

最简单的基于FFmpeg的libswscale的示例附件：测试图片生成工具

2014年12月29日 12:07:56 阅读数：9218

最简单的基于FFmpeg的libswscale的示例系列文章列表：

[最简单的基于FFmpeg的libswscale的示例（YUV转RGB）](#)

[最简单的基于FFmpeg的libswscale的示例附件：测试图片生成工具](#)

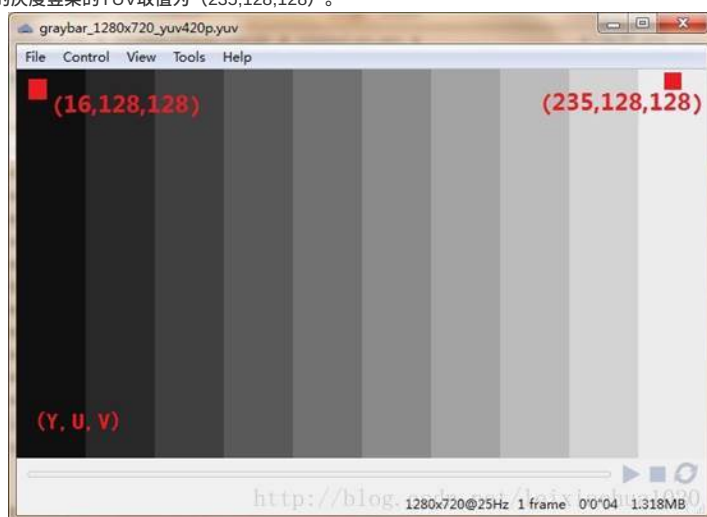
本文记录一个自己写的简单的测试图片生成工具：simplest_pic_gen。该工具可以生成视频测试时候常用的RGB/YUV格式的测试图片。包括灰阶测试图，彩条图，彩色条纹图，RGB渐变彩条图，YUV渐变彩条图，颜色视频等。下面简单介绍一下这些测试图片的生成函数。

这里有一点需要注意：查看生成的图片需要使用RGB/YUV播放器。

灰阶测试图

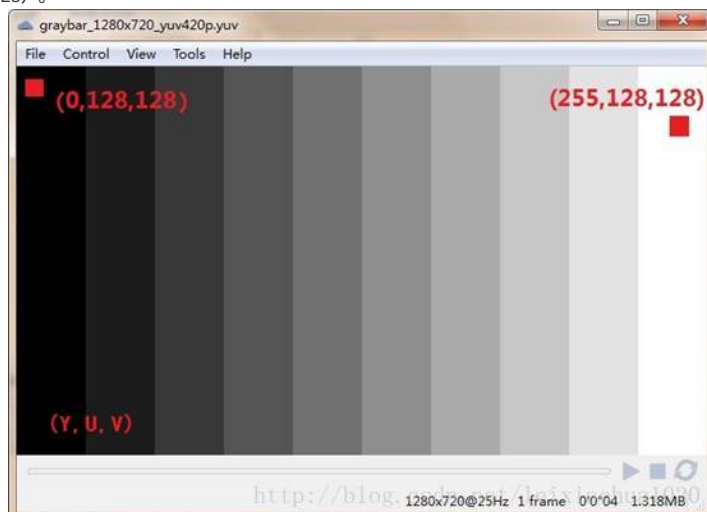
亮度取值为16-235的灰阶测试图

下面这张图是一张灰阶测试图的示例。这张图的分辨率是1280x720，像素格式是YUV420P，亮度的取值范围是16-235，一共包含了10级的灰度。最左边的灰度竖条的YUV取值为（16,128,128），最右边的灰度竖条的YUV取值为（235,128,128）。

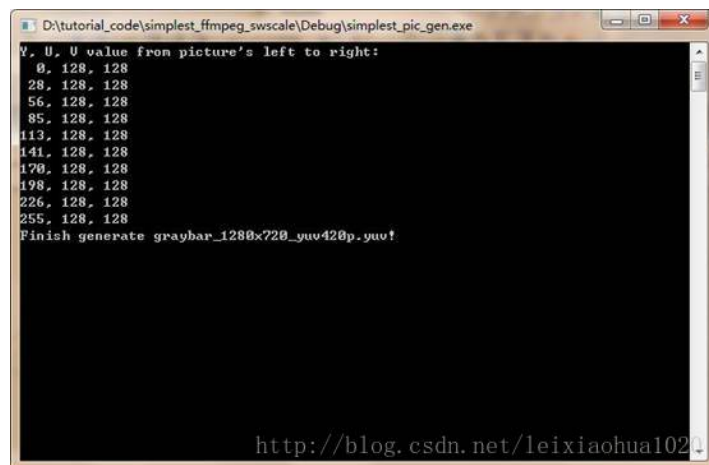


亮度取值为0-255的灰阶测试图

下面这张图的分辨率是1280x720，像素格式是YUV420P，亮度的取值范围是0-255，一共包含了10级的灰度。最左边的灰度竖条的YUV取值为（0,128,128），最右边的灰度竖条的YUV取值为（255,128,128）。



在生成灰度图的同时，程序会打印出每一个灰阶的YUV取值。



函数原型

gen_yuv420p_graybar()是用于生成灰阶测试图的函数，该函数的定义如下。

```
[cpp]
1.  /**
2.   * Generate Picture contains Gray Bar changing from Black to White in YUV420P Format
3.   *
4.   * @param width      the width of picture.
5.   * @param height     the height of picture.
6.   * @param barnum     the number of Bars in the picture.
7.   * @param ymin       the minimum value of luminance.
8.   * @param ymax       the maximum value of luminance.
9.   * @return 0 if finished, -1 if there are errors.
10.  */
11.  int gen_yuv420p_graybar(int width, int height, int barnum, unsigned char ymin, unsigned char ymax);
```

简单解释每个参数的含义：

width：图像宽

height：图像高

barnum：灰阶数量

ymin：亮度最小取值

ymax：亮度最大取值

如果函数成功运行的话，会生成一个名称为“graybar_%dx%d_yuv420p.yuv”的YUV420P格式的文件（其中%d%d代表了图像的宽和高）。

例如，生成分辨率为1280x720的上文中的灰阶图的代码如下。

亮度取值范围为16-235：

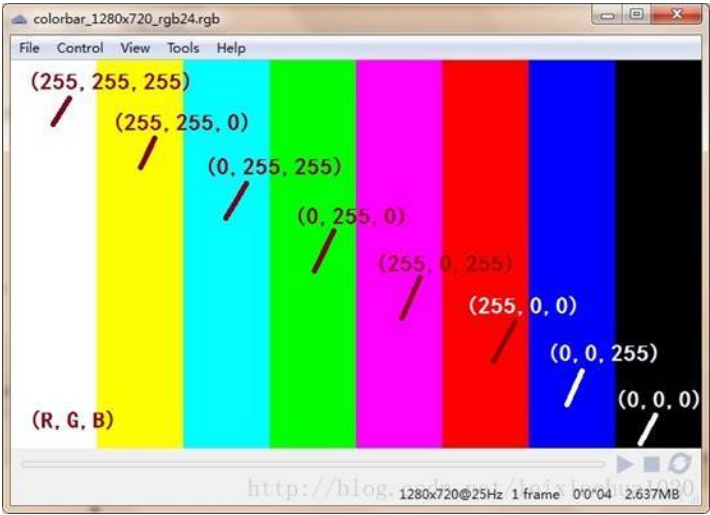
```
[cpp]
1.  gen_yuv420p_graybar(1280, 720, 10, 16, 235);
```

亮度取值范围为0-255

```
[cpp]
1.  gen_yuv420p_graybar(1280, 720, 10, 0, 255);
```

彩条测试图

在电视节目的制作播出及设备维护中，最常用的莫过于彩条信号了。这是由于彩条信号能正确反映出各种彩色的亮度、色调和饱和度，是检验视频通道传输质量最方便的手段。下面这张图是一张彩条测试图的示例。这张图的分辨率是1280x720，像素格式是RGB24，包含了电视系统中常见的“白黄青绿品红蓝黑”彩条。



