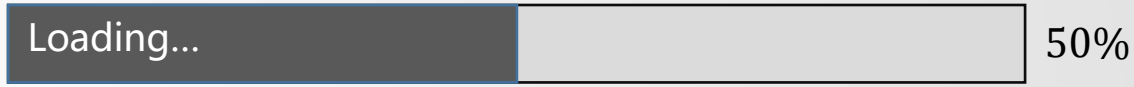
# 图片转ASCII字符画程序分享

汇报人：李牧原 王睿思 周心怡 周靖淇

# 千里之行，始于Array



# 1.array类

int* data = NULL;//底层数组
int index = 0;//标记当前元素的位置
int shape[16] = { 0 };//记录当前数组的维度信息，我们reshape就只用改改这里的数据就好了
int axisNum = 0;//记录原数组的维数
int nowAxis = 0;//记录当前数组的维数
bool check = true;//这个是我自己添加的，记录是否需要删除某个数组，一会析构函数会用到

# 2.[]重载

这个相对简单
1，可以再函数里定义一个对象
2，让它的data指针指向原数组的data
3，将它的nowAxis减1
4，再将shape里的信息修改为低一维度
5，最后将形参index的值赋给它，再返回，就完成了

**举个栗子：**

**a(2,3,3),存的是1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,**

**想访问a[1][2][2]**

**a[1]:**

**int* data =**

**NULL;1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,**

**int index = 1*3*3;**

**int shape[16] = { 3,3 };**

**int axisNum = 3;**

**int nowAxis = 2;**

**bool check = true;**

**a[1][2]:**

**int* data =**

**NULL;1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,**

**int index = 1*3*3+2*3;**

**int shape[16] = { 3 };**

**int axisNum = 3;**

**int nowAxis = 1;**

**bool check = true;**

**a[1][2][2]:**

**int* data =**

**NULL;1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,**

**int index = 1*3*3+2*3+2;**

**int shape[16] = { 1 };**

**int axisNum = 3;**

**int nowAxis = 0;**

**bool check = true;**

**18**

# 3.reshape

这个也简单，比如Array T(198),shape[0]是198，你把它reshape成T(2,33,3),其实只用吧shape改成shape[0]是2，shape[1]是33，shape[2]是3，就好了。
（别的方式记录维度信息也可行）

# 3.析构函数（贼坑）

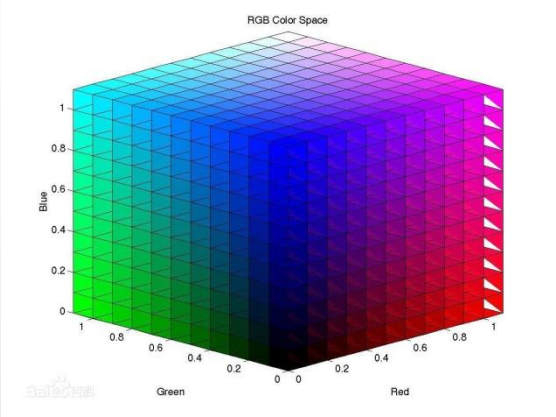
这里真的一不小心就把原数组给删了导致系统报错。
我这里用了两种方法判断是否需要删除。
第一是判断axisNum与nowAxis是否相同，不同的是子数组，就不删。
第二，用成员check标记

# 灰度转字符画

# ·灰度转字符画

概念简介

**RGB彩色图像**中，一种彩色由R（红色），G（绿色），B（蓝色）三原色按比例混合而成。

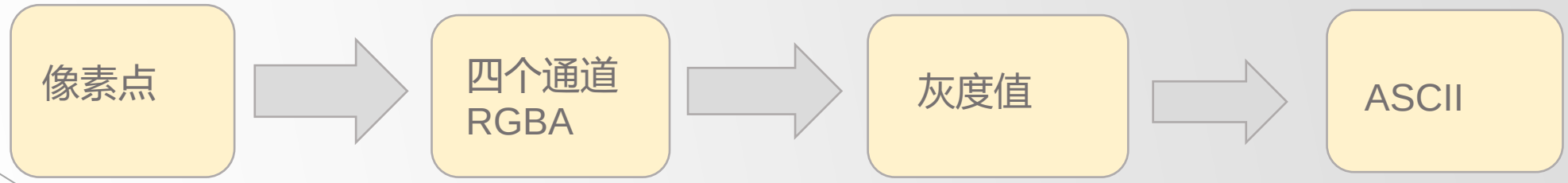



**灰度图像**是用不同饱和度的黑色来表示每个图像点，比如用8位 0-255数　字表示“灰色”程度，每个像素点只需要一个灰度值

# ·灰度转字符画

思路

彩色图片的每个像素点会有四个通道，四个通道值作为一个像素点的信息（RGBA），通过下面的算法转化为单通道的灰度值。
字符越复杂越能形容深色。
最后匹配一个灰度值接近的字符给这个像素点。



# ·灰度转字符画

像素点提取RGB［源已给出，不需要自己写］

理解：
将图片视作多个像素点组成
（此处可考虑合并像素点以缩小图片）
像素点的RGB提取
1、x*y*4 = 每个像素点的信息数量［初源已给出］
2、将像素点信息（RGB）放入ｄａｔａ中

```
BYTE *data;
data = new BYTE[x * y * 4];
memset(data, 0, x * y * 4);
```

```
UINT stride = x * 4;
hr = m_pConvertedSourceBitmap->CopyPixels(nullptr, stride, x * y * 4, data);
```

