pygrafix Documentation

Release 0.0.1

Orson Peters

May 19, 2012

CONTENTS

pygrafix is a Python/Cython hardware-accelerated 2D graphics library.

Contents:

CHAPTER

ONE

INTRODUCTION

This is a short introduction to pygrafix.

1.1 Why this project?

There are two major game/graphics libraries for Python out there, Pygame and Pyglet. Pygame is built ontop of the SDL library and is very mature. It however lacks a clean interface and has serious performance issues if you aren't very careful about using it (and even then some things are plain impossible). It also has some great parts, mostly the non-graphics modules are quite good. Then there is pyglet, a pure-python approach to graphics. It is very promising and has a much cleaner API but uses an awkward bottom-left coordinate system. It is also written in pure Python code with ctypes, which means potential performance issues can lie below the surface due to the amount of wrapping going on. This project tries to combine the great parts of both (and other) libraries while removing the awkwardness.

1.2 Why Cython?

Cython allows the rapid development of Python to be combined with the speed of C. It is also very easy to wrap C libraries with, resulting in great performance too. Since I already decided parts of the library will be in C Cython has no portability drawbacks of some kind. It will run on any system capable of running Python, and produce portable C code. If necesarry you could even ship the C code produced by Cython; this means your code base is completely portable, even for people that can't/don't want to install Cython. This way they will only need Python and a C compiler (you will most likely be shipping binaries to your end users removing the need of the C compiler too).

1.3 On what external libraries does pygrafix rely?

pygrafix currently relies on GLFW for windowing, OpenGL for graphics rendering, GLEW for more OpenGL functionality and stb_image for image writing/reading. GLFW and stb_image are shipped with the source of pygrafix, GLEW and OpenGL are not. To compile you need the developer 32-bit version of Python, a C compiler, CMake and Cython installed.

1.4 What are the core design goals?

- Opening and using windows and recieving input
- Loading common image formats into textures

- Loading and using fonts
- A fast 2D sprite system supporting many transformations.
- Providing clear, fast and object-oriented interfaces to all features
- Loading and playing back sound

1.5 What other features might get added?

- Particle systems
- Offscreen rendering targets
- (2D) game specific high-performance modules (fast collision, vector, etc)

1.6 What features will not get added?

- Networking
- Game-engine specific stuff
- 3D functionality

CHAPTER

TWO

COMPILING

Compiling pygrafix can be pretty hard for a novice, but these instructions should help you on your way.

2.1 Compiling on Linux

First you need to install the required software if you don't have it yet. If you don't have CMake yet (though that's unlikely) get it:

```
$ sudo apt-get install cmake
```

Now we need Cython. Cython can be downloaded from the repos but often is outdated. So download it from http://www.cython.org/ and build it from source (very easy).

Now we need to compile pygrafix itself.

First you need to compile GLFW 3. GLFW 3 is currently an unstable branch of GLFW, so it's not in the repos. This also means that you shouldn't install it. In order to compile GLFW3 you should go into the libs/glfw directory and make a new directory "build". **cd** into the directory and call **cmake** on the above directory. After that you want to call **make** (BUT NOT **make install**):

```
$ mkdir libs/glfw/build
$ cd libs/glfw/build
$ cmake ..
$ make
```

After it's done go back to the top directory of pygrafix and invoke the build script:

```
$ cd ../../..
$ sudo python setup.py install
```

You're done!

2.2 Compiling on Windows

Windows doesn't come with a C compiler, so I recommend installing GCC. This can be done by installing MinGW through TDM-GCC (http://tdm-gcc.tdragon.net/). Make sure to get the 32-bit version, just like Python.

You'll also need CMake, so get it from http://www.cmake.org/ (win32 installer). The same goes for Cython, get it from http://www.cython.org/ (win32 installer).

First we need to compile GLFW:

```
> mkdir libs\glfw\build
> cd libs\glfw\build
> cmake -G "MinGW Makefiles" ..
> make
```

And then pygrafix itself:

```
> cd ..\..\..
> python setup.py build --compiler=mingw32
> python setup.py install
```

It might be possible that you get an error about "-mno-cygwin". In order to fix this you must go to your python install folder, find a file named "distutils.py" and remove all occurences of "-mno-cygwin". There sadly is no other way.

