Intro to Psychtoolbox
A.P. Saygin
MatlabFun 2014

Readings:
Schneider Notes Part 2

Psychophysics Toolbox, Psychtoolbox, PTB, or PTB3 (to differentiate it from older versions) is a free toolbox for Matlab. You can find all about it on the website/wiki: www.psychtoolbox.org [1]

When you use PTB, you should cite the following papers:

Brainard, D.H. (1997) The Psychophysics Toolbox, *Spatial Vision 10*:443-446.
Pelli, D.G. (1997) The VideoToolbox software for visual psychophysics: Transforming numbers into movies, *Spatial Vision 10*:437-442.

As in: "The stimuli were presented and responses collected using Matlab (Mathworks, Natick, MA) and the Psychophysics Toolbox (Brainard, 1997; Pelli, 1997).

This is from the PTB wiki :

"The attraction of using computer displays for visual psychophysics is that they allow software specification of the stimulus. Programs to run experiments are often written in a low-level language (e.g. C or Pascal)[2] to achieve full control of the hardware for precise stimulus display. Although these low-level languages provide power and flexibility, they are not conducive to rapid program development. Interpreted languages (e.g. BASIC, LISP, Mathematica, and Matlab) are abstracted from hardware details and provide friendlier development environments, but don't provide the hardware control needed for precise stimulus display. The Psychophysics Toolbox is a software package that adds this capability to the Matlab application on Macintosh, Linux and Windows computers."

That's pretty clear, I hope. When you run experiments, precision is very important. We don't want any imprecision in colors, or tiny timing delays even if you cannot consciously perceive them!

The wiki also mentions: "Brand-new users who've never programmed before will find that they're learning three things when they start using the toolbox: Matlab, how to create stimuli and measure responses, and how to organize an experiment. There's almost no overlap between those three topics."

To give you an idea: So far, you have done a lot of the first, and some work towards the second (all the image & sound manipulations), and the third (functions, structures, cell arrays,

[1] Not .com, which is the website of a motivational speaker; though you may need some motivation at times when struggling with code!
[2] Tangent: Does anyone still use Pascal? I haven't even heard it mentioned since I was in college.

files). We are going to achieve very basic competence in all three areas quite quickly and then your projects will further advance you gradually.

Some recommendations:

- DEMOS: Go to Psychtoolbox/PsychDemos folder to see a list of demos. You can run them as well as look at the code, which can help.

Note: Some demos may not run on the lab computers. Most likely it will be the fancy demos that won't work. Some fixes may be possible by updating the graphics drivers but I am not too optimistic since this is largely a hardware issue. I am hoping this won't limit you in developing your projects…

- CRASH: You will almost certainly crash PTB in your programming adventures. Read (and save a copy or print this: http://psychtoolbox.org/wikka.php?wakka=FaqHowDoIExitScreen. Another tip is, if possible, to use a screen size that does not cover your entire desktop (this means you specify size when you call Screen('OpenWindow'), see below). That way if your program crashes you can still access Matlab command window in the back to kill Screen.

- SYNC Troubles: If you see some timing errors, complaints about sync or VBL, see this: http://docs.psychtoolbox.org/SyncTrouble. It's long, but well worth reading.

- OPENGL: OpenGL is a library for computer graphics. It allows us to create applications which render high-quality color images with 3D objects. You are not required to go into details of OpenGL at the moment. I'll post links and resources on the website for those of you who may want to use it for your projects. http://www.opengl.org/about/overview.

-WARNINGS: You can disable PTB's warnings about graphics and timing for the time being by including the following in your code:

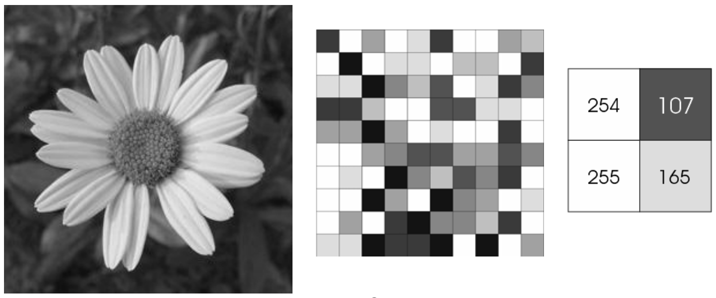
```
Screen('Preference', 'Verbosity', 0);
Screen('Preference', 'SkipSyncTests',1);
Screen('Preference', 'VisualDebugLevel',0);
```

Type **Screen('Preference?')** in the command line to get more info.

