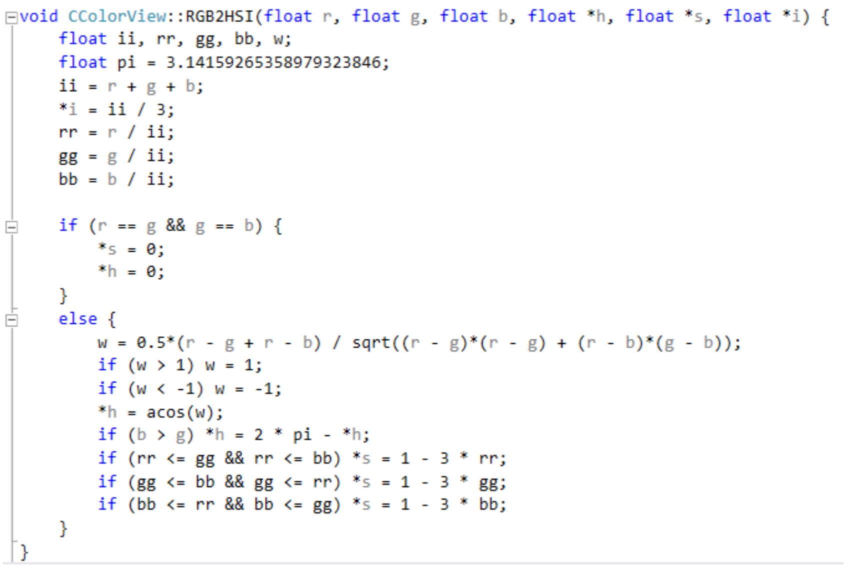
Programming Assignment No.4

Due Date: May 8th, 2018

Q1. Implement the color image saturation modification (75%, 50%, 25%, 0%) in HSI color space.

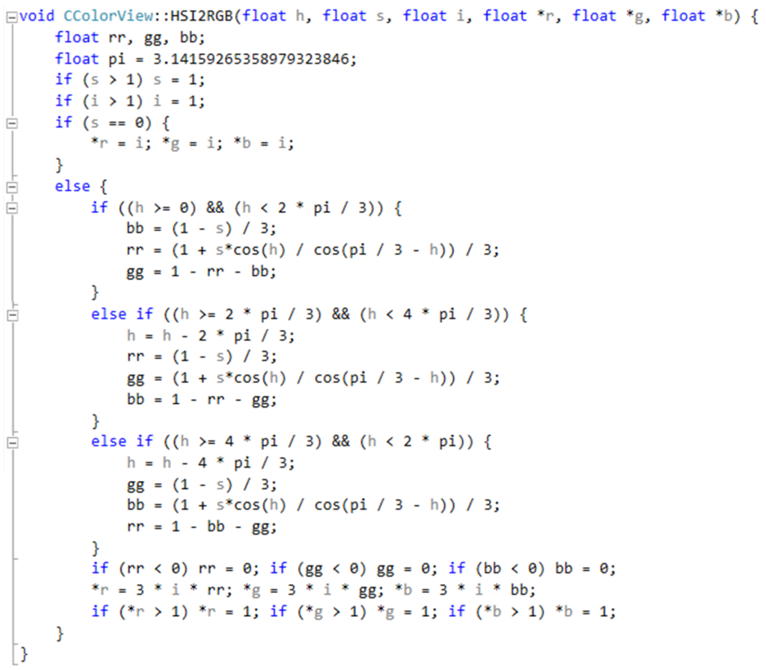
In order to modify saturation value, we need to first transform color space from RGB to HSI. Then we modify the saturation (S) value and re-transform it back to RGB color space.