# ·灰度转字符画

## 实现的关键点

RGB转灰度值（Gray）
降低误差：

一个心理学公式：Grey = 0.299*R + 0.587*G + 0.114*B
∵整数运算又比浮点数运算快得多
∴将浮点数运算转化为整数运算：
Gray = (R * 299 + G * 587 + B * 114) / 1000
∵要减小整数截断带来的误差
∴四舍五入
Gray = (R * 299 + G * 587 + B * 114+500) / 1000

```
static Array temp(x, y);
int *gray = new int[x*y];
int ggay = 0;
int j = 0;
//循环所有的像素点
for (DWORD i = 0; i < x * y * 4; i += 4)
{
    //计算灰度值,一组数据算一次

    gray[j] = ((int)data[i] * 299 + (int)data[i + 1] * 587 + (int)data[i + 2] * 114 + 500) / 1000;


    j++;


}

long int count = 0;
for (int x1 = 0; x1 < x; x1++)
{
    for (int y1 = 0; y1 < y; y1++)
    {
        temp[x1][y1] = gray[count];
        count++;
    }
}



_out = data; _x = x; _y = y;
```

# ·灰度转字符画

RGB转灰度值（Gray）·进一步优化［拓展］

∴位操作快于整数除法
∴移位操作加速遍历

处理（以十位精度为例，其他精度同理）

Gray = 0.299*R + 0.587*G + 0.114*B
Gray = （1024*299*R+ 1024*587*G + 1024*114*B）÷（1024*1000）
Gray = （306176*R+601088*G + 116736*B）÷（1024*1000）
Gray = （306.176*R+601.088*G + 116.736*B）÷（1024）
Gray = （306*R+601*G + 116*B）÷（1024）//截断误差
Gray = （306*R+601*G + 116*B） >> 10;

此时最大误差：

(0.176*255+0.088*255 + 0.736*255) ÷1024 = 255÷1024=0.249

可能会导致1个灰度值的波动

**思考：如果用移位操作还可以怎么降低误差?**

（提示：想办法保留小数部分作用）

PS：感兴趣的小伙伴可以私下试试精度多少的呈现效果最佳~

# ·灰度转字符画 

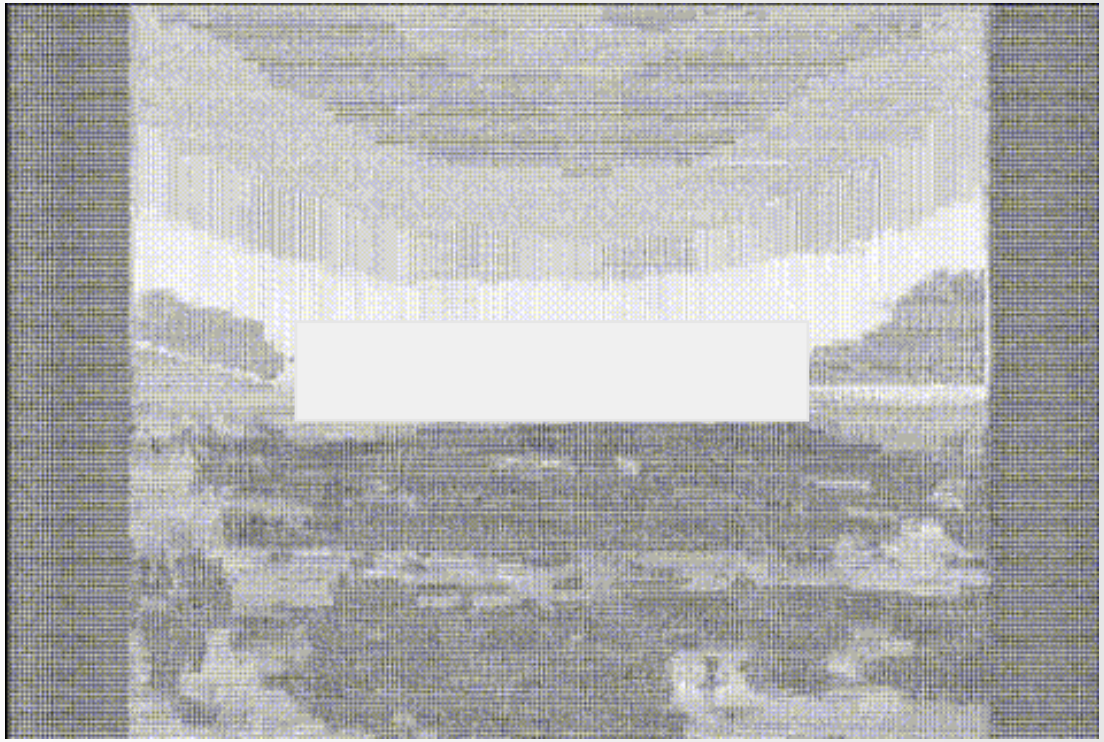
灰度图转ASCII字符图［源已给出］

灰度值转ASCII码

本次作业不需要我们自己筛选出对应灰度值的ASCII码
原ｒｅｓｏｕｒｃｅｓ已经给出１５个等级的灰度对应的ASCII码

```
char ascll[] = { 'M', 'N', 'H', 'Q', '$', 'O', 'C', '?', '7', '>', '!', ':', '-', ';', '.' };
```

