DLIP 중간고사 공부

Histogram

Histogram Equalization

Histogram Equalization는 이미지의 명암 대비를 개선하기 위해 사용되는 기법입니다. 이 방법은 이미지의 Histogram을 재분포하여 전체 명암 구간에 걸쳐 균일하게 만듭니다. 수학적으로, Histogram Equalization는 누적 분포 함수(Cumulative Distribution Function, CDF)를 이용합니다. 아래는 Histogram Equalization의 mathematical equation입니다.

y = T(x) = (L - 1) * CDF(x)

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

$$p_r(r_j) = \frac{n_j}{MN}$$

여기서 x는 입력 픽셀 값, y는 출력 픽셀 값, L은 출력 이미지의 명암 구간의 크기, CDF(x)는 입력 픽셀 값 x의 누적 분포 함수입니다.

example_code_Histogram_Equalization.cpp

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace cv;

int main() {
    // 이미지 로드
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);

    // 결과 이미지를 저장할 Mat 객체 생성
    Mat equalizedImage;

    // Histogram Equalization 적용
    equalizeHist(image, equalizedImage);

    // 결과 이미지 출력
    imshow("Original Image", image);
    imshow("Equalized Image", equalizedImage);
    waitKey(0);

    return 0;
}
```

Thresholding

Global Binary Thresholding

Global Binary Thresholding은 이미지에서 경계 값을 기준으로 픽셀 값을 이진화하는 기법입니다. 이 기법은 모든 픽셀에 대해 동일한 경계 값을 사용하며, 주어진 경계 값보다 큰 픽셀은 하나의 값(예: 255)으로, 작거나 같은 픽셀은 다른 값(예: 0)으로 설정합니다. 수식은 다음과 같습니다.

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T \\ 0, & \text{if } f(x,y) \le T \end{cases}$$

여기서 x,y는 픽셀 좌표, f(x,y)는 입력 픽셀 값, g(x,y)는 출력 픽셀 값, T는 threshold value(경계 값)입니다.

Pseudo code for Global Binary Thresholding:

An iterative algorithm of finding T

1. Analyze the intensity histogram and select the initial estimation of T (usually the mean of the image intensity). Let the intensity of the input image is defined as g(x,y).
2. Segment the image by two groups on the histogram using the value of T

$$G_1 \text{ if } g(x,y) > T$$
$$G_2 \text{ if } g(x,y) \le T$$

3. Find the mean of G_1 and G_2 (i.e. m_1 and m_2)
4. The new T value at k^{th} iteration

$$T_k = \frac{1}{2}(m_1 + m_2)$$

5. repeat from step 2 until $\Delta T < \sigma$, where $\Delta T < T_k - T_{k-1}$

```
An iterative algorithm of finding T
1. 이미지의 intensity histogram을 분석하고 T(임계값)의 초기 추정치를 선택합니다(보통 이미지 intensity의 평균값).
2. T 값으로 이미지를 두 그룹으로 분할합니다. 이 때, G1은 최대값이고, G2는 최소값입니다.
3. G1과 G2의 평균값을 찾습니다. 이 값을 각각 m1과 m2라고 부릅니다.
4. k번째 반복에서의 새로운 T 값
5. Tk = 1/2*(m1, m2)
6. (\Delta)T < \singma 인 경우까지 2단계부터 반복합니다. 여기서, \Delta T는 T_k - T_(k-1)입니다.
```

example_code_Global_Binary_Thresholding.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    // 이미지 로드
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);

    // 결과 이미지를 저장할 Mat 객체 생성
    Mat thresholded;

    // Global Binary Thresholding 적용
    int threshold = 128;
    int maxValue = 255;
    int minValue = 0;
    threshold(image, thresholded, threshold, maxValue, THRESH_BINARY);

    // 결과 이미지 출력
    imshow("Original Image", image);
    imshow("Thresholded Image", thresholded);
    waitKey(0);

    return 0;
}
```