The C++ code to transform color space between HSI and RGB is as following.



```cpp
void CColorView::RGB2HSI(float r, float g, float b, float *h, float *s, float *i) {
    float ii, rr, gg, bb, w;
    float pi = 3.14159265358979323846;
    ii = r + g + b;
    *i = ii / 3;
    rr = r / ii;
    gg = g / ii;
    bb = b / ii;

    if (r == g && g == b) {
        *s = 0;
        *h = 0;
    }
    else {
        w = 0.5*(r - g + r - b) / sqrt((r - g)*(r - g) + (r - b)*(g - b));
        if (w > 1) w = 1;
        if (w < -1) w = -1;
        *h = acos(w);
        if (b > g) *h = 2 * pi - *h;
        if (rr <= gg && rr <= bb) *s = 1 - 3 * rr;
        if (gg <= bb && gg <= rr) *s = 1 - 3 * gg;
        if (bb <= rr && bb <= gg) *s = 1 - 3 * bb;
    }
}
```

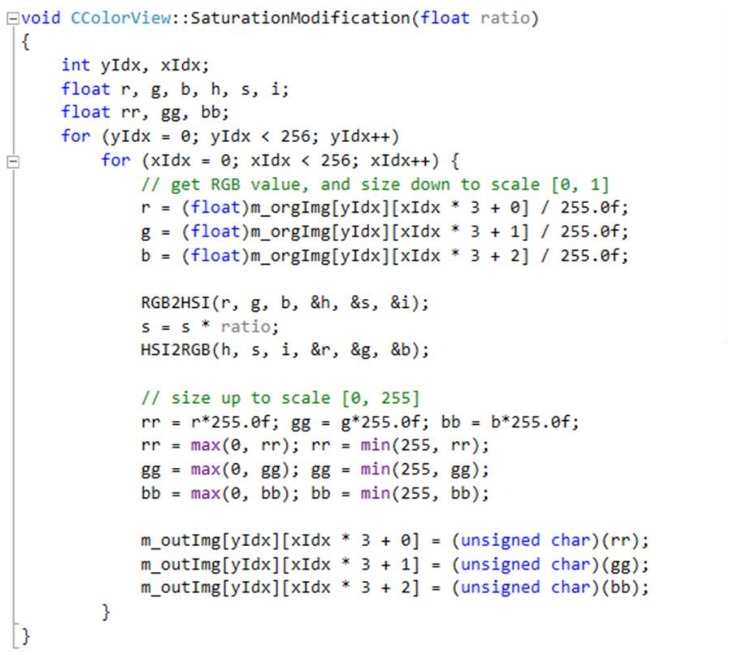
Figure 1 C++ code for changing color space (RGB → HSI)



```cpp
void CColorView::HSI2RGB(float h, float s, float i, float *r, float *g, float *b) {
    float rr, gg, bb;
    float pi = 3.14159265358979323846;
    if (s > 1) s = 1;
    if (i > 1) i = 1;
    if (s == 0) {
        *r = i; *g = i; *b = i;
    }
    else {
        if ((h >= 0) && (h < 2 * pi / 3)) {
            bb = (1 - s) / 3;
            rr = (1 + s*cos(h) / cos(pi / 3 - h)) / 3;
            gg = 1 - rr - bb;
        }
        else if ((h >= 2 * pi / 3) && (h < 4 * pi / 3)) {
            h = h - 2 * pi / 3;
            rr = (1 - s) / 3;
            gg = (1 + s*cos(h) / cos(pi / 3 - h)) / 3;
            bb = 1 - rr - gg;
        }
        else if ((h >= 4 * pi / 3) && (h < 2 * pi)) {
            h = h - 4 * pi / 3;
            gg = (1 - s) / 3;
            bb = (1 + s*cos(h) / cos(pi / 3 - h)) / 3;
            rr = 1 - bb - gg;
        }
        if (rr < 0) rr = 0; if (gg < 0) gg = 0; if (bb < 0) bb = 0;
        *r = 3 * i * rr; *g = 3 * i * gg; *b = 3 * i * bb;
        if (*r > 1) *r = 1; if (*g > 1) *g = 1; if (*b > 1) *b = 1;
    }
}
```