接下来让我们进入 **动画** 时代



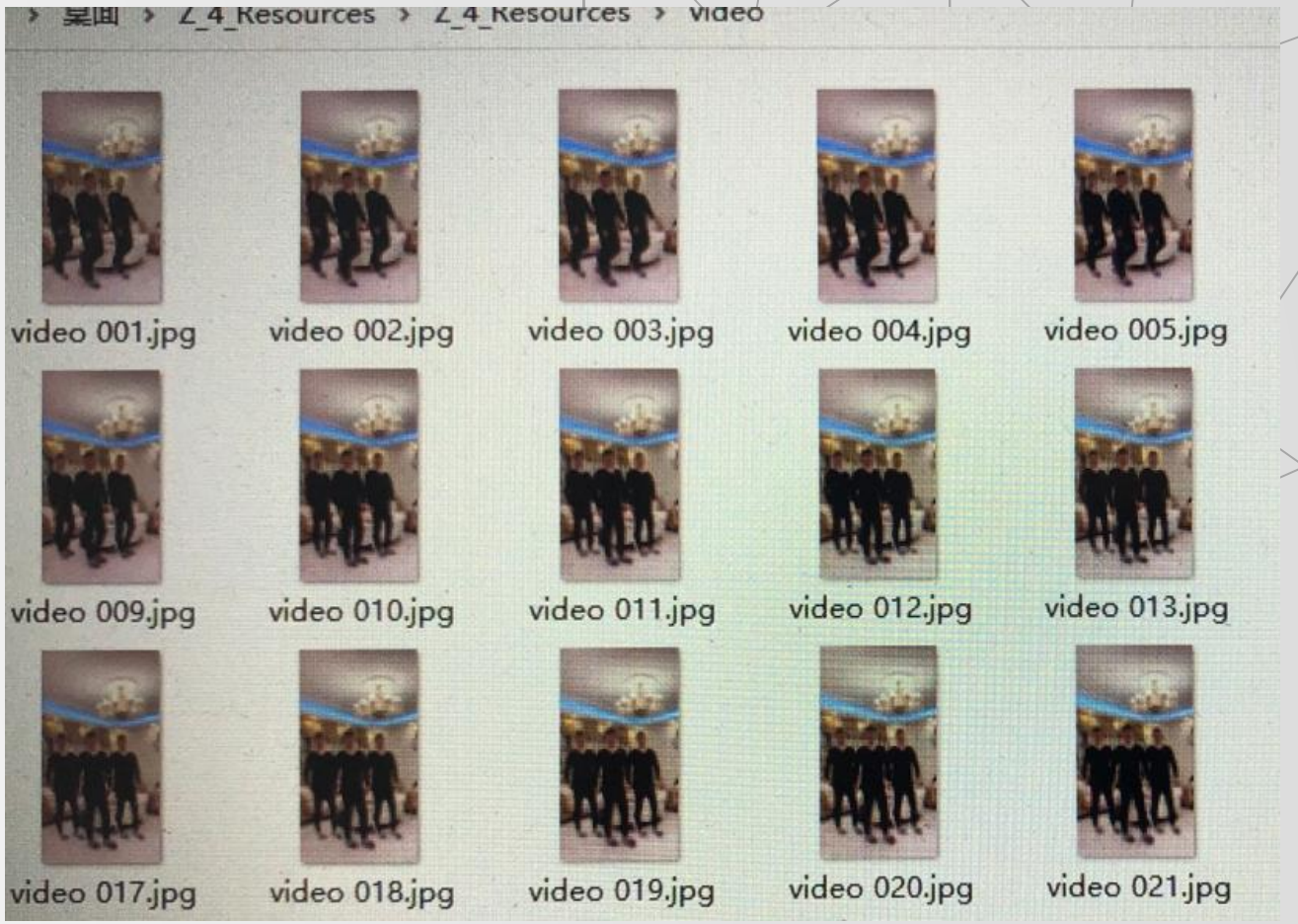
# 字符画之视频实现

Loading...

100%

现在我们已经成功的实现了将图片转化成字符画。
那么我们应该怎么做，才能将单幅的画连成视频呢？

视频的原理：连续的图像变化每秒超过24帧（frame）画面以上时，根据视觉暂留原理，人眼无法辨别单幅的静态画面；看上去是平滑连续的视觉效果，这样连续的画面叫做视频。



