes. Maintains geometric properties like maximizing the minimum angle of triangles.

3. **Correspondence Mapping:** Establishes point-to-point correspondences between the two triangulated shapes. Aligns the meshes to ensure smooth morphing transitions.

4. **Morphing Algorithm:** Interpolates between the two triangular tedshapes. Gradually transforms one shape into another while preserving structural integrity.

5. **Visualization and Output:** Generates the final morphed shape.

## 2.3. SOFTWARE AND TOOLS USED

1. **Programming Languages: Python:** Widely used for implementing Delaunay triangulation algorithms, with libraries like SciPy. Spatial and matplotlib for visualization.

2. **C++:** Known for its efficiency in computational geometry tasks, often used for high-performance implementations.

3. **Libraries and Frameworks: OpenCV:** Provides functions for Delaunay triangulation and image processing, making it ideal for morphing applications.

4. **CGAL (Computational Geometry Algorithms Library):** Offers robust tools for Delaunay triangulation and other geometric computations.

5. **Dlib:** Useful for facial landmark detection, which can be combined with Delaunay triangulation for face morphing.

# WORKING PRINCIPLE

1. **Point Sampling:** The shapes to be morphed are represented as sets of points. These points are sampled from the boundaries or surfaces of the shapes.

2. **Triangulation:** Delaunay triangulation is applied to these points. This creates a mesh of triangles that optimally connects the points, ensuring no point lies inside the circumcircle of any triangle. This property helps avoid skinny triangles and ensures a robust structure.

3. **Correspondence Mapping:** Corresponding points between the two shapes are identified. This step is crucial for ensuring that the morphing process transitions smoothly from one shape to another.

4. **Interpolation:** The vertices of the triangles are interpolated between the two shapes. This interpolation can be linear or follow more complex mathematical functions, depending on the desired morphing effect.

5. **Mesh Transformation:** The triangles in the mesh are transformed based on the interpolated vertices. This transformation ensures that the intermediate shapes maintain structural integrity.

6. **Rendering:** The intermediate shapes are rendered to visualize the morphing process. This step may involve smoothing or additional processing to enhance visual quality.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Input: Define your shapes (each shape is a series of 2D coordinates)
# Butterfly → Bird Example
shape1_butterfly = np.array([[-0.4, 0], [-0.3, 0.3], [-0.2, 0.4], [0, 0.6],
                             [0.2, 0.4], [0.3, 0.3], [0.4, 0], [0.3, -0.3],
                             [0.2, -0.4], [0.4, -0.6], [-0.2, -0.4], [-0.3, -0.3], [-0.4, 0]])
shape2_bird = np.array([[-0.5, 0], [-0.3, 0.4], [-0.1, 0.5], [0.1, 0.6], [0.3, 0.5],
                        [0.5, 0.3], [0.6, 0.1], [0.5, -0.1], [0.3, -0.2], [0, 0], [-0.5, 0]])

# Moon → Sun
shape1_moon = np.column_stack((0.5 * np.cos(np.linspace(0, 2 * np.pi, 100)) - 0.2 * np.cos(2 * np.linspace(0, 2 * np.pi, 100)),
                               0.5 * np.sin(np.linspace(0, 2 * np.pi, 100))))
shape2_sun = np.column_stack((0.6 * np.cos(np.linspace(0, 2 * np.pi, 100)) + 0.1 * np.cos(8 * np.linspace(0, 2 * np.pi, 100)),
                              0.6 * np.sin(np.linspace(0, 2 * np.pi, 100)) + 0.1 * np.sin(8 * np.linspace(0, 2 * np.pi, 100))))

# Tree → Lightning
shape1_tree = np.array([[-0.1, -0.5], [-0.1, 0], [-0.3, 0.2], [-0.1, 0.4], [0, 0.6],
                        [0.1, 0.4], [0.3, 0.2], [0.1, 0.4], [0.1, -0.5], [-0.1, -0.5]])
shape2_lightning = np.array([[0, 0.5], [-0.2, 0.2], [0.1, 0.1], [-0.1, -0.1],
                             [0.3, -0.3], [0.1, -0.5], [-0.1, -0.3], [0, 0.5]])

# Wave → Dolphin
shape1_wave = np.column_stack((0.5 * np.cos(np.linspace(0, 2 * np.pi, 100)) + 0.2 * np.cos(3 * np.linspace(0, 2 * np.pi, 100)),
                               0.5 * np.sin(np.linspace(0, 2 * np.pi, 100))))
shape2_dolphin = np.column_stack((0.5 * np.cos(np.linspace(0, 2 * np.pi, 100)) + 0.1 * np.sin(3 * np.linspace(0, 2 * np.pi, 100)),
                                  0.3 * np.sin(np.linspace(0, 2 * np.pi, 100)) + 0.2 * np.cos(4 * np.linspace(0, 2 * np.pi, 100))))
```

**Figure (3.1)**