Figure 2 C++ code for changing color space (HSI → RGB)

The C++ code of function modifying saturation by given ratio is as following.



```
void CColorView::SaturationModification(float ratio)
{
    int yIdx, xIdx;
    float r, g, b, h, s, i;
    float rr, gg, bb;
    for (yIdx = 0; yIdx < 256; yIdx++)
        for (xIdx = 0; xIdx < 256; xIdx++) {
            // get RGB value, and size down to scale [0, 1]
            r = (float)m_orgImg[yIdx][xIdx * 3 + 0] / 255.0f;
            g = (float)m_orgImg[yIdx][xIdx * 3 + 1] / 255.0f;
            b = (float)m_orgImg[yIdx][xIdx * 3 + 2] / 255.0f;

            RGB2HSI(r, g, b, &h, &s, &i);
            s = s * ratio;
            HSI2RGB(h, s, i, &r, &g, &b);

            // size up to scale [0, 255]
            rr = r*255.0f; gg = g*255.0f; bb = b*255.0f;
            rr = max(0, rr); rr = min(255, rr);
            gg = max(0, gg); gg = min(255, gg);
            bb = max(0, bb); bb = min(255, bb);

            m_outImg[yIdx][xIdx * 3 + 0] = (unsigned char)(rr);
            m_outImg[yIdx][xIdx * 3 + 1] = (unsigned char)(gg);
            m_outImg[yIdx][xIdx * 3 + 2] = (unsigned char)(bb);
        }
}
```

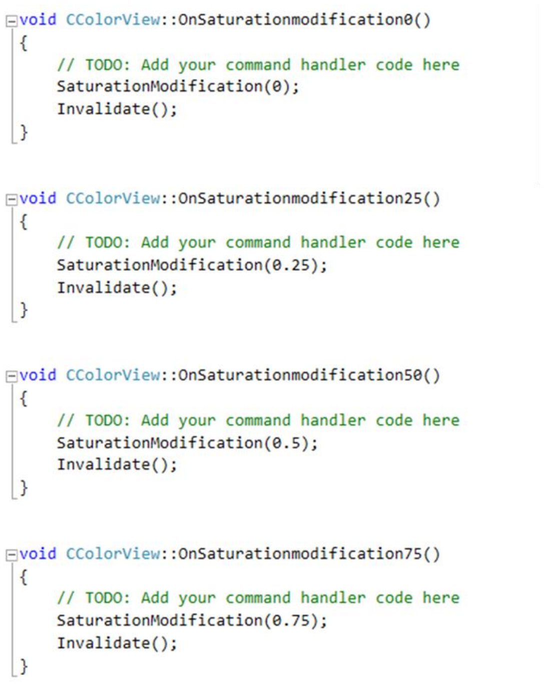
Figure 3 C++ code for modifying saturation by given ratio

Note that we should add functions above in ColorView.h as following.

```
void RGB2HSI(float r, float g, float b, float *h, float *s, float *i);
void HSI2RGB(float h, float s, float i, float *r, float *g, float *b);
void SaturationModification(float ratio);
```

Figure 4 Added C++ code on ColorView.h

Then, menu buttons for saturation modification (75%, 50%, 25%, 0%) correspond to each handler function as following.



```
void CColorView::OnSaturationmodification0()
{
    // TODO: Add your command handler code here
    SaturationModification(0);
    Invalidate();
}

void CColorView::OnSaturationmodification25()
{
    // TODO: Add your command handler code here
    SaturationModification(0.25);
    Invalidate();
}

void CColorView::OnSaturationmodification50()
{
    // TODO: Add your command handler code here
    SaturationModification(0.5);
    Invalidate();
}

void CColorView::OnSaturationmodification75()
{
    // TODO: Add your command handler code here
    SaturationModification(0.75);
    Invalidate();
}
```