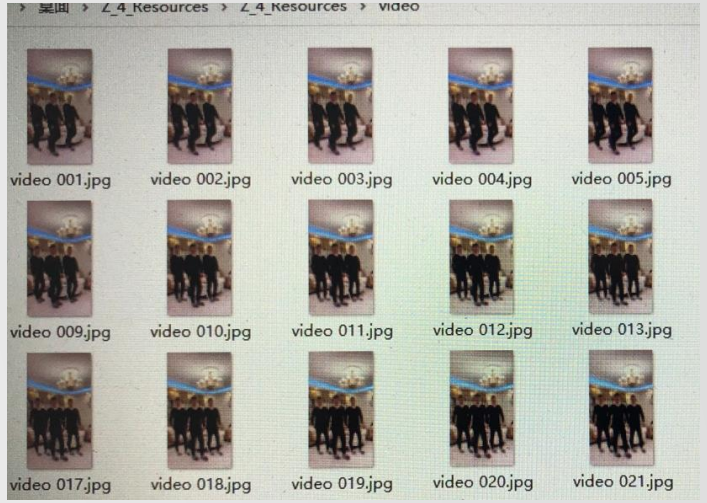

所以每0.4秒就应该换一张图片。

# 第一步：将视频转换为jpg格式

这里可以直接到b站下载画质和帧数最低的视频。这种视频。本来就是24帧。画质也还算合适。不需要我们过多的调整。

然后我们把视频丢到如图所示的这个软件，就可以把它变成一系列的JPG图片。

视频

我们可以发现，这些图片的名称是很有规律的。


```
char picna[30] = { 'v','i','d','e','o','\\','\\','v','i','d','e','o',' ',
    i/100+'0',(i%100)/10+'0',(i % 10) + '0','.','j','p','g',0,0 };
```

于是我们就可以通过这样的一个数组，再利用循环，将图片一个一个的读取出来，然后连接成视频。

```
Goutput();
//playv();
return 0;   已用时间 <= 1,298ms
```

但是如果是一张一张的读取再显示，每张图片都要一秒多，所以这样显然不行。

# 第二步 将视频转化为字符画，再用txt文件储存

```
char txtma[30]= { 't','x','t','p','i','c','\\','\\',
    i / 100 + '0',(i % 100) / 10 + '0',(i % 10) + '0','.','t','x','t',0,0 };
```

为了缩短播放每张图片所需要的时间，我们可以先将每张图片都转化成字符画，然后用txt文件进行储存，播放的时候再一张一张读取出来。具体命名的方式可以参考之前的读取图片。

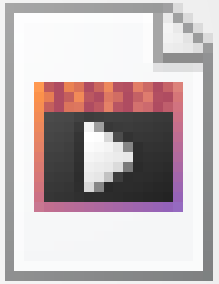
```
char buf[398][15000] = { 0 };
```

他可以先将所有的txt文件都读取出来。储存在这样一个二维数组里。进一步缩短时间。

# 第三步 播放视频

现在我们已经将所有的数据都读到一个二维数组里。
问题是用何种方式将他们给播放出来呢?

我想到的第一个方法是直接夸cout。

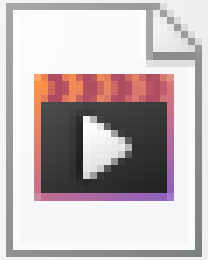


1080p.mov

肉眼可见的卡

所以我们要充分利用FastPrinter这个头文件里面的函数。
这里我使用的是void FastPrinter::setData函数
这个函数可以在指定大小的控制台上输出字符
具体的使用方法可以参考demo里的示例

总之这个函数它真的是 **非 常 快**。



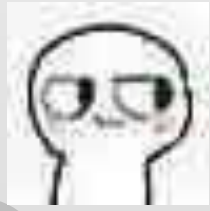
1080p(1).mov

**纵享丝滑！**

# 接下来让我们进入彩色时代！

# 为了使我们的分享变得有特色
# 于是开始了研究**彩色**的漫漫长路...


120%



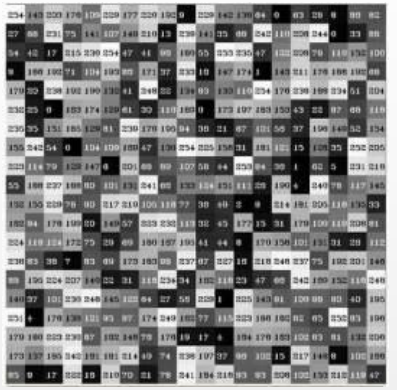
# 灰度值、RGB值、像素值的关系

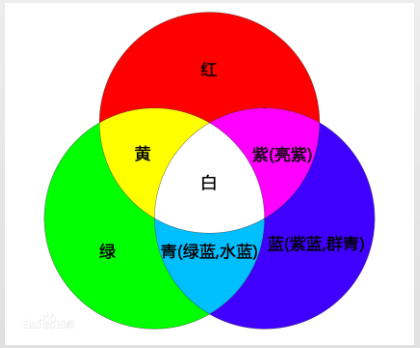
- 灰度也可以认为是亮度，简单来说就是色彩的深浅程度
- 灰度值范围一般从0到255,白色为255,黑色为0

- RGB色彩模式是通过对红(R)、绿(G)、蓝(B)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色的

- 像素值是原稿图像被数字化时由计算机赋予的值，它代表了原稿某一小方块的平均亮度信息

灰度就是没有色彩，RGB色彩分量全部相等
对于灰度图像来说灰度值就是像素值





# 如何实现彩色字符画

- 首先想到的当然是 


- 然而 **本次作业不允许使用外部库完成，**


- 只能先从控制台入手，于是找到了下面两个函数
- system(“color xx”)（xx为两个十六进制数字)
- setConsoleTextAttribute()函数
- （简单试验后发现控制台的颜色总是和语句运行之后的最后一个system('color xx')的颜色修改的命令保持一致，因此无法输出不同的色彩。）

