

# 视音频数据处理入门：RGB、YUV像素数据处理

 [blog.csdn.net/leixiaohua1020/article/details/50534150](https://blog.csdn.net/leixiaohua1020/article/details/50534150)

≡ 分类：

=====

视音频数据处理入门系列文章：

[视音频数据处理入门：RGB、YUV像素数据处理](#)

[视音频数据处理入门：PCM音频采样数据处理](#)

[视音频数据处理入门：H.264视频码流解析](#)

[视音频数据处理入门：AAC音频码流解析](#)

[视音频数据处理入门：FLV封装格式解析](#)

[视音频数据处理入门：UDP-RTP协议解析](#)

=====

有段时间没有写博客了，这两天写起博客来竟然感觉有些兴奋，仿佛找回了原来的感觉。前一阵子在梳理以前文章的时候，发现自己虽然总结了各种视音频应用程序，却还缺少一个适合无视音频背景人员学习的“最基础”的程序。因此抽时间将以前写过的代码整理成了一个小项目。这个小项目里面包含了一系列简单的函数，可以对RGB/YUV视频像素数据、PCM音频采样数据、H.264视频码流、AAC音频码流、FLV封装格式数据、UDP/RTP协议数据进行简单处理。这个项目的一大特点就是没有使用任何的第三方类库，完全借助于C语言的基本函数实现了功能。通过对这些代码的学习，可以让初学者迅速掌握视音频数据的基本格式。有关上述几种格式的介绍可以参考文章《[\[总结\]视音频编解码技术零基础学习方法](#)》。

从这篇文章开始打算写6篇文章分别记录上述6种不同类型的视音频数据的处理方法。本文首先记录第一部分即RGB/YUV视频像素数据的处理方法。视频像素数据在视频播放器的解码流程中的位置如下图所示。



本文分别介绍如下几个RGB/YUV视频像素数据处理函数：

- 分离YUV420P像素数据中的Y、U、V分量
- 分离YUV444P像素数据中的Y、U、V分量
- 将YUV420P像素数据去掉颜色（变成灰度图）
- 将YUV420P像素数据的亮度减半
- 将YUV420P像素数据的周围加上边框
- 生成YUV420P格式的灰阶测试图
- 计算两个YUV420P像素数据的PSNR
- 分离RGB24像素数据中的R、G、B分量
- 将RGB24格式像素数据封装为BMP图像
- 将RGB24格式像素数据转换为YUV420P格式像素数据
- 生成RGB24格式的彩条测试图

本文中的RGB/YUV文件需要使用RGB/YUV播放器才能查看。YUV播放器种类比较多，例如[YUV Player Deluxe](#)，或者开源播放器（参考文章[《修改了一个YUV/RGB播放器》](#)）等。

## 函数列表

### (1) 分离YUV420P像素数据中的Y、U、V分量

本程序中的函数可以将YUV420P数据中的Y、U、V三个分量分离开来并保存成三个文件。函数的代码如下所示。

```
1. int simplest_yuv420_split(char *url, int w, int h,int num){
```

```

2. FILE *fp=fopen(url,"rb+");
3. FILE *fp1=fopen("output_420_y.y","wb+");
4. FILE *fp2=fopen("output_420_u.y","wb+");
5. FILE *fp3=fopen("output_420_v.y","wb+");
6. unsigned char *pic=(unsigned char *)malloc(w*h*3/2);
7. for(int i=0;i
8.     fread(pic,1,w*h*3/2,fp);
9.     fwrite(pic,1,w*h,fp1);
10.    fwrite(pic+w*h,1,w*h/4,fp2);
11.    fwrite(pic+w*h*5/4,1,w*h/4,fp3);
12. }
13. free(pic);
14. fclose(fp);
15. fclose(fp1);
16. fclose(fp2);
17. fclose(fp3);
18. return 0;
19. }

```

调用上面函数的方法如下所示。

```
1. simplest_yuv420_split("lena_256x256_yuv420p.yuv",256,256,1);
```

从代码可以看出，如果视频帧的宽和高分别为w和h，那么一帧YUV420P像素数据一共占用 $w \times h \times 3/2$  Byte的数据。其中前 $w \times h$  Byte存储Y，接着的 $w \times h \times 1/4$  Byte存储U，最后 $w \times h \times 1/4$  Byte存储V。上述调用函数的代码运行后，将会把一张分辨率为256x256的名称为lena\_256x256\_yuv420p.yuv的YUV420P格式的像素数据文件分离成为三个文件：

*output\_420\_y.y : 纯Y数据，分辨率为256x256。*

*output\_420\_u.y : 纯U数据，分辨率为128x128。*

*output\_420\_v.y : 纯V数据，分辨率为128x128。*

注：本文中像素的采样位数一律为8bit。由于1Byte=8bit，所以一个像素的一个分量的采样值占用1Byte。

程序输入的原图如下所示。

lena\_256x256\_yuv420p.yuv

程序输出的三个文件的截图如下图所示。在这里需要注意输出的U、V分量在YUV播放器中也是当做Y分量进行播放的。

output\_420\_y.y

output\_420\_u.y和output\_420\_v.y

## (2)分离YUV444P像素数据中的Y、U、V分量

本程序中的函数可以将YUV444P数据中的Y、U、V三个分量分离开来并保存成三个文件。函数的代码如下所示。

```
1. int simplest_yuv444_split(char *url, int w, int h, int num)
   {
2.     FILE *fp=fopen(url, "rb+");
3.     FILE *fp1=fopen("output_444_y.y", "wb+");
4.     FILE *fp2=fopen("output_444_u.y", "wb+");
5.     FILE *fp3=fopen("output_444_v.y", "wb+");
6.     unsigned char *pic=(unsigned char *)malloc(w*h*3);
7.     for(int i=0; i
8.         fread(pic, 1, w*h*3, fp);
9.         fwrite(pic, 1, w*h, fp1);
10.        fwrite(pic+w*h, 1, w*h, fp2);
11.        fwrite(pic+w*h*2, 1, w*h, fp3);
12.    }
13.    free(pic);
14.    fclose(fp);
15.    fclose(fp1);
16.    fclose(fp2);
17.    fclose(fp3);
18.    return 0;
19. }
```