Figure 5 C++ code for modifying saturation (0%, 25%, 50%, 75%)

Note that we should add handler as following. It can be easily done with class wizard.

```
ON_COMMAND(ID_SATURATIONMODIFICATION_0, &CColorView::OnSaturationmodification0)
ON_COMMAND(ID_SATURATIONMODIFICATION_25, &CColorView::OnSaturationmodification25)
ON_COMMAND(ID_SATURATIONMODIFICATION_50, &CColorView::OnSaturationmodification50)
ON_COMMAND(ID_SATURATIONMODIFICATION_75, &CColorView::OnSaturationmodification75)
```

Figure 6 Adding handlers for saturation modification

From attached project, you can click on this menu button (ColorProcesssing – Saturation Modification) to conduct saturation modification.

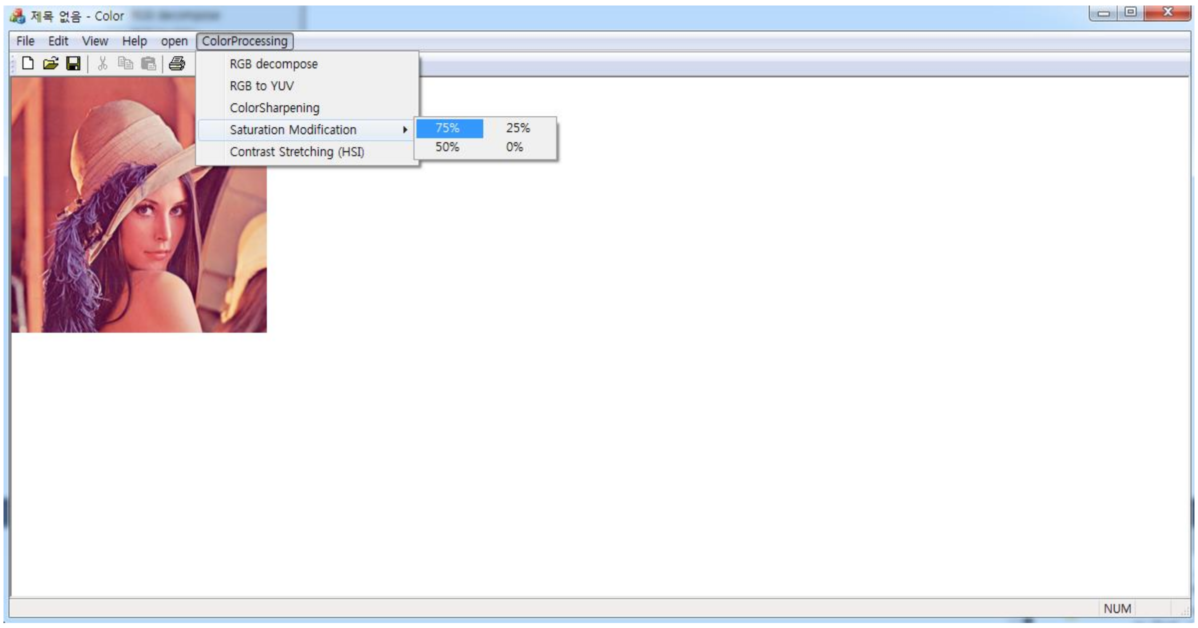


Figure 7 How to access saturation modification

This is the result of saturation modification on Lena image.