```python
# Function to match the points in two shapes
def match_points(shape1, shape2):
    if len(shape1) > len(shape2):
        shape2 = np.vstack([shape2, shape2[:len(shape1) - len(shape2)]])
    elif len(shape1) < len(shape2):
        shape1 = np.vstack([shape1, shape1[:len(shape2) - len(shape1)]])
    return shape1, shape2

# Match points to have the same number of points in each shape
shape1_butterfly, shape2_bird = match_points(shape1_butterfly, shape2_bird)
shape1_moon, shape2_sun = match_points(shape1_moon, shape2_sun)
shape1_tree, shape2_lightning = match_points(shape1_tree, shape2_lightning)
shape1_wave, shape2_dolphin = match_points(shape1_wave, shape2_dolphin)

# Function to animate the transition between the shapes
def generate_intermediate_steps(shape1, shape2, num_frames=50):
    steps = []
    for t in np.linspace(0, 1, num_frames):
        intermediate_shape = (1 - t) * shape1 + t * shape2
        steps.append(intermediate_shape)
    return steps

# Generate intermediate shapes for each pair of shapes
steps_butterfly_bird = generate_intermediate_steps(shape1_butterfly, shape2_bird)
steps_moon_sun = generate_intermediate_steps(shape1_moon, shape2_sun)
steps_tree_lightning = generate_intermediate_steps(shape1_tree, shape2_lightning)
steps_wave_dolphin = generate_intermediate_steps(shape1_wave, shape2_dolphin)

# Initialize figure for displaying all intermediate steps
fig, axes = plt.subplots(2, 2, figsize=(12, 12))  # A 2x2 grid of subplots for intermediate steps
plt.subplots_adjust(wspace=0.3, hspace=0.3)
```

**Figure (3.2)**

```python
# Animation function
def animate(frame):
    t = frame / 50
    axes[0, 0].clear()
    axes[0, 0].plot((1 - t) * shape1_butterfly[:, 0] + t * shape2_bird[:, 0],
                    (1 - t) * shape1_butterfly[:, 1] + t * shape2_bird[:, 1], "r-", linewidth=2)
    axes[0, 0].set_title("Butterfly → Bird")
    axes[0, 0].axis("equal")
    axes[0, 0].axis("off")

    axes[0, 1].clear()
    axes[0, 1].plot((1 - t) * shape1_moon[:, 0] + t * shape2_sun[:, 0],
                    (1 - t) * shape1_moon[:, 1] + t * shape2_sun[:, 1], "b-", linewidth=3)
    axes[0, 1].set_title("Moon → Sun")
    axes[0, 1].axis("equal")
    axes[0, 1].axis("off")

    axes[1, 0].clear()
    axes[1, 0].plot((1 - t) * shape1_tree[:, 0] + t * shape2_lightning[:, 0],
                    (1 - t) * shape1_tree[:, 1] + t * shape2_lightning[:, 1], "g-", linewidth=2)
    axes[1, 0].set_title("Tree → Lightning")
    axes[1, 0].axis("equal")
    axes[1, 0].axis("off")

    axes[1, 1].clear()
    axes[1, 1].plot((1 - t) * shape1_wave[:, 0] + t * shape2_dolphin[:, 0],
                    (1 - t) * shape1_wave[:, 1] + t * shape2_dolphin[:, 1], "m-", linewidth=2)
    axes[1, 1].set_title("Wave → Dolphin")
    axes[1, 1].axis("equal")
    axes[1, 1].axis("off")

