
Vision Egg Documentation

Release 1.2.1

Andrew Straw

September 21, 2014

1	About	3
2	Citing the Vision Egg	5
3	Quick links	7
4	Full Documentation Contents	9
4.1	Vision Egg Citations	9
4.2	Documentation	11
4.3	Documentation/FAQ	11
4.4	Documentation/Tutorial	15
4.5	Download and Install	21
4.6	Download and Install/An Installation Overview	21
4.7	Download and Install/Install on IRIX	22
4.8	Download and Install/Install on Linux	23
4.9	Download and Install/Install on MacOSX	26
4.10	Download and Install/Install on Windows	29
4.11	Intro and Overview/About this wiki	30
4.12	Intro and Overview/Calibration	31
4.13	Intro and Overview/FrameRates	31
4.14	Intro and Overview/Platforms	33
4.15	Screenshots	34
4.16	Intro and Overview/Synchronization	42
4.17	Intro and Overview/Technologies	43
4.18	Miscellaneous	45
4.19	Miscellaneous/CreditsAndThanks	46
4.20	Miscellaneous/LabView	47
4.21	Miscellaneous/SimilarSoftware	48
4.22	Miscellaneous/UsersAndFriends	49
4.23	News	49
4.24	Quest	53
5	Indices and tables	57

News

1. *Frontiers in Neuroscience commentary* - 2009-10-05
2. *talk at SciPy 09 (video available online)* - 2009-08-20
3. *talk and tutorial at CNS*2009* - 2009-07-22
4. *Vision Egg 1.2.1 released* - 2009-07-21
5. *Vision Egg article in Frontiers in Neuroinformatics* - 2008-10-08
6. *BCPy2000, using the Vision Egg, released* - 2008-10-01

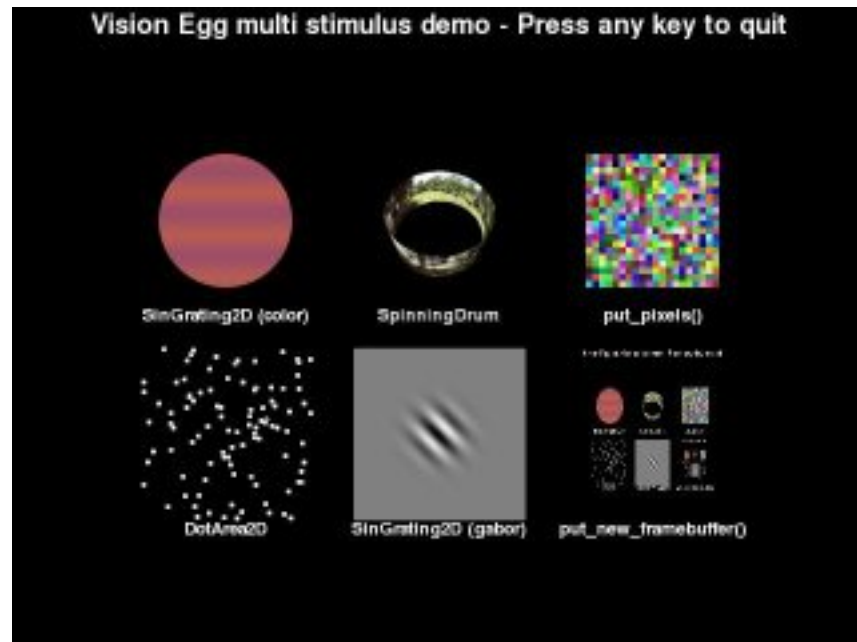
The Vision Egg is a powerful, flexible, and free way to produce stimuli for vision research experiments.

About

The Vision Egg is a high level interface between Python and OpenGL. In addition to methods for automatic generation of traditional visual stimuli such as sinusoidal gratings and random dot patterns, it has a number of functions for moving numeric data, images, movies, text, and 3D objects to and from your video card and allowing use of some of its features like perspective distortion. Therefore, it is also useful for anyone wishing to make use of the features of today's graphics cards.

- Perform experiments using an inexpensive PC and standard consumer graphics card
- Perform experiments using a graphics workstation if special features needed
- Data acquisition and other realtime hardware control capabilities useful in electrophysiology and fMRI experiments, including gaze-contingent stimuli
- Dynamically generated stimuli can be changed in realtime via software or external hardware
- Produce traditional stimuli to replace legacy systems
- Produce stimuli not possible using other hardware
- Demo programs to get you started right away
- Run stimuli on your laptop - great for talks
- Free, open-source software

Screenshot



See the *Screenshots*.

By harnessing the power of today's consumer graphics cards, producing visual stimuli of research quality now requires no specialized hardware beyond a relatively recent computer and graphics card.

Based on open standards, it runs on anything from cheap PCs to expensive special hardware for special needs. For example, running on some platforms, such as SGI workstations, the Vision Egg has a 10-bit luminance dynamic range (both pixel depth and DAC) and precise frame-by-frame control.

The Vision Egg is open source software (GNU LGPL). Therefore, you can be assured of a product that meets your needs but does not lock you in. Download it today and give it a try!

Citing the Vision Egg

If you use the Vision Egg for anything that results in a scientific publication, I ask that you please cite my paper in your publications. The citation is:

- Straw, Andrew D. (2008) Vision Egg: An Open-Source Library for Realtime Visual Stimulus Generation. *Frontiers in Neuroinformatics*. doi: 10.3389/neuro.11.004.2008 [link](#)

To see a partial list of publications that made use of the VisionEgg, see [Vision Egg Citations](#).

Quick links

- [downloads](#) (Check the *Download and Install* page for more information.)
- [Vision Egg @github](#)

Full Documentation Contents

Contents:

4.1 Vision Egg Citations

4.1.1 Citing the Vision Egg

If you use the Vision Egg for anything that results in a scientific publication, I ask that you please cite my paper in your publications. The citation is:

- Straw, Andrew D. (2008) Vision Egg: An Open-Source Library for Realtime Visual Stimulus Generation. *Frontiers in Neuroinformatics*. doi: 10.3389/neuro.11.004.2008 [link](#)

4.1.2 Papers using the Vision Egg

See also [Google Scholar's list of papers that cite the Vision Egg manuscript](#).

(Please add your papers here.)

- Sakano, Y., & Allison, R. S. (2014). Aftereffect of motion-in-depth based on binocular cues: Effects of adaptation duration, interocular correlation, and temporal correlation. *Journal of Vision*, 14(8):21, 1-14, [link](#), doi:10.1167/14.8.21.
- Sakano, Y., Allison, R. S., & Howard, I. P. (2012). Motion aftereffect in depth based on binocular information. *Journal of Vision* 12(1):11, 1–15, [link](#), doi:10.1167/12.1.11.
- Scalf, P.E., Beck, D.M. (2010) Competition in Visual Cortex Impedes Attention to Multiple Items. *Journal of Neuroscience* 30(1): 161-169 [link](#)
- Helia B. Schonhaler, Tamara A. Franz-Odenaal, Corinne Hodel, Ines Gehring, Robert Geisler, Heinz Schwarz, Stephan C.F. Neuhauss and Ralf Dahm (2010) The zebrafish mutant bumper shows a hyperproliferation of lens epithelial cells and fibre cell degeneration leading to functional blindness. *Mechanisms of Development* [link](#)
- Walther, D. B., Caddigan, E., Fei-Fei, L. and Beck, D.M. (2010) Natural Scene Categories Revealed in Distributed Patterns of Activity in the Human Brain. *Journal of Neuroscience* [link](#)
- Bissig, D., Berkowitz, B.A., (2009). Manganese-enhanced MRI of layer-specific activity in the visual cortex from awake and free-moving rats. *Neuroimage*, 44(3):627-635. PMID: 19015035 doi: doi:10.1016/j.neuroimage.2008.10.013
- Mueller, J.P., Neuhauss, S.C.F. (2009) Quantitative measurements of the optokinetic response in adult fish . *Journal of Neuroscience Methods*. [link](#)

- Linares, D., Holcombe, A.O., White, A.L. (2009) Where is the moving object now? Judgments of instantaneous position show poor temporal precision (SD = 70 ms). *Journal of Vision*. [link](#)
- Bolzon, DM, Nordström, K & O'Carroll, DC (2009) Local and Large-Range Inhibition in Feature Detection *J. Neurosci.* 9(45):14143-14150; doi:10.1523/JNEUROSCI.2857-09.2009
- Nordström, K & O'Carroll, DC (2009) The motion after-effect: local and global contributions to contrast sensitivity *Proc.Roy.Soc.Lond. B.* 276:1545-1554; doi:10.1098/rspb.2008.1932
- Angelique C. Paulk, James Phillips-Portillo, Andrew M. Dacks, Jean-Marc Fellous, and Wulfila Gronenberg (2008) The Processing of Color, Motion, and Stimulus Timing Are Anatomically Segregated in the Bumblebee Brain. *The Journal of Neuroscience* June 18, 2008, **28**(25):6319-6332; doi:10.1523/JNEUROSCI.1196-08.2008 [link](#)
- Paulk AC, Gronenberg W. 2008. Higher order visual input to the mushroom bodies in the bee, *Bombus impatiens*. *Arthropod Struct & Dev* Volume 37, Issue 6, Pages 443-458 (November 2008) doi:10.1016/j.asd.2008.03.002
- Nordström K., Barnett P., Moyer de Miguel I., Brinkworth R., O'Carroll D.C. (2008) Sexual Dimorphism in the Hoverfly Motion Vision Pathway. *Current Biology*, Volume 18 ,Issue 9, Pages 661 - 667. doi:10.1016/j.cub.2008.03.061
- Fry, S. N., Rohrseitz, N., Straw, A. D. and Dickinson, M. H. (2008). **TrackFly** - Virtual Reality for a behavioral system analysis in free-flying fruit flies *J. Neurosci. Methods*. doi: 10.1016/j.jneumeth.2008.02.016
- Halko, M. A., Mingolla, E., & Somers, D. C. (2008). Multiple mechanisms of illusory contour perception. *Journal of Vision*, 8(11):17, 1-17, <http://journalofvision.org/8/11/17/>, doi:10.1167/8.11.17.
- Howard, C.J. & Holcombe, A.O. (2008) Tracking the changing features of multiple objects: Progressively poorer precision and progressively greater lag. *Vision Research*, 48(9):1164-1180. [pdf](#)
- Wiederman SD, Shoemaker PA, O'Carroll DC (2008) A Model for the Detection of Moving Targets in Visual Clutter Inspired by Insect Physiology. *PLoS ONE* 3(7): e2784 doi:10.1371/journal.pone.0002784
- Graham, L.J., del Abajo, R., Gener, T. and Fernandez, E. (2007) A Method of Combined Single-cell Electrophysiology and Electroporation. *Journal of Neuroscience Methods* V160(1): 69-74
- Geurten, BRH, Nordström, K, Sprayberry, JDH, Bolzon, DM & O'Carroll, DC (2007) Neural mechanisms underlying target detection in a dragonfly centrifugal neuron. *J. Exp.Biol.* **210**, 3277-3284 doi:10.1242/jeb.008425
- Nordström K, Barnett PD, O'Carroll DC (2006) Insect detection of small targets moving in visual clutter. *PLoS Biology* **4**(3): 378-386. [link](#)
- Straw, A.D., Warrant, E.J., & O'Carroll, D. C. (2006) A 'bright zone' in male hoverfly (*Eristalis tenax*) eyes and associated faster motion detection and increased contrast sensitivity. *Journal of Experimental Biology* 2006 **209**(21): 4339-4354. [link](#)
- Paul D. Barnett, Karin Nordström, & David C. O'Carroll (2006) Retinotopic Organization of Small-Field-Target-Detecting Neurons in the Insect Visual System. *Current Biology* 2007 **17**(7): 569-578. [link](#)
- Blanche TJ, Spacek MA, Hetke JF, Swindale NV. (2005) Polytrodes: high-density silicon electrode arrays for large-scale multiunit recording. *J Neurophysiol.* 93(5):2987-3000.
- Shoemaker, P.A., Straw, A.D., & O'Carroll, D.C. (2005) Velocity constancy and models for wide-field visual motion detection in insects. *Biological Cybernetics* **93**(4): 275-287.
- SN Fry, P Müller, H-J Baumann, AD Straw, M Bichsel, & D Robert (2004) Context-dependent stimulus presentation to freely moving animals in 3D. *Journal of Neuroscience Methods* **135**(1-2): 149-157. [link](#)

4.2 Documentation

4.2.1 Article in Frontiers in Neuroinformatics

A good first stop is the article about the Vision Egg:

- Straw, Andrew D. (2008) Vision Egg: An Open-Source Library for Realtime Visual Stimulus Generation. *Frontiers in Neuroinformatics*. doi: 10.3389/neuro.11.004.2008 [link](#)

4.2.2 Vision Egg Programmer's Manual

The [Vision Egg Programmer's Manual](#) provides a high-level overview of the Vision Egg and is the best source of information when getting started.

You can also download the [PDF Version](#).

4.2.3 Vision Egg Library Reference

The [Vision Egg Library Reference](#) is generated automatically from the source code.

4.2.4 Frequently Asked Questions

See [Documentation/FAQ](#).

4.2.5 Tutorial

See [Documentation/Tutorial](#).

Also, the demo programs demonstrate most of the Vision Egg features in a more immediate way.

4.2.6 Mailing List

For discussion of the Vision Egg, visit (and subscribe to) the mailing list at <http://www.freelists.org/list/visionegg>

Once you've subscribed, you can email the list by sending email to visionegg@freelists.org .

4.2.7 Help on the Wiki Software

See [HelpContents](#)

4.3 Documentation/FAQ

'[\[\[Navigation\(siblings\)\]\]](#)'_

Contents

- Documentation/FAQ
 - I enter my desired framerate in the “What will your monitor refresh’s rate be (Hz)” field of the Vision Egg startup GUI, but I get a different refresh rate. Why?
 - I am unable to open a graphics window. I get the following exception: “pygame.error: Couldn’t find matching GLX visual”
 - How to start without the configuration GUI appearing?
 - I can not get the demos to work. The GUI Graphics configuration launches correctly when I try grating.py but when I click “ok” a new gray window opens for a few seconds and then closes. Can anyone help???
 - How do I turn off the configuration window?
 - I have set up my PC with a dual boot and have been gradually getting used to Linux. I would like to make the transition solely to Linux and the open source movement. However, I am running into one dependency problem after another and have tried RPM and tar.gz files. I even have a few experienced Linux users around and we still can not get all the dependencies installed.
 - How do I specify where on my screen the display should go? If I create a window, I know how to direct the graphics to a certain location within that window, but how do I set the position of the window itself?
 - When using certain stimuli, including the demo versions of the moving grating and put_pixels, I sometimes see an artifact that travels from the bottom of the window to the top. It usually looks like a slightly jagged line, and usually travels across the entire screen slowly to the top. I’m not sure if this is a programming issue or something that is specific to my monitor. It’s an LCD monitor running at 60Hz, and that’s what vision egg is set to use.
 - When I run a stand-alone VisionEgg windows executable created with py2exe, I get the error “IOError: [Errno 2] No such file or directory: ‘...library.zip\OpenGL\version’
 - What are my anti-aliased points appearing as squares (instead of circles)?