SOME CONCEPTS:

Pixels and Images: A pixel is a unit (typically square) of digital information. The pixels are laid out in a grid, which is an image. In Matlab an image is a matrix. You can think of each element of a matrix as a pixel.

Bit: A Bit is a digit in the binary number system. It can be 0 or 1. 8 bits = 1 byte. 1000 bytes = Megabytes and so on…



Pixel depth: What is an 8 bit image? A 16 bit image? Here, a bit is a unit of color information in a pixel. A 1 bit pixel can be 1 or 0, on or off, which will translate to black or white. An 8 bit pixel has greater pixel depth or bit depth. It can contain and display much more color information. As each bit can have 2 values, 8 bits can have 2x2x2x2x2x2x2x2 = 2^8 = 256 choices, which means 256 colors. An 8 bit image is likely to be a greyscale image, where one number determines the color of each pixel. An 8-bit color image will typically have 8 bits on each channel (red, green and blue), with a total of 24 bits used to represent color. High resolution color images typically have 16 bits per channel (48 bits total). An image's file size depends on the total pixel content of the image. Each pixel in your 8 bit image would use 1 byte of memory. The more pixels, and the more pixel depth, the larger the file.

Frame rate: Frame rate is the frequency at which a device (e.g., your monitor) produces unique consecutive images called frames. Most LCD monitors and laptop screens will be operating at 60 Hz, which means 60 frames per second.

Texture, Screen, Window: You will hear these terms a lot with PTB.
Recommended: What's the difference between window, screen and texture?
http://psychtoolbox.org/wikka.php?wakka=FaqTextureWindow

On screen window: This is where you display your stimulus. It's not complicated (unless you attach multiple monitors, you won't need to worry about this too much). What you want to do is prepare your stimuli and then tell Psychtoolbox to put it on the on screen window.

Off screen window: Whenever you draw something with PTB it will usually be drawn 'off screen'. The image exists in memory at this point and is not visible. You then need to use 'Flip' function of Screen so that the off screen window gets copied onto the on screen window (the one you can see).

Window pointer: This is a number that designates a specific window you created. You can create many windows. To use a window, you pass its window pointer to the Screen function. That way, Screen knows which window you want it to work on.

Buffer and 'Flip': A buffer is temporary storage. In the present context, what you will do is prepare what you want to display on the on screen window in a back buffer (off screen) and then copy the contents to the on screen. You can create as many off screen windows as you want. But in PTB3, we refer to the front buffer and a back buffer. We prep an image in the back buffer (see off screen window). When we 'Flip', this becomes displayed on the on screen and we get a new buffer to play with (off screen). If you had a movie which had 200 frames, you could display it by flipping the buffers on each frame. This is faster than using 200 off screen windows and copying them to the on screen one by one (although that would also work and older versions of PTB did just that).

SCREEN

This is the heart of Psychtoolbox. **Screen** is a loaded function that will be used in almost every experimental program with Psychtoolbox. Read its help file. In fact, I have pasted it on these notes. (See below).

Screen is like many functions in one and its syntax will require you to specify how you want to use it. The typical format is:

Screen('SomeCommand', Parameters, ...)

Type **Screen** on the command line to see a list. That's a lot of functionality! Fortunately each of these different uses of **Screen** has its own help file. Using screen will not be different from the functions you've been using, except the syntax is a bit more complicated looking. Just like any function, **Screen** has inputs and outputs… But you need to give it an argument which tells it what to do, such as **'Flip'** or **'OpenWindow'** and each of these has their own input and output properties. You can think of them as functions of their own.

Before you can do anything on the display, you have to open a main window.

To get the help, type:

Screen OpenWindow?

Or

Screen('OpenWindow?')

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

This means the function takes in 2 required arguments 'OpenWindow' and windowPtrOrScreenNumber and can take 7 more optional arguments.

