Psychtoolbox

教程

ZJU 心理系2012年短学期某小组整理翻译

李冠婕 田波 温雪琦 叶圣炉 张皓 朱焓泓

（按拼音排序）

建议在学习了 matlab 的基础以后再来学习 PTB，matlab 教程网上可以找很多。

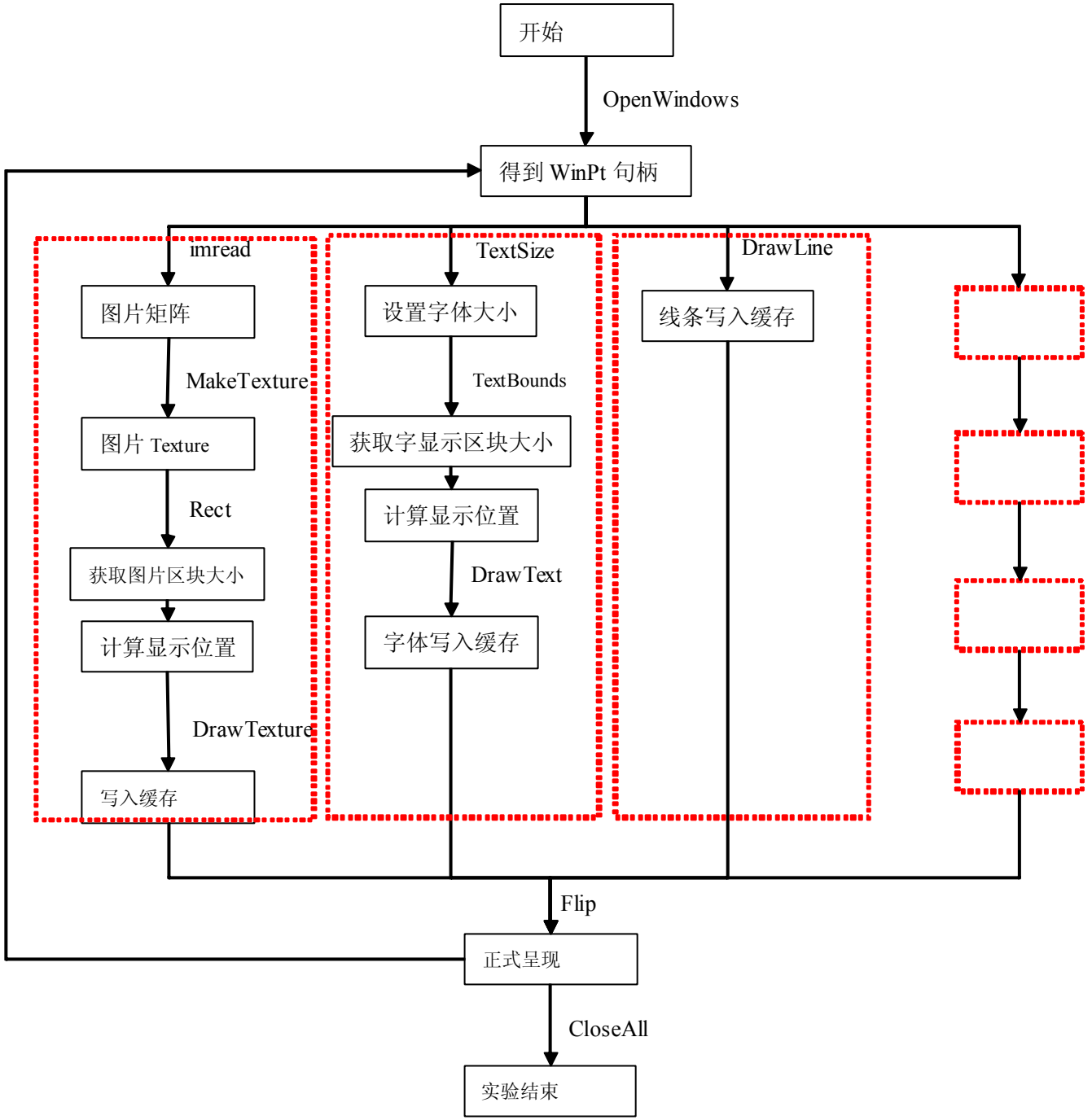
本文并不能教会你 PTB，本文只是提供一个基本的思路和一个框架，让你更好地理解 PTB，并更方便地检索。

```
win_bk_color=[125,125,125];

[winPt, winRect]=Screen('OpenWindow',0,win_bk_color);

a=imread('imgname.jpg');
b=Screen('MakeTexture',winPt,a);
Screen('DrawTexture',winPt,b);

Screen('Flip',winPt);
WaitSecs(2);
Screen('CloseAll');
return
```



以上脚本可以呈现一张图片，它用红色虚线框框起的部分可以用右边红色虚线框的内容进行替换。通过改变红色虚线框的内容，一般刺激呈现都可以实现。

如果需要计时，使用 toc 和 tic

如果需要记录按键，可以使用 KbCheck 扫描键盘

函数大全

% 打开窗口或Texture

[windowPtr,rect]=Screen('OpenWindow',windowPtrOrScreenNumber); 详细
[windowPtr,rect]=Screen('OpenOffscreenWindow',windowPtrOrScreenNumber);详细
textureIndex=Screen('MakeTexture', WindowIndex, imageMatrix);详细
Screen('Close');详细
Screen('CloseAll');详细

% 画线或者填充的形状

currentbuffer = Screen('SelectStereoDrawBuffer', windowPtr);详细
Screen('DrawLine', windowPtr [,color], fromH, fromV, toH, toV);详细
Screen('DrawArc',windowPtr,[color],[rect],startAngle,arcAngle)详细
Screen('FrameArc',windowPtr,[color],[rect],startAngle,arcAngle)详细
Screen('FillArc',windowPtr,[color],[rect],startAngle,arcAngle)详细
Screen('FillRect', windowPtr);详细
Screen('FrameRect', windowPtr);详细
Screen('FillOval', windowPtr);详细
Screen('FrameOval', windowPtr);详细
Screen('FramePoly', windowPtr);详细
Screen('FillPoly', windowPtr);详细

% 在窗口中输出字

textModes = Screen('TextModes');详细
oldCopyMode=Screen('TextMode', windowPtr);详细
oldTextSize=Screen('TextSize', windowPtr);详细
oldStyle=Screen('TextStyle', windowPtr);详细
[oldFontName,oldFontNumber]=Screen(windowPtr,'TextFont');详细
[normBoundsRect, offsetBoundsRect]= Screen('TextBounds', windowPtr, text);详细
[newX,newY]=Screen('DrawText', windowPtr, text);详细

oldTextColor=Screen('TextColor', windowPtr);详细
oldTextBackgroundColor=Screen('TextBackgroundColor', windowPtr);详细

% 复制图像，快，在Texture、幕后屏幕和当前的窗口中
A =IMREAD(FILENAME,FMT) 读入图片，详细
[resident [texidresident]] = Screen('PreloadTextures', windowPtr [, texids]);详细
Screen('DrawTexture', windowPointer, texturePointer);详细
Screen('DrawTextures', windowPointer, texturePointer(s));详细
Screen('CopyWindow', srcWindowPtr, dstWindowPtr)详细

% 复制图像，慢，在矩阵与窗口中
imageArray=Screen('GetImage', windowPtr); 详细

% 刷新当前窗口
Screen('Flip', windowPtr);详细
Screen('AsyncFlipBegin', windowPtr);详细
Screen('AsyncFlipEnd', windowPtr);详细
Screen('AsyncFlipCheckEnd', windowPtr);详细
Screen('WaitUntilAsyncFlipCertain', windowPtr);详细
Screen('GetFlipInfo', windowPtr);详细
Screen('DrawingFinished', windowPtr);详细
Screen('WaitBlanking', windowPtr);详细

%获取鼠标和键盘输入
[x,y,buttons,focus,valuators,valinfo] = GetMouse()详细
[keyIsDown, secs, keyCode, deltaSecs] = KbCheck()详细

%时间
wakeup=WaitSecs(s) 详细
TSTART =TIC 详细
T =TOC 详细

%文件操作
FID =FOPEN(FILENAME,PERMISSION)详细
[A,COUNT] =FSCANF(FID,FORMAT,SIZE)详细
FPRINTF(FID, FORMAT, A, ...)详细
ST =FCLOSE(FID)详细

详细介绍

[windowPtr,rect]=Screen('OpenWindow',*windowPtrOrScreenNumber* [,*color*]
[,*rect*][,*pixelSize*][,*numberOfBuffers*][,*stereomode*][,*multisample*][,*imagingmode*][,*specialFlags*]);
返回

作用：
打开一个显示屏幕。

参数：
WindowRtrOrScreenNumber：窗口句柄，指定屏幕，0代表带菜单栏的主屏幕。
Color：像素的颜色（标量或[r g b]），默认为白色。
Rect：矩形，至少含有一个像素。若窗口句柄为*WindowRtr*，rect在窗口坐标中（初始点为左上），默认为整个窗口；若为*ScreenNumber*，*rect*在屏幕坐标中，默认为整个屏幕。
PixelSize：每个像素的深度（bits），默认深度不变。
NumberOfBuffers：使用缓冲数，设置为大于2有助于PTB的发展与调试，但会破坏实际实验。
Stereomode：使用的立体显示算法类型，0（默认）表示单视场观察。
Multisample：启用显卡的自动抗锯齿化处理，颜色采样，数字越大，显示质量越高，但消耗更多的内存且由于更高的计算需求使得帧数率的减少。
Imagingmode：启用PTB内部的图像处理通道，默认为关闭。
SpecialFlags：木有。

返回

[windowPtr,rect]=Screen('OpenOffscreenWindow',*windowPtrOrScreenNumber* [,*color*] [,*rect*] [,*pixelSize*] [,*specialFlags*] [,*multiSample*]);
返回