4.3.1 I enter my desired framerate in the “What will your monitor refresh’s rate be (Hz)” field of the Vision Egg startup GUI, but I get a different refresh rate. Why?

The Vision Egg does not set your refresh rate, only the resolution. The refresh rate is determined automatically by your video drivers for the specified resolution. So, you must tell your video drivers what the “proper” refresh rate for that resolution is. In Windows, try switching to that resolution, change your refresh rate for that resolution, and go back to your original resolution, start the Vision Egg and see what happens. Also, [PowerStrip](#) from Entech may help significantly.

Unfortunately, setting the refresh rate would be OS and video driver dependent and is not something the Vision Egg does. I think [PowerStrip](#) does a fantastic job on Windows, Mac OS X only seems to have a handful of video modes, and on linux and IRIX you can generate your own arbitrary modes, albeit not as easily as with [PowerStrip](#).

4.3.2 I am unable to open a graphics window. I get the following exception: “pygame.error: Couldn’t find matching GLX visual”

An attempt was made to open a graphics window, but the requested mode was not found. Try setting the total bit depth to 0 and each RGBA component to 0 to allow automatic matching. Start with a small window (e.g. 640x480) and do not try fullscreen initially.

4.3.3 How to start without the configuration GUI appearing?

You have 2 options to start the VE without the configuration GUI. The first way is to replace the line `screen = get_default_screen()` with the line `screen = VisionEgg.Core.Screen(size = (600,600))`. You may wish to specify more arguments as listed in the [Reference manual](#). The second is to set the environment variable `VISIONEGG_GUI_INIT` to 0.

4.3.4 I can not get the demos to work. The GUI Graphics configuration launches correctly when I try `grating.py` but when I click “ok” a new gray window opens for a few seconds and then closes. Can anyone help???

Your version of Numeric may not be matched to the version with which PyOpenGL was built. Install the Numeric version with which PyOpenGL was built, or rebuild PyOpenGL with your current version of Numeric.

4.3.5 How do I turn off the configuration window?

Pick your method:

- Edit your config file. The location of this file is in the window itself.
- Set the environment variable `VISIONEGG_GUI_INIT=0`.
- Add the following to your script before you create the screen:

```
import VisionEgg; VisionEgg.config.VISIONEGG_GUI_INIT = 0 (or from
VisionEgg import *; config.VISIONEGG_GUI_INIT = 0)
```

Note that all variables in `VisionEgg.config` can be modified this way. Please see the documentation for the Configuration module for more information.

4.3.6 I have set up my PC with a dual boot and have been gradually getting used to Linux. I would like to make the transition solely to Linux and the open source movement. However, I am running into one dependency problem after another and have tried RPM and tar.gz files. I even have a few experienced Linux users around and we still can not get all the dependencies installed.

Even though I like the package manager of my preferred linux distribution (Debian), I often install Python packages myself, particularly when I run into trouble like this. Also, experience on a number of platforms has convinced me that it allows me to be most platform-independent to use Python’s `distutils` (simple command-line interface) for installing Python packages. Python’s `distutils` system is the same across all Python platforms and versions back to 1.5.7, and there are countless Python packages you’ll find in source form but won’t be packaged for individual operating systems.

I do use the binary package manager of my linux system for the python executable and sometimes other packages if they’re available. If they’re not immediately available (or functioning) I use Python’s `distutils`.

If you have issues with a particular dependency, try sending a message to the appropriate mailing list.

4.3.7 How do I specify where on my screen the display should go? If I create a window, I know how to direct the graphics to a certain location within that window, but how do I set the position of the window itself?

The screen position is determined automatically by the OS, I think. SDL (which is wrapped by `pygame`, which is wrapped by the Vision Egg) has some environment variables which are supposed to allow specification of screen

position. see: <http://www.freelists.org/archives/visionegg/08-2003/msg00000.html>

Unfortunately, I think these environment variables aren't "officially supported" in SDL, and thus are subject to future non inclusion. (In fact, a recent build of pygame/SDL on Windows XP seems to ignore them, which I was planning on following up... Wait, it looks like it's discussed here: <http://www.devolution.com/pipermail/sdl/2005-April/068395.html>)

4.3.8 When using certain stimuli, including the demo versions of the moving grating and `put_pixels`, I sometimes see an artifact that travels from the bottom of the window to the top. It usually looks like a slightly jagged line, and usually travels across the entire screen slowly to the top. I'm not sure if this is a programming issue or something that is specific to my monitor. It's an LCD monitor running at 60Hz, and that's what vision egg is set to use.

This is the well-known 'tearing' artifact. Googling that should give you a better idea of what the problem is. You can either try to fix it by clicking the "sync buffer swaps to vertical retrace" button on the Vision Egg initial GUI popup window, or by forcing this (often called "VSync") in your video drivers. If you need further help, let us know, and be sure to specify your OS and video card.

4.3.9 When I run a stand-alone VisionEgg windows executable created with `py2exe`, I get the error "IOError: [Errno 2] No such file or directory: '...library.zip\OpenGL\version'"

The problem is that distutils does not copy non-python resources, and `pyOpenGL` expects a "version" file to be installed next to `OpenGL__init__.py`. The most straight-forward solution is to patch `OpenGL__init__.py` with a `try/except` block around the code that opens the version file, as described here: [Mike Fletcher post to python-list](#)

Line 13:17 of `OpenGL.__init__.py`, replace with:

```
try:
    filename = os.path.join(os.path.dirname(__file__), 'version')
    __version__ = string.strip(open(filename).read())
except Exception, err:
    __version__ = '2.0.2.02'
```

After applying this patch rebuild the application with `py2exe`.

4.3.10 What are my anti-aliased points appearing as squares (instead of circles)?

Support for antialiased points is driver dependent. The details on this web page are extremely helpful:

<http://homepage.mac.com/arekkusu/bugs/invariance/HWAA.html>

An alternative is the `gluDisk()` utility function:

```
quad = gluNewQuadric()
gluQuadricDrawStyle(quad, GLU_FILL)
...
gluDisk(quad, 0, 0.1, 50, 1)
...
gluDeleteQuadric(quad)
```

This creates a disk with zero inner radius, 0.1 outer radius, 50 slices, one loop. Note that the radius dimension is in world dimensions, whereas the `glPointSize()` call uses pixel dimensions.

You can use this construct in conjunction with display lists to improve rendering efficiency:

```
def initDisk(self):
    """ Call this function from your constructor or other initialization code. """
    quad = gluNewQuadric()
    gluQuadricDrawStyle(quad, GLU_FILL)

    # Construct representative disk
    self.diskList = glGenLists(1)
    glNewList(self.diskList, GL_COMPILE)
    gluDisk(quad, 0, 0.1, 50, 1)
    glEndList()
    gluDeleteQuadric(quad)

def draw(self):
    ...
    # Move to the place where you want your disk/point
    glTranslate(x, y, 0)
    # Render disk
    glCallList(self.diskList)
    ...
```

4.4 Documentation/Tutorial

Contents

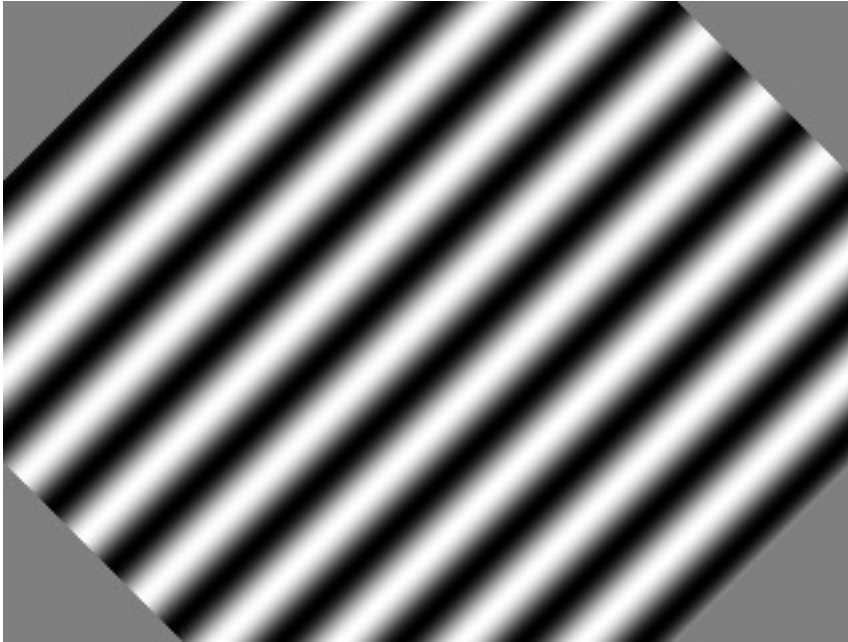
- Documentation/Tutorial
 - Overview
 - grating.py
 - target.py
 - targetBackground.py
 - put_pixels.py
 - Many more demos are included!

4.4.1 Overview

This page describes a few of the many of the demo scripts included with the Vision Egg. Click on the images to see a 320x240 12 frame per second [QuickTime](#) movie of the output of the Vision Egg. Note that this is very small and slow compared to what you would see running the same demo on your computer, especially in fullscreen mode. These demos and many more are a [Download and Install](#) away.

4.4.2 grating.py

The grating demo is a simple Vision Egg application script to show basic operation. First, the screen is initialized, which opens the OpenGL window. Second, a grating object is created with specified parameters of size, position, spatial frequency, temporal frequency, and orientation. Next, a viewport object is created, which is an intermediary between stimuli and screens. Finally, a presentation object is created, which controls the main loop and realtime behavior of any Vision Egg script.



```
#!/usr/bin/env python
"""Sinusoidal grating calculated in realtime."""
#####
# Import various modules #
#####
import VisionEgg
VisionEgg.start_default_logging(); VisionEgg.watch_exceptions()
from VisionEgg.Core import *
from VisionEgg.FlowControl import Presentation
from VisionEgg.Gratings import *
#####
# Initialize OpenGL window/screen #
#####
screen = get_default_screen()
#####
# Create sinusoidal grating object #
#####
stimulus = SinGrating2D(position      = ( screen.size[0]/2.0, screen.size[1]/2.0 ),
                        anchor        = 'center',
                        size          = ( 300.0 , 300.0 ),
                        spatial_freq   = 10.0 / screen.size[0], # units of cycles/pixel
                        temporal_freq_hz = 1.0,
                        orientation    = 45.0 )
#####
# Create viewport - intermediary between stimuli and screen #
#####
viewport = Viewport( screen=screen, stimuli=[stimulus] )
#####
# Create presentation object and go! #
#####
p = Presentation(go_duration=(5.0,'seconds'),viewports=[viewport])
p.go()
```

4.4.3 target.py

The target demo creates a small target which moves across the screen. The target is spatially anti-aliased by default, meaning that the edges can have colors intermediate between the target and the background to reduce *jaggies*. This demo also introduces the concept of *controllers* which allow realtime control of a Vision Egg stimulus through any number of means, including a data acquisition device, a network connection, or software control.



```
#!/usr/bin/env python
"""A moving target."""
#####
# Import various modules #
#####
import VisionEgg
VisionEgg.start_default_logging(); VisionEgg.watch_exceptions()
from VisionEgg.Core import *
from VisionEgg.FlowControl import Presentation, Controller, FunctionController
from VisionEgg.MoreStimuli import *
from math import *
#####
# Initialize the various bits #
#####
# Initialize OpenGL graphics screen.
screen = get_default_screen()
# Set the background color to white (RGBA).
screen.parameters.bgcolor = (1.0,1.0,1.0,1.0)
# Create an instance of the Target2D class with appropriate parameters.
target = Target2D(size = (25.0,10.0),
                  color = (0.0,0.0,0.0,1.0), # Set the target color (RGBA) black
                  orientation = -45.0)
# Create a Viewport instance
viewport = Viewport(screen=screen, stimuli=[target])
# Create an instance of the Presentation class. This contains the
# the Vision Egg's runtime control abilities.
p = Presentation(go_duration=(10.0,'seconds'),viewports=[viewport])
#####
# Define controller #
```

```
#####
# calculate a few variables we need
mid_x = screen.size[0]/2.0
mid_y = screen.size[1]/2.0
max_vel = min(screen.size[0],screen.size[1]) * 0.4
# define position as a function of time
def get_target_position(t):
    global mid_x, mid_y, max_vel
    return ( max_vel*sin(0.1*2.0*pi*t) + mid_x , # x
            max_vel*sin(0.1*2.0*pi*t) + mid_y ) # y
# Create an instance of the Controller class
target_position_controller = FunctionController(during_go_func=get_target_position)
#####
# Connect the controllers with the variables they control #
#####
p.add_controller(target,'position', target_position_controller )
#####
# Run the stimulus! #
#####
p.go()
```

4.4.4 targetBackground.py

The targetBackground demo illustrates how easy it is to combine multiple stimuli. A spatially anti-aliased small target is drawn as before, but this occurs over a spinning drum.

This demo also introduces more power of OpenGL – coordinate transforms that occur in realtime via projections. In the Vision Egg, a projection is a parameter of the viewport. In the default case (such as for the small target), the viewport uses pixel coordinates to create an orthographic projection. This allows specification of stimulus position and size in units of pixels. However, a projection also allows other 3D to 2D projections, such as that used to draw the spinning drum. This drum, which is defined in 3D, is drawn using a perspective projection. Because the drum uses a different projection than the small target, it needs its another viewport to link it to the screen.



```

#!/usr/bin/env python
"""Moving target over a spinning drum."""
#####
# Import various modules #
#####
from VisionEgg import *
start_default_logging(); watch_exceptions()
from VisionEgg.Core import *
from VisionEgg.FlowControl import Presentation, Controller, FunctionController
from VisionEgg.MoreStimuli import *
from VisionEgg.Textures import *
import os
from math import *
# Initialize OpenGL graphics screen.
screen = get_default_screen()
#####
# Create the target #
#####
# Create an instance of the Target2D class with appropriate parameters
target = Target2D(size = (25.0,10.0),
                  color = (1.0,1.0,1.0,1.0), # Set the target color (RGBA) black
                  orientation = -45.0)
# Create a viewport for the target
target_viewport = Viewport(screen=screen, stimuli=[target])
#####
# Create the drum #
#####
# Get a texture
filename = os.path.join(config.VISIONEGG_SYSTEM_DIR, "data", "panorama.jpg")
texture = Texture(filename)
# Create an instance of SpinningDrum class
drum = SpinningDrum(texture=texture, shrink_texture_ok=1)
# Create a perspective projection for the spinning drum
perspective = SimplePerspectiveProjection(fov_x=90.0)
# Create a viewport with this projection
drum_viewport = Viewport(screen=screen,
                        projection=perspective,
                        stimuli=[drum])
#####
# Create an instance of the Presentation class #
#####
# Add target_viewport last so its stimulus is drawn last. This way the
# target is always drawn after (on top of) the drum and is therefore
# visible.
p = Presentation(go_duration=(10.0, 'seconds'), viewports=[drum_viewport, target_viewport])
#####
# Define controllers #
#####
# calculate a few variables we need
mid_x = screen.size[0]/2.0
mid_y = screen.size[1]/2.0
max_vel = min(screen.size[0], screen.size[1]) * 0.4
# define target position as a function of time
def get_target_position(t):
    global mid_x, mid_y, max_vel
    return ( max_vel*sin(0.1*2.0*pi*t) + mid_x , # x
            max_vel*sin(0.1*2.0*pi*t) + mid_y ) # y
def get_drum_angle(t):

```

```

    return 50.0*math.cos(0.2*2*math.pi*t)
# Create instances of the Controller class
target_position_controller = FunctionController(during_go_func=get_target_position)
drum_angle_controller = FunctionController(during_go_func=get_drum_angle)
#####
# Connect the controllers with the variables they control #
#####
p.add_controller(target,'position', target_position_controller )
p.add_controller(drum,'angular_position', drum_angle_controller )
#####
# Run the stimulus! #
#####
p.go()

```

4.4.5 put_pixels.py

The put_pixels demo puts arbitrary array data to the screen. For the sake of simplicity this example uses only solid, uniformly colored arrays. The screen is updated with a new array on every frame, which will reveal tearing artifacts if you do not have buffer swaps synchronized to VSync.