调用上面函数的方法如下所示。

```
1. simplest_yuv444_split("lena_256x256_yuv444p.yuv", 256, 256, 1);
```

从代码可以看出，如果视频帧的宽和高分别为w和h，那么一帧YUV444P像素数据一共占用w\*h\*3 Byte的数据。其中前w\*h Byte存储Y，接着的w\*h Byte存储U，最后w\*h Byte存储V。上述调用函数的代码运行后，将会把一张分辨率为256x256的名称为lena\_256x256\_yuv444p.yuv的YUV444P格式的像素数据文件分离成为三个文件：



*output\_444\_y.y* : 纯Y数据, 分辨率为256x256。  
*output\_444\_u.y* : 纯U数据, 分辨率为256x256。  
*output\_444\_v.y* : 纯V数据, 分辨率为256x256。

输入的原图如下所示。

输出的三个文件的截图如下图所示。

*output\_444\_y.y*

*output\_444\_u.y*

*output\_444\_v.y*

### (3) 将YUV420P像素数据去掉颜色 (变成灰度图)

本程序中的函数可以将YUV420P格式像素数据的彩色去掉, 变成纯粹的灰度图。函数的代码如下。

```
1. int simplest_yuv420_gray(char *url, int w, int h,int num)
   {
2.     FILE *fp=fopen(url,"rb+");
3.     FILE *fp1=fopen("output_gray.yuv","wb+");
4.     unsigned char *pic=
       (unsigned char *)malloc(w*h*3/2);
5.     for(int i=0;i
6.         fread(pic,1,w*h*3/2,fp);
7.         memset(pic+w*h,128,w*h/2);
8.         fwrite(pic,1,w*h*3/2,fp1);
9.     }
10.    free(pic);
11.    fclose(fp);
12.    fclose(fp1);
13.    return 0;
14. }
```

调用上面函数的方法如下所示。

```
1. simplest_yuv420_gray("lena_256x256_yuv420p.yuv",256,256,1);
```





从代码可以看出，如果想把YUV格式像素数据变成灰度图像，只需要将U、V分量设置成128即可。这是因为U、V是图像中的经过偏置处理的色度分量。色度分量在偏置处理前的取值范围是-128至127，这时候的无色对应的是“0”值。经过偏置后色度分量取值变成了0至255，因而此时的无色对应的就是128了。上述调用函数的代码运行后，将会把一张分辨率为256x256的名为lena\_256x256\_yuv420p.yuv的YUV420P格式的像素数据文件处理成名为output\_gray.yuv的YUV420P格式的像素数据文件。输入的原图如下所示。

处理后的图像如下所示。

#### (4)将YUV420P像素数据的亮度减半

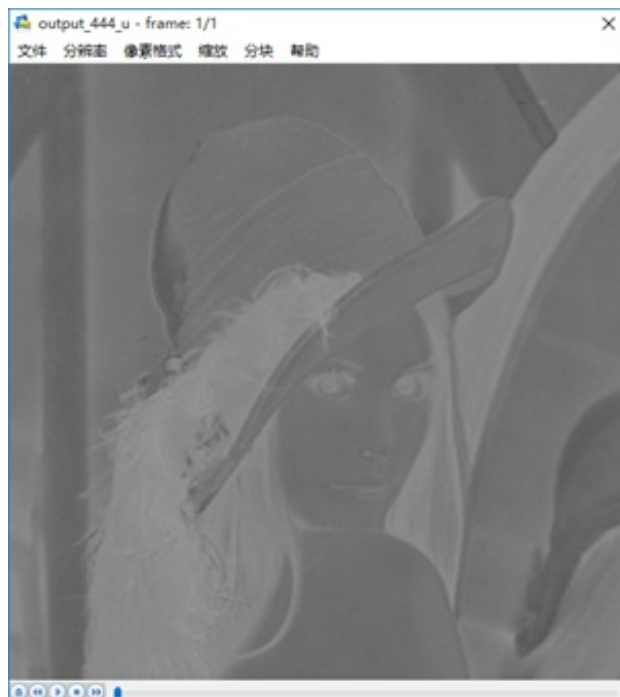
本程序中的函数可以通过将YUV数据中的亮度分量Y的数值减半的方法，降低图像的亮度。函数代码如下所示。

```
1. int simplest_yuv420_halfy(char *url, int w, int h,int num)
   {
2.     FILE *fp=fopen(url,"rb+");
3.     FILE *fp1=fopen("output_half.yuv","wb+");
4.     unsigned char *pic=
       (unsigned char *)malloc(w*h*3/2);
5.     for(int i=0;i
6.         fread(pic,1,w*h*3/2,fp);
7.         for(int j=0;j
8.             unsigned char temp=pic[j]/2;
9.             pic[j]=temp;
10.        }
11.        fwrite(pic,1,w*h*3/2,fp1);
12.    }
13.    free(pic);
14.    fclose(fp);
15.    fclose(fp1);
16.    return 0;
17. }
```

调用上面函数的方法如下所示。

```
1. simplest_yuv420_halfy("lena_256x256_yuv420p.yuv",256,256,1);
```

