ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/23986167

Generating Stimuli for Neuroscience Using PsychoPy

Article *in* Frontiers in Neuroinformatics · February 2008

DOI: 10.3389/neuro.11.010.2008 · Source: PubMed

CITATIONS
877

READS
2,822

Some of the authors of this publication are also working on these related projects:

Project Nonlinear processing in mid-level vision View project

Project PsychoPy View project

All content following this page was uploaded by Jonathan Peirce on 20 May 2014.

The user has requested enhancement of the downloaded file.

frontiers in
NEUROINFORMATICS

ORIGINAL RESEARCH ARTICLE
published: 15 January 2009
doi: 10.3389/neuro.11.010.2008

Generating stimuli for neuroscience using PsychoPy

Jonathan W. Peirce*

Nottingham Visual Neuroscience, School of Psychology, University of Nottingham, Nottingham, UK

Edited by:
Rolf Kötter, Radboud University Nijmegen, The Netherlands

Reviewed by:
Andrew D. Straw, California Institute of Technology, USA
Peter Tass, Forschungszentrum Jülich, Germany

****Correspondence:***
Jonathan Peirce, School of Psychology, University of Nottingham, University Park, Nottingham NG7 2RD, UK.
e-mail: jon@peirce.org.uk

PsychoPy is a software library written in Python, using OpenGL to generate very precise visual stimuli on standard personal computers. It is designed to allow the construction of as wide a variety of neuroscience experiments as possible, with the least effort. By writing scripts in standard Python syntax users can generate an enormous variety of visual and auditory stimuli and can interact with a wide range of external hardware (enabling its use in fMRI, EEG, MEG etc.). The structure of scripts is simple and intuitive. As a result, new experiments can be written very quickly, and trying to understand a previously written script is easy, even with minimal code comments. PsychoPy can also generate movies and image sequences to be used in demos or simulated neuroscience experiments. This paper describes the range of tools and stimuli that it provides and the environment in which experiments are conducted.

Keywords: Python, psychophysics, software, neuroscience, vision, fMRI, EEG, MEG

INTRODUCTION

The majority of experiments in modern neuroscience require the presentation of auditory or visual stimuli to subjects while a measure is taken of their ability to see, remember or interact with that stimulus, or of the brain activity that results from its presentation. As a result, neuroscience needs for tools that allow the accurate presentation of stimuli and collection of participant responses. Those tools should be as easy to use as possible to reduce the time spent constructing experiments, while being able to deliver as wide a variety of stimuli and experimental designs as possible to reduce the variety of software that a single scientist needs to learn to use. Additionally the ideal software package should be open-source, such that scientists can fully examine the code and know exactly what is being done "under the hood", it should be platform independent and it should, of course, be free.

This article describes PsychoPy, an open-source software library that allows a very wide range of visual and auditory stimuli and a great variety of experimental designs to be generated within a very powerful script-driven framework based on Python. It is built entirely on open-source libraries and technologies, such that the user can, if they desire, examine all of the code that contributes to the stimuli they present. By leveraging the power of Python, and several existing cross-platform Python libraries, the software is fully platform independent and is being used in a number of labs worldwide on Windows, Mac OS X and Linux.

A previous publication (Peirce, 2007) describes the design philosophy and underlying mechanisms of PsychoPy and its relationship to other software packages, such as Vision Egg (Straw, 2008) and Psychophysics Toolbox (Brainard, 1997; Pelli, 1997). This paper focuses on its use, describing more of the variety of stimuli that the library can generate and present (images, dot arrays, text and movies), the environment in which experiments are developed and the latest developments and additions to the software.

MATERIALS AND METHODS

PsychoPy has been under active development since 2003 and, at time of writing, had reached version 0.95.2. The code is now largely stable (it is largely backward-compatible between versions) and is sufficiently complete and bug-free that it is used as the standard means of conducting psychophysical and/or neuroimaging experiments in a number of labs worldwide. The software is still very much under development however; stimuli are still being added, code is still being optimised and the user interface is being refined constantly. There is a mailing list where users can report bugs, discuss improvements and get help in general use of the software.

PYTHON

One of the strengths of PsychoPy is its use of Python. The high-level functions and libraries available in Python make it an ideal language in which to develop such software. The platform independence that PsychoPy enjoys is based very much on the fact that it is based on pure Python code, using libraries such as *wxPython*, *pyglet* and *numpy* that have been written to be as platform independent as is technically possible. The fact that Python now has such a large user base means that there is a large community of excellent programmers developing libraries that PsychoPy can make use of. The fact that Python can be used in such a wide variety of ways (for example, in the author's own lab Python is used not only for stimulus presentation but also for data analysis, for the generation of publication-quality figures, for computational modelling and for various general purpose scripts to manipulate files) means that in many cases this is likely to be the only programming language that a scientist need learn, with the obvious benefits in time that result. By nature of its clean, readable, and powerful syntax combined with its free and open-source release model Python is clearly a very popular language that is continuously growing and developing further. Where Matlab has, in the past, benefited from its large user base and wide variety of applications to science, Python stands to benefit even more.