This demo also illustrates an alternative to using the FlowControl module by using pygame's event handling.

```

#!/usr/bin/env python
import VisionEgg
VisionEgg.start_default_logging(); VisionEgg.watch_exceptions()
from VisionEgg.Core import *
import pygame
from pygame.locals import *
screen = get_default_screen()
screen.set( bgcolor = (0.0,0.0,0.0) ) # black (RGB)
white_data = (Numeric.ones((100,200,3))*255).astype(Numeric.UnsignedInt8)
red_data = white_data.copy()
red_data[:, :, 1:] = 0 # zero non-red channels
blue_data = white_data.copy()
blue_data[:, :, :-1] = 0 # zero non-blue channels
frame_timer = FrameTimer() # start frame counter/timer
count = 0
quit_now = 0
# This style of main loop is an alternative to using the
# VisionEgg.FlowControl module.
while not quit_now:
    for event in pygame.event.get():
        if event.type in (QUIT,KEYDOWN,MOUSEBUTTONDOWN):
            quit_now = 1
    screen.clear()
    count = (count+1) % 3
    if count == 0:
        pixels = white_data
    elif count == 1:
        pixels = red_data
    elif count == 2:
        pixels = blue_data
    screen.put_pixels(pixels=pixels,
                     position=(screen.size[0]/2.0,screen.size[1]/2.0),
                     anchor="center")
    swap_buffers() # display what we've drawn
    frame_timer.tick() # register frame draw with timer

```



```
frame_timer.log_histogram()
```

4.4.6 Many more demos are included!

4.5 Download and Install

4.5.1 Stable release - 1.2.1

Full source code and binary installers for Windows and Mac OSX

All files are at the [SourceForge downloads page](#) and [PyPI page](#).

Packages require Python and other dependencies. See the appropriate installation page for more details.

For platform-specific notes, see:

- [Download and Install/Install on Windows](#)
- [Download and Install/Install on MacOSX](#)

Source code of the demos

If you've installed the Windows .exe installer or the Mac OS X .dmg installer, you don't have the demo programs. These are available with the full source code in the "demo" directory.

4.6 Download and Install/An Installation Overview

'[[Navigation(siblings)]]'_

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Here is the big picture of what must be installed to get the Vision Egg up and running. This information may be enough to get the python-savvy person started immediately. Otherwise, read the detailed installation instructions for your platform.

Dependency list:

1. OpenGL
2. Python 2.3 or greater
3. The following Python packages:
 - a) numpy
 - b) Python Imaging Library
 - c) PyOpenGL
 - d) pygame
 - e) Pyro (optional)

Install the Vision Egg from the source code package by changing to the base visionegg directory (e.g. `'cd visionegg-1.1'`) and then executing `'python setup.py install'`

For more specific instructions, including information about platform-specific binary installers, click on your platform: [Windows_](#),

```
`Mac OS X`_, linux_, IRIX_
.. _Windows: Download_and_Install/Install_on_Windows
.. _`Mac OS X`: Download_and_Install/Install_on_MacOSX
.. _linux: Download_and_Install/Install_on_Linux
.. _IRIX: Download_and_Install/Install_on_IRIX
```

4.7 Download and Install/Install on IRIX

[\[\[Navigation\(siblings\)\]\]](#)_

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Installing the Vision Egg for SGI IRIX is similar to installing for linux. This document will covers mainly the differences from the linux installation process. I have installed the Vision Egg on IRIX64 6.5 using SGI's cc compiler.

Many of the tools used in the development of the Vision Egg are included on SGI's freeware distribution, so you might want to have that handy. (Despite SGI's choice, "freeware" is not the best term for open source software. "Open source software" has a couple more syllables, but is probably preferred by the authors of open source software, including the Vision Egg!) This method makes extensive use of SGI's Software Manager to install as much of the basic software as possible. Undoubtedly, because all of this software is open source, you could compile it yourself, as well.

Installing Python

Python compiles without trouble from the source, but I've had troubles getting the Tkinter module compiled. Therefore, I've been using precompiled Python from SGI's "freeware" distribution.

Installing Python Imaging Library (PIL)

I've had the following the error when building PIL::

```
The source file "tk.h" is unavailable.
1 catastrophic error detected in the compilation of "./_imagingtk.c".
```

I've just edited the file "Setup" and commented out the lines involving _imagingtk.

After building, my installation was::

```
cp PIL.pth /usr/freeware/lib/python2.1/site-packages/
mkdir /usr/freeware/lib/python2.1/site-packages/PIL
cp *.so PIL/* /usr/freeware/lib/python2.1/site-packages/PIL
```

Installing PyOpenGL

I built from the PyOpenGL2 cvs repository checked out on 5 June 2002.

I changed the following lines in config/irix.cfg::

```
[General]
build_togl=0
include_dirs=/usr/include:/usr/include/X11:/usr/freeware/include

[GLUT]
libs=Xi:Xmu
```

I had to regenerate (using swig1.3a5, which built just fine from source) the C interface files using `''python setup.py build_w''`. From there, it was `''python setup.py build''` and then, as root `''python setup.py install''`. After I wrote this document, I noticed that GLUT was not working, but I have not had a chance to go back and fix it. I believe the solution is simply to point PyOpenGL to libGLUT in the irix.cfg file and re-build.

Installing SDL

I installed from the `''freeware''` distribution. Make sure you install the subpackage `''SDL-1.2.3 full shared libraries''` from the package `''SDL-1.2.3 Simple DirectMedia Library''`.

Installing pygame

Once I had SDL installed as above, installing pygame 1.5 was easy. The only quirk is to point pygame at SGI's default installation directory. Set your environment variable LOCALBASE to `''/usr/freeware''`.

Installing the Vision Egg

This is a snap. Get the Vision Egg, decompress, move to that directory, and run `''python setup.py install''`. You probably need to have root access.

Check your installation with the `''check-config.py''` program. Also run this script if you run into any installation errors.

4.8 Download and Install/Install on Linux

[\[\[Navigation\(siblings\)\]\]](#) _

4.8.1 Quickstart for Ubuntu Hardy Heron (8.04)

You can use my repository which includes a .deb package of the Vision Egg, integrated with the Ubuntu repositories for dependencies. See the ["Ubuntu packages" instructions on the motmot wiki](#). (This wiki is generally about other software I wrote, but you can skip steps 8-10 and install python-visionegg instead.) Note that this package only installs the Vision Egg library. To get the demos or documentation, download the source package. – Andrew Straw

4.8.2 More general instructions

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Here are the all the steps in detail required to get the Vision Egg to run on your linux system.

****This page details the build-from-source approach. Your linux distribution probably has these packages available to your package manager.****

=====
OpenGL
=====

OpenGL is probably already installed on your system. If you're running linux, the tricky part may be getting hardware acceleration. This is accomplished by getting your X Windows server into hardware accelerated mode. This is beyond the scope of this document, but there are many good documents on the internet.

=====
Installing Python
=====

Python installation is straightforward, with ample documentation. Your particular distribution of linux probably comes with Python installed, but in case it doesn't, or if the Vision Egg doesn't work with your distribution's version of Python, these instructions detail building Python from the C source. These instructions were written for Python 2.1.1, but Python 2.2.1 is nearly identical.

The Python website is <http://www.python.org>

To build from source, unpack using the usual gunzip and tar routine. In the expanded Python source directory run `''./configure''`. Then type `''make''`. You can test Python by running `''make test''`. If all goes well, type `''make install''`.

By default, building from the source does not compile Tkinter support. Tkinter is very useful for creating GUIs in Python, and the Vision Egg, especially a few demo programs, uses it. The core functionality of the Vision Egg will work without Tkinter, but it's best to get it working at this stage. Edit the Modules/Setup file in the Python source directory, look for `''_tkinter''`, and uncomment the various lines needed to get it to compile. Run `''make''` again in the root of the source directory to get `_tkinter` to compile and then `''make install''` (again) to install it.

=====
Installing Python packages
=====

Installing Numeric Python

The Numeric Python website is <http://numpy.sourceforge.net>

For Numeric-21.0, run `''python setup.py install''`

For Numeric-20.0, make sure to run `''setup_all.py''`. In addition to the basic Numeric modules, this command makes the FFT and other libraries, which are very useful, but not needed by the Vision Egg. (These extra modules are made by default in release 21.)

Installing Python Imaging Library (PIL)

The PIL download area is <http://www.secretlabs.com/downloads/index.htm#pil>

Imaging-1.1.2 was used for development.

Unfortunately, PIL can be tricky to install. The detailed instructions in the README are pretty good, but this package can be a bit tricky to install.

The shell commands I use to build the Imaging packages are:

```
::
cd Imaging-1.1.2/libImaging/
./configure
make
cd ..
make -f Makefile.pre.in boot
make
```

If you have errors with the `''make''` step that say something like `''tk8.0''` not found, open `''Makefile''` and change `''tk8.0''` to `''tk8.3''` and `''tcl8.0''` to `''tcl8.3''`. Of course this assumes you have version 8.3 of tk and tcl. If you don't have tcl, open the `''Setup''` file and comment out the `''_imagingtk''` lines.

If you have errors with the `''make''` step that say something like `''can't locate file: -ljpeg''`, download and install those libraries or comment out the appropriate lines in `''Setup''`. I've had trouble trying to build with those lines removed from the `''Setup''` file, so I just downloaded and installed the libraries. These libraries are very easy to compile and install. Just run `''./configure''` and `''make install''`. Under Mac OS X, I couldn't get a static or shared library to compile from the sources, so I used the version that fink installed for me.

If you have to edit `''Setup''`, you'll have to run `''make -f Makefile.pre.in boot''` and `''make''` again.

Now, Imaging is compiled, and you must copy the files to Python's local package directory. (How to find out what it is? It's usually `''/usr/lib/python2.1/site-packages''` or `''/usr/local/lib/python2.1/site-packages''`.)

```
::
cp PIL.pth /usr/lib/python2.1/site-packages
mkdir /usr/lib/python2.1/site-packages/PIL
cp *.so PIL/* /usr/lib/python2.1/site-packages/PIL
```

Installing PyOpenGL

PyOpenGL installation is well documented and straightforward in my experience in linux. (Not necessarily so with other platforms!) I've had trouble getting the GL/ARB/texture_compression.i file to compile with the OpenGL headers that came with my nVidia card. I have a patch that fixes this problem, if you're interested.

Installing pygame

The Vision Egg uses pygame as a Python binding to SDL. SDL is used to initialize an OpenGL window in a cross platform way. I have always had good fortune with distribution installed SDL, although building from source has always worked as well.

Once SDL is installed, installation of pygame is straightforward using the Python distutils. Just type `''python setup.py install''` from the pygame source directory.

Install the Vision Egg

Install vision egg by changing to the base visionegg directory and execute `''python setup.py install''`. You will need appropriate privileges on your system to install.

Check your installation with the `''check-config.py''` program. Also run this script if you run into any installation errors.

4.9 Download and Install/Install on MacOSX

4.9.1 Snow Leopard, OS X 10.6, VisionEgg1.2.1

I (Alex Holcombe) installed Vision Egg on an Intel iMac. I needed to:

Install Pygame(1.9.1), PyOpenGL(3.0.0), and PIL (Imaging-1.1.7), and **VisionEgg_** (1.2.1) from source by in terminal typing

```
python2.6 setup.py install
```

in the directory of each. The python2.6 is because the default installation of python, python2.5, had a problem (can't remember what it was now). Only in the case of PIL, I had to preface the above command with 'sudo', otherwise it didn't have permission to copy some scripts to the appropriate place.

From an email by Jens Kremkow:

Dear Andrew, David,

I managed to get it running on Snow Leopard. There were some issues with pygame. This was the way I got VisionEgg to run.

1. I use MacPorts to install python2.6 and the dependencies.
<http://www.macports.org>

2. Problems started when I tried to install py-game with MacPorts, it wanted to install a full python

3. I downloaded pygame from the svn and installed it from source
<http://www.pygame.org/wiki/cvs>

4. In pygame/Setup.in I changed the SDL path to the one of MacPorts (/opt/local/include/SDL)
 # SDL
 SDL = -I/opt/local/include/SDL -D_REENTRANT -lSDL

and I had to comment this line, because the camera resulted in a building error
 # _camera src/_camera.c src/camera_v4l2.c src/camera_v4l.c \$(SDL) \$(DEBUG

5. In pygame/config_darwin.py I also had to add the MacPort path
 line 32: BASE_DIRS = '/', os.path.expanduser('~'), '/opt/local/'
 line 113: incdirs = ['/opt/local/include', '/opt/local/include/freetype2/freetype', '/opt/local/includ
 line 114: libdirs = ['/opt/local/lib']

Maybe some of these path settings are redundant, but this way it compiled.

I think that was all the trouble I had. I hope it helps.

