Data acquisition:

A hyperspectral image database of flower petals was built using the SpecimIQ. Each captured image has a size of 512 x 512 x 204 and their information are stored in “hdr” and “raw” files.

Data preprocessing and precomputation in Python:

Firstly, the hyperspectral image was white corrected. Thereafter, to reduce the huge amount of data loaded during spectral image visualization, the spectral dimension of the hyperspectral image was reduced using a dimensionality reduction technique known as Principal Component Analysis (PCA). Reducing the data size increases the loading time of the visualization tool and reduces the amount of memory needed to store or render the spectral image.

PCA was applied on the white balanced spectral image by computing the eigen vectors and values of the spectral image using its covariance matrix. The eigen vectors were then sorted by the eigenvalues in descending order, and the projection matrix was computed by performing a matrix multiplication operation on the eigenvector and the spectral image data. The first 6-components that retained about 99% of the variance were selected with their respective eigen vectors. Also, the average of the spectral image was computed to minimize the average reconstruction error during the reconstruction process.

The components, eigenvector, and average vector are negative and/or positive float values of size (6 x 512 x 512), (6 x 204), and (1 x 204) respectively.

Since we are required to resample the reflectance spectra of the spectral image, the eigen vector data for each component and the average vector were interpolated resulting to an eigen vector of size (6 x 600), and that of the average vector, (1 x 600).

For color reconstruction, XYZ values were precomputed for different virtual lights by first multiplying the reflectance spectra of each illuminant, L by their corresponding CIE color matching function, then normalizing the result so Y = 1. In our case, we have 16 illuminants, each illuminant and CIExyz has a wavelength range that covers 400nm to 781nm, with 1nm resolution. Computing XYZ for the illuminants results in a float value data of size (16 x 3).

The computed components, eigen vectors, average vector, and illuminants XYZ data were then stored in a JSON file for the visualization stage.

Spectral and color reconstruction using shader:

The precomputed components in the JSON file were loaded and stored as textures in a 2D texture array. The eigen vectors, average vector, and illuminants XYZ data were also loaded, and essential variables were computed from the sizes of these data.

For spectral reconstruction, depending on the band selected by the user (between 400 to 999), the following data are passed to the fragment shader: 2D texture array of 6 components, eigen vector of selected band (or wavelength) with size (6 x 1), and average value of selected band which is a float value.

In the fragment shader, the color of the fragment to be rendered is reconstructed using the given formula:

$$Fragment_{color} = \left(\sum_{c=0}^{c=d-1} Components(i,j,c) * eigenvector[c] \right) - average$$

Where (i, j) is the position of the fragment and 'Components' is a 2D texture array that contains the PCA components.

For color reconstruction, each fragment's tri-stimulus values are computed when a virtual light is illuminated. For this operation, the user selects a virtual light, and its precomputed XYZ values are passed to the fragment shader. Thereafter, the user selects three wavelengths which corresponds to R, G, and B channels. Similar to spectral reconstruction, these wavelengths' eigen vectors, average values, and 2D texture array (containing six components) are passed to the fragment shader.

The fragment's tri-stimulus values, XYZ are computed by reconstructing the fragment's spectral reflectances for the three selected wavelengths which corresponds to R, G, and B channels. Thereafter, the reconstructed spectral reflectances is multiplied by the illuminant's XYZ values as shown in the formula below.

$$X_{Fragment} = \left(\sum_{c=0}^{c=d-1} Components(i,j,c) * eigenvector_{\lambda=red}[c] * X_{illuminant} \right) - average$$

$$Y_{Fragment} = \left(\sum_{c=0}^{c=d-1} Components(i,j,c) * eigenvector_{\lambda=green}[c] * Y_{illuminant} \right) - average$$

$$Z_{Fragment} = \left(\sum_{c=0}^{c=d-1} Components(i,j,c) * eigenvector_{\lambda=blue}[c] * Z_{illuminant} \right) - average$$

The computed CIEXYZ tri-stimulus values of the fragment are then converted to Linear RGB or sRGB using a transformation matrix.

Problems I experienced while implementing the methods:

Most of the problems I encountered while working on this practical work were related to data manipulation and less of technical issues. These avoidable data manipulation issues or mistakes include not using the correct data type (float) while loading the data on the client side, not fetching the correct data in the fragment shader, normalizing the PCA components values, etc.

In my implementation, subtracting the average value from the reconstructed data results in loss of information for some reason. To prevent this, I added an offset value to the average value. This value can be changed dynamically from the GUI.