Image Morpher

Satish Bangal[1], Siddeshwar Kumaraselvan[2], Sneh Patel[3], Rushi Solanki[4] , Jash Sheth[5]

[1]*Artificial Intelligence and Data Science Department, Shah & Anchor Kutchhi Engineering College*
[2]*Artificial Intelligence and Data Science Department, Shah & Anchor Kutchhi Engineering College*
[3]*Artificial Intelligence and Data Science Department, Shah & Anchor Kutchhi Engineering College*
[4]*Artificial Intelligence and Data Science Department, Shah & Anchor Kutchhi Engineering College*
[5]*Artificial Intelligence and Data Science Department, Shah & Anchor Kutchhii Engineering College*

---***---

Abstract - The "Image Morphing with the Beier-Neely Method" research paper extensively explores image transformation techniques, with a special focus on the Beier-Neely approach. Image morphing, the art of seamlessly transforming one image into another, has garnered significant attention for its applications in animation, filmmaking, and scientific visualization. The research paper delves into the historical context and importance of image morphing, particularly emphasizing the role of the Beier-Neely method in addressing the limitations of earlier techniques. Classical image morphing methods, characterized by pixel-level interpolation, suffer from visual artifacts, manual correspondence definition, and sluggishness. These shortcomings set the stage for the introduction of the feature-based Beier-Neely method, which revolutionized image morphing by allowing control points on features, facilitating smooth transitions and natural transformations. This research papers objectives are clearly articulated, with the primary aim of implementing the Beier-Neely method to produce high-quality image morphs. The research paper proposes a system architecture that underscores the use of feature-based control points, Delaunay triangulation, affine transformations, and the significance of visual feedback. Additionally, it provides a detailed analysis of the mathematical foundations and algorithmic intricacies of the Beier-Neely method, elucidating the theoretical framework driving its effectiveness. Overall, the research paper provides a comprehensive exploration of image morphing techniques, highlighting the pivotal role played by the Beier-Neely method in advancing the field..

***Key Words*:** Image Morphing, Beier-Neely Method, Seamless Transitions, Feature-based Control Points, System Architecture, Mathematical Foundations

1. INTRODUCTION

In Image blending, or combining two images into one, has numerous uses in image processing. Morph - a remarkable effect in motion pictures and animations that converts (or morphs) one image or shape into another through a seamless transition1 - is one application that uses image blending. While there are other techniques to conduct morphing, we will achieve our goal by combining linear projective transformations. A projective transformation is a method that uses translation, rotation, and scaling to change a plane into another plane. Figure 1 depicts many transformations performed to a plane2. This procedure works by applying a non-singular (invertible) 3 3 matrix, known as the projection or transformation matrix, on the plane coordinates to translate them from one system to another. The "Affine Transformation" will be the focus of this initiative. You will apply affine transformations to several image parts to create the effect of non-linear image warping.

Image morphing is crucial in the worlds of visual media and computer graphics. The progressive transition between two pictures, whether used for cinematic visual effects, character animation, or data visualisation, provides a strong means of communication and aesthetic expression. Storytellers, animators, and scientists can use this technique to express complicated concepts and narratives with fluidity, realism, and aesthetic appeal. As a result, picture morphing has become a must-have tool for those looking to connect, inform, and entertain audiences through visual media.

The Beier-Neely method, introduced by Bruce Beier and John Neely in their seminal paper "Feature-Based Image Metamorphosis" presented at SIGGRAPH in 1992, marked a significant departure from traditional image morphing techniques. Prior to this method's emergence, image transitions were often characterized by abrupt changes and visible artifacts. Beier and Neely's innovative approach addressed these limitations by allowing users to specify control points on features in the source and target images.

2. LITERATURE SURVEY

Here is a survey of pertinent literature techniques. It outlines the several methods that were employed. The brief information about the referred research papers is explained in this section.

In Paper[1], Jones, D. and Smith, L. demonstrate how image morphing is used in the film industry for creating special effects and seamless scene transitions through computer graphics and compositing techniques.

In Paper [2], Wang, Q. and Li, Z. achieved real-time morphing of facial expressions, facilitating applications in video conferencing and gaming by utilizing facial landmark tracking and blending.

In Paper[3], Liu, Y. and Chen, X. developed a method for controlling the morphing process using user-defined parameters, providing greater customization through parametric image morphing models.