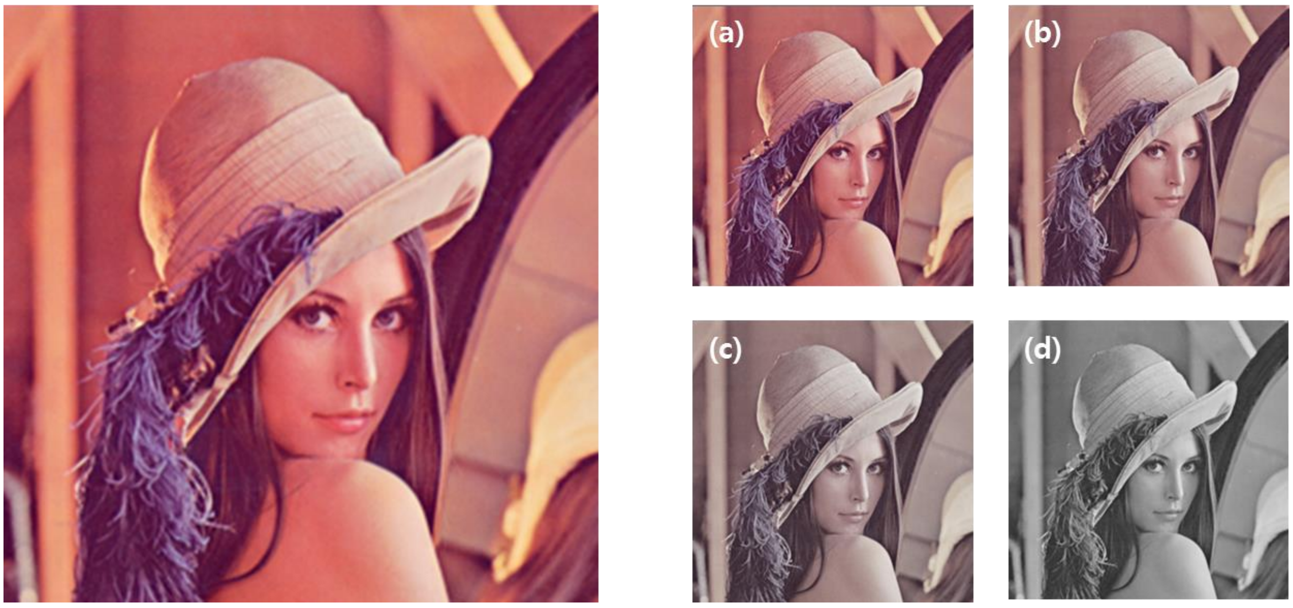


Figure 8 Original image (Left), and saturation modified image - (a) 75%, (b) 50%, (c) 25%, (d) 0%

Q2. Implement color image contrast stretching using following mapping function in HSI or YUV color space.

Contrast stretching is a simple image enhancement technique that attempts to improve the contrast in an image by 'stretching' the range of intensity values it contains to span a desired range of values. It is generally described on a gray scale image. But it can also be used on color images by applying the same method separately to the Red, Green and Blue components of the RGB color intensity values of the image.

We will implement color image contrast stretching in HSI color space, since intensity is related to luminance factor of the image, and thus we can simply conduct stretching on intensity to achieve contrast stretching. We can also stretch Y value, if we are using it in YUV scale. (H, S, U, V are chrominance factor.)

$$\textit{mapping function } f(y) = \begin{cases} 0 & \textit{if } x < \alpha \\ \dfrac{x-\alpha}{\beta-\alpha} & \textit{if } \alpha \le x < \beta \\ 1 & \textit{otherwise} \end{cases}$$

*Note: Determine α, β as you like.

