颜色空间是一个三维坐标系统，每一种颜色由一个点表示。在 RGB 颜色空间中，红，绿，蓝是基本元素。RGB 格式是显示器通常使用的格式。在 YUV 空间中，每一个颜色有一个亮度信号 Y，和两个色度信号 U 和 V。亮度信号是强度的感觉，它和色度信号断开，这样的话强度就可以在不影响颜色的情况下改变。YUV 格式通常用于 PAL制，即欧洲的电视传输标准，而且缺省情况下是图像和视频压缩的标准。  
  
YUV 使用RGB的信息，但它从全彩色图像中产生一个黑白图像，然后提取出三个主要的颜色变成两个额外的信号来描述颜色。把这三个信号组合回来就可以产生一个全彩色图像。  
  
YUV 使用红，绿，蓝的点阵组合来减少信号中的信息量。Y 通道描述 Luma 信号，它与亮度信号有一点点不同，值的范围介于亮和暗之间。 Luma 是黑白电视可以看到的信号。U (Cb) 和 V (Cr) 通道从红 (U) 和蓝 (V) 中提取亮度值来减少颜色信息量。这些值可以从新组合来决定红，绿和蓝的混合信号。  
  
YUV和RGB的转换:  
Y = 0.299 R + 0.587 G + 0.114 B  
U = -0.1687 R - 0.3313 G + 0.5 B + 128  
V = 0.5 R - 0.4187 G - 0.0813 B + 128  
R = Y + 1.402 (V-128) G  
= Y - 0.34414 (U-128) - 0.71414 (V-128) B  
= Y + 1.772 (U-128)  
  
以前，一直没明白yuv和YcbCr之间的差异，想必有些朋友也会有同样的疑惑。所以，我看完之后就记载下来了。   
一、和rgb之间换算公式的差异  
yuv<-->rgb  
Y'= 0.299\*R' + 0.587\*G' + 0.114\*B'  
U'= -0.147\*R' - 0.289\*G' + 0.436\*B' =0.492\*(B'- Y')  
V'= 0.615\*R' - 0.515\*G' - 0.100\*B' =0.877\*(R'- Y')  
R' = Y' + 1.140\*V'  
G' = Y' - 0.394\*U' - 0.581\*V'  
B' = Y' + 2.032\*U'  
yCbCr<-->rgb  
Y’ = 0.257\*R' + 0.504\*G' + 0.098\*B' + 16  
Cb' = -0.148\*R' - 0.291\*G' + 0.439\*B' + 128  
Cr' = 0.439\*R' - 0.368\*G' - 0.071\*B' + 128  
R' = 1.164\*(Y’-16) + 1.596\*(Cr'-128)  
G' = 1.164\*(Y’-16) - 0.813\*(Cr'-128) -0.392\*(Cb'-128)  
B' = 1.164\*(Y’-16) + 2.017\*(Cb'-128)  
Note: 上面各个符号都带了一撇，表示该符号在原值基础上进行了gamma correction  
二、来源上的差异  
yuv色彩模型来源于rgb模型，  
该模型的特点是将亮度和色度分离开，从而适合于图像处理领域。  
应用：basic colormodel used in analogue color TV broadcasting.  
YCbCr模型来源于yuv模型。YCbCr is ascaled and offset version of the YUV color space.  
应用：数字视频，ITU-RBT.601 recommendation  
ps:  
通过上面的比较可以确定，我们在h.264,mpeg等编码标准中用的yuv其实是YcbCr，大家不要被名称搞混淆了。  
  
  
  
  
人类视觉系统(HVS)相比亮度来说对于颜色不是那么敏感的。在RGB颜色空间中，三种颜色被平等地看待，并用相同的分辨率  
存放起来。但是通过把亮度与颜色信息分离，并对亮度值取更高的分辨率可以更有效地表示一个颜色图像。  
  
YCbCr颜色空间和它的变换(通常写为YUV)是一种流行而高效的表示一个颜色图像的方法。Y是亮度值，由R,G,B的加权平均可以  
得到： Y=krR +kgG + kbB  
这里k是加权因子。  
  
颜色信号可以由不同的颜色差别来表示：  
Cb = B-Y  
Cr = R-Y  
Cg = G-Y  
对于一个颜色图像的完整的描述由给定Y和三个色差:Cb,Cr,Cg来表示。  
  
目前为止，我们的表示方法好像并不那么好，因为相比RGB表示来说，我们这次用了四个参数。然后Cb+Cr+Cg是一个常数，那么  
我们只需要两个色度参数就可以了，第三个可以通过其他两个计算出来。在YCbCr空间中，只有Y和Cb,Cr值被传输和存储，而且  
Cb和Cr的分辨率可以比Y低，因为人类视觉系统对于亮度更加敏感。这就减少了表示图像的数据量。通常的观察情况下，RGB和  
YCbCr表示的图像看上去没有什么不同。对于色度采用比亮度低的分辨率进行采样是一种简单而有效的压缩办法。  
  
一个RGB图像可以在捕捉之后转换为YCbCr格式用来减少存储和传输负担。在显示图象之前，再转回为RGB.注意没有必要去指明  
分别的加权值kg（因为kb+kr+kg=1)，而且G可以从YCbCr中解压出来，这说明不需要存储和传输Cg参数。  
  
Y = kr R + (1-kb-kr)G + kb B  
Cb = 0.5/(1-kb) \* (B-Y)  
Cr = 0.5/(1-kr) \* (R-Y)  
  
R = Y + (1-kr)/0.5 \* Cr  
G = Y - 2kb(1-kb)/(1-kb-kr) \* Cb -2kr(1-kr)/(1-kb-kr) \* Cr  
B = Y + (1-kb)/0.5 \* Cb  
  
ITU-R的BT.601决议定义了kb=0.114,kr=0.299，那么代换参数就有了如下等式：  
  
Y = 0.299R + 0.587G + 0.114B  
Cb = 0.564(B - Y )  
Cr = 0.713(R - Y )  
  
R = Y + 1.402Cr  
G = Y - 0.344Cb - 0.714Cr  
B = Y + 1.772Cb  
  
**YCbCr采样格式**  
4:4:4采样就是说三种元素Y,Cb,Cr有同样的分辨率,这样的话,在每一个像素点上都对这三种元素进行采样.数字4是指在水平方向上对于各种元素的采样率,比如说,每四个亮度采样点就有四个Cb的Cr采样值.  4:4:4采样完整地保留了所有的信息值.  
4:2:2采样中(有时记为YUY2),色度元素在纵向与亮度值有同样的分辨率,而在横向则是亮度分辨率的一半(4:2:2表示每四个亮度值就有两个Cb和Cr采样.)  4:2:2视频用来构造高品质的视频彩色信号.  
  
