moderngl_window Documentation

Release 2.3.0

Einar Forselv

Oct 24, 2020

PROGRAMMING GUIDE

A cross platform helper library for ModernGL making window creation and resource loading simple.

Note: Please report documentation improvements/issues on github. Writing documenation is diffucult and we can't do it without you. Pull requests with documentation improvements are also greatly appreciated.

CHAPTER

ONE

INSTALLATION

1.1 Installing with pip

moderngl-window is available on PyPI:

```
pip install moderngl-window
```

1.2 Optional dependencies

We try to have as few requirements as possible and instead offer optional dependencies. You can create your own window types and loaders and don't want to force installing unnecessary dependencies.

By default we install pyglet as this is the default window type as it small and pretty much work out of the box on all platforms.

Optional dependencies for loaders:

```
# Wavefront / obj loading
pip install moderngl-window[pywavefront]
# STL loading
pip install moderngl-window[trimesh]
```

Installing dependencies for window types:

```
pip install moderngl-window[PySide2]
pip install moderngl-window[pyqt5]
pip install moderngl-window[glfw]
pip install moderngl-window[PySDL2]
```

Installing optional dependencies this way should ensure a compatible version is installed.

For glfw and sdl2 windows you also need install the library itself. Thees are also available as packages on linux and homebrew on OS X. For windows the DLLs can simply be placed in the root of your project.

- GLFW : https://www.glfw.org/
- SDL2 : https://www.libsdl.org/download-2.0.php

1.3 Installing from source

```
# clone repo (optionally clone over https)
git clone git@github.com:moderngl/moderngl-window.git
cd moderngl-window

# Create your virtualenv and activate
# We assume the user knows how to work with virtualenvs

# Install moderngl-window in editable mode
pip install -e .

# Install optional dev dependencies covering all window and loader types
pip install -r requirements.txt
```

Installing the package in editable mode will make you able to run tests and examples. We highly recommend using virtualenvs.

1.4 Running examples

Assuming you installed from source you should be able to run the examples in the *examples* directory directly after installing the dev requirements in the root of the project:

```
pip install -r requirements.txt
```

1.5 Running tests

Install test requirements:

```
pip install -r tests/requirements.txt
```

Run tests with `tox`:

```
# Run for specific environment
tox -e py35
tox -e py36
tox -e py37

# pep8 run
tox -e pep8

# Run all environments
tox
```

CHAPTER

TWO

BASIC USAGE (WINDOWCONFIG)

Note: This section is only relevant when using *WindowConfig*. Go to the Custom Usage section if you provide your own window and context or want more control.

Using the *WindowConfig* interface is the simplest way to start with moderngl-window. This can work for smaller projects and implies that this library provides the window and moderngl context.

The API docs for this class alone should cover a lot of ground, but we'll go through the basics here.

2.1 Basic example

The *WindowConfig* is simply a class you extend to customize/implement initialization, window parameters, rendering code, keyboard input, mouse input and access simpler shortcut methods for loading resources.

```
import moderngl_window as mglw

class Test(mglw.WindowConfig):
    gl_version = (3, 3)
    window_size = (1920, 1080)

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        # Do initialization here
        self.prog = self.ctx.program(...)
        self.vao = self.ctx.vertex_array(...)
        self.texture = self.ctx.texture(self.wnd.size, 4)

    def render(self, time, frametime):
        # This method is called every frame
        self.vao.render()

# Blocking call entering rendering/event loop
mglw.run_window_config(Test)
```

The *WindowConfig* instance will by default receive three external instances in `__init__` that can be accessed later with `self`.