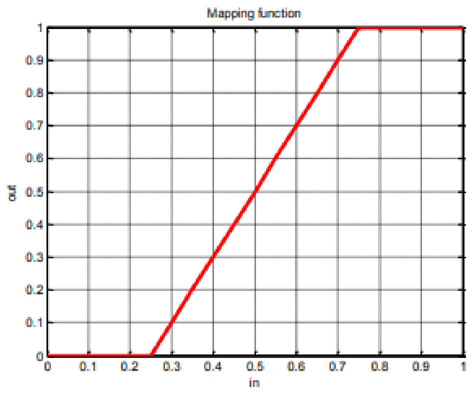
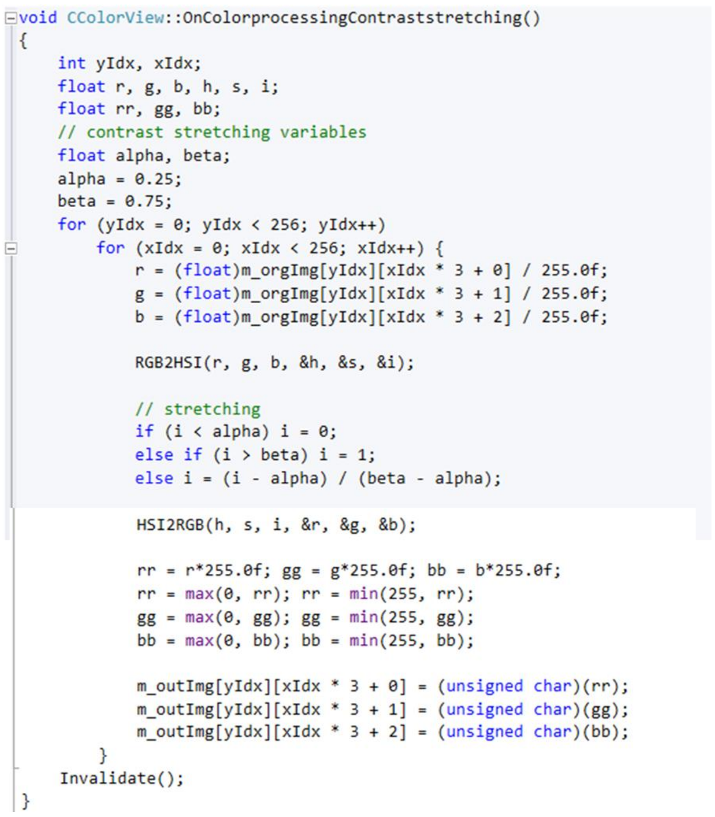


Figure. Example of mapping function for contrast stretching

We followed mapping function as suggested, with value $\alpha = 0.25, \beta = 0.75$.

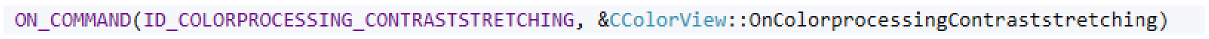
The C++ code for our implementation is as following.



```cpp
void CColorView::OnColorprocessingContraststretching()
{
    int yIdx, xIdx;
    float r, g, b, h, s, i;
    float rr, gg, bb;
    // contrast stretching variables
    float alpha, beta;
    alpha = 0.25;
    beta = 0.75;
    for (yIdx = 0; yIdx < 256; yIdx++)
        for (xIdx = 0; xIdx < 256; xIdx++) {
            r = (float)m_orgImg[yIdx][xIdx * 3 + 0] / 255.0f;
            g = (float)m_orgImg[yIdx][xIdx * 3 + 1] / 255.0f;
            b = (float)m_orgImg[yIdx][xIdx * 3 + 2] / 255.0f;

            RGB2HSI(r, g, b, &h, &s, &i);

            // stretching
            if (i < alpha) i = 0;
            else if (i > beta) i = 1;
            else i = (i - alpha) / (beta - alpha);

            HSI2RGB(h, s, i, &r, &g, &b);

            rr = r*255.0f; gg = g*255.0f; bb = b*255.0f;
            rr = max(0, rr); rr = min(255, rr);
            gg = max(0, gg); gg = min(255, gg);
            bb = max(0, bb); bb = min(255, bb);

            m_outImg[yIdx][xIdx * 3 + 0] = (unsigned char)(rr);
            m_outImg[yIdx][xIdx * 3 + 1] = (unsigned char)(gg);
            m_outImg[yIdx][xIdx * 3 + 2] = (unsigned char)(bb);
        }
    Invalidate();
}
```

Figure 9 C++ code for contrast stretching

Note that we should add handler as following. It can be easily done by class wizard.



```cpp
ON_COMMAND(ID_COLORPROCESSING_CONTRASTSTRETCHING, &CColorView::OnColorprocessingContraststretching)
```

Figure 10 Adding handler for contrast stretching

From attached project, you can click on this menu button (ColorProcesssing – Contrast Stretching (HIS)) to conduct contrast stretching.