Another thing, CMake might be giving you an error that it can not find a working GCC. I fixed this problem by opening my MinGW install directory and copy binmake.exe to binmingw32-make.exe.

CHAPTER

THREE

DISCLAIMER

Large amounts of code and ideas have been... lent from other projects. Keeping all copyright notices in the code would be unpractical, thus I have created a directoy called “licenses”. All copyright owners will get a place in this folder, either by their own or their project’s name. Also pygrafix’s license can be found there.

Modules:

CHAPTER

FOUR

PYGRAFIX — GENERAL PYGRAFIX FUNCTIONS

This module is only there to group all the other modules, and for getting metadata about pygrafix.

pygrafix.**get_version**()

Returns a tuple of three ints containing the current pygrafix version in the format *(major, minor, revision)*.

CHAPTER

FIVE

PYGRAFIX.DRAW — FUNCTIONS FOR DRAWING SHAPES

This module allows you to draw simple shapes like lines and polygons.

pygrafix.draw.**line**(*start_point*, *end_point*[, *color*[, *width*[, *edge_smoothing*[, *blending*]]]])

This function draws a line between *start_point* and *end_point*. Both points must have the form *(x, y)*. *color* must have the form *(red, green, blue[, alpha])* with all components *0 <= c <= 1*. *blending* can be any of *"add"*, *"multiply"*, *"mix"* or *None*.

The defaults are an opaque white line, 1 pixel wide, with no smoothing and mix blending.

pygrafix.draw.**rectangle**(*position*, *size*[, *color*[, *edge_smoothing*[, *blending*]]])

Draws a filled rectangle. *position* must be given in the form *(x, y)*, *size* in the form *(width, height)*. *color* must have the form *(red, green, blue[, alpha])* with all components *0 <= c <= 1*. *blending* can be any of *"add"*, *"multiply"*, *"mix"* or *None*.

The defaults are an opaque white rectangle, with no smoothing and mix blending.

pygrafix.draw.**rectangle_outline**(*position*, *size*[, *color*[, *width*[, *edge_smoothing*[, *blending*]]]])

Draws the outline of a rectangle. *position* must be given in the form *(x, y)*, *size* in the form *(width, height)*. *width* is a number specifying the width of the line used. The thickness of the line moves inward, so the total size of the outline is still given by *size*, and not a combination of *size* and *width*. *color* must have the form *(red, green, blue[, alpha])* with all components *0 <= c <= 1*. *blending* can be any of *"add"*, *"multiply"*, *"mix"* or *None*.

The defaults are an opaque white rectangle, with no smoothing and mix blending.

pygrafix.draw.**polygon**(*vertices*[, *color*[, *edge_smoothing*[, *blending*]]])

This function draws a polygon with the given vertices. *vertices* must be a list of *(x, y)* tuples. At least 3 vertices must be given. *color* must have the form *(red, green, blue[, alpha])* with all components *0 <= c <= 1*. *blending* can be any of *"add"*, *"multiply"*, *"mix"* or *None*.

The defaults are an opaque white polygon, with no smoothing and mix blending.

pygrafix.draw.**polygon_outline**(*vertices*[, *color*[, *width*[, *edge_smoothing*[, *blending*]]]])

This function draws the outline of a polygon with the given vertices. *vertices* should be a list of *(x, y)* tuples. At least 3 vertices must be given. *color* must have the form *(red, green, blue[, alpha])* with all components *0 <= c <= 1*. *blending* can be any of *"add"*, *"multiply"*, *"mix"* or *None*.

The defaults are an opaque white 1 pixel outline, with no smoothing and mix blending.

CHAPTER

SIX

PYGRAFIX.IMAGE — WORKING WITH IMAGE FILES

This module is used for loading images into a format pygrafix can understand.