从代码可以看出，如果打算将图像的亮度减半，只要将图像的每个像素的Y值取出来分别进行除以2的工作就可以了。图像的每个Y值占用1 Byte，取值范围是0至255，对应C语言中的unsigned char数据类型。上述调用函数的代码运行后，将会把一张分辨率为256x256的名为lena\_256x256\_yuv420p.yuv的YUV420P格式的像素数据文件



处理成名称为output\_half.yuv的YUV420P格式的像素数据文件。输入的原图如下所示。

处理后的图像如下所示。

### (5)将YUV420P像素数据的周围加上边框

本程序中的函数可以通过修改YUV数据中特定位置的亮度分量Y的数值，给图像添加一个“边框”的效果。函数代码如下所示。



```
1. int simplest_yuv420_border(char *url, int w, int h,int border,int num){
2.     FILE *fp=fopen(url,"rb+");
3.     FILE *fp1=fopen("output_border.yuv","wb+");
4.     unsigned char *pic=(unsigned char *)malloc(w*h*3/2);
5.     for(int i=0;i
6.         fread(pic,1,w*h*3/2,fp);
7.         for(int j=0;j
8.             for(int k=0;k
9.                 if(k(w-border)||j(h-border)){
10.                     pic[j*w+k]=255;
11.                 }
```

```

12.     }
13. }
14.     fwrite(pic,1,w*h*3/2,fp1);
15. }
16. free(pic);
17. fclose(fp);
18. fclose(fp1);
19. return 0;
20. }

```

调用上面函数的方法如下所示。



```
1. simplest_yuv420_border("lena_256x256_yuv420p.yuv",256,256,20,1);
```

从代码可以看出，图像的边框的宽度为border，本程序将距离图像边缘border范围内的像素的亮度分量Y的取值设置成了亮度最大值255。上述调用函数的代码运行后，将会把一张分辨率为256x256的名称为lena\_256x256\_yuv420p.yuv的YUV420P格式的像素数据文件处理成名称为output\_border.yuv的YUV420P格式的像素数据文件。输入的原图如下所示。



处理后的图像如下所示。

## (6) 生成YUV420P格式的灰阶测试图

本程序中的函数可以生成一张YUV420P格式的灰阶测试图。  
函数代码如下所示。



```
1. int simplest_yuv420_graybar(int width, int height,int ymin,int ymax,int barnum,char *url_out){
2.     int barwidth;
3.     float lum_inc;
4.     unsigned char lum_temp;
5.     int uv_width,uv_height;
6.     FILE *fp=NULL;
7.     unsigned char *data_y=NULL;
8.     unsigned char *data_u=NULL;
9.     unsigned char *data_v=NULL;
10.    int t=0,i=0,j=0;
11.    barwidth=width/barnum;
```

```

12. lum_inc=((float)(ymax-ymin))/((float)(barnum-1));
13. uv_width=width/2;
14. uv_height=height/2;
15. data_y=(unsigned char *)malloc(width*height);
16. data_u=(unsigned char *)malloc(uv_width*uv_height);
17. data_v=(unsigned char *)malloc(uv_width*uv_height);
18. if((fp=fopen(url_out,"wb+"))==NULL){
19.     printf("Error: Cannot create file!");
20.     return -1;
21. }
22. printf("Y, U, V value from picture's left to right:\n");
23. for(t=0;t<(width/barwidth);t++){
24.     lum_temp=ymin+(char)(t*lum_inc);
25.     printf("%3d, 128, 128\n",lum_temp);
26. }
27. for(j=0;j
28.     for(i=0;i
29.         t=i/barwidth;
30.         lum_temp=ymin+(char)(t*lum_inc);
31.         data_y[j*width+i]=lum_temp;
32.     }
33. }
34. for(j=0;j
35.     for(i=0;i
36.         data_u[j*uv_width+i]=128;
37.     }
38. }
39. for(j=0;j
40.     for(i=0;i
41.         data_v[j*uv_width+i]=128;
42.     }
43. }
44. fwrite(data_y,width*height,1,fp);
45. fwrite(data_u,uv_width*uv_height,1,fp);
46. fwrite(data_v,uv_width*uv_height,1,fp);
47. fclose(fp);
48. free(data_y);
49. free(data_u);
50. free(data_v);

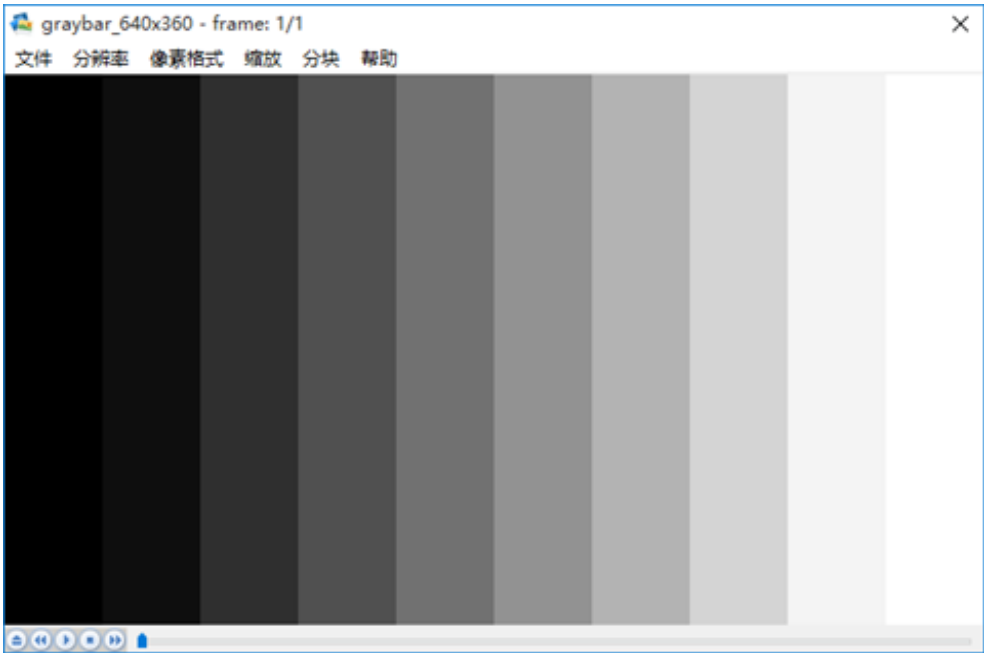
```

```
51.     return 0;
52. }
```