在流行的4:2:0采样格式中(常记为YV12)Cb和Cr在水平和垂直方向上有Y分辨率的一半.4:2:0有些不同，因为它**并不是**指在**实际**采样中使用4:2:0，而是在编码史中定义这种编码方法是用来区别于4:4:4和4:2:2方法的).  
4:2:0采样被广泛地应用于消费应用中，比如视频会议，数字电视和DVD存储中。因为每个颜色差别元素中包含了四分之一的Y采样元素量，那么4:2:0YCbCr视频需要刚好4:4:4或RGB视频中采样量的一半。  
  
4:2:0采样有时被描述是一个"每像素12位"的方法。这么说的原因可以从对四个像素的采样中看出.使用4:4:4采样，一共要进行12次采样，对每一个Y,Cb和Cr，就需要12\*8=96位，平均下来要96/4=24位。使用4:2:0就需要6\*8=48位，平均每个像素48/4=12位。  
  
在一个4:2:0隔行扫描的视频序列中，对应于一个完整的视频帧的Y,Cb,Cr采样分配到两个场中。可以得到，隔行扫描的总采样数跟渐进式扫描中使用的采样数目是相同的。  
  
2.5 视频格式  
这本书中描述的视频压缩标准可以压缩很多种视频帧格式。实际中，捕捉或转化一个中间格式或一系列中间格式是很平常的事情。  
CIF就是一种常见的流行的格式，并由它衍生出了4CIF和Sub-QCif。帧分辨率的选择取决于应用程序，可使用的存储量以及传输带宽。比如说4CIF对于标准定义的电视和DVD视频来说是合适的,CIF和QCIF在视频会议中是常被使用的格式。QCIF和SQCIF对于移动设备的多媒体程序来说是合适的，在这样的情况下，显示分辨率和码率都是有限的。以下是各种格式的具体使用位数的需求（使用4:2:0采样，对于每个元素用8个位大小表示)：  
  
格式： Sub-QCIF 亮度分辨率： 128\*96 每帧使用的位: 147456  
格式： QCIF 亮度分辨率： 176\*144 每帧使用的位: 304128  
格式： CIF 亮度分辨率： 352\*288 每帧使用的位: 1216512  
格式： 4CIF 亮度分辨率： 704\*576 每帧使用的位: 4866048  
  
  
一种在电视信号中被应用的很广的数字视频信号编码格式就是ITU-R的BT.601-5 提案。亮度元素被在13.5MHz下采样，而亮度值则在6.75MHz下采样，这样就形成了一个4:2;2的Y:Cb:Cr采样结果。采样数字信号的参数取决于视频码率(对于NTSC来说是30Hz,对于PAL/SECAM来说是25Hz)。NTSC的30Hz是对低空间分辨率的补偿，这样总的码率就是216Mbps.实际显示的激活部分的区域要比总量小，因为它去掉了在一帧边缘处的水平和垂直空白间隔。每一个采样都有0-255的采样范围。0和255两个等级被留作同步，而且激活的亮度信号被限制到26(黑色）到235(白色)之间.  
  
  
  
  
  
YUV  
  
视频编解码器功能   
视频编码器要求YUV4:2:0格式的视频输入，因此可能根据应用需要进行视频输入的预处理，即对YUV4:2:2隔行扫描(例如从摄像机)到YUV 4:2:0非隔行扫描转换，仅抽取但不过滤UV分。对视频解码器而言，还需要进行后处理，以将解码的YUV 4:2:0数据转换为RGB进行显示，包括：YUV 4:2:0到RGB转换；16位或12位RGB显示格式；0到90度旋转，实现横向或纵向显示。此外，视频编解码器通常还要求具有以下功能和特性：   
支持MPEG-4简单类 0、1 与 2 级；   
兼容H.263与 MPEG-4 编解码标准；   
MPEG-4视频解码器支持的可选项有：AC/DC预测、可逆可变长度编码(RVLC)、再同步标志(RM)、数据分割(DP)、错误隐藏专利技术、支持每个宏块4个运动矢量(4MV)、自由运动补偿、解码VOS层；   
MPEG-4视频编码器选项有：RVLC、RM、DP、支持每个宏块4个运动矢量(4MV)、报头扩展码、支持编码期间码率改变、支持编码期间编码帧率改变、插入或不插入可视对象序列起始码；   
支持编码期间序列中插入I帧；   
支持编码器自适应帧内刷新(AIR)；   
支持多编解码器，可用相同代码运行多个编解码器实例。   
  
RGB  
  
红绿蓝(RGB)是计算机显示的基色，RGB565支持的色深可编程至高达每像素16位，即 RGB565(红色5位，绿色6位，蓝色5位)。  
  
YCbCr   
  
在DVD、摄像机、数字电视等消费类视频产品中，常用的色彩编码方案是YCbCr，其中Y是指亮度分量，Cb指蓝色色度分量，而Cr指红色色度分量。人的肉眼对视频的Y分量更敏感，因此在通过对色度分量进行子采样来减少色度分量后，肉眼将察觉不到的图像质量的变化。主要的子采样格式有 YCbCr 4:2:0、YCbCr 4:2:2 和 YCbCr 4:4:4。   
4:2:0表示每4个像素有4个亮度分量，2个色度分量 (YYYYCbCr)，仅采样奇数扫描线，是便携式视频设备(MPEG-4)以及电视会议(H.263)最常用格式；4：2：2表示每4个像素有4个亮度分量，4个色度分量(YYYYCbCrCbCr)，是DVD、数字电视、HDTV 以及其它消费类视频设备的最常用格式；4：4：4表示全像素点阵(YYYYCbCrCbCrCbCrCbCr)，用于高质量视频应用、演播室以及专业视频产品。   
  
  
  
小知识：RGB与YUV  
  
计算机彩色显示器显示色彩的原理与彩色电视机一样，都是采用R（Red）、G（Green）、B（Blue）相加混色的原理：通过发射出三种不同强度的电子束，使屏幕内侧覆盖的红、绿、蓝磷光材料发光而产生色彩。这种色彩的表示方法称为RGB色彩空间表示（它也是多媒体计算机技术中用得最多的一种色彩空间表示方法）。  
根据三基色原理，任意一种色光F都可以用不同分量的R、G、B三色相加混合而成。  
  
F = r [ R ] + g [ G ] + b [ B ]  
  
其中，r、g、b分别为三基色参与混合的系数。当三基色分量都为0（最弱）时混合为黑色光；而当三基色分量都为k（最强）时混合为白色光。调整r、g、b三个系数的值，可以混合出介于黑色光和白色光之间的各种各样的色光。  
那么YUV又从何而来呢？在现代彩色电视系统中，通常采用三管彩色摄像机或彩色CCD摄像机进行摄像，然后把摄得的彩色图像信号经分色、分别放大校正后得到RGB，再经过矩阵变换电路得到亮度信号Y和两个色差信号R－Y（即U）、B－Y（即V），最后发送端将亮度和色差三个信号分别进行编码，用同一信道发送出去。这种色彩的表示方法就是所谓的YUV色彩空间表示。  
采用YUV色彩空间的重要性是它的亮度信号Y和色度信号U、V是分离的。如果只有Y信号分量而没有U、V分量，那么这样表示的图像就是黑白灰度图像。彩色电视采用YUV空间正是为了用亮度信号Y解决彩色电视机与黑白电视机的兼容问题，使黑白电视机也能接收彩色电视信号。  
YUV与RGB相互转换的公式如下（RGB取值范围均为0-255）：  
  