- `self.ctx`: The `moderngl.Context` created by the configured window type
- `self.wnd`: The window instance
- `self.timer`: The *moderngl_window.timers.clock.Timer* instance to control the current time (Values passed into `render`)

2.2 Resource loading

The *WindowConfig* class has built in shortcuts to the resource loading system.

```
self.load_texture('background.png')
self.load_texture_array('tiles.png', layers=16)
self.load_program('myprogram.glsl')
self.load_text('textfile.txt')
self.load_json('config.json')
self.load_binary('data.bin')
self.load_scene('cube.obj')
self.load_scene('city.gltf')
```

All paths used in resource loading are relative to an absolute path provided in the *WindowConfig*.

```
from pathlib import Path

class Test(mglw.WindowConfig):
    resource_dir = (Path(__file__).parent / 'resources').resolve()
```

If you need more than one search path for your resources, the *moderngl_window.resources* module has methods for this.

2.3 Generic events and window types

The *WindowConfig* interface depends on the built in window types or a self-provided window implementation of `BaseWindow`. These window implementations convert window, key and mouse events into a unified system so the user can switch between different window types without altering the code.

Window libraries are not perfect and may at times work sub-optimally on some platforms. They might also have different performance profiles. The ability to switch between window types by just changing a config value can be an advantage.

You can change what window class is used by passing in the `--window` option. Optionally you can modify the *WINDOW* attribute directly.

2.4 Command line arguments

The *run_window_config()* method also reads arguments from `sys.argv` making the user able to override config values in the class.

Example:

```
python test.py --window glfw --fullscreen --vsync --samples 16 --cursor false --size␣
↪800x600
```

See code for *moderngl_window.parse_args()* for more details.

2.5 Window events

```
def resize(self, width: int, height: int):
    print("Window was resized. buffer size is {} x {}".format(width, height))

def close(self):
    print("The window is closing")

def iconify(self, iconify: bool):
    print("Window was iconified:", iconify)
```

2.6 Keyboard input

Implement the `key_event` and `unicode_char_entered` method to handle key events.

```
def key_event(self, key, action, modifiers):
    # Key presses
    if action == self.wnd.keys.ACTION_PRESS:
        if key == self.wnd.keys.SPACE:
            print("SPACE key was pressed")

        # Using modifiers (shift and ctrl)

        if key == self.wnd.keys.Z and modifiers.shift:
            print("Shift + Z was pressed")

        if key == self.wnd.keys.Z and modifiers.ctrl:
            print("ctrl + Z was pressed")

    # Key releases
    elif action == self.wnd.keys.ACTION_RELEASE:
        if key == self.wnd.keys.SPACE:
            print("SPACE key was released")

def unicode_char_entered(self, char: str):
    print('character entered:', char)
```

2.7 Mouse input

Implement the `mouse_*` methods to handle mouse input.

```
def mouse_position_event(self, x, y, dx, dy):
    print("Mouse position:", x, y, dx, dy)

def mouse_drag_event(self, x, y, dx, dy):
    print("Mouse drag:", x, y, dx, dy)

def mouse_scroll_event(self, x_offset: float, y_offset: float):
    prtin('Mouse wheel:', x_offset, y_offset)

def mouse_press_event(self, x, y, button):
    print("Mouse button {} pressed at {}, {}".format(button, x, y))
```

(continues on next page)

(continued from previous page)

```
def mouse_release_event(self, x: int, y: int, button: int):
    print("Mouse button {} released at {}, {}".format(button, x, y))
```

CHAPTER

THREE

WINDOW GUIDE

We support the following window types:

- pyglet
- glfw
- sdl2
- pyside2
- pyqt5
- headless

3.1 Using built in window types

The library provides shortcuts for window creation in the *moderngl_window* module that will also handle context activation.

The *moderngl_window.conf.Settings* instance has sane default parameters for a window. See the *WINDOW* attribute.

```
import moderngl_window
from moderngl_window.conf import settings

settings.WINDOW['class'] = 'moderngl_window.context.glfw.Window'
settings.WINDOW['gl_version'] = (4, 1)
# ... etc ...

# Creates the window instance and activates its context
window = moderngl_window.create_window_from_settings()
```

There are more sane ways to apply different configuration values through convenient methods in the *Settings* class.