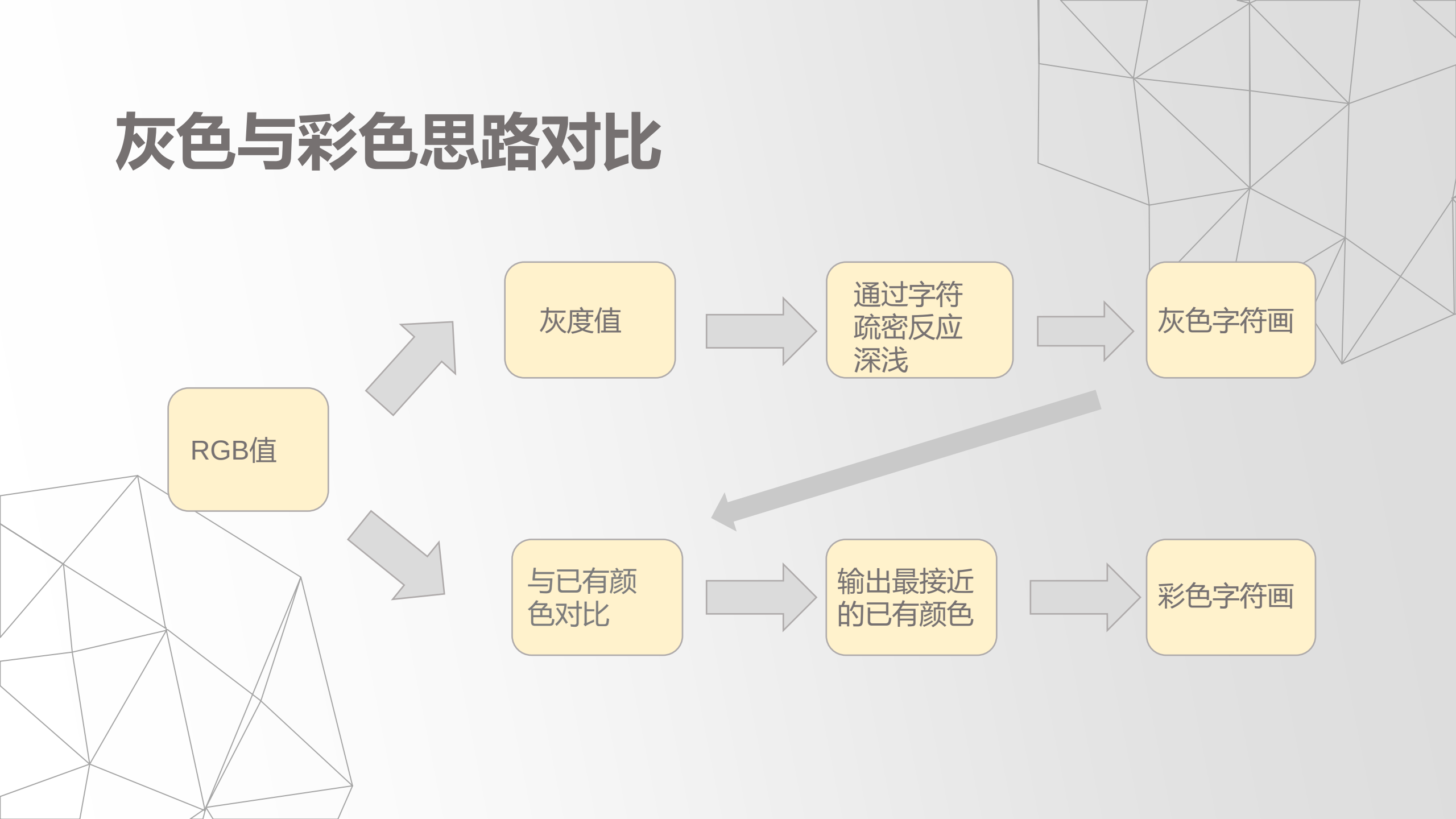
# ◆SetConsoleTextAttribute函数√

- ◆ BOOL SetConsoleTextAttribute(HANDLE hConsoleOutput, WORD wAttributes);
- ◆ 第一个参数标准输出句柄STD_OUTPUT_HANDLE
- ◆ 第二个参数控制颜色

```
void COLOR_PRINT(const char s, int color)
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(handle, color);
    cout<<'s';
    SetConsoleTextAttribute(handle, 7);
}
```

0 = 黑色 8 = 灰色
1 = 蓝色 9 = 淡蓝色
2 = 绿色 10 = 淡绿色
3 = 浅绿色 11 = 淡浅绿色
4 = 红色 12 = 淡红色
5 = 紫色 13 = 淡紫色
6 = 黄色 14 = 淡黄色
7 = 白色 15 = 亮白色