Morphology

Dilation

Dilation은 이웃에 있는 모든 픽셀 중 최댓값이 출력 픽셀의 값이 되는 기법이다. 이진 영상의 경우, 값이 1인 이웃 픽셀이 하나라도 있으면 픽셀은 1로 설정된다. Dilation은 객체를 시각적으로 더 두드러지게 만들고 객체의 작은 구멍을 메운다. 선은 더 두껍게, 채워진 형태는 더 크게 표시된다.

example_code_Dilation.cpp

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    // 이미지 로드
    Mat image = imread("example.jpg");

    // 구조 요소 설정 (이 경우 5x5 크기의 사각형)
    Mat element = getStructuringElement(MORPH_RECT, Size(5, 5));

    // dilation 적용
    Mat dilated;
    dilate(image, dilated, element);

    // 결과 이미지 출력
    imshow("Dilated Image", dilated);
    waitKey(0);

    return 0;
}
```

Erosion

Erosion은 이웃에 있는 모든 픽셀 중 최솟값이 출력 픽셀의 값이 되는 기법이다. 이진 영상의 경우, 값이 0인 이웃 픽셀이 하나라도 있으면 픽셀은 0으로 설정된다. Erosion은 객체를 시각적으로 덜 두드러지게 만들고 객체의 경계를 깍아낸다. 선은 더 얇게, 채워진 형태는 더 작게 표시된다.

example_code_Erosion.cpp

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    // 이미지 로드
    Mat image = imread("example.jpg");

    // 구조 요소 설정 (이 경우 5x5 크기의 사각형)
    Mat element = getStructuringElement(MORPH_RECT, Size(5, 5));

    // erosion 적용
    Mat eroded;
```

```
    erode(image, eroded, element);

    // 결과 이미지 출력
    imshow("Eroded Image", eroded);
    waitKey(0);

    return 0;
}
```

Opening & Closing

Opening: Erosion -> Dilation

- 작은 객체나 노이즈 제거
- 객체 분리

Closing: Dilation -> Erosion

- 객체 내부의 작은 구멍 메우기
- 객체 완전한 형태 만들기

example_code_Opening_Closing.cpp

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    // 이미지 로드
    Mat image = imread("example.jpg");

    // 구조 요소 설정 (이 경우 5x5 크기의 사각형)
    Mat element = getStructuringElement(MORPH_RECT, Size(5, 5));

    // opening 적용 (erosion -> dilation)
    Mat opened;
    morphologyEx(image, opened, MORPH_OPEN, element);

    // 결과 이미지 출력
    imshow("Opened Image", opened);
    waitKey(0);

    // closing 적용 (dilation -> erosion)
    Mat closed;
    morphologyEx(image, closed, MORPH_CLOSE, element);

    // 결과 이미지 출력
    imshow("Closed Image", closed);
    waitKey(0);

    return 0;
}
```

Filtering

2D Convolution

2D Convolution은 이미지에 필터(커널)를 적용하여 새로운 이미지를 생성하는 과정이다. 수식은 다음과 같다.

$$\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x-s,y-t)$$

example_code_Convolution.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);
    Mat output;

    // 커널 정의
    Mat kernel = (Mat_<float>(3, 3) << -1, -1, -1,
                                       -1,  8, -1,
                                       -1, -1, -1);

    // Convolution 적용
    filter2D(image, output, -1, kernel);

    imshow("Original Image", image);
    imshow("Convolved Image", output);
    waitKey(0);

    return 0;
}
```

Correlation

Correlation은 이미지와 필터(커널)간의 유사도를 계산하는 과정이다. 수식은 다음과 같다.

$$\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)$$

example_code_Correlation.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);
    Mat output;

    // Define kernel
```

```
    Mat kernel = (Mat_<float>(3, 3) << -1, -1, -1,
                                       -1,  8, -1,
                                       -1, -1, -1);