`pygrafix.image.`**load**(*filename*[, *file*[, *decoder*]])
Loads an image from a file. If *file* is passed *filename* will be used as a hint for the filetype. Optionally you can specify a *decoder* argument which will be used for decoding the image, for more information about decoders read `pygrafix.image.codecs`. Returns a `Texture` object.

class `pygrafix.image.`**Texture**(*internal_texture*[, *region*])
Creates a `Texture` object from an `InternalTexture` object. Normally you shouldn't have to create `Texture` objects yourself, use `load()` instead. The optional region argument is the region to use, this defaults to *(0, 0, internal_texture.width, internal_texture.height)*.

width
The width of the texture. Note that when the *region* attribute is set that this will return the width of the region. To get the actual texture width use *internal_texture.width*. Read-only.

height
The height of the texture. Note that when the *region* attribute is set that this will return the height of the region. To get the actual texture width use *internal_texture.height*. Read-only.

region
A tuple in the form *(x, y, width, height)* that describes a region of the internal texture that gets represented by this texture. Read-write.

internal_texture
The internal texture used for this texture. Read-only.

copy([*lazycopy*])
Returns a copy of this texture. If *lazycopy* is True this function equals `texture.get_region(0, 0, texture.width, texture.height)`. *lazycopy* is False by default.

get_region([*x*[, *y*[, *width*[, *height*]]]])
Returns a `Texture` object that represents a region of this texture, starting *(x, y)* pixels from the topleft of this texture, spanning *(width, height)* pixels. Any changes to the original texture will be represented in this region too, use `texture.copy().get_region(...)` if that's undesired behaviour.

class `pygrafix.image.`**InternalTexture**(*imgdata*)
Creates a `InternalTexture` object from an `ImageData` object. Normally you shouldn't have to create `InternalTexture` objects yourself, use `load()` instead.

width
The width of the texture. Read-only.

height
The height of the texture. Read-only.

class pygrafix.image.**ImageData**(*width*, *height*, *format*, *data*)
The format used to represent raw image data. *format* can be any of *"RGBA", "RGB", "LA", "A"*. Data must be bytes data given in the format described. Only 8-bit channels are supported.

CHAPTER

SEVEN

PYGRAFIX.IMAGE.CODECS — MANAGING CODECS

This module is used for managing codecs to be used by pygrafix. pygrafix supports a few formats out of the box, but by adding image codecs it's possible to add more codecs for any format.

exception pygrafix.image.codecs.**ImageDecodeException**
The error raised when decoding an image fails.

exception pygrafix.image.codecs.**ImageEncodeException**
The error raised when encoding an image fails.

pygrafix.image.codecs.**get_decoders**([*filename*])
Return all decoders that could possibly decode the format described by the extension of *filename*. If *filename* is not given it returns all decoders.

pygrafix.image.codecs.**get_encoders**([*filename*])
Return all encoders that can encode the format described by the extension of *filename*. If *filename* is not given it returns all encoders.

pygrafix.image.codecs.**add_decoder**(*decoder*)
Adds *decoder* to pygrafix. A decoder must support two methods: `get_extensions()` and `decode(file, filename)`.

`get_extensions()` must return an iterable of extensions the decoder can decode, for example:

```
def get_extensions(self):
    return (".bmp", ".png")
```

`decode(file, filename)` must attempt to decode the file object *file*. *filename* is a hint regarding the containing file type (which can be a full filename or just the extension). If, for any reason, the decoder is not able to decode *file* it must raise `ImageDecodeException`. If it succeeds it must return an `ImageData` object containing the decoded data.

pygrafix.image.codecs.**add_encoder**(*encoder*)
Adds *encoder* to pygrafix. An encoder must support two methods: `get_extensions()` and `encode(imgdata, file, filename)`.

`get_extensions()` must return an iterable of extensions the encoder can encode, for example:

```
def get_extensions(self):
    return (".bmp", ".png")
```

`encode(imgdata, file, filename)` must encode the data found in *imgdata* into the file object *file*. *filename* is a hint into which file type the data must be encoded (which can be a string containing the full filename or just the extension). *imgdata* is passed as an `ImageData` object.

