UNIVERSITY
OF LONDON

CM2030 Graphics Programming
End-term
Commentary: Final Assignment [002]

TABLE OF CONTENTS

Thresholding in RGB, HSV, and CMYK

Thresholding in **RGB**, **HSV**, and **CMYK** color models is determined by a threshold value set through a slider.

RGB

rgb_thresholdFilter(img, channel, slider):

```
// Apply threshold to the specified channel
if (channel === 'red' && r > slider.value()) {
    r = 255;
} else if (channel === 'red') {
    r = 0;
}

if (channel === 'green' && g > slider.value()) {
    g = 255;
} else if (channel === 'green') {
    g = 0;
}

if (channel === 'blue' && b > slider.value()) {
    b = 255;
} else if (channel === 'blue') {
    b = 0;
}
```

Varying the threshold value affects how much of **each color channel** (red, green, or blue) is retained or turned off, with higher values preserving more intense colors and lower values eliminating them.

HSV

```
// Apply threshold to the Hue value
if (h > hsv_thresholdSlider.value() / 255 * 360) {
    // Keep the pixel as is (meets the threshold condition)
    imgOut.pixels[index + 0] = h * 255 / 360;   // Hue (0-360)
    imgOut.pixels[index + 1] = s * 255;         // Saturation (0-255)
    imgOut.pixels[index + 2] = v * 255;         // Value (0-255)
} else {
    // Set the pixel to black (does not meet the threshold condition)
    imgOut.pixels[index + 0] = 0;
    imgOut.pixels[index + 1] = 0;
    imgOut.pixels[index + 2] = 0;
}
```

Adjusting the threshold on the **hue** channel selectively isolates specific colors, and as the threshold changes, more colors are either retained or filtered out, affecting image clarity.

CMYK

```
// Apply threshold to black channel
if (black > cmyk_thresholdSlider.value() / 255)
{
    // Keep the pixel as is
    imgOut.pixels[index + 0] = cyan * 255;
    imgOut.pixels[index + 1] = magenta * 255;
    imgOut.pixels[index + 2] = yellow * 255;
    imgOut.pixels[index + 3] = black * 255;
}
else
{
    // Set the pixel to white
    imgOut.pixels[index + 0] = 255;
    imgOut.pixels[index + 1] = 255;
    imgOut.pixels[index + 2] = 255;
    imgOut.pixels[index + 3] = 255;
}
```

Adjusting the threshold value on the **black** channel emphasizes darker regions, where increasing the threshold highlights more black areas.

One key finding is that threshold values significantly impact which features are visible, often highlighting specific colors or areas but potentially eliminating subtle variations in the image.

Problems Encountered

Blurry Text

One issue faced was blurry text when drawing it in the `draw()` function. This occurred because the text was continuously redrawn every frame, leading to pixel blending and reduced clarity.

Solution:
To use `createGraphics()` to render static text once, storing it in a separate graphics buffer (`graphics_before_snapshot` and `graphics_after_snapshot`). These buffers were then drawn using `image()`, preventing unnecessary redrawing. This approach significantly improved text sharpness while maintaining performance. Additionally, it allowed for clear separation between pre- and post-processing instructions, enhancing user readability and interface clarity in the image processing application.