功能：
打开一个幕后（offscreen）窗口。

参数：
WindowRtrOrScreenNumber：窗口句柄，指定屏幕，0代表带菜单栏的主屏幕。
Color：每个像素初始背景颜色（标量或[r g b]或[r g b a]），默认为白色。
Rect：幕后屏幕的尺寸，至少包含一个像素。如果是windowPtr，那么该参数指整个窗口；若是screenNumber，该参数指整个屏幕；若screenNumber=-1，参数指主屏幕的尺寸。
PixelSize：每个像素的深度（bits）。
SpecialFlags：特殊性能，默认为0。若设为2，幕后窗口精度较高。
Multisample：用以抗锯齿绘制的采样数，默认为0。

返回

---

textureIndex=Screen('MakeTexture', *WindowIndex*, *imageMatrix* [, *optimizeForDrawAngle*=0] [, *specialFlags*=0] [, *floatprecision*=0] [, *textureOrientation*=0] [, *textureShader*=0]);
返回

功能：
将二维或三维的模型'*imageMatrix*'转换为开放图形语言（OpenGL）结构，并返回一个指针（index），用于'DrawTexture'，表示此结构。

参数：
imageMatrix：由单色平面（plane），图层平面，RGB三色平面或RGBA平面构成。
optimizeForDrawAngle=0：请求psychtoolbox优化结构，尤其是在某个特定的旋转角度快速绘制。默认为0，表示垂直绘制的优化。
specialFlags=0：系统默认算法，2为PTB的高质量的滤波算法，4为更快的结构创作方法。
floatprecision=0：定义了结构被储存和执行的精度。默认值为0，储存精度为每个颜色成分8 bit，通过imread()读取。
textureOrientation=0：给结构打上特殊的orientation。

textureShader=0：可选，默认关闭GLSL shader 程序。

返回

Screen('Close', [*windowOrTextureIndex or list of textureIndices/offscreenWindowIndices*]);
返回

作用：
关闭一个幕前（onscreen）或幕后（offscreen）的窗口或一个结构。若不提供*windowOrTextureIndex*，所有的幕后的窗口都将关闭，只留下幕前的窗口。如果你想关闭部分幕后的窗口或结构，可以将想关闭的窗口handles放入一个vector中。

返回

Screen('CloseAll');
返回

作用：
关闭所有打开的屏幕前、屏幕后的窗口、结构、电影和视频资源。释放几乎所有资源。

返回

---

currentbuffer = Screen('SelectStereoDrawBuffer', *windowPtr* [, *bufferid*] [, *param1*]);
返回

作用：
在立体声显示模式下选择绘制命令的缓冲区，或者返回当前或旧的缓冲区选择，存入*'currentbuffer'*之中。（此功能仅适用于立体声（stereo mode））

参数：
windowPtr：onscreen的立体声窗口所对应的指针（pointer）。
bufferid：0表示选择左眼缓冲区，1表示选择右眼缓冲区。此命令须在每个Screen('Flip')命令之后或重建缓冲选择之后调用。
param1：此参数的意义取决于活跃的立体声模式。

返回

Screen('DrawLine', *windowPtr* [,*color*], *fromH*, *fromV*, *toH*, *toV* [,*penWidth*]);
返回

功能：
画一条线。

参数：
windowPtr：窗口句柄。
color：每个像素的颜色（标量或[r g b a]矢量），默认为黑色。
fromH：初始的x坐标。
fromV：初始的y坐标。
toH：结束的x坐标。
toV：结束的y坐标。
penWidth：默认为1。

返回

Screen('DrawArc',*windowPtr*,[*color*],[*rect*],*startAngle*,*arcAngle*)
返回

作用：
画一条与矩形内切的弧线。

参数：
color,等级数值或RGB颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。
startAngle, *arcAngle*弧线的角度从垂直线顺时针度量。

返回

Screen('FrameArc',*windowPtr*,[*color*],[*rect*],*startAngle*,*arcAngle*, [*penWidth*], [*penHeight*], [*penMode*])
返回

作用：
画一条和矩形内切的弧线。

参数：
color,等级数值或RGB颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。
startAngle, *arcAngle*角度从垂直线顺时针度量。
penWidth、*penHeight*是画线用的笔的长度和宽度，即定义画笔粗细。在OS-X系统中，画笔宽度和高度必须相等，并且*penMode*（画笔模式）一般会忽略。

与楼上的区别可能是拥有更多的设置吧~

返回

Screen('FillArc',*windowPtr*,[*color*],[*rect*],*startAngle*,*arcAngle*)
返回

作用：
画一条和矩形内切的实心圆弧。

参数：
color,等级数值或RGBA颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。
startAngle, *arcAngle*角度从垂直线顺时针度量。

返回

Screen('FillRect', *windowPtr* [,*color*] [,*rect*])

返回

作用：
填充一个或多个矩形。

参数:
color,等级数值或RGB或RGBA颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。

说明：
这个函数可以用来清空窗口。
除了填充一个矩形，这个函数还可以用来填充定义好的一系列矩形，这比一个个画快多了。
填充n个矩形方法：定义*rect*为n列 4行的矩阵，每一列定义一个矩形，比如rect(1,5) =第五个矩阵的左边界，rect(2,5) =第五个矩阵的上边界，rect(3,5) =第五个矩阵的右边界，rect(4,5) =第五个矩阵的下边界。
颜色，如果想让每个矩形颜色不同，则定义*color*为n列3(RGB)或4(RGBA)行的矩阵，在第i列定义第i个矩阵的颜色。

返回

Screen('FrameRect', *windowPtr* [,*color*] [,*rect*] [,*penWidth*]);
返回

作用：
画一个或多个矩形框。

参数：
color,等级数值或RGB或RGBA颜色矩阵。空着则为系统默认值，即黑色。
空着则为系统默认值，即标准颜色对照表中的黑色。
*rect*系统默认值为整个窗口。

说明：
除了画一个矩形框，这个函数还可以用来画定义好的一系列矩形框，这比一个个画快多了。

画n个矩形框方法：定义*rect*为n列 4行的矩阵，每一列定义一个矩形框，比如rect(1,5) =第五个矩阵框的左边界，rect(2,5) =第五个矩阵框的上边界，rect(3,5) =第五个矩阵框的右边界，rect(4,5) =第五个矩阵框的下边界。
颜色，如果想让每个矩形框颜色不同，则定义*color*为n列3(RGB)或4(RGBA)行的矩阵，在第i列定义第i个矩阵框的颜色。
penWidth用来定义画笔及矩形框线条的粗细，可以所有矩形框使用一种画笔，或者用矩阵定义每个矩形框使用不同画笔。penWidth的默认值为1像素单元。

返回

Screen('FillOval', *windowPtr* [,*color*] [,*rect*] [,*perfectUpToMaxDiameter*]);
返回

作用：
用*color*定义的颜色填充一个或多个内切*rect*的椭圆。

参数：
color,等级数值或RGB颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。

说明：
除了可以给一个椭圆填充颜色，这个函数还可以用来填充一系列定义好的椭圆，整个比一个个填快多了。
填充n个椭圆方法：定义*rect*为n列 4行的矩阵，每一列定义一个椭圆，比如rect(1,5) =第五个椭圆的左边界，rect(2,5) =第五个椭圆的上边界，rect(3,5) =第五个椭圆的右边界，rect(4,5) =第五个椭圆的下边界。
颜色，如果想让每个椭圆颜色不同，则定义*color*为n列3(RGB)或4(RGBA)行的矩阵，在第i列定义第i个椭圆的颜色。
任选参数*perfectUpToMaxDiameter*定义所有椭圆的最大直径，单位为1个像素。其默认值为全屏，这样所有的椭圆都会完整，当然这样速度就会变慢。如果您知道您想画的椭圆的直径不会超过某个具体值，您可以将该值作为参数值使填充速度更快。【不是很懂这个参数，这样对吗？】

返回

Screen('FrameOval', *windowPtr* [,*color*] [,*rect*] [,*penWidth*] [,*penHeight*] [,*penMode*]);
返回

作用：

画一个或多个内切*rect*的椭圆框。

参数：
color,等级数值或RGBA颜色矩阵。空着则为系统默认值，即黑色。
*rect*系统默认值为整个窗口。
画笔，默认的*penWidth*和*penHeight*都为1像素单元，*penMode*通常会忽略。在OS-X系统中，画笔宽度和高度必须相等，如果设定不同的值，FrameOval函数会自动选取较大的那个数值作为画笔宽度。对于非圆的椭圆画笔的宽度在各个区域不一致，这是一个众所周知的缺点。

说明：
除了可以画一个椭圆框，这个函数还可以用来画充一系列定义好的椭圆框，整个比一个个填快多了。
填充n个椭圆框方法：定义*rect*为n列 4行的矩阵，每一列定义一个椭圆框，比如rect(1,5) =第五个椭圆框的左边界，rect(2,5) =第五个椭圆快的上边界，rect(3,5) =第五个椭圆框的右边界，rect(4,5) =第五个椭圆框的下边界。
颜色，如果想让每个椭圆框颜色不同，则定义*color*为n列3(RGB)或4(RGBA)行的矩阵，在第i列定义第i个椭圆框的颜色。
画笔，如果所有椭圆框一次画成，则*penHeight*和*penMode*会被忽略，都用*penWidth*的值来定义各个椭圆框的线条。同上，可以设定一个*penWidth*的值画所有椭圆框，也可以用矩阵给各个椭圆框定义一个*penWidth*。

返回

Screen('FramePoly', *windowPtr* [,*color*], *pointList* [,*penWidth*]);
返回

作用：
画一个多边形（轮廓）。

参数：
color：是你想往每个像素里填充的颜色参数（标量或RGBA向量），为空（默认）是白色。
pointList：是一个矩阵，每一行指定一个顶点的（x，y）坐标。

返回

Screen('FillPoly', *windowPtr* [,*color*], *pointList* [, *isConvex*]);

返回

作用：
填满一个多边形（包括内部）。
参数：
color：是你想往每个像素里填充的颜色参数（标量或RGBA向量），为空（默认）是白色。
pointList：是一个矩阵，每一行指定一个顶点的（x，y）坐标。
*isConvex:*允许你告诉程序多边形是凸还是凹（凸：值为1，凹：值为0），从而使程序可以跳过多边形凹凸性测试，从而在对时间敏感的脚本中节省一点计算时间。
说明：
画一个填满的多边形是一项相当复杂和缓慢的计算机操作。一般来说，画凸多边形是相当快的，画非自相交的多边形就要慢很多(> 30x slower)，画自相交的多边形就非常慢了，甚至慢2000倍。如果你要用很多点画很不规则的多边形，在某种程度上进行预处理，或者在提交给'FillPoly'前把它们分解成一系列凸的或者规则的多边形，可能是一个好主意。另外，为了优化你想要的类型的多边形的绘画，你可以使用一些自定义编写的绘图功能。