    // Flip the kernel horizontally and vertically for correlation
    Mat flippedKernel;
    flip(kernel, flippedKernel, -1);

    // Apply correlation
    filter2D(image, output, -1, flippedKernel, Point(-1, -1), 0,
BORDER_DEFAULT);

    imshow("Original Image", image);
    imshow("Correlated Image", output);
    waitKey(0);

    return 0;
}
```

Normalized 2D Spatial Filtering Equation

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t) f(x+s, y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

example_code_Normalized_2D_Spatial_Filtering_Equation.cpp

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);
    Mat output;

    // Define kernel
    Mat kernel = (Mat_<float>(3, 3) << 1, 1, 1,
                                       1, 1, 1,
                                       1, 1, 1);

    // Normalize kernel
    kernel = kernel / 9.0;

    // Apply normalized 2D spatial filtering
    filter2D(image, output, -1, kernel, Point(-1, -1), 0, BORDER_DEFAULT);

    imshow("Original Image", image);
    imshow("Filtered Image", output);
    waitKey(0);

    return 0;
}
```

Padding

Padding은 이미지 처리 작업 중 주변 픽셀이 필요한 경우, 이미지 경계를 확장하여 처리하는 기법이다. 주로 컨볼루션 및 필터링 작업에서 사용된다. 이미지 경계에서 발생하는 이슈를 해결하기 위해 패딩을 사용한다.

패딩을 사용하지 않으면 컨볼루션 작업을 수행할 때 원본 이미지의 경계 부분에 있는 픽셀들은 이웃 픽셀이 충분하지 않아 올바르게 처리되지 않는다. 이러한 문제를 "padded edges issue" 라고 부른다. 패딩을 적용하면 이 문제를 해결할 수 있다.

OpenCV에서는 다양한 패딩 유형을 제공한다:

1. BORDER_CONSTANT: 테두리에 일정한 값으로 채운다.
2. BORDER_REPLICATE: 원본 이미지 경계의 가장자리 픽셀 값을 사용하여 패딩을 채운다.
3. BORDER_REFLECT: 원본 이미지 경계를 기준으로 반사된 값으로 패딩한다. (ex: gfd|abc|cde)
4. BORDER_WRAP: 원본 이미지의 반대쪽 가장자리 값을 가져와 패딩한다. (ex: cde|abc|abc)

적절한 패딩 유형을 선택함으로써 컨볼루션 및 필터링 작업에서 발생하는 경계 문제를 해결할 수 있다.

example_code_Padding.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg");
    Mat padded_image;

    // 이미지에 패딩 추가
    copyMakeBorder(image, padded_image, 5, 5, 5, 5, BORDER_CONSTANT,
Scalar(0));

    imshow("Original Image", image);
    imshow("Padded Image", padded_image);
    waitKey(0);

    return 0;
}
```

Smoothing

Box	Gaussian
$\frac{1}{K^2}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & 1 & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$

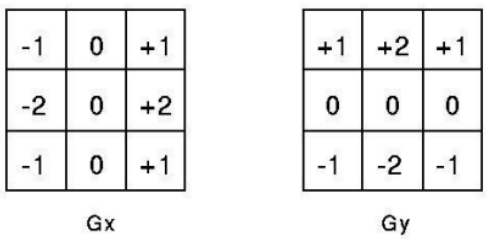
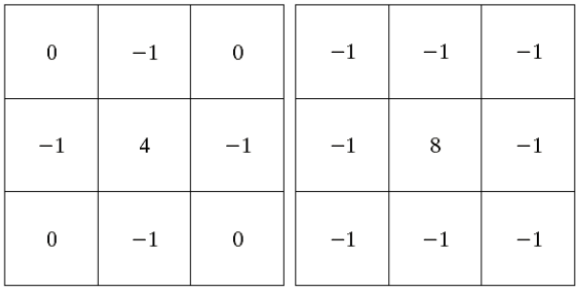
example_code_Box.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg");
    Mat box_filtered;

    // Box Filter 적용
    blur(image, box_filtered, Size(5, 5));

    imshow("Original Image", image);
    imshow("Box Filtered Image", box_filtered);
    waitKey(0);

    return 0;
}
```

example_code_Gaussian.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg");
    Mat gaussian_filtered;