Y = 0.299R + 0.587G + 0.114B  
U = -0.147R - 0.289G + 0.436B  
V = 0.615R - 0.515G - 0.100B  
  
R = Y + 1.14V  
G = Y - 0.39U - 0.58V  
B = Y + 2.03U  
  
在DirectShow 中，常见的RGB格式有RGB1、RGB4、RGB8、RGB565、RGB555、RGB24、RGB32、ARGB32等；常见的YUV格式有 YUY2、YUYV、YVYU、UYVY、AYUV、Y41P、Y411、Y211、IF09、IYUV、YV12、YVU9、YUV411、 YUV420等。作为视频媒体类型的辅助说明类型（Subtype），它们对应的GUID见表2.3。  
  
表2.3 常见的RGB和YUV格式  
  
GUID   格式描述  
MEDIASUBTYPE\_RGB1    2色，每个像素用1位表示，需要调色板  
MEDIASUBTYPE\_RGB4    16色，每个像素用4位表示，需要调色板  
MEDIASUBTYPE\_RGB8    256色，每个像素用8位表示，需要调色板  
MEDIASUBTYPE\_RGB565    每个像素用16位表示，RGB分量分别使用5位、6位、5位  
MEDIASUBTYPE\_RGB555    每个像素用16位表示，RGB分量都使用5位（剩下的1位不用）  
MEDIASUBTYPE\_RGB24    每个像素用24位表示，RGB分量各使用8位  
MEDIASUBTYPE\_RGB32    每个像素用32位表示，RGB分量各使用8位（剩下的8位不用）  
MEDIASUBTYPE\_ARGB32    每个像素用32位表示，RGB分量各使用8位（剩下的8位用于表示Alpha通道值）  
MEDIASUBTYPE\_YUY2    YUY2格式，以4:2:2方式打包  
MEDIASUBTYPE\_YUYV    YUYV格式（实际格式与YUY2相同）  
MEDIASUBTYPE\_YVYU    YVYU格式，以4:2:2方式打包  
MEDIASUBTYPE\_UYVY    UYVY格式，以4:2:2方式打包  
MEDIASUBTYPE\_AYUV    带Alpha通道的4:4:4 YUV格式  
MEDIASUBTYPE\_Y41P    Y41P格式，以4:1:1方式打包  
MEDIASUBTYPE\_Y411    Y411格式（实际格式与Y41P相同）  
MEDIASUBTYPE\_Y211    Y211格式  
MEDIASUBTYPE\_IF09    IF09格式  
MEDIASUBTYPE\_IYUV    IYUV格式  
MEDIASUBTYPE\_YV12    YV12格式  
MEDIASUBTYPE\_YVU9    YVU9格式  
  
下面分别介绍各种RGB格式。  
  
¨ RGB1、RGB4、RGB8都是调色板类型的RGB格式，在描述这些媒体类型的格式细节时，通常会在BITMAPINFOHEADER数据结构后面跟着一个调色板（定义一系列颜色）。它们的图像数据并不是真正的颜色值，而是当前像素颜色值在调色板中的索引。以RGB1（2色位图）为例，比如它的调色板中定义的两种颜色值依次为0x000000（黑色）和0xFFFFFF（白色），那么图像数据001101010111…（每个像素用1位表示）表示对应各像素的颜色为：黑黑白白黑白黑白黑白白白…。  
  
¨ RGB565使用16位表示一个像素，这16位中的5位用于R，6位用于G，5位用于B。程序中通常使用一个字（WORD，一个字等于两个字节）来操作一个像素。当读出一个像素后，这个字的各个位意义如下：  
     高字节              低字节  
R R R R R G G G     G G G B B B B B  
可以组合使用屏蔽字和移位操作来得到RGB各分量的值：  
  
#define RGB565\_MASK\_RED    0xF800  
#define RGB565\_MASK\_GREEN  0x07E0  
#define RGB565\_MASK\_BLUE   0x001F  
R = (wPixel & RGB565\_MASK\_RED) >>11;   // 取值范围0-31  
G = (wPixel & RGB565\_MASK\_GREEN)>> 5;  // 取值范围0-63  
B = wPixel & RGB565\_MASK\_BLUE;        // 取值范围0-31  
  
¨ RGB555是另一种16位的RGB格式，RGB分量都用5位表示（剩下的1位不用）。使用一个字读出一个像素后，这个字的各个位意义如下：  
     高字节             低字节  
X R R R R G G       G G G B B B B B       （X表示不用，可以忽略）  
可以组合使用屏蔽字和移位操作来得到RGB各分量的值：  
  
#define RGB555\_MASK\_RED    0x7C00  
#define RGB555\_MASK\_GREEN  0x03E0  
#define RGB555\_MASK\_BLUE   0x001F  
R = (wPixel & RGB555\_MASK\_RED) >>10;   // 取值范围0-31  
G = (wPixel & RGB555\_MASK\_GREEN)>> 5;  // 取值范围0-31  
B = wPixel & RGB555\_MASK\_BLUE;        // 取值范围0-31  
  
¨ RGB24使用24位来表示一个像素，RGB分量都用8位表示，取值范围为0-255。注意在内存中RGB各分量的排列顺序为：BGR BGR BGR…。通常可以使用RGBTRIPLE数据结构来操作一个像素，它的定义为：  
  
typedef struct tagRGBTRIPLE {   
BYTE rgbtBlue;    // 蓝色分量  
BYTE rgbtGreen;   // 绿色分量  
BYTE rgbtRed;     // 红色分量  
} RGBTRIPLE;  
  
¨ RGB32使用32位来表示一个像素，RGB分量各用去8位，剩下的8位用作Alpha通道或者不用。（ARGB32就是带Alpha通道的 RGB32。）注意在内存中RGB各分量的排列顺序为：BGRA BGRA BGRA…。通常可以使用RGBQUAD数据结构来操作一个像素，它的定义为：  
  
typedef struct tagRGBQUAD {  
BYTE    rgbBlue;      // 蓝色分量  
BYTE    rgbGreen;     // 绿色分量  
BYTE    rgbRed;       // 红色分量  
BYTE    rgbReserved;  // 保留字节（用作Alpha通道或忽略）  
} RGBQUAD;  
  