Window classes can of course also be instantiated manually if preferred, but this can generate a bit of extra work.

```
import moderngl_window

window_str = 'moderngl_window.context.pyglet.Window'
window_cls = moderngl_window.get_window_cls(window_str)
window = window_cls(
    title="My Window",
    gl_version=(4, 1),
    size=(1920, 1080),
```

(continues on next page)

(continued from previous page)

```
    ...,
)
moderngl_window.activate_context(ctx=window.ctx)
```

You could also simply import the class directory and instantiate it, but that defeats the purpose of trying to be independent of a specific window library.

The rendering loop for built in windows is simple:

```
while not window.is_closing:
    window.clear()
    # Render stuff here
    window.swap_buffers()
```

The `swap_buffers` method is important as it also pulls new input events for the next frame.

CHAPTER

FOUR

OLD GUIDE

When not using a *WindowConfig* instance, there are a few simple steps to get started.

4.1 Register the moderngl.Context

When not using the built in window types you need to at least tell moderngl_window what your `moderngl.Context` is.

```
import moderngl
import moderngl_window

# Somewhere in your application a standalone or normal context is created
ctx = moderngl.create_standalone_context(require=330)
ctx = moderngl.create_context(require=330)

# Make sure you activate this context
moderngl_window.activate_context(ctx=ctx)
```

If there is no context activated the library will raise an exception when doing operations that requires one, such as texture and scene loading.

When using the built in window types the context activation is normally done for you on creation.

4.2 Register resource directories

The resource loading system uses relative paths. These paths are relative to one or multiple directories we registered in the resource system.

The *moderngl_window.resources* module has methods for this.

```
from pathlib import Path
from moderngl_window import resources

# We recommend using pathlib
resources.register_dir(Path('absolute/path/to/resource/dir').resolve())
# .. but strings also works
resources.register_dir('absolute/path/to/resource/dir')
```

These need to be absolute paths or an exception is raised. You can register as many paths as you want. The resource system will simply look for the file in every registered directory in the order they were added until it finds a match.

This library also supports separate search directories for shader programs, textures, scenes and various data files.

CHAPTER

FIVE

EVENT GUIDE

Work in progress

CHAPTER

SIX

THE RESOURCE SYSTEM

6.1 Resource types

The resource system has four different resource types/categories it can load.

- **Programs** : Shader programs (vertex, geometry, fragment, tessellation, compute)
- **Textures** : Textures of all different variations
- **Scenes**: Wavefront, GLTF2 and STL scenes/objects
- **Data**: A generic "lane" for everything else

Each of these resource categories have separate search directories, one or multiple loader classes and a *ResourceDescription* class we use to describe the resource we are loading with all its parameters.

6.2 Resource paths

Resources are loaded using relative paths. These paths are relative to one or multiple search directories we register using the *resources* module.

For simple usage were we have one or multiple resource directories with mixed resource types (programs, textures etc.) we can use the simplified version, *register_dir()*.

```
from pathlib import Path
from moderngl_window import resources

# pathlib.Path (recommended)
resources.register_dir(Path('absoulte/path/using/pathlib'))

# Strings and/or os.path
resources.register_dir('absolute/string/path')
```

A resource finder system will scan through the registered directories in the order they were added loading the first resource found.

For more advanced usage were resources of different types are separated we can register resource type specific search directories:

- *register_program_dir()*
- *register_texture_dir()*
- *register_scene_dir()*
- *register_data_dir()*

This can be handy when dealing with larger quantities of files. These search directories are stored in the *Settings* instance and can for example be temporarily altered if needed. This means you can separate local and global resources in more complex situations. It could even be used to support themes by promoting a theme directory overriding global/default resources or some default theme directory.

6.3 Resource descriptions

Resource descriptions are basically just classes acting as bags of attributes describing the resource we are requesting. We have four standard classes.

- *ProgramDescription*
- *TextureDescription*
- *SceneDescription*
- *DataDescription*

Example:

```
from moderngl_window.meta import TextureDescription

# We are aiming to load wood.png horizontally flipped
# with generated mipmaps and high anisotropic filtering.
TextureDescription(
    path='wood.png',
    flip=True,
    mipmap=True,
    anisotropy=16.0,
)
```

New resource description classes can be created by extending the base *ResourceDescription* class. This is not uncommon when for example making a new loader class.