“白黄青绿品红蓝黑”彩条中每种颜色的RGB取值如下所示：

| 颜色 | (R, G, B)取值 |
|----|-----------------|
| 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) |

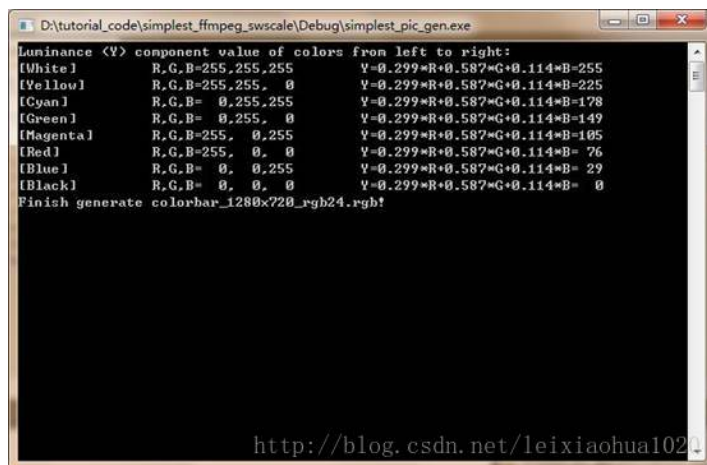
很多人会奇怪，这8个彩条信号的顺序为什么是“白黄青绿品红蓝黑”？其实，它们是按照它们的亮度进行排序的。RGB转换为YUV的过程中，可以通过RGB计算该颜色的亮度。计算的公式如下所示。

$$Y=0.299*R + 0.587*G + 0.114*B$$

把上述8个颜色的R,G,B取值带入上述公式，可以得到每种颜色的亮度取值，如下所示：

| 颜色 | 亮度取值 |
|----|------|
| 白 | 255 |
| 黄 | 225 |
| 青 | 178 |
| 绿 | 149 |
| 品 | 105 |
| 红 | 76 |
| 蓝 | 29 |
| 黑 | 0 |

在生成彩条图像之后，程序会打印出彩条信号的颜色信息，如下图所示。



函数原型

gen_rgb24_colorbar()是用于生成彩条测试图的函数，该函数的原型如下。

```
[cpp]
1. /**
2.  * Generate Picture contains standard Color Bar in RGB24 Format
3.  *
4.  * @param width      the width of picture.
5.  * @param height     the height of picture.
6.  * @return 0 if finished, -1 if there are errors.
7.  */
8. int gen_rgb24_colorbar(int width, int height);
```

简单解释每个参数的含义：

- width：图像宽
- height：图像高

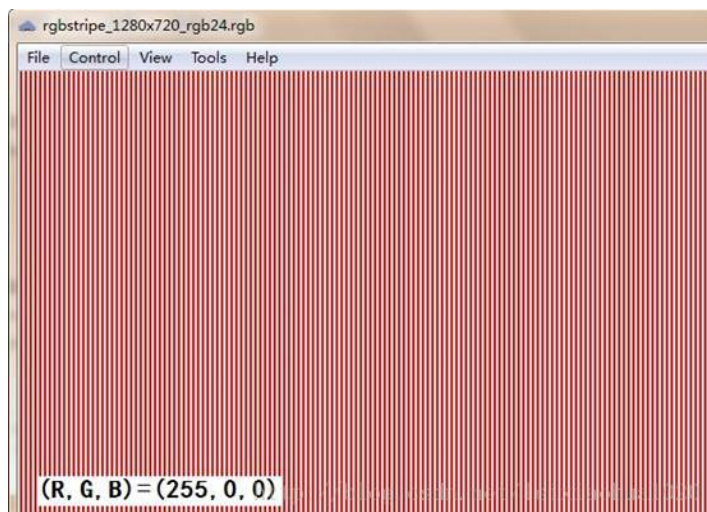
如果函数成功运行的话，会生成一个名称为“colorbar_%dx%d_rgb24.rgb”的RGB24格式的文件（其中%dx%d代表了图像的宽和高）。

例如，生成分辨率为1280x720的上文中的彩条图的代码如下。

```
[cpp]
1. gen_rgb24_colorbar(1280,720);
```

彩色条纹图

条纹图也是常见的一种测试图。下面这张图是一张彩色条纹图的示例。这张图的分辨率是1280x720，像素格式是RGB24，条纹的颜色为红色。其中竖直条纹的宽度为1像素，条纹之间的间隔也是1像素。



函数原型

gen_rgb24_stripe()是用于生成会接测试图的函数，该函数的原型如下。

```

1.  /**
2.   * Generate Picture contains Stripe in RGB24 Format
3.   *
4.   * @param width      the width of picture.
5.   * @param height     the height of picture.
6.   * @param r           Red component of stripe
7.   * @param g           Green component of stripe
8.   * @param b           Blue component of stripe
9.   * @return 0 if finished, -1 if there are errors.
10.  */
11.  int gen_rgb24_stripe(int width, int height,
12.                      unsigned char r, unsigned char g, unsigned char b)

```

简单解释每个参数的含义：

width：图像宽

height：图像高

r：条纹的R分量取值

g：条纹的G分量取值

b：条纹的B分量取值

如果函数成功运行的话，会生成一个名称为"rgbstripe_ %dx%d_ rgb24. rgb"的RGB24格式的文件（其中 %dx%d 代表了图像的宽和高）。

例如，生成分辨率为1280x720的上文中的彩色条纹图的代码如下。

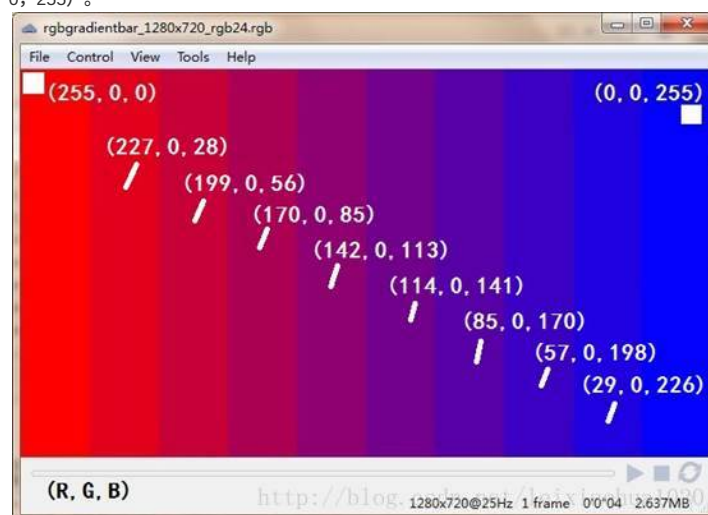
```

1.  gen_rgb24_stripe(1280,720,255,0,0);

```

RGB渐变彩条图

下面这张图是一张RGB渐变彩条图的示例。这张图的分辨率是1280x720，一共包含了10个彩条，像素格式是RGB24，RGB颜色从红色（RGB分别取值为255，0，0）逐渐变换为蓝色（RGB分别取值为0，0，255）。



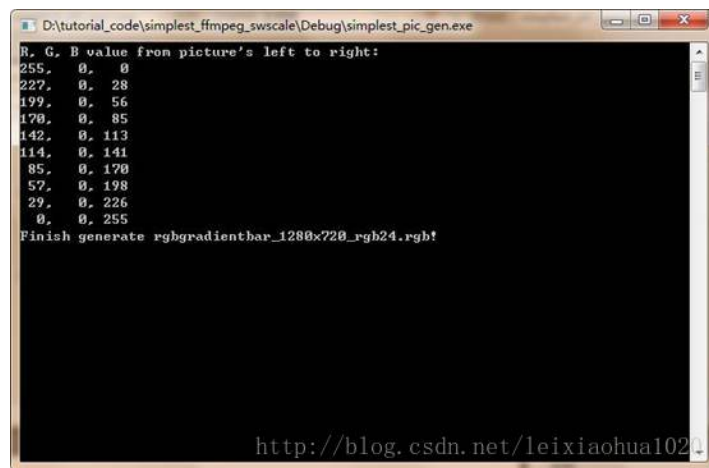
每个彩条的RGB取值如下所列：

```

255, 0, 0
227, 0, 28
199, 0, 56
170, 0, 85
142, 0, 113
114, 0, 141
85, 0, 170
57, 0, 198
29, 0, 226
0, 0, 255