调用上面函数的方法如下所示。

```
1. simplest_yuv420_graybar(640, 360,0,255,10,"graybar_640x360.yuv");
```

从源代码可以看出，本程序一方面通过灰阶测试图的亮度最小值ymin，亮度最大值ymax，灰阶数量barnum确定每一个灰度条中像素的亮度分量Y的取值。另一方面还要根据图像的宽度width和图像的高度height以及灰阶数量barnum确定每一个灰度条的宽度。有了这两方面信息之后，就可以生成相应的图片了。上述调用函数的代码运行后，会生成一个取值范围从0-255，一共包含10个灰度条的YUV420P格式的测试图。测试图的内容如下所示。



从程序也可以得到从左到右10个灰度条的Y、U、V取值，如下所示。

Y	U	V
0	128	128
28	128	128
56	128	128
85	128	128
113	128	128
141	128	128
170	128	128
198	128	128
226	128	128

255	128	128
-----	-----	-----

## (7)计算两个YUV420P像素数据的PSNR

PSNR是最基本的视频质量评价方法。本程序中的函数可以对比两张YUV图片中亮度分量Y的PSNR。函数的代码如下所示。

```

1. int simplest_yuv420_psnr(char *url1,char *url2,int w,int h,int num){
2.     FILE *fp1=fopen(url1,"rb+");
3.     FILE *fp2=fopen(url2,"rb+");
4.     unsigned char *pic1=(unsigned char *)malloc(w*h);
5.     unsigned char *pic2=(unsigned char *)malloc(w*h);
6.     for(int i=0;i
7.         fread(pic1,1,w*h,fp1);
8.         fread(pic2,1,w*h,fp2);
9.         double mse_sum=0,mse=0,psnr=0;
10.    for(int j=0;j
11.        mse_sum+=pow((double)(pic1[j]-pic2[j]),2);
12.    }
13.    mse=mse_sum/(w*h);
14.    psnr=10*log10(255.0*255.0/mse);
15.    printf("%5.3f\n",psnr);
16.    fseek(fp1,w*h/2,SEEK_CUR);
17.    fseek(fp2,w*h/2,SEEK_CUR);
18. }
19. free(pic1);
20. free(pic2);
21. fclose(fp1);
22. fclose(fp2);
23. return 0;
24. }
```

调用上面函数的方法如下所示。

```
1. simplest_yuv420_psnr("lena_256x256_yuv420p.yuv","lena_distort_256x256_yuv420p.yuv",256,256,1);
```

对于8bit量化的像素数据来说，PSNR的计算公式如下所示。

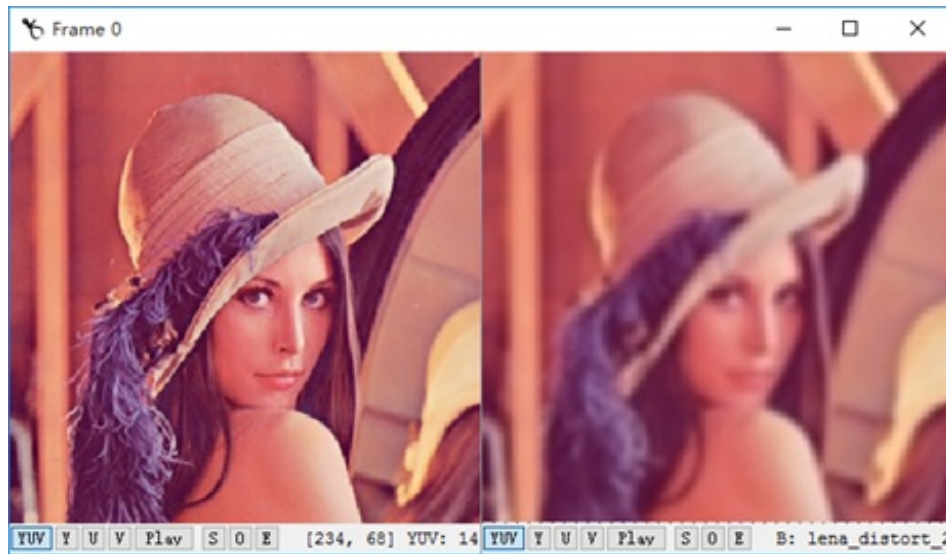
上述公式中mse的计算公式如下所示。

$$\text{PSNR} = 10 * \log_{10}\left(\frac{255^2}{\text{MSE}}\right)$$



其中M，N分别为图像的宽高， $x_{ij}$ 和 $y_{ij}$ 分别为两张图像的每一个像素值。PSNR通常用于质量评价，就是计算受损图像与原始图像之间的差别，以此来评价受损图像的质量。本程序输入的两张图像的对比图如下图所示。其中左边的图像为原始图像，右边的图像为受损图像。