HARDWARE ACCELERATED GRAPHICS

One of the goals of PsychoPy was to generate stimuli in *real-time*, that is to update the character of a stimulus on a frame-by-frame basis as needed without losing temporal precision. For static stimuli this is an

unnecessary benefit, but for moving stimuli, where the alternative is to pre-compute a movie sequence it makes for much cleaner experimental code, with fewer delays (some experiments would previously require several seconds or even minutes before running where they computed the stimulus movies). The possibility of real-time stimulus manipulations also allows experiments to alter based on input form the participant such that, for example, a stimulus might be moved fluidly under mouse (or even eye-movement) control, or the next stimulus can be generated based on the previous response.

In order to achieve good temporal precision, while updating stimuli in real-time from an interpreted language like Python or Matlab, it has been essential to make good use of the hardware accelerated graphics capabilities of modern computers. Most modern machines have very powerful graphics processing units that can perform a lot of the calculations necessary to present stimuli at a precise point in space and time and to update that stimulus frequently. The OpenGL specification determines, fairly precisely, what a graphics card should do given various commands, such that platform independence is largely maintained (there are certain aspects, such as the synchronisation of drawing with the screen vertical refresh that are graphics card and/or platform dependent). PsychoPy 0.95 is fully compatible with the OpenGL 1.5 specification but makes use of further facilities that were added to OpenGL 2.0 on graphics cards and drivers where these are available. Nearly all modern graphics cards are capable of using OpenGL (although they may need updated drivers) and perfectly adequate cards from nVidia or ATI, that support the OpenGL 2.0 extensions, can be currently purchased and added to a desktop computer of any platform for roughly £30.

PLATFORM INDEPENDENCE

Platform independence is a particular goal of PsychoPy. Computer technologies change rapidly and the relative advantages of different platforms can vary equally quickly. Scientists should not need to learn a whole new set of tools just because they have decided to switch their main computer platform, and should be able to share code and experiments with colleagues using other platforms. Perfect independence is never possible because of hardware differences between computers. Some such differences are obvious; for example, Apple Macs have not supported parallel ports directly for several years so scripts using parallel port communication cannot work on those platforms. Other differences are subtle and unnoticed by most users. An example of this is that the OpenGL specification allows for the frame not to be cleared after a swap of the "front" and "back" buffers during a screen refresh, but does not specify whether the new back buffer is maintained from the previous back buffer (most useful for the continuity of drawing frames) or retrieved from the previous front buffer (as implied by the term "swapping" buffers). As a result, the behaviour is free to, and does, vary between manufacturers.

In the *vast* majority of cases, however, thanks to the hard work of the developers of libraries such as *pyglet*, *numpy* and *wxPython*, a PsychoPy script will run identically on all platforms.

RESULTS

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

PsychoPy was developed as a Python package that could be imported from scripts needing to present stimuli. For new users of Python that has certain disadvantages; users need to install Python and other dependent libraries separately, they need some form of text editor to write the scripts and they need to know where to find the text, including error messages, that scripts might output. Although none of these are difficult (and may seem obvious to an experienced programmer or user of command-line operating systems), they were impediments to new users, particularly from Windows and "traditional" Mac platforms. PsychoPy now comes with a built-in code editor (PsychoPyIDE), complete with code auto-completion, code folding and help tips. Scripts can be run directly from the editor and code output is directed to another window in the application (see **Figure 1**). When this output includes error messages these show up as URL-style links that take the user directly to the line on which the error occurred.

On Windows, installation is very straightforward using simple double-clickable installers. On Intel-based Apple Macintosh computers running OS X an application bundle is provided that contains its own copy of Python and all the dependent libraries. This has a number of advantages. The first is that it installs simply as a single application that can be dragged into the Applications folder (or other location) and can be removed equally easily by simply sending to the trash. As well as being easy to install by this method, distributing PsychoPy with its own copy of Python has two major advantages: PsychoPy's developers know what libraries have been installed and that they are compatible and the user knows that it won't interfere with any existing Python installation that they have (such as previous installs, or the Apple system Python). For more experienced Python users, who may wish to install to their own customised set of libraries, the standard Python-style methods of installing from source distributions are also available.

On Linux the dependencies can be installed simply from simple `apt-get` commands and PsychoPy is then easily installed from its source distribution.

MODULE STRUCTURE

As with most Python packages, PsychoPy contains a number of submodules, which can be imported relatively independently (some depend on each other) depending on the task at hand. This is useful in keeping related functions and classes together in meaningful units. For instance, the following will import modules useful in presenting visual and auditory stimuli and collecting responses (events) from the subject:

```
from psychopy import visual, core, event
```

The main modules that can be imported from PsychoPy, and the main libraries that they depend upon are shown in **Figure 2**.