```

在生成渐变彩条图像之后，程序会打印出彩条信号的颜色信息，如下图所示。



函数原型

gen_rgb24_rgbgradient_bar ()是用于生成渐变彩条图的函数，该函数的原型如下。

```
[cpp]
1.  /**
2.   * Generate Picture contains Color Bar Changing from source color
3.   * to destination color in RGB24 Format
4.   *
5.   * @param width      the width of picture.
6.   * @param height     the height of picture.
7.   * @param barnum      the number of Bars in the picture.
8.   * @param src_r       Red component of source color.
9.   * @param src_g       Green component of source color.
10.  * @param src_b       Blue component of source color.
11.  * @param dst_r       Red component of destination color.
12.  * @param dst_g       Green component of destination color.
13.  * @param dst_b       Blue component of destination color.
14.  * @return 0 if finished, -1 if there are errors.
15.  */
16.  int gen_rgb24_rgbgradient_bar(int width, int height,int barnum,
17.      unsigned char src_r,unsigned char src_g,unsigned char src_b,
18.      unsigned char dst_r,unsigned char dst_g,unsigned char dst_b)
```

简单解释每个参数的含义：

- width：图像宽
- height：图像高
- barnum：彩条数量
- src_r：左侧颜色R分量
- src_g：左侧颜色G分量
- src_b：左侧颜色B分量
- dst_r：右侧颜色R分量
- dst_g：右侧颜色G分量
- dst_b：右侧颜色B分量

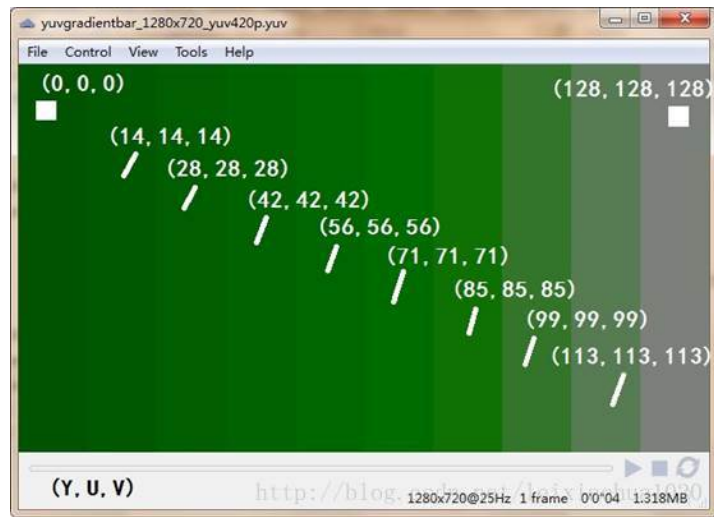
如果函数成功运行的话，会生成一个名称为“rgbgradientbar_%dx%d_rgb24.rgb”的RGB24格式的文件（其中%dx%d代表了图像的宽和高）。

例如，生成分辨率为1280x720的上文中的渐变彩条图的代码如下。

```
[cpp]
1.  gen_rgb24_rgbgradient_bar(1280,720,10,255,0,0,0,0,255);
```

YUV渐变彩条图

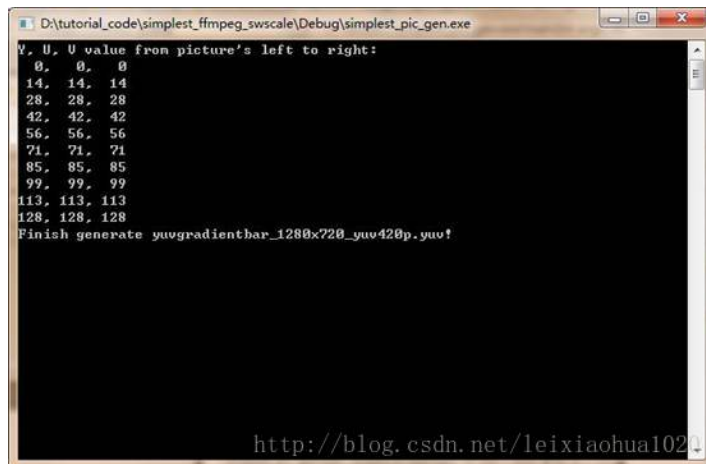
下面这张图是一张YUV渐变彩条图的示例。这张图的分辨率是1280x720，一共包含了10个彩条，像素格式是YUV420P，YUV颜色从绿色（YUV分别取值为0，0，0）逐渐变换为灰色（YUV分别取值为128，128，128）。



每个彩条的YUV取值如下所列：

```
0, 0, 0
14, 14, 14
28, 28, 28
42, 42, 42
56, 56, 56
71, 71, 71
85, 85, 85
99, 99, 99
113, 113, 113
128, 128, 128
```

在生成渐变彩条图像之后，程序会打印出彩条信号的颜色信息，如下图所示。



函数原型

gen_yuv420p_yuvgradient_bar()是用于生成渐变彩条图的函数，该函数的原型如下。

```
1.  /**
2.   * Generate Picture contains Color Bar Changing from source color
3.   * to destination color in YUV420P Format
4.   *
5.   * @param width      the width of picture.
6.   * @param height     the height of picture.
7.   * @param barnum     the number of Bars in the picture.
8.   * @param src_y      Luma component of source color.
9.   * @param src_u      U component of source color.
10.  * @param src_v      V component of source color.
11.  * @param dst_y      Luma component of destination color.
12.  * @param dst_u      U component of destination color.
13.  * @param dst_v      V component of destination color.
14.  * @return 0 if finished, -1 if there are errors.
15.  */
16.  int gen_yuv420p_yuvgradient_bar(int width, int height, int barnum,
17.                                  unsigned char src_y, unsigned char src_u, unsigned char src_v,
18.                                  unsigned char dst_y, unsigned char dst_u, unsigned char dst_v)
```

简单解释每个参数的含义：

- width：图像宽
- height：图像高
- barnum：彩条数量
- src_y：左侧颜色Y分量
- src_u：左侧颜色U分量
- src_v：左侧颜色V分量
- dst_y：右侧颜色Y分量
- dst_u：右侧颜色U分量
- dst_v：右侧颜色V分量

如果函数成功运行的话，会生成一个名称为“yuvgradientbar_ %dx%d_yuv420p.yuv”的YUV420P格式的文件（其中 %dx%d代表了图像的宽和高）。例如，生成分辨率为1280x720的上文中的渐变彩条图的代码如下。

```
[cpp]
1. gen_yuv420p_yuvgradient_bar(1280,720,10,0,0,0,128,128,128);
```

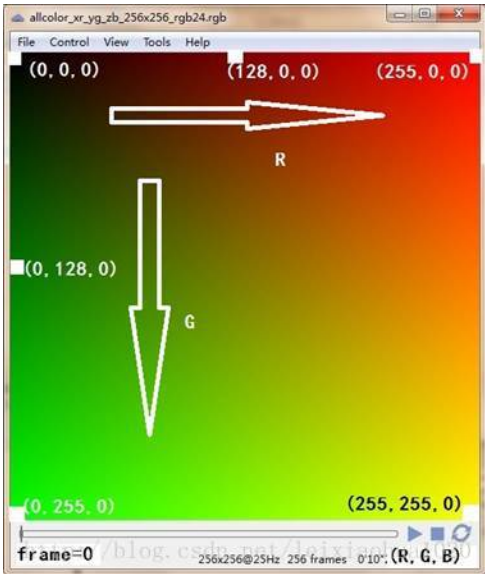
颜色视频

RGB颜色视频

“RGB颜色视频”不是一幅图像，而是一段视频文件。这个视频中包含了RGB24中的所有颜色。通过这个视频，可以了解RGB各个分量对颜色的影响。下面简单记录一下这个视频的规则：

