彩色空间转换实验

基本原理：

RGB和YUV彩色空间的基础知识；数据类型的分析

1）不带参数的主函数定义void main（）

带参数的主函数定义void main（int argc，char\*argv[ ]）

调用：命令名 参数1 参数2 ... 参数n

参数值：argc=n+1 argv[0]=命令名 argv[1]=参数1 argv[2]=参数2 ... argv[n]=参数n

2）文件相关指针

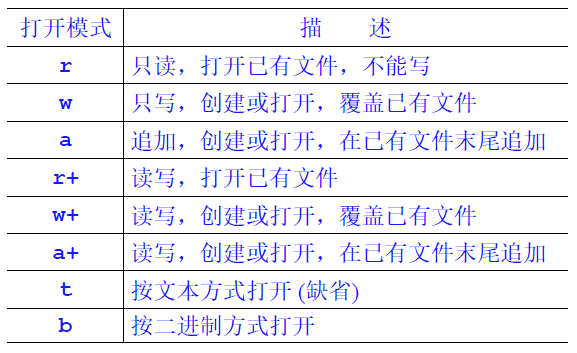
FILE \*fp;

FILE \*fopen（char \*filename，char \*mode）;

filename是要打开的文件路径

mode是要打开的模式

若成功，返回指向被打开文件的指针；若出错，返回空指针NULL

举例：

//指针定义

char \*rgbFileName = NULL;

FILE \*rgbFile = NULL;

//参数获取

rgbFileName = argv[1];

//打开函数

rgbFile = fopen(rgbFileName, "rb");

if (rgbFile == NULL)

{

printf("cannot find rgb file\n");

exit(1);

}

else

{

printf("The input rgb file is %s\n", rgbFileName);

}

fclose（FILE \*fp）;

fp是要关闭的文件指针

若成功，返回0；若出错，返回EOF（-1）

不用的文件应

feof（FILE \*fp）;

fp是文件指针

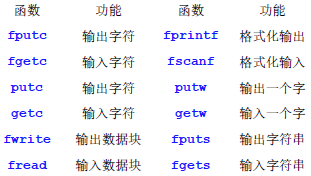
若文件结束，返回非零值；若文件尚未结束，返回0

fputc（int c, FILE \*fp）;

c是要输出到文件的字符

fp是文件指针

若成功，返回输出的字符；若失败，返回EOF

fgetc（FILE \*fp）;

fp是文件指针

若成功，返回输入的字符；若失败或文件结束，返回EOF

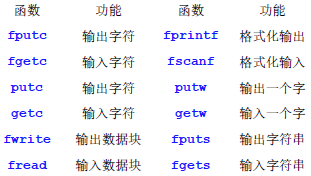
fwrite/fread（void \*buffer, size\_t size, size\_t count, FILE \*fp）;

buffer是要写/读的数据块地址

size是要写/读的每个数据项的字节数

count是要写/读的数据项数量

fp是文件指针

若成功，返回实际写/读的数据项数量；若失败，一般返回0

举例：

//开辟缓存空间

rgbBuf = (u\_int8\_t\*)malloc(frameWidth \* frameHeight \* 3);

yBuf = (u\_int8\_t\*)malloc(frameWidth \* frameHeight);

uBuf = (u\_int8\_t\*)malloc((frameWidth \* frameHeight) / 4);

vBuf = (u\_int8\_t\*)malloc((frameWidth \* frameHeight) / 4);

//读取数据

fread(rgbBuf, 1, frameWidth \* frameHeight \* 3, rgbFile);

//写入数据

fwrite(yBuf, 1, frameWidth \* frameHeight, yuvFile);

fwrite(uBuf, 1, (frameWidth \* frameHeight) / 4, yuvFile);

fwrite(vBuf, 1, (frameWidth \* frameHeight) / 4, yuvFile);

fprintf/fscanf（FILE \*fp, char \*format[,address/argument,...]）;

fputs(char \*s, FILE \*fp);

若成功，返回输出字符个数；若失败，返回EOF

fgets（char \*s, int n, FILE \*fp）;