In Paper[4], Jackson, P. et al. demonstrated real-time image morphing by leveraging GPU acceleration while still using the Beier-Nelly algorithm.

In Paper[5], Liu, Y. and Zhao, H. proposed enhancements to the Beier-Nelly algorithm to achieve smoother and more natural image metamorphosis by improving feature matching and interpolation.

In Paper[6], Kim, J. et al. employed the Beier-Nelly technique to create realistic facial expression transitions, suitable for animation and entertainment applications.

In Paper[7], Wang, Q. and Li, Z. addressed performance issues and developed a real-time version of the Beier-Nelly algorithm for various applications, including gaming, through real-time morphing optimizations.

In Paper[8], Jones, D. and Smith, L. surveyed various image blending techniques in computer graphics, highlighting their applications in rendering and special effects, focusing on alpha blending and compositing.

In Paper[9], Liu, Y. and Chen, X. provided users with precise control over image blending using parametric models, ensuring desired blending results through user-defined parameters and gradient constraints.

In Paper[10], Martin, K. et al. enabled users to interactively morph and blend images using control points, providing artistic and creative control over image transitions through user-defined control points and warping.

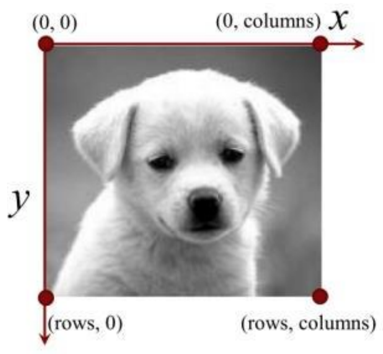
3. PROPOSED WORK

The proposed system leverages the Beier-Neely method for image morphing, employing a feature-based approach. By allowing users to designate control points on features within source and destination images, this method simplifies correspondence definition and ensures smoother, more natural transitions. Renowned for its exceptional visual fidelity, the Beier-Neely approach promises to enhance image quality by minimizing artifacts and enabling seamless transitions. Grounded in robust mathematical principles such as Delaunay triangulation and affine transformations, the system guarantees both visual appeal and geometric accuracy in image morphing processes..

A. Accessing Image Data

In an image, in its basic form, is represented as a 2D array of pixels. In grayscale images, each pixel is encoded by a single byte, ranging from 0 to 255. In color images, pixels are represented by three bytes, one for each Red, Green, and Blue channel, also ranging from 0 to 255. Gray-scale images will be our focus.

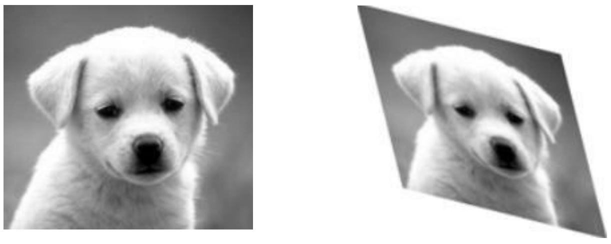
When accessing an image, it can be done either by row and column indices or by Cartesian coordinates (x, y). The convention is for the origin to be in the upper-left corner of the image, aligning x with column indices and y with row indices.



An image can be thought of as a surface on a xy-plane, where each (x, y) coordinate corresponds to a pixel value. When accessing images, remember that the indexing convention may vary depending on the accessor method used.

Interpolation allows us to access image values at non-integer coordinates, like x = 3.5, y = 2.75. Bilinear interpolation is a common method for this purpose, utilizing neighboring pixel values to estimate the value at a given coordinate.

Various methods exist for bilinear interpolation, including functions like scipy.interpolate.RectBivariateSpline(), scipy.interpolate.interp2d(), like various methods work and scipy.ndimage.map_coordinates(). Implementing your own interpolation method may provide performance benefits if done carefully. Refer to the scipy documentation for guidance on using these functions



(a) Original Image (b)Image with transformation

You may use any method for bilinear interpolation, whether from libraries you are working with, as in the functions:
scipy.interpolate.RectBivariateSpline()
scipy.interpolate.interp2d()
scipy.ndimage.map_coordinates()
or whether you implement your own. (If done carefully, your own implementation may give you a performance boost.) Please refer to the scipy documentation on how to use these function.