返回

---

textModes = Screen('TextModes');
返回

作用：
返回一个以文本模式正确命名的字符串单元阵列。
注意：
在目前的Psychtoolbox中，没有一个模式是被支持的，所以无论设置了什么文本模式，都会被忽略并且无效。
（喂，这个很坑爹有木有。。。）

返回

---

oldCopyMode=Screen('TextMode', *windowPtr* [,*textMode*]);
返回

作用：

为指定的窗口设置或获得文本模式。
说明：
这个功能目前还没有任何效果。甚至连不同的文本模式会有什么不同种类的效果，这个特性是否会被实现都还不明确。（个人认为这里的特性是指不同文本模式有不同效果）

返回

---

oldTextSize=Screen('TextSize', *windowPtr* [,*textSize*]);
返回

作用：
为指定窗口读取/设置字体大小。
说明：
你可以通过调用Screen('Preference', 'DefaultFontSize')为新的窗口指定一个默认字体大小。最初的默认字体大小取决于操作系统。
返回

---

oldStyle=Screen('TextStyle', *windowPtr* [,*style*]);
返回
作用：
为指定窗口内将会出现的文本获得/设置字体风格。
说明：
追求个人风格的实用参数，它们可能是OR'd（不理解这里是什么意思）。在计算机TextFace()里可以发现更多。在M$-Windows和GNU/Linux系统中，这些设置只有一部分是值得称道的-所有设置都被认可，但是其中一些在Windows和Linux中会被自动忽略。在Windows系统中，默认字体风格是加粗，这是因为可以增强可读性。
各数值（代号）代表的字体风格：0=标准,1=加粗,2=斜体,4=下划线,8=轮廓（可能是中空）,32=浓缩（缩小）,64=伸展。
通过调用Screen('Preference', 'DefaultFontStyle')函数，你可以为新窗口设置默认字体风格。最初的默认字体风格取决于操作系统。
不是所有的字体都支持全部风格设置的。当前字体不支持的风格会被自动忽略。
返回

[oldFontName,oldFontNumber]=Screen('TextFont', windowPtr [,fontNameOrNumber]);
返回
作用：

为将要出现在当前窗口的文本获得/设置字体。

说明：
你可以询问当前的字体，或者通过数值（代号）或名称（例如：'Helvetica'）指定所需的字体。但是每一台计算机字体代号是不一致的，并且在MS-Windows和Linux系统中，用代号表示的方式是不被支持的，还会被自动忽略，所以为了可靠性和可移植性请使用字体名称。字体代号仅在OS-9 Psychtoolbox和向后兼容的老版本中可以使用。

字体名称可以是一串最大长度为255的字符串，例如'Helvetica'，也可以是包含一个最长为255字符的字符串的列表，例如{'Helvetica'}。默认字体取决于操作系统，并且是为了（使程序）有更强的可读性被挑选出来的。你可以通过调用Screen('Preference', 'DefaultFontName')函数来进行查询或更改操作。

在OS/X中请求一个不存在的字体是被允许的，但是不会有任何作用。如果你介意，可以再次调用TextFont，以便查明你是否成功使用你想要的字体。看字体样本。

在Linux系统中，你既可以提供一个字体名称（PTB将选择一个具有该名称，并且同时满足大小和风格需求的匹配字体），也可以提供一个编码各种字体属性的完整的X-Windows字体说明符字符串。在Linux下的xfontsel-print命令允许您查询所有可用的字体，并且根据需求提供一个规范字符串。根据选中文本的渲染器（个人认为是指文本中字体的各种效果），Linux可以挑出提供的字体；如果你需要的是不存在的字体，DrawText命令会运行失败并弹出错误信息。然而，在Linux系统中使用默认FTGL文本渲染器，在字体选择中Linux的选择标准将稍微宽松。

返回

[*normBoundsRect*, *offsetBoundsRect*]= Screen('TextBounds', *windowPtr*, *text* [,*x*] [,*y*] [,*yPositionIsBaseline*] [,*swapTextDirection*]);
返回

作用:
相当于设置热区，当光标经过窗口的某一区域，将会显示*text*'字符串.

参数:
normBoundsRect 用于定义显示字符串（用像素表示）的边界框大小;
offsetBoundsRect 用于定义热区，当光标经过热区时会显示字符串.

说明:
只有用缺省默认采用的高品质文本框(应该是指不要设置文本框参数) 才能给出恰好合适的文本框. 可供选择的快速文本框品质较低, 可能会导致转行之后的字符显示不出来(超出文本框显示的区域.) 调用参数所要求的字符串格式及其他参数请见Screen('DrawText') . (即下面这条函数)
返回

[*newX*,*newY*]=Screen('DrawText', *windowPtr*, *text* [,*x*] [,*y*] [,*color*] [,*backgroundColor*] [,*yPositionIsBaseline*] [,*swapTextDirection*]);
返回

作用:
显示文字.
参数:
用于定义所显示文字的位置与色彩.
x、*y*定义文本框的左上角, 缺省默认使用之前一条显示文字命令中所用的坐标, 如果之前没有显示文字的命令则使用启动代码中定义的(0, 0) .
" color" 采用标准色彩表上定义的数字定义所显示的文字色彩(标准色彩代码或RGB三通道或RGB+Alpha四通道) , 缺省默认为黑色.
*yPositionIsBaseline*将该参数设为1即可将默认的横排文字设置为纵排文字, 此时y值用于标定文本框的位置(没有设定y值则缺省默认为顶部) . 纵排文字将会使所有默认的文字特征缺省设置不成立.
*swapTextDirection*将该参数设为1可使文字顺序从默认的从左到右设置为从右到左.
newX、*newY* 用于返回文本框的右下角.
*backgroundColor*定义文字背景色彩, 缺省为透明(即显示当前窗口已经保存, 正在呈现的图象) 设定文字背景色彩需要精确设定*backgroundColor*的参数值, 如果需要用Alpha通道混合色彩, 需调用函数Screen('Preference','TextAlphaBlending',1); 以及Screen('Blendfunction', …) ;. 注意文字与其背景在不同的操作系统上或者在不同的文本框显示效果可能是不同的, 甚至根本不支持显示, 所以不要对文字显示功能寄予太高希望… …
说明:
可显示的文字包括双字节字符编码的字符(如中文) . 标准的Matlab用字符(8位字符) 字符串会按照当前设定的文字代码换算方式显示文字, 缺省默认采纳用户操作系统语言环境所用的文字代码系统. 如欲改变文字代码系统, 可调用Screen('Preference', 'TextEncodingLocale',newencoding) 例如, 欲使用统一码转换格式(UTF-8)的多字节字符编码, 调用Screen('Preference', 'TextEncodingLocale', 'UTF-8'); 如果所用字符串不是ASCII编码, 建议在用本函数调用该字符串之前, 将其转换为8位字符型的数据, 以免Matlab将其显示为乱码.
如欲直接调用包含双字节编码字符的字符串, 需要将文字转化为双精度的矩阵, 如mytext = double(myunicodetext); 再用本函函数用这一双精度矩阵, 屏幕即可将双精度数据直接显示为双字节编码的字符.
如果采用缺省(自动生成)的文字边界框, 任何操作系统均支持双字节编码字符的显示. 当然也可以自定义字体, 但须注意并非所有字体都包含双字节编码的字符.
如果选择快速边界框, 由于其品质较低上述防止编码格式混乱而造成乱码的措施将无法采用, 双字节编码字符将无法显示. 但该边界框最大的优点是显

示迅速. 可以通过在程序开头调用Screen('Preference','TextRenderer',0); 函数设置快速边界框.
另外, Screen('Preference', ...); 可用于设定显示文字的诸多属性, 如设置Alpha通道与防乱码模式.
选择文本框: 调用Screen('Preference','TextRenderer',Type); . Type赋值代表文本框类型.
0: 快速文本框, 不支持汉字, 防乱码模式与其它文字特征设置, 但是速度快, 可用于快速显示文字.
1: 高品质文本框: 显示慢, 但支持汉字, 防乱码模式与其他文字特征设置, 如居中, 下划线等. (Windows系统只支持上述两种文本框)

返回

---

oldTextColor=Screen('TextColor', *windowPtr* [,*colorVector*]);
返回

作用:
为特定窗口读取或设置文字颜色.

---

oldTextBackgroundColor=Screen('TextBackgroundColor', *windowPtr* [,*colorVector*]);
返回

作用:
为特定窗口读取或设置文字背景颜色.
参数:
缺省默认背景色为[0,0,0,0], 即全黑, 与窗口背景色相同, 相当于透明. 欲设置非透明文字背景色至少需使Alpha通道颜色取值非零. 定义文字Alpha通道可调用函数Screen('Preference', 'TextAlphaBlending', 1); 以显示文字背景色.

返回

[*resident* [*texidresident*]] = Screen('PreloadTextures', *windowPtr* [, *texids*]);
返回

作用:
调用此函数把要显示的文字写入内存, 以便用快速文本框显示文字. 该函数可以在Trial开始之前存入待显示文字以节省内存调用的时间延迟.
参数:
windowPtr 用于定义将要显示预存文字的窗口.