6.4 Loading resources

Now that we know about the different resource categories, search paths and resource descriptions, we're ready to actually load something.

Loading resources can in some situation be a bit verbose, but you can simplify by wrapping them in your own functions if needed. The *WindowConfig* class is already doing this and can be used as a reference.

```
from moderngl_window.resources import (
    textures,
    programs,
    scenes,
    data,
)
from moderngl_window.meta import (
    TextureDescription,
    ProgramDescription,
    SceneDescription,
    DataDescription,
)
```

6.4.1 Textures

```
# Load a 2D texture
texture = textures.load(TextureDescription(path='wood.png'))

# Load wood.png horizontally flipped with generated mipmaps and high anisotropic␣
↪filtering.
textures.load(TextureDescription(path='wood.png', flip=True, mipmap=True,␣
↪anisotropy=16.0))

# Load a texture array containing 10 vertically stacked tile textures
textures.load(TextureDescription(path='tiles.png', layers=10, mipmap=True,␣
↪anisotrpy=8.0))
```

6.4.2 Programs

```
# Load a shader program in a single glsl file
program = programs.load(ProgramDescription(path='fun.glsl'))

# Load a shader program from multiple glsl files
program = programs.load(
    ProgramDescription(
        vertex_shader='sphere_vert.glsl',
        geometry_shader='sphere_geo.glsl',
        fragment_shader='sphere_fs.glsl',
    )
)
```

6.4.3 Scenes

```
# Load a GLTF2 scene
scene = scenes.load(SceneDescription(path="city.gltf"))

# Load a wavefront scene
scene = scenes.load(SceneDescription(path="earth.obj"))

# Load an STL file
scene = scenes.load(SceneDescription(path="apollo_landing_site_18.stl"))
```

6.4.4 Data

```
# Load text file
text = data.load(DataDescription(path='notes.txt'))

# Load config file as a dict
config_dict = data.load(DataDescription(path='config.json'))

# Load binary data
data = data.load(DataDescription(path='data.bin', kind='binary))
```

For more information about supported parameters see the api documentation.

CHAPTER

SEVEN

MODERNGL_WINDOW

General helper functions aiding in the boostrapping of this library.

moderngl_window.**setup_basic_logging**(*level: int*)
Set up basic logging

Parameters **level** (*int*) – The log level

moderngl_window.**activate_context**(*window: moderngl_window.context.base.window.BaseWindow = None*, *ctx: moderngl.context.Context = None*)
Register the active window and context. If only a window is supplied the context is taken from the window. Only a context can also be passed in.

Keyword Arguments

- **window** (window) – The window to activate
- **ctx** (*moderngl.Context*) – The moderngl context to activate

moderngl_window.**window**()
Obtain the active window

moderngl_window.**ctx**()
Obtain the active context

moderngl_window.**get_window_cls**(*window: str = None*) → Type[moderngl_window.context.base.window.BaseWindow]
Attempt to obtain a window class using the full dotted python path. This can be used to import custom or modified window classes.

Parameters **window** (*str*) – Name of the window

Returns A reference to the requested window class. Raises exception if not found.

moderngl_window.**get_local_window_cls**(*window: str = None*) → Type[moderngl_window.context.base.window.BaseWindow]
Attempt to obtain a window class in the moderngl_window package using short window names such as pyglet or glfw.

Parameters **window** (*str*) – Name of the window

Returns A reference to the requested window class. Raises exception if not found.

moderngl_window.**find_window_classes**() → List[str]
Find available window packages

Returns A list of available window packages

moderngl_window.**create_window_from_settings**() → moderngl_window.context.base.window.BaseWindow

Creates a window using configured values in *moderngl_window.conf.Settings.WINDOW*. This will also activate the window/context.