下面介绍各种YUV格式。YUV格式通常有两大类：打包（packed）格式和平面（planar）格式。前者将YUV分量存放在同一个数组中，通常是几个相邻的像素组成一个宏像素（macro-pixel）；而后者使用三个数组分开存放YUV三个分量，就像是一个三维平面一样。表2.3中的YUY2到 Y211都是打包格式，而IF09到YVU9都是平面格式。（注意：在介绍各种具体格式时，YUV各分量都会带有下标，如Y0、U0、V0表示第一个像素的YUV分量，Y1、U1、V1表示第二个像素的YUV分量，以此类推。）  
  
¨ YUY2（和YUYV）格式为每个像素保留Y分量，而UV分量在水平方向上每两个像素采样一次。一个宏像素为4个字节，实际表示2个像素。（4:2:2的意思为一个宏像素中有4个Y分量、2个U分量和2个V分量。）图像数据中YUV分量排列顺序如下：  
Y0 U0 Y1 V0    Y2 U2 Y3 V2 …  
  
¨ YVYU格式跟YUY2类似，只是图像数据中YUV分量的排列顺序有所不同：  
Y0 V0 Y1 U0    Y2 V2 Y3 U2 …  
  
¨ UYVY格式跟YUY2类似，只是图像数据中YUV分量的排列顺序有所不同：  
U0 Y0 V0 Y1    U2 Y2 V2 Y3 …  
  
¨ AYUV格式带有一个Alpha通道，并且为每个像素都提取YUV分量，图像数据格式如下：  
A0 Y0 U0 V0    A1 Y1 U1 V1 …  
  
¨ Y41P（和Y411）格式为每个像素保留Y分量，而UV分量在水平方向上每4个像素采样一次。一个宏像素为12个字节，实际表示8个像素。图像数据中YUV分量排列顺序如下：  
U0 Y0 V0 Y1    U4 Y2 V4 Y3    Y4 Y5 Y6 Y8 …   
  
¨ Y211格式在水平方向上Y分量每2个像素采样一次，而UV分量每4个像素采样一次。一个宏像素为4个字节，实际表示4个像素。图像数据中YUV分量排列顺序如下：  
Y0 U0 Y2 V0    Y4 U4 Y6 V4 …  
  
¨ YVU9格式为每个像素都提取Y分量，而在UV分量的提取时，首先将图像分成若干个4 x 4的宏块，然后每个宏块提取一个U分量和一个V分量。图像数据存储时，首先是整幅图像的Y分量数组，然后就跟着U分量数组，以及V分量数组。IF09格式与YVU9类似。  
  
¨ IYUV格式为每个像素都提取Y分量，而在UV分量的提取时，首先将图像分成若干个2 x 2的宏块，然后每个宏块提取一个U分量和一个V分量。YV12格式与IYUV类似。  
  
¨ YUV411、YUV420格式多见于DV数据中，前者用于NTSC制，后者用于PAL制。YUV411为每个像素都提取Y分量，而UV分量在水平方向上每4个像素采样一次。YUV420并非V分量采样为0，而是跟YUV411相比，在水平方向上提高一倍色差采样频率，在垂直方向上以U/V间隔的方式减小一半色差采样，如图2.12所示。  
  
  
  
  
  
  
  
  
YUV转换为RGB的公式  
  
  
第一个公式是YUV转换RGB（范围0-255）时用的，第二个公式是用在YUV(601)也成为YCbCr转换RGB（范围0-255)时用的。   
  
1.Y =  0.299R + 0.587G + 0.114B  
   U= -0.147R - 0.289G + 0.436B  
   V=  0.615R - 0.515G - 0.100B  
  
   R= Y             + 1.14V  
   G= Y - 0.39U - 0.58V  
   B= Y + 2.03U  
  
2.B= 1.164 \* (Y - 16) + 2.018 \* (U - 128)  
   G=1.164 \* (Y - 16) -   0.38 \* (U - 128) -0.813 \* (V - 128)  
   R=1.164 \* (Y - 16)                            + 1.159 \* (V - 128)  
  
  
  
  
  
程序读出来显现的不正确,源代码大概是这样的:  
        Mywidth = 176;  
Myheight = 144;  
tmp = (uchar \*)malloc(Mywidth \* Myheight\*3);  
buffer = (uchar \*)malloc(Mywidth \* Myheight\*4);  
device\_fd = open("/dev/video0",O\_RDONLY);  
static struct video\_window vidwin;  
vidwin.width = Mywidth;  
vidwin.height = Myheight;  
ioctl(device\_fd, VIDIOCSWIN, &vidwin);  
  
  
read(device\_fd, tmp, Mywidth\*Myheight\*3);  
for(int i = 0; i < 176 \* 144; ++i)  
{  
buffer[4\*i] = tmp[3\*i];//first bit is blue  
buffer[4\*i + 1] = tmp[3\*i + 1];//second bit is green  
buffer[4\*i + 2] = tmp[3\*i + 2] ;//third bit is red  
buffer[4\*i + 3] = 130;//forth bit  
}  
//后面这此是用QT库写的,意思是将buffer的内容转为image再转为pixmap,然后显示出来  
QImage img(buffer, Mywidth, Myheight, 32,NULL, 0, QImage::LittleEndian);   
QPixmap pic;  
pic.convertFromImage(img);  
PixmapVideo->setPixmap(pic);  
  
  
  