For example:
[w,rect]=Screen('OpenWindow',0, 0, []);

The argument 'OpenWindow' is a string telling Screen what to do. The second argument is which computer monitor (or offscreen window) you want to use. Typically this will be 0. If you have multiple monitors attached things may get a little more complicated. For now, just use one monitor. The third argument indicates that you want the screen to be black (0). The fourth argument [], means use the default, in this case the experiment screen will cover the whole (monitor) screen. You may want to use a smaller experiment window while you're developing your code (see CRASH)

This will returns 2 variables: **w** and **rect.** That means when you call it, you must specify these variables.[3] **w** is a pointer or handle to the window. You can think of it as a variable that points to the window. Like we did with files. You will need it when you want to put stimuli on this screen

rect is a variable that will contain the size of the window. It will have 4 numbers in it, which are the positions of the start and end coordinates of your rectangle.

Window and texture dimensions are defined as a 1 x 4 vector: [X Y Width Height]

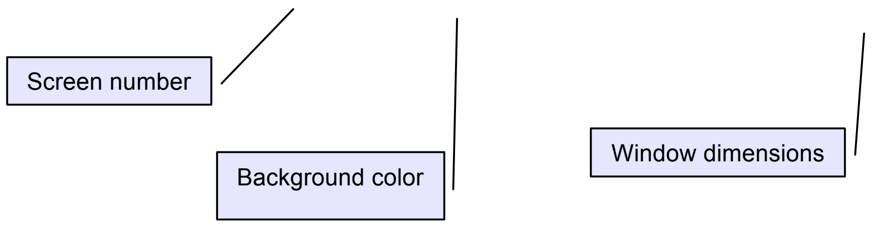
The coordinate system has 0,0 in the upper left-hand corner. The order of these coordinates can be coded as LeTteRBox (Left, Top, Right, Bottom). Note that the coordinates start from 0 and not 1. This is a C language convention and you just have to remember it.

If you just type **rect**, you can see the size of your current monitor's window.

For each Screen function, the help will tell you in detail what all these options mean.

What will the following do?

```
[w,rect] = Screen ('OpenWindow', 0, [117 117 117], [0 0 800 600]);
```



Once PTB has opened a main window you can draw into it. There is a "back buffer" onto which everything is drawn (off screen), so that things will only become visible when you flip the buffers:

[3] You can call them whatever you want). E.g., **[billy, bob] = Screen ('OpenWindow', 0);** will give these values to **billy** and **bob** (which are just random names you probably don't want to use).

```
Screen('FillOval', w, [255 0 0], [0 0 10 10]);
Screen('Flip', w);
```

Some other commands for drawing:

FillRect, FillPoly, FillArc, DrawLine, DrawText, ...

You can switch on and off the mouse pointed by using:
ShowCursor;
HideCursor;

What if you want to draw something at the center of the screen?
Try the functions **CenterRect** or **RectCenter**. (They're two different functions)

When your program is finished, close the window:
Screen('CloseAll')

<u>From here to the end is the help for Screen:</u>

```
Screen is a MEX file for precise control of the video display. Screen has
  many functions; type "Screen" for a list:
        Screen

  For explanation of any particular screen function, just add a question
  mark "?". E.g. for 'OpenWindow', try either of these equivalent forms:
        Screen('OpenWindow?')
        Screen OpenWindow?

  All the Screen Preference settings are documented together:
        Screen Preference?

  General Screen ARGUMENTS, common to most subfunctions of Screen:

  "windowPtr" argument: Screen 'OpenWindow' and 'OpenOffscreenWindow' both
  return a windowPtr, a number that designates the window you just
  created. You can create many windows. To use a window, you pass its
  windowPtr to the Screen function you want to apply to that window.

  "rect" argument: "rect" is a 1x4 matrix containing the upper left and
  lower right coordinates of an imaginary box containing all the pixels.
```

Thus a rect [0 0 1 1] contains just one pixel. All screen and window coordinates follow Apple Macintosh conventions. (In Apple's the pixels occupy the space between the coordinates.) Coordinates can be local to the window (i.e. 0,0 origin is at upper left of window), or local to the screen (origin at upper left of screen), or "global", which follows Apple's convention of treating the entire desktop (all your screens) as one big screen, with origin at the upper left of the main screen, which has the menu bar. Historically we've had two different orderings of the elements of rect, so, for general compatibility, all of the Psychophysics Toolbox refers to the elements symbolically, through RectLeft, RectTop, etc. Since 2/97, we use Apple's standard ordering: RectLeft=1, RectTop=2, RectRight=3, RectBottom=4.