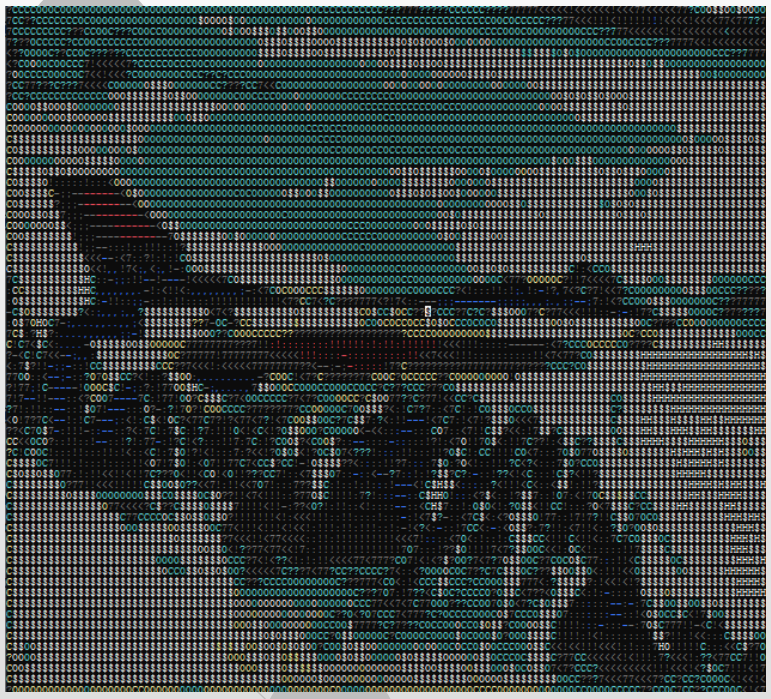
# 灰色与彩色思路对比



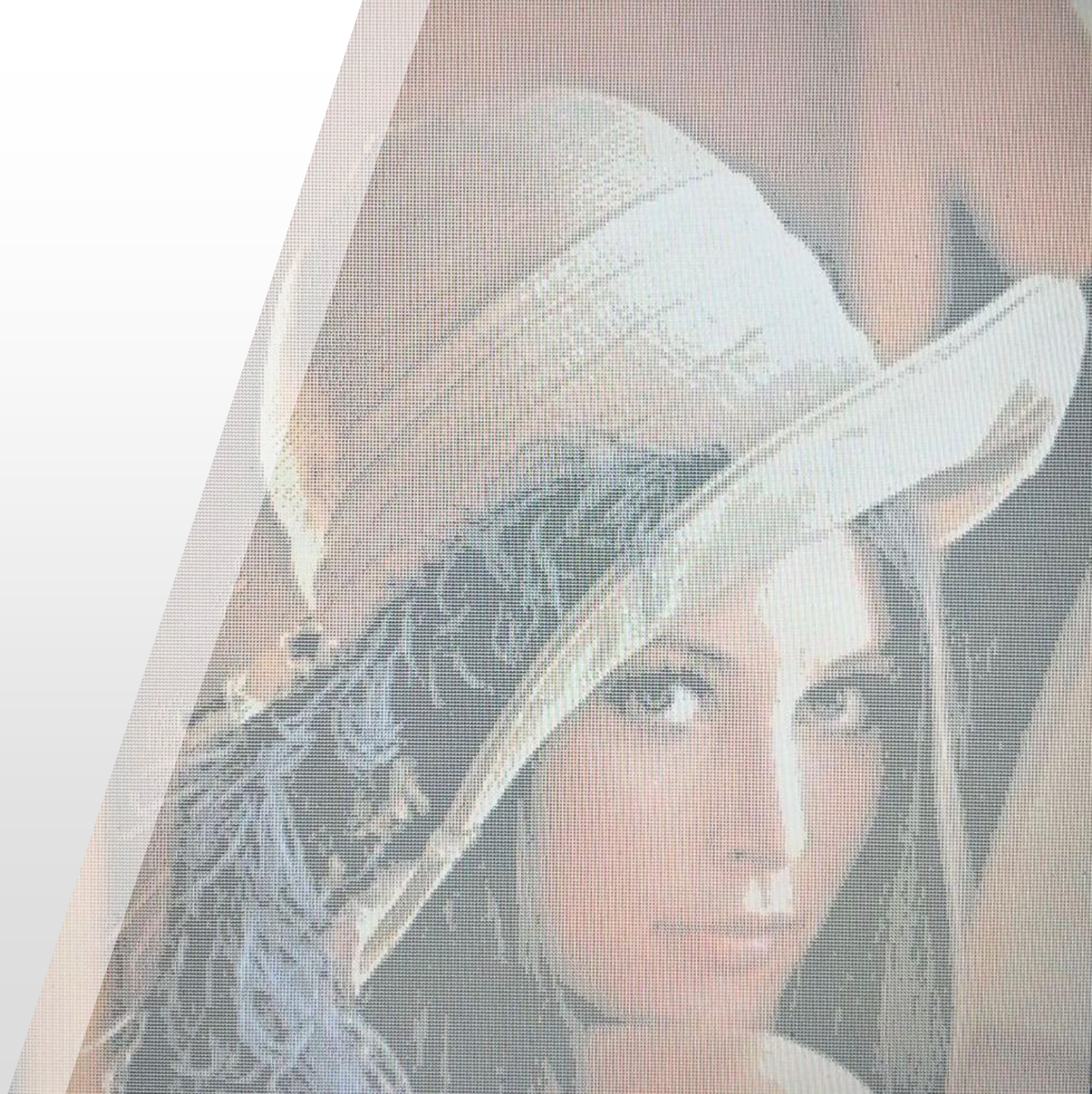
SSSSSSSSSSSSSSS

- 除开黑色还剩15种颜色，用ps分别算出了它们的RGB值
- 用得出的RGB值与这15种颜色比较（此处没有想到什么好办法，就取的RGB总值相差最小的）
- 在前面灰色字符的基础上用void COLOR_PRINT(const char s, int color){...}改变颜色
- （可以看到因为控制台颜色的限制颜色与色彩与原图是有出入的）

```
int ori[15][3] = { 0,55,218,19,161,14,58,150,221,197,15,
int col = 0;
int com[15][4];
for (int i = 0; i < 15; i++)
{
    com[i][0] = abs(ori[i][0] - r);
    com[i][1] = abs(ori[i][1] - g);
    com[i][2] = abs(ori[i][2] - b);
    com[i][3] = com[i][0] + com[i][1] + com[i][2];
}
```



# 接下来就是控制台扩展以及美丽的easyx实现



- 前面，我们已经成功将图片转成了灰度字符画，同时并不满足于此的你也利用控制台提供的颜色实现了彩色字符画，但是，追求卓越的你并不满足与此，你希望控制台的色彩能够更加丰富多彩一点，但是你需要的颜色种类可能是：256×256×256种，而控制台只有……16种（这还得包括白和黑）