$$MSE = \frac{1}{M * N} \sum_{i=1}^M \sum_{j=1}^N (x_{ij} - y_{ij})^2$$



经过程序计算后得到的PSNR取值为26.693。PSNR取值通常情况下都在20-50的范围内，取值越高，代表两张图像越接近，反映出受损图像质量越好。

## (8) 分离RGB24像素数据中的R、G、B分量

本程序中的函数可以将RGB24数据中的R、G、B三个分量分离开来并保存成三个文件。函数的代码如下所示。

```
1. int simplest_rgb24_split(char *url, int w, int h, int num){
2.     FILE *fp=fopen(url,"rb+");
3.     FILE *fp1=fopen("output_r.y", "wb+");
4.     FILE *fp2=fopen("output_g.y", "wb+");
5.     FILE *fp3=fopen("output_b.y", "wb+");
6.     unsigned char *pic=(unsigned char *)malloc(w*h*3);
7.     for(int i=0; i
8.         fread(pic, 1, w*h*3, fp);
9.         for(int j=0; j
10.             fwrite(pic+j, 1, 1, fp1);
11.             fwrite(pic+j+1, 1, 1, fp2);
12.             fwrite(pic+j+2, 1, 1, fp3);
13.         }
14.     }
15.     free(pic);
16.     fclose(fp);
17.     fclose(fp1);
18.     fclose(fp2);
```

```

19.  fclose(fp3);
20.  return 0;
21. }

```

调用上面函数的方法如下所示。

```
1. simplest_rgb24_split("cie1931_500x500.rgb", 500, 500,1);
```

从代码可以看出，与YUV420P三个分量分开存储不同，RGB24格式的每个像素的三个分量是连续存储的。一帧宽高分别为w、h的RGB24图像一共占用w\*h\*3 Byte的存储空间。RGB24格式规定首先存储第一个像素的R、G、B，然后存储第二个像素的R、G、B...以此类推。类似于YUV420P的存储方式称为Planar方式，而类似于RGB24的存储方式称为Packed方式。上述调用函数的代码运行后，将会把一张分辨率为500x500的名称为cie1931\_500x500.rgb的RGB24格式的像素数据文件分离成为三个文件：

```

output_r.y : R数据，分辨率为256x256。
output_g.y : G数据，分辨率为256x256。

output_b.y : B数据，分辨率为256x256。