*texids*用于写入全部需要预存的句柄，用逗号隔开（写成一个矢量的格式。）如果不设置该参数，PTB自动写入全部文字。
*resident*返回是否全部指定的句柄均被写入，返回值为1表示全部成功写入。
texidresident 返回各段被写入文本的具体文字内容。
说明：内存不足可能导致文本写入失败。
返回

Screen('DrawTexture', *windowPointer*, *texturePointer* [,*sourceRect*] [,*destinationRect*] [,*rotationAngle*] [, *filterMode*] [, *globalAlpha*] [, *modulateColor*] [, *textureShader*] [, *specialFlags*] [, *auxParameters*]);
返回

作用：
在指定窗口（用'windowPointer' 定义）显示指定文字（用' texturePointer' 定义）。
参数：
*sourceRect*用于指定文本框中将要显示的部分文字（按矩形的格式定义，缺省默认为整个文本框）
*destinationRect*用于定义指定窗口中将要用于呈现文字的部分（缺省为屏幕中央）。
*rotationAngle*定义旋转角，缺省默认不旋转。
*filterMode*定义文字被放大/缩小/移动后，计算各像素点颜色的方式（例如*sourceRect*与*destinationRect*，大小不一致，或*sourceRect*中定义了某些象素点的值）。（可以赋值为0和1，缺省默认值为1。这里面计算像素点的方式属于专业术语，不敢妄自翻译，通常情况下应该不用设置这一参数。）
*globalAlpha*用于定义全部所显示文字的透明度（Alpha值）。可取值区间为0 ~ 1（完全透明至完全不透明）。如果Alpha通道值与文字透明度Alpha值均被给出，则最终呈现效果的Alpha值是上述两者的乘积。
*modulateColor*用于定义所显示的全部文字的颜色（包括每个通道的数值。因此如果设置此参数，将自动无视*globalAlpha* 参数。）比如，定义为[128 255 0] 即表示设置一个RGBA四通道颜色，其G和Alpha通道为最大值，而B为空，R为一半。你可以先用任意参数值生成所要的文字字形，生成文字之后再调用*modulateColor* 将其设置为你需要的颜色（不一定要在设定文字时就给出准确的颜色设定。）
textureShader：可省略的参数，用于给动态显示的文字（要求该文字仅呈现一次）添加着色渲染功能，比如给一条滚动的字幕添加扫描效果或遮挡效果等。如果之前已经通过Screen('MakeTexture') 给文本设置了渲染，或者PTB已经自动生成了渲染效果，则在这里调用的渲染参数无效。如果给需要多次呈现的文字添加渲染效果，或者需要呈现更复杂的视觉效果，最好调用Screen('TransformTexture') 指令，可以添加更复杂的效果，应用也更灵活。
specialFlags：可选指令，用于定义之前已经设置好的文字呈现参数中，有哪些在呈现本条文字时予以采用。比如将此处的值设为kPsychUseTextureMatrixForRotation" ，表示'dstRect' 文本框是正立的，但其中的文字是旋转后的效果（如果是'srcRect' ，表示文本框与其中的文字一起旋转。）又如将此处的值设为kPsychDontDoRotation，则之前设置过的旋转角将不会用于旋转当前要呈现的文字，但是其他没有调用*specialFlags'* 的文本仍然会呈现为旋转后的效果。

auxParameters: 可选指令，要求格式为把参数写成四维矢量或四的倍数（目前最多可支持32维矢量.）如果程序中要显示的文字想设太多参数以至于其他的指令当中存不下这么多数据,可以先调用该函数将诸多参数按数组的形式存起来（即保存为’auxParameter0…n’）具体参见’helpProceduralShadingAPI’ .

说明:

如欲在同一窗口上显示多条文本（包括当前显示窗口和当前未显示的窗口），须调用Screen（‘DrawTextures’）函数. 该函数可设置的参数与本函数相同，但可把同样的参数设置一次性赋给多条文本.

返回

Screen('DrawTextures', *windowPointer*, *texturePointer*(s) [, *sourceRect*(s)] [, *destinationRect*(s)] [, *rotationAngle*(s)] [, *filterMode*(s)] [, *globalAlpha*(s)] [, *modulateColor*(s)] [, *textureShader*] [, *specialFlags*] [, *auxParameters*]);

返回

作用：

一次性绘制许多textures 或者把一个texture 绘制在多个地方。

参数：

这个函数接收的参数与Screen('DrawTexture') 相同，只是它专门为显示多个textures 优化过。

说明：

省略一些参数时，被省略的函数将使用默认的设置。
参数只提供一个值时，这个值会应用于所有的绘制。
通过向量或者矩阵可以为参数提供多个值，可以完成多个绘制。
如果你要进行多个绘制，那么每个参数的数量需要匹配。

例:

a）把一个texture 绘制在不同地方和不同方向：提供一个texture 句柄作为*texturePointer* ，提供一个4行n列的矩阵作为*destinationRect* ，这矩阵是n个地点的目标矩形，提供一个n 维向量作为*rotationAngles* ，这个向量中装着n个要绘制的texture 的方向。
b）n个texture 绘制在n 个地方：同a），只是提供一个有n个元素的向量作为*texturePointers* ，对应的texture 会被以对应的角度绘制在对应的位置。

返回

Screen('CopyWindow', *srcWindowPtr*, *dstWindowPtr*, [*srcRect*], [*dstRect*], [*copyMode*])
返回

作用：
在on-screen与off-screen间复制图片。

参数：
srcWindowPtr 源窗口句柄
dstWindowPtr 目标窗口句柄
srcRect 源矩形，默认是srcWindowPtr 的窗口大小
dstRect 目标矩形，默认是dstWindowPtr 的窗口大小
copyMode 可以自定，但经常忽视它。

说明：
如果你真的想让快速地复制图像，使用MakeTexture 或DrawTexture 吧，他们也允许旋转绘制和更进一步的混合操作。
目前的CopyWindow还有一些限制——不可以在同一个off-screen 窗口间复制图像，不可以在不同的onscreen 窗口间复制。Onscreen向Offscreen 复制时，源矩形与目标矩形的大小要一致。
返回

---

imageArray=Screen('GetImage', *windowPtr* [,*rect*] [,*bufferName*] [,*floatprecision*=0] [,*nrchannels*=3])
返回

作用：
慢速地从窗口或texture复制一幅image 到Matlab 或Octave ，默认返回值是一个uint8 向量。

参数：
*imageArray*默认返回的*imageArray*有3层，是一张RGB图片。Matlab/Octave 中对uint8做运算会发生错误，需要用double来进行转换。比如“imageArray/255”会产生一个错误，但是“double(imageArray)/255”就不会。参见Screen('PutImage')和Screen('CopyWindow')。
windowPtr onscreen窗口、offscreen窗口或者textrue的句柄，其中的图片会被作为返回值。
rect 要复制的矩形分区域，默认值是整个窗口。
bufferName 一个字符串，这个字符串指定了要复制图片的缓冲区。对于offscreen窗口和texture来说，bufferName没有意义，会被忽略。对于onscreen

窗口，默认值是'frontBuffer',也就是说，这个函数会返回那一刻被试所看到的东西。如果立体帧序列（frame-sequential stereo ）开启，那么 'frontLeftBuffer' 会返回被试左眼看到的东西，'frontRightBuffer' 返回他右眼看到的。如果双缓冲（double-buffering）开启的话，'backBuffer' 可以获得被试在下一次运行Screen('Flip')时所看到的图像；如果此时立体帧序列也被开启，类似地，你也可以用'backLeftBuffer'与 'backRightBuffer'。'aux0Buffer' - 'aux3Buffer'分别返回OpenGL AUX缓冲区0到3。只有在你知道这是什么意思是才能查询AUX 缓冲，否则你的脚本很容易崩溃。这是在PTB的debug中尤为重要。如果图片管道（imagingpipeline ）开启，你也可以调用'drawBuffer'来返回没加工的缓冲器。
floatprecision 如果你把它设为1，那么返回的图片数据就不是uint8类型了，而是double的矩阵。有一点值得注意的是返回的矩阵式0.0到1.0范围的，而不是原来的0-255。使用浮点数来读取，这只在texture、offscreen窗口或者帧缓冲是浮点数，并且图片管道被激活以及HDR模式被选中是才有利。
nrchannels 是颜色通路的数字，默认是3通道（RGB）。1是只有红色/亮度，2是红色与绿色/亮度+Alpha，3是RGB，4是RGBA。

说明：
在Screen('AsyncFlipBegin')造成异步翻转的等待的时侯，不可以对onscreen窗口调用这个函数！要先完成翻转。尽管texture、onscreen和offscreen窗口异步翻转是也可以回读（readback），但是不提倡这样做，因为这会对表现有影响。

返回

Screen('*PutImage*', *windowPtr*, *imageArray* [,*rect*])
返回

作用：
把*imageArray*的矩阵复制到窗口，慢速的。

参数：
rect 是窗口坐标. 整个图片会复制到*rect*中，必要时会进行缩放。*rect*的默认值是*imageArray*的矩形，在窗口中间。窗口中像素的方向就像Matlab的命令窗口里的文本一样. 第一个像素在左上角，行是横向的。
*imageArray*可以是一个亮度矩阵，一个RGB颜色矩阵，或者一个RGBA颜色矩阵，数据类型可以是8位无符号（比较快）或者双精度。

说明：
需要注意的是，这个函数有点慢并且不是很灵活，如果需要更快速和更灵活地复制图片矩阵，不妨看下'MakeTexture'和'DrawTexture'函数。

返回

[*VBLTimestamp StimulusOnsetTime FlipTimestamp Missed Beampos*] = Screen('*Flip*', *windowPtr* [, *when*] [, *dontclear*] [, *dontsync*] [, *multiflip*]);
返回

作用：
在帧回描时，翻转（flip）前后显示面，并返回完成时间戳。

参数：
*windowPtr*是将在翻转时需要显示出来的onscreen窗口的id。
when 用于指定翻转的时间，默认值为0.如果设为0，那么将在下一次视频重绘（video retrace）时翻转。如果数字大于0，那么它将在系统时间when到达后的下一次视频重绘时翻转。
*dontclear*如果设为1，帧缓冲（framebuffer ）在翻转后不被清除，这样就可以不断地增加刺激在上面。它的默认值为0，在每次翻转后会把帧缓冲设为背景颜色。把值设为2可以保持帧缓冲不变，这样做的话就把建立缓冲的工作交给你了——在翻转后帧缓冲处于一种没有被定义的状态。，
dontsync 默认为0，在帧回描（vertical retrace）时进行的翻转，脚本的执行会暂停，直到翻转结束后才继续。如果为1，翻转将在帧回描时发生，但是脚本并不暂停，这意味着Flip函数的返回的时间戳是无效的。如果设为2，的话，刺激会立即显示，并不等待帧回描。
"*multiflip*" 默认值为0。如果设置成一个比0大的数字，那么Flip会翻转除了所设数字以外的所有onscreen窗口，这样，就可以在多个刺激显示器上同步进行刺激。比如立体声设备和haploscopes。You need to (somehow) synchronize all attached displays for this to operate tear-free.