PRESENTING STIMULI

A subset of the available visual stimuli is shown as a screenshot in **Figure 3**.

Windows

Most experiments begin with creating a window into which visual stimuli or instructions can be presented. In PsychoPy this can be achieved in a full screen mode or in a normal window, with the mouse either shown or hidden. Furthermore, multiple windows can be created at one time and these may be presented on any physical

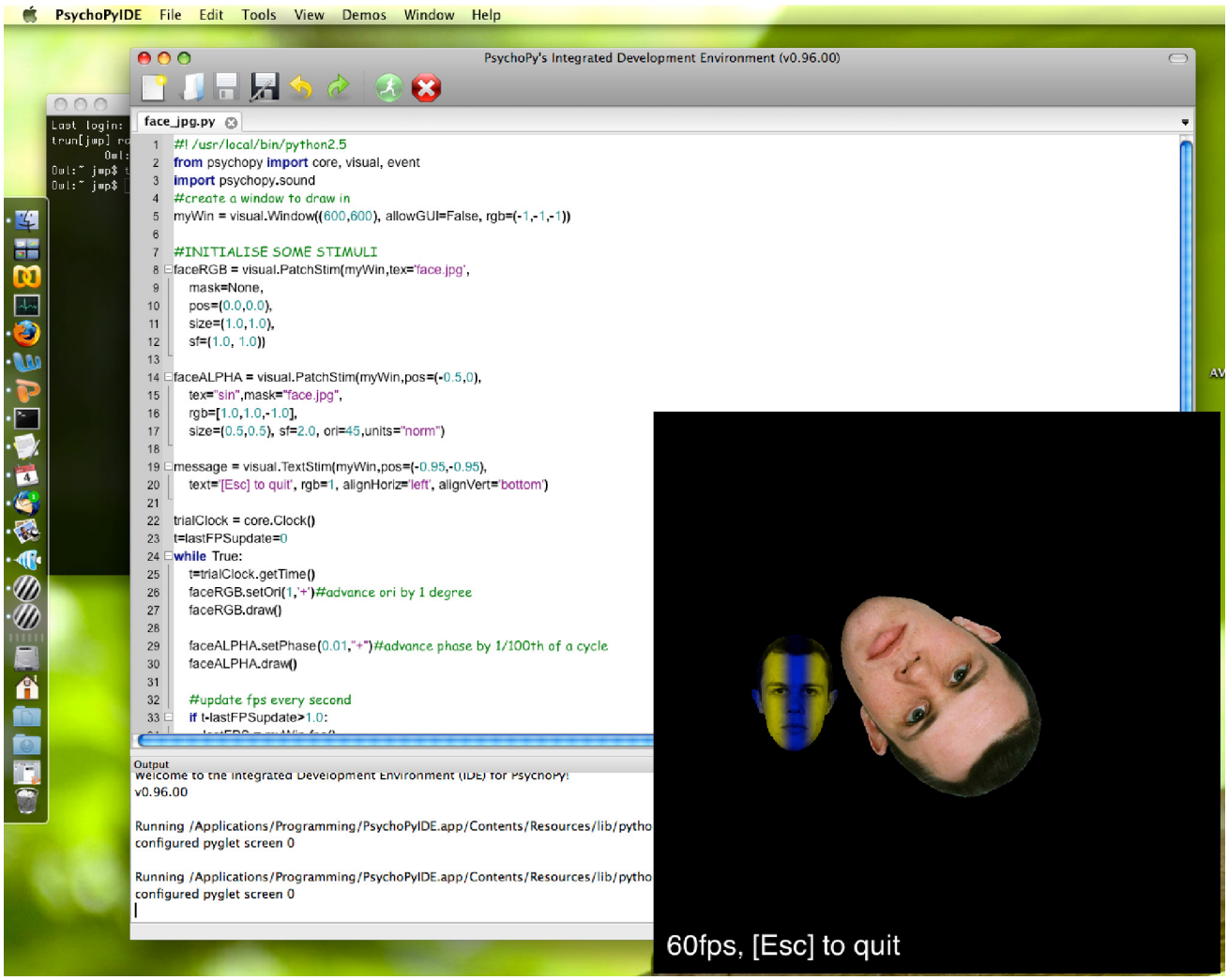

FIGURE 1 |The integrated development environment (IDE) running one of the demo scripts. Multiple scripts can be opened at once in the editor, appearing as tabs. There is a menu from which demos can be easily loaded for a quick view of how to use various aspects of the program. Output from the running script is displayed in the panel at the bottom of the window and scripts can be started and forced to quit directly from the editor. Although the OS X version is shown here, the editor runs on all platforms.

screen if more than one is connected. This makes the presentation of binocular stimuli straightforward.