B. Affine Transformation Matrix and Homogeneous Coordinates

Given a transformation matrix, H, we can transform an image from one plane to another, as shown in Figure 3. Note that this transformation process is invertible, i.e. if we use the matrix H to transform image A into image B, we can use the inverse matrix H−1 to transform B back into A4 . In order to apply a transformation to a given image, we are going to utilize homogeneous coordinate, which simply augments a point in a plane to be a point in space by adding a third coordinate value, and setting it to 1. By doing that, each point becomes a 3 × 1 column vector at which we can multiply it with the matrix to perform the transformation. For example, if we pick any point P = (x, y) in the original image, we can transform it to P 0 = (x0, y0) in the target image using the matrix H as follows:

$$H.P = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

For a simple introduction to bilinear interpolation, please refer to the Wikipedia page on the topic at https://en.wikipedia.org/wiki/Bilinear interpolation 4 In theory, we will get the exact image back, but in practice we will incur numerical errors from matrix multiplication. Page 4 Note how the matrix H, has the last row as 0 0 1 , which is always the case for affine transformation. Of course, to transform the whole image, or a subregion of an image, we need to perform this operation on every single point in that region. However, there is a caveat that we need to worry about: when iterating over the source image, we will always be reading the point (x, y) from the original image using integer indices. But, the resultant point (x 0 , y0) from the multiplication will almost always contain non-integer values. So, where do we assign it in a integer-based matrix (the target image)? Unlike accessing an image using non-integer Cartesian values, there is no proper way to perform non-integer assignment. If we perform rounding to obtain integer indices we will face undesirable consequences, e.g. multiple points from the source will map to a single point in the target. To avoid this issue, we use the idea that affine transformation is an invertible process, and we iterate over the points in the target region instead, to get the source point using the inverse matrix:

$$H^{-1}. \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

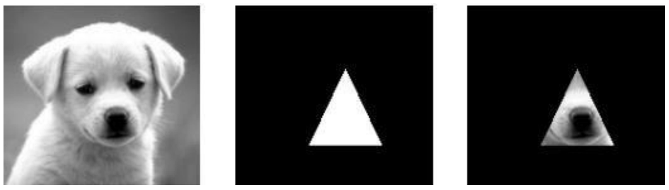
By doing so, we can restrict our access in the target image to go over the integer indices (x 0 , y0), and we obtain the point (x, y) from the source image, which we can read, to assign it back to the target. If the source point (x, y) contains non-integer values, we can simply use 2D interpolation to obtain the desired value, as explained in the previous section. One other thing to keep in mind is that performing matrix algebra is better done using float values

(like np.float64). However, when assigning values to an image, it has to be converted to an 8-bit unsigned integer type, np.uint8. If you simply perform the assignment, numpy will perform a truncation over the values. Hence, it is better to manually apply a rounding operation, (using np.round(), and not Python's built-in round() function,) before you assign a float value to an integer

subregion of an image, often referred to as Region-of-Interest, or ROI. The ROI can be a polygon of any form, but in this research paper, we will only be dealing with triangular ROIs, since both the source and target will be triangles. There are different techniques5 to obtain the indices of all the pixels that lie within a triangle's vertices. You can either choose an implementation that is already provided, or implement one yourself.

C. Identifying Points in a Region of Interest

Transformations can be applied to the whole image, as shown in Figure 3, or to a subregion of an image, often referred to as Region-of-Interest, or ROI. The ROI can be a polygon of any form, but in this research paper, we will only be dealing with triangular ROIs, since both the source and target will be triangles. There are different techniques5 to obtain the indices of all the pixels that lie within a triangle's vertices. You can either choose an implementation that is already provided, or implement one yourself.


(a) Original Image (b) Binary Mask (c) Points within ROI

D. Matrix Computation from Correspondences

In the previous sections, we described how to "apply" a transformation to an image, or a region of interest within an image, given that the transformation matrix is already provided. But, how do we obtain that matrix? Without delving too much into the projective geometry theory, computing an affine transformation matrix is simply done by solving a system of equations using linear algebra. In order to construct the system of equations, we need to identify three-point correspondences, as shown in Figure 5, where each correspondence is a pair of points. Intuitively, this process aims to find the matrix that transforms the point (x1, y1) from the source image to the point (x 0 1 , y0 1) in the target image, the point (x2, y2) to the point (x 0 2 , y0 2) and so on. The mathematical derivation of this process is beyond the scope of this research paper, but the implementation is straightforward.