VBLTimestamp Flip会(比较随机的地)返回高精度的系统时间的估计值(在很短的时间内)，这个时间是翻转实际发生的时间，返回在参数'*VBLTimestamp*'中.
*StimulusOnsetTime*刺激出现的时间返回在参数'*StimulusOnsetTime*'中.
Beampos 是在时间测量发生时，显示器的扫描线的位置(在做判断正确的测试（correctness test）时有用).
FlipTimestamp 是Flip函数执行完后的时间戳. *FlipTimestamp*和*VBLTimestamp* 的差值就是Flips执行时所花费的时间。当你想把脚本与回描同步起来时，这个计时误差会非常有用：比如促发一些像EEG、fMRI的采集设备或者重放一段声音。
"*Missed*" 指明有刺激所请求的呈现期限是否有丢失的情况。 一个负数表示deadlines have been satisfied.正数表示deadline-miss. 对deadline-miss的自动检查并不是万无一失的——尽管它能在大多数实验中像预期的那样运行，但它也有返回错误的正数或者负数的可能。如果你对时间特别挑剔，请用函数提供的时间戳或者用其他方法来进行你的实验。

返回

[*VBLTimestamp StimulusOnsetTime FlipTimestamp Missed Beampos*] = Screen('AsyncFlipBegin', *windowPtr* [, *when*] [, *dontclear*] [, *dontsync*] [, *multiflip*]);

返回

作用：
为给定的onscreen窗口安排一个前后显示的异步翻转。

变量：
windowPtr 是在翻转时需要显示出内容的onscreen窗口的id。
when 是请求的刺激开始的时间，一个为0的数值或者留空将在下一次垂直回扫时翻转。
其他变量 参见Flip

说明：
如果这个命令在调用时，以前的一个预定的异步翻转还在进行，那么它会先等那个异步翻转完成，并返回它的结果（比如*timestamps* ）。如果这种情况没有发生，那么它会返回最近完成的同步或异步翻转。等待前一个预定的翻转的完成并返回那个翻转的结果，这是一个方便的功能。在大多数情况下，为了获得脚本运行和翻转时间的更多掌控，你可以用一些终结形的命令来收集最终的结果以及异步翻转操作的时间安排，这些命令有 Screen('AsyncFlipCheckEnd') or Screen('AsyncFlipEnd') ，他们将在下文中提到。

Screen('AsyncFlipBegin',...)与Screen('Flip', ...)的区别在于Screen('Flip', ...)同步地运行：代码的执行会被暂停，直到翻转完才得以继续；比如至少要在when之后。Screen('AsyncFlipBegin')会把一切都在所要求的时间*when*准备好，如果*when*缺失了，就在下一次垂直回描时准备好——它可以再瞬间就返回你的代码。你的代码可以继续执行并作其他事情。比如在翻转其他onscreen窗口时同时读取键盘和鼠标的信息。

使用Screen('AsyncFlipCheckEnd') 和Screen('AsyncFlipEnd')，你可以检查同步翻转的过程的状态，或者等翻转完成后，收集那些被终止的翻转的时间信息。

一般来说，除非你有充分的理由，否则你应该尽量避免使用异步翻转，而使用更方便的Flip来替代它。这是因为异步翻转有好几个附加条件：

-- 在异步翻转时，你不能用Screen()和OpenGL：你不可以做有关texture或者offscreen窗口的事情，在它们的父onscreen窗口处于异步翻转状态时。你只能处理那些没有参与到异步翻转操作的onscreen窗口。

-- 如果你开启了Psychtoolbox的图片处理管道，那么在异步翻转时的大部分限制就放宽了：你可以在异步翻转时在任何窗口里绘制任何东西，哪怕是那个正在翻转的窗口。只有Screen('GetImage') 命令在异步翻转onscreen窗口时是禁止的，而且，在offscreen窗口上它有一些潜在的问题在异步翻转的

过程中。然而，这只是理论上的。实际上，在许多操作系统中，显卡驱动和显卡都不能真正处理并行绘制的负载或者异步翻转，这是系统的bug或者设计缺陷。在这样的系统中，你可以观察到不一致的时间安排，不好的表现，甚至会出现视觉刺激损坏、故障，甚至硬件系统崩溃【比如 Apple MacOS/X 10.4.11 with ATI Radeon X1600】!

--甚至更严格地，为了可靠性，有关Screen/OpenGL的命令都要避免，因为许多图形硬件和驱动做不到在不降低刺激开始计时的精确性的情况下进行同时的图形操作。例如，你可能会经历更多的丢失刺激的截止时间和时间故障——或者在不同的计算机、显卡和操作系统上发布会有不同的表现。最后，它允许你做声音、I/O、键盘检查等与Screen无关的事情。

-- 翻转的并行的程序为CPU、GPU还有操作系统带来的附加的负担，增加额外的开销；如果你的系统不堪重负，那么这可能会引起绘画质量下降、时间安排问题以及小故障。

-- 在异步翻转时运行的代码——就像并行代码段一样——正确地执行很难，调试也很有挑战性。

-- 使用非零的*multiflip*参数是禁止的。

-- 异步更新伽马表运行时貌似不可靠。

-- 立体模式10(分离的onscreen窗口) 的立体刺激显示在运行时可能不会按照预定的计时，有可能不一致。

-- 使用 'UserspaceBufferDrawingPrepare' 钩链住图片管道不是被允许的。

我们的立场是：大部分代码都可以不使用异步翻转高效地完成，所以这个特性是提供给少数有特殊需求的场合使用的，那些用了这个特性以后收获比付出还要大的场合。

返回

[*VBLTimestamp StimulusOnsetTime FlipTimestamp Missed Beampos*] = Screen('AsyncFlipEnd', *windowPtr*);
返回

作用：
等待先前安排的异步翻转操作完成（更多帮助请参见AsyncFlipBegin）。

说明：
这个命令将会等待onscreen窗口*windowPtr*，然后返回操作的结果。比如所有刺激开始的时间戳和他们的特征信息。
返回信息的更多解释请参见Flip。
如果你在没有调用'AsyncFlipBegin' 的情况下使用这个函数，无论之前是同步的翻转Screen('Flip')还是异步翻转，它都会很快地返回最近一次完成的翻转的信息。
Screen('AsyncFlipCheckEnd') 提供一个无阻断的，这个命令的轮询版本——哪怕操作没有完成也不会暂停。
返回

[*VBLTimestamp StimulusOnsetTime FlipTimestamp Missed Beampos*] = Screen('AsyncFlipCheckEnd', *windowPtr*);
返回

作用：
检查安排的异步翻转操作是否完成（更多帮助请参见AsyncFlipBegin）。

使用：
这个命令将会检查onscreen窗口*windowPtr*是否完成，然后返回操作的结果。比如所有刺激开始的时间戳和他们的特征信息。
返回信息的更多解释请参见Flip。
如果翻转操作还没有完成，那么*'VBLTimestamp'* 返回值的为0，你需要过会儿再试。
如果你在没有调用'AsyncFlipBegin' 的情况下使用这个函数，无论之前是同步的翻转Screen('Flip')还是异步翻转，它都会很快地返回最近一次完成的翻转的信息。
Screen('AsyncFlipEnd') 提供一个这个命令的无阻断的版本——程序会暂停直到操作完成。

返回

[VBLTimestamp StimulusOnsetTime swapCertainTime] = Screen('WaitUntilAsyncFlipCertain', windowPtr);
返回

Wait until it is certain that a previously initiated
Screen('AsyncFlipBegin',...); operation for onscreen window 'windowPtr' will
happen at next vertical retrace or has happened already, then return timestamps

of when flip operation has happened or when the flip operation will happen.
This function only works on Linux and MacOS/X and only with recent AMD/ATI hardware of the Radeon X1000/HD2000/... series or later (or the equivalent FireGL cards). On MacOS/X, the PsychtoolboxKernelDriver needs to be loaded for this to work (see help PsychtoolboxKernelDriver).
The method tries to "read the mind" of your graphics card to find out if the card will swap the display buffers at the next vertical retrace to interrogate if the swap is certain, ie., not possibly delayed by graphics card overload, other overload conditions or system timing jitter/delays.
Ideally it will allow your script to know when stimulus onset will happen a few milliseconds before stimulus onset. This allows, e.g., to initiate some other operations that need to be tightly and reliably synchronized to stimulus onset and that have some small startup latency, so they need some headstart. Examples would be sound output or output of some slow trigger signal which takes multiple milliseconds to take action.
This feature only works in fullscreen mode and is considered experimental - It may or may not work reliably on your setup. Good luck!
Returns a high-precision estimate of the system time (in seconds) when the actual flip will- or has happened in the return argument 'VBLTimestamp'. An estimate of Stimulus-onset time is returned in 'StimulusOnsetTime'.
'swapCertainTime' is the system time when certainty of bufferswap was detected.

返回

---

info = Screen('GetFlipInfo', windowPtr [, infoType=0] [, auxArg1]);
返回
Returns a struct with miscellaneous info about finished flips on the specified onscreen window.

This function is currently only supported on Linux with the free graphics drivers.

The function allows you to enable logging of timestamps and other status information about all completed bufferswaps, as triggered via Screen('Flip'), Screen('AsyncFlipBegin') etc. Whenever a flip completes, a little info struct is stored in a internal queue. This function allows you to fetch these info structs from the queue. It allows you to enable and disable logging of these info structs. Logging is disabled by default.

"windowPtr" is the handle of the onscreen window for which info should be returned.
"infoType" If left out or set to zero, the flip handle for the last invocation of flip is returned. E.g., if a value of 53 is returned, then the info struct with a info.SwapbuffersCount field of 53 will contain the info for that last invocation of flip.
This allows to associate specific flips with the returned logged timestamps and other info.
If set to 1, logging of flip completion info is enabled.
If set to 2, logging of flip completion info is disabled.
If set to 3, the oldest stored flip completion info is returned in a struct 'info'.