    // Gaussian Filter 적용
    GaussianBlur(image, gaussian_filtered, Size(5, 5), 0);

    imshow("Original Image", image);
    imshow("Gaussian Filtered Image", gaussian_filtered);
    waitKey(0);

    return 0;
}
```

Sharpening

Sobel	Laplacian
	Laplacian without/with diagonal term 

example_code_Sobel.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);
    Mat sobel_x, sobel_y;

    // Sobel 필터 적용
    Sobel(image, sobel_x, CV_32F, 1, 0);
    Sobel(image, sobel_y, CV_32F, 0, 1);

    imshow("Original Image", image);
    imshow("Sobel X", sobel_x);
    imshow("Sobel Y", sobel_y);
    waitKey(0);

    return 0;
}
```

example_code_Laplacian.cpp

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;

int main() {
    Mat image = imread("example.jpg", IMREAD_GRAYSCALE);
    Mat laplacian_filtered;

    // Laplacian 필터 적용
    Laplacian(image, laplacian_filtered, CV_32F);

    imshow("Original Image", image);
    imshow("Laplacian Filtered Image", laplacian_filtered);
    waitKey(0);

    return 0;
}
```

Scale problem:

To sharpen an image, add Laplacian image to the original image

$$g(x,y) = f(x,y) \pm c[\nabla^2 f(x,y)]$$

example_code_Scale_problem.cpp

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main() {
```

```
    // 원본 이미지 로드
    String image_path = "path/to/your/image.jpg";
    Mat image = imread(image_path, IMREAD_COLOR);

    if (image.empty()) {
        cout << "이미지를 읽을 수 없습니다: " << image_path << endl;
        return -1;
    }

    // 이미지를 그레이스케일로 변환
    Mat gray_image;
    cvtColor(image, gray_image, COLOR_BGR2GRAY);

    // 그레이스케일 이미지에 Laplacian 필터 적용
    Mat laplacian;
    Laplacian(gray_image, laplacian, CV_64F);

    // Laplacian 이미지 정규화
    Mat normalized_laplacian;
    normalize(laplacian, normalized_laplacian, 0, 255, NORM_MINMAX, CV_8U);

    // Laplacian 이미지를 원본 그레이스케일 이미지에 추가하여 선명한 이미지 생성
    Mat sharpened_image;
    add(gray_image, normalized_laplacian, sharpened_image);

    // 원본 및 선명한 이미지 표시
    imshow("원본 이미지", gray_image);
    imshow("선명한 이미지", sharpened_image);

    waitKey(0);
    return 0;
}
```

Camera Calibration

Camera model

$$ {}^{im}\mathbf{p} = \mathbf{K}\begin{bmatrix}\mathbf{R} & | \mathbf{T}\end{bmatrix} {}^{O}\mathbf{P} $$

$$ \underbrace{{}^{C}Z \begin{bmatrix} {}^{im}x \\ {}^{im}u \\ 1 \end{bmatrix}}_{\{Im\}} = \underbrace{\begin{bmatrix} -f/s_x & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}}_{M_{int}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}}_{M_{ext}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{\{O\}} $$

R T

scale

$p_{im} = KP_c$ $\quad$ $X_C = [R \mid T]\, X_O$ $\quad$ {C}

- 여기서, 초점 길이(f_x, f_y), 광학 중심(c_x, c_y),

- K는 카메라의 내부 속성을 나타냅니다. 이 행렬은 주로 렌즈 왜곡, 초점 거리 및 이미지 센서와 관련된 매개 변수를 포함합니다.
- R은 카메라의 방향을 나타냅니다.
- t는 월드 좌표계의 원점을 카메라 좌표계로 변환하는 데 사용되는 벡터입니다. 이것은 카메라의 위치를 나타냅니다.