- 此刻的你只想赶快丢弃这垃圾的控制台投入到EasyX的怀抱之中，然而……

  注意：本次作业不允许使用外部库完成

- 但是……这真的是控制台的极限了吗？

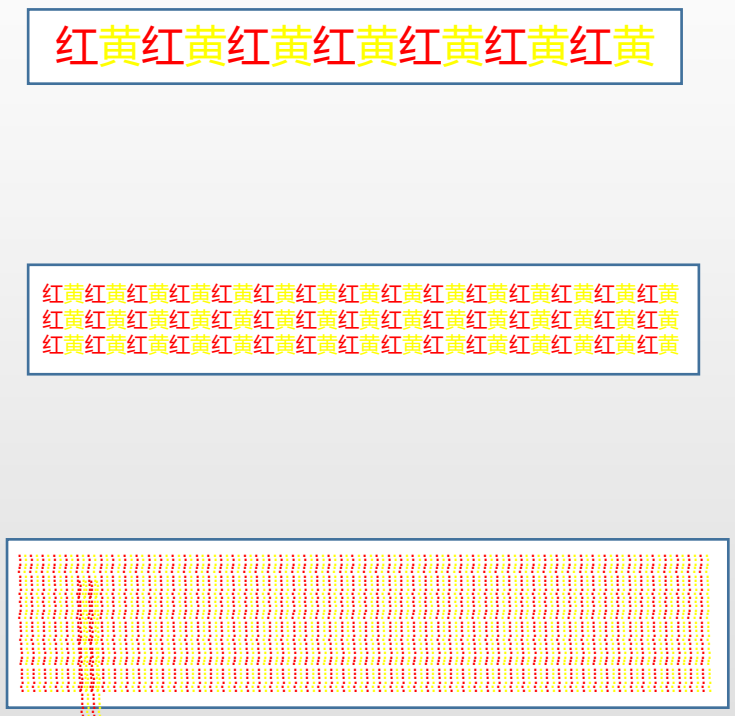
# 控制台颜色扩展：

- 对于字符本身，控制台只能提供16种基础颜色，而其中背景色还要占用一种，实际能使用的颜色只有15种，所以对于颜色界限分明的图片可以实现基本的彩色，但是如果处理涉及到颜色过渡的图片时候似乎就有些力不从心。

- 比如控制台可以为我们提供红色，也可以为我们提供黄色，那么如何才能够扩展控制台的输出颜色使之可以完成从红色到黄色之间输出橙色的过渡呢。

- 这时，我们此前为了处理控制台输出的字符画被纵向拉长而采用的用两个字符表示一个像素点的处理，为我们提供了一个解决的方法。

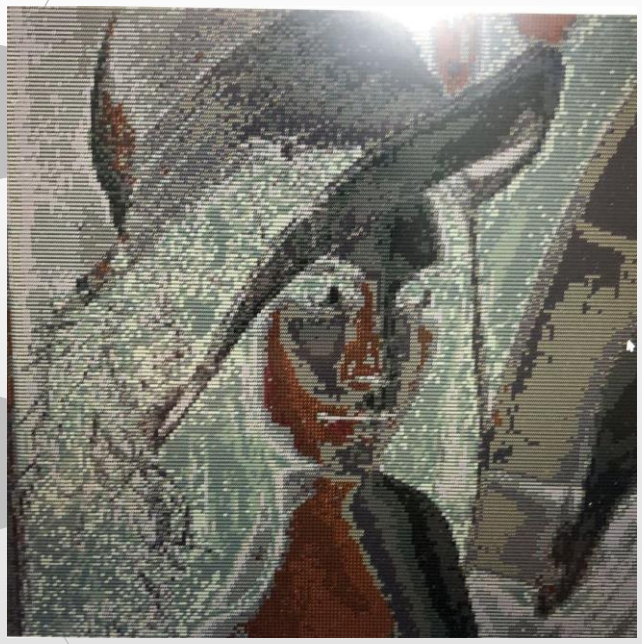
# 控制台颜色扩展




用两个不同颜色的字符表示一个像素，
随着字符的缩小，橙色逐渐开始显现

# 好了控制台我又来了

- 我们先简单按照各通道的RGB的值在不同区段内所呈现出的主要颜色将三通道进行分段：例如当RGB值全部在50以下的时候实际上就可以将该像素点就可以近似用纯黑来替代，进而可以通过输出黑色的字符来实现这个效果。于是我们先简单地分了它20段……