Returns The Window instance

moderngl_window.**run_window_config**(*config_cls: moderngl_window.context.base.window.WindowConfig*, *timer=None*, *args=None*) → None

Run an WindowConfig entering a blocking main loop

Parameters **config_cls** – The WindowConfig class to render

Keyword Arguments

- **timer** – A custom timer instance
- **args** – Override sys.args

moderngl_window.**create_parser**()

Create an argparser parsing the standard arguments for WindowConfig

moderngl_window.**parse_args**(*args=None*, *parser=None*)

Parse arguments from sys.argv

Passing in your own argparser can be user to extend the parser.

Keyword Arguments

- **args** – override for sys.argv
- **parser** – Supply your own argparser instance

CHAPTER

EIGHT

MODERNGL_WINDOW.CONF.SETTINGS

moderngl_window.conf.**Settings**
Bag of settings values. New attributes can be freely added runtime. Various apply* methods are supplied so the user have full control over how settings values are initialized. This is especially useful for more custom usage. And instance of the *Settings* class is created when the *conf* module is imported.

Attribute names must currently be in upper case to be recognized.

Some examples of usage:

```
from moderngl_window.conf import settings

# Mandatory settings values
try:
    value = settings.VALUE
except KeyError:
    raise ValueError("This settings value is required")

# Fallback in code
value = getattr(settings, 'VALUE', 'default_value')

# Pretty printed string representation for easy inspection
print(settings)
```

8.1 Methods

Settings.**__init__**()
Initialize settings with default values

Settings.**apply_default_settings**() → None
Apply keys and values from the default settings module located in this package. This is to ensure we always have the minimal settings for the system to run.

If replacing or customizing the settings class you must always apply default settings to ensure compatibility when new settings are added.

Settings.**apply_settings_from_env**() → None
Apply settings from MODERNGL_WINDOW_SETTINGS_MODULE environment variable. If the environment variable is undefined no action will be taken. Normally this would be used to easily be able to switch between different configuration by setting env vars before executing the program.

Example:

```
import os
from moderngl_window.conf import settings

os.environ['MODERNGL_WINDOW_SETTINGS_MODULE'] = 'python.path.to.module'
settings.apply_settings_from_env()
```

Raises **ImproperlyConfigured if the module was not found** –

Settings.**apply_from_module_name**(*settings_module_name: str*) → None
Apply settings from a python module by supplying the full pythonpath to the module.

Parameters **settings_module_name** (*str*) – Full python path to the module

Raises **ImproperlyConfigured if the module was not found** –

Settings.**apply_from_dict**(*data: dict*) → None
Apply settings values from a dictionary

Example:

```
>> from moderngl_window.conf import settings
>> settings.apply_dict({'SOME_VALUE': 1})
>> settings.SOME_VALUE
1
```

Settings.**apply_from_module**(*module: module*) → None
Apply settings values from a python module

Example:

```
my_settings.py module containing the following line:
SOME_VALUE = 1

>> from moderngl_window.conf import settings
>> import my_settings
>> settings.apply_module(my_settings)
>> settings.SOME_VALUE
1
```

Settings.**apply_from_cls**(*cls*) → None
Apply settings values from a class namespace

Example:

```
>> from moderngl_window.conf import settings
>> class MySettings:
>>     SOME_VALUE = 1
>>
>> settings.apply(MySettings)
>> settings.SOME_VALUE
1
```

Settings.**apply_from_iterable**(*iterable: Union[collections.abc.Iterable, generator]*) → None
Apply (key, value) pairs from an interable or generator

Settings.**to_dict**()
Create a dict representation of the settings Only uppercase attributes are included

Returns dict representation

Return type dict

8.2 Attributes

`Settings.WINDOW`
Window/screen properties. Most importantly the `class` attribute decides what class should be used to handle the window.

```
# Default values
WINDOW = {
    "gl_version": (3, 3),
    "class": "moderngl_window.context.pyglet.Window",
    "size": (1280, 720),
    "aspect_ratio": 16 / 9,
    "fullscreen": False,
    "resizable": True,
    "title": "ModernGL Window",
    "vsync": True,
    "cursor": True,
    "samples": 0,
}
```