PsychoPy windows can also be given information about the monitor that they are being presented in, such as its physical size and distance from the participant (this information can be provided as part of the script or from a dialogue box as part of the development environment). Once provided with the necessary information PsychoPy will then allow the user to specify their stimulus attributes such as size and location in any of a variety of meaningful units, such as cm or degrees of visual angle. If the monitor has been colour calibrated with a spectro-radiometer, a process which can also be automated from within PsychoPy, then the colour of stimuli can also be specified in a biologically relevant colour space. For example, using the MB-DKL cone-opponent space (Derrington et al., 1984; MacLeod and Boynton, 1979) allows isoluminant stimuli to be generated trivially from within scripts.

Windows are double-buffered, meaning that any drawing commands are initially executed to a hidden window (the back buffer) and are only translated to the screen on the next vertical blank (VBL) period after the `Window.flip()` command has been called. On most systems (a very small number of graphics card do not support the feature) this will then pause the running of the thread, such that no further commands are executed until the frame has been refreshed. This feature of synchronising to the VBL can be used as a mechanism to control timing during an experiment, since the period between VBLs is extremely consistent and precise.

PatchStim

The most widely-used stimulus in PsychoPy is the PatchStim, used to control a visual patch on the screen. Patches can contain any bitmap-style data, including periodic textures (such as sinusoidal gratings or repetitive lines) or photographic images. These

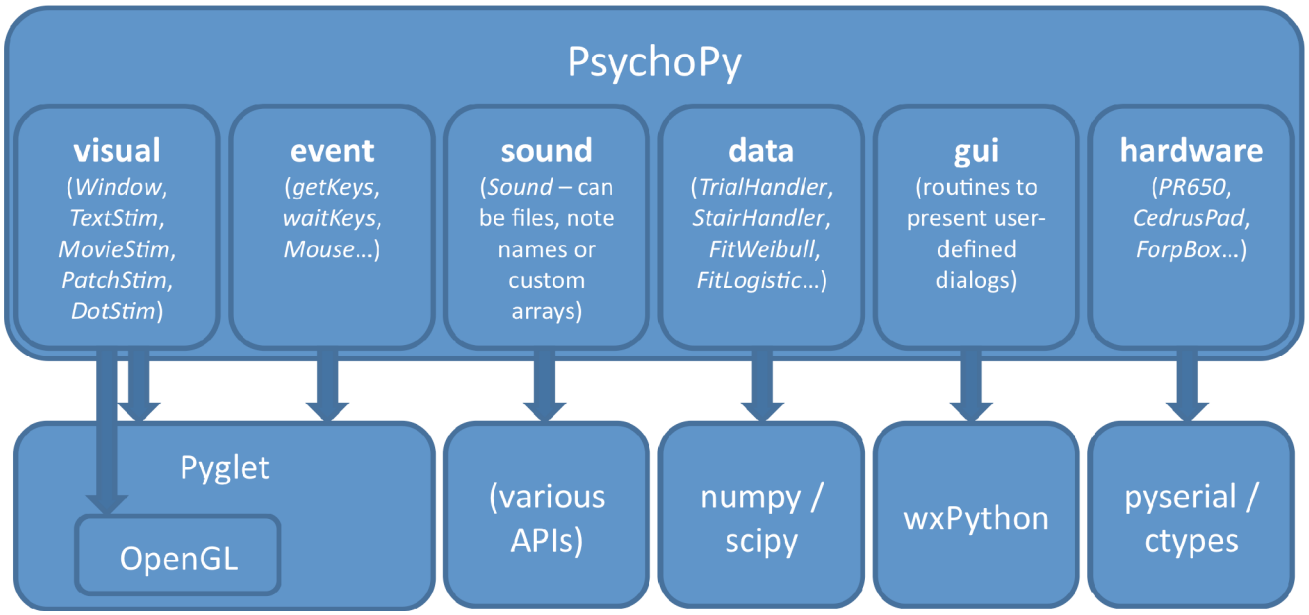

FIGURE 2 | The structure of PsychoPy. PsychoPy comprises a number of sub-modules for controlling different aspects of an experimental setup, from stimulus presentation to analysis of data. In turn these use a number of dependent libraries, that typically have a very good degree of platform-independence.