Best,
 Jens

And another email from John Christie:

Maybe I did something special but I got python 2.6.3 and pygame 1.9.1 dmg installers, the PIL install

4.9.2 Leopard, OSX 10.5.x, VisionEgg 1.2.x

The easy way (These instructions are finished)

This is a short version of more verbose instructions here: <http://www.scidav.org/installs>

1. Get the appropriate version of XCode from <http://developer.apple.com/mac>
2. Get the Enthought Python Distribution from <http://www.enthought.com/products/epd.php>
 - Includes almost all dependencies you'll want (good)
 - Not free to non-academics (but you're probably an academic)
3. Make a link to trick installers into thinking you have python.org 2.5 installed:
 - `cd /Library/Frameworks/Python.framework/Versions; sudo ln -s 5.0.0 2.5`
 - replace "5.0.0" with whatever your EPD version is, and if your EPD contains a newer python (e.g. 2.6), use that as the second argument.
4. Run the installer (for your version of python) from <http://pygame.org/download.shtml>
5. Install the **VisionEgg** source from <http://sourceforge.net/projects/visionegg/files/visionegg/>
 - `tar xzvf visionegg-1.2.1.tar.gz`
 - `cd visionegg-1.2.1`
 - `python setup.py install`

Note: the **VisionEgg** seems to work fine with pygame 1.9. Since this is only used to obtain the OpenGL context, I am not too concerned with testing this. Pygame 1.9 doesn't rely on pyobjc. If you insist on pygame 1.8 or earlier, you can easy_install pyobjc.

The hard way (These instructions are not yet finished)

- Get Python from python.org
- Install libjpeg according to <http://simonwilder.wordpress.com/2009/06/17/fixing-pil-ioerror-decoder-jpeg-not-available/>
- Install PIL from source from <http://effbot.org/downloads/Imaging-1.1.6.tar.gz>
- Install pygame from <http://www.pygame.org/ftp/pygame-1.8.1release-py2.5-macosx10.5.zip>
- Install PyOpenGL from http://downloads.sourceforge.net/sourceforge/pyopengl/PyOpenGL-3.0.0.tar.gz?use_mirror=superb-west

Installing from source with v1.1

- Download the source package from the sourceforge site
- Install all the necessary packages (numpy, PIL, PyOpenGL, pygame, pyobjc) from [here](#). The preferred version of python currently is 2.4, so ensure that when you type “python” at a Terminal, you get 2.4. If that command starts a different version, install **MacPython2_4** and change the “Current” aliases in the Frameworks directory (probably `/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-packages/`) to 2.4. Note that `/System/Library` has its own version of Python that you can ignore
- If you don’t have it, get Xcode from apple.com so that you can compile the source code
- Install by typing “python setup.py install” from the “visionegg-1.1” directory.

4.9.3 Instructions for Tiger, 10.4

I ([Alex Holcombe](#)) installed Vision Egg using the following procedure. Each of the following packages were downloaded from [here](#). These are Universal versions, and worked on my 2.33 GHz Intel **MacBook_ Pro**, as well as a G5 non-Intel PowerPC.

1. I installed [Xcode 2.4](#) (free Apple compiler)
2. “**Universal-MacPython_-2.4.3-2006-04-07**” was installed by double-clicking.
3. I removed all Python 2.3 references that were on my machine (this step should be unnecessary)
4. “[numpy-1.0.1-py2.4-macosx10.4-2006-12-12.dmg](#)” installed by double-clicking.
5. “[PIL-1.1.5-py2.4-macosx10.4](#)” installed by double-clicking.
6. “[PyOpenGL-2.0.2.01-py2.4-macosx10.4](#)” installed by double-clicking.
7. “[pygame-1.8.0pre-py2.4-macosx10.4a](#)” installed by double-clicking.
8. “[pyobjc-1.4-python2.4-macosx10.4.dmg](#)” installed by double-clicking.
9. The [Vision Egg source](#), `visionegg-1.1.tar.gz` file was downloaded and unpacked.
10. “python setup.py install” was executed from Terminal.app when in the “visionegg-1.1” directory

COMPLICATION. In my case, for some reason PIL was installed in step 4 with the wrong file permissions. This caused `check-config.py` to complain that it could not import Image. Once I changed all file permissions so all could read and execute (“`chmod +rx`”) for all the PIL files and directories (at `/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-packages/`), then everything worked. Hopefully this was caused by me switching to superuser at some point and forgetting to exit or something, let the mailing list or this wiki know if you run into this problem yourself.

[More details](#) about the new Universal Python2.4 packages used above.

See also the [Kubovy Lab setup page](#), who are no longer using Vision Egg per se but are using most of the same libraries.

4.9.4 Abbreviated Instructions for Tiger, Python 2.5

Ensure the XCode development tools are installed. Download and install SDL, SDL_ttf, SDL_mixer, and SDL_image BINARIES from www.libsdl.org (drag them into /Library/Frameworks). Download and install all dependencies from SOURCE: PIL, Numeric (NOT numarray), pygame, PyOpenGL 3, pyobjc. Download Vision Egg source. Install the source

4.10 Download and Install/Install on Windows

4.10.1 Instructions for Vision Egg 1.2.1

If in doubt about how to install from a .egg file, use the corresponding .exe file instead.

with Python 2.5

First, download and install these dependencies:

- [python-2.5.4.msi](#)
- [PIL-1.1.6.win32-py2.5.exe](#)
- [PyOpenGL-2.0.2.01.py2.5-numpy24.exe](#) (VE 1.2 has issues with PyOpenGL 3.0 on Windows. See [SF#2817196](#))
- [pygame-1.8.0.win32-py2.5.msi](#)
- [numpy-1.3.0-win32-superpack-python2.5.exe](#)
- [setuptools-0.6c8.win32-py2.5.exe](#)

Next, download and install a Vision Egg binary compiled for Python 2.5:

- [visionegg-1.2.1.win32-py2.5.exe](#), or [visionegg-1.2.1-py2.5-win32.egg](#)

Optional:

- [QuickTime](#)

with Python 2.6

PyOpenGL 3.0 (the only PyOpenGL available for Python 2.6) has a bug that prevents the VE from working properly on Windows. Use PyOpenGL 2 and Python 2.5 until it's fixed. [The bug report \(SF#2817196\)](#)

First, download and install these dependencies:

- [python-2.6.6.msi](#)
- [PIL-1.1.6.win32-py2.6.exe](#)
- [PyOpenGL-3.0.1.win32.exe](#)
- [pygame-1.8.1release.win32-py2.6.msi](#)
- [numpy-1.3.0-win32-superpack-python2.6.exe](#)

- [ssetuptools-0.6c9.win32.exe](#)

Next, download and install the Vision Egg binary compiled for Python 2.6. I will upload them to PyPI once the PyOpenGL bug is fixed:

- [‘attachment:visionegg-1.2.1.win32-py2.6.exe’_visionegg-1.2.1.win32-py2.6.exe‘attachment:None’_](#), or [‘attachment:visionegg-1.2.1-py2.6-win32.egg’_visionegg-1.2.1-py2.6-win32.egg‘attachment:None’_](#)

Optional:

- [QuickTime](#)

4.10.2 Alternate installers

[Christoph Gohlke](#) offers unofficial Windows binaries for various Python packages, including 64-bit versions of the Vision Egg.

4.10.3 Old Instructions for Vision Egg 1.1

Download and install:

- [python-2.5.4.msi](#)
- [PIL-1.1.6.win32-py2.5.exe](#)
- [PyOpenGL-2.0.2.01.py2.5-numpy24.exe](#) (VE 1.1.x has issues with PyOpenGL 3.)
- [pygame-1.8.0.win32-py2.5.msi](#)
- [numpy-1.3.0-win32-superpack-python2.5.exe](#)
- [setuptools-0.6c8.win32-py2.5.exe](#)

Optional:

- [QuickTime](#)

4.11 Intro and Overview/About this wiki

[‘\[\[Navigation\(siblings\)\]\]’_](#)

4.11.1 Basics

The website is running on a [MoinMoin](#) installation. All pages are publicly modifiable, so if you see something that’s wrong, fix it. If you see something you don’t like, adjust to your taste. Login and subscribe to changes on any page.

You can also create new pages. I have attempted to maintain the hierarchy of the old Vision Egg website by using the tabs along the top. If at all possible, please create a new page as a child of an already existing parent (for example, this page is a child of [FrontPage_](#)).

4.11.2 Links

Here are some of the links provided with the wiki software:

- [HelpContents_](#)

- [FindPage_](#)
- [RecentChanges_](#) to this website

4.11.3 Attachments

I have disabled file uploading as a security precaution. If you'd like to upload an image or other file to this website, either email it to me, drop it in my lab's ftp server (instructions [here](#)), or coordinate a time with me where I enable file uploading.

4.12 Intro and Overview/Calibration

[\[\[Navigation\(siblings\)\]\]'_](#)

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Controlling the luminance of the display precisely is important for many experiments. The Vision Egg system and overcome the inherent non-linearity present in most displays. The result is a system where

Here is the result of some measurements I performed on my own display using a calibrated photometric (Ltd.) and `*ephys_gui*`, an application that comes with the Vision Egg.

```
.. image:: luminance_calibration.png
   :width: 400
   :height: 300
   :alt: luminance calibration
```

Notes

This test does not test the calibration near low contrasts near the threshold for human vision. Video per color or more) gamma lookup tables are well suited for low contrast experiments. Even better are framebuffers, although OpenGL support for this is currently lacking in all consumer cards that I know (Matrox Parhelia).

I do not have the capabilities to perform color calibration -- this test was performed by locking the

4.13 Intro and Overview/FrameRates

[\[\[Navigation\(siblings\)\]\]'_](#)

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

There are two different values which both get called "frame rate". The first is the number of frames a graphics card draws to its framebuffer per second. The second is the number of frames per second drawn on a monitor from the graphics card. The frame rate of the monitor is a constant set by video drivers, and the frame rate of drawing to the framebuffer is determined by how long it takes for a particular program to complete all the elements in a scene. Obviously, it is a waste of resources to draw more frames to the framebuffer than will be displayed on the monitor, although in some cases there may be reasons for doing so. (For example, if program control and drawing are done in the same loop, the control latency

will be shorter at faster frame rates. Benchmarking 3D games or graphics cards is another reason why one might want to do this.) For the rest of this document, "frame rate" will refer to the refresh rate of the monitor.

The absolute upper limit of the monitor's frame rate is specified by the "maximum vertical refresh frequency". In typical professional displays this value may be 160 Hz. Several support 180 Hz, and a few (such as the LGE Flatron 915 FT Plus) go up to 200 Hz. Unfortunately, your video driver may only allow certain pre-defined frame rates. See below and the installation details page of your platform for platform-specific information about setting monitor refresh rates.

What frame rate do you want? If your stimulus involves high-speed motion, more frames per second is better, simply because it reduces temporal aliasing. At a minimum, your monitor's flicker rate (equal to frame rate on a CRT) should exceed the flicker fusion frequency of your subject.

If your stimulus involves slow motion and your system can only move on-screen objects in integer pixel increments (this is not the case with the Vision Egg), you again want a high frame rate to maximize the smoothness of motion if an object must alternately move an integer number of pixels and then remain in place.

With these arguments in favor of high frame rates, why might you want slower frame rates? It is easier on the (typically analog) electronic circuitry involved in the production and amplification of the video signal, and therefore square edges on waveforms will be closer to truly square, producing better spatial crispness. You may be able to draw more pixels per frame at a slower frame rate because of limitations in your monitor's "horizontal refresh frequency" and "pixel bandwidth" in addition to your video card's "RAMDAC/pixel clock".

Platform-specific tools to create custom video modes

XFree86, the video driver for linux, lets the user specify exactly the timing parameters of the video signal in the "modeline" section of the XF86Config file.

For Windows, RefreshForce_ (freeware) is very useful. PowerStrip_ (shareware) allows arbitrary setting of video mode. Furthermore, 'Refresh Rate Fixer'_ may be of use.

On Mac OS X, switchResX_ may provide a way to set the frame rate other than the default values.

SGI IRIX has its own video mode tools such as setmon which allow arbitrary video modes to be produced.

The 'XFree86 Video Timing HOWTO'_ is an excellent resource for understanding the intricacies of the timing of video signals, even if you don't use X windows.

Multi-tasking operating systems and dropped frames

The simplest way to guarantee that your monitor is updated on every frame is to put the control and drawing routines in a loop which cannot be interrupted. This can be done with special graphics cards which have an on-board processor separate from the computer's main CPU, or in operating systems which give programs complete control of the CPU. Unfortunately, OpenGL graphics cards must be controlled from the CPU, and most modern operating systems have a kernel which can suspend any program. There is the (currently untried) possibility of running the Vision Egg on a system with a realtime OS which would also solve the problem completely.

We believe the Vision Egg, or another OpenGL solution which has the ability to move objects in sub-pixel increments, provides the absolute best way to produce smooth motion.

```
.. _RefreshForce: http://www.pagehosting.co.uk/rf
.. _PowerStrip: http://www.entechtaiwan.com/ps.htm
.. _`Refresh Rate Fixer`: http://www.radeon2.ru/refreshfix_eng.html
.. _switchResX: http://www.madrau.com/html/SRX/indexSRX.html
.. _`XFree86 Video Timing HOWTO`: http://www.tldp.org/HOWTO/XFree86-Video-Timings-HOWTO
```

4.14 Intro and Overview/Platforms

[\[\[Navigation\(siblings\)\]\]](#) [_](#)

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

The Vision Egg is known to run on:

- * Windows (tested with '95, 2000 and XP)
- * linux (x86 needed for hardware accelerated OpenGL)
- * Mac OS X (tested with 10.2 and 10.3)
- * SGI IRIX
- * Probably just about anything else with OpenGL

Performance of your video card and its interaction with your system is very significant when running the Vision Egg. All of those computer gamers' reviews are relevant!

In my experience, under Windows 2000 I get no frame skipping at a 200 Hz frame rate.

Linux offers a low latency kernel and a scheduler with adjustable priority, but even so, the Vision Egg (under kernel 2.4.12 at least) occasionally skips the occasional frame. (Note: `since linux 2.5.4`_, the kernel has been pre-emptible, possibly eliminating frames skipped for non-Vision Egg reasons. To the best of my knowledge, no one has tested this yet. Here is `more information on the 2.6 kernel`_.)

On Mac OS X, switchResX_ may allow video modes other than Apple's defaults, making high framerates possible. Mac OS X also has a realtime scheduler, which can eliminate dropped frames.

The greatest appeal of SGI IRIX is that more than 8 bits per color are possible in the framebuffer and in the DACs. The Vision Egg has been tested on an SGI Fuel workstation with a V10 graphics card, and is

known to support these 10 bits per channel in the framebuffer and in the DACs. (Recent consumer video cards have 10 bit DACs. However, many of these cards have only an 8 bit framebuffer which then passes through a 10 bit gamma table. The Matrox Parhelia and ATI Radeon 9700 have full 10 bit framebuffers, but this mode does not work with those vendors' current OpenGL drivers. Please let me know if you find otherwise.)

****For these reasons, I recommend Windows 2000 as the premiere operating system on which to run the Vision Egg. A recent Athlon or Pentium 4 with a recent video card from nVidia or ATI is ample power to run the Vision Egg.****

That being said, the thing to remember is that the Vision Egg is cross platform, so you will be poised to take advantage of any particular features of any platform. And with the wars raging between the makers of graphics cards, the situation can only get better!