Other Properties:

- `gl_version`: The minimum required major/minor OpenGL version
- `size`: The window size to open.
- `aspect_ratio` is the enforced aspect ratio of the viewport.
- `fullscreen`: True if you want to create a context in fullscreen mode
- `resizable`: If the window should be resizable. This only applies in windowed mode.
- `vsync`: Only render one frame per screen refresh
- `title`: The visible title on the window in windowed mode
- `cursor`: Should the mouse cursor be visible on the screen? Disabling this is also useful in windowed mode when controlling the camera on some platforms as moving the mouse outside the window can cause issues.
- `Samples`: Number if samples used in multisampling. Values above 1 enables multisampling.

The created window frame buffer will by default use:

- RGBA8 (32 bit per pixel)
- 24 bit depth buffer
- Double buffering
- color and depth buffer is cleared for every frame

`Settings.SCREENSHOT_PATH`
Absolute path to the directory screenshots will be saved by the screenshot module. Screenshots will end up in the project root of not defined. If a path is configured, the directory will be auto-created.

`Settings.PROGRAM_FINDERS`
Finder classes for locating programs/shaders.

```
# Default values
PROGRAM_FINDERS = [
    "moderngl_window.finders.program.FileSystemFinder",
]
```

Settings.**TEXTURE_FINDERS**
Finder classes for locating textures.

```
# Default values
TEXTURE_FINDERS = [
    "moderngl_window.finders.texture.FileSystemFinder",
]
```

Settings.**SCENE_FINDERS**
Finder classes for locating scenes.

```
# Default values
SCENE_FINDERS = [
    "moderngl_window.finders.scene.FileSystemFinder",
]
```

Settings.**DATA_FINDERS**
Finder classes for locating data files.

```
# Default values
DATA_FINDERS = [
    "moderngl_window.finders.data.FileSystemFinder",
]
```

Settings.**PROGRAM_DIRS**
Lists of *str* or *pathlib.Path* used by `FileSystemFinder` to looks for programs/shaders.

Settings.**TEXTURE_DIRS**
Lists of *str* or *pathlib.Path* used by `FileSystemFinder` to looks for textures.

Settings.**SCENE_DIRS**
Lists of *str* or *pathlib.Path* used by `FileSystemFinder` to looks for scenes (obj, gltf, stl etc).

Settings.**DATA_DIRS**
Lists of *str* or *pathlib.Path* used by `FileSystemFinder` to looks for data files.

Settings.**PROGRAM_LOADERS**
Classes responsible for loading programs/shaders.

```
# Default values
PROGRAM_LOADERS = [
    'moderngl_window.loaders.program.single.Loader',
    'moderngl_window.loaders.program.separate.Loader',
]
```

Settings.**TEXTURE_LOADERS**
Classes responsible for loading textures.

```
# Default values
TEXTURE_LOADERS = [
    'moderngl_window.loaders.texture.t2d.Loader',
    'moderngl_window.loaders.texture.array.Loader',
]
```

Settings.**SCENE_LOADERS**
Classes responsible for loading scenes.

```
# Default values
SCENE_LOADERS = [
    "moderngl_window.loaders.scene.gltf.GLTF2",
    "moderngl_window.loaders.scene.wavefront.ObjLoader",
    "moderngl_window.loaders.scene.stl_loader.STLLoader",
]
```

Settings.**DATA_LOADERS**
Classes responsible for loading data files.

```
# Default values
DATA_LOADERS = [
    'moderngl_window.loaders.data.binary.Loader',
    'moderngl_window.loaders.data.text.Loader',
    'moderngl_window.loaders.data.json.Loader',
]
```

CHAPTER

NINE

MODERNGL_WINDOW.SCREENSHOT

moderngl_window.screenshot.**create**(*source: Union[moderngl.framebuffer.Framebuffer, moderngl.texture.Texture], file_format='png', name: str = None, mode='RGB', alignment=1*)

Create a screenshot from a `moderngl.Framebuffer` or `moderngl.Texture`. The screenshot will be written to *SCREENSHOT_PATH* if set or `cwd` or an absolute path can be used.