CHAPTER

EIGHT

PYGRAFIX.RESOURCE — MANAGING RESOURCE LOCATIONS

This module is used for managing resource locations from which pygrafix can load resources.

pygrafix.resource.**add_location**(*location*)

Adds a resource location to pygrafix. These resource locations are used in the functions `get_path()`, `get_file()`, `get_location()` and `exists()` as well as other resource loading functions throughout pygrafix (for example `pygrafix.image.load()`).

The *location* variable can be a string containing a path, a string containing the path to a zipfile as well as a custom object.

If a custom object is passed it must support at least the following methods:

```
class CustomLocation:
    # tries to open filename in this location with the correct mode
    # if for any reason this fails or the resource is not found raise IOError
    def open(self, filename, mode = "rb"):
        pass

    # return True if filename exists in this location, otherwise False
    def isfile(self, filename):
        pass

    # return an absolute path to filename in this location, whether it exists or not
    # if absolute paths are not applicable return anything you find appropriate, but don't raise
    def getpath(self, filename):
        pass
```

pygrafix.resource.**get_path**(*resource*)

resource is a string containing the filename of a resource. This function will look through all resource locations and return an absolute path to the resource. `IOError` is raised if the resource was not found.

Note: an absolute path is not always available/appropriate, for example a path into a zipfile. Use this function for printing purposes only, or with care.

pygrafix.resource.**get_file**(*resource*)

resource is a string containing the filename of a resource. This function will look through all resource locations for the file and return a file object opened in binary reading mode. `IOError` is raised if the resource was not found.

pygrafix.resource.**get_location**(*resource*)

resource is a string containing the filename of a resource. This function will look through all resource locations for the resource, and if it's found the function will return the containing location.

pygrafix.resource.**exists**(*resource*)
resource is a string containing the filename of a resource. This function will look through all resource locations and return True if the resource exists, otherwise False.

pygrafix.resource.**get_script_home**()
Returns a string containing the directory of the program entry module.

For ordinary Python scripts, this is the directory containing the __main__ module. For executables created with py2exe the result is the directory containing the running executable file. For OS X bundles created using Py2App the result is the Resources directory within the running bundle.

If none of the above cases apply and the file for `__main__` cannot be determined the working directory is returned.

pygrafix.resource.**get_settings_path**(*name*)
Returns a string containing a directory to save user preferences.

Different platforms have different conventions for where to save user preferences, saved games, and settings. This function implements those conventions. Note that the returned path may not exist: applications should use `os.makedirs()` to construct it if desired.

On Linux, a hidden directory *name* in the user's home directory is returned.

On Windows (including under Cygwin) the *name* directory in the user's `Application Settings` directory is returned.

On Mac OS X the *name* directory under `~/Library/Application Support` is returned.

CHAPTER

NINE

PYGRAFIX.SPRITE — FAST SPRITES

This module gives you fast sprites that can be moved, rotated, scaled and colored.

class pygrafix.sprite.**Sprite**(*texture*)

Creates a new sprite with the texture *texture*. This is the core rendering functionality of pygrafix. A sprite can be moved, rotated (around a point within the sprite), scaled (with different scales for x and y), flipped and colored. All of this only affects the sprite, not the texture that it uses. Multiple sprites can be created off of one texture.

x

The horizontal position of the sprite.

y

The vertical position of the sprite.

position

A property which can be used for reading/modifying *x* and *y* at the same time. For example:

```
>>> print(sprite.x, sprite.y, sprite.position)
5, 10, (5, 10)
>>> sprite.position = (60, 7)
>>> print(sprite.x, sprite.y, sprite.position)
60, 7, (60, 7)
```

anchor_x

The horizontal position of the sprites' anchor.

anchor_y

The vertical position of the sprites' anchor.

anchor

A shorthand for assigning to *anchor_x* and *anchor_y* at the same time.