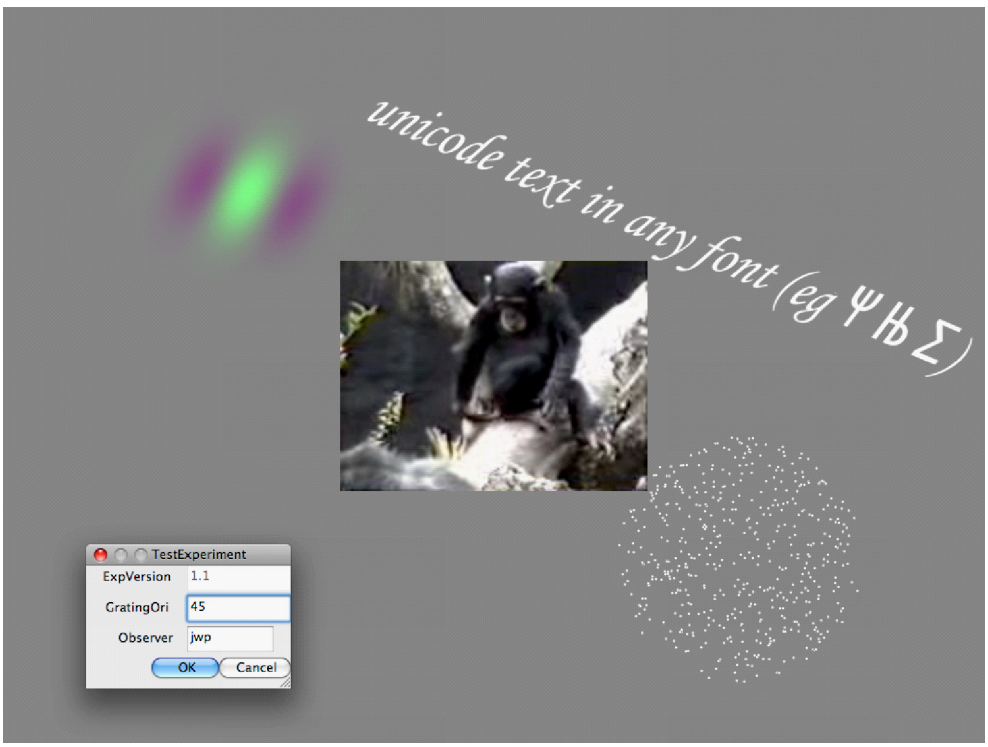

FIGURE 3 | A sample of PsychoPy components. Within the `Window` is a coloured Gabor from `PatchStim`, some rotated Unicode text from the `TextStim` and 500-dot `DotStim`. The central image is actually a `MovieStim`. All the stimuli are dynamic and being updated simultaneously at 60Hz, without any dropped frames. Also shown is a dialog (`gui.DlgFromDict`) to receive information about the current experiment.

also support alpha masks, which define the transparency of the stimulus across the patch and can therefore determine the shape, or "envelope" of the stimulus. These stimuli can be manipulated in real-time in a wide variety of useful ways; the bitmaps can be rotated, have their phase shifted, change the number of cycles in either dimension etc.

As a result, PatchStim stimuli can be used to present a wide variety of image-based objects, either those used typically in visual psychophysics (gratings, Gabors etc…) or those in higher-level psychology and cognitive neuroscience studies (such as photographic images) or to create simple geometric shapes such as fixation points and arrows.

TextStim

Another common experimental requirement is the presentation of text to subjects, either as instructions or as actual stimuli. PsychoPy has a stimulus that provides simple access to clear, anti-aliased text in any true-type font available on the host system (obviously more can be installed). These stimuli are fully compatible with Unicode, so that symbols and non-English characters can be included. Text objects can be coloured in any of the colour spaces and referred to by any coordinate system for which the window has been calibrated (see Windows). They can also be rotated arbitrarily and in real-time.

Sound

PsychoPy also provides direct and simple access to methods for presenting auditory stimuli. Sound objects can be created from files (wav, mpg), from pure tones (the user specifies the duration and either frequency or the name of the note and octave on a standard scale) or can be generated from arbitrary waveforms using the standard *numpy* library in Python. Sound objects can be played in full stereo in asynchronous threads, so as to overlap as necessary with each other and with visual presentations.

The ability to play arbitrary stereo waveforms as sounds makes PsychoPy perfectly capable of running full auditory psychophysical experiments, but the sounds can equally easily be used just to present feedback tones to subjects carrying out basic experimental tasks.

DotStim

A common stimulus in visual neuroscience is the random dot pattern (e.g. see Scase et al., 1996), also known as the Random Dot Kinematogram and this is provided in PsychoPy by the DotStim object. This allows either an array of dots, or an array of other PsychoPy stimuli (e.g. PatchStims) to be drawn as a field. The position of the dot elements can then be automatically updated by a variety of rules, for instance where a number of target dots move in a given direction while the remaining (distracter) dots move in random directions. This type of stimulus makes heavy use of OpenGL optimisations and allows a large number of dot elements (several hundred) to be drawn and updated in realtime without dropping frames.

MovieStim

PsychoPy can present movies in a variety of formats including mpeg, DivX, avi and Quicktime, allowing studies using natural scene stimuli or biological motion displays. As with most other stimulus types, these can also be transformed in a variety of ways (e.g. rotated, flipped, stretched) in real-time.