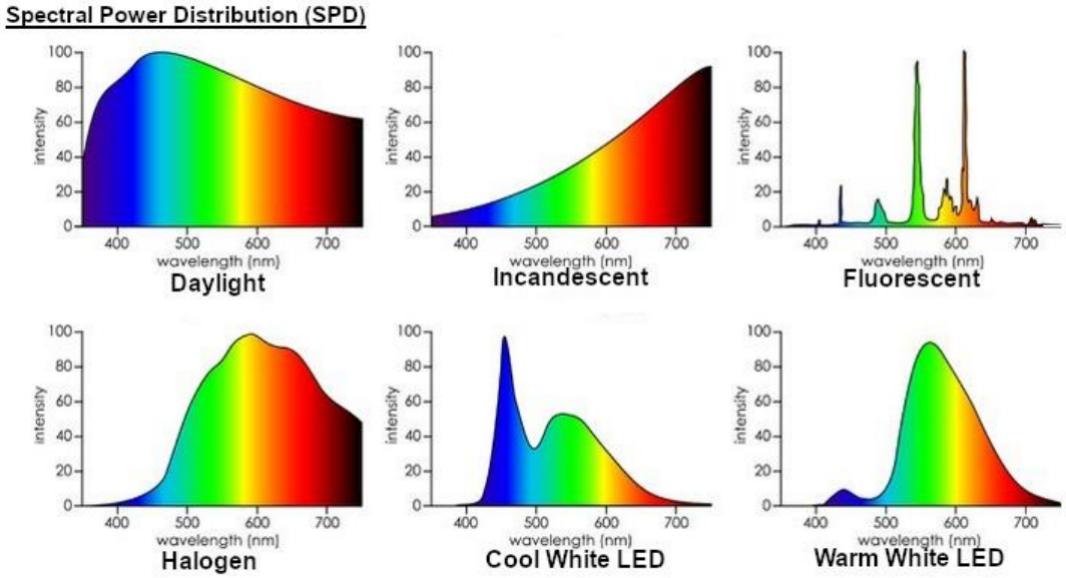
Color Image Processing

Color Spectrum

- Radiance: 특정 방향에서 빛의 강도를 나타내는 물리적 속성이다.
- Luminance: 빛의 밝기를 나타내며, 사람의 눈이 빛의 강도를 인식하는 방식을 반영한다.
- Brightness (Achromatic): 색이 없는 빛의 밝기를 나타낸다.

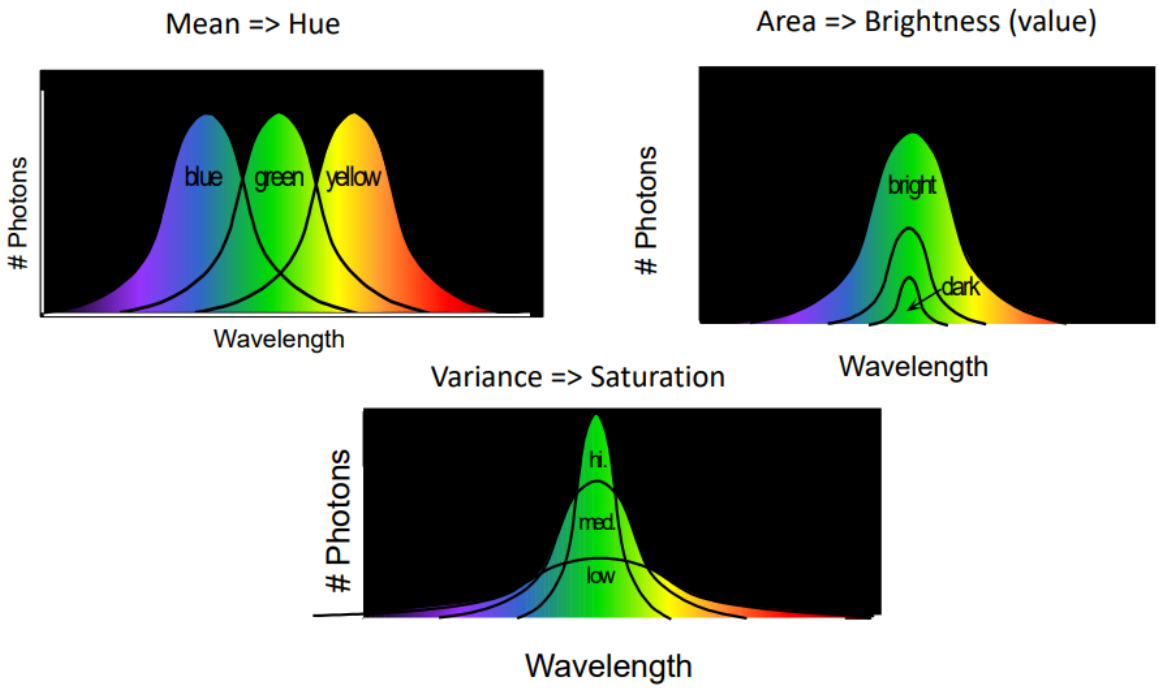
Spectral Power Distribution (SPD)