The anchor of a sprite is used to determine how to place a sprite, even when scaled and rotated. The anchor of a sprite also rotates and scales with the sprite. Finally when a sprite is rendered pygrafix makes sure that the anchor point of the sprite always lies on the sprites' *position*.

rotation

The rotation of the sprite. The sprite will be rotated around the anchor.

width

Shorthand for `sprite.texture.width`. Read-only.

height

Shorthand for `sprite.texture.height`. Read-only.

size

Shorthand for `(sprite.width, sprite.height)`. Read-only.

draw([*scale_smoothing*[, *edge_smoothing*[, *blending*]]])
Draws the sprite as defined by it's properties. *scale_smoothing* is a boolean indicating whether the sprite should be drawn nicely smoothed when scaled or pixelated. *blending* can be any of "add", "multiply" and "mix", or None to disable blending.

pygrafix.sprite.**draw_batch**(*sprites*[, *preserve_order*[, *scale_smoothing*[, *edge_smoothing*[, *blending*]]]])
Draws a list of sprites in one go. This is the main rendering function of pygrafix, and depending on the application this function will do the most work. This function draws each sprite in *sprite* with the attributes *scale_smoothing*, *edge_smoothing* and *blending* as describute in `Sprite.draw()`.

This function is the most efficient when a lot of sprites use the same `InternalTexture` and sprites with the same texture are grouped together. By default no particular order of drawing is guaranteed by this function, for speed purposes sprites are sorted on texture. If you absolutely need a specific order of drawing, pass True to *preserve_order* (by default it's False) or consider slicing up your drawing in smaller batches.

CHAPTER

TEN

PYGRAFIX.WINDOW — MANAGING WINDOWS

This module allows you to open and manage your windows to be used for pygrafix.

class pygrafix.window.**Window**([*width*[, *height*[, *title*[, *fullscreen*[, *resizable*[, *refresh_rate*[, *vsync*[, *bit_depth*]]]]]]]])

Creates a new window. *width* and *height* give the size of the new window, *title* is a string for the window caption, *fullscreen* is a boolean indicating whether the new window is fullscreen or not. *resizable* is a boolean indicating whether the new window is resizable by the user. *refresh_rate* is the refresh rate in Hz (only used in fullscreen). *vsync* is a boolean indicating whether vsync should be enabled. *bit_depth* is a tuple in the form *(red, green, blue, alpha)* indicating how much bits should be used for each channel.

If *width* is zero (the default), it will be calculated as `width = (4/3) * height`, if *height* is not zero. If *height* is zero (the default), it will be calculated as `height = (3/4) * width`, if *width* is not zero. If both *width* and *height* are zero, *width* will be set to 640 and *height* to 480.

If no title is given the default title "pygrafix window" is chosen. The default value for *resizable* is False. If *refresh_rate* is zero (the default) the system's refresh rate will be used. The default value for *vsync* is True. If *bit_depth* is not given pygrafix will choose the best values.

pygrafix only supports double-buffered windows. This means that all drawing gets done on the back buffer and the user only ever sees the front-buffer. This is done to prevent half-done frames from showing up to the user. You swap the front and the back buffer by calling `flip()`.

All attributes are read-write unless said otherwise.

width
The width of the screen in pixels.

height
The height of the screen in pixels.

size
The size of the window in the form *(width, height)*.

position
The position of the window in the form *(x, y)*. *x* and *y* are measured in pixels relative to the topleft of the screen.

resizable
A boolean indicating if the window is resizable by the user (you can always resize the window from the code). Read-only.

refresh_rate
The refresh rate of the window. Only applicable in full-screen. Read-only.

vsync
A boolean indicating whether vsync is enabled or not.

mouse_cursor
Defines the cursor mode. Legal modes are *"normal"*, *"hidden"* and *"captured"*. In normal mode the regular hardware cursor is used and all mouse position functions work normally. In hidden mode everything is the same, except the cursor is not shown. Captured mode is radically different, the cursor is hidden and is not blocked by window boundaries. This last mode is commonly used for first-person-shooters.

key_repeat
A boolean indicating whether key repeating is enabled or not.

title
The title of this window.

fullscreen
A boolean indicating whether the window is fullscreen or not.

close()
Closes the window.

is_open()
Returns whether the window is open or not.

poll_events()
Calling this will pump through new window events like keypresses. Call this at least once per frame.

wait_events()
Does the same as the `poll_events()` but sleeps the process until an event is triggered.

minimize()
Minimizes the window.

restore()
Restores the window.