FillEllips 函数填充指定的椭圆。椭圆心为（sx, sy），X 轴半径为 rx，Y 轴半径为 ry。   
　　FillSector 函数填充由圆弧和两条半径形成的扇形。圆心为（x, y），半径为 r，起始弧度为 ang1，终止弧度为 ang2。   
　　FillPolygon 函数填充多边形。pts 表示多边形各个顶点，vertices 表示多边形顶点个数。   
　　FloodFill 从指定点（x, y）开始填注。   
　　需要注意的是，所有填充函数使用当前画刷属性（颜色），并且受当前光栅操作的影响。  
　　  
　　下面的例子说明了如何使用FillCircle 和 FillEllipse 函数填充圆或者椭圆。假定给定了两个点，pts[0] 和 pts[1]，其中 pts[0] 是圆心或者椭圆心，而 pts[1] 是圆或者椭圆外切矩形的一个顶点。  
　　  
　　　　　　　 int rx= ABS (pts[1].x - pts[0].x);  
　　　　　　　　int ry= ABS (pts[1].y - pts[0].y);  
　　  
　　　　　　　　if (rx== ry)  
　　　　　　　　　　FillCircle(hdc, pts[0].x, pts[0].y, rx);  
　　　　　　　　else  
　　　　　　　　　　FillEllipse(hdc, pts[0].x, pts[0].y, rx, ry);  
　　  
　　5 建立复杂区域  
　　  
　　除了利用填充生成器进行填充绘制以外，我们还可以使用填充生成器建立由封闭曲线包围的复杂区域。我们知道，MiniGUI 当中的区域是由互不相交的矩形组成的，并且满足 x-y-banned 的分布规则。利用上述的多边形或者封闭曲线生成器，可以将每条扫描线看成是组成区域的高度为 1 的一个矩形，这样，我们可以利用这些生成器建立复杂区域。MiniGUI 利用现有的封闭曲线生成器，实现了如下的复杂区域生成函数：  
　　  
　　BOOL GUIAPIInitCircleRegion (PCLIPRGN dst, int x, int y, int r);  
　　BOOL GUIAPIInitEllipseRegion (PCLIPRGN dst, int x, int y, int rx, int ry);  
　　BOOL GUIAPIInitPolygonRegion (PCLIPRGN dst, const POINT\* pts, int vertices);  
　　BOOL GUIAPIInitSectorRegion (PCLIPRGN dst, const POINT\* pts, int vertices);  
　　  
　　利用这些函数，我们可以将某个区域分别初始化为圆、椭圆、多边形和扇形区域。然后，可以利用这些区域进行点击测试（PtInRegion 和 RectInRegion），或者选择到 DC 当中作为剪切域，从而获得特殊显示效果。  
　　  
　　6 直接访问显示缓冲区  
　　  
　　在新的 GDI 接口中，我们添加了用来直接访问显示缓冲区的函数，原型如下：  
　　  
　　Uint8\*GUIAPI LockDC (HDC hdc, const RECT\* rw\_rc, int\* width, int\* height, int\*pitch);  
　　void GUIAPIUnlockDC (HDC hdc);  
　　  
　　LockDC 函数锁定给定 HDC 的指定矩形区域（由矩形 rw\_rc指定，设备坐标），然后返回缓冲区头指针。当 width、height、pitch 三个指针不为空时，该函数将返回锁定之后的矩形有效宽度、有效高度和每扫描线所占的字节数。   
　　UnlockDC 函数解开已锁定的 HDC。   
　　锁定一个 HDC 意味着 MiniGUI 进入以互斥方式访问显示缓冲区的状态。如果被锁定的 HDC 是一个屏幕 DC（即非内存 DC），则该函数将在必要时隐藏鼠标光标，并锁定 HDC 对应的全局剪切域。在锁定一个 HDC 之后，程序可通过该函数返回的指针对锁定区域进行访问。需要注意的是，不能长时间锁定一个 HDC，也不应该在锁定一个 HDC时进行其他额外的系统调用。  
　　  
　　假定以锁定矩形左上角为原点建立坐标系，X 轴水平向右，Y 轴垂直向下，则可以通过如下的公式计算该坐标系中（x, y）点对应的缓冲区地址（假定该函数返回的指针值为 frame\_buffer）：  
　　  
　　　 Uint8\*pixel\_add = frame\_buffer + y \* (\*pitch) + x \* GetGDCapability (hdc, GDCAP\_BPP);  
　　  
　　根据该 HDC 的颜色深度，就可以对该象素进行读写操作。作为示例，下面的程序段随机填充锁定区域：  
　　  
　　　 int i,width, height, pitch;  
　　　　RECT rc ={0, 0, 200, 200};  
　　　　int bpp =GetGDCapability (hdc, GDCAP\_BPP);  
　　　　Uint8\*frame\_buffer = LockDC (hdc, &rc, &width, &height, &pitch);  
　　　　Uint8\* row= frame\_buffer;  
　　  
　　　　for (i =0; i < \*height; i++) {  
　　　　　　memset(row, rand ()%0x100, \*width \* bpp);  
　　　　　　row +=\*pitch;  
　　　　}  
　　  
　　　　UnlockDC(hdc);  
　　  
　　  
　　7 YUV 覆盖和 Gamma 校正  
　　  
　　为了增强 MiniGUI对多媒体的支持，我们增加了对YUV 覆盖（Overlay）和 Gamma 校正的支持。  
　　  
　　7.1 YUV 覆盖（Overlay）  
　　  
　　多媒体领域中，尤其在涉及到 MPEG 播放时，通常使用 YUV 颜色空间来表示颜色，如果要在屏幕上显示一副 MPEG 解压之后的图片，则需要进行 YUV 颜色空间到 RGB 颜色空间的转换。YUV 覆盖最初来自一些显示芯片的加速功能。这种显示芯片能够在硬件基础上完成YUV 到 RGB 的转换，免去软件转换带来的性能损失。在这种显示芯片上建立了 YUV 覆盖之后，可以直接将 YUV 信息写入缓冲区，硬件能够自动完成 YUV 到 RGB 的转换，从而在 RGB 显示器上显示出来。在不支持YUV 覆盖的显示芯片上，MiniGUI也能够通过软件实现 YUV 覆盖，这时，需要调用 DisplayYUVOverlay 函数将 YUV 信息转换并缩放显示在建立 YUV 覆盖的 DC 设备上。  
　　  
　　MiniGUI 提供的 YUV 覆盖操作函数原型如下：  
　　  
　　/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*YUV overlay support \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
　　/\* 最常见的视频覆盖格式.  
　　\*/  
　　#defineGAL\_YV12\_OVERLAY　0x32315659　　/\* Planar mode: Y + V + U　(3 planes) \*/  
　　#defineGAL\_IYUV\_OVERLAY　0x56555949　　/\* Planar mode: Y + U + V　(3 planes) \*/  
　　#defineGAL\_YUY2\_OVERLAY　0x32595559　　/\* Packed mode: Y0+U0+Y1+V0 (1plane) \*/  
　　#defineGAL\_UYVY\_OVERLAY　0x59565955　　/\* Packed mode: U0+Y0+V0+Y1 (1plane) \*/  
　　#defineGAL\_YVYU\_OVERLAY　0x55595659　　/\* Packed mode: Y0+V0+Y1+U0 (1 plane)\*/  
　　  
　　/\* 该函数创建一个视频输出覆盖  
　　\*/  
　　GAL\_Overlay\*GUIAPI CreateYUVOverlay (int width, int height,  
　　　　　　　　　　Uint32format, HDC hdc);  
　　  
　　/\* 锁定覆盖进行直接的缓冲区读写，结束后解锁 \*/  
　　intGAL\_LockYUVOverlay (GAL\_Overlay \*overlay);  
　　voidGAL\_UnlockYUVOverlay (GAL\_Overlay \*overlay);  
　　  
　　#defineLockYUVOverlay GAL\_LockYUVOverlay  
　　#defineUnlockYUVOverlay GAL\_UnlockYUVOverlay  
　　  
　　/\* 释放视频覆盖 \*/  
　　voidGAL\_FreeYUVOverlay (GAL\_Overlay \*overlay);  
　　#defineFreeYUVOverlay GAL\_FreeYUVOverlay  
　　  
　　/\* 将视频覆盖传送到指定 DC 设备上。该函数能够进行 2 维缩放  
　　\*/  
　　void GUIAPIDisplayYUVOverlay (GAL\_Overlay\* overlay, const RECT\* dstrect);  
　　  
　　有关视频格式的信息，可参见：  
　　  
　　http://www.webartz.com/fourcc/indexyuv.htm  
　　  
　　有关颜色空间的相互关系的息，可参见：  
　　  
　　http://www.neuro.sfc.keio.ac.jp/~aly/polygon/info/color-space-faq.html  
　　  
　　7.2 Gamma 校正  
　　  
　　Gamma 校正通过为 RGB 颜色空间的每个颜色通道设置 Gamma 因子，来动态调整 RGB 显示器上的实际 RGB 效果。需要注意的是，Gamma 校正需要显示芯片的硬件支持。  
　　  
　　应用程序可以通过SetGamma 函数设置 RGB 三个颜色通道的 Gamma 校正值。该函数原型如下：  
　　  
　　intGAL\_SetGamma (float red, float green, float blue);  
　　#defineSetGamma GAL\_SetGamma  
　　  
　　线性 Gamma 校正值的范围在 0.1 到 10.0 之间。如果硬件不支持 Gamma 校正，该函数将返回 -1。  
　　  
　　应用程序也可以通过SetGammaRamp 函数设置 RGB 三个颜色通道的非线性 Gamma 校正值。该函数原型如下：  
　　  
　　intGAL\_SetGammaRamp (Uint16 \*red, Uint16 \*green, Uint16 \*blue);  
　　#defineSetGammaRamp GAL\_SetGammaRamp  
　　  
　　int GAL\_GetGammaRamp(Uint16 \*red, Uint16 \*green, Uint16 \*blue);  
　　#defineGetGammaRamp GAL\_GetGammaRamp  
　　  
　　函数SetGammaRamp 实际设置的是每个颜色通道的Gamma 转换表，每个表由 256 个值组成，表示设置值和实际值之间的对应关系。当设置屏幕上某个象素的RGB 分别为 R、G、B 时，实际在显示器上获得的象素RGB 值分别为：red[R]、green[G]、blue**。如果硬件不支持 Gamma 校正，该函数将返回 -1。**  
  