```

输入的原图是一张标准的CIE 1931色度图。该色度图右下为红色，上方为绿色，左下为蓝色，如下所示。

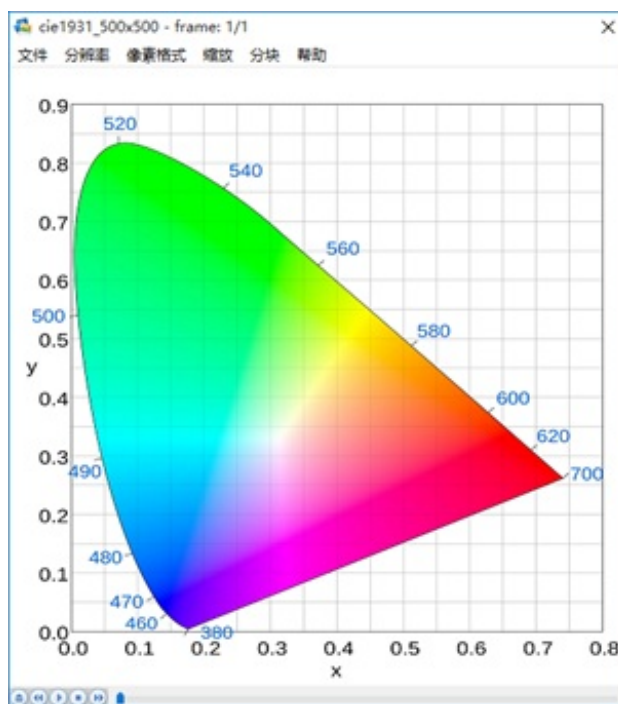
R数据图像如下所示。

G数据图像如下所示。

B数据图像如下所示。

## (9)将RGB24格式像素数据封装为BMP图像

BMP图像内部实际上存储的就是RGB数据。本程序实现了对RGB像素数据的封装处理。通过本程序中的函数，可以将RGB数据封装成为一张BMP图像。



```

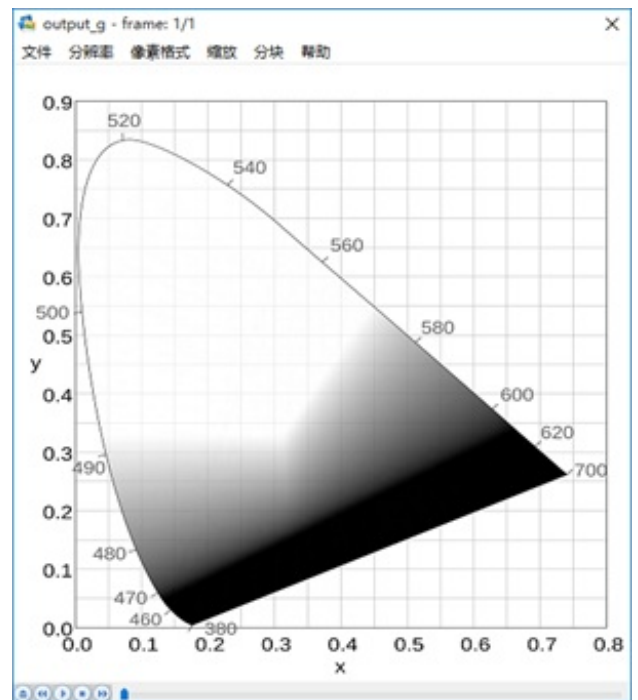
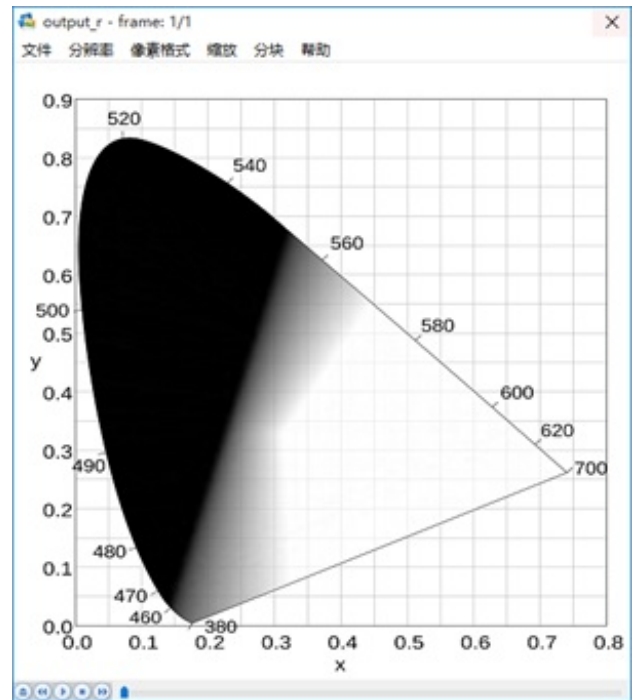
1. int simplest_rgb24_to_bmp(const char *rgb24path,int width,int height,const char *bmppath){
2.     typedef struct
3.     {
4.         long imageSize;
5.         long blank;
6.         long startPosition;
7.     }BmpHead;
8.     typedef struct

```

```

9.  {
10.    long Length;
11.    long width;
12.    long height;
13.    unsigned short colorPlane;
14.    unsigned short bitColor;
15.    long zipFormat;
16.    long realSize;
17.    long xPels;
18.    long yPels;
19.    long colorUse;
20.    long colorImportant;
21. }InfoHead;
22. int i=0,j=0;
23. BmpHead m_BMPHeader={0};
24. InfoHead m_BMPInfoHeader={0};
25. char bfType[2]='B','M';

```



```

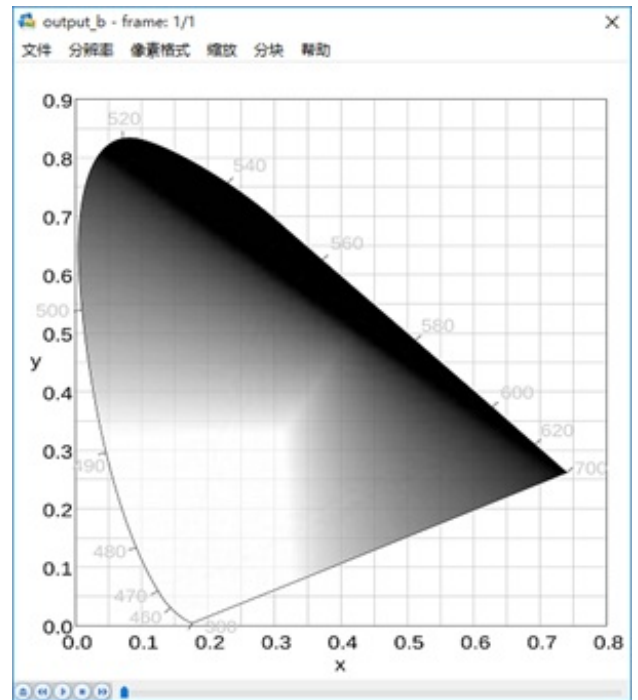
26. int header_size=sizeof(bfType)+sizeof(BmpHead)+sizeof(InfoHead);
27. unsigned char *rgb24_buffer=NULL;
28. FILE *fp_rgb24=NULL,*fp_bmp=NULL;
29. if((fp_rgb24=fopen(rgb24path,"rb"))==NULL){
30.     printf("Error: Cannot open input RGB24 file.\n");
31.     return -1;
32. }
33. if((fp_bmp=fopen(bmppath,"wb"))==NULL){
34.     printf("Error: Cannot open output BMP file.\n");
35.     return -1;
36. }

```

```

37.  rgb24_buffer=
      (unsigned char *)malloc(width*height*3);
38.  fread(rgb24_buffer,1,width*height*3,fp_rgb24);

```



```

39.  m_BMPHeader.imageSize=3*width*height+header_size;
40.  m_BMPHeader.startPosition=header_size;
41.  m_BMPInfoHeader.Length=sizeof(InfoHead);
42.  m_BMPInfoHeader.width=width;
43.  m_BMPInfoHeader.height=-height;
44.  m_BMPInfoHeader.colorPlane=1;
45.  m_BMPInfoHeader.bitColor=24;
46.  m_BMPInfoHeader.realSize=3*width*height;
47.  fwrite(bfType,1,sizeof(bfType),fp_bmp);
48.  fwrite(&m_BMPHeader,1,sizeof(m_BMPHeader),fp_bmp);
49.  fwrite(&m_BMPInfoHeader,1,sizeof(m_BMPInfoHeader),fp_bmp);
50.  for(j =0;j
51.      for(i=0;i
52.          char temp=rgb24_buffer[(j*width+i)*3+2];
53.          rgb24_buffer[(j*width+i)*3+2]=rgb24_buffer[(j*width+i)*3+0];
54.          rgb24_buffer[(j*width+i)*3+0]=temp;
55.      }
56.  }
57.  fwrite(rgb24_buffer,3*width*height,1,fp_bmp);
58.  fclose(fp_rgb24);
59.  fclose(fp_bmp);
60.  free(rgb24_buffer);
61.  printf("Finish generate %s!\n",bmppath);
62.  return 0;
63.  return 0;

