Color Conversion

2017/4/18

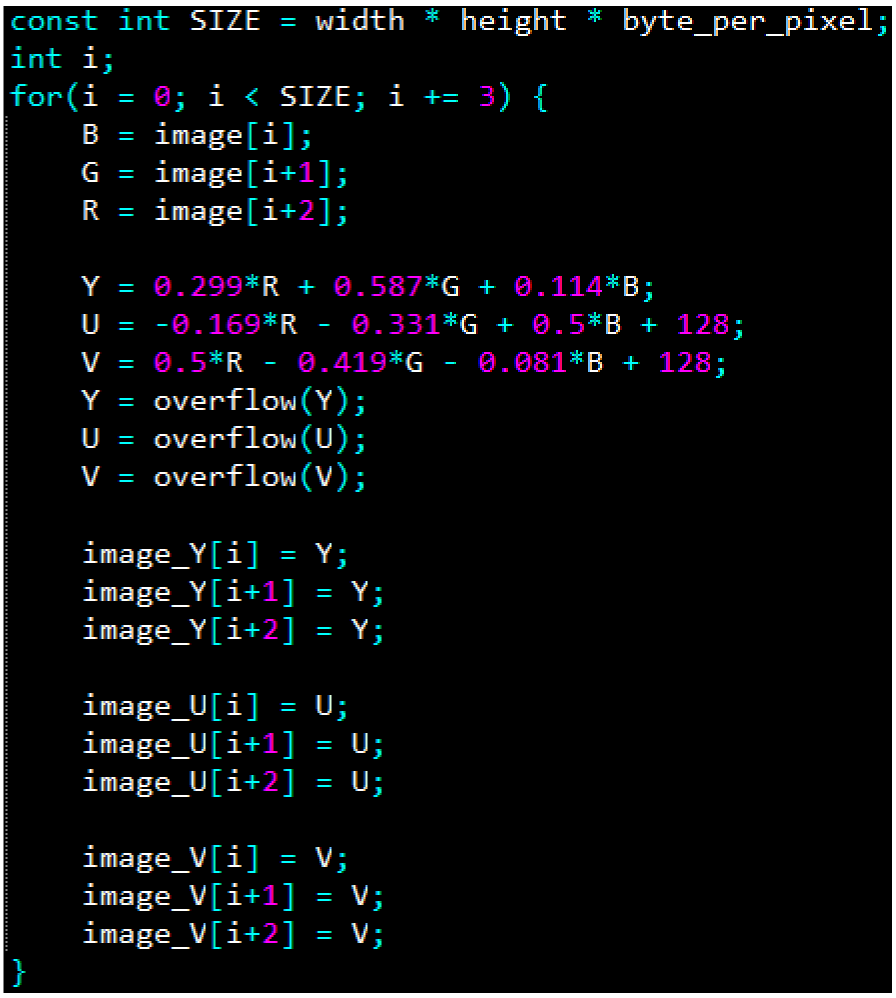
范真瑋

RGB to YUV

實驗內容：

- **撰寫RGB to YUV的C/C++程式**

程式碼：

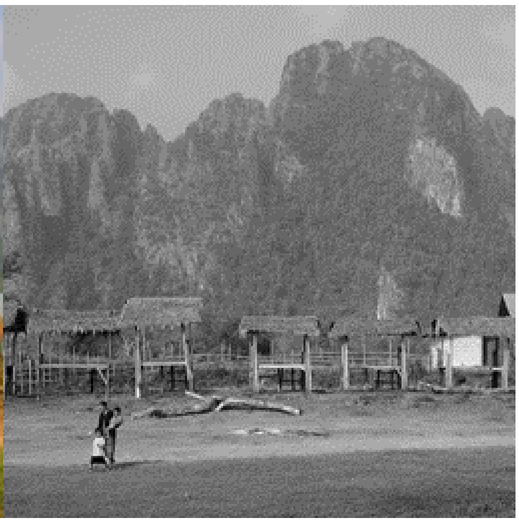


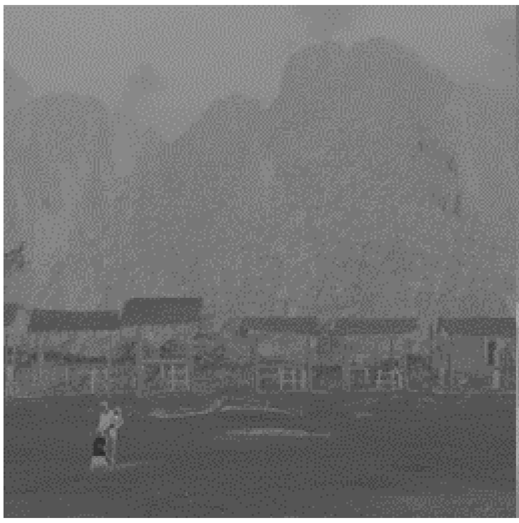
```
const int SIZE = width * height * byte_per_pixel;
int i;
for(i = 0; i < SIZE; i += 3) {
    B = image[i];
    G = image[i+1];
    R = image[i+2];

    Y = 0.299*R + 0.587*G + 0.114*B;
    U = -0.169*R - 0.331*G + 0.5*B + 128;
    V = 0.5*R - 0.419*G - 0.081*B + 128;
    Y = overflow(Y);