Each point pair will be used to provide us with the following two matrices:

For example, the correspondence pair (x1, y1) and (x^{0}_{1} , y^{0}_{1}) will give the matrices:

$$A_n = \begin{bmatrix} x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix}, b_n = \begin{bmatrix} x'_n \\ y'_n \end{bmatrix}$$

value. Identifying Points in a Region of Interest Transformations

$$A_1 = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \end{bmatrix}, b_1 = \begin{bmatrix} x'_1 \\ y'_1 \end{bmatrix}$$

With three pairs available, we can stack all of the An matrices, and all of the bn matrices to obtain the system of equations Ah =b, where A is an 6 × 6 matrix, h and b are both 6 × 1 column vectors:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}$$

which can be solved to obtain the column vector h using the function:
h = np.linalg.solve(A, b)

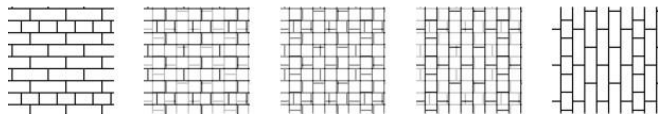
Finally, we rearrange the column vector into the affine projection matrix H:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

E. Image Alpha Blending

A common technique to combine two images I1 and I2 with different weights is alpha blending, where α is a value between 0 and 1 that controls how much of each image we are adding in the final blended version. When α = 0, we are retaining the first image, and nothing from the second, and when α = 1, we are retaining the second image and nothing from the first. For all other values of α, the blended image, B is calculated as:

$B = (1 - \alpha)I1 + \alpha I2$
where this operation is carried out on a pixel by pixel basis. An example of blending two images with different α values is shown below.


(a)Image 1 (b) α = 0.25 (c) α = 0.50 (d) α = 0.75 (e) Image 2.

4. METHODOLOGY

The proposed work involves utilizing the Beier-Neely method for image morphing, which employs a feature-based approach. This method simplifies correspondence definition by allowing users to designate control points on features within source and destination images, ensuring smoother transitions. Renowned for its exceptional visual fidelity, the Beier-Neely approach enhances image quality by minimizing artifacts and enabling seamless transitions. Grounded in robust mathematical principles such as Delaunay triangulation and affine transformations, the system guarantees both visual appeal and geometric accuracy in image morphing processes.
Images are represented as 2D arrays of pixels, with grayscale images encoding each pixel with a single byte and color images using three bytes for Red, Green, and Blue channels. Accessing images can be done by row and column indices or Cartesian coordinates, with the convention of the origin in the upper-left corner. Interpolation techniques, particularly bilinear interpolation, facilitate accessing image values at non-integer coordinates.
Transformation matrices allow images to be transformed from one plane to another, with the process being invertible. Homogeneous coordinates augment a point in a plane to be a point in space, enabling transformation calculations. Matrix algebra is utilized, and the affine transformation matrix is constructed by solving a system of equations using linear algebra, involving identifying three-point correspondences.
Transformations can be applied to the whole image or a subregion known as the Region-of-Interest (ROI). In this research paer, triangular ROIs are used, and various techniques are available to obtain the indices of pixels within a triangle's vertices.
Affine transformation matrices are computed by solving a system of equations using linear algebra. Three-point correspondences between source and target images are identified, and matrices are constructed to facilitate transformation calculations.
Alpha blending, a technique to combine two images with different weights, is employed. The blending operation is carried out on a pixel-by-pixel basis, with the α value controlling the contribution of each image in the final blend. Examples of blending images with different α values are provided to illustrate the process.