COLLECTING RESPONSES

Most experiments also need to receive and store information about responses from subjects. For PsychoPy, this can be achieved via a number of simple means; keyboards, mice, joysticks and specialised hardware such as button boxes. The simplest possible input method is to examine recent events from the keyboard using the `event.getKeys()` and `event.waitKeys()` functions. These allow the user to see what keys have been pressed since the last call or to wait until one has been pressed (and may be restricted to a small number of allowed keys). The `event.Mouse` object allows PsychoPy users to determine where the mouse is at any given moment or whether a mouse button has been pressed with simple methods such as `getPos()`, `getWheelRel()` (to retrieve the relative movement of the mouse scroll wheel) and `getPressed()`. **Code Snippet 1** demonstrates how to use these mouse and keyboard facilities to control a drifting Gabor patch (a sinusoidal grating in a Gaussian-shaped envelope) in real-time within a PsychoPy window.

INTEGRATING WITH HARDWARE

Many input/output devices can be accessed directly from within PsychoPy by emulating keyboards or rodents. For example, the fORP MR-compatible button boxes (Current Designs, Philadelphia, USA) are capable of outputting signals that emulate key presses on a standard keyboard (e.g. keys 1–4 can represent buttons with key 5 representing a trigger pulse from an MRI scanner). Many touch-sensitive screens simply emulate a mouse press at the location where the screen was touched, and can therefore be used within PsychoPy as if a mouse event had occurred. These often provide the simplest methods of input to an experimental program. On other occasions these are unsuitable, either because the nature of the information being transmitted does not easily emulate such devices or because those devices are already in use. For example, what happens if you need button-box input as well as, and separate from, keyboard input?

PsychoPy also provides simple and complete access to input and output via serial and parallel ports (or via USB serial/parallel emulators, on systems where direct hardware ports are unavailable). An example of the use of serial and parallel port communications is shown in **Code Snippet 2**. Typically the parallel port is used to control and receive simple triggers in switching a current from high (+5 V) to low (0 V) or vice-versa and particularly useful in informing other hardware (such as an Electroencephalography device) of the precise onset of an event in PsychoPy. Serial ports can be used to pass more complex information, such as text characters or data in bytes at a fixed rate and are still heavily used by a large number of scientific devices because of their relative simplicity. For example, PsychoPy uses the serial port protocol to communicate with a PR650 spectrophotometer (Photo Research Inc, Chatsworth, USA) sending commands to begin measurements and receiving data back from the device such as the full power spectrum of the currently presented screen.

Some devices may also make use of calls from binary-compiled dynamically-loaded libraries (dlls on the Windows platform, dylibs on OS X). In particular most devices connecting via USB, Firewire or PCI cards will come with drivers that fall into this category. Python provides a module called `ctypes` (as of version 2.5), which allows seamless calls to any such drivers and dynamic libraries directly from Python itself.

Through one of these methods, any hardware that can communicate with your computer, can also communicate with Python and PsychoPy.

```
from psychopy import visual, core, event # import the PsychoPy libraries

#create a window to draw in
myWin = visual.Window((600.0,600.0), allowGUI=True)

#initialise some stimuli
fixSpot = visual.PatchStim(myWin,
        tex="none", mask="gauss", #no texture and a Gaussian shape
        pos=(0,0), size=(0.05,0.05), #size and location as fraction of window
        rgb=[-1.0,-1.0,-1.0]) #the colour of the fixation (black)
grating = visual.PatchStim(myWin,pos=(0.5,0),
        tex="sin",mask="gauss", #grating texture and a Gaussian shape
        rgb=[1.0,0.5,-1.0], #
        size=(1.0,1.0), sf=(3,0)) #set the size and the grating cycles
myMouse = event.Mouse(win=myWin) #a mouse object related to our window
message = visual.TextStim(myWin,pos=(-0.95,-0.9), #a TextStim to provide info
        alignHoriz='left', height=0.08,#specifying the size of the font
        text='left-drag=SF, right-drag=pos, scroll=ori') #and the actual text

for frameN in range(2000): #for 2000 frames
    #handle key presses each frame
    for key in event.getKeys(): #returns keys pressed this frame
        if key in ['escape','q']:
            core.quit()

    #get mouse events
    mouse_dX,mouse_dY = myMouse.getRel() #get position relative to previous
    mouse1, mouse2, mouse3 = myMouse.getPressed()
    #based on the mouse button and change in position, change the stimulus
    if (mouse1): #if button 1 is down (ie left-click)
        grating.setSF(mouse_dX/200.0, '+')
    elif (mouse3): #else if button 3 is down (ie right-click)
        grating.setPos([mouse_dX/400.0, -mouse_dY/400.0], '+')

    #Handle the mouse wheel(s)
    wheel_dX, wheel_dY = myMouse.getWheelRel()
    #change the grating orientation according to the wheel
    grating.setOri(wheel_dY*5, '+') #2 clicks will give 10deg rotation
    event.clearEvents() #get rid of other, unprocessed events

    #draw our stimuli (every frame)
    fixSpot.draw() #visual stimuli have a simple 'draw' function
    grating.setPhase(0.05, '+') #advance grating by 0.05 cycles per frame
    grating.draw()
    message.draw()
    myWin.flip() #update the window

core.quit() #when we're done (Python loops finish when code indentation ends)
```

CODE SNIPPET 1 | Presenting stimuli under real-time control. This demo script controls a drifting grating in real-time according to input from the mouse. It demonstrates the use of the Window, PatchStim, TextStim and Mouse objects and how to get keyboard input from the participant. These objects have associated methods that allow them to have their attributes changed.