    U = overflow(U);
    V = overflow(V);

    image_Y[i] = Y;
    image_Y[i+1] = Y;
    image_Y[i+2] = Y;

    image_U[i] = U;
    image_U[i+1] = U;
    image_U[i+2] = U;

    image_V[i] = V;
    image_V[i+1] = V;
    image_V[i+2] = V;
}
```

實驗結果及分析：

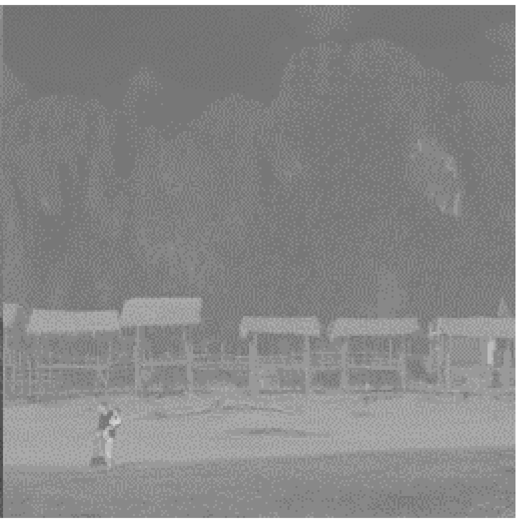
Original RGB image

Y





U



V

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299000 & 0.587000 & 0.114000 \\ -0.168736 & -0.331264 & 0.500002 \\ 0.500000 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