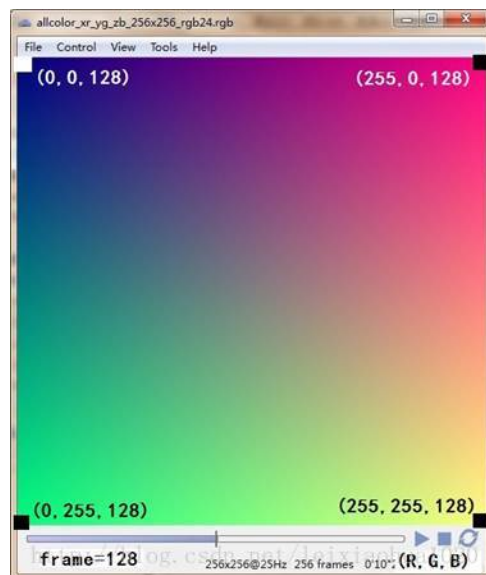
- 视频的宽为256，高为256，视频的帧数为256
- 最左边的像素的R分量取值为0，从左至右每个像素的R分量的取值依次加1
- 最上面的像素的G分量取值为0，从上至下每个像素的G分量的取值依次加1
- 第1帧的所有像素的B分量取值为0，每增加一帧该帧像素的B分量的取值依次加1

所以可以理解为一个坐标系，原点在视频的左上角，X轴对应R分量，Y轴对应G分量，Z轴（时间轴）对应B分量。该视频的第0帧如下图所示。

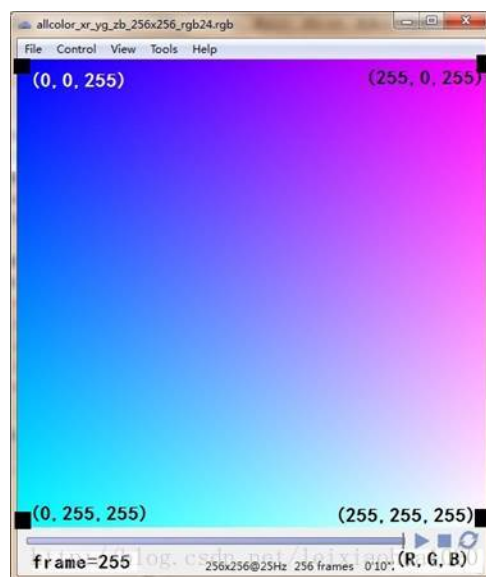


从图中可以看出，左上角为黑色（R，G，B取值0，0，0）；右上角为红色（R，G，B取值0，0，255）；左下角为绿色（R，G，B取值0，255，0）；右下角为黄色（R，G，B取值255，255，0）。

该视频的第128帧如下图所示。



可以看出当蓝色分量增加至128的时候，颜色发生了较大的变化。
该视频的第255帧如下图所示。



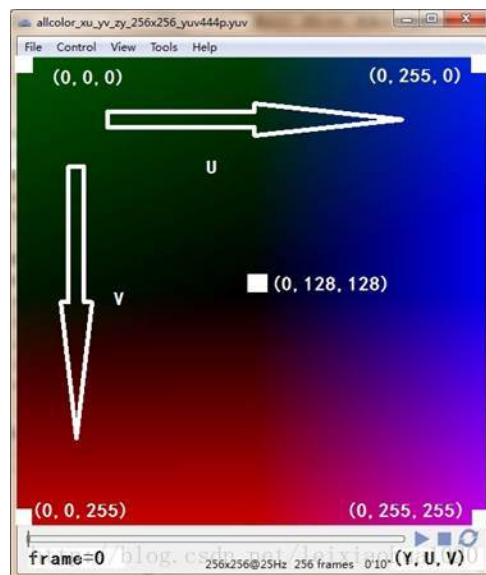
从图中可以看出，左上角为蓝色（R，G，B取值0，0，255）；右上角为品色（R，G，B取值255，0，255）；左下角为青色（R，G，B取值0，255，255）；右下角为白色（R，G，B取值255，255，255）。

YUV颜色视频

“RGB颜色视频”中包含了YUV444中的所有颜色。通过这个视频，可以了解YUV各个分量对颜色的影响。下面简单记录一下这个视频的规则：

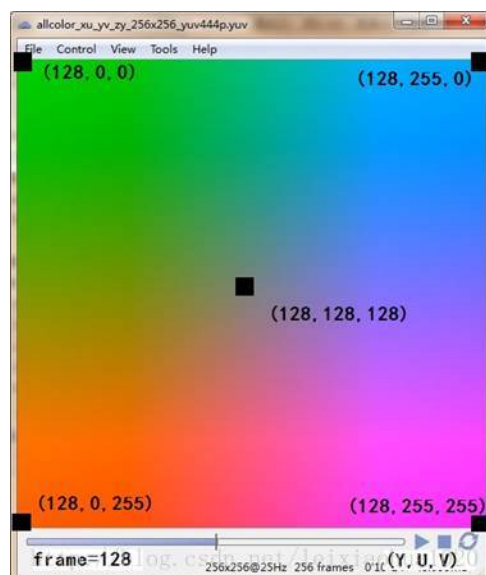
- 视频的宽为256，高为256，视频的帧数为256
- 最左边的像素的U分量取值为0，从左至右每个像素的U分量的取值依次加1
- 最上面的像素的V分量取值为0，从上至下每个像素的V分量的取值依次加1
- 第1帧的所有像素的Y分量取值为0，每增加一帧该帧像素的Y分量的取值依次加1

所以可以理解为一个坐标系，原点在视频的左上角，X轴对应U分量，Y轴对应V分量，Z轴（时间轴）对应Y分量。
该视频的第0帧如下图所示。

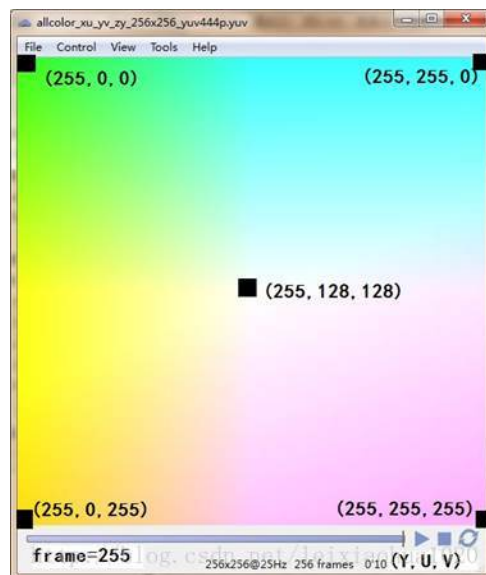


从图中可以看出，左上角颜色偏绿（Y，U，V取值0，0，0）；右上角颜色偏蓝（Y，U，V取值0，0，255）；左下角颜色偏红（Y，U，V取值0，255，0）；右下角颜色偏品色（Y，U，V取值255，255，0）。而正中央是黑色（Y，U，V取值0，128，128）。在这个地方可能很多人会有疑问，认为Y，U，V取值为0，0，0的时候按理说应该是黑色。实际上U，V是加了偏置的分量，而偏置量就是128。所以“纯正”的黑色实际上对应的是Y，U，V取值为0，128，128的颜色。

