Matlab and Psychophysics Toolbox Seminar
Part 3. Timing and Input

The Psychophysics Toolbox allows precise timing using highly accurate clocks. The function `GetSecs` returns the number of seconds that have elapsed on the main CPU clock since the last reboot. The function `WaitSecs(x)` waits for `x` seconds.

You can get an idea of how fast your computer (and the `GetSecs` function) runs using the following test:

```
>>times=zeros(1,10000);
>>for i=1:10000; times(i)=GetSecs; end
>>plot(times)
```

You can see that time increases linearly (hopefully) over time. You can look at the timing between iterations of the loop like this:

```
>>plot(diff(times))
```

On the *x*-axis is the loop interation number, and on the *y*-axis is the time that elapsed since the previous iteration of the loop, in seconds. So, 0.001 on the y-axis indicates one millisecond. You can see that the time between iterations of the loop is very fast, but still measurable.

Depending on your operating system (especially Mac OSX), you may see spikes in this graph that indicate that some process in the background is using some of the CPU time.

When timing is critical, you will want to tell the Psychophysics Toolbox to monopolize the CPU such that the background processes don't interfere. The way to do this is to use the `Priority` function.

Try the following (Note that if you have not previously done so, running the `Priority` function on Mac OSX will require that you run a script to allow the function to disable an updating process that normally runs in the background. Follow the instructions that are printed upon invoking `Priority` for the first time. If you don't get any warning, then this procedure has already been performed for your machine.):

```
>>Priority(9)  % or 7 for Windows
>>for i=1:10000; times(i)=GetSecs; end
>>Priority(0)
```

and plot the times and time differentials, as above. You shouldn't see any large spikes in the timing. `Priority(9)` tells the Psychtoolbox to request the maximum amount of CPU time. `Priority(0)` returns it to normal (and restarts the background updating process on Mac OSX).

It is sometimes useful to run this sort of timing loop when you are testing a stimulus to see how long it takes you to draw a single frame. For instance, if you are drawing a complicated animation, depending on the speed of your computer, it may take more than one refresh cycle of the monitor to draw the frame, in which case the animation won't look smooth. More on this below, as well as next week's lab.

Let's test the drawing speed of the computer you are using with this script:

```
[w,rect] = Screen('OpenWindow',0);
n=10000;                    % number of circles to draw
r=50;                       % radius of circle
times=zeros(1,n);
x=rand(1,n)*rect(3);
y=rand(1,n)*rect(4);
colors = 256*rand(n,3);
start_time = GetSecs;
for i=1:n
     Screen('FillOval',w,colors(i,:),[x(i)-r,y(i)-r,x(i)+r,y(i)+r]);
     times(i) = GetSecs-start_time;
end
Screen('Flip',w)
kbwait;
Screen('Close',w)
```

After you run this script, `times(n)` gives the total time required to draw all of the circles, and `diff(times)` gives the time between each drawing. You won't see the circles being drawn because they are not revealed until you execute the `Screen('Flip')` function.

Now try increasing the radius of the circles, `r` (and also decreasing `n`, if it takes too long). You should see that it takes longer to draw a larger object. If the radius is very large and you are using a slow computer, it may take longer than one screen refresh to draw each circle.

Display synchronization

As we discussed last week, the new Psychophysics Toolbox utilizes double buffering to avoid video artifacts, which eliminates many of the difficulties present using the previous version. The `Screen('Flip')` function pauses briefly until the end of the monitor refresh cycle and then returns control to the user to allow drawing for the next video frame. The exact time at which the stimulus will appear depends on how far into the video refresh cycle your monitor is at the time you issue the `Screen('Flip')` command.

Let's modify our script so that it draws one circle per video frame. Add this line immediately at the beginning of the for loop:

```
Screen('Flip',w,[],1);  % doesn't work for Windows
```

The extra parameters in the command tell the function not to clear the screen each time the buffers are flipped (overriding the default to clear the screen to the background color). Also, make sure that your `n` is no more than a few hundred, or the script will take a long time to run. Now, `plot(times)` and `plot(diff(times))`. What do you see? What is the number you see in the latter case?

Image timing

Last week we used the `Screen('DrawTexture')` function to draw an image to the screen. This function is very fast and should be comparable in speed to the shape drawing commands. You can test its timing with the following modifications to the script:

```
[w,rect] = Screen('OpenWindow',0);
img=imread('myimage.jpg');  % read in your image
t = Screen('MakeTexture',w,img);
[sy sx sz] = size(img);
n=500;  % number of images to draw
times=zeros(1,n);
x=rand(1,n)*rect(3);
y=rand(1,n)*rect(4);
start_time = GetSecs;
for i=1:n
    Screen('DrawTexture',w,t,[], ...
       [x(i)-sx/2,y(i)-sy/2,x(i)+sx/2,y(i)+sy/2]);
    times(i) = GetSecs-start_time;
end
Screen('Flip',w);
KbWait;
Screen('CloseAll')
```

Now do `plot(diff(times),'.')`. Notice the distribution of drawing times. How does resizing the image affect the drawing times? Add the lines

```
sx=sx/2;
sy=sy/2;
```

after the second line of your script. What if you double the size instead of halving it?

Now add a `Screen('Flip',w,[],1);` line into your `for` loop. How does this affect the times?

Displaying timed images

Now we are ready to write a simple script to display images in a particular timing sequence. Let's say we want to display an image ten times, once every 3 seconds, for 500 ms each time (you could modify this to present different images each time). This looks like a lot to type in, but much of it can be copied from the scripts you have already written.

```
img=imread('myimage.jpg');
[sy sx sz] = size(img);
[w,rect] = Screen('OpenWindow',0,[0 0 0]);
t=Screen('MakeTexture',w,img);
nreps=10;   % number of images to draw
on_time=0.5;
repeat_time = 3;
x=rand(1,nreps)*rect(3);
y=rand(1,nreps)*rect(4);
start_time = GetSecs;
next_on_time = 0;
next_off_time = on_time;
i=0;
while GetSecs - start_time < nreps * repeat_time
  thetime = GetSecs-start_time;
  if thetime >= next_on_time
    i=i+1;
    dest_rect = [x(i)-sx/2,y(i)-sy/2,x(i)+sx/2,y(i)+sy/2];
    Screen('Flip',w);
    Screen('DrawTexture',w,t,[],dest_rect);
    next_on_time = next_on_time + repeat_time;
  elseif thetime >= next_off_time
    Screen('Flip',w);
    next_off_time = next_off_time + repeat_time;
  end
end
Screen('FillRect',w,[255 255 255]); % white screen at end
Screen('Flip',w);
KbWait;
Screen('CloseAll')
```

This is just one example of a timing loop. There are many different ways to do these. If you want to be sure that your images are displayed at exactly the right times, you could check using the `GetSecs` function or the output arguments of `Screen('Flip')`.

Keyboard input

The Psychophysics Toolbox provides a fast function for measuring key responses, `KbCheck`. `KbCheck` is an instantaneous function, meaning that it returns the current state of the keyboard. Its syntax is (see the help file):

```
[keyIsDown,secs,keyCode] = KbCheck;
```

`keyIsDown` is `true` if any key is pressed. `secs` returns the current time when the function is called, and `keyCode` is an array of all the possible key values. If key number `2` is depressed, then `keyCode(2)` will be `true`. Multiple keys can be pressed at once, and `keyCode` will register them all. The the `KbName` function allows you to figure out which key code corresponds to which key.

Try this program:

```
fprintf('Press any key(s). Press escape key to exit.\n');
while KbCheck; end % wait until all keys are released
escapekey = KbName('escape');
while 1
    keyisdown = 0;
    while ~keyisdown
        [keyisdown, secs, keycode] = KbCheck;
        WaitSecs(0.001) % delay to prevent CPU hogging
    end
    fprintf('Current key(s) down: %s which is %s\n', ...
        char(int2str(find(keycode))), ...
        char(KbName(keycode))');
    if keycode(escapekey)
        break;
    end
end
```

Reaction time

Often you will want to measure the response time in reaction to a stimulus that you present. You might, for example, present a stimulus and instruct the subject to press a key as fast as possible thereafter. Here's an example reaction time program (on the next page). Press the space bar as soon as the white circle flashes. Extra credit for the person with the fastest mean reaction time.

Note the `WaitSecs(0.001)` command in the `KbCheck` loop. It is necessary to introduce some delays in tight loops like this, or else it will hog the CPU time and the operating system may penalize the program, shutting it down for several seconds.

```
[w,rect] = Screen('OpenWindow',0,[0 0 0]);
ntrials=10;      % number of trials
r=50;            % radius of circle in pixels
tmin = 1;        % minimum time between trials
tmax = 3;        % maximum
rtime = zeros(1,ntrials);
x0=rect(3)/2;
y0=rect(4)/2;
rkey=KbName('Space'); %response key = space bar
for i=1:ntrials
     keyisdown = 1;
     while(keyisdown) % first wait until all keys are released
          [keyisdown,secs,keycode] = KbCheck;
          WaitSecs(0.001); % delay to prevent CPU hogging
     end
     % draw fixation point
     Screen('FrameRect',w,[255 255 255],[x0-3,y0-3,x0+3,y0+3]);
     Screen('Flip',w);
     wait_time = rand * (tmax-tmin) + tmin;
     start_time = GetSecs;
     while(~keycode(rkey))
          if GetSecs-start_time > wait_time
              wait_time = Inf;    % so as not to repeat this part
              Screen('FillOval',w,[255 255 255],[x0-r,y0-r,x0+r,y0+r]);
              Screen('Flip',w);
              time0=GetSecs;
          end
          [keyisdown,secs,keycode] = KbCheck;
          WaitSecs(0.001); % delay to prevent CPU hogging
     end
     rtime(i)=secs-time0;
     Screen('Flip',w);
end
Screen('Close',w)
avg_rtime = 1000*mean(rtime)    % mean reaction time in milliseconds
```

Mouse input

In addition to the keyboard, the Psychophysics Toolbox also allows you to read the mouse using the `GetMouse` function. It works much the same way the `KbCheck` function does, returning the instantaneous state of the mouse. Try modifying the above program to read the mouse input. Note that since `GetMouse` doesn't return a `secs` argument, you'll have to issue a `secs=GetSecs` command immediately upon getting a positive click from the mouse.

Also try `MouseTraceDemo2OSX`. You should open the script and look at its code.