Parameters

- **source** – The framebuffer or texture to screenshot
- **file_format** (*str*) – formats supported by PIL (png, jpeg etc)
- **name** (*str*) – Optional file name with relative or absolute path
- **mode** (*str*) – Components/mode to use
- **alignment** (*int*) – Buffer alignment

CHAPTER

TEN

MODERNGL_WINDOW.CONTEXT

10.1 base.window.WindowConfig

moderngl_window.context.base.window.**WindowConfig**

Creating a `WindowConfig` instance is the simplest interface this library provides to open and window, handle inputs and provide simple shortcut method for loading basic resources. It's appropriate for projects with basic needs.

Example:

```
import moderngl_window

class MyConfig(moderngl_window.WindowConfig):
    gl_version = (3, 3)
    window_size = (1920, 1080)
    aspect_ratio = 16 / 9
    title = "My Config"
    resizable = False
    samples = 8

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        # Do other initialization here

    def render(self, time: float, frametime: float):
        # Render stuff here with ModernGL

    def resize(self, width: int, height: int):
        print("Window was resized. buffer size is {} x {}".format(width, height))

    def mouse_position_event(self, x, y, dx, dy):
        print("Mouse position:", x, y)

    def mouse_press_event(self, x, y, button):
        print("Mouse button {} pressed at {}, {}".format(button, x, y))

    def mouse_release_event(self, x: int, y: int, button: int):
        print("Mouse button {} released at {}, {}".format(button, x, y))

    def key_event(self, key, action, modifiers):
        print(key, action, modifiers)
```

10.1.1 Methods

`WindowConfig.__init__`(*ctx: moderngl.context.Context = None*, *wnd: moderngl_window.context.base.window.BaseWindow = None*, *timer: moderngl_window.timers.base.BaseTimer = None*, ***kwargs*)

Initialize the window config

Keyword Arguments

- **ctx** (*moderngl.Context*) – The moderngl context
- **wnd** – The window instance
- **timer** – The timer instance

classmethod `WindowConfig.run`()

Shortcut for running a `WindowConfig`.

This executes the following code:

```
import moderngl_window
moderngl_window.run_window_config(cls)
```

`WindowConfig.render`(*time: float*, *frame_time: float*)

Renders the assigned effect

Parameters

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

`WindowConfig.resize`(*width: int*, *height: int*)

Called every time the window is resized in case the we need to do internal adjustments.

Parameters

- **width** (*int*) – width in buffer size (not window size)
- **height** (*int*) – height in buffer size (not window size)

`WindowConfig.close`()

Called when the window is closed

classmethod `WindowConfig.add_arguments`(*parser: argparse.ArgumentParser*)

Add arguments to default argument parser. Add arguments using `add_argument(..)`.

Parameters **parser** (*ArgumentParser*) – The default argument parser.

`WindowConfig.key_event`(*key: Any*, *action: Any*, *modifiers: moderngl_window.context.base.keys.KeyModifiers*)

Called for every key press and release. Depending on the library used, key events may trigger repeating events during the pressed duration based on the configured key repeat on the users operating system.

Parameters

- **key** – The key that was press. Compare with self.wnd.keys.
- **action** – self.wnd.keys.ACTION_PRESS or ACTION_RELEASE
- **modifiers** – Modifier state for shift, ctrl and alt

`WindowConfig.mouse_position_event`(*x: int*, *y: int*, *dx: int*, *dy: int*)

Reports the current mouse cursor position in the window

Parameters

- **x** (*int*) – X postion of the mouse cursor
- **y** (*int*) – Y position of the mouse cursor
- **dx** (*int*) – X delta postion
- **dy** (*int*) – Y delta position

WindowConfig.**mouse_press_event**(*x: int*, *y: int*, *button: int*)
Called when a mouse button in pressed

