1. Importing Libraries

```
import tkinter as tk
from tkinter import filedialog
import cv2
import numpy as np
from PIL import Image, ImageTk
```

- **tkinter (tk):** A Python library for creating graphical user interfaces (GUIs).
- **filedialog:** A submodule of tkinter for opening file dialogs.
- **cv2 (OpenCV):** An open-source computer vision and machine learning software library.
- **numpy (np):** A library for numerical computing with support for large, multi-dimensional arrays and matrices.
- **PIL (Image, ImageTk):** Python Imaging Library, used for opening, manipulating, and saving many different image file formats. ImageTk is used to display images in tkinter.

2. Initialize the GUI

```
root = tk.Tk()
root.title("Image Processing App")
```

- Creates the main window of the application with the title "Image Processing App".

3. Create a Grid Layout

```
root.grid_rowconfigure(0, weight=1)
root.grid_columnconfigure(0, weight=1)
root.grid_columnconfigure(1, weight=1)
```

- Configures the grid layout of the main window to have two columns with equal weight, allowing them to resize evenly.

4. Create Canvas Objects

```
left_canvas = tk.Canvas(root, width=500, height=600)
right_canvas = tk.Canvas(root, width=500, height=600)
```

- Creates two canvas widgets within the main window to display images.

5. Add Labels for the Canvases

```
left_label = tk.Label(root, text="Original Image")
right_label = tk.Label(root, text="Processed Image")
```

- Creates labels for the left and right canvases to indicate their purpose.

6. Place the Canvas Objects

```
left_canvas.grid(row=1, column=0, padx=10, pady=10, sticky="nsew")
right_canvas.grid(row=1, column=1, padx=10, pady=10, sticky="nsew")
```

- Places the left and right canvas widgets in the main window using the grid layout.

7. Global Variables

```
original_image = None
processed_image = None
```

- Initializes global variables to store the original and processed images.

8. Open Image Function

```
def open_image():
    global original_image
    file_path = filedialog.askopenfilename()
    if file_path:
        original_image = cv2.imread(file_path)
        display_image(original_image, left_canvas)
```

- Defines a function to open an image file using a file dialog.
- Uses OpenCV (cv2) to read the image file.
- Calls the `display_image` function to display the loaded image on the left canvas.

9. Display Image Function

```
def display_image(img, canvas):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(img_rgb)
    photo = ImageTk.PhotoImage(image=img_pil)
    canvas.create_image(0, 0, anchor=tk.NW, image=photo)
    canvas.image = photo
```

- Converts the OpenCV image (in BGR format) to a PIL image (in RGB format).
- Creates a Tkinter-compatible photo image.
- Uses the canvas's `create_image` method to display the photo image on the canvas.

10. Image Processing Operations (e.g., crop, blur, erode)

- Functions like `crop_image`, `blur_image`, `erode_image`, etc., perform various image processing operations using OpenCV functions.
- These functions modify the global variable `processed_image` and update the right canvas with the processed image using the `display_image` function.

11. Create Buttons, Menus, and Status Bar

- Buttons and menus are created to perform actions like opening an image, applying filters, etc.
- A status bar is added to display status messages.

12. Bind Keyboard Shortcuts

- Keyboard shortcuts like Ctrl+O to open an image and Ctrl+Q to quit the application are bound to corresponding functions.

13. Start the GUI Event Loop

```
root.mainloop()
```

- Starts the main event loop of the GUI, allowing the user to interact with the application.

Summary of Libraries:

- **OpenCV (cv2):** Used for image loading, manipulation, and processing.
- **Tkinter (tk):** Used for creating the GUI and handling user interactions.
- **PIL (Image, ImageTk):** Used for converting images between different formats and displaying images in Tkinter.

This code combines the capabilities of OpenCV for image processing with Tkinter for creating a user-friendly GUI, allowing users to load, process, and visualize images interactively.

Detailed Explanation of the various functions made:

1. Blur Image Function

```
def blur_image():
    global processed_image
    if original_image is not None:
        blurred = cv2.GaussianBlur(original_image, (5, 5), 0)
        processed_image = blurred
        display_image(processed_image, right_canvas)
```

- **Functionality:** Blurs the original image using a Gaussian blur filter.
- **OpenCV Function:** `cv2.GaussianBlur()`
- **Parameters:**
  - `original_image`: The original image to be blurred.
  - `(5, 5)`: Kernel size for the Gaussian blur filter (5x5 kernel).
  - `0`: Standard deviation of the Gaussian kernel along the x and y directions (0 indicates that it is calculated automatically based on the kernel size).
- **Global Variables:** Updates the `processed_image` global variable with the blurred image and displays it on the right canvas.

2. Unblur Image Function

```
def unblur_image():
    global processed_image
    if original_image is not None:
        kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
        sharpened = cv2.filter2D(original_image, -1, kernel)
        processed_image = sharpened
        display_image(processed_image, right_canvas)
```

- **Functionality:** Applies a sharpening filter to the original image to "unblur" it.
- **OpenCV Function:** `cv2.filter2D()`
- **Parameters:**
  - `original_image`: The original image to be sharpened.
  - `-1`: Depth of the output image (set to -1 to use the same depth as the input image).
  - `kernel`: Sharpening kernel used for the filter.
- **Global Variables:** Updates the `processed_image` global variable with the sharpened image and displays it on the right canvas.