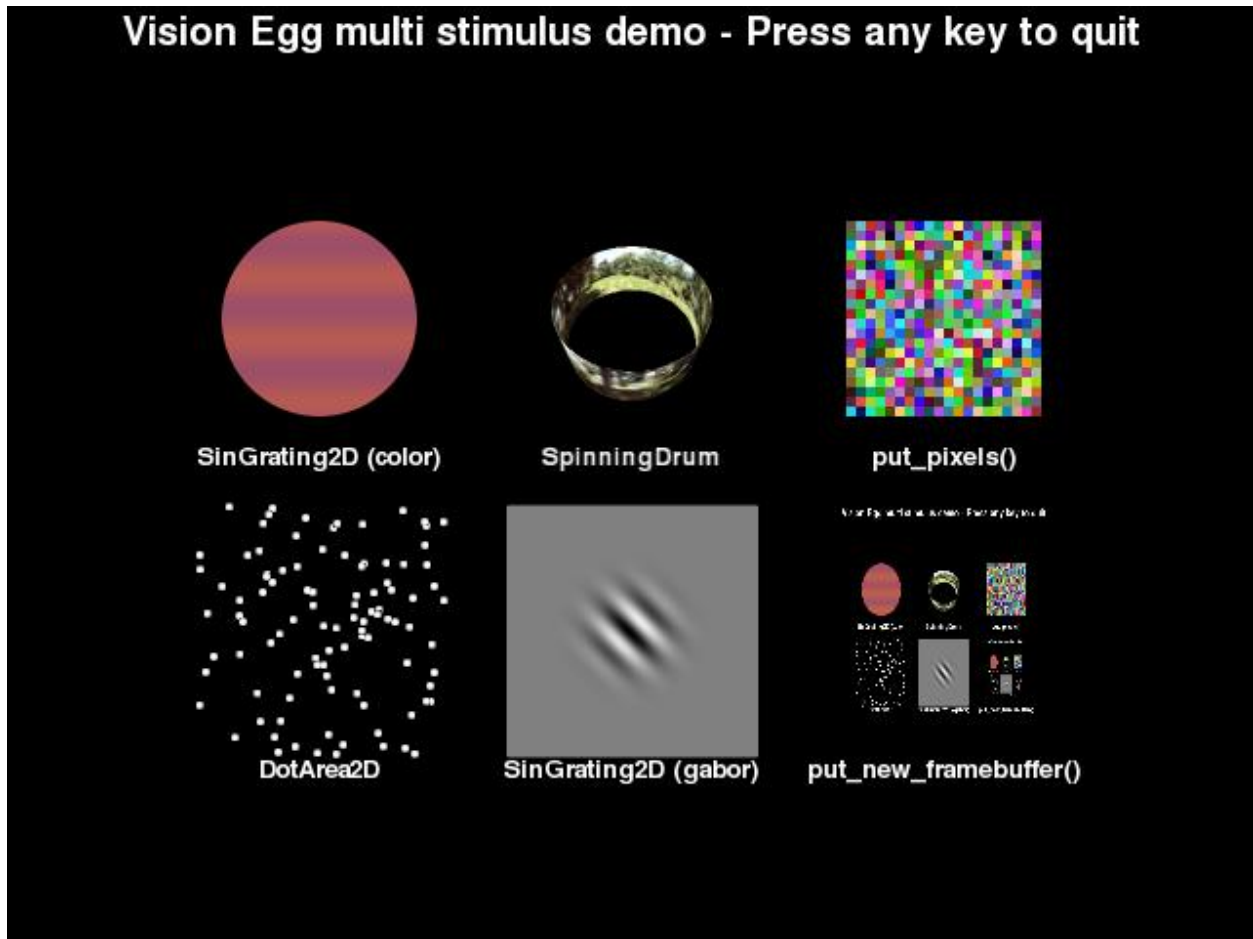
```
.. _`since linux 2.5.4`: http://www.linuxdevices.com/news/NS3989618385.html
.. _`more information on the 2.6 kernel`: http://www.linuxdevices.com/articles/AT7751365763.html
.. _switchResX: http://www.madrau.com/html/SRX/indexSRX.html
```

4.15 Screenshots

All of the screenshots below were produced with demos that come with the Vision Egg.

4.15.1 Multiple stimulus demo

multi_stim.py shows a color sinusoidal grating (viewed through a circular mask), a random dot stimulus, a perspective-distorted drum from the perspective of a moving camera, a gaussian-windowed sinusoidal grating (gabor wavelet), "blitting" of arbitrary pixels from a numeric array, and a (recursive) copy of the framebuffer.



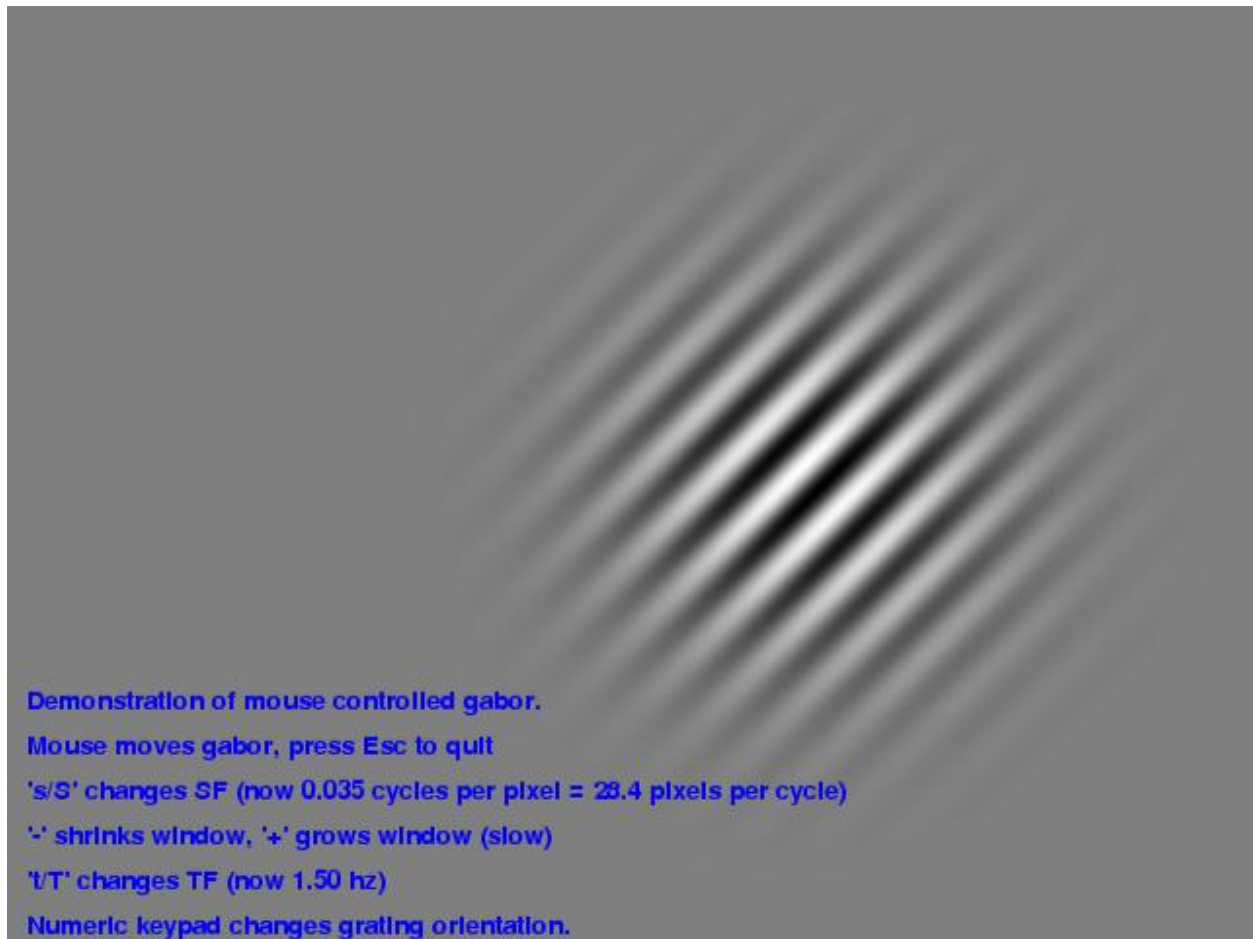
4.15.2 QuickTime demo

You can use the Vision Egg to play QuickTime (currently Mac OS X only) and MPEG (all platforms) movies. You can place them as normal textures, allowing you to reshape them, warp them with various distortions, or superimpose other graphics or text.



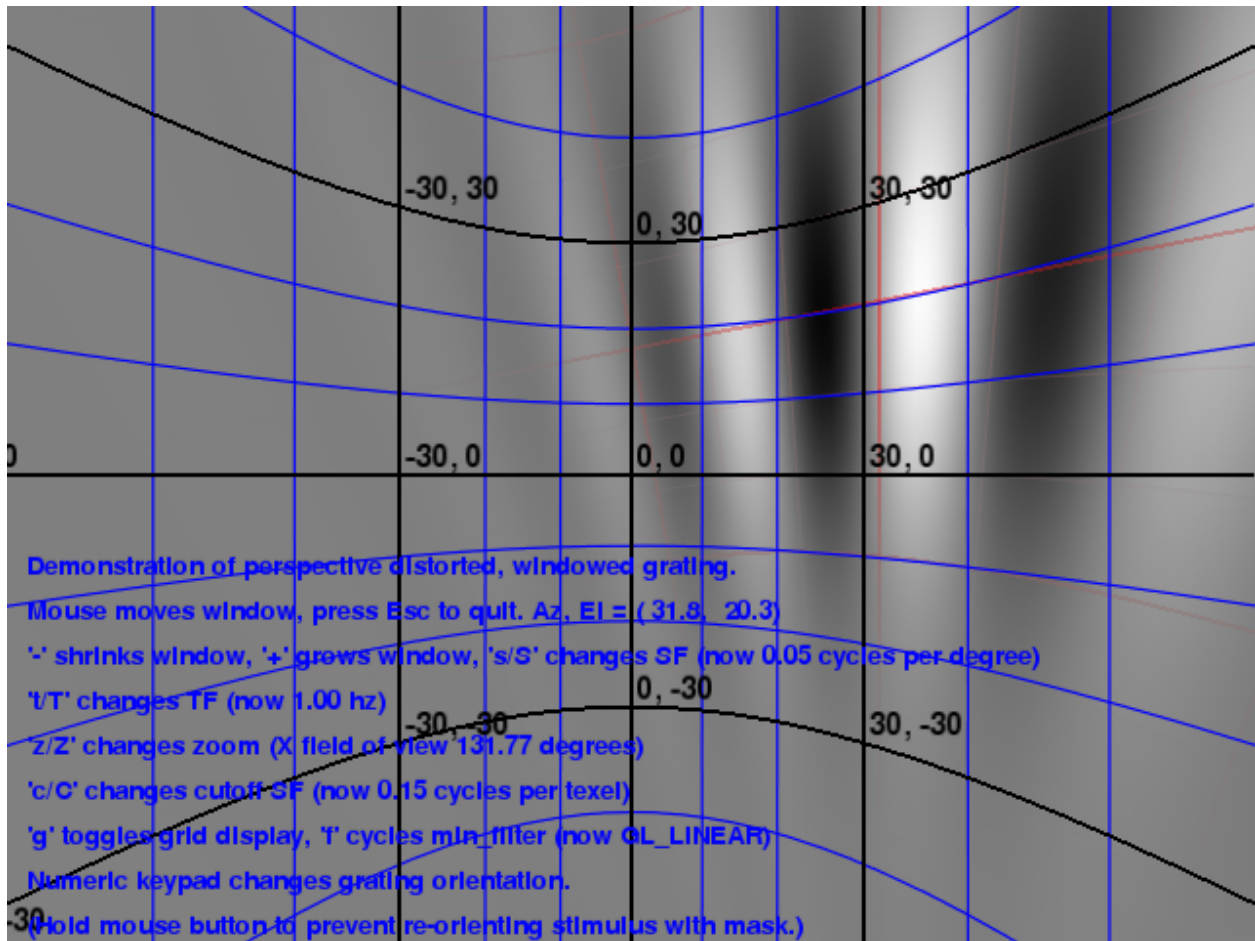
4.15.3 mouse_gabor_2d demo

mouse_gabor_2d.py illustrates that stimuli are generated in realtime. Parameters such as spatial frequency and orientation can be updated without skipping a frame.

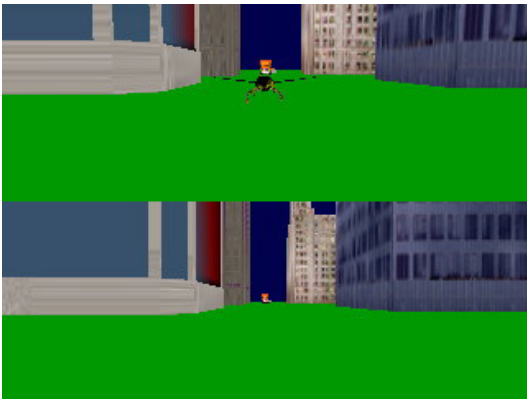
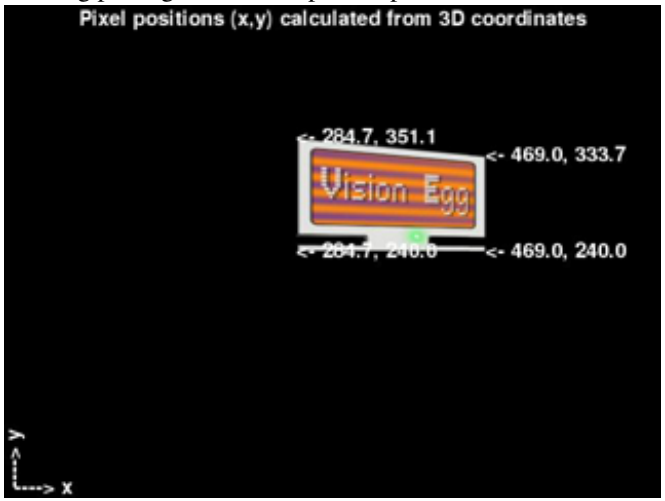


4.15.4 mouse_gabor_perspective demo

mouse_gabor_perspective.py shows perspective distortion along with realtime stimulus generation. Such wide fields of view are important for research on insects, for example, because many insects have a very large visual field of view. A grid of iso-azimuth and iso-elevation lines is superimposed on the grating in this screenshot. This grid represents visual coordinates of a fixed observer looking at the middle of the screen. A second, dimmer grid shows the reference coordinates system for the grating.



