专业：求是科学班（计算机）
姓名：蒋仕彪
学号：3170102587
日期：2018/10/24

浙江大学实验报告

课程名称：图象信息处理　　指导老师：宋明黎　　成绩：_____
实验名称：Assignment-1 RGB-YUV 互转

一、实验目的和要求

学习 bmp 类型的图象存储格式，掌握 bmp 类型文件的读写与 RGB 与 YUV 类型的转换。通过只保留 YUV 格式中的 Y 生成灰度图像，并学会通过调节 Y 来调节图片亮度。

二、实验内容和原理

老师给出的 RGB 转 YUV 的公式如下：

- Lightness is computed as:

$$Y = 0.3R + 0.59G + 0.11B$$

- Color differences U, V are computed as:

$$U = 0.493(B - Y)$$
$$V = 0.877(R - Y)$$

因为最终生成的 YUV 还要转会 RGB 输出，我觉得 Y 的两位精度不是很够。于是我上网搜索了一下，找到了更精确一点的公式：

```
YUV RGB_To_YUV(RGB cur){
    YUV ret;
    double Y = 0.299 * cur.R + 0.587 * cur.G + 0.114 * cur.B;
    ret.Y = round(Y);
    ret.U = round(0.493 * (cur.B - Y));
    ret.V = round(0.877 * (cur.R - Y));
    return ret;
}
```

解方程组，可以得到逆变换（YUV 转 RGB）：

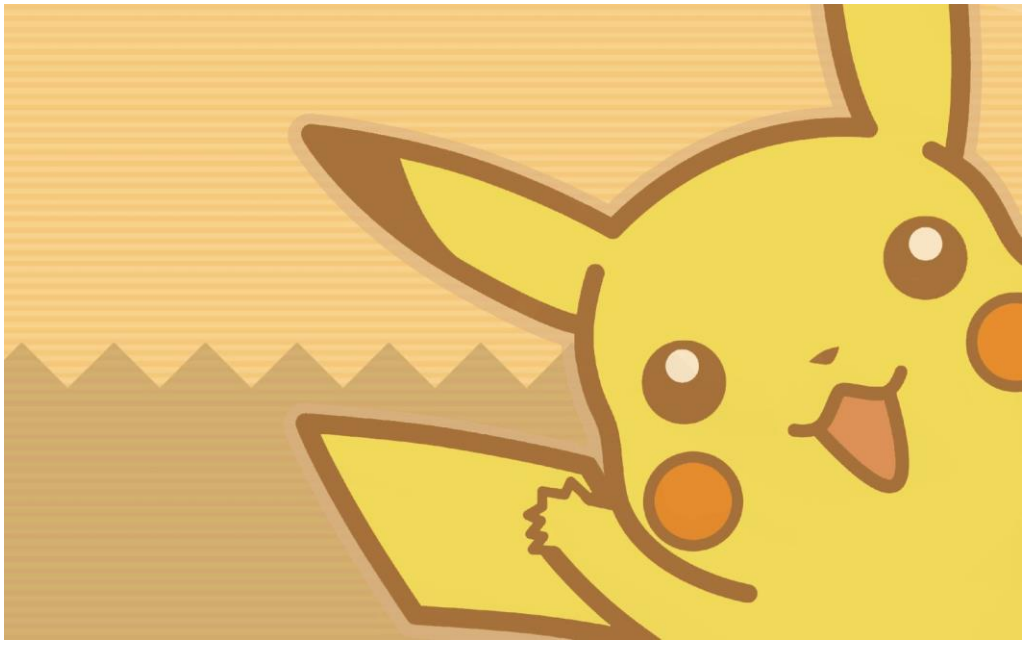
```
RGB YUV_To_RGB(YUV cur){
    RGB ret;
    ret.R = round(cur.Y + 1.14 * cur.V);
    ret.G = round(cur.Y - 0.395 * cur.U - 0.581 * cur.V);
    ret.B = round(cur.Y + 2.033 * cur.U);
    return ret;
}
```