若成功，返回s首地址；若失败，返回NULL

从fp输入字符串到s中，输入n-1个字符，或遇到换行符或EOF为止，读完后自动在字符串末尾添加 ’\0’

rewind（FILE \*fp）;

fp是文件指针

使文件位置指针重新返回文件开头

fseek（FILE \*fp, long offset, int whence）;

fp是文件指针

offset是偏移量

whence是起始位置

随机改变文件的位置指针

SEEK\_SET(0)是文件开始

SEEK\_CUR(1)是文件当前位置

SEEK\_END(2)是文件末尾

ftell（FILE \*fp）;

fp是文件指针

返回fp所指向文件中的读写位置

3）动态数组和指针

①申请空间

char \*name;

name = (char \*)malloc(20);

float \*pf;

pf = (float \*)malloc(sizeof(float)\*20);

double \*pd;

pd = (double \*)malloc(sizeof(double)\*50);

②赋值

int \*pi;

pi = (int \*)malloc(sizeof(int)\*100);

\*pi = 0;

\*(pi+1) = 1;

\*(pi+2) = 2;

int j;

for(j=0; j<100; j++)

\*(pi+j) = 0;

举例：

for (i = 0; i < size; i++)

{

g = b + 1;

r = b + 2;

\*y = (unsigned char)( RGBYUV02990[\*r] + RGBYUV05870[\*g] + RGBYUV01140[\*b]);

\*u = (unsigned char)(- RGBYUV01684[\*r] - RGBYUV03316[\*g] + (\*b)/2 + 128);

\*v = (unsigned char)( (\*r)/2 - RGBYUV04187[\*g] - RGBYUV00813[\*b] + 128);

b += 3;

y ++;

u ++;

v ++;

}

③空间回收

free（filename）;

4）RGB to YUV文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区等）

读取RGB文件，抽取RGB数据写入缓冲区

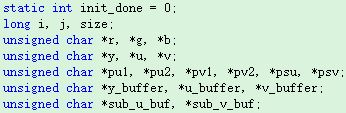
调用RGB2YUV的函数实现RGB到YUV数据的转换

写YUV文件

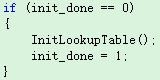
程序收尾工作（关闭文件，释放缓冲区）

②代码分析

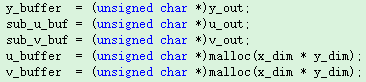
定义变量

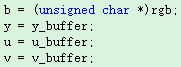


调用快速查找表

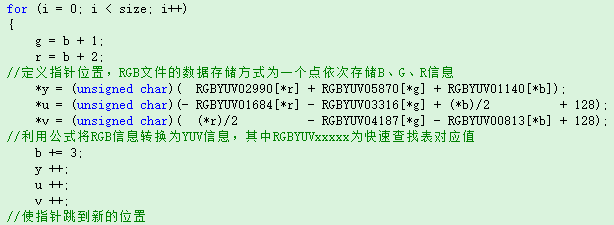


定义指针、开辟缓存空间

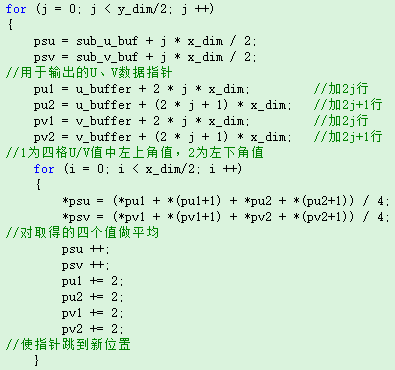




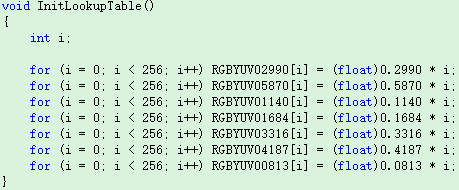
将RGB数据转换为YUV数据



将转换为的U、V数据转换为4：2：0格式



附查找表



5）YUV to RGB文件转换

①流程分析

程序初始化（打开两个文件、定义变量和缓冲区）

读取YUV文件，抽取YUV数据写入缓冲区

调用YUV2RGB的函数实现YUV到RGB数据的转换

写RGB文件

程序收尾工作（关闭文件，释放缓冲区）

②代码分析

定义变量

调用快速查找表

定义指针、开辟缓存空间

将YUV数据转换为RGB数据

附查找表