Functions to be called in setup() of sketch.js:

```
function setup_graphicsBefore()
{
    graphics_before_snapshot = createGraphics(width, height);
    graphics_before_snapshot.fill(255);
    graphics_before_snapshot.textAlign(CENTER);
    graphics_before_snapshot.textSize(20);
    graphics_before_snapshot.text("IMAGE PROCESSING APPLICATION", width/2, 20);
    graphics_before_snapshot.textSize(12);
    graphics_before_snapshot.text("Press `SPACEBAR`: take snapshot", width/2, 40);
}
```

```
function setup_graphicsAfter()
{
    graphics_after_snapshot = createGraphics(width, height);
    graphics_after_snapshot.fill(255);
    graphics_after_snapshot.textAlign(CENTER);

    graphics_after_snapshot.textSize(20);
    graphics_after_snapshot.text("IMAGE PROCESSED", width/2, 20);

    graphics_after_snapshot.textAlign(LEFT);
    graphics_after_snapshot.textSize(12);
    graphics_after_snapshot.text("Press `R`: refresh", width*3/4, 15);
    graphics_after_snapshot.text("Press `1`: greyscale face", width*3/4, 30);
    graphics_after_snapshot.text("Press `2`: blur face", width*3/4, 45);
    graphics_after_snapshot.text("Press `3`: hsv face", width*3/4, 60);
    graphics_after_snapshot.text("Press `4`: pixelate face", width*3/4, 75);
    graphics_after_snapshot.text("Click image to save", 0, 15);

    graphics_after_snapshot.text("Your image:", col1, row1-5);
    graphics_after_snapshot.text("Greyscale (20% brighter):", col2, row1-5);
    graphics_after_snapshot.text("Red channel:", col1, row2-5);
    graphics_after_snapshot.text("Green channel:", col2, row2-5);
    graphics_after_snapshot.text("Blue channel:", col3, row2-5);
    graphics_after_snapshot.text("Red threshold:", col1, row3-5);
    graphics_after_snapshot.text("Green threshold:", col2, row3-5);
    graphics_after_snapshot.text("Blue threshold:", col3, row3-5);
    graphics_after_snapshot.text("Your image:", col1, row4-5);
    graphics_after_snapshot.text("RGB-HSV:", col2, row4-5);
    graphics_after_snapshot.text("RGB-CMYK:", col3, row4-5);
    graphics_after_snapshot.text("Face detection:", col1, row5-5);
    graphics_after_snapshot.text("HSV threshold:", col2, row5-5);
    graphics_after_snapshot.text("CMYK threshold:", col3, row5-5);
    graphics_after_snapshot.text("Invert & colour shift:", col1, row6-5);
    graphics_after_snapshot.text("Symmetry:", col2, row6-5);
    graphics_after_snapshot.text("Wave distort:", col3, row6-5);
}
```

In draw():

```
if (!snapshot)
{
    image(graphics_before_snapshot, 0, 0); // Texts
    image(capture, width/2-200, space, 400, 300);
}
else
{
    image(graphics_after_snapshot, 0, 0); // Texts
```

Unable to Apply Filters on `detectedFace`

The issue arose when trying to apply filters to `detectedFace` extracted from `snapshot.get()`. The face region was successfully drawn using `image()`, but applying a filter function directly to` detectedFace` failed.

Solution:

Instead of passing `detectedFace` directly to the filter function, the code was modified to use `**faceReplacement**`, ensuring compatibility with the filtering logic. The updated approach correctly processes the detected face and applies the selected filter before rendering it. This ensures proper face filtering while maintaining accurate positioning and display in the image processing application.

```
function preload() {
    faceReplacement = loadImage('meme.png');
}
```

```
function drawFaces(option)
{
    const filters = {
        1: greyscaleFilter,
        2: blurFilter,
        3: rgb_to_hsv,
        4: pixelateFilter
    };

    for (let face of faces) {
        if (face[4] > 4) { // Confidence threshold

            // Extract the detected face region
            var detectedFace = snapshot.get(face[0], face[1], face[2], face[3]);

            image(detectedFace, width*3/4, 2*space, 50, 50);  // WORKED WELL, CAN BE DRAWN
            textAlign(LEFT);
            text("Face detected:", width*3/4, 2*space);

            const filter = filters[option];
            if (!filter) continue;

            // const filteredFace = filter(detectedFace); // BUT FAILED HERE
            const filteredFace = filter(faceReplacement);
            image(filteredFace, face[0] + col1, face[1] + row5, face[2], face[3]);
        }
    }
}
```

In sketch.js's draw():

```
image(snapshot, col1, row5, w, h);
drawFaces(faceFilterOption);
```

Project Progress and Solutions to Challenges

This project is almost completed as I expected, but I faced an issue when trying to apply the filter on `detectedFace`. As the filter failed to work directly on the extracted face, I resorted to using another image, `faceReplacement`, to replace `detectedFace` for the filtering process.