The info struct contains the following fields:

OnsetTime: Visual stimulus onset time of the completed Screen('Flip') operation.
OnsetVBLCount: Video refresh cycle count when the flip completed.
SwapbuffersCount: Serial number of this info struct. Corresponds to the handle returned for 'infoType' zero.
SwapType: How was the flip executed? Low level info about strategy chosen by GPU.
Note: Currently only PAGEFLIP flips are considered to have reliable timing and

trustworthy timestamps!
返回

[telapsed] = SCREEN('DrawingFinished', windowPtr [, dontclear][, sync]);
返回
Tell Psychtoolbox that no further drawing commands will be issued to 'windowPtr' before the next Screen('Flip') or Screen('AsyncFlipBegin') command.

This is a hint that allows to optimize drawing performance on some occasions. Don't issue this command multiple times between a Flip, it will degrade performance or even cause undefined stimulus display!

You must provide the same value for 'dontclear' flag that you're going to pass to the following Flip command, if you pass such an optional flag to the Flip command.

You can time the execution of all drawing commands between the most recent Flip and this command by setting the optional flag sync=1. In that case, telapsed is the elapsed time at drawing completion. Don't set the sync - Flag for real experiments, it will degrade performance!

Some recent graphics cards provide a more fine-grained way to measure the time spent drawing and processing your stimulus. See the help of 'Screen GetWindowInfo?' about 'infoType' settings 5 and 6 on how to use that mechanism. That method also has the advantage of measuring precise without degrading overall performance.
返回

---

framesSinceLastWait = Screen('WaitBlanking', windowPtr [, waitFrames]);
返回
Wait for specified number of monitor refresh intervals, stopping PTB's execution

until then. Select waitFrames=1 (or omit it, since that's the default) to wait for the beginning of the next frame. "windowPtr" is the pointer to the onscreen window for which we should wait for. framesSinceLastWait contains the number of video refresh intervals from the last time Screen('WaitBlanking') or Screen('Flip') returned until return from this call to WaitBlanking. Please note that this function is only provided to keep old code from OS-9 PTB running. Use the Screen('Flip') command for all new code as this allows for much higher accuracy and reliability of stimulus timing and enables a huge number of new and very useful features! COMPATIBILITY TO OS-9 PTB: If you absolutely need to run old code for the old MacOS-9 or Windows Psychtoolbox, you can switch into a compatibility mode by adding the command Screen('Preference', 'EmulateOldPTB', 1) at the very top of your script. This will restore Offscreen windows and WaitBlanking functionality, but at the same time disable most of the new features of the OpenGL Psychtoolbox. Please do not write new experiment code in the old style! Emulation mode is pretty new and may contain significant bugs, so use with great caution!

返回

IMREAD 从图形文件中读取图片.

返回

A = IMREAD(*FILENAME*,*FMT*)

从字符串*FILENAME*所指定的图形文件中读取黑白/彩色图片. 若该图片不在当前文件夹也不在当前MATLAB路径, 则需在*FILENAME*中给出路径.

字符串*FMT*需要给出所指定图片的文件格式, 如'gif'等. 如欲了解可以支持的文件格式, 请调用IMFORMATS函数. 如果IMREAD找不到用字符串FILENAME指定的文件, 则会搜索名为*FILENAME.FMT*的文件.

返回值A为一数组, 其中包括该图片的数据信息. (灰度图片为A[M][N], 全彩图片为A[M][N][3], 用CMYK定义的图片为A[M][N][4]; 参见特定格式信息部分当中的TIFF一条.) A的数据类型由图片中每点字节数决定, 如24色值返回的是8位整形数据, 因为其RGB当中的每种成分均以8比特的数据定义. 参见特定格式信息部分当中关于所支持数据大小的讨论, 附注中给出了位数的具体定义方式.

[*X*, *MAP*] = IMREAD(*FILENAME*, *FMT*)

字符串X定义索引图, MAP定义颜色映射. 颜色映射值自动换算为[0,1] 之间的数据.

[…] = IMREAD(*FILENAME*) 不需指定图片格式，该函数根据图片大小推算其格式.

[…] = IMREAD(*URL*,…) 从URL地址读取网上图片. 地址中必须包括协议类型(如”http://”)

附注
位数指代表每一象素点的比特数. 位数 = 每一象素点所包含的颜色成分 * 定义每种颜色成分的比特数. 因此常规所用的RGB三色每色8比特，每一象素点的位数为24. 但是，48位象素点是指8比特的6种颜色成分，还是16比特的3种颜色成分？见下面的特定格式信息部分.

特定格式信息(按文件格式名的字母顺序排列)
BMP - Windows位图
支持位数 定义

1-bit 逻辑代数
4-bit 8位整型
8-bit 8位整型
16-bit 8位整型，每象素点一种颜色成分
24-bit 8位整型，每象素点三种颜色成分
32-bit 8位整型，每象素点三种颜色成分(用1个字节填充颜色)

CUR -- 光标文件(MS是咱们平常没用过的图像文件类型… …我也这样赶脚)
支持位数 定义

1-bit 逻辑代数
4-bit 8位整型
8-bit 8位整型

特有指令:
[...] = IMREAD(...,*IDX*)
从多重图象文件或光标文件中读取一张图片. *IDX*为一整型数据，用于指定多重图象中欲读取图象的顺序，如IDX = 3 即读取文件中的第三张图；该值缺省默认读取第一张图.

[*A*,*MAP*,*ALPHA*] = IMREAD(...) 返回与光标文件相关的定义透明信息的数据.

GIF -- Graphics Interchange Format
支持位数 定义

1-bit 逻辑代数
2~8-bit 8位整型

特有指令:
[...] = IMREAD(...,*IDX*) 从多帧GIF图片中读取一帧或若干帧. *IDX*必须为整型数据(读取不止一帧时, 将IDX写为矢量的形式. 如IDX = 1:5 将返回GIF中的前5帧.)

[...] = IMREAD(...,'Frames',*IDX*) 与上一条相同, 但*IDX*可以定义为’all’ , 读取所有帧.
注意: 由于GIF文件的格式, 如需调用其中一帧图片, 则需读取整个GIF文件中的每一帧. 因此建议将IDX定义为一个矢量或者’all’ 而不是写一个循环逐个读取同一GIF中的每一帧.

HDF -- 层次结构式数据
支持位数 定义

8-bit 8位整型
24-bit 8位整型, 每象素点三种颜色成分

特有指令:
[...] = IMREAD(...,*REF*)
从多重图象的HDF文件中读取一张图. REF为一整型数据, 用于定义HDF文件中所调用图片的代码; 如REF = 12, 则读取HDF中代码为12的图. (注意在HDF文件中图片代码与其顺序无关. 可以用IMFINFO来根据图片顺序编辑代码.) 缺省默认读取文件中的第一张图.

ICO -- 图标文件
见CUR.

JPEG -- Joint Photographic Experts Group
IMREAD可以读取各种JPEG(基础JPEG + 各种拓展形式)
支持位数 定义

8-bit 8位整型, 黑白 / RGB
12-bit 16位整型, 黑白
16-bit 16位整型, 黑白
36-bit 16位整型, RGB(每种颜色成分12比特)

JPEG 2000 - Joint Photographic Experts Group 2000
支持位数 定义

1-bit 逻辑代数
2~8-bit 8位整型
9~16-bit 16位整型

注意: 不支持JPEG 2000 索引图.

特殊指令:
[...] = IMREAD(..., 'Param1', value1, 'Param2', value2, ...) 用参数值的数据对定义读取操作.
以下详述各种参数及其数值代表的含义.

'ReductionLevel' 非负整数, 指定图像分辨率的下降程度. 其值L表示分辨率下降2^L. 缺省默认为0, 即分辨率不下降. 分辨率下降程度由
'WaveletDecompositionLevels' 定义的总分解层数确定, 详见IMFINFO函数的返回值.

'PixelRegion' {ROWS, COLS}. IMREAD 返回由ROWS与COLS确定的图片部分, 这两个参数都必须定义为二维矢量, 包括起始行数和截止行数.若
'ReductionLevel' 大于0, 则ROWS与COLS将与分辨率下降的系数共同作用.

'V79Compatible' 布尔数值. 该值为真表示将颜色转换为之前IMREAD所用的颜色格式(黑白或RGB) , 该值为假返回原始图片的色彩格式. 缺省默认为假.
(该功能限MATLAB 7.9 [R2009b]及更早版本)

PBM -- 可转移位图
支持位数 定义

1-bit 逻辑代数

PCX -- Windows 画笔
支持位数 定义

1-bit 逻辑代数 黑白
8-bit 8位整型 黑白或索引图
24-bit 8位整型, 每象素点三种颜色成分

PGM -- 可转移黑白图
支持位数 定义

1~16-bit 8位整型

PNG -- Portable Network Graphics
支持位数 定义

1-bit 逻辑代数 黑白
2-bit 8位整型 黑白
4-bit 8位整型 黑白
8-bit 8位整型 黑白或索引图
16-bit 8位整型 黑白或索引图
24-bit 8位整型, 每象素点三种颜色成分
48-bit 16位整型, 每象素点三种颜色成分

特殊指令:
[...] = IMREAD(...,'BackgroundColor',*BG*)

BG用于定义背景颜色，调用此函数可使图片中透明的象素点呈现为该背景色. *BG*值为'none' 表示无背景色. 索引图的*BG*为[1,P]之间的整数，P代表颜色映射的长度. 黑白图片的*BG*取值范围为[0,1]，RGB图片的*BG*应为三维矢量，矢量中每一数值的取值范围均为[0,1].
如果调用了Alpha输出参数（见下条所述），*BG*的缺省默认值为'none'. 如果PNG文件包含了背景色，则该色彩为*BG*的默认值. 如果未调用Alpha输出参数，文件也不包含背景色，则索引图的*BG*缺省值为1，黑白图片的缺省值为0，RGB的缺省值为[0 0 0].

[A,MAP,ALPHA] = IMREAD(...) 返回正在呈现的Alpha通道值；如未呈现，则ALPHA = []. 若定义了'BackgroundColor'，则ALPHA = []. 注意，如果文件中包含黑白图片或全彩图片，MAP处可以为空.