**函数GetGammaRamp 获得当前的 Gamma 转换表。**  
  
**Gamma 校正的最初目的，是为了能够在显示器上精确还原一副图片。Gamma 值在某种程度上表示的是某个颜色通道的对比度变化。但 Gamma 在多媒体和游戏程序中有一些特殊用途――通过 Gamma 校正，可以方便地获得对比度渐进效果。**  
  
**8 小结**  
  
**本文描述了自MiniGUI 1.1.0Pre4 版本发布以来新增的 GDI 接口。这些接口涉及到曲线和填充生成器、复杂曲线的绘制、封闭曲线填充、复杂区域的创建、直接访问 FrameBuffer、YUV 覆盖和 Gamma 校正等等。通过本文的介绍，相信读者能够对 MiniGUI 的新 GDI 接口有一个更加全面的认识。**  
  
  
**Content**  
**1.驱动**  
**2.应用程序书写**  
  
**1.驱动**  
**Linux-2.4.18默认带有ov511摄像头的驱动，在USB设备中选中，并激活video选项，即可对ov511支持。**  
  
**2.应用程序书写**  
**这是一个在MiniGui下的程序，可以用QVFB模拟，可以可以在板子上跑，效果还可以。**

|  |
| --- |
| 1. v4l.h 2. /\* 3. \* w3v4l.h 4. \* 5. \* Copyright (C) 1998 - 2000 Rasca, Berlin 6. \* EMail: thron@gmx.de 7. \* 8. \* This program is free software; you canredistribute it and/or modify 9. \* it under the terms of the GNU GeneralPublic License as published by 10. \* the Free Software Foundation; eitherversion 2 of the License, or 11. \* (at your option) any later version. 12. \* 13. \* This program is distributed in the hopethat it will be useful, 14. \* but WITHOUT ANY WARRANTY; without eventhe implied warranty of 15. \* MERCHANTABILITY or FITNESS FOR APARTICULAR PURPOSE.  See the 16. \* GNU General Public License for moredetails. 17. \* 18. \* You should have received a copy of theGNU General Public License 19. \* along with this program; if not, write tothe Free Software 20. \* Foundation, Inc., 675 Mass Ave,Cambridge, MA 02139, USA. 21. \*/ 22. #ifndef \_\_W3V4L\_H\_\_ 23. #define \_\_W3V4L\_H\_\_ 24. **class** CV4L{ 25. **public**: 26. CV4L(); 27. CV4L(**char** \*szDevName); 28. ~CV4L(); 29. **bool** init(**int** channel, **int** width, intheight); 30. unsigned **char** \*Read(); 31. **void** destroy(); 32. **private**: 33. **char** \*szDevName; 34. **bool** initialized; 35. unsigned **char** \*m\_Buff; 36. **int** m\_BuffSize; 37. **int** fdVideo; 38. **int** m\_Width, m\_Height; 39. **int** m\_MapSize; 40. }; 41. #endif 42. v4l.cpp 43. /\* 44. \* v4l.c 45. \* 46. \* 47. \* This program is free software; you canredistribute it and/or modify 48. \* it under the terms of the GNU GeneralPublic License as published by 49. \* the Free Software Foundation; eitherversion 2 of the License, or 50. \* (at your option) any later version. 51. \* 52. \* This program is distributed in the hopethat it will be useful, 53. \* but WITHOUT ANY WARRANTY; without eventhe implied warranty of 54. \* MERCHANTABILITY or FITNESS FOR APARTICULAR PURPOSE.  See the 55. \* GNU General Public License for moredetails. 56. \* 57. \* You should have received a copy of theGNU General Public License 58. \* along with this program; if not, write tothe Free Software 59. \* Foundation, Inc., 675 Mass Ave,Cambridge, MA 02139, USA. 60. \*/ 61. #include <stdio.h> 62. #include <stdlib.h> 63. #include <unistd.h> 64. #include <sys/types.h> 65. #include <sys/stat.h> 66. #include <fcntl.h> 67. #include <sys/ioctl.h> 68. #include <sys/mman.h> 69. #include <linux/types.h> 70. #include <string.h> 71. #include <linux/videodev.h> 72. #include "v4l.h" 73. /\* v4l\_init() 74. \* function: init the video device 75. \* references: 76. \*    dev: device name. 77. \*    input: channel number of video\_channel structure. 78. \*    width: width value of video\_window structure 79. \*    height: height value of video\_window structure 80. \*/ 81. **bool** CV4L::init(**int** channel, **int** width, intheight) 82. { 83. **int** fd; 84. **struct** video\_capability vid\_caps; 85. **struct** video\_mbuf vid\_mbuf; 86. **struct** video\_channel vid\_chnl; 87. // open the video device 88. fd = open (szDevName, O\_RDWR); 89. **if** (fd == -1) { 90. perror (szDevName); 91. **return** **false**; 92. } 93. fdVideo = fd; 94. // get video\_capability structrue 95. **if** (ioctl (fd, VIDIOCGCAP, &vid\_caps)== -1) { 96. perror ("ioctl (VIDIOCGCAP)"); 97. **return** **false**; 98. } 99. // get the buffer information in video\_mbufstructure 100. // if can't use mmap() 101. **if** (ioctl (fd, VIDIOCGMBUF, &vid\_mbuf)== -1) { 102. **struct** video\_window vid\_win; 103. m\_MapSize = 0; 104. // set video window information 105. **if** (ioctl(fd, VIDIOCGWIN, &vid\_win) !