Parameters

- **x** (*int*) – X position the press ocurred
- **y** (*int*) – Y position the press ocurred
- **button** (*int*) – 1 = Left button, 2 = right button

WindowConfig.**mouse_release_event**(*x: int*, *y: int*, *button: int*)
Called when a mouse button in released

Parameters

- **x** (*int*) – X position the release ocurred
- **y** (*int*) – Y position the release ocurred
- **button** (*int*) – 1 = Left button, 2 = right button

WindowConfig.**mouse_drag_event**(*x: int*, *y: int*, *dx: int*, *dy: int*)
Called when the mouse is moved while a button is pressed.

Parameters

- **x** (*int*) – X postion of the mouse cursor
- **y** (*int*) – Y position of the mouse cursor
- **dx** (*int*) – X delta postion
- **dy** (*int*) – Y delta position

WindowConfig.**mouse_scroll_event**(*x_offset: float*, *y_offset: float*)
Called when the mouse wheel is scrolled.

Some input devices also support horizontal scrolling, but vertical scrolling is fairly universal.

Parameters

- **x_offset** (*int*) – X scroll offset
- **y_offset** (*int*) – Y scroll offset

WindowConfig.**unicode_char_entered**(*char: str*)
Called when the user entered a unicode character.

Parameters **char** (*str*) – The character entered

WindowConfig.**load_texture_2d**(*path: str*, *flip=True*, *flip_x=False*, *flip_y=True*, *mipmap=False*, *mipmap_levels: Tuple[int, int] = None*, *anisotropy=1.0*, ***kwargs*)
→ moderngl.texture.Texture
Loads a 2D texture

Parameters **path** (*str*) – Path to the texture relative to search directories

Keyword Arguments

- **flip** (*boolean*) – (Use `flip_y) Flip the image vertically (top to bottom)

- **flip_x** (*boolean*) – Flip the image horizontally (left to right)
- **flip_y** (*boolean*) – Flip the image vertically (top to bottom)
- **mipmap** (*bool*) – Generate mipmaps. Will generate max possible levels unless *mipmap_levels* is defined.
- **mipmap_levels** (*tuple*) – (base, max_level) controlling mipmap generation. When defined the *mipmap* parameter is automatically *True*
- **anisotropy** (*float*) – Number of samples for anisotropic filtering
- ****kwargs** – Additional parameters to TextureDescription

Returns Texture instance

Return type moderngl.Texture

WindowConfig.**load_texture_array**(*path: str*, *layers: int = 0*, *flip=True*, *mipmap=False*, *mipmap_levels: Tuple[int, int] = None*, *anisotropy=1.0*, ***kwargs*) → moderngl.texture_array.TextureArray

Loads a texture array.

Parameters **path** (*str*) – Path to the texture relative to search directories

Keyword Arguments

- **layers** (*int*) – How many layers to split the texture into vertically
- **flip** (*boolean*) – Flip the image horizontally
- **mipmap** (*bool*) – Generate mipmaps. Will generate max possible levels unless *mipmap_levels* is defined.
- **mipmap_levels** (*tuple*) – (base, max_level) controlling mipmap generation. When defined the *mipmap* parameter is automatically *True*
- **anisotropy** (*float*) – Number of samples for anisotropic filtering
- ****kwargs** – Additional parameters to TextureDescription

Returns The texture instance

Return type moderngl.TextureArray

WindowConfig.**load_texture_cube**(*pos_x: str = None*, *pos_y: str = None*, *pos_z: str = None*, *neg_x: str = None*, *neg_y: str = None*, *neg_z: str = None*, *flip=False*, *flip_x=False*, *flip_y=False*, *mipmap=False*, *mipmap_levels: Tuple[int, int] = None*, *anisotropy=1.0*, ***kwargs*) → moderngl.texture_cube.TextureCube

Loads a texture cube.

Keyword Arguments

- **pos_x** (*str*) – Path to texture representing positive x face
- **pos_y** (*str*) – Path to texture representing positive y face
- **pos_z** (*str*) – Path to