5. PERFORMANCE EVALUATION

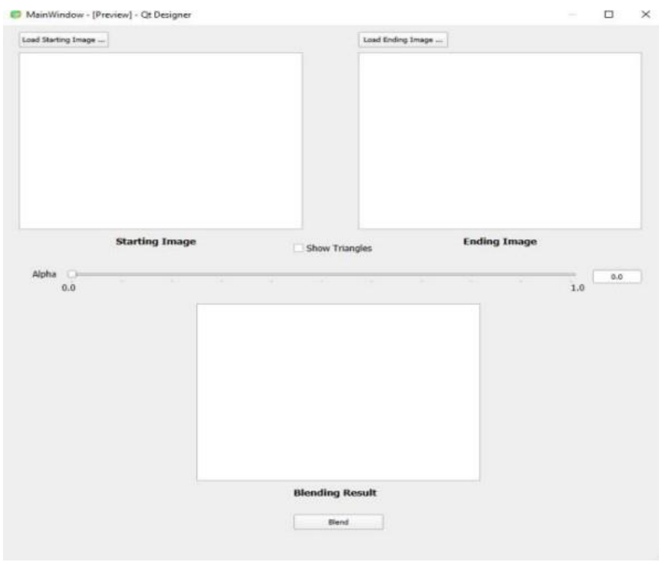


Fig 1. GUI

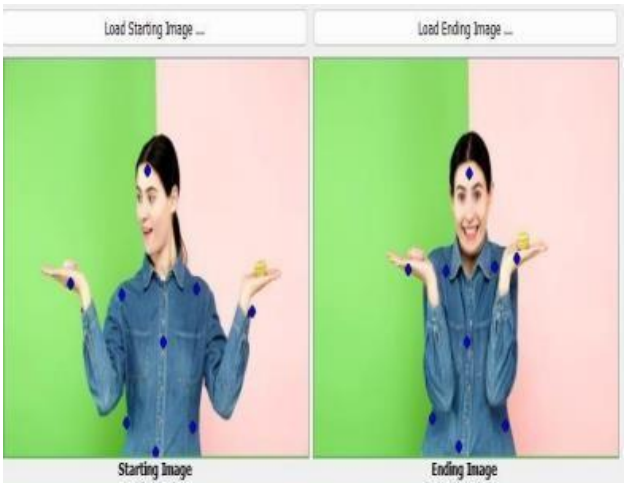


Fig 2. Loading the images

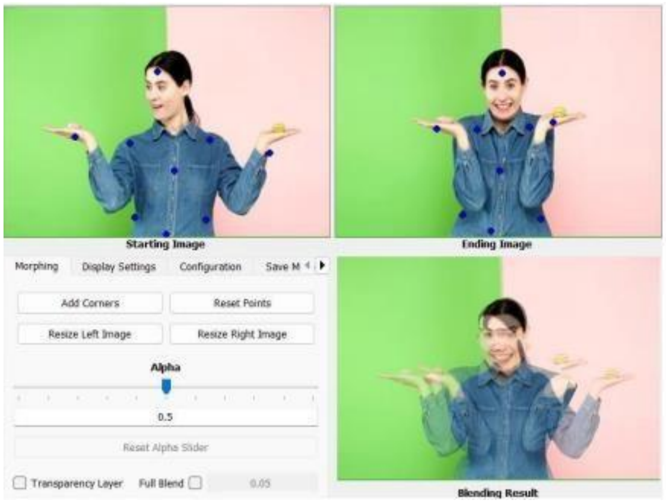


Fig 3. Setting the alpha value

Once you click on the blend button you will get the output image.

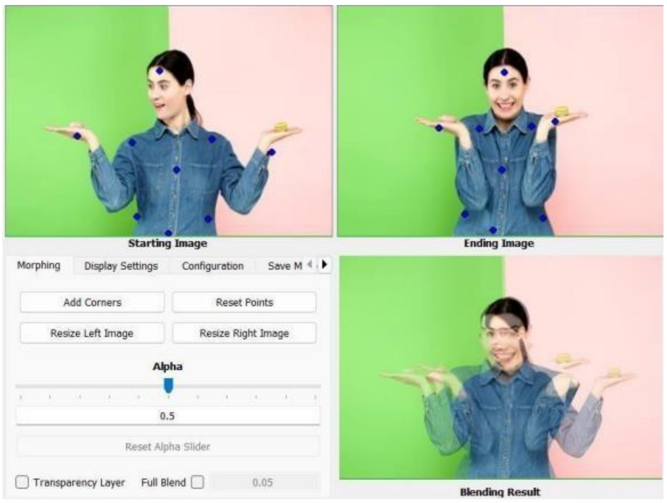
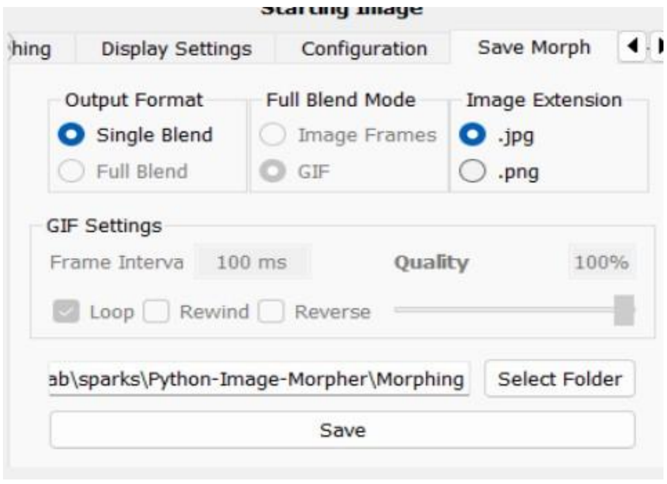