PPM -- 可转移点图
支持位数 定义

1~16-bit 8位整型

RAS -- 太阳光栅
支持位数 定义

1-bit 逻辑代数 位图
8-bit 8位整型 索引图
24-bit 8位整型，每象素点三种颜色成分
32-bit 8位整型，每象素点三种颜色成分 + Alpha

TIFF -- 标记图像文件格式
支持位数 定义

1-bit 逻辑代数
8-bit 8位整型
12-bit 16位整型，黑白或索引图
16-bit 16位整型，黑白或索引图
24-bit 8位整型，3种颜色成分
32-bit 8位整型，4种颜色成分
36-bit 16位整型，3种颜色成分

48-bit 16位整型, 3种颜色成分
64-bit 双精度, 4种颜色成分

注意: IMREAD支持8-bit整型与32-bit浮点型的TIFF图象, 包括上述各种压缩与色隙的组合形式, 以及32-bit的IEEE浮点型图象.

特殊指令:
A = IMREAD(...)
返回RGB, CIELAB, ICCLAB或CMYK格式下的颜色数值. 如果使用CMYK, 则A数组为A[M][N][4].

[...] = IMREAD(..., 'Param1', value1, 'Param2', value2, ...)
用参数值的数据对定义读取操作. 有以下三种参数可供选择: uses
'Index' 正整数, 用于指定多重图象TIFF文件中所需读取的图像. 如果'Index' = 3, 则读取文件中的第三张图.

'Info' 结构数组, IMFINFO的结果. 读取多重图像TIFF文件时将IMFINFO的值传至 'Info' 可使IMREAD更快地找到图象.

'PixelRegion' {ROWS, COLS}. IMREAD 返回由ROWS与COLS确定的图片部分, 这两个参数都必须定义为二维矢量或三维矢量: 二维矢量包括起始行数和截止行数; 三位矢量包括起始行数, 增量与截止行数, 允许图象降位采样.

XWD -- X Window Dump
支持位数 定义

1-bit 逻辑代数
8-bit 8位整型

详见libtiffcopyright.txt 文件.
返回
---
wakeup=WaitSecs(s)
返回
Waits "s" seconds with high precision. The timing precision depends on
the model of your computer, but a well configured system will be accurate

to about 1 millisecond if your script is executed with realtime-priority
(See help Priority) and well written.

WaitSecs optionally returns the time at "wakeup" in seconds, just as a
WaitSecs(s); wakeup = GetSecs; would do, but with less overhead.

WaitSecs(s) is similar to Matlab's built-in PAUSE(s) command. The
advantage of WaitSecs(s) is that it is much more accurate. However, PAUSE
can be turned 'ON' and 'OFF', which is useful for scripts.

You can also use WaitSecs to wait until a specific time 'when' is reached,
instead of waiting for a specific interval. This is a drift-free approach,
even suitable for waiting a given interval, because errors can't accumulate:

wakeup = WaitSecs('UntilTime', when);

TIMING ADVICE: the first time you access any MEX function or M file,
Matlab takes several hundred milliseconds to load it from disk.
Allocating a variable takes time too. Usually you'll want to omit those
delays from your timing measurements by making sure all the functions you
use are loaded and that all the variables you use are allocated, before
you start timing. MEX files stay loaded until you flush the MEX files
(e.g. by changing directory or calling CLEAR MEX). M files and variables
stay in memory until you clear them.
返回

[x,y,buttons,focus,valuators,valinfo] =GetMouse([windowPtrOrScreenNumber][, mouseDev])
返回
Returns the current (x,y) position of the cursor and the up/down state
of the mouse buttons. "buttons" is a 1xN matrix where N is the number of
mouse buttons. Each element of the matrix represents one mouse button.

The element is true (1) if the corresponding mouse button is pressed and false (0) otherwise.

If an optional windowPtr argument for an onscreen window is provided, GetMouse will also return the window focus state as optional 4th return argument 'focus'. 'focus' is 1 if the window has input focus and zero otherwise.

The optional 'mouseDev' parameter allows to select a specific mouse or pointer device to query if your system has multiple pointer devices. Currently Linux only, silently ignored on other operating systems.

On Linux, the optional 'valuator' return argument contains the current values of all axis on a multi-axis device, ie., a device which not only has an x- and y-axis like a conventional mouse. E.g., digitizer tablets (like the "Wacom" pen tablets), may also have axis (also called "valuators") which report pen rotation, pen tilt and yaw angle wrt. the tablet surface, distance to the tablet surface, or normal and tangential pen pressure. Touchpads or trackpads may return contact area with the finger, or pressure. Joysticks may return info about additional sliders, wheels or other controls beyond the deflection of the joystick itself.

'valuators' is a vector with one double value per axis on Linux. On OS/X or MS-Windows, valuator is an empty matrix.

The optional 'valinfo' struct array contains one struct per valuator. The struct contains fields with info about a valuator, e.g., minimum and maximum value, resolution and a label. This is only supported on Linux. On other systems it is an empty matrix.

```
% Test if any mouse button is pressed.
if any(buttons)
fprintf('Someone''s pressing a button.\n');
end

% Test if the first mouse button is pressed.
if buttons(1)
fprintf('Someone''s pressing the first button!\n');
end

% Test if the second mouse button is pressed.
if length(buttons)>=2 & buttons(2)
fprintf('Someone''s pressing the second button!\n');
end

length(buttons) tells you how many buttons there are on your mouse.

The cursor position (x,y) is "local", i.e. relative to the origin of
the window or screen, if supplied. Otherwise it's "global", i.e. relative
to the origin of the main screen (the one with the menu bar).

NOTE: If you use GetMouse to wait for clicks, don't forget to wait
for the user to release the mouse button, ending the current click, before
you begin waiting for the next mouse press.

Alternatively, you can also use the GetClicks() function to wait for
mouse-clicks and return the mouse position of first click and the number
of mouse button clicks.

fprintf('Please click the mouse now.\n');
[x,y,buttons] = GetMouse;
```

```
while any(buttons) % if already down, wait for release
[x,y,buttons] = GetMouse;
end
while ~any(buttons) % wait for press
[x,y,buttons] = GetMouse;
end
while any(buttons) % wait for release
[x,y,buttons] = GetMouse;
end
fprintf('You clicked! Thanks.\n');
```

返回

[keyIsDown, secs, keyCode, deltaSecs] =KbCheck([deviceNumber])

返回

Return keyboard status (keyIsDown), time (secs) of the status check, and keyboard scan code (keyCode).

keyIsDown 1 if any key, including modifiers such as <shift>, <control> or <caps lock> is down. 0 otherwise.

secs Time of keypress as returned by GetSecs.

keyCode A 256-element logical array. Each bit within the logical array represents one keyboard key. If a key is pressed, its bit is set, othewise the bit is clear. To convert a keyCode to a vector of key numbers use FIND(keyCode). To find a key's keyNumber use KbName or KbDemo.

deltaSecs Time in seconds since this KbCheck query and the most recent previous query (if any). This value is in some

sense a confidence interval, e.g., for reaction time measurements. If KbCheck returns the information that a key is pressed by the subject, then the subject could have pressed the key down anytime between this invocation of KbCheck at time 'secs' and the most recent previous invocation. Therefore, 'deltaSecs' tells you about the interval in which depression of the key(s) might have happened: [secs - deltaSecs; secs]. for practical purpose this means that "measured" RT's can't be more accurate than 'deltaSecs' seconds - the interval between the two most recent keyboard checks. Please note however, that standard computer keyboards can incur additional delays and timing uncertainty of up to 50 msecs, so the real uncertainty can be higher than 'deltaSecs' -- 'deltaSecs' is just a lower bound!

KbCheck and KbWait determine whether any key is down now, including the meta keys: <caps lock>, <shift>, <command>, <control>, and <option>. The only key not reported is the start key (triangle) used to power on your computer.

Some users of Laptops experienced the problem of "stuck keys": Some keys are always reported as "down", so KbWait returns immediately and KbCheck always reports keyIsDown == 1. This is often due to special function keys. These keys or system functionality are assigned vendor specific key codes, e.g., the status of the Laptop lid (opened/closed) could be reported by some special keycode. Whenever the Laptop lid is open, this key will be reported as pressed. You can work around this problem by passing a list of keycodes to be ignored by KbCheck and KbWait. See "help DisableKeysForKbCheck" on how to do this.

Keys pressed by the subject often show up in the Matlab command window as well, cluttering that window with useless character junk. You can prevent this from happening by disabling keyboard input to Matlab: Add a ListenChar(2); command at the beginning of your script and a ListenChar(0); to the end of your script to enable/disable transmission of keypresses to Matlab. If your script should abort and your keyboard is dead, press CTRL+C to reenable keyboard input -- It is the same as ListenChar(0). See 'help ListenChar' for more info.

GetChar and CharAvail are character-oriented (and slow), whereas KbCheck and KbWait are keypress-oriented (and fast). If only a meta key was hit, KbCheck will return true, because a key was pressed, but CharAvail will return false, because no character was generated. See GetChar.

KbCheck and KbWait are MEX files, which take time to load when they're first called. They'll then stay loaded until you flush them (e.g. by changing directory or calling CLEAR MEX).

OSX, Linux, and Windows with Octave or Matlab R2007a and later: _____

KbCheck uses the PsychHID function, a general purpose function for reading from the Human Interface Device (HID) class of USB devices.

KbCheck queries the first USB-HID keyboard device by default. Optionally, when multiple keyboards are attached to your machine, you can pass in a 'deviceNumber': When 'deviceNumber' is -1, KbCheck will query all keyboard devices and return their "merged state" - The 'keyCode' vector will represent the state of all keys of all keyboards, and the 'keyIsDown' flag will be equal to one if at least one key on any of the keyboards is pressed. When 'deviceNumber' is -2, KbCheck will query all keypad devices (if any) and return their "merged state", and when

'deviceNumber' is -3, KbCheck will query all keyboard and keypad devices and return their "merged state". When 'deviceNumber' is greater than 0, it will query only the specified HID keyboard device corresponding to that 'deviceNumber'. The function GetKeyboardIndices() allows to query the device numbers of all attached keyboards, or keyboards matching specific criteria, and the function GetKeypadIndices() allows the same for keypads.