生成灰度图像时，直接取 YUV 中的 Y 即可。这样转回 RGB 后，直接设 R=G=B=Y 即可输出。

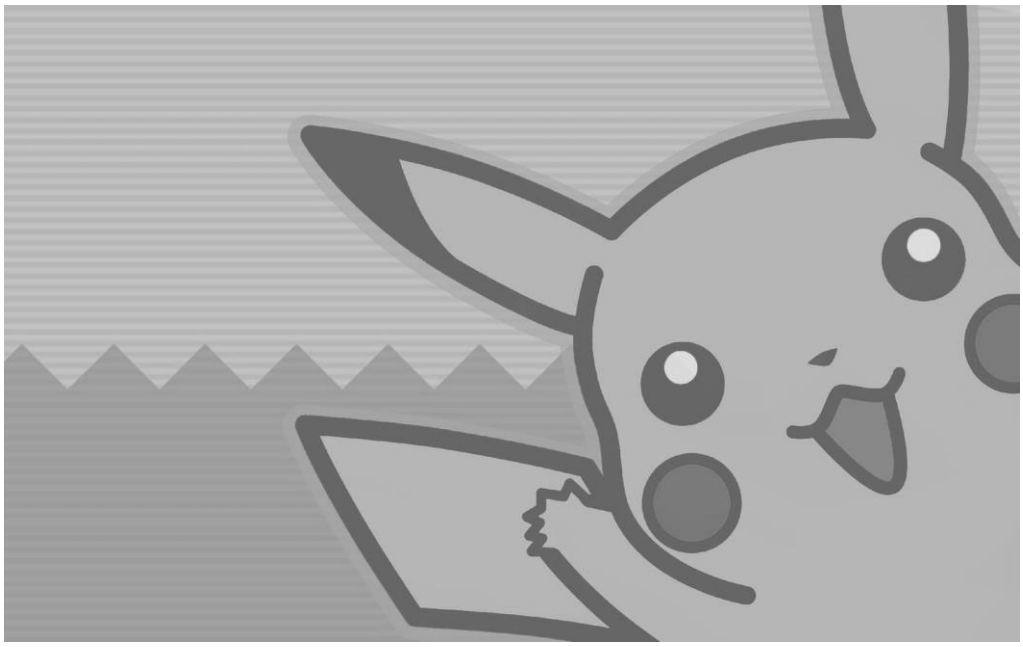
还有一个要求是，将 Y 均匀分布。我找的图的 Y 取值为 83~233，我就通过一个函数，把它们均匀地映射到 0~255 上了。效果很显著。

调亮度的实验，我就直接把所有 Y 减掉了 80，再转回 RGB。值得注意的是，YUV 格式本身会有负数，我一开始没注意到，就频频出错。最后我改用 int 存 YUV 格式就能达到效果了。