- 好了你能看到这确实也是彩色，但是……结果还是让人大失所望

- 你发现此时的色彩还是不够丰富于是你决定：放弃  继续细化颜色梯度

# 好了控制台我又双叒叕来了

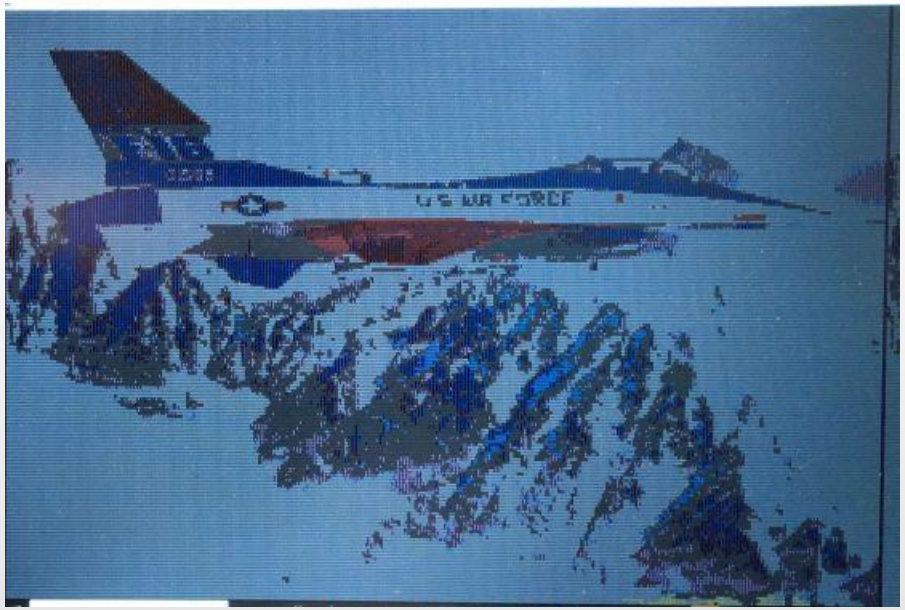
- 这一次我“简单”分他40段……

- 效果拔群！但是你还是可以改进！

  (这里为了图片效果将所有字符都设置成了M)

- 于是追求卓越的你决定……

快乐到飞起，控制台再见，EasyX它不香么……



# 又到了放飞自我的环节了

# EasyX实现高配彩色字符画

控制台由于缺乏RGB方式设置字体颜色的操作，终究只能完成低配版的彩色字符画，为了实现高配的彩色字符画，我们再次请出了我们本学期无数次使用到的巨佬——**EasyX**


有了EasyX终于可以使用RGB形式来控制输出字符的颜色了，将我们读入的图片的RGB的值作为参数传递给settextcolor()函数，然后把字符打印出来，大功告成。

- 然后你就得到了“你以为”的效果：


没错，我们确实实现了彩色，但是……似乎还差了那么……一点点

其实，出现这种图片黑化现象十分严重的现象，是因为所输出的字符太细，导致其所占用的空间比例不够而使得图片发黑“魔化”，所以我们只需要将这些字符settextstyle()加粗或者换用中文字符即可解决。

改用中文

```
//char asciiStrength[] = { 'M','N','H','Q','$','O','C','?','7','>','!',':','-',';','.' };
TCHAR asciiStrength[5][10] = { _T("爨"),_T("嗨"),_T("黑"),_T("丝"),_T("开") };
```

```
settextstyle(3, 3, _T("consolas"), 0, 0, 1000, 0, 0, 0);
```

加粗字体（默认为0）

- 然后我们得到了:


现在好了很多，但是……似乎还是差了那么真正的一点点

图片还是有一些地方的色彩比较异常，而产生的这种现象的原因还是上面提到的问题，只不过这一次的原因是此前给出的程序中是将颜色深的地方设置为较为明亮的'M',而颜色较浅的地方则设置为较为暗淡的'.'，而在黑色的背景上这样反而会导致效果不佳，因此我们只需要转换一下对应规则即可（当然也可以是中文）

```
//·将灰度分为15级，每一级由一个ascii字符表示，强度越大ascii字符内容越少(越白)LF
char·asciiStrength[]·=·{·'M','N','H','Q','$','O','C','?','7','>','!',':','-',';','.'·};
```

```
//将灰度分为15级，每一级由一个ascii字符表示，强度越大ascii字符内容越多（越白）
char asciiStrength[] = { '.','.',';','-',':','!','>','7','?','C','O','$','Q','H','N','M' };
```

- 最后我们终于得到了:

我们终于用字符画出了原图

# 感谢聆听

意犹未尽？欢迎私下联系我们一起讨论噢~

汇报人：李牧原 王睿思 周心怡 周靖淇