=-1) { 106. vid\_win.width = width; 107. vid\_win.height= height; 108. ioctl (fd, VIDIOCSWIN, &vid\_win); 109. } 110. } **else** { 111. m\_MapSize = vid\_mbuf.size; 112. m\_BuffSize = m\_MapSize; 113. } 114. #ifdef DEBUG 115. printf ("%s: mbuf.size=%d\n",\_\_FILE\_\_, vid\_mbuf.size); 116. #endif 117. **if** (channel > -1) { 118. vid\_chnl.channel = channel; 119. **if** (ioctl (fd, VIDIOCGCHAN, &vid\_chnl)== -1) { 120. perror ("ioctl (VIDIOCGCHAN)"); 121. } **else** { 122. vid\_chnl.channel = channel; 123. **if** (ioctl (fd, VIDIOCSCHAN, &vid\_chnl)== -1) { 124. perror ("ioctl (VIDIOCSCHAN)"); 125. } 126. } 127. } 128. **if** (m\_MapSize > 0) { 129. m\_Buff = (unsigned **char** \*)mmap(0,m\_MapSize, PROT\_READ|PROT\_WRITE,MAP\_SHARED,fd,0); 130. **if** ((unsigned **char** \*) -1 == (unsigned **char**\*)m\_Buff) { 131. perror ("mmap()"); 132. close (fd); 133. **return** **false**; 134. } 135. } **else** { 136. m\_BuffSize = width \* height \* 3; 137. m\_Buff = (unsigned **char** \*)malloc(m\_BuffSize); 138. } 139. m\_Width = width; 140. m\_Height = height; 141. **return** **true**; 142. } 143. unsigned **char** 144. \*CV4L::Read() 145. { 146. **struct** video\_mmap vid\_mmap; 147. **if** (m\_MapSize == 0) { 148. printf (\_\_FILE\_\_": reading image ..\n"); 149. **if** (read (fdVideo, (**void** \*)m\_Buff,m\_BuffSize) <= 0) { 150. free (m\_Buff); 151. **return** (0); 152. } 153. } **else** { 154. vid\_mmap.format =VIDEO\_PALETTE\_RGB565;  //VIDEO\_PALETTE\_RGB24; 155. vid\_mmap.frame = 0; 156. vid\_mmap.width = m\_Width; 157. vid\_mmap.height= m\_Height; 158. **if** (ioctl (fdVideo, VIDIOCMCAPTURE,&vid\_mmap) == -1) { 159. perror ("ioctl(VIDIOCMCAPTURE)"); 160. **return** (0); 161. } 162. **if** (ioctl (fdVideo, VIDIOCSYNC,&vid\_mmap) == -1) { 163. perror ("ioctl (VIDIOCSYNC)"); 164. **return** (0); 165. } 166. } 167. **return** m\_Buff; 168. } 169. **void** CV4L::destroy() 170. { 171. **if** (fdVideo >= 0) { 172. **if** (m\_MapSize == 0) 173. free (m\_Buff); 174. **else** 175. munmap (m\_Buff, m\_MapSize); 176. close (fdVideo); 177. } 178. } 179. CV4L::CV4L() 180. { 181. // 182. } 183. CV4L::CV4L(**char** \*\_szDevName) 184. { 185. szDevName = (**char**\*)malloc(strlen(\_szDevName)+1); 186. strcpy(szDevName, \_szDevName); 187. //init(0, int width, int height); 188. // 189. } 190. CV4L::~CV4L() 191. { 192. destroy(); 193. } 194. video.cpp 195. /\* 196. \*\* $Id: helloworld.c,v 1.7 2003/06/1307:15:49 weiym Exp $ 197. \*\* 198. \*\* Listing 2.1 199. \*\* 200. \*\* helloworld.c: Sample program for MiniGUIProgramming Guide 201. \*\*     The first MiniGUI application. 202. \*\* 203. \*\* Copyright (C) 2003 Feynman Software. 204. \*\* 205. \*\* License: GPL 206. \*/ 207. #include <stdio.h> 208. #include <minigui/common.h> 209. #include <minigui/minigui.h> 210. #include <minigui/gdi.h> 211. #include <minigui/window.h> 212. #include <string.h> 213. #include <malloc.h> 214. #include "v4l.h" 215. #define VID\_W 320 216. #define VID\_H 240 217. **static** BITMAP bmp; 218. CV4L \*cVid; //"/dev/video0" 219. unsigned **char** \*buf; 220. **static** **int** count = 0; 221. **static** **int** xdir = -3; 222. **static** **int** ydir = -3; 223. **static** **int** x = 1; 224. **static** **int** y = 11; 225. /\*bool SwitchBuf24ToBmp16(unsigned char\*buf24, PBITMAP pBmp) 226. { 227. myBmp.bits = buf24; 228. //ExpandMyBitmap (HDC\_SCREEN, &bmp,&myBmp, rgb, 0); 229. printf ("bmp.bmType = %i\n",bmp.bmType); 230. printf ("bmp.bmBitsPerPixel =%i\n", bmp.bmBitsPerPixel); 231. printf ("bmp.bmBytesPerPixel =%i\n", bmp.bmBytesPerPixel); 232. printf ("bmp.bmAlpha = %i\n",bmp.bmAlpha); 233. printf ("bmp.bmColorKey = %i\n",bmp.bmColorKey); 234. printf ("bmp.bmWidth = %i\n",bmp.bmWidth); 235. printf ("bmp.bmHeight = %i\n",bmp.bmHeight); 236. printf ("bmp.bmPitch = %i\n",bmp.bmPitch); 237. printf ("bmp.bmBits = %i\n",bmp.bmBits); 238. printf ("bmp.bmAlphaPixelFormat =%i\n", bmp.bmAlphaPixelFormat); 239. return true; 240. }\*/ 241. **void** FillBitmap(BITMAP \*bmp) 242. { 243. bmp->bmType = 0; 244. bmp->bmBitsPerPixel = 16; 245. bmp->bmBytesPerPixel = 2; 246. bmp->bmAlpha = 0; 247. bmp->bmColorKey = 0; 248. bmp->bmWidth = VID\_W; 249. bmp->bmHeight = VID\_H; 250. bmp->bmPitch = VID\_W\*2; 251. bmp->bmBits = NULL; 252. bmp->bmAlphaPixelFormat = NULL; 253. } 254. **void** FillMyBitmap(PMYBITMAP my\_bmp) 255. { 256. my\_bmp->flags = MYBMP\_RGBSIZE\_3 | MYBMP\_TYPE\_BGR| MYBMP\_FLOW\_DOWN; 257. my\_bmp->frames = 1; 258. my\_bmp->depth = 24; 259. my\_bmp->alpha = 0; 260. my\_bmp->reserved[0] = 