has_focus()
Returns a boolean indicating whether this window has focus.

is_minimized()
Returns a boolean indicating if the window is minimized.

switch_to()
Makes this window the active window (the window that is drawn on).

flip()
This flips the front and the back and makes everything that has been drawn visible to the user. Call this once per frame.

get_mouse_position()
Returns the position of the mouse relative to the topleft of the screen in the form *(x, y)*.

is_key_pressed(*key*)
Returns True if *key* is pressed, else False. *key* can be a key constant from `pygrafix.window.key` or an alphanumeric string of length one (for example *"A"*).

is_mouse_button_pressed(*button*)
Returns True if *button* is pressed, else False. *button* can be a mouse button constant from `pygrafix.window.mouse`.

clear([*red*[, *green*[, *blue*]]])
Clears the whole screen to the given color.

get_screen_data([*position*[, *size*[, *buffer*]]])
Returns an ImageData object containing the current contents of the screen. You can select a sub-part of the screen with *position* and *size*. Position must have the form *(x, y)* and size *(width, height)*. The optional argument *buffer* may be *"front"* or *"back"* and defaults to *"front"*. The front buffer is what the user currently sees, the back buffer is the buffer you do your drawing on.

save_screenshot(*filename*[, *file*])
Saves a screenshot of this window into a file. If *file* is given *filename* will be used as a hint for the filetype.

get_fps()
Returns the frames per second. This value is calculated from how often flip() gets called with an algorithm that slightly smoothes out FPS changes.

pygrafix.window.**get_active_window**()
Returns the active window. The active window is the window any draw calls will target.

pygrafix.window.**get_open_windows**()
Returns a list of all opened windows.

pygrafix.window.**get_video_modes**()
Returns a list of all legal video modes in the form *(width, height, (redbits, greenbits, bluebits))*.

pygrafix.window.**get_desktop_video_mode**()
Returns the desktop video mode in the form *(width, height, (redbits, greenbits, bluebits))*.

CHAPTER

ELEVEN

PYGRAFIX.WINDOW.KEY — KEY CONSTANTS

This module contains key constants that are used with pygrafix.window.

pygrafix.window.key._0
pygrafix.window.key._1
pygrafix.window.key._2
pygrafix.window.key._3
pygrafix.window.key._4
pygrafix.window.key._5
pygrafix.window.key._6
pygrafix.window.key._7
pygrafix.window.key._8
pygrafix.window.key._9

Number constants.

pygrafix.window.key.A
pygrafix.window.key.B
pygrafix.window.key.C
pygrafix.window.key.D
pygrafix.window.key.E
pygrafix.window.key.F
pygrafix.window.key.G
pygrafix.window.key.H
pygrafix.window.key.I
pygrafix.window.key.J
pygrafix.window.key.K
pygrafix.window.key.L
pygrafix.window.key.M
pygrafix.window.key.N
pygrafix.window.key.O
pygrafix.window.key.P
pygrafix.window.key.Q
pygrafix.window.key.R
pygrafix.window.key.S
pygrafix.window.key.T
pygrafix.window.key.U
pygrafix.window.key.V
pygrafix.window.key.W
pygrafix.window.key.X

pygrafix.window.key.**Y**
pygrafix.window.key.**Z**
Alphabetic constants.

pygrafix.window.key.**F1**
pygrafix.window.key.**F2**
pygrafix.window.key.**F3**
pygrafix.window.key.**F4**
pygrafix.window.key.**F5**
pygrafix.window.key.**F6**
pygrafix.window.key.**F7**
pygrafix.window.key.**F8**
pygrafix.window.key.**F9**
pygrafix.window.key.**F10**
pygrafix.window.key.**F11**
pygrafix.window.key.**F12**
pygrafix.window.key.**F13**
pygrafix.window.key.**F14**
pygrafix.window.key.**F15**
pygrafix.window.key.**F16**
pygrafix.window.key.**F17**
pygrafix.window.key.**F18**
pygrafix.window.key.**F19**
pygrafix.window.key.**F20**
pygrafix.window.key.**F21**
pygrafix.window.key.**F22**
pygrafix.window.key.**F23**
pygrafix.window.key.**F24**
pygrafix.window.key.**F25**
F-key constants.