TIMING

Timing is a critical issue for many experiments in neuroscience and psychology. Many studies require a temporal precision to within a few milliseconds, or even in the sub-millisecond range. PsychoPy provides various methods to achieve very precise timing of events and to synchronise with other devices. This is achieved by means of synchronising drawing to the VBL of the monitor, by the use of very precise clocks on the host CPU and by access to rapid communication ports such as the serial and parallel ports.

PsychoPy (like most such software) uses a double-buffered method of rendering, whereby stimuli are initially drawn into a back buffer, a virtual screen in the memory of the graphics card.

```
from psychopy import core, parallel, serial

#initialise ports
serialPort = serial.Serial("COM1", baudrate=115200, bytesize=8, parity='N',
      stopbits=1, timeout=0.0001)
parallel.setPortAddress(0x378) #need to know your parallel port address

#set pin 2 to high and send a command to Cedrus RB730
parallel.setPin(pinNumber=2, state=1) #set pin 2 to high
serialPort.writelines("_d1") #send a command to the serial port

core.wait(0.5)

#set pin 2 to low and read response from Cedrus RB730
parallel.setPin(pinNumber=2, state=0) #set pin 2 to low
nCharsToGet = serialPort.inWaiting()
message = serialPort.read(nCharsToGet)#read the current characters
print message
```

CODE SNIPPET 2 | The use of serial and parallel ports to control hardware and synchronisation. The demo sends a command to the serial port (in this case the command would request information from a Cedrus box about its type and version) and reads the response after a 0.5-s pause. During this period pin 2 on the parallel port is set to high.

At the point when the VBL occurs (signifying the end of one frame and the beginning of the next) the contents of this back buffer are flipped with the actual screen buffer. When the command `Window.flip()` is sent, PsychoPy will halt all processing (or processing just in this thread if multiple threads are being used) until the graphics card signals that a frame flip has occurred. Since these frame flips occur at a very precise interval they can be used as a very precise timing mechanism and by executing a command immediately after the flip one can be certain that it is time-locked to the presentation of that stimulus frame.

The precision of this system can break down when frames are dropped – if too many commands are attempted (e.g. too many stimuli are drawn) between frames then the VBL may occur before the request to flip the buffers occurred, in which case the frame will remain unchanged for twice the normal period. In some cases this will be unimportant (e.g. if it occurs during an inter-trial interval it is likely to be irrelevant). At other times it could cause a slip in the timing of the study, causing a stimulus to be presented longer than intended. For dynamic stimuli it may change the perceptual appearance of the stimulus, causing a smoothly-moving stimulus to stutter in its motion, for instance.

PsychoPy alleviates this hazard by using the graphics card processor as much as possible for calculations involved in drawing, such as the transformations needed in rotating, scaling and blending multiple stimuli. For simple experiments, using just a few standard stimuli, almost any modern computer is likely to have the processing power to draw multiple stimuli without dropping frames. For studies needing large numbers of stimuli updating every frame, the need for faster computers and graphics cards exerts itself. In particular, the use of computers with "onboard" graphics processors (such as the GMA 950 graphics processor that comes on many Intel processors) is not recommended – even the cheapest nVidia and ATI graphics cards will easily outperform these chips. Also, as complexity increases, so does the need to write more efficient experiment scripts. Often this is simply a case of finding ways to reduce the number of commands executed, for example by manipulating large lists of numbers as *numpy* arrays rather than iterating operations in for-loops. Sometimes it may mean having a better understanding of the speed of operations that will result from the command – giving a `PatchStim` a new texture is time-consuming if the texture is large, whereas changing its orientation or colour has a relatively small overhead, so preloading textures into stimuli is a good idea whenever possible.

Although PsychoPy and Python are potentially (subject to a well-written script) very precise in their reporting and generation of stimuli, there are a number of hardware limitations in most experimental setups that limit the absolute temporal accuracy of studies. The most obvious is the temporal resolution of the presentation device (typically a monitor or projector) but many experimenters are also unaware of the inherent latencies of other hardware components in their system. In general, these limit the accuracy rather than precision of the studies, since the latencies are relatively constant, but are nevertheless worthy of exploration.