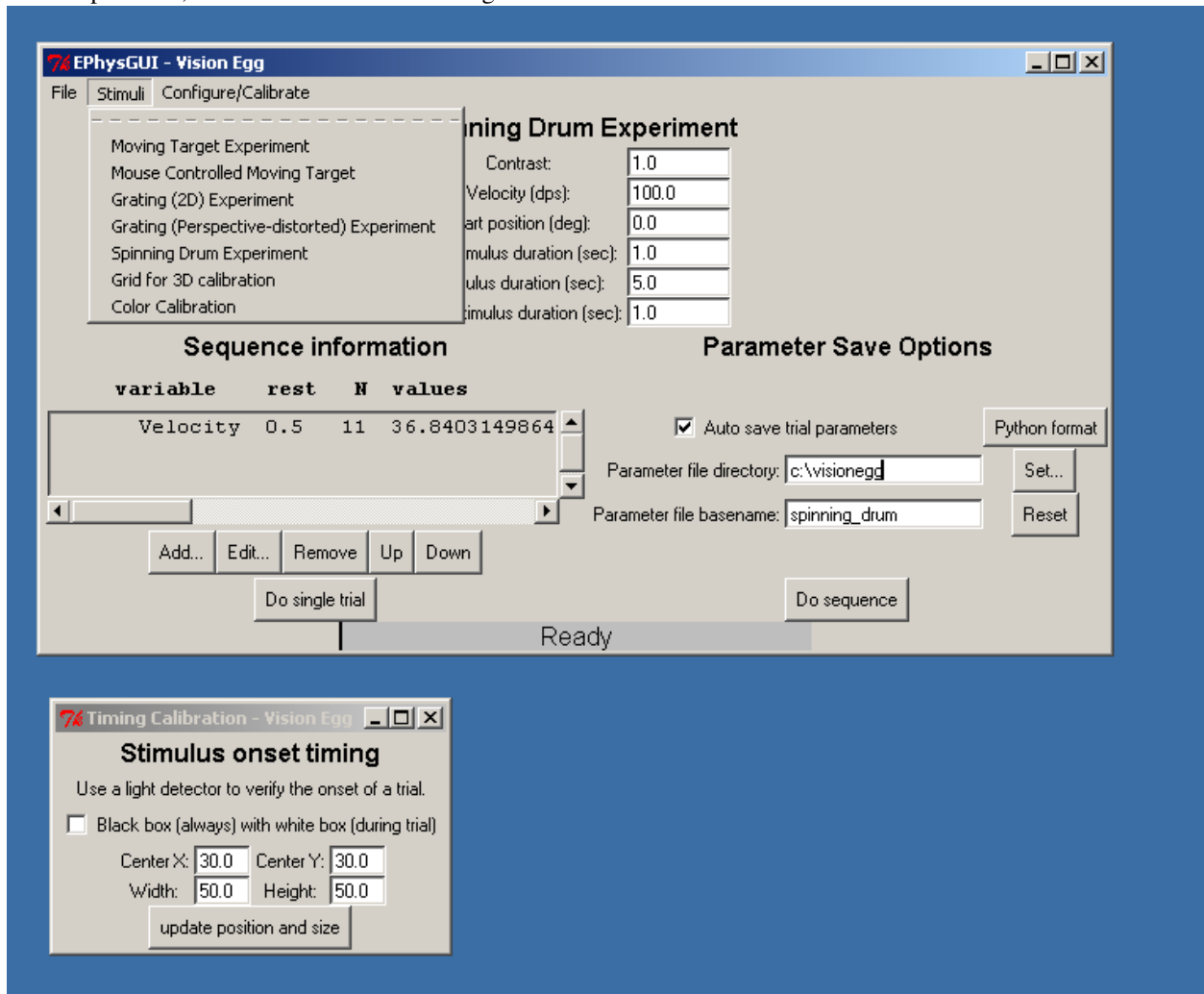
4.15.5 other demos

<p>lib3ds-demo</p> <p>3D models in the .3ds (3D Studio Max) format can be loaded. See the future page for more information regarding complex 3D models.</p> 	<p>convert3d_to_2d</p> <p>Coordinates given in 3D can be calculated to 2D screen coordinates. This may be useful for a number of reasons, including pasting text over important parts of a 3D scene.</p> 
---	---

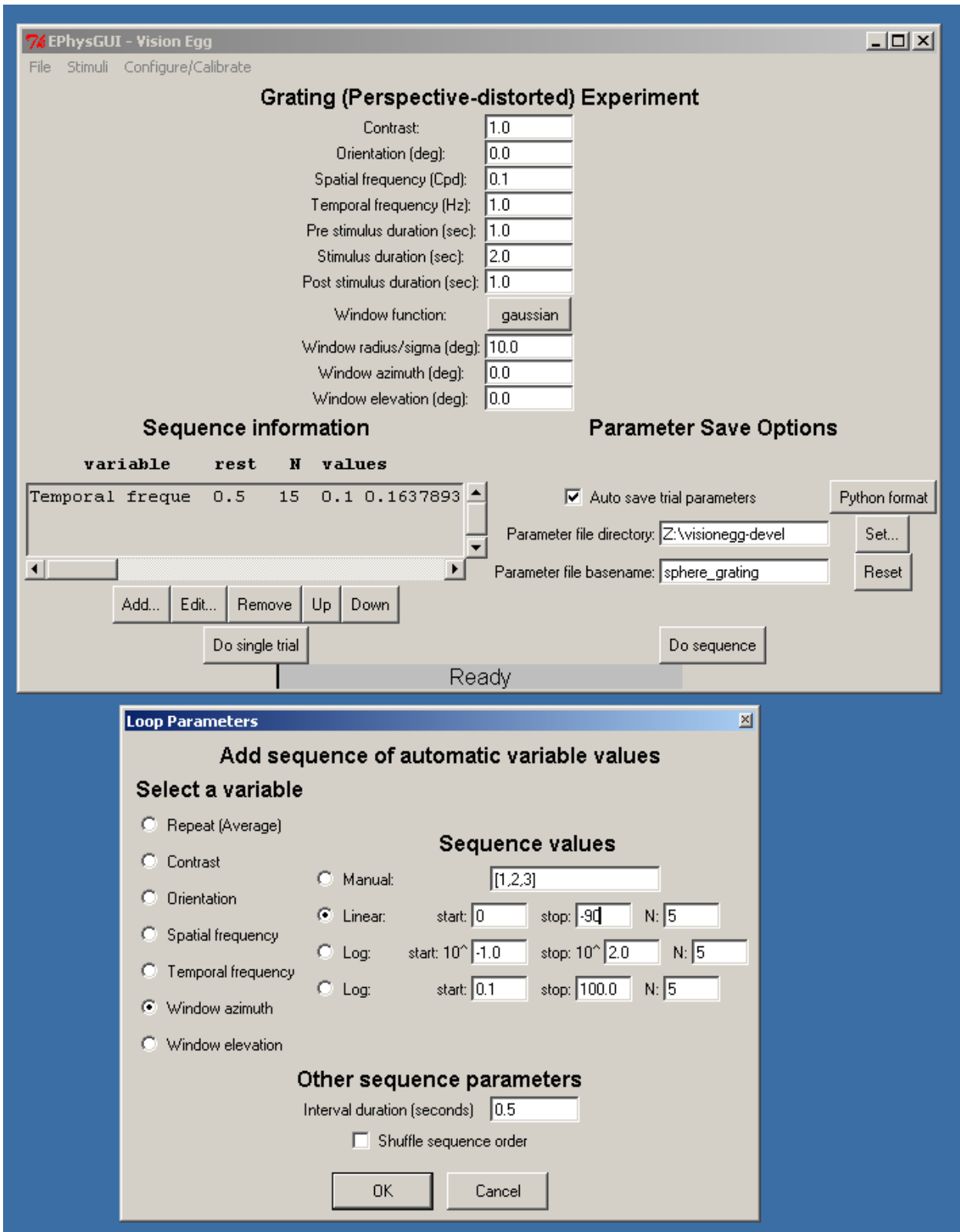
4.15.6 Electrophysiology GUI

The `ephys_gui` and `ephys_server` are your gateway to electrophysiology experiments using the Vision Egg. Because I am an electrophysiologist, this is where I have optimized the user interface. There is a modular design which allows you to copy any of the existing experiment “modules” and use them as a template for generating your own experiment using any of the built-in stimulus types.

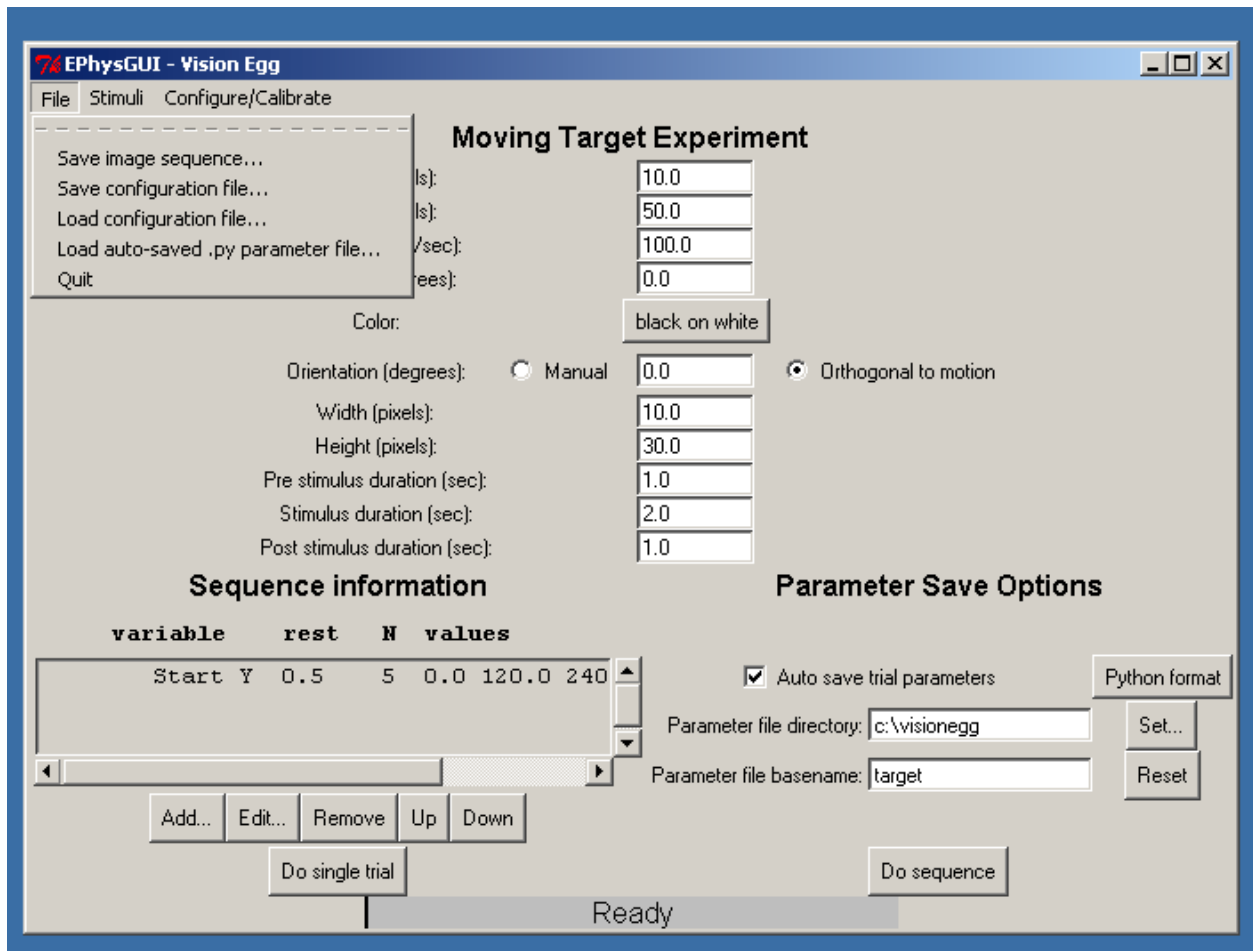
The first screenshot shows the stimuli included by default, the main window with parameter entry for the spinning drum experiment, and the stimulus onset timing calibration window.



The screenshot below shows the perspective-distorted sinusoidal grating experiment and the sequencer in use.

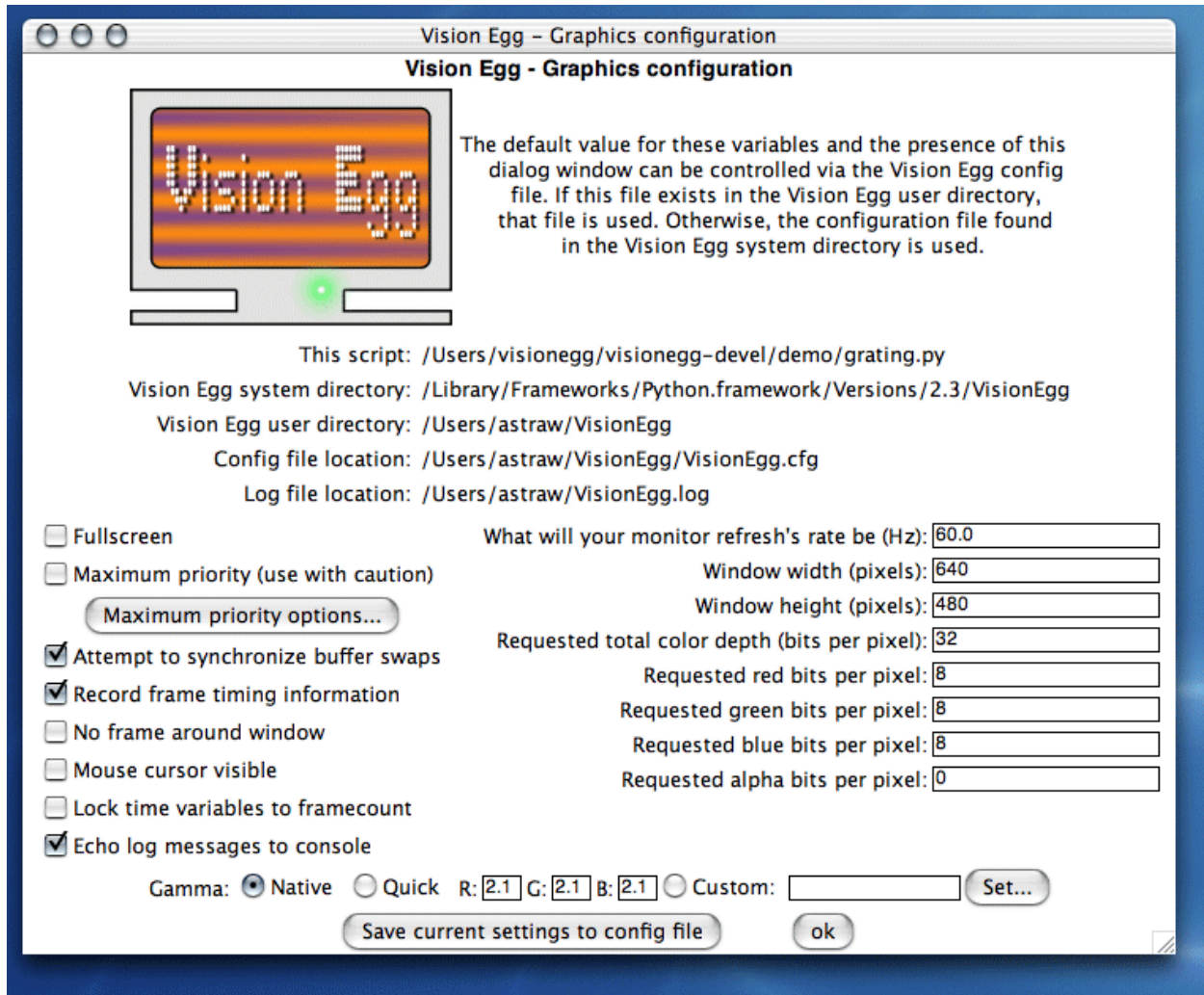


The screenshot below shows that you have the ability to load parameter files and re-play them. Also a particular trial can be (re)played as an image sequence so you can turn it into a movie.