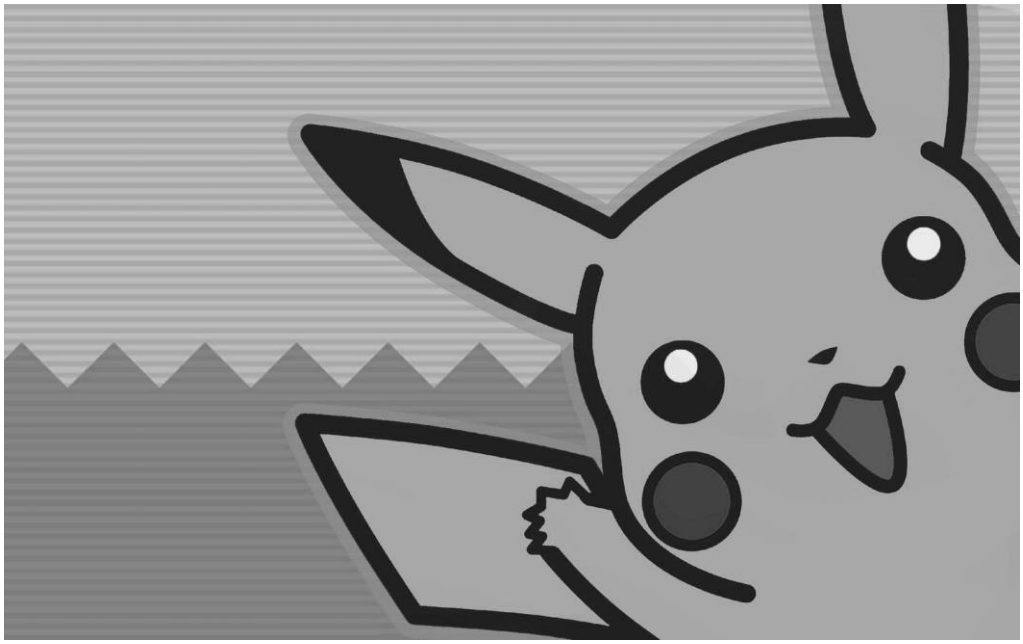
三、成果展示



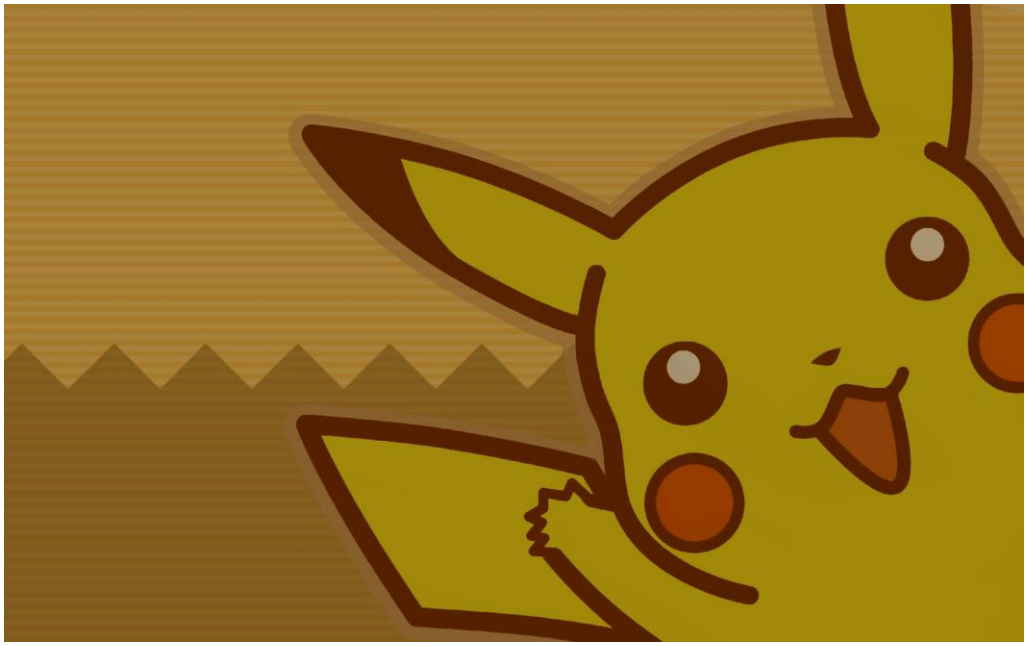
原图



直接转灰度图像



将 Y 均匀分布后的灰度图像



将原图调暗后的图片

四、源代码与分析

```
#include <stdio.h>
#include <windows.h>
#include <assert.h>
#include <math.h>

using namespace std;

FILE* fin  = fopen("Origin.bmp", "rb");   //读入文件

struct RGB{
    BYTE R;
    BYTE G;
    BYTE B;
};


struct YUV{
    int Y;
    int U;
    int V;
};
//YUV 格式可能会有负数，就直接用 int 存了

YUV RGB_To_YUV(RGB cur){

    YUV ret;
    double Y = 0.299 * cur.R + 0.587 * cur.G + 0.114 * cur.B;
    ret.Y = round(Y);
    ret.U = round(0.493 * (cur.B - Y));
    ret.V = round(0.877 * (cur.R - Y));
    return ret;
}


BYTE In(int cur){
    if (cur > 255) cur = 255;
    if (cur < 0)   cur = 0;
    return (BYTE)cur;
}
//担心 YUV 转 RGB 时导致 RGB 范围出错，写一个框定范围的函数
```

```
RGB YUV_To_RGB(YUV cur){
    RGB ret;
    ret.R = In(round(cur.Y + 1.14 * cur.V));
    ret.G = In(round(cur.Y - 0.395 * cur.U - 0.581 * cur.V));
    ret.B = In(round(cur.Y + 2.033 * cur.U));
    return ret;
}

void Rearrange(BYTE *Gray, int S){
    int Max = Gray[0], Min = Gray[0];
    for (int i = 1; i < S; i++){
        if (Gray[i] > Max) Max = Gray[i];
        if (Gray[i] < Min) Min = Gray[i];
    }
    assert(Min < Max);
    fprintf(stderr, "%d %d\n", Min, Max);
    for (int i = 0; i < S; i++){
        Gray[i] = round((Gray[i] - Min) * 1.0 / (Max - Min) * 255);
        assert(Gray[i] >= 0 && Gray[i] <= 255);
    }
}
//重新分配灰度值


void Printbmp(BITMAPFILEHEADER &bmfh, BITMAPINFOHEADER &bmih, RGB *grid, char *str){
    FILE *fout = fopen(str, "wb");
    fwrite(&bmfh, sizeof(bmfh), 1, fout);
    fwrite(&bmih, sizeof(bmih), 1, fout);
    fwrite(grid, sizeof(RGB), bmih.biHeight * bmih.biWidth, fout);
}
//输出图像

int main(){
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    fread(&bmfh, sizeof(BITMAPFILEHEADER), 1, fin);
    fread(&bmih, sizeof(BITMAPINFOHEADER), 1, fin);
    int S = bmih.biHeight * bmih.biWidth;
    RGB *Origin = new RGB[S];
    fread(Origin, sizeof(RGB), S, fin);
    //读入图像

    assert(bmih.biBitCount == 24);
    //确定是 24 位 bmp 图像
```

```
    YUV *New = new YUV[S];
    BYTE *Gray = new BYTE[S];
    RGB *Print = new RGB[S];

    int Min = 255, Max = 0;
    for (int i = 0; i < S; i++){
        New[i] = RGB_To_YUV(Origin[i]);
        Gray[i] = New[i].Y;
    }

    for (int i = 0; i < S; i++)
        Print[i].R = Print[i].G = Print[i].B = Gray[i];
    char str1[] = "Gray1.bmp";
    Printbmp(bmfh, bmih, Print, str1);
    //直接输出灰度图像

    Rearrange(Gray, S);
    for (int i = 0; i < S; i++)
        Print[i].R = Print[i].G = Print[i].B = Gray[i];
    char str2[] = "Gray2.bmp";
    Printbmp(bmfh, bmih, Print, str2);
    //均匀分布后输出灰度图像


    for (int i = 0; i < S; i++){
        New[i].Y -= 80;
        Print[i] = YUV_To_RGB(New[i]);
    }
    char str3[] = "Luminance.bmp";
    Printbmp(bmfh, bmih, Print, str3);
    //将图片调暗后输出

    return 0;
}
```