该视频的第128帧如下图所示。



可以看出随着Y分量的增加，颜色发生了一些变化。
该视频的第255帧如下图所示。



可以看出，尽管Y分量从0增长到255，但是实际上色调变化不大，只是亮度变化很大。这是因为U，V分量存储了色度信息，Y分量存储了亮度信息。

函数原型

gen_allcolor_video()是用于生成渐变彩条图的函数，该函数的原型如下。

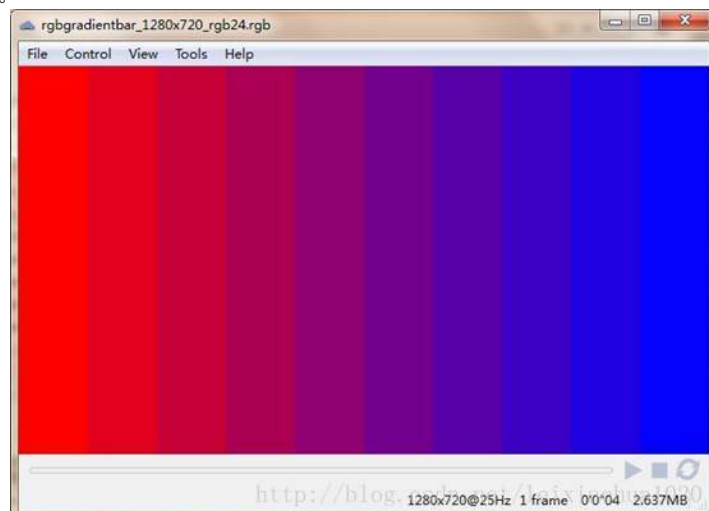
```
[cpp]
1.  /**
2.   * Generate a video in 256x256 and has 256 frames that contains all the colors.
3.   * Each color is shown in 1 pixel. They are mapped as follows:
4.   * In RGB24
5.   * R component's value is increasing with the growth of width (X-axis);
6.   * G component's value is increasing with the growth of height (Y-axis);
7.   * B component's value is increasing with the growth of frame number (Z-axis).
8.   * In YUV444P
9.   * U component's value is increasing with the growth of width (X-axis);
10.  * V component's value is increasing with the growth of height (Y-axis);
11.  * Y component's value is increasing with the growth of frame number (Z-axis).
12.  *
13.  * This function now support to draw YUV444P/RGB24 format pixel.
14.  *
15.  * @return 0 if finished, -1 if there are errors.
16.  */
17.  int gen_allcolor_video();
```

该函数没有参数，直接调用即可生成上述视频。

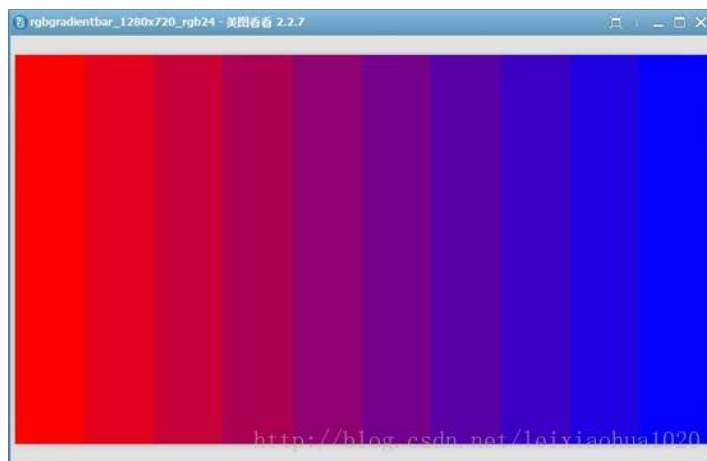
工具函数：RGB24转BMP

本工具除了可以生成测试图片外，还提供了一个简单的工具函数：RGB24转BMP。经过转换后，原本只能用专用的RGB/YUV播放器查看的像素数据，就可以直接拿图片浏览器查看了。

例如输入的RGB24像素数据如下所示。





而输出的BMP图片如下所示。



RGB24转换BMP有以下2个关键点：

- (1) 在RGB数据前面加上文件头
- (2) 把RGB24数据中的“R”和“B”位置互换（因为BMP中的RGB24实际的存储方式是bgrbgrbgr...）。

源代码

```
[cpp]  
1.  /**
2.  * 最简单的测试图片生成工具
3.  * Simplest Pic Gen
4.  *
5.  * 雷霄骅 Lei Xiaohua
6.  * leixiaohua1020@126.com
7.  * 中国传媒大学/数字电视技术
8.  * Communication University of China / Digital TV Technology
9.  * http://blog.csdn.net/leixiaohua1020
10. *
11. * 本程序可以生成多种RGB/YUV格式的测试图像。包括：
12. * 灰阶图 [YUV420P]
13. * 彩条图 [RGB24]
14. * 彩色条纹图 [RGB24]
15. * RGB渐变彩条图 [RGB24]
16. * YUV渐变彩条图 [YUV420P]
17. * 颜色视频 [RGB24][YUV444P]
18. *
19. * This software can generate several picture that used for
20. * test:
21. * Gray Bar Picture [YUV420P]
22. * Color Bar Picture [RGB24]
23. * Color Stripe Picture [RGB24]
24. * RGB Gradient Bar Picture [RGB24]
25. * YUV Gradient Bar Picture [YUV420P]
26. * All Color Video [RGB24][YUV444P]
27. *
28. */
29.
30. #include <stdio.h>
31. #include <malloc.h>
32.
33.
34. /**
35. * Generate Picture contains Stripe in RGB24 Format
36. *
37. * @param width the width of picture.
38. * @param height the height of picture.
39. * @param r Red component of stripe
40. * @param g Green component of stripe
41. * @param b Blue component of stripe
42. * @return 0 if finished, -1 if there are errors.
43. */
44. int gen_rgb24_stripe(int width, int height,
45. unsigned char r,unsigned char g,unsigned char b){
46.
47. unsigned char *data=NULL;
48. char filename[100]={0};
49. FILE *fp=NULL;
50. int i=0,j=0;
51.
52. //Check
53. if(width<=0||height<=0){
54. printf("Error: Width, Height cannot be 0 or negative number!\n");
55. printf("Default Param is used.\n");
56. width=640;
57. height=480;
58. }
59.
```

```

60.     data=(unsigned char *)malloc(width*height*3);
61.
62.     sprintf(filename,"rgbstripe_%dx%d_rgb24.rgb",width,height);
63.     if((fp=fopen(filename,"wb+"))==NULL){
64.         printf("Error: Cannot create file!");
65.         return -1;
66.     }
67.
68.     for(j=0;j<height;j++){
69.         for(i=0;i<width;i++){
70.             if(i%2!=0){
71.                 data[(j*width+i)*3+0]=r;
72.                 data[(j*width+i)*3+1]=g;
73.                 data[(j*width+i)*3+2]=b;
74.             }else{//White
75.                 data[(j*width+i)*3+0]=255;
76.                 data[(j*width+i)*3+1]=255;
77.                 data[(j*width+i)*3+2]=255;
78.             }
79.         }
80.     }
81.     fwrite(data,width*height*3,1,fp);
82.     fclose(fp);
83.     free(data);
84.     printf("Finish generate %s!\n",filename);
85.     return 0;
86. }
87.
88.
89. /**
90.  * Generate Picture contains Gray Bar changing from Black to White in YUV420P Format
91.  *
92.  * @param width      the width of picture.
93.  * @param height     the height of picture.
94.  * @param barnum     the number of Bars in the picture.
95.  * @param ymin       the minimum value of luminance.
96.  * @param ymax       the maximum value of luminance.
97.  * @return 0 if finished, -1 if there are errors.
98.  */
99. int gen_yuv420p_graybar(int width, int height,int barnum,unsigned char ymin,unsigned char ymax){
100.
101.     int barwidth;
102.     float lum_inc;
103.     unsigned char lum_temp;
104.     int uv_width,uv_height;
105.     FILE *fp=NULL;
106.     unsigned char *data_y=NULL;
107.     unsigned char *data_u=NULL;
108.     unsigned char *data_v=NULL;
109.     int t=0,i=0,j=0;
110.     char filename[100]={0};
111.
112.     //Check
113.     if(width<=0||height<=0||barnum<=0){
114.         printf("Error: Width, Height or Bar Number cannot be 0 or negative number!\n");
115.         printf("Default Param is used.\n");
116.         width=640;
117.         height=480;
118.         barnum=10;
119.     }
120.     if(width%barnum!=0){
121.         printf("Warning: Width cannot be divided by Bar Number without remainder!\n");
122.     }
123.     barwidth=width/barnum;
124.     lum_inc=((float)(ymax-ymin))/((float)(barnum-1));
125.     uv_width=width/2;
126.     uv_height=height/2;
127.
128.     data_y=(unsigned char *)malloc(width*height);
129.     data_u=(unsigned char *)malloc(uv_width*uv_height);
130.     data_v=(unsigned char *)malloc(uv_width*uv_height);
131.
132.     sprintf(filename,"graybar_%dx%d_yuv420p.yuv",width,height);
133.     if((fp=fopen(filename,"wb+"))==NULL){
134.         printf("Error: Cannot create file!");
135.         return -1;
136.     }
137.
138.     //Output Info
139.     printf("Y, U, V value from picture's left to right:\n");
140.     for(t=0;t<(width/barwidth);t++){
141.         lum_temp=ymin+(char)(t*lum_inc);
142.         printf("%3d, 128, 128\n",lum_temp);
143.     }
144.     //Gen Data
145.     for(j=0;j<height;j++){
146.         for(i=0;i<width;i++){
147.             t=i/barwidth;
148.             lum_temp=ymin+(char)(t*lum_inc);
149.             data_y[j*width+i]=lum_temp;
150.         }