pygrafix.window.key.**KP_0**
pygrafix.window.key.**KP_1**
pygrafix.window.key.**KP_2**
pygrafix.window.key.**KP_3**
pygrafix.window.key.**KP_4**
pygrafix.window.key.**KP_5**
pygrafix.window.key.**KP_6**
pygrafix.window.key.**KP_7**
pygrafix.window.key.**KP_8**
pygrafix.window.key.**KP_9**
pygrafix.window.key.**KP_DECIMAL**
pygrafix.window.key.**KP_DIVIDE**
pygrafix.window.key.**KP_MULTIPLY**
pygrafix.window.key.**KP_SUBTRACT**
pygrafix.window.key.**KP_ADD**
pygrafix.window.key.**KP_ENTER**
pygrafix.window.key.**KP_EQUAL**
Keypad constants.

pygrafix.window.key.**RIGHT**
pygrafix.window.key.**LEFT**
pygrafix.window.key.**DOWN**
pygrafix.window.key.**UP**
Directional keys.

pygrafix.window.key.**COMMA**
pygrafix.window.key.**PERIOD**
pygrafix.window.key.**SEMICOLON**
pygrafix.window.key.**SLASH**
pygrafix.window.key.**BACKSLASH**
pygrafix.window.key.**APOSTROPHE**
pygrafix.window.key.**GRAVE_ACCENT**
pygrafix.window.key.**LEFT_BRACKET**
pygrafix.window.key.**RIGHT_BRACKET**
pygrafix.window.key.**MINUS**
pygrafix.window.key.**EQUAL**
Punctation and special symbol keys.

pygrafix.window.key.**SPACE**
pygrafix.window.key.**ENTER**
pygrafix.window.key.**TAB**
Whitespace keys.

pygrafix.window.key.**CAPS_LOCK**
pygrafix.window.key.**SCROLL_LOCK**
pygrafix.window.key.**NUM_LOCK**
Lock keys.

pygrafix.window.key.**ESCAPE**
pygrafix.window.key.**BACKSPACE**
pygrafix.window.key.**INSERT**
pygrafix.window.key.**DELETE**
pygrafix.window.key.**PAGE_UP**
pygrafix.window.key.**PAGE_DOWN**
pygrafix.window.key.**HOME**
pygrafix.window.key.**END**
pygrafix.window.key.**PRINT_SCREEN**
pygrafix.window.key.**PAUSE**
pygrafix.window.key.**MENU**
Miscellaneous keys.

pygrafix.window.key.**LEFT_SHIFT**
pygrafix.window.key.**LEFT_CONTROL**
pygrafix.window.key.**LEFT_ALT**
pygrafix.window.key.**LEFT_SUPER**
pygrafix.window.key.**RIGHT_SHIFT**
pygrafix.window.key.**RIGHT_CONTROL**
pygrafix.window.key.**RIGHT_ALT**
pygrafix.window.key.**RIGHT_SUPER**
Modifier keys.

pygrafix.window.key.**WORLD_1**
pygrafix.window.key.**WORLD_2**
Non-US keys #1 and #2.

CHAPTER

TWELVE

PYGRAFIX.WINDOW.MOUSE — MOUSE CONSTANTS

This module contains mouse constants that are used with pygrafix.window.

pygrafix.window.mouse.**LEFT**
pygrafix.window.mouse.**MIDDLE**
pygrafix.window.mouse.**RIGHT**
Constants for respective the left, middle and right mouse button.

pygrafix.window.mouse.**MOUSE1**
pygrafix.window.mouse.**MOUSE2**
pygrafix.window.mouse.**MOUSE3**
pygrafix.window.mouse.**MOUSE4**
pygrafix.window.mouse.**MOUSE5**
pygrafix.window.mouse.**MOUSE6**
pygrafix.window.mouse.**MOUSE7**
pygrafix.window.mouse.**MOUSE8**
Constants for mouse button 1 through 8.

CHAPTER

THIRTEEN

INDICES AND TABLES

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

PYTHON MODULE INDEX

p