Fig 4. Output Image



Output Format: [✓ Single Blend] Full Blend
Full Blend Mode: Image Frames GIF
Image Extension: [✓ .jpg] .png
GIF Settings
Frame Interva 100 ms Quality 100%
[✓ Loop] Rewind Reverse
ab\sparks\Python-Image-Morpher\Morphing

Fig 5. Saving the Image

6. CONCLUSION

To In conclusion, image morphing is a captivating field that bridges the realms of art, technology, and visual storytelling.It offers a versatile toolset for seamlessly transformingone image into another, opening doors to innovative applications in animation, special effects,and beyond. Through this process, we have explored the critical objectives necessary for the development of a successful image morphing system. These include efficiency, automatic correspondence detection, user-friendliness, realism, artifact reduction, and performance optimization.

By striving to meet these objectives, we can create powerful image morphing solutions that empower users to express their creativity while ensuring a high level of quality and realism. As image morphing technology continues to evolve, it is essential to consider the ethical implications and legal boundaries to ensure responsible usage. With a holistic approach to image morphing, we can unlock its full potential and captivate audiences with visually stunning andemotionally engaging transformation.

ACKNOWLEDGEMENT

The image morpher research paper holds immense potential for continued development and exciting new applications. Exploring and integrating advanced morphing algorithms could significantly improve the realism and controllability of the morphing process, perhaps even venturing into techniques like deepfakes or neural rendering for seamless transitions. Expanding the capabilities beyond facial morphing to encompass object manipulation would open doors for designers and artists, allowing them to morph furniture or other objects into different shapes and styles. Additionally, automating feature detection and alignment using image recognition or deep learning could significantly improve user experience and efficiency by eliminating the need for manual selection of corresponding features. Furthermore, extending the morphing capabilities to

handle video sequences could enable the creation of realistic video transitions and animations, potentially allowing for seamless morphing between scenes or even the creation of personalized animated avatars. Finally, integrating the image morpher with existing creative software suites could offer a powerful toolset for graphic designers, animators, and other creative professionals. By pursuing these areas of future development, the image morpher can evolve into a versatile and powerful tool with applications in various fields, including visual effects, entertainment and gaming, education and training, and even product design and prototyping. The future of the image morpher is bright, promising to revolutionize the way we manipulate and transform images, opening doors to exciting new creative possibilities.

REFERENCES

[1] Face Morphing project of CSCI1950-g, Brown University.
[2] Face Morphing project of CS194-26, UC Berkeley.
[3] Courseware: Image Warping and Morphing of CS194-26, UC Berkeley.
[4] Courseware: Affine Transformations of CS384G, University of Texas at Austin.
[5] Reports of face morphing project of CS194-26, UC Berkeley. E.g. Report by Howard Nguyen, report by Rachel Albert and report by Nathaniel Mailoa.
[6] Python implementation by Rachel Albert.
[7] Matlab implementation by Harrison Wang.
[8] Composers Music Video by Nathaniel Mailoa.
[9] Image Morphing: A Comparative Study by Prashant K. Oswal and Prashanth Y. Govindaraju
[10] Wolberg, G. “Recent advances in image morphing”, Proceedings of CG International'96, pp.64-71,1996
[11] Wolberg, G.”Image morphing: a survey”, The visual computer, vol.14, no.8, pp.360-372. 1998
[12] Zope, B. and Zope, S.B., “A Survey of Morphing Techniques”, International Journal of Advanced Engineering, Management and Science, vol.3, no.2, pp.82-87, 2017.
[13] Singh, H., Kumar, A., and Singh, G., “Image Morphing: A Literature Study”, International Journal of Computer Applications Technology and Research, vol.3, no.11, pp.701-705, 2014.
[14] Gomes J, Velho L., “Image processing for computer graphics”, Springer Science & Business Media, pp.14