```

```

151.     }
152.     for(j=0;j<uv_height;j++){
153.         for(i=0;i<uv_width;i++){
154.             data_u[j*uv_width+i]=128;
155.         }
156.     }
157.     for(j=0;j<uv_height;j++){
158.         for(i=0;i<uv_width;i++){
159.             data_v[j*uv_width+i]=128;
160.         }
161.     }
162.     fwrite(data_y,width*height,1,fp);
163.     fwrite(data_u,uv_width*uv_height,1,fp);
164.     fwrite(data_v,uv_width*uv_height,1,fp);
165.     fclose(fp);
166.     free(data_y);
167.     free(data_u);
168.     free(data_v);
169.     printf("Finish generate %s!\n",filename);
170. }
171.
172. /**
173.  * Generate Picture contains standard Color Bar in RGB24 Format
174.  *
175.  * @param width    the width of picture.
176.  * @param height   the height of picture.
177.  * @return 0 if finished, -1 if there are errors.
178.  */
179. int gen_rgb24_colorbar(int width, int height){
180.
181.     unsigned char *data=NULL;
182.     int barwidth;
183.     char filename[100]={0};
184.     FILE *fp=NULL;
185.     int i=0,j=0;
186.     int lum;
187.     float r_coeff=0.299,g_coeff=0.587,b_coeff=0.114;
188.
189.     //Check
190.     if(width<=0||height<=0){
191.         printf("Error: Width, Height cannot be 0 or negative number!\n");
192.         printf("Default Param is used.\n");
193.         width=640;
194.         height=480;
195.     }
196.     if(width%8!=0)
197.         printf("Warning: Width cannot be divided by Bar Number without remainder!\n");
198.
199.     data=(unsigned char *)malloc(width*height*3);
200.     barwidth=width/8;
201.
202.     sprintf(filename,"colorbar %dx%d_rgb24.rgb",width,height);
203.     if((fp=fopen(filename,"wb+"))==NULL){
204.         printf("Error: Cannot create file!");
205.         return -1;
206.     }
207.
208.     printf("Luminance (Y) component value of colors from left to right:\n");
209.     lum=r_coeff*255.0+g_coeff*255.0+b_coeff*255.0;
210.     printf("[White] \tR,G,B=255,255,255\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
211.         r_coeff,g_coeff,b_coeff,lum);
212.     lum=r_coeff*255.0+g_coeff*255.0+b_coeff*0.0;
213.     printf("[Yellow] \tR,G,B=255,255, 0\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
214.         r_coeff,g_coeff,b_coeff,lum);
215.     lum=r_coeff*0.0+g_coeff*255.0+b_coeff*255.0;
216.     printf("[Cyan] \tR,G,B= 0,255,255\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
217.         r_coeff,g_coeff,b_coeff,lum);
218.     lum=r_coeff*0.0+g_coeff*255.0+b_coeff*0.0;
219.     printf("[Green] \tR,G,B= 0,255, 0\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
220.         r_coeff,g_coeff,b_coeff,lum);
221.     lum=r_coeff*255.0+g_coeff*0.0+b_coeff*255.0;
222.     printf("[Magenta]\tR,G,B=255, 0,255\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
223.         r_coeff,g_coeff,b_coeff,lum);
224.     lum=r_coeff*255.0+g_coeff*0.0+b_coeff*0.0;
225.     printf("[Red] \tR,G,B=255, 0, 0\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
226.         r_coeff,g_coeff,b_coeff,lum);
227.     lum=r_coeff*0.0+g_coeff*0.0+b_coeff*255.0;
228.     printf("[Blue] \tR,G,B= 0, 0,255\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
229.         r_coeff,g_coeff,b_coeff,lum);
230.     lum=r_coeff*0.0+g_coeff*0.0+b_coeff*0.0;
231.     printf("[Black] \tR,G,B= 0, 0, 0\t Y=%.3f*R+%.3f*G+%.3f*B=%3d\n",
232.         r_coeff,g_coeff,b_coeff,lum);
233.
234.     for(j=0;j<height;j++){
235.         for(i=0;i<width;i++){
236.             int barnum=i/barwidth;
237.             switch(barnum){
238.                 case 0:{
239.                     data[(j*width+i)*3+0]=255;
240.                     data[(j*width+i)*3+1]=255;
241.                     data[(j*width+i)*3+2]=255;

```

```

242.         break;
243.     }
244.     case 1:{
245.         data[(j*width+i)*3+0]=255;
246.         data[(j*width+i)*3+1]=255;
247.         data[(j*width+i)*3+2]=0;
248.         break;
249.     }
250.     case 2:{
251.         data[(j*width+i)*3+0]=0;
252.         data[(j*width+i)*3+1]=255;
253.         data[(j*width+i)*3+2]=255;
254.         break;
255.     }
256.     case 3:{
257.         data[(j*width+i)*3+0]=0;
258.         data[(j*width+i)*3+1]=255;
259.         data[(j*width+i)*3+2]=0;
260.         break;
261.     }
262.     case 4:{
263.         data[(j*width+i)*3+0]=255;
264.         data[(j*width+i)*3+1]=0;
265.         data[(j*width+i)*3+2]=255;
266.         break;
267.     }
268.     case 5:{
269.         data[(j*width+i)*3+0]=255;
270.         data[(j*width+i)*3+1]=0;
271.         data[(j*width+i)*3+2]=0;
272.         break;
273.     }
274.     case 6:{
275.         data[(j*width+i)*3+0]=0;
276.         data[(j*width+i)*3+1]=0;
277.         data[(j*width+i)*3+2]=255;
278.
279.         break;
280.     }
281.     case 7:{
282.         data[(j*width+i)*3+0]=0;
283.         data[(j*width+i)*3+1]=0;
284.         data[(j*width+i)*3+2]=0;
285.         break;
286.     }
287. }
288.
289.     }
290. }
291. fwrite(data,width*height*3,1,fp);
292. fclose(fp);
293. free(data);
294. printf("Finish generate %s!\n",filename);
295. }
296.
297. /**
298.  * Generate Picture contains Color Bar Changing from source color
299.  * to destination color in RGB24 Format
300.  *
301.  * @param width      the width of picture.
302.  * @param height     the height of picture.
303.  * @param barnum     the number of Bars in the picture.
304.  * @param src_r      Red component of source color.
305.  * @param src_g      Green component of source color.
306.  * @param src_b      Blue component of source color.
307.  * @param dst_r      Red component of destination color.
308.  * @param dst_g      Green component of destination color.
309.  * @param dst_b      Blue component of destination color.
310.  * @return 0 if finished, -1 if there are errors.
311.  */
312. int gen_rgb24_rgbgradient_bar(int width, int height,int barnum,
313. unsigned char src_r,unsigned char src_g,unsigned char src_b,
314. unsigned char dst_r,unsigned char dst_g,unsigned char dst_b){
315.
316.     unsigned char *data=NULL;
317.     int barwidth;
318.     float r_inc,g_inc,b_inc;
319.     unsigned char r_temp,g_temp,b_temp;
320.     char filename[100]={0};
321.     FILE *fp=NULL;
322.     int t=0,i=0,j=0;
323.
324.     //Check
325.     if(width<=0||height<=0||barnum<=0){
326.         printf("Error: Width, Height or Bar Number cannot be 0 or negative number!\n");
327.         printf("Default Param is used.\n");
328.         width=640;
329.         height=480;
330.     }
331.     if(width%barnum!=0)
332.         printf("Warning: Width cannot be divided by Bar Number without remainder!\n");