# Create animation
ani = animation.FuncAnimation(fig, animate, frames=50, interval=100, blit=False)
# Save animation as GIF
ani.save("multi_shape_morphing.gif", writer="pillow")
plt.show()
```

**Figure (3.3)**

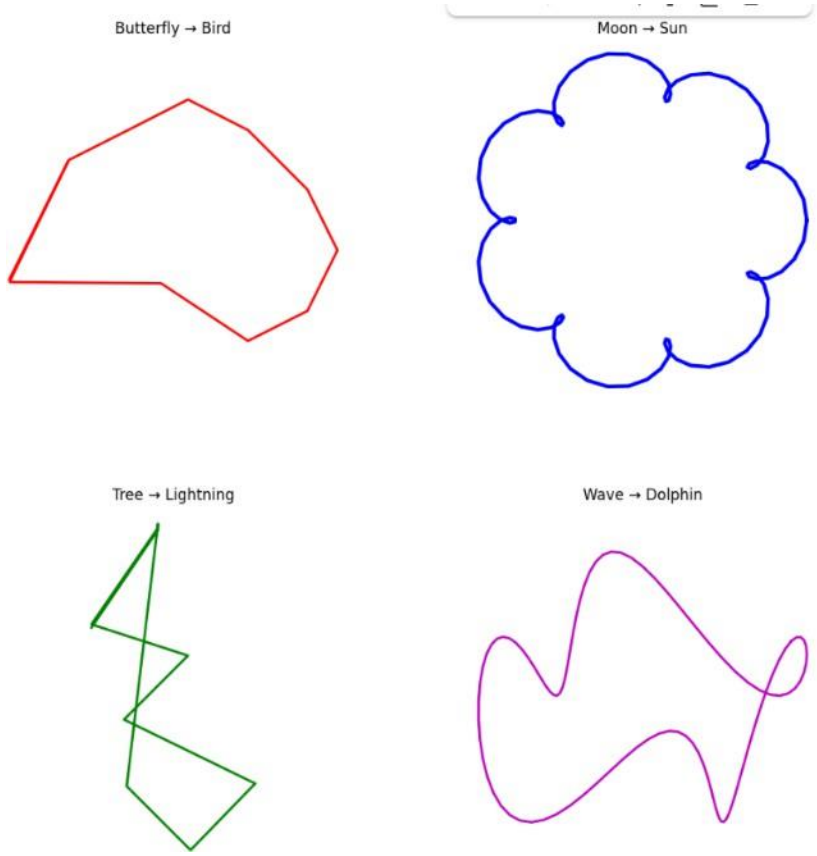



**Figure (3.4)**

# RESULTS AND DISCUSSION

**4.1 Noise Handling:** Delaunay triangulation is computationally efficient, making it suitable for real-time applications. Its ability to adapt to varying densities of points ensures smooth transitions during morphing. The method is robust against noise, as the triangulation inherently minimizes distortions by avoiding skinny triangles.

**4.2 Challenges in Complex Shapes**: For highly intricate or irregular shapes, achieving a watertight mesh can be challenging. Researchers are exploring hybrid methods that combine Delaunay triangulation with other techniques to address these issues

**4.3 Applications in Animation**: In computer graphics, Delaunay triangulation is often used for creating seamless animations between two shapes, ensuring that the intermediate frames are visually coherent.

```python
import cv2
import numpy as np
import imageio
import matplotlib.pyplot as plt

# Load the two images
img1 = cv2.imread("/content/amit.jpg")
img2 = cv2.imread("/content/modi.jpg")

# Validate image loading
if img1 is None or img2 is None:
    raise ValueError("One or both image paths are incorrect or missing.")

# Resize second image to match the first image size
img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

# Blending factor (0.5 = equal contribution from both)
alpha = 0.5

# Create the blended (morphed) image
blended = cv2.addWeighted(img1, 1 - alpha, img2, alpha, 0)

# Convert BGR to RGB for correct display in matplotlib
blended_rgb = cv2.cvtColor(blended, cv2.COLOR_BGR2RGB)

# Show the result
plt.imshow(blended_rgb)
plt.axis("off")
plt.title("Single-Step Morphed Image")
plt.show()

# Save the output
cv2.imwrite("morphed_single_image.jpg", blended)
print("✅ Saved: morphed_single_image.jpg")
```

**Figure (4.1)**

Input 1 Input 2

**Figure (4.2)**


**Figure (4.3)**

## 4.4 Code Functionality

1. Identifies key points (landmarks) on both shapes for triangulation.
2. Constructs Delaunay triangulations for the key points of both shapes.
3. Ensures that the triangulation adheres to the geometric properties of Delaunay.
4. Maps corresponding triangles between the two shapes based on the triangulated meshes.

Aligns the triangles for smooth morphing.

# CONCLUSION

In conclusion, shape morphing using Delaunay triangulation is a robust and efficient technique for transforming one shape into another while preserving structural integrity. By leveraging the geometric properties of Delaunay triangulation, such as maximizing the minimum angle of triangles and maintaining spatial relationships, this method ensures smooth and visually appealing transitions

The approach has proven to be highly versatile, with applications in computer graphics, animation, scientific visualization, and even medical imaging. Its ability to handle complex shapes and adapt to various scenarios makes it a valuable tool in computational geometry.

# REFERNCES

[1] T. Beier and S. Neely, "Feature-Based Image Metamorphosis," Proc. SIGGRAPH, 1992.
[2] Dlib: http://dlib.net/
[3] OpenCV: https://opencv.org/
[4] Learn OpenCV: https://www.learnopencv.com/face-morph-using-opencv-cpp-python/
[5] J. Noh and U. Neumann, "A Survey of Video-Based Motion Capture," Computer Graphics Forum, vol. 22, no. 4,