3. Transparency Meter Function

```
def transparency_meter():
    global processed_image
    if original_image is not None:
        gray = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
        ret, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
        transparency_level = np.sum(mask == 255) / (mask.shape[0] *
mask.shape[1])
        processed_image = cv2.bitwise_and(original_image, original_image,
mask=mask)
        display_image(processed_image, right_canvas)
```

- **Functionality:** Determines the transparency level of the original image and applies a binary mask to make transparent areas visible.
- **OpenCV Functions:**
  - `cv2.cvtColor()`: Converts the original image to grayscale.
  - `cv2.threshold()`: Applies a threshold to the grayscale image to create a binary mask.
  - `cv2.bitwise_and()`: Applies the binary mask to the original image.
- **Parameters:**
  - `original_image`: The original image for which transparency is measured.
- **Global Variables:** Updates the `processed_image` global variable with the masked image and displays it on the right canvas.

4. Erode Image Function

```
def erode_image():
    global processed_image
    if original_image is not None:
        kernel = np.ones((5, 5), np.uint8)
        eroded = cv2.erode(original_image, kernel, iterations=1)
        processed_image = eroded
        display_image(processed_image, right_canvas)
```

- **Functionality:** Erodes the original image to reduce the size of foreground objects.
- **OpenCV Function:** `cv2.erode()`
- **Parameters:**
  - `original_image`: The original image to be eroded.
  - `kernel`: Structuring element used for erosion.
  - `iterations`: Number of times erosion is applied.
- **Global Variables:** Updates the `processed_image` global variable with the eroded image and displays it on the right canvas.

5. Dilate Image Function

```
def dilate_image():
    global processed_image
    if original_image is not None:
        kernel = np.ones((5, 5), np.uint8)
        dilated = cv2.dilate(original_image, kernel, iterations=1)
        processed_image = dilated
        display_image(processed_image, right_canvas)
```

- **Functionality:** Dilates the original image to increase the size of foreground objects.
- **OpenCV Function:** `cv2.dilate()`
- **Parameters:**
  - `original_image`: The original image to be dilated.
  - `kernel`: Structuring element used for dilation.
  - `iterations`: Number of times dilation is applied.
- **Global Variables:** Updates the `processed_image` global variable with the dilated image and displays it on the right canvas.

6. Show Histogram Function

```
def show_histogram():
    if original_image is not None:
        import matplotlib.pyplot as plt
        color = ('b', 'g', 'r')
        for i, col in enumerate(color):
            hist = cv2.calcHist([original_image], [i], None, [256], [0,
256])
            plt.plot(hist, color=col)
            plt.xlim([0, 256])
        plt.show()
```

- **Functionality:** Displays the histogram of the original image.
- **OpenCV Function:** `cv2.calcHist()`
- **Parameters:**
  - `original_image`: The original image for which the histogram is calculated.
- **Global Variables:** No global variables are modified in this function. It directly displays the histogram using Matplotlib.

7. Apply Filter Functions

7.1. Summation Filter Function

```
def apply_filter(filter_type):
    global processed_image
    if original_image is not None:
        if filter_type == "summation":
            kernel = np.ones((5, 5), np.float32) / 25
            filtered = cv2.filter2D(original_image, -1, kernel)
        ...
```

- **Functionality:** Applies a summation filter to the original image.
- **OpenCV Function:** `cv2.filter2D()`
- **Parameters:**
  - `original_image`: The original image to be filtered.
  - `filter_type`: The type of filter to apply ("summation" in this case).
- **Global Variables:** Updates the `processed_image` global variable with the filtered image and displays it on the right canvas.

7.2. Derivative Filter Function

```
def apply_filter(filter_type):
    ...
    elif filter_type == "derivative":
        kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
        filtered = cv2.filter2D(original_image, -1, kernel)
    ...
```

- **Functionality:** Applies a derivative filter to the original image.
- **OpenCV Function:** `cv2.filter2D()`
- **Parameters:**
  - `original_image`: The original image to be filtered.
  - `filter_type`: The type of filter to apply ("derivative" in this case).

- **Global Variables:** Updates the `processed_image` global variable with the filtered image and displays it on the right canvas.

Summary:

These functions provide various image processing capabilities:

- **Blur, Unblur:** Modify the appearance of the image by applying blurring or sharpening filters.
- **Transparency Meter:** Calculates the transparency level of the image and applies a binary mask.
- **Erode, Dilate:** Morphological operations to modify the shape and size of objects in the image.
- **Histogram:** Displays the histogram of pixel intensity values in the image.
- **Summation, Derivative Filters:** Apply custom filters to the image for specific effects or enhancements.

These functions leverage OpenCV's extensive capabilities for image processing, enabling a wide range of modifications and enhancements to images displayed in the GUI.