```

```

333.
334.
335.     data=(unsigned char *)malloc(width*height*3);
336.     barwidth=width/barnum;
337.     r_inc=((float)(dst_r-src_r))/((float)(barnum-1));
338.     g_inc=((float)(dst_g-src_g))/((float)(barnum-1));
339.     b_inc=((float)(dst_b-src_b))/((float)(barnum-1));
340.
341.     sprintf(filename,"rgbgradientbar %dx%d_rgb24.rgb",width,height);
342.     if((fp=fopen(filename,"wb+"))==NULL){
343.         printf("Error: Cannot create file!");
344.         return -1;
345.     }
346.
347.     //Output Info
348.     printf("R, G, B value from picture's left to right:\n");
349.     for(t=0;t<(width/barwidth);t++){
350.         r_temp=src_r+(char)(t*r_inc);
351.         g_temp=src_g+(char)(t*g_inc);
352.         b_temp=src_b+(char)(t*b_inc);
353.         printf("%3d, %3d, %3d\n",r_temp,g_temp,b_temp);
354.     }
355.
356.     for(j=0;j<height;j++){
357.         for(i=0;i<width;i++){
358.             t=i/barwidth;
359.             r_temp=src_r+(char)(t*r_inc);
360.             g_temp=src_g+(char)(t*g_inc);
361.             b_temp=src_b+(char)(t*b_inc);
362.             data[(j*width+i)*3+0]=r_temp;
363.             data[(j*width+i)*3+1]=g_temp;
364.             data[(j*width+i)*3+2]=b_temp;
365.         }
366.     }
367.     fwrite(data,width*height*3,1,fp);
368.     fclose(fp);
369.     free(data);
370.     printf("Finish generate %s!\n",filename);
371.     return 0;
372. }
373.
374. /**
375.  * Generate Picture contains Color Bar Changing from source color
376.  * to destination color in YUV420P Format
377.  *
378.  * @param width      the width of picture.
379.  * @param height     the height of picture.
380.  * @param barnum     the number of Bars in the picture.
381.  * @param src_y      Luma component of source color.
382.  * @param src_u      U component of source color.
383.  * @param src_v      V component of source color.
384.  * @param dst_y      Luma component of destination color.
385.  * @param dst_u      U component of destination color.
386.  * @param dst_v      V component of destination color.
387.  * @return 0 if finished, -1 if there are errors.
388.  */
389. int gen_yuv420p_yuvgradient_bar(int width, int height,int barnum,
390.     unsigned char src_y,unsigned char src_u,unsigned char src_v,
391.     unsigned char dst_y,unsigned char dst_u,unsigned char dst_v){
392.
393.     int uv_width,uv_height;
394.     unsigned char *data_y=NULL;
395.     unsigned char *data_u=NULL;
396.     unsigned char *data_v=NULL;
397.     FILE *fp=NULL;
398.     int barwidth,uv_barwidth;
399.     float y_inc,u_inc,v_inc=0;
400.     unsigned char y_temp,u_temp,v_temp=0;
401.     char filename[100]={0};
402.     int t=0,i=0,j=0;
403.     //Check
404.     if(width<=0||height<=0||barnum<=0){
405.         printf("Error: Width, Height or Bar Number cannot be 0 or negative number!\n");
406.         printf("Default Param is used.\n");
407.         width=640;
408.         height=480;
409.     }
410.     if(width%barnum!=0)
411.         printf("Warning: Width cannot be divided by Bar Number without remainder!\n");
412.
413.     uv_width=width/2;
414.     uv_height=height/2;
415.     data_y=(unsigned char *)malloc(width*height);
416.     data_u=(unsigned char *)malloc(uv_width*uv_height);
417.     data_v=(unsigned char *)malloc(uv_width*uv_height);
418.     barwidth=width/barnum;
419.     uv_barwidth=barwidth/(width/uv_width);
420.     y_inc=((float)(dst_y-src_y))/((float)(barnum-1));
421.     u_inc=((float)(dst_u-src_u))/((float)(barnum-1));
422.     v_inc=((float)(dst_v-src_v))/((float)(barnum-1));
423.

```



```

424.     sprintf(filename, "yuvgradientbar_%dx%d_yuv420p.yuv", width, height);
425.     if((fp=fopen(filename, "wb+"))==NULL){
426.         printf("Error: Cannot create file!");
427.         return -1;
428.     }
429.
430.     //Output Info
431.     printf("Y, U, V value from picture's left to right:\n");
432.     for(t=0; t<(width/barwidth); t++){
433.         y_temp=src_y+(char)(t*y_inc);
434.         u_temp=src_u+(char)(t*u_inc);
435.         v_temp=src_v+(char)(t*v_inc);
436.         printf("%3d, %3d, %3d\n", y_temp, u_temp, v_temp);
437.     }
438.
439.     //Gen Data
440.     for(j=0; j<height; j++){
441.         for(i=0; i<width; i++){
442.             t=i/barwidth;
443.             y_temp=src_y+(char)(t*y_inc);
444.             data_y[j*width+i]=y_temp;
445.         }
446.     }
447.     for(j=0; j<uv_height; j++){
448.         for(i=0; i<uv_width; i++){
449.             t=i/uv_barwidth;
450.             u_temp=src_u+(char)(t*u_inc);
451.             data_u[j*uv_width+i]=u_temp;
452.         }
453.     }
454.     for(j=0; j<uv_height; j++){
455.         for(i=0; i<uv_width; i++){
456.             t=i/uv_barwidth;
457.             v_temp=src_v+(char)(t*v_inc);
458.             data_v[j*uv_width+i]=v_temp;
459.         }
460.     }
461.     fwrite(data_y, width*height, 1, fp);
462.     fwrite(data_u, uv_width*uv_height, 1, fp);
463.     fwrite(data_v, uv_width*uv_height, 1, fp);
464.     fclose(fp);
465.     free(data_y);
466.     free(data_u);
467.     free(data_v);
468.     printf("Finish generate %s!\n", filename);
469.     return 0;
470. }
471.
472. /**
473.  * Convert RGB24 format to BMP format
474.  *
475.  * @param rgb24path    path of input RGB24 file.
476.  * @param bmpfilepath  path of output BMP file
477.  * @param width        the width of picture.
478.  * @param height       the height of picture.
479.  * @return 0 if finished, -1 if there are errors.
480.  */
481. int rgb24_to_bmp(char *rgb24path, char *bmpfilepath, int width, int height)
482. {
483.     typedef struct
484.     {
485.         long imageSize;
486.         long blank;
487.         long startPosition;
488.     } BmpHead;
489.
490.     typedef struct
491.     {
492.         long Length;
493.         long width;
494.         long height;
495.         unsigned short colorPlane;
496.         unsigned short bitColor;
497.         long zipFormat;
498.         long realSize;
499.         long xPels;
500.         long yPels;
501.         long colorUse;
502.         long colorImportant;
503.     } InfoHead;
504.
505.     int i=0, j=0;
506.     BmpHead m_BMPHeader={0};
507.     InfoHead m_BMPInfoHeader={0};
508.     char bfType[2]={'B', 'M'};
509.     int header_size=sizeof(bfType)+sizeof(BmpHead)+sizeof(InfoHead);
510.     unsigned char *rgb24_buffer=NULL;
511.     FILE *fp_rgb24=NULL, *fp_bmp=NULL;
512.
513.     if((fp_rgb24=fopen(rgb24path, "rb"))==NULL){
514.         printf("Error: Cannot open input RGB24 file.\n");
515.         return -1;