[optional arguments]: Brackets in the function list, e.g. [color], indicate optional arguments, not matrices. Optional arguments must be in order, without omitting earlier ones, but you can use the empty matrix [] as a place holder, with the same effect as omitting it.

WHEN YOU GET A MATLAB ERROR

If your computer only has one screen (the typical scenario) and your program produces a Matlab error while your full-screen window is open, you'll hear the beep, but you won't be able to see the Matlab Command Window. Follow the instructions below for bringing forward the command window, then type clear screen to flush just the Screen MEX file, or "clear mex" to flush all the MEX files. When flushed, as part of its exit sequence, Screen closes all its windows, restores the screen's normal color table, and shows the cursor. Or you can get just those effects, without flushing, by calling Screen('CloseAll') or sca - which is an abbreviation for Screen('CloseAll').

You can use Matlab's EVAL command to do this for you automatically. E.g. if your program is called "foo.m", run your program by calling EVAL:

```
        eval('foo','clear screen;error(''error in foo'')')
```

If an error occurs in FOO, Matlab, instead of halting, will execute the second argument to EVAL, which restores your screen and reports the error.

OpenGL: ____

Instead of offscreen windows, the OpenGL Psychtoolbox uses fast rendering and OpenGL textures for animation. With the exception of matrices, all drawing may be done during the animation loop directly to the onscreen window, rather than being rendered to offscreen windows before the start of the movie. Matrices are converted to Textures before the start of the

animation and, like offscreen windows in OS 9, may be quickly copied to
an onscreen window during movie play. Offscreen windows are still supported
if you need to draw very complex stimuli. You can draw the stimulus into
an offscreen window and then quickly copy the window into the onscreen
window. For most purposes however, it is possible to draw directly into
the backbuffer of your offscreen window and make the backbuffer visible
on next vertical blank by a call to Screen('Flip', windowPtr).

See MovieDemoOSX and DriftDemoOSX for examples of how to create and show
movies this way.

Off-screen windows are invisible, but useful as an intermediate place to
create and store images for later display. Copying from window to window
is very fast. It's easy to precompute a series of off-screen windows
and then show them as a movie, in real time, one per video frame:

```
% make movie
window=Screen('OpenWindow', 0, 0);
rect=[0 0 200 200];
for i=1:100
        movie(i)=Screen('OpenOffscreenWindow', window, 0, rect);
        Screen('FillOval', movie(i), 255, [0 0 2 2]*(i-1));
end;

% show movie
for i=[1:100 100:-1:1] % forwards and backwards
        Screen('CopyWindow',movie(i),window,rect,rect);
        Screen('Flip', window);
end;
Screen('CloseAll');
```

Stopping programs:

Command-zero brings the Matlab Command window forward. (Type a zero
"0" while holding the apple-cloverleaf "command" key down.)

Ctrl-C halts any program. (Type a "c" while holding down the "Ctrl"
key). Sometimes, Ctrl-C fails to halt progams executing in a Matlab process
run with the "-nojvm" option. To halt a runaway Psychtoolbox script in
Psychtoolbox you might resort to the Windows Task Manager to kill
the Matlab process. (Use Ctrl-Alt-Delete to open a window from which
you can start the Task Manager.)

Windows:

Ctrl-Alt-Delete allows you to launch the Windows task manager, which reduces the Psychtoolbox onscreen windows when it opens. (Simultaneosly press the "Ctrl", "Alt", and "Delete" keys.) There are also simpler ways of reducing the Psychtoolbox window which are specific to particular versions of Windows.

Windows 2000:	Alt-Tab will bring another application to the foreground, minimizing the Matlab Psychtoolbox window.

OS-X:
Apple-Command-Escape executes "Force Quit" on Matlab, closing Matlab and all of its windows.

Linux:
Ctrl-Alt-Escape, followed by a mouse click kills the onscreen windows and your Matlab session.

See "help PsychDemos" for many demos which demonstrate Screen's capabilities.

Differences in Screens capabilities between different operating systems are discussed in the online help for the different subfunctions, our "PsychDemos" if differences apply, and on the Psychtoolbox Wiki under "Platform Differences and writing portable code".