This workaround helped move the project forward, but in the future, I would debug the filter application more thoroughly to ensure it works directly on dynamic regions like `detectedFace`.

Extensions

Filters

The filters extension introduces unique image manipulation effects that enhance the user's creative control. The **invertAndColorShiftFilter** inverts colors and randomly shifts hues, while the **symmetryFilter** mirrors the left side of an image to the right, creating a symmetrical effect. The **waveDistortionFilter** distorts images with sine wave patterns, providing a dynamic, wavy appearance. These filters give users the ability to apply artistic transformations to their images, making it a fun and engaging feature for image editing.

```
function symmetryFilter(img)
{
    var imgOut = createImage(img.width, img.height);
    imgOut.loadPixels();
    img.loadPixels();

    let midX = img.width / 2;

    // Loop through the left half of the image
    for (let y = 0; y < img.height; y++) {
        for (let x = 0; x < midX; x++) {
            let col = img.get(x, y);  // Get pixel color at (x, y)
            imgOut.set(x, y, col); // Copy pixel to the left side of the output image
            imgOut.set(img.width - x - 1, y, col); // Reflect pixel to the right side
        }
    }

    imgOut.updatePixels();
    return imgOut;
}
```

Save Processed Images

This extension allows users to save processed images with a simple mouse click, providing a seamless way to capture their work. Upon applying various filters , users can save the altered versions by clicking on the corresponding images displayed on the screen. The feature adds significant value by enabling users to store their processed images for future use or sharing, making the application more practical and interactive for creative exploration and image management.

```
function mousePressed()
{
    // Check if a snapshot exists
    if (!snapshot) return;

    // Define image positions and save if clicked
    checkAndSaveImage(snapshot, col1, row1, w, h, "original_image.png");
    checkAndSaveImage(greyscaleFilter(snapshot), col2, row1, w, h, "greyscale_image.png");

    checkAndSaveImage(rgbFilter(snapshot, 1, 0.5, 0.5), col1, row2, w, h, "red_tint_image.png");
    checkAndSaveImage(rgbFilter(snapshot, 0.5, 1, 0.5), col2, row2, w, h, "green_tint_image.png");
    checkAndSaveImage(rgbFilter(snapshot, 0.5, 0.5, 1), col3, row2, w, h, "blue_tint_image.png");

    checkAndSaveImage(rgb_thresholdFilter(snapshot, 'red', red_thresholdSlider), col1, row3, w, h, "red_threshold_image.png");
    checkAndSaveImage(rgb_thresholdFilter(snapshot, 'green', green_thresholdSlider), col2, row3, w, h, "green_threshold_image.png");
    checkAndSaveImage(rgb_thresholdFilter(snapshot, 'blue', blue_thresholdSlider), col3, row3, w, h, "blue_threshold_image.png");

    checkAndSaveImage(snapshot, col1, row4, w, h, "original2_image.png");
    checkAndSaveImage(rgb_to_hsv(snapshot), col2, row4, w, h, "hsv_image.png");
    checkAndSaveImage(rgb_to_cmyk(snapshot), col3, row4, w, h, "cmyk_image.png");

    checkAndSaveImage(snapshot, col1, row5, w, h, "original3_image.png");
    checkAndSaveImage(hsv_thresholdFilter(rgb_to_hsv(snapshot)), col2, row5, w, h, "hsv_threshold_image.png");
    checkAndSaveImage(cmyk_thresholdFilter(rgb_to_cmyk(snapshot)), col3, row5, w, h, "cmyk_threshold_image.png");

    checkAndSaveImage(invertAndColorShiftFilter(snapshot), col1, row6, w, h, "inverted_colourShift_image.png");
    checkAndSaveImage(symmetryFilter(snapshot), col2, row6, w, h, "symmetry_image.png");
    checkAndSaveImage(waveDistortionFilter(snapshot), col3, row6, w, h, "wave_distortion_image.png");
}

// Function to check if the mouse click is inside an image and save it
function checkAndSaveImage(img, x, y, w, h, filename) {
    if (mouseX > x && mouseX < x + w && mouseY > y && mouseY < y + h) {
        save(img, filename);
    }
}
```