0; 261. my\_bmp->reserved[1] = 0; 262. my\_bmp->transparent = 0; 263. my\_bmp->w = 240; 264. my\_bmp->h = 180; 265. my\_bmp->pitch = 240\*3; 266. my\_bmp->size = 240\*180\*3; 267. my\_bmp->bits = NULL; 268. } 269. **static** **int** HelloWinProc(**HWND** hWnd, intmessage, **WPARAM** wParam, **LPARAM** lParam) 270. { 271. **HDC** hdc; 272. RECT rc; 273. **switch** (message) { 274. **case** MSG\_PAINT: 275. hdc = BeginPaint (hWnd); 276. **if** (bmp.bmBits) 277. FillBoxWithBitmap (hdc, 0,0, bmp.bmWidth, bmp.bmHeight, &bmp); 278. SetBkColor (hdc, RGB2Pixel (hdc, 0xFF, 0xFF, 0xFF)); 279. SetBkMode (hdc, BM\_TRANSPARENT); 280. SetTextColor (hdc, RGB2Pixel (hdc, 0xFF, 0x00, 0x00)); 281. rc.left = 0; 282. rc.top = 0; 283. rc.right = 300; 284. rc.bottom = 60; 285. TextOut (hdc, 0, 0, "??????"); 286. **if**(x >= VID\_W || x <=0) 287. { 288. xdir = 0 - xdir; 289. } 290. **if**(y >= VID\_W || y <=10) 291. { 292. ydir = 0 - ydir; 293. } 295. x+= xdir; 296. y+= ydir; 297. TextOut (hdc, x, y, "kdf"); 298. Rectangle (hdc, 0, 0, bmp.bmWidth, bmp.bmHeight); 299. /\*FillBoxWithBitmap (hdc, 100, 0, 200, 200, &bmp); 300. Rectangle (hdc, 100, 0, 300, 200); 301. FillBoxWithBitmapPart (hdc, 0, 200, 400, 200, 0, 0, &bmp, 10, 10); 302. Rectangle (hdc, 0, 200, 400, 400);\*/ 303. EndPaint (hWnd, hdc); 304. **return** 0; 305. **case** MSG\_CREATE: 306. /\*if (LoadBitmap (HDC\_SCREEN, &bmp, "bkgnd.bmp")) 307. return -1;\*/ 308. **return** 0; 309. **case** MSG\_CLOSE: 310. UnloadBitmap (&bmp); 311. DestroyMainWindow (hWnd); 312. PostQuitMessage (hWnd); 313. **return** 0; 314. } 315. **return** DefaultMainWinProc(hWnd, message, wParam, lParam); 316. } 317. **int** MiniGUIMain (**int** argc, **const** **char**\*argv[]) 318. { 319. MSG Msg; 320. **HWND** hMainWnd; 321. MAINWINCREATE CreateInfo; 322. #ifdef \_LITE\_VERSION 323. SetDesktopRect(0, 0, 1024, 768); 324. #endif 325. CreateInfo.dwStyle = WS\_VISIBLE | WS\_BORDER | WS\_CAPTION; 326. CreateInfo.dwExStyle = WS\_EX\_NONE; 327. CreateInfo.spCaption = "Hello, world"; 328. CreateInfo.hMenu = 0; 329. CreateInfo.hCursor = GetSystemCursor(0); 330. CreateInfo.hIcon = 0; 331. CreateInfo.MainWindowProc = HelloWinProc; 332. CreateInfo.lx = 0; 333. CreateInfo.ty = 0; 334. CreateInfo.rx = VID\_W; 335. CreateInfo.by = VID\_H; 336. CreateInfo.iBkColor = COLOR\_lightwhite; 337. CreateInfo.dwAddData = 0; 338. CreateInfo.hHosting = HWND\_DESKTOP; 340. hMainWnd = CreateMainWindow (&CreateInfo); 342. **if**(hMainWnd == HWND\_INVALID) 343. **return** -1; 344. ShowWindow(hMainWnd, SW\_SHOWNORMAL); 345. ////////////////////////////////////////////////////////////// 346. //1. Create my video class CV4L. 347. cVid = **new** CV4L("/dev/video0"); 349. ////////////////////////////////////////////////////////////// 350. //2. Init the video device with channel and map size. 351. **if**(cVid->init(0, VID\_W, VID\_H) == **false**) 352. **return** -1; 353. //FillMyBitmap(&myBmp); 354. FillBitmap(&bmp); 355. ////////////////////////////////////////////////////////////// 356. //3. Read the data from video device. 357. **if**(buf = cVid->Read ()) 358. { 359. bmp.bmBits = buf; 360. //InvalidateRect (); 361. SendMessage (hMainWnd, MSG\_PAINT, 0, 0); 362. } 363. **while** (**true**) 364. { 365. **if** (!HavePendingMessage (hMainWnd)) 366. { 367. **if** (!GetMessage (&Msg, hMainWnd)) 368. **break**; 369. TranslateMessage (&Msg); 370. DispatchMessage (&Msg); 371. } **else** { 372. ////////////////////////////////////////////////////////////// 373. // 3. Read the data from video device. 374. **if** (buf = cVid->Read ()) 375. { 376. bmp.bmBits = buf; 377. SendMessage (hMainWnd,MSG\_PAINT, 0, 0); 378. } **else** { 379. // if Buffer is Null, vedeodevice have pluged out. 380. PostQuitMessage (hMainWnd); 381. } /\* end of read video buffer \*/ 382. } /\* end of HavePendingMessage() \*/ 383. }/\* end of while \*/ 384. ////////////////////////////////////////////////////////////// 385. //4. destr oy the CV4L class, and release resources. 386. cVid->destroy (); 387. UnloadBitmap (&bmp); 388. //free (bmp.bmBits); 389. MainWindowThreadCleanup (hMainWnd); 390. **return** 0; 391. } 392. #ifndef \_LITE\_VERSION 393. #include <minigui/dti.c> 394. #endif |

**把这些代码添加到miniugi-1.3.3的helloworld程序中去编译一下就可以了，也可以用如下命令**  
**g++ -o video -I/minigui/include-L/minigui/lib video.cpp v4l.cpp**