translate from *RGB* to *YUV*

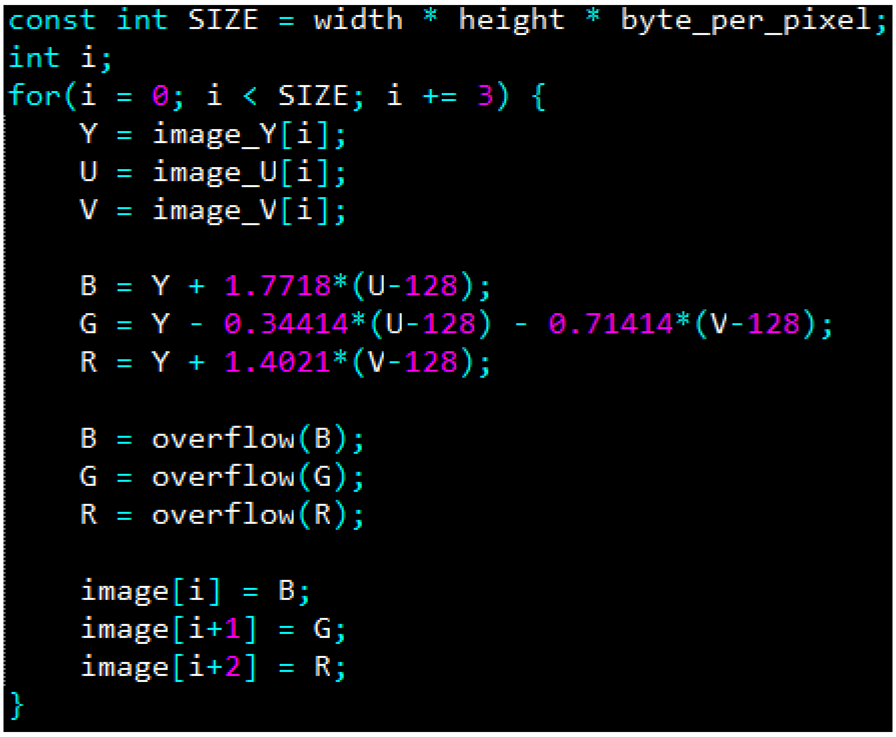
透過公式將 RGB 轉成 YUV，並檢查是否發生 overflow。

YUV to RGB

實驗內容：

撰寫YUV to RGB的C/C++程式

程式碼：



```
const int SIZE = width * height * byte_per_pixel;
int i;
for(i = 0; i < SIZE; i += 3) {
    Y = image_Y[i];
    U = image_U[i];
    V = image_V[i];

    B = Y + 1.7718*(U-128);
    G = Y - 0.34414*(U-128) - 0.71414*(V-128);
    R = Y + 1.4021*(V-128);

    B = overflow(B);
    G = overflow(G);
    R = overflow(R);

    image[i] = B;
    image[i+1] = G;
    image[i+2] = R;
}
```

實驗結果及分析：

converted

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.40210 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.77180 & 0.0 \end{bmatrix} + \begin{bmatrix} Y \\ U-128 \\ V-128 \end{bmatrix}$$

translate from *YUV* to *RGB*

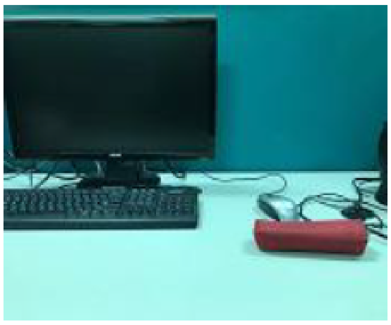
透過公式將 YUV 轉回 RGB，並檢查是否發生 overflow。

計算目標物重心座標

實驗內容：

計算紅色(255, 0 , 0)物體位置

- 讀取RGB格式的影像 (red.bmp, 176×144像素)
- 將RGB格式轉換為YUV格式
- 對YUV格式中的V 進行二值化處理
- 進行侵蝕(Erosion)運算將不必要的元素去除
- 計算目標重心座標

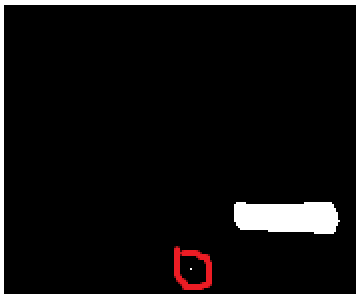

red.bmp

程式碼：

```
int thresholding(double data) {
    if(data > 129) {
        return 255;
    }
    else {
        return 0;
    }
}
```

```
const int SIZE = width * height * byte_per_pixel;
int i, sum_X = 0, sum_Y = 0;
for(i = 0; i < SIZE; i += 3) {
    B = image[i];
    G = image[i+1];
    R = image[i+2];

    V = 0.5*R - 0.419*G - 0.081*B + 128;
    V = thresholding(V);

    image_V[i] = V;
    image_V[i+1] = V;
    image_V[i+2] = V;
    if(V == 255) {
        sum_X += x;
        sum_Y += y;
        printf("%d %d\n", x, y);
        ++countnum;
    }
    ++x;
    if(x > width) {
        x = 0;
        ++y;
    }
}

sum_X /= countnum;
sum_Y /= countnum;
int cg = sum_X + sum_Y * width;
image_V[cg] = 255;
image_V[cg+1] = 255;
image_V[cg+2] = 255;
//printf("%d %d\n", sum_X, sum_Y);
```

實驗結果及分析：



由於不清楚如何計算座標位置，所以最後計算出的重心位置是錯誤的。