4.15.7 Starting the Vision Egg

This is the introductory window for any Vision Egg program. (It's appearance can also be turned off.)



4.16 Intro and Overview/Synchronization

‘[[Navigation(siblings)]]’_

A final timing consideration is synchronization with data acquisition or other hardware. A common paradigm is to trigger data acquisition hardware from the Vision Egg. There are several ways to do this with the Vision Egg.

Perhaps the simplest is to control a digital output on the computer’s parallel port so that this value goes from low to high when the first frame of a stimulus is drawn. Support for this is provided for linux and win32. Unfortunately, the parallel port is can only be updated when OpenGL is instructed to swap buffers, not when the monitor actually refreshes. If you need timing accuracy better than your inter-frame interval, the best solution is to *arm* the trigger of your data aquisition device (with the parallel port output or a network command) immediately before the stimulus begins and trigger with the vertical sync pulse from the VGA cable itself.

Alternatively, you could begin data acquisition less precisely, digitize the vertical sync pulse going to the monitor along with your usual data, and align the data after the experiment is complete.

Important note: As described by Sol Simpson of SR Research, `swap_buffers()` will return immediately if no buffer is already waiting to be swapped. If a buffer *is* waiting to be swapped, then `swap_buffers()` waits until *that* buffer is swapped. Thus, your drawing actually appears as an extra frame of later than the naive expectation in this case.

Another similar note: Some projectors also introduce an additional frame of latency.

Another method, or a method to validate timing, is to use a photo detector on a patch of screen whose brightness might go from dark to light at the onset of the experiment.

Here is a figure taken from the technical specifications (Figure 5) of Burr-Brown/Texas Instruments' OPT101 Monolithic Photodiode and Single-Supply Transimpedance Amplifier that shows how this chip can be used in such a way.

'attachment:OPT101_circuit.png'_

4.17 Intro and Overview/Technologies

'[[Navigation(siblings)]]'_

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Here is a description of technology at the core of the Vision Egg. The computer language used is Python, and graphics are drawn using the OpenGL standard. Ample documentation for these technologies is available, both online and in print. This document covers why the choice was made to rely on these technologies, and why you can trust them with your experiments. It also provides some insight into how the Vision Egg works and the philosophy of its construction.

Python - <http://www.python.org>

* What is Python?

The Python computer programming language is free, easy to learn, easy to use, powerful, and flexible. Python supports modern programming techniques such as object oriented programming, threading, and high-level data types. There are countless modules for virtually any programming task. Therefore, Python is well suited for complex programming tasks. There is much Python advocacy on the internet, but let me just say that Python was chosen for the Vision Egg because creating something like this project would be a much larger burden without such a fantastic language.

* Why an interpreted language for real-time stimulus control?

A program written in an interpreted language will run more slowly than a well-written C program. But there are two reasons why Python works well for this job. The first reason is that data array manipulation is performed with high-performance, compiled C code via the Numeric module of Python. In other words, high-level Python code is only used to direct computationally intensive tasks.

The second reason why Python works for this task is that computer monitors can only be refreshed at their maximum vertical frequency, which typically ranges up to 200 Hz. At 200 Hz, your program and the graphics card have about 5 milliseconds to prepare the next frame. Modern computers can perform enormous numbers of calculations in that time, especially if the hardware is specialized for the task, such as graphics processing on a graphics card. In other words, it's fast enough. This is evidenced by the fact that running under Windows 2000, on an Athlon 1800+ system with an nVidia GeForce 2 Pro card, no frames are skipped while updating the monitor at 200 Hz. Although I

do not know of any exhaustive testing, **the Vision Egg runs for hours at a time without skipping a single frame on standard, consumer-level hardware.**

The biggest timing-related concern is unrelated to choice of programming language. A multitasking operating system might take control of the CPU from the stimulus generating program for an unacceptably long period. This is the biggest potential cause of frame-skipping latencies.

* Comparison of Python with other languages

Matlab is frequently used by vision scientists because it is well suited for data analysis and numerical modeling. However, it would be difficult to reproduce the functionality of the Vision Egg directly in Matlab because of the lack of many language features such as (useful) object oriented programming, an OpenGL interface, and easy access to routines written in C.

The C language is used for various bits of the Vision Egg, but it would be a very difficult tool with which to write an application of the scale of the Vision Egg. With today's fast computers, execution speed is at less of a premium than a scientist or programmer's time. The main role of C within the Vision Egg is as interface code that allows Python programs to call platform-specific functions. Performance critical routines could be written in C, but these optimizations have not been needed in general. It does remain an option to re-code the drawing performed by a particular stimulus class or even the entire main loop in C.

If you prefer another language, you are welcome to use the ideas found within the Vision Egg because it is open-source. Also, it would be possible to embed Python and the Vision Egg within a C program.

OpenGL - <http://www.opengl.org>

* What is OpenGL?

OpenGL is the ubiquitous cross platform way to access hardware accelerated graphics processing in today's video cards. It is an open standard 2D and 3D graphics programming interface that recent video cards support. While OpenGL is defined in C, the PyOpenGL_ project makes the OpenGL interface available in Python.

OpenGL offers features not available on traditional 2D graphics cards. Recent graphics cards are capable of realtime image scaling, sub-pixel image re-sampling and movement, perspective projections and other image warping, true color support for framebuffers and DACs greater than 8 bits (no more color lookup tables!). These are just a few of the features that OpenGL offers and that the Vision Egg uses.

The Vision Egg comes with many standard stimuli, such as sinusoidal gratings, moving rectangles, random dots, images, and checkerboards. Therefore, you may not need to learn OpenGL to take advantage of it. But if you do learn OpenGL, you can extend the Vision Egg to do anything your graphics card is capable of. OpenGL is complex and is therefore challenging to learn, but it is a standard,

so there is an incredible wealth of information on it.

Other bits used by the Vision Egg

There are a several pieces of code that extend Python in various ways required by the Vision Egg. Thanks to the developers of these great packages! PyOpenGL_ brings OpenGL to Python, pygame_ and SDL_ create OpenGL windows in a cross-platform way and get keyboard and mouse input (among many other features that the Vision Egg does not use), 'Numeric Python (Numpy)_' handles vectors and matrices of numeric data, the 'Python Imaging Library (PIL)_' handles images, and (optionally) Pyro_ allows communication between Python programs running on the same network.

```
.. _PyOpenGL: http://pyopengl.sourceforge.net
.. _pygame: http://www.pygame.org
.. _SDL: http://www.libsdl.org
.. _'Numeric Python (Numpy)': http://www.pfdubois.com/numpy
.. _'Python Imaging Library (PIL)': http://www.pythonware.com/products/pil/index.htm
.. _Pyro: http://pyro.sourceforge.net
```

4.18 Miscellaneous

'[[Navigation(children)]]'

Contents

- Miscellaneous
 - Mailing List
 - Develop
 - Eye Tracking

4.18.1 Mailing List

For discussion of the Vision Egg, visit (and subscribe to) the mailing list at <http://www.freelists.org/list/visionegg>

Once you've subscribed, you can email the list by sending email to visionegg@freelists.org.

4.18.2 Develop

Getting the Vision Egg off the ground and maintaining it has been done by AndrewStraw. However, to really thrive, the Vision Egg needs you. As an open-source project, you are free to modify the Vision Egg to suit your needs and extend it to do your experiments. It would benefit the entire vision science community, however, if you incorporate your changes back into the distribution.

Please, if you experience a bug or a Vision Egg-related wish, send it to the mailing list. Open source projects such as this one depend on the users to drive development, and reporting of bugs is the first place to start. Furthermore, other users may benefit from your report, even if you've solved the bug for yourself.

To download the development tree, see the SourceRepository page.

4.18.3 Eye Tracking

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

`SR Research`, the makers of eye tracking hardware and software, have released Pylink_.

Pylink can be used with the Vision Egg!

According to SR Research::

Pylink allows for tracker control, real-time data access, and external synchronization with eye data via custom messaging.

Many people find Python to be a simpler, yet still powerful, alternative to C. Pylink can also be used in combination with the excellent third party open source Vision Egg software; providing a combined visual presentation and eye tracking scripting package.

Distributed with Pylink is a modified Vision Egg demo using realtime tracker data to move a Gaussian-windowed grating in a gaze-contingent fashion. Following this example, it should be easy to create other VisionEgg/Pylink scripts for a variety of vision experiments involving eye tracking.

```
.. _`SR Research`: http://www.eyelinkinfo.com
.. _`Pylink`: http://www.eyelinkinfo.com/mount_software.php#Python
```

Question: Is pylink open source?

"The pylink module just wraps our C library for the EyeLink system, so although the pylink source itself is 'open', it is just a thin wrapper for a C API that is not open.

Sol Simpson
SR Research Ltd."

4.19 Miscellaneous/CreditsAndThanks

[[Navigation(siblings)]]' We would like to thank everyone who has contributed suggestions, comments, and most of all, code to the Vision Egg.

To date, the Vision Egg has been primarily developed by **AndrewStraw_**, a post-doctoral scholar at **Caltech** in the lab of **Michael Dickinson**.

This project began while I was a Ph.D. student at the **University of Washington** and later at the **University of Adelaide** in the lab of **David O'Carroll**. While a Ph.D. student, I was supported by Predoctoral Fellowship in the Biological Sciences from the **Howard Hughes Medical Institute**.

4.19.1 Code contributors:

Jamie Theobald, University of Washington (Labview code)

Major enhancements to the ephys server/GUI code to use normal (or slightly modified) demo scripts in this environment were done by Imran Ali and Lachlan Dowd in the lab of David O'Carroll at the University of Adelaide.

An initial patch for stereo support sent by Yuichi Sakano and Kevin J. MacKenzie at York University.

Parallel port enhancements by Hubertus Becker, University of Tübingen.

Arrow and FilledCircle stimuli by Hubertus Becker, University of Tübingen.

DaqKeyboard and ResponseControl by Hubertus Becker, University of Tübingen.

Full screen anti-aliasing support (FSAA) by Mark Halko, Boston University.

Various patches by Tony Arkles (University of Saskatchewan), including a suggestion to separate camera motions from the GL_PROJECTION matrix and put them in the GL_MODELVIEW matrix, where they belong.

Patch for VISIONEGG_SYSTEM_DIR by Nick Knouf, MIT.

4.20 Miscellaneous/LabView

‘[[Navigation(siblings)]]’_

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

Thanks to Jamie Theobald (Graduate Program in Neurobiology and Behavior, Zoology Department, University of Washington) for contributing Labview 6 code to control gratings made with the Vision Egg. He writes::

As of yesterday, my own Labview program - written for the Innisfree Picasso - is completely functional with the Vision Egg. It took some messing around, but I can now do experiments as well or better than before.

I intend to phase out my Labview programs over time - in favor of Python - but in the meantime it is quickest to interface with what I already have. And it's created a set of tools in Labview that control the Vision Egg gratings. It can't take advantage of everything that the Vision Egg can do, but I don't think you can do that without knowing Python. And it's not a bad start, either. It does everything I've ever needed with gratings (the contrast ramps are really nice). It uses go loops with daq triggering for synchronizing.

The Labview code is in the `''visionegg-contrib''` package at the 'Vision Egg SourceForge files page'. Control Grating.vi is the main program file in that package.

To the best of my knowledge, this is the setup that he uses:

```
.. image:: labview_schematic.gif
   :width: 434
   :height: 194
   :alt: Computer setup overview
```

Here are some screenshots he sent from Labview running on Mac OS 9.

```
.. image:: grating_control.gif
   :width: 455
   :height: 426
   :alt: Grating Control Labview panel
```

```
.. image:: waveforms.gif
   :width: 815
```

```
:height: 538
:alt: Edit Grating panel

.. image:: innards.gif
:width: 996
:height: 654
:alt: VI diagram

.. _`Vision Egg SourceForge files page`: http://sourceforge.net/project/showfiles.php?group\_id=40846
```

4.21 Miscellaneous/SimilarSoftware

‘[[Navigation(siblings)]]’_

.. raw:: html Rendering of reStructured text is not possible, please install Docutils.

There are as many ways of creating visual stimuli as there are vision experimenters. Here I have listed all current solutions that I know of. If you know of others, let me (astraw@users.sourceforge.net) know.

At a first instance, check out Hans Strasburger’s “Software for visual psychophysics” webpage, which has much more coverage than this page. I have here mainly limited listings to projects that appear to be under active development or support.

The PsychToolbox_ is a great, free program that uses Matlab to display stimuli. Historically it is a Mac OS Classic application, but now a Windows port is available. A native OpenGL Mac OS X version is on its way. (Source code is available but not open source according to the ‘Open Source Initiative’.)

PsyScript_ is a Macintosh program for generating stimuli. Free, open source (GNU Public License). Uses AppleScript and a language based on ECL (experiment control language).

Psyscope (classic Mac OS) users will be pleased to note ‘Psyscope X’_, is being ported to Mac OS X under the GNU GPL.

Presentation_ Free (no longer appears to be free: 12/2005) Windows based stimulus generation. Uses DirectX.

WinVis_ web-based and Matlab based stimulus generation programs.

VSG_ uses proprietary hardware and software to generate stimuli. It lacks any hardware-accelerated 3D features.

‘OpenRT and OpenRT-3D’_ OpenGL stimulus generation. Free, open source. (These links do not work any more... but there is OpenRT at <http://www.openrt.de/>, which is a

PsychoPy_ uses python and OpenGL together. Free, open source.

PXLab_ a collection of Java classes for running psychological experiments. Free.

Guimigolus_ is a free extension to VisionEgg, that is able to

guarantee framelossless presentation of Stimuli. We were able to create stimuli that run for **13 hours at 200Hz with not one frame lost** on a Win2k-System. This extension provides the programmer with a cpu-meter showing the time used for each frame, which is a great aid to time critical development. It has a very complex and great algorithm for measurement of buffer overruns.