```



64. }

调用上面函数的方法如下所示。

```
1. simplest_rgb24_to_bmp("lena_256x256_rgb24.rgb",256,256,"output_lena.bmp");
```

通过代码可以看出，改程序完成了主要完成了两个工作：

- 1)将RGB数据前面加上文件头。
- 2)将RGB数据中每个像素的“B”和“R”的位置互换。

BMP文件是由BITMAPFILEHEADER、BITMAPINFOHEADER、RGB像素数据共3个部分构成，它的结构如下图所示。



其中前两部分的结构如下所示。在写入BMP文件头的时候给其中的每个字段赋上合适的值就可以了。

```
1. typedef struct tagBITMAPFILEHEADER
2. {
3.     unsigned short int  bfType;
4.     unsigned long      bfSize;
5.     unsigned short int  bfReserverd1;
6.     unsigned short int  bfReserverd2;
7.     unsigned long      bfbfOffBits;
8. }BITMAPFILEHEADER;
9. typedef struct tagBITMAPINFOHEADER
10. {
11.     long biSize;
12.     long biWidth;
13.     long biHeight;
14.     short int  biPlanes;
15.     short int  biBitcount;
16.     short int  biCompression;
17.     long  biSizeImage;
18.     long  biXPelsPermeter;
19.     long  biYPelsPermeter;
20.     long  biClrUsed;
21.     long  biClrImportant;
22. }BITMAPINFOHEADER;
```

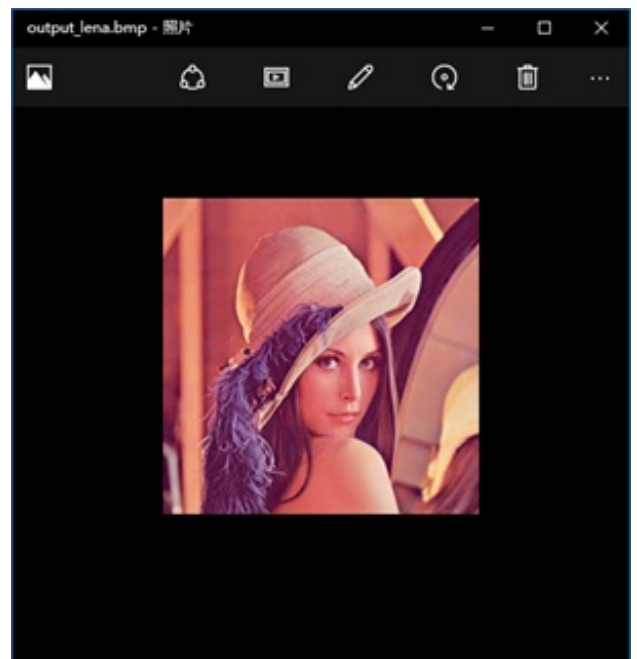
BMP采用的是小端（Little Endian）存储方式。这种存储方式中“RGB24”格式的像素的分量存储的先后顺序为B、G、R。由于RGB24格式存储的顺序是R、G、B，所以需要将“R”和“B”顺序作一个调换再进行存储。

下图为输入的RGB24格式的图像lena\_256x256\_rgb24.rgb。

下图封装为BMP格式后的图像output\_lena.bmp。封装后的图像使用普通的看图软件就可以查看。

### (10)将RGB24格式像素数据转换为YUV420P格式像素数据

本程序中的函数可以将RGB24格式的像素数据转换为YUV420P格式的像素数据。函数的代码如下所示。



```
1. unsigned char clip_value(unsigned char x,unsigned char min_val,unsigned char max_val){
2.     if(x>max_val){
3.         return max_val;
4.     }else if(x
5.         return min_val;
6.     }else{
7.         return x;
8.     }
```

```

9. }
10. bool RGB24_TO_YUV420(unsigned char *RgbBuf,int w,int h,unsigned char *yuvBuf)
11. {
12.     unsigned char*ptrY, *ptrU, *ptrV, *ptrRGB;
13.     memset(yuvBuf,0,w*h*3/2);
14.     ptrY = yuvBuf;
15.     ptrU = yuvBuf + w*h;
16.     ptrV = ptrU + (w*h*1/4);
17.     unsigned char y, u, v, r, g, b;
18.     for (int j = 0; j
19.         ptrRGB = RgbBuf + w*j*3 ;
20.         for (int i = 0;i
21.             r = *(ptrRGB++);
22.             g = *(ptrRGB++);
23.             b = *(ptrRGB++);
24.             y = (unsigned char)(( 66 * r + 129 * g + 25 * b + 128) >> 8) + 16 ;
25.             u = (unsigned char)(( -38 * r - 74 * g + 112 * b + 128) >> 8) + 128 ;
26.             v = (unsigned char)(( 112 * r - 94 * g - 18 * b + 128) >> 8) + 128 ;
27.             *(ptrY++) = clip_value(y,0,255);
28.             if (j%2==0&& i%2 ==0){
29.                 *(ptrU++) =clip_value(u,0,255);
30.             }
31.             else{
32.                 if (i%2==0){
33.                     *(ptrV++) =clip_value(v,0,255);
34.                 }
35.             }
36.         }
37.     }
38.     return true;
39. }
40. int simplest_rgb24_to_yuv420(char *url_in, int w, int h,int num,char *url_out){
41.     FILE *fp=fopen(url_in,"rb+");
42.     FILE *fp1=fopen(url_out,"wb+");
43.     unsigned char *pic_rgb24=(unsigned char *)malloc(w*h*3);
44.     unsigned char *pic_yuv420=(unsigned char *)malloc(w*h*3/2);
45.     for(int i=0;i
46.         fread(pic_rgb24,1,w*h*3,fp);
47.         RGB24_TO_YUV420(pic_rgb24,w,h,pic_yuv420);