On MS-Windows 2000 and earlier, KbCheck can address individual keyboards. On Windows-XP and later, it can't.

As a little bonus, KbCheck can also query other HID human input devices which have keys or buttons as if they were keyboards. If you pass in the deviceIndex of a mouse (GetMouseIndices will provide with them), it will report mouse button state as keyboard state. Similar behaviour usually works with Joysticks, Gamepads and other input controllers.

返回

FOPEN 打开文件

返回

FID = FOPEN(*FILENAME*)

FID = FOPEN(FILENAME,PERMISSION)

[FID, MESSAGE] = FOPEN(FILENAME,...)返回一个依赖于系统的错误消息如果打开不成功的话。

[FID, MESSAGE] = FOPEN(FILENAME,PERMISSION,MACHINEFORMAT) 打开指定文件通过指定的许可的字符以及用DREAD读取数据或者用FWRITE写入数据作为*MACHINEFORMAT*给出的格式。

[FID, MESSAGE] = FOPEN(FILENAME,PERMISSION,MACHINEFORMAT,ENCODING) 打开指定的文件使用指定的*PERMISSION*和*MACHINEFORMAT*。

[FILENAME,PERMISSION,MACHINEFORMAT,ENCODING] = FOPEN(FID)返回被MATLAB打开相关的文件和标识符FID使用的文件名、许可、机器格式、字符编码值。MATLAB并不决定这些通过从打开的文件读取信息的输出值。对于这些参数当文件被打开没有被指定时,MATLAB返回其默认值。编码的字符串是标准的字符编码方式的名字,可能和调用FOPEN,打开的文件中的编码参数不相同。

一个无效的FID返回空字符串的所有输出参数。

返回一个包含文件行向量标识符的所有被当前的用户打开的文件(但不是1或2)。
W ’和’ A ’权限不自动执行大量的当前输出缓冲区在输出操作之后。

作用：
是用来打开*FILENAME*文件来进行读取。

参数：
*PERMISSION*打开文件*FILENAME*时，只有指定的*PERMISSION*才是被许可的。
‘r’ 打开文件来读入
‘w’ 打开文件来写入；放弃已有的内容
‘a’ 打开或创建一个文件来写入；数据附加到文件末尾
‘r+’ 打开（不要新建）文件来读取或者写入
‘w+’打开或者新建一个文件来读取或者写入；放弃已有的内容
‘a+’ 打开或新建一个文件来读取或者写入；数据附加到文件末尾
‘W’ 打开文件或写入没有自动刷新
‘A’ 打开文件来添加内容没有自动刷新
*FILENAME*可以是一个和MATLABPATH相关的部分路径名只有当该文件被打开来读取时。
你可以以二进制模式(默认)或文本模式打开文件。
在二进制模式下，没有元素被挑出来进行特殊的处理。
在计算机中的文本模式下，回车字符的前一个换行符被删除输入和添加到换行之前。
添加’ t ’在许可的字符串之后就可以打开一个文件以文本模式，例如‘rt’和‘w+t’。（在Unix中，文本和二进制模式是一样的，所以这样是无效的。在计算机系统中，这是很关键的。）
如果以更新模式(“+”)打开文件,你必须使用一个FSEEK或FREWIND在输入命令之间FREAD,FSCANF,FGETS,或FGETL以及在输出命令之间像FWRITE或FPRINTF。
你也必须使用一个FSEEK或者FREWIND在输入和输出命令之间。
两个文件的标识符是可用的,并且不需要自动打开。它们是FID=1 (standard output) 和 FID=2 (standard error).

*MACHINEFORMAT*是如下的几种字符串之一：
‘native’或者‘n’——本地机器格式——默认的
‘ieee-be’或者‘l’——IEEE浮点以低位优先的字节排序
‘ieee-be’或‘b’——IEEE浮点以高位优先字节排序

‘ieee-le.l64 ’ 或 ‘ a ’ - IEEE浮点以低位优先的字节排序和64位长数据类型
‘ieee-be. l64’ 或 ‘s’ ——IEEE浮点以高位优先字节排序和64位长数据类型
*ENCODING*是一个字符串,指定与字符编码方式相关联的文件。它必须是空字符串('')或一个名称或别名作为编码模式。一些例子像 ‘UTF-8’, ‘latin1’, ‘US-ASCII’,以及 ‘Shift_JIS’。对于常见的名字和别名,请参见网站http://www.iana.org/assignments/character-sets。如果编码是未指定的或者是空字符串(''),MATLAB使用默认的编码方案。

返回

FSCANF 从文件读取格式化的数据。返回

作用:
[A,COUNT] = FSCANF(*FID,FORMAT,SIZE*) 从文件读取数据文件指定的标识符 *FID*,将它转换为按照指定的格式字符串,并返回在矩阵*A* 。

参数:
A 转换后指定格式的字符串矩阵。
*COUNT*一个可选的输出参数,返回成功读取元素的数量。
*FID*是一个从FOPEN中获得的整型文件标识符。
*SIZE*是可选的;它限制了从文件读取字符的长度;如果没有指定,整个文件被默认;如果指定,有效条目如下:
N 读取至多*N*个元素为一个列向量
inf 至多读取到文件的结尾
[M,N] 至多可以读取*M***N*个元素填充*M***N*阶矩阵，以行的顺序填取。*N*可以是*inf*，但不能等于***M***
如果矩阵*A*只为字符转换的结果以及*SIZE*并不是[*M,N*]的形式，之后会返回一行向量。
FORMAT是一个字符串,其中包含普通字符和/或C语言转换规范.转换规范涉及到字符%,可选的分配压制星号和宽度字段,和转换字符d,i,o,u,x,e,f,g,s,c和[...]。完整的ANSI C支持这些转换字符是提供符合“预期”的MATLAB的行为。对于一个完全转换字符规范,请参阅语言参考指南或C手册。
如果一个元素使用%s阅读可能会导致几个MATLAB矩阵元素被使用,各含一个字符。必须使用%c读空格字符。因为格式%s跳过了所有的空格。MATLAB读取字符使用的编码方式与档案。FOPEN可以更多信息。
如果该格式字符串包含普通字符,MATLAB将从文件读取之后装换成MATLAB的内部表示的字符相匹配。
混合字符和数字转换规格使得矩阵为数值和任何字符读取显示为它们的数字值,每MATLAB矩阵元素一个字符。
FSCANF不同于它的C语言在一个重要的方面。它“矢量化”以返回一个矩阵参数。格式字符串通过文件循环直到文件的末尾或者把指定的数据量的大小读入后为止。

返回

FPRINTF 返回

作用：

FPRINTF(*FID*, *FORMAT*, *A*, ...)编写格式化的数据文本文件。
FPRINTF(*FORMAT*, *A*, ...)格式化数据,把结果显示在屏幕。
COUNT= FPRINTF(...)返回FPRINT写入的字节数。

参数：

A 应用格式到所有数组A的元素以及任何以列排列的附加数组,并将数据写入到一个文本文件。
*FID*是一个整形的文件标识符。从FOPEN获得*FID*,或者将它设置为1(标准输出，屏幕)或2(标准误差)。FPRINT使用调用的FOPEN中指定的编码方式。
*FORMAT*是一个字符串,描述了输出的格式字段,可以包括组合以下:
转换规范,其中包括一个%字符,一个转换字符(如d,i,o,u,x,f,e,g，c或s),以及可选的标记,宽度,和精确字段。
更多细节,在命令提示符输入“doc fprintf”。
逐字打印文本。
缺省字符，包括：
\b 退格 '' 单引号
\f 换页 %% 百分比字符
\n 换行 \\ 反斜杠符号
\r 回车 \xN 十六进制数N
\t 水平制表符 \N 八进制数N
在大多数情况下,\ n足够用于单个换行符。
然而,如果您要使用微软记事本创建一个文件,要用\r\n来换行。
注意：
如果你用一个整数或字符串转换为一个数值,它包含一个分数,MATLAB覆盖指定的转换,并使用%e。
数值转换只打印复数的实部。
*COUNT*FPRINT写入的字节数。

返回

FCLOSE 关闭文件
返回

作用：
ST = FCLOSE(*FID*) 关闭文件的
ST = FCLOSE('all') 关闭所有打开的文件，除了*FID*为0,1,2的那些文件。

参数：
ST为函数返回值，FID为文件参数，由fopen函数打开文件时产生。
当文件成功关闭返回0，否则返回-1.
如果*FID*不表示一个打开的文件或者*FID*等于0,1或2，则flcose显示错误。

返回

TIC 打开了一个停表
返回

作用：
TSTART =TIC 输出量保存了一个时间点，但这个值只有当调用TOC时才有用。

使用：
tic和toc函数一起用来计算程序运行的时间。
tic自身可以保存此时的时间，而toc可以计算两个tic确定的时刻间的时间。
调用*TSTART*= TIC为*TSTART*这个输出量保存了一个时间点，但这个值只有当调用TOC时才有用。
例如： 计算贝塞尔函数的和所用的最少和平均时间。

```
REPS = 1000; minTime = Inf; nsum = 10;
tic;
for i=1:REPS
tstart = tic;
sum = 0; for j=1:nsum, sum = sum + besselj(j,REPS); end
telapsed = toc(tstart);
```

```
minTime = min(telapsed,minTime);
end
averageTime = toc/REPS;
```

返回

TOC 函数读取停表的时间。

返回

作用：

T =TOC 返回以秒为单位计算两个最相邻的TIC到TOC函数间的时间。

TOC(*TSTART*)计算TIC指令下产生的*TSTART*间的时间。

说明：

tic和toc函数一起用来计算程序运行的时间。

例如： 计算贝塞尔函数的和所用的最少和平均时间。

```
REPS = 1000; minTime = Inf; nsum = 10;
tic;
for i=1:REPS
tstart = tic;
sum = 0; for j=1:nsum, sum = sum + besselj(j,REPS); end
telapsed = toc(tstart);
minTime = min(telapsed,minTime);
end
averageTime = toc/REPS;
```

返回