TEAM – Faster Than Mathews

Members And Contributions:

IMT2022001 – Siddharth Menon: Rotation and Sharpen Effects

IMT2022082 – Sathvik S Rao: Threading, gaussian blur,writing java objects to files

IMT2022511 – K V Nikhilesh: Inversion and logging and displaying the logs of all the effects

IMT2022507 – Vishruth Vijay: Dominant colour, hue saturation Effects
IMT2022116 – Mallikarjun C: Brightness, Contrast, filtering the logs based on date and time
IMT2022006 – Shreyas Arun: Grayscale effect, flip effect

Documentation Of the effects and features:

BRIGHTNESS

The brightness effect uniformly increases or decreases the intensity of all pixels in the image, making it appear brighter or darker. The effect operates by altering the pixel values across the entire image according to the specified brightness value.

EXPLANATION OF CODE

Function addValueToPixels takes a 2D vector pixels representing an image and a value of type float as inputs. It traverses through each pixel in the image and modifies its RGB components by adding the value after subtracting 100. This adjustment ensures the brightness alteration of the image. The function adjusts the resulting RGB values between 0 and 255 to maintain valid color representations.

CONTRAST

The contrast effect alters the difference between the light and dark areas, enhancing the overall visual sharpness of the image. When a value is used to

adjust contrast, it typically scales the difference between the darkest and lightest pixels in the image.

EXPLANATION OF CODE

Function multiplyValueToPixels changes the contrast of an image represented by a 2D vector pixels by adjusting the RGB values of each pixel. It calculates a factor based on the provided value, altering the contrast by scaling pixel values around a midpoint of 128. The function loops through each pixel, subtracts 128 to center the pixel range, applies the calculated factor to adjust contrast, and recenters the values around 128 before clamping them between 0 and 255 to ensure they remain within the valid color range.

FLIP EFFECT

The flip effect, when applied to an image, essentially mirrors or flips the content horizontally or vertically.

- Horizontal Flip: For a horizontal flip, every pixel in a row is rearranged in reverse order, swapping the leftmost pixel with the rightmost one, the second leftmost with the second rightmost, and so on, effectively flipping the image along a vertical axis.
- Vertical Flip: A vertical flip rearranges rows of pixels in reverse order, so the top row becomes the bottom row and so forth, flipping the image along a horizontal axis. These operations don't alter the content within the image.

EXPLANATION OF CODE

Function, flipImage, takes in an image represented by a 2D vector imageVector and two integer values, horizontalFlipValue and verticalFlipValue. If horizontalFlipValue equals 1, it iterates through each row in the image and reverses the order of pixels within each row using the reverse function from the standard library. Else If verticalFlipValue is 1, the function reverses the order of rows in the entire image vector, flipping the image vertically.

GAUSSIAN BLUR

The Gaussian blur effect smooths or blurs an image by reducing the high-frequency components through a convolution operation with a Gaussian filter kernel. This process follows a Gaussian distribution.

- For each pixel in the image, a weighted average is calculated from its neighboring pixels, where the weights correspond to a Gaussian distribution.
- The size of the neighborhood considered determines the extent of blurring: a larger kernel results in more significant blurring.
- This blurring process reduces sharp edges and details.

The kernel is made based on normal distribution.

EXPLANATION OF CODE

It begins by generating a Gaussian kernel based on the provided radius parameter and a calculated sigma value, which determines the extent of blurring. The sigma value calculated is based on opencv documentation. The image is then padded to handle edge cases during convolution. For each pixel in the image, a weighted average is calculated by convolving the pixel's neighborhood with the generated Gaussian kernel. The weighted sums for the red, green, and blue channels are computed separately, updating the pixel values in the original image accordingly. The process repeats for all pixels in the image. This results in a smoother appearance, reducing high-frequency components and softening edges in the image.

GRAYSCALE

The grayscale effect converts the colored image into a monochromatic representation, where each pixel is represented only by its brightness value. This transformation eliminates color information. Applying the grayscale effect removes color information from the image, transforming it into a black-and-white or grayscale representation based on the brightness of each pixel.

EXPLANATION OF CODE

This C++ code implements a simple grayscale conversion for an image represented by a 2D vector image. The function convertToGrayscale traverses

through each pixel in the image using nested loops. For each pixel, it calculates the average of its R, G, B values. This average represents the luminance or brightness level for that pixel in grayscale. Then, it sets the red, green, and blue components of the pixel to this calculated average value, effectively creating a grayscale effect by making all RGB channels equal, resulting in a monochromatic representation of the image where each pixel's color is replaced by its brightness level.

HUE SATURATION

The hue-saturation effect alters the appearance of an image by adjusting its colors while keeping the overall brightness intact. This adjustment involves modifying the hue (the type of color), saturation (the intensity or purity of the color), and sometimes the brightness of the image.

- Hue: Changing the hue shifts the entire color spectrum. For instance, shifting from red to blue or from yellow to green.
- Saturation: Adjusting saturation either intensifies or desaturates colors.