SPD는 빛의 파장별 강도를 나타낸다. 일반적으로 파장에 따른 상대적인 강도를 그래프로 표시한다.



Color Model

HSV (Hue, Saturation, Value): 색상, 채도, 명도로 구성된 색 공간이다. 색상은 색의 종류를, 채도는 색의 선명함을, 명도는 색의 밝기를 나타낸다.



HSV 색 공간은 Hue(색상), Saturation(채도), Value(명도)로 구성된 색 공간입니다. RGB 색 공간과 달리 색상, 채도, 밝기를 각각 별도의 채널로 표현하기 때문에 인간이 인식하는 색상 특성과 비슷한 방식으로 색을 표현합니다.

1. Hue (색상): 색상은 색의 종류를 나타내며, 일반적으로 원형으로 표시됩니다. Hue 값은 각도로 표현되며, 0도에서 360도 사이의 값으로 색상을 나타냅니다. 예를 들어, 빨간색은 0도, 녹색은 120도, 파란색은 240도로 표시됩니다.
2. Saturation (채도): 채도는 색의 선명함을 나타내며, 0에서 1 사이의 값으로 표현됩니다. 채도가 0이면 색이 회색으로 변하고, 채도가 1이면 가장 선명한 색상을 가집니다.
3. Value (명도): 명도는 색의 밝기를 나타내며, 0에서 1 사이의 값으로 표현됩니다. 명도가 0이면 검은색이 되고, 명도가 1이면 가장 밝은 색상이 됩니다.

RGB 색 공간과 비교할 때, HSV 색 공간은 색상을 조정하거나 추출하는 작업에 적합합니다. 예를 들어, 특정 색상의 객체를 추적하거나 이미지에서 색상을 분리할 때, HSV 색 공간을 사용하면 작업이 더 간단해집니다.

Tri-stimulus values

SPD를 기반으로 하는 RGB 색 공간의 값을 나타낸다. 여기서 R, G, B는 빨강, 초록, 파랑을 각각 나타낸다.

- Luminance of RGB
  - 인간의 눈은 각 색의 루미넌스를 다르게 인식한다. 일반적으로 G > R > B 순이다.
    - Simple luminance: Y=(R+G+B)/3
    - Consider Human eyes luminance: Y=0.2125R+0.7154G+0.0721B

InRange

InRange 함수는 주어진 범위 내에 있는 픽셀을 이진 이미지로 변환하는 데 사용된다. 이진 이미지에서 범위 내의 픽셀은 255(흰색)로, 범위 밖의 픽셀은 0(검은색)으로 설정된다.

```
```