Not much documentation, and only german one, because we developed it in a course at

the Carl v. Ossietzky University Oldenburg/Germany. Authors are Daniel Migowski and Guido Scherp. Download the file here:
<http://artis.uni-oldenburg.de/~migo/Guimigolus-0.7.win32.exe>

We won't develop Guimigolus any further, so we give it away now under GPL. Maybe it helps someone.

```
.. _`Software for visual psychophysics`: http://www.visionscience.com/documents/strasburger.html
.. _PsychToolbox: http://www.psychtoolbox.org
.. _`Open Source Initiative`: http://www.opensource.org/docs/definition.php
.. _PsyScript: http://www.maccs.mq.edu.au/~tim/psyscript
.. _Psyscope: http://psyscope.psy.cmu.edu
.. _`Psyscope X`: http://psy.ck.sissa.it

.. _Presentation: http://www.neurobehavioralsystems.com/software/presentation/index.html
.. _WinVis: http://www.neurometrics.com/winvis/index.jsp
.. _VSG: http://www.crs1td.com/catalog/vsg25

.. _`OpenRT and OpenRT-3D`: http://www.cs.dal.ca/~macinnwj

.. _PsychoPy: http://psychpy.sourceforge.net
.. _PXLab: http://www.pxlab.de
```

4.22 Miscellaneous/UsersAndFriends

[`\[\[Navigation\(siblings\)\]\]`_](#)

[AndrewStraw_ MartinSpacek_](#)

4.23 News

4.23.1 Frontiers in Neuroscience commentary - 2009-10-05

Rolf Kötter wrote [A primer of visual stimulus presentation software about the Vision Egg and PsychoPy for *Frontiers in Neuroscience*](#).

4.23.2 talk at SciPy 09 (video available online) - 2009-08-20

AndrewStraw gave a talk on the Vision Egg and another piece of his software, [Motmot](#), at [SciPy 09](#). See the talk at http://www.archive.org/details/scipy09_day1_05-Andrew_Straw . The Vision Egg part starts at 25:18.

4.23.3 talk and tutorial at CNS*2009 - 2009-07-22

AndrewStraw is giving a talk and demo/tutorial on the Vision Egg and another piece of his software, the [Motmot camera utilities](#) at CNS*2009 at the [Python for Neuroinformatics Workshop](#).

4.23.4 Vision Egg 1.2.1 released - 2009-07-21

Get it from [PyPI](#) while it's hot.

4.23.5 Vision Egg article in Frontiers in Neuroinformatics - 2008-10-08

An article about the Vision Egg has now been accepted for publication. The citation is:

- Straw, Andrew D. (2008) Vision Egg: An Open-Source Library for Realtime Visual Stimulus Generation. *Frontiers in Neuroinformatics*. doi: 10.3389/neuro.11.004.2008 [link](#)

This article covers everything from descriptions of some of the high-level Vision Egg features, to low-level nuts-and-bolts descriptions of OpenGL and graphics hardware relevant for vision scientists. Also, there is an experiment measuring complete latency (from USB mouse movement to on-screen display). In the best configuration (a 140Hz CRT with vsync off), this latency averaged about 12 milliseconds.

If you use the Vision Egg in your research, I'd appreciate citations to this article.

4.23.6 BCPy2000, using the Vision Egg, released - 2008-10-01

The Vision Egg has found another application: Brain-Computer Interfaces.

Dr. Jeremy Hill (Max Planck Institute for Biological Cybernetics) announced the release of [BCPy2000](#), a framework for realtime biosignal analysis in python, based on the modular, popular (but previously pythonically challenged) [BCI2000](#) project.

4.23.7 Vision Egg 1.1.1 released - 2008-09-18

This is primarily a bug fix release to Vision Egg 1.1.

Changes for 1.1.1

- Various small bugfixes and performance improvements:
- Removed old CVS cruft from `VisionEgg/PyroClient.py` `VisionEgg/PyroHelpers.py`
- Fix trivial documentation bugs to have the correct version number.
- Workaround pygame/SDL issue when creating Font objects. (r1491, reported by Jeremy Hill)
- bugfix: allow 4D as well as 3D vectors to specify vertices (r1472, r1474)
- fix comments: improve description of coordinate system transforms (r1473)
- Use standard Python idiom (r1475)
- Further removal of 'from blah import *' (r1476, r1501)
- Minor performance improvement (r1486)
- Remove unintended print statement (r1487 thanks to Jeremy Hill)

- properly detect String and Unicode types (r1470, reported by Dav Clark)
- update license to mention other code (r1502)

4.23.8 Vision Egg 1.1 released - 2008-06-07

This release brings the Vision Egg up to date with numpy and the new ctypes-based PyOpenGL, and includes lots of smaller bugfixes and removes old cruft that had been accumulating.

Changes for 1.1

- Explicitly require Python 2.3 by removing duplicate of Python standard library's logging module and assume the "True" and False" are defined. There were probably other assumptions depending on 2.3 creeping into the code, as well.
- Removed Lib3DS module.
- Workaround PyOpenGL 3.0.0a and 3.0.0b1 bug in glLoadMatrixf().
- Fix SphereMap.AzElGrid_() to properly draw iso elevation and azimuth lines at specified intervals.
- Change to use numpy at the core instead of Numeric. (Require numpy, no longer require Numeric.)
- Require setuptools
- No longer supporting Python 2.2
- Update Textures to accept numpy arrays as data source (by Martin Spacek)
- Update to work with PyOpenGL 3
- Changes so to reflect standard idioms on Python project layouts: src/ directory renamed to VisionEgg/ (but .c files need to be moved back to src/), use 'package_data' distutils argument. This enables setuptools to work with the Vision Egg.
- QuickTime movies work in Windows (need to check on OS X).
- FilledCircle stimulus is now anti-aliased. Thanks to Peter Jurica and Gijs Plomp.
- Added demo/texture3D_alpha.py

4.23.9 Vision Egg 1.0 released - 2006-01-03

Changes for 1.0

- Major enhancements to the ephys server/GUI code to use normal (or slightly modified) demo scripts in this environment were one by Imran Ali and Lachlan Dowd in the lab of David O'Carroll at the University of Adelaide.
- An initial patch for stereo support sent by Yuichi Sakano and Kevin J. MacKenzie at York University.
- Parallel port enhancements by Hubertus Becker, University of Tübingen.
- Arrow and FilledCircle stimuli by Hubertus Becker, University of Tübingen.
- DaqKeyboard and ResponseControl by Hubertus Becker, University of Tübingen.
- Full screen anti-aliasing support (FSAA) by Mark Halko, Boston University.
- Various patches by Tony Arkles (University of Saskatchewan), including a suggestion to separate camera motions from the GL_PROJECTION matrix and put them in the GL_MODELVIEW matrix, where they belong.

- Patch for VISIONEGG_SYSTEM_DIR by Nick Knouf, MIT.
- Added win32_vretrace.WaitForRetrace() (but it's not used for much, yet)
- Enhancements to EPhys Server/GUI sequencer
- Added 'lat-long rectangle' to available 3D masking windows
- Moved controller.CONSTANTS into FlowControl module namespace
- Numerous bugfixes

4.23.10 Quest.py announced - 2005-04-08

The popular QUEST algorithm by Denis Pelli has been ported to Python. See the Quest page for more details.

4.23.11 Pylink (by SR Research) - Eye tracking in Python - 2004-02-25

SR Research, the makers of eye tracking hardware and software, have released [Pylink](#).

Pylink can be used with the Vision Egg!

According to SR Research:

Pylink allows for tracker control, real-time data access, and external synchronization with eye data via custom messaging.

Many people find Python to be a simpler, yet still powerful, alternative to C. Pylink can also be used in combination with the excellent third party open source Vision Egg software; providing a combined visual presentation and eye tracking scripting package.

Distributed with Pylink is a modified Vision Egg demo using realtime tracker data to move a Gaussian-windowed grating in a gaze-contingent fashion. Following this example, it should be easy to create other !VisionEgg/Pylink scripts for a variety of vision experiments involving eye tracking.

4.23.12 Release 0.9.9 - 2003-09-19

The Vision Egg 0.9.9 is here!

There are several major improvements. (The few changes that may break old code are detailed in the release notes included in this email).

There is nothing more I intend to add before I release Version 1.0 – this is a release candidate subject to final testing and bug fixing, so I would appreciate all the abuse you can put it through. In particular, test/conform.py runs many tests on your system and reports the output.

Changes for 0.9.9:

- Screen.put_pixels() method for blitting of raw pixel data
- Support for QuickTime movies (currently Mac OS X only)
- Redesign of type check system for accuracy and clarity
- TrueType font rendering with SDL_ttf2
- Textures with alpha-bugfixes and examples
- Positioning of viewports and 2D stimuli can use relative positioning anchors

- Now requires Python 2.2 – new style classes used to restrict attribute access
- Now requires pygame 1.5
- Renamed `timing_func()` to `time_func()`
- EPhysGUI saves absolute time a trial was started (to reconstruct all stimuli)
- Allow use of pygame Surfaces as source of texture data
- Mipmaps of sphere-mapped sinusoidal grating to prevent spatial aliasing
- De-emphasis on Presentation and Controller classes (moved to FlowControl module)
- Changed orientations such that 0 degrees is right and 90 degrees is up.
- Bugfix in SphereMap module – gaussian formula produced windows too wide by $2/\sqrt{2}$
- Allow conversion of 3D vertices into 2D screen coordinates
- Added wireframe azimuth/elevation grid with text labels
- Allow arbitrary orientation of textures and text with angle parameter
- FrameTimer class now available for use in your own main loops
- Use Python 2.3 logging module (copy included for use with Python 2.2)
- No installation of demos or documentation (get source or demo package)
- Many small enhancements and bugfixes

New tests:

- high-voltage regulation test for displays (Brainard et al., 2002)
- incomplete DC restoration test for displays (Brainard et al., 2002)
- unit-test suite: among many other things, pixel accuracy of textures

New demos:

- `mpeg.py` plays MPEG movies (currently seeking a free movie to include)
- `quicktime.py` plays QuickTime movies (currently Mac OS X only)
- `convert3d_to_2d.py` converts 3D positions to 2D positions
- `dots_simple_loop.py` uses own loop rather than Presentation class
- `makeMovie2.py` makes a movie with `get_framebuffer_as_image()` function
- `mouse_gabor_2d.py` shows a gabor wavelet under mouse and keyboard control
- `mouse_gabor_perspective.py` is `sphereGratingWindowed.py` improved and renamed
- `mouseTarget_user_loop.py` uses own loop rather than Presentation class
- `multi_stim.py` shows many stimuli at once

4.24 Quest

Contents

- Quest
 - Overview
 - Download
 - Example
 - References
 - License

4.24.1 Overview

The `Quest` algorithm has been ported directly from the MATLAB sources available with the `PsychToolbox`. The MATLAB source code was written by Denis Pelli. Thanks to Denis for allowing it to be released under the BSD license to the (Python) world.

This Python version **does not depend on the Vision Egg**, and may be useful in other contexts.

4.24.2 Download

Download the source directly from the `Quest` package at the [SourceForge download page](#).

4.24.3 Example

The intensity scale is abstract, but usually we think of it as representing log contrast. Specify true threshold of simulated observer: 0.5

Estimate threshold: 0.4

```
Trial 1 at 0.7 is right
Trial 2 at 0.4 is right
Trial 3 at -0.2 is wrong
Trial 4 at 0.7 is right
Trial 5 at 0.4 is right
Trial 6 at 0.4 is wrong
Trial 7 at 1.1 is right
Trial 8 at 0.9 is right
Trial 9 at 0.7 is right
Trial 10 at 0.7 is right
```

36 ms/trial

Mean threshold estimate is 0.60 +/- 0.38

Quest beta analysis. Beta controls the steepness of the Weibull function.

Now re-analyzing with beta as a free parameter. . . .

```
logC    sd    beta    sd    gamma
 0.62   0.39   3.7    4.5    0.500
```

Actual parameters of simulated observer:

```
logC    beta    gamma
 0.50    3.5     0.50
```

The example above is taken directly from the `demo()` function of `Quest.py` and is a direct translation of the demo included in the original MATLAB source:

```

print 'The intensity scale is abstract, but usually we think of it as representing log contrast.'
tActual = None
while tActual is None:
    sys.stdout.write('Specify true threshold of simulated observer: ')
    input = raw_input()
    try:
        tActual = float(input)
    except:
        pass

tGuess = None
while tGuess is None:
    sys.stdout.write('Estimate threshold: ')
    input = raw_input()
    try:
        tGuess = float(input)
    except:
        pass

tGuessSd = 2.0 # sd of Gaussian before clipping to specified range
pThreshold = 0.82
beta = 3.5
delta = 0.01
gamma = 0.5
q=QuestObject(tGuess,tGuessSd,pThreshold,beta,delta,gamma)

# Simulate a series of trials.
trialsDesired=100
wrongRight = 'wrong', 'right'
timeZero=time.time()
for k in range(trialsDesired):
    # Get recommended level. Choose your favorite algorithm.
    tTest=q.quantile()
    #tTest=q.mean()
    #tTest=q.mode()
    tTest=tTest+random.choice([-0.1,0,0.1])
    # Simulate a trial
    timeSplit=time.time(); # omit simulation and printing from reported time/trial.
    response=q.simulate(tTest,tActual)
    print 'Trial %3d at %4.1f is %s'%(k+1,tTest,wrongRight[response])
    timeZero=timeZero+time.time()-timeSplit;

    # Update the pdf
    q.update(tTest,response);
# Print results of timing.
print '%.0f ms/trial'%(1000*(time.time()-timeZero)/trialsDesired)
# Get final estimate.
t=q.mean()
sd=q.sd()
print 'Mean threshold estimate is %4.2f +/- %.2f'%(t,sd)
#t=QuestMode(q);
#print 'Mode threshold estimate is %4.2f'%t
print '\nQuest beta analysis. Beta controls the steepness of the Weibull function.\n'
q.beta_analysis()
print 'Actual parameters of simulated observer:'
print 'logC beta      gamma'
print '%5.2f      %4.1f      %5.2f'%(tActual,q.beta,q.gamma)

```

4.24.4 References

- Watson, A. B. and Pelli, D. G. (1983) QUEST: a Bayesian adaptive psychometric method. *Percept Psychophys*, 33 (2), 113-20.
- Pelli, D. G. (1987) The ideal psychometric procedure. *Investigative Ophthalmology & Visual Science*, 28 (Suppl), 366.
- King-Smith, P. E., Grigsby, S. S., Vingrys, A. J., Benes, S. C., and Supowit, A. (1994) Efficient and unbiased modifications of the QUEST threshold method: theory, simulations, experimental evaluation and practical implementation. *Vision Res*, 34 (7), 885-912.

4.24.5 License

The Python Quest package is released under a BSD-style license. (The Vision Egg itself has a LGPL license.)

Indices and tables

- *genindex*
- *modindex*
- *search*