```

```

515.         return -1;
516.     }
517.     if((fp_bmp=fopen(bmppath,"wb"))==NULL){
518.         printf("Error: Cannot open output BMP file.\n");
519.         return -1;
520.     }
521.
522.     rgb24_buffer=(unsigned char *)malloc(width*height*3);
523.     fread(rgb24_buffer,1,width*height*3,fp_rgb24);
524.
525.     m_BMPHeader.imageSize=3*width*height+header_size;
526.     m_BMPHeader.startPosition=header_size;
527.
528.     m_BMPInfoHeader.Length=sizeof(InfoHead);
529.     m_BMPInfoHeader.width=width;
530.     //BMP storage pixel data in opposite direction of Y-axis (from bottom to top).
531.     m_BMPInfoHeader.height=-height;
532.     m_BMPInfoHeader.colorPlane=1;
533.     m_BMPInfoHeader.bitColor=24;
534.     m_BMPInfoHeader.realSize=3*width*height;
535.
536.     fwrite(bfType,1,sizeof(bfType),fp_bmp);
537.     fwrite(&m_BMPHeader,1,sizeof(m_BMPHeader),fp_bmp);
538.     fwrite(&m_BMPInfoHeader,1,sizeof(m_BMPInfoHeader),fp_bmp);
539.
540.     //BMP save R1|G1|B1,R2|G2|B2 as B1|G1|R1,B2|G2|R2
541.     //It saves pixel data in Little Endian
542.     //So we change 'R' and 'B'
543.     for(j =0;j<height;j++){
544.         for(i=0;i<width;i++){
545.             char temp=rgb24_buffer[(j*width+i)*3+2];
546.             rgb24_buffer[(j*width+i)*3+2]=rgb24_buffer[(j*width+i)*3+0];
547.             rgb24_buffer[(j*width+i)*3+0]=temp;
548.         }
549.     }
550.     fwrite(rgb24_buffer,3*width*height,1,fp_bmp);
551.     fclose(fp_rgb24);
552.     fclose(fp_bmp);
553.     free(rgb24_buffer);
554.     printf("Finish generate %s!\n",bmppath);
555.     return 0;
556. }
557.
558.
559.
560.
561. /**
562.  * Generate a video in 256x256 and has 256 frames that contains all the colors.
563.  * Each color is shown in 1 pixel. They are mapped as follows:
564.  * In RGB24
565.  * R component's value is increasing with the growth of width (X-axis);
566.  * G component's value is increasing with the growth of height (Y-axis);
567.  * B component's value is increasing with the growth of frame number (Z-axis).
568.  * In YUV444P
569.  * U component's value is increasing with the growth of width (X-axis);
570.  * V component's value is increasing with the growth of height (Y-axis);
571.  * Y component's value is increasing with the growth of frame number (Z-axis).
572.  *
573.  * This function now support to draw YUV444P/RGB24 format pixel.
574.  *
575.  * @return 0 if finished, -1 if there are errors.
576.  */
577. int gen_allcolor_video(){
578.
579.     unsigned char *data=NULL;
580.     char filename[100]={0};
581.     FILE *fp=NULL;
582.     int width=256,height=256,frames=256;
583.     int i=0,j=0,k=0;
584.
585.     //From left to right (width, X-axis),R increasing from 0 to255
586.     //From Top to bottom (height, Y-axis),G increasing from 0 to255
587.     //From 0 to 255 frames (time, Z-axis),B increasing from 0 to255
588.     data=(unsigned char *)malloc(width*height*3);
589.     sprintf(filename,"allcolor_xr_yg_zb_%dx%d_rgb24.rgb",width,height);
590.     if((fp=fopen(filename,"wb+"))==NULL){
591.         printf("Error: Cannot create file!");
592.         return -1;
593.     }
594.     for(k=0;k<frames;k++){
595.         for(j=0;j<height;j++){
596.             for(i=0;i<width;i++){
597.                 data[(j*width+i)*3+0]=i;
598.                 data[(j*width+i)*3+1]=j;
599.                 data[(j*width+i)*3+2]=k;
600.             }
601.         }
602.         fwrite(data,width*height*3,1,fp);
603.         printf("Finish generate frame %d!\n",k);
604.     }
605.     fclose(fp);
606.     free(data);

```

```

607.     printf("Finish generate %s!\n", filename);
608.
609.     //From left to right (width, X-axis),U increasing from 0 to255
610.     //From Top to bottom (height, Y-axis),V increasing from 0 to255
611.     //From 0 to 255 frames (time, Z-axis),Y increasing from 0 to255
612.     data=(unsigned char *)malloc(width*height);
613.     sprintf(filename, "allcolor_xu_yv_zy_%dx%d_yuv444p.yuv", width, height);
614.     if((fp=fopen(filename, "wb+"))==NULL){
615.         printf("Error: Cannot create file!");
616.         return -1;
617.     }
618.     for(k=0; k<frames; k++){
619.         for(j=0; j<height; j++){//Y
620.             for(i=0; i<width; i++){
621.                 data[j*width+i]=k;
622.             }
623.         }
624.         fwrite(data, width*height, 1, fp);
625.         for(j=0; j<height; j++){//U
626.             for(i=0; i<width; i++){
627.                 data[j*width+i]=i;
628.             }
629.         }
630.         fwrite(data, width*height, 1, fp);
631.         for(j=0; j<height; j++){//V
632.             for(i=0; i<width; i++){
633.                 data[j*width+i]=j;
634.             }
635.         }
636.         fwrite(data, width*height, 1, fp);
637.         printf("Finish generate frame %d!\n", k);
638.     }
639.     fclose(fp);
640.     free(data);
641.     printf("Finish generate %s!\n", filename);
642.
643.     return 0;
644. }
645.
646.
647.
648.
649.
650. int main(int argc, char* argv[])
651. {
652.     //All picture's resolution is 1280x720
653.     //Gray Bar, from 16 to 235
654.     gen_yuv420p_graybar(1280, 720, 10, 16, 235);
655.     //Color Bar
656.     gen_rgb24_colorbar(1280, 720);
657.     //10 bars, RGB changed from 255,0,0 to 0,0,255
658.     gen_rgb24_rgbgradient_bar(1280, 720, 10, 255, 0, 0, 0, 255);
659.     //10 bars, RGB changed from 0,0,0 to 128,128,128
660.     gen_yuv420p_yuvgradient_bar(1280, 720, 10, 0, 0, 0, 128, 128, 128);
661.     //RGB24 to BMP
662.     rgb24_to_bmp("colorbar_1280x720_rgb24.rgb", "colorbar_1280x720_rgb24.bmp", 1280, 720);
663.     //Red stripe
664.     gen_rgb24_stripe(1280, 720, 255, 0, 0);
665.     //Gen color video
666.     gen_allcolor_video();
667.     return 0;
668. }

```

运行结果

程序运行完后，会生成上文中叙述的几种测试图。

下载

Simplest FFmpeg Swscale

项目主页

SourceForge : <https://sourceforge.net/projects/simplestffmpegswscale/>

Github : https://github.com/leixiaohua1020/simplest_ffmpeg_swscale

开源中国 : http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_swscale

CDSN下载地址 : <http://download.csdn.net/detail/leixiaohua1020/8292175>

本教程是最简单的基于FFmpeg的libswscale进行像素处理的教程。它包含了两个工程：
simplest_ffmpeg_swscale: 最简单的libswscale的教程。
simplest_pic_gen: 生成各种测试图片的工具。

更新-1.1 (2015.2.13)=====

这次考虑到了跨平台的要求，调整了源代码，保证了该项目代码可以在个平台上编译通过。

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/8445671>

SourceForge上已经更新。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/42135539>

文章标签：[测试图](#) [灰度图](#) [彩条信号](#) [渐变图](#) [颜色](#)

个人分类：[我的开源项目](#) [FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由[spyyg](#)生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com