EXPLANATION OF CODE

This C++ code implements a function, applyHueSaturation, which adjusts the hue and saturation of an image represented by a 2D vector of pixels. It begins by iterating through each pixel in the image and converts its RGB values to the HSL (Hue, Saturation, Lightness) color space. The code calculates the hue angle and saturation factor based on the input parameters. It then adjusts the hue by adding the provided angle value and modifies the saturation by scaling it according to the saturation factor. Afterward, it converts the adjusted HSL values back to the RGB color space.

INVERT EFFECT

The invert effect, when applied to an image, transforms the colors by inverting each pixel's RGB values. This transformation is achieved by subtracting the current RGB values from the maximum possible value for each channel, which is 255.

EXPLANATION OF CODE

This C++ code defines a function called inversion that applies the invert effect to an image represented by a 2D vector of pixels. The function iterates through each pixel in the image using nested loops. For each pixel, it performs an invert operation by subtracting the current red (r), green (g), and blue (b) channel values from 255 (which represents the maximum intensity for an 8-bit color channel). This subtraction effectively reverses the color by producing the inverse of each channel's intensity, creating the inverted effect.

ROTATION EFFECT

The rotation effect modifies the orientation of an image by altering its angles, typically in increments of 90 degrees (clockwise or counterclockwise) or other specified angles. This transformation involves rearranging the pixels in the image grid to create a new arrangement based on the desired rotation angle.

EXPLANATION OF CODE

This C++ code defines a function named applyRotation that rotates an image represented by a 2D vector of pixels based on the parameter. The function employs different logic for each rotation angle: 90 degrees clockwise, 180 degrees, and 90 degrees counterclockwise. For a 90-degree clockwise rotation, the function creates a new vector rotatedImage with dimensions swapped, then copies the pixels from the original image to their new positions in the rotated image, accounting for the rotation transformation. Similarly, for a 180-degree rotation, the function creates a new image vector and copies pixels from the original image to the corresponding inverted positions in the rotated image. Lastly, for a 90- degree counterclockwise rotation, it transposes the original image by swapping rows and columns and then copies the pixels into the rotated image.

SEPIA EFFECT

The sepia effect transforms an image's appearance to give it a vintage, brownish tone. The process of applying the sepia effect involves converting the colors of each pixel in the image from the original RGB (Red, Green, Blue) color space to a sepia color scheme. This transformation involves:
1. Desaturation: Reducing the intensity of colors creating a faded appearance.

2. Applying Tones: Adjusting the color channels (typically red and green) to achieve the sepia effect. This might involve increasing the intensity of red while slightly reducing green and blue components, giving the image a brownish tint.

EXPLANATION OF CODE

This C++ code defines a function named applySepia that applies the sepia effect to an image represented by a 2D vector of pixels. Within nested loops iterating through each pixel in the image, the function calculates new red, green, and blue values based on weighted combinations of the original pixel's RGB components. These coefficients represent the transformation applied to each channel to get the sepia-toned look. To ensure the resulting values fall within the valid RGB range of 0 to 255, the function uses the min and max functions to clip the values, preventing them from exceeding the allowable range.

SHARPEN EFFECT

The sharpen effect enhances its perceived sharpness and detail by emphasizing edges and fine details within the image. This effect increases the contrast along edges. The sharpen effect essentially enhances the local contrast and details in an image, making edges more defined and prominent.

EXPLANATION OF CODE

Function named applySharpen that applies a sharpening effect to an image represented by a 2D vector of pixels. It uses a predefined 3x3 sharpening kernel with specific weights to highlight edges and details within the image. Nested loops are used to apply the sharpening kernel. Weighted sum of neighbouring pixels is calculated and then the calculated sumRed, sumGreen and sumBlue values using the calculated kernel weights are applied to the pixels.

DOMINANT COLOR

The dominant color is determined based on the frequency of hues and saturations present in the image. The resulting dominant hue and saturation are then used to convert all pixels in the image to the corresponding RGB values. The code uses

functions for hue-saturation-to-RGB conversion and auxiliary functions to calculate hue and saturation.

EXPLANATION OF CODE

The code identifies the dominant hue and saturation in the image and applies the dominant color to all pixels. The function applyDominantColor iterates through each pixel in the image, calculating its hue and saturation. It divides the hue and saturation values by scaling factor s and updates the frequency map. Determines the most frequent hue and saturation combination. Applies the dominant hue and saturation to all pixels and clamps the RGB values. The function hueSaturationtoRgb converts hue from degrees to the RGB color space using the HSL-to-RGB conversion formula. It adjusts the RGB values based on the calculated hue, saturation, and brightness. Converts the floating-point RGB values to integers and updates the references.

LOGGING:

A Logging service has been implemented in the project. In the 'Threading' directory in the project, there is an ExecuteLoggingService class that is instantiated in each of the effect implementation files. Logs are written into by this ExecuteLoggingService object, and the logs can be filtered by date and time and by the type of effect. Threading is used to implement logging.

THREADINGl

The purpose of using multithreading is to perform 2 tasks concurrently:

1) Photo service

2) Adding a log to the Log list based on the photo effect applied

The main thread runs the photo service code, and the ExecuteLoggingService Thread runs the code which adds the log into the log list and to the object file

which stores the log list. The implementation of the threading can be seen in any file in the effectImplementation folder. The 'Threading' folder contains the ExecuteLoggingService class which extends Thread.