MEDIAL AXIS TRANSFORMATION:

Continuous erosion and dilation are two operations that are commonly used in image processing. They involve modifying the shape of an image by removing or adding pixels to its boundaries.

The basic idea behind image skeletonization is to find a set of pixels that represent the "backbone" or "centerline" of the image. These pixels should be located as close as possible to the center of the image, and should be able to capture the essential features of the object represented by the image.

To achieve this, the skeletonization process typically involves applying a series of erosion and dilation operations to the original image, with the goal of gradually reducing the size of the object while preserving its overall shape.

During the erosion process, the image is gradually "eroded" by removing pixels from its boundaries. This has the effect of making the object smaller and more compact, while preserving its overall shape.

During the dilation process, the image is gradually "expanded" by adding pixels to its boundaries. This has the effect of making the object larger and more "blob-like", but it also helps to fill in any gaps or holes in the original image.

By alternating between these two processes, it is possible to gradually transform the original image into a "skeleton" or "centerline" representation that captures the essential features of the object. The resulting skeleton is typically a thin set of lines or curves that represents the approximate centerline of the object, and can be used for a variety of purposes, such as object recognition, shape analysis, and computer vision applications.

CODE:

```
import cv2
import numpy as np
import copy

# Load the image and convert to grayscale
image = cv2.imread('cvSmall.png',cv2.IMREAD_GRAYSCALE)

# Threshold the image to obtain a binary image
ret, binary_image = cv2.threshold(image, 210, 255, cv2.THRESH_BINARY)

# Invert the binary image to make object pixels 1 and background pixels 0
binary_image = cv2.bitwise_not(binary_image)

# Create a copy of the thresholded image for visualization purposes
threshold_backup = copy.deepcopy(binary_image)

# Initialize the skeleton image as an all-zero array of the same size as the thresholded
image
skeleton = np.zeros_like(binary_image)
```

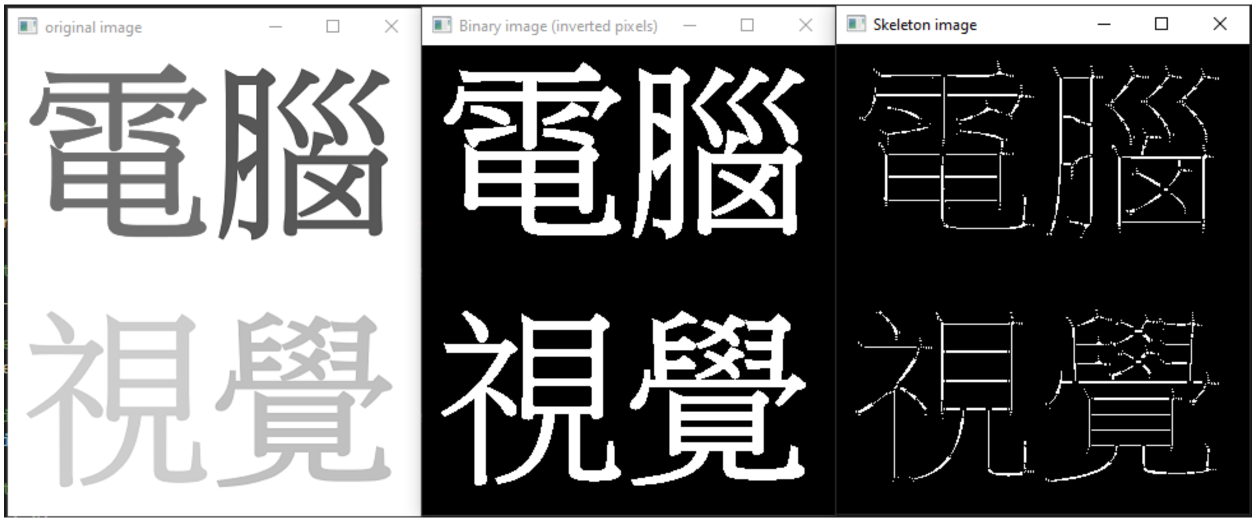
```
# Perform the medial axis transform using a for loop
while np.sum(binary_image) > 0:
    # Erode the thresholded image
    eroded = cv2.erode(binary_image, None)
    # Dilate the eroded image to obtain a temporary image
    temp = cv2.dilate(eroded, None)
    # Subtract the temporary image from the original thresholded image
    temp = cv2.subtract(binary_image, temp)
    # Use a bitwise OR operation to add the temporary image to the skeleton image
    skeleton = cv2.bitwise_or(skeleton, temp)
    # Update the thresholded image to be the eroded image
    binary_image = eroded.copy()

# Display the original image, the thresholded image, and the resulting skeleton image
while(True):
    cv2.imshow('original image', image)
    cv2.imshow("Binary image (inverted pixels)",threshold_backup)
    cv2.imshow("Skeleton image",skeleton)

    # Wait for the user to press the 'ESC' key to exit
    if cv2.waitKey(1) == 27:
        break

# Destroy all open windows
cv2.destroyAllWindows()
```

<u>THE OUTPUT IMAGE ON EXECUTING THE ABOVE CODE IS GIVEN BELOW:</u>



original image
電腦
視覺
Binary image (inverted pixels)
電腦
視覺
Skeleton image
電腦
視覺