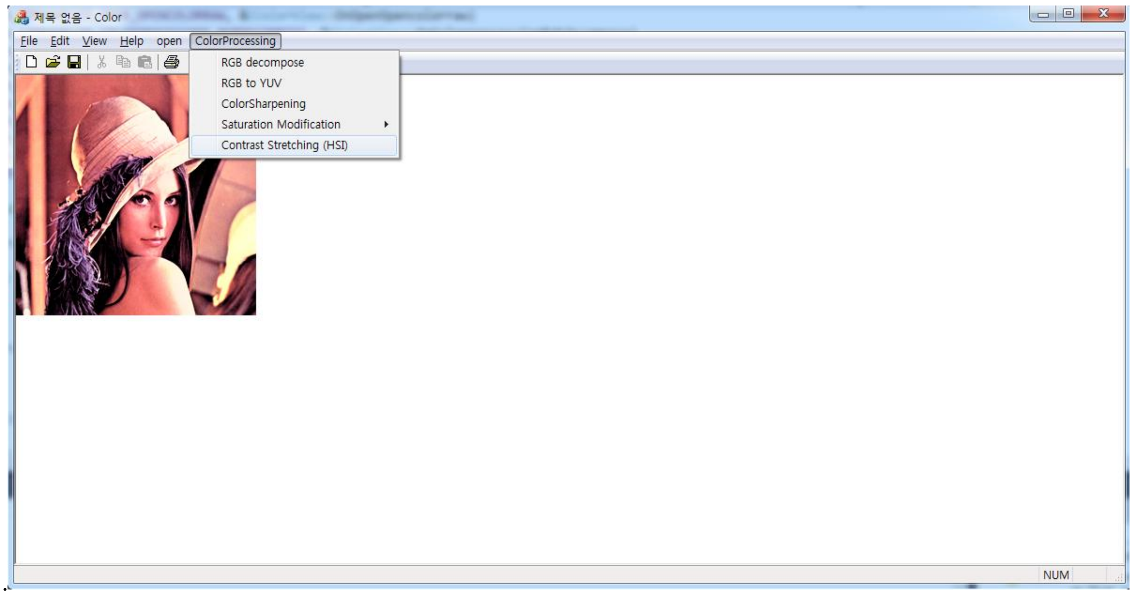


Figure 11 How to access contrast stretching

This is the result of contrast stretching on Lena image.

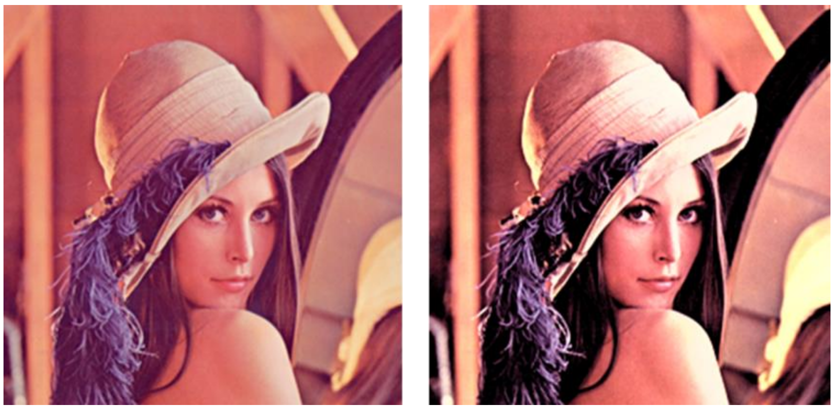


Figure 12 Original image (Left), and contrast stretched image (Right)