```

```

48.     fwrite(pic_yuv420,1,w*h*3/2,fp1);
49. }
50. free(pic_rgb24);
51. free(pic_yuv420);
52. fclose(fp);
53. fclose(fp1);
54. return 0;
55. }

```

调用上面函数的方法如下所示。

```
1. simplest_rgb24_to_yuv420("lena_256x256_rgb24.rgb",256,256,1,"output_lena.yuv");
```

从源代码可以看出，本程序实现了RGB到YUV的转换公式：

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

$$U = -0.147 \cdot R - 0.289 \cdot G + 0.463 \cdot B$$

$$V = 0.615 \cdot R - 0.515 \cdot G - 0.100 \cdot B$$

在转换的过程中有以下几点需要注意：

- 1) RGB24存储方式是Packed，YUV420P存储方式是Packed。
- 2) U，V在水平和垂直方向的取样数是Y的一半

转换前的RGB24格式像素数据lena\_256x256\_rgb24.rgb的内容如下所示。

转换后的YUV420P格式的像素数据output\_lena.yuv的内容如下所示。



## (11)生成RGB24格式的彩条测试图

本程序中的函数可以生成一张RGB24格式的彩条测试图。函数代码如下所示。

```

1. int simplest_rgb24_colorbar(int width, int height,char *url_out){
2.     unsigned char *data=NULL;
3.     int barwidth;
4.     char filename[100]={0};

```



```

5.  FILE *fp=NULL;
6.  int i=0,j=0;
7.  data=(unsigned char *)malloc(width*height*3);
8.  barwidth=width/8;
9.  if((fp=fopen(url_out,"wb+"))==NULL){
10.     printf("Error: Cannot create file!");
11.     return -1;
12. }
13. for(j=0;j
14.     for(i=0;i
15.         int barnum=i/barwidth;
16.         switch(barnum){
17.             case 0:{
18.                 data[(j*width+i)*3+0]=255;
19.                 data[(j*width+i)*3+1]=255;
20.                 data[(j*width+i)*3+2]=255;
21.                 break;
22.             }
23.             case 1:{
24.                 data[(j*width+i)*3+0]=255;
25.                 data[(j*width+i)*3+1]=255;
26.                 data[(j*width+i)*3+2]=0;
27.                 break;
28.             }
29.             case 2:{
30.                 data[(j*width+i)*3+0]=0;
31.                 data[(j*width+i)*3+1]=255;
32.                 data[(j*width+i)*3+2]=255;
33.                 break;
34.             }
35.             case 3:{
36.                 data[(j*width+i)*3+0]=0;
37.                 data[(j*width+i)*3+1]=255;
38.                 data[(j*width+i)*3+2]=0;
39.                 break;
40.             }
41.             case 4:{
42.                 data[(j*width+i)*3+0]=255;
43.                 data[(j*width+i)*3+1]=0;

```



```

44.         data[(j*width+i)*3+2]=255;
45.         break;
46.     }
47. case 5:{
48.     data[(j*width+i)*3+0]=255;
49.     data[(j*width+i)*3+1]=0;
50.     data[(j*width+i)*3+2]=0;
51.     break;
52. }
53. case 6:{
54.     data[(j*width+i)*3+0]=0;
55.     data[(j*width+i)*3+1]=0;
56.     data[(j*width+i)*3+2]=255;
57.     break;
58. }
59. case 7:{
60.     data[(j*width+i)*3+0]=0;
61.     data[(j*width+i)*3+1]=0;
62.     data[(j*width+i)*3+2]=0;
63.     break;
64. }
65. }
66. }
67. }
68. fwrite(data,width*height*3,1,fp);
69. fclose(fp);
70. free(data);
71. return 0;
72. }

```

调用上面函数的方法如下所示。

```
1. simplest_rgb24_colorbar(640, 360,"colorbar_640x360.rgb");
```

从源代码可以看出，本程序循环输出“白黄青绿品红蓝黑”8种颜色的彩条。这8种颜色的彩条的R、G、B取值如下所示。

颜色	(R, G, B)
白	(255, 255, 255)

黄	(255, 255, 0)
青	( 0, 255, 255)
绿	( 0, 255, 0)
品	(255, 0, 255)
红	(255, 0, 0)
蓝	( 0, 0, 255)
黑	( 0, 0, 0)

生成的图像截图如下所示。



## 下载

### Simplest mediadata test

#### 项目主页

SourceForge : <https://sourceforge.net/projects/simplest-mediadata-test/>

Github : [https://github.com/leixiaohua1020/simplest\\_mediadata\\_test](https://github.com/leixiaohua1020/simplest_mediadata_test)

开源中国 : [http://git.oschina.net/leixiaohua1020/simplest\\_mediadata\\_test](http://git.oschina.net/leixiaohua1020/simplest_mediadata_test)

CSDN下载地址 : <http://download.csdn.net/detail/leixiaohua1020/9422409>

本项目包含如下几种视音频数据解析示例：

- (1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
- (2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
- (3)H.264码流分析程序。可以分离并解析NALU。
- (4)AAC码流分析程序。可以分离并解析ADTS帧。
- (5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
- (6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。

**雷霄骅 (Lei Xiaohua)**  
**leixiaohua1020@126.com**  
**<http://blog.csdn.net/leixiaohua1020>**