Frame rates and monitor technology

The most fundamental limitation to the temporal precision of most studies is the frame rate of the monitor, and this varies dependent on the particular monitor technology. Cathode ray tube screens typically operate at refresh rates ranging 60–200 Hz, dependent on the monitor and the resolution of the display. For the majority of the frame period (say 12 ms for an 85-Hz refresh rate) pixels are being drawn sequentially in lines progressing from the top of the screen to the bottom. When the beam illuminating the pixels reaches the bottom of the screen there is a pause of around 1.5 ms while it returns to top, ready to draw the next frame (this is the VBL period). The obvious result is that visual stimuli cannot be changed at a rate greater than the frame rate – when a stimulus is scheduled for drawing, for example following some user response, it cannot be drawn until the next refresh of the screen. A less obvious result is that stimuli are drawn as much as 10 ms apart, even on the same frame, depending on their screen position.

LCD panel displays (either projectors or monitors) are typically limited to a screen refresh rate of 60 Hz and therefore share the problem of having a limited rate at which stimuli can be changed. They do not, however, draw the lines to the screen sequentially and so do not suffer from the problem that parts of the screen are drawn before others. On the other hand, the response time of these displays is considerably slower – an LCD switching from black to white changes rather gradually, over a period of around 20 ms. In cases where the screen is changed very rapidly this can have profound effects. For instance, if a stimulus is intended to flash black and white on alternating screens, it is unlikely on these monitors to reach full black and full white and a lower contrast stimulus will result.

The use of USB devices

Commonly the need for timing accuracy comes from the need to know how long a participant took to respond to the presentation of a stimulus, where their response is measured by pressing a button on a keyboard or response box. Unfortunately these devices are often USB-based and this introduces another temporal lag of, typically, 10–20 ms. Again, for a given device and computer system it is likely to be relatively constant, affecting the absolute accuracy of the response time measurement more than the precision.

DISCUSSION

PsychoPy is already a very useful tool for running experiments that require visual and auditory stimuli in a wide variety of environments. It is platform-independent, entirely free, simple to use and extremely versatile. It is also continuously improving in the variety of stimuli it can present, the accuracy and speed with which it can present them and in its ease of installation and use. As an open-source project its continued development benefits from its increasing user base, and that of the wider Python community. Python is also a language suitable for a wide variety of other tasks, including complex data analysis and computational modelling. Data can be shared easily between PsychoPy and other Python-based packages (e.g. using stored *numpy* arrays), or can be exported to other programs using comma-separated or tab-delimited text files.

The variety of stimuli that PsychoPy can produce and its temporal precision in generating these in real-time make it an ideal environment for many neuroscience endeavours. It was originally designed for psychophysical studies in vision, but is also an ideal package for presenting stimuli in more traditional cognitive psychology experiments, including the ability to interface with touch-screens and, by virtue of its simple interface to parallel and serial ports, it is already being used by a number of labs for fMRI, MEG, EEG. PsychoPy is relatively young. Although it has been used as standard in the author's lab since 2004 it has been used in other labs only since 2006. The community around it is growing however; at the time of writing the package had been downloaded 5000 times and has an active mailing list with 50 members.

A great deal more information is available from the project's website (http://www.psychopy.org), including tutorials, demonstration code and reference material for the writing of scripts.

ACKNOWLEDGEMENTS

PsychoPy has been developed with support from a BBSRC project grant (BB/C50289X/1), a Wellcome Trust Grant and seed funding grants from The Royal Society and the University of Nottingham. Many thanks to all those that have provided constructive criticism, and destructive testing, especially Dr. B.S. Webb.

REFERENCES

Brainard, D. H. (1997). The psychophysics toolbox. *Spat. Vis.* 10, 433–436.

Derrington, A. M., Krauskopf, J., and Lennie, P. (1984). Chromatic mechanisms in lateral geniculate nucleus of macaque. *J. Physiol.* 357, 241–265.

MacLeod, D. I., and Boynton, R. M. (1979). Chromaticity diagram showing cone excitation by stimuli of equal luminance. *J. Opt. Soc. Am.* 69, 1183–1186.

Peirce, J. W. (2007). PsychoPy-Psychophysics software in Python. *J. Neurosci. Methods* 162, 8–13.

Pelli, D. G. (1997). The VideoToolbox software for visual psychophysics: transforming numbers into movies. *Spat. Vis.* 10, 437–442.

Scase, M. O., Braddick, O. J., and Raymond, J. E. (1996). What is noise for the motion system? *Vision Res.* 36, 2579–2586.

Straw, A. D. (2008). Vision egg: an open-source library for realtime visual stimulus generation. *Front. Neuroinformatics* 2, 4.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 09 September 2008; paper pending published: 27 October 2008; accepted: 19 December 2008; published online: 15 January 2009.

*Citation: Peirce JW (2009) Generating stimuli for neuroscience using PsychoPy. Front. Neuroinform. (2009) **2**:10. doi: 10.3389/neuro.11.010.2008*

Copyright © 2009 Peirce. This is an open-access article subject to an exclusive license agreement between the authors and the Frontiers Research Foundation, which permits unrestricted use, distribution, and reproduction in any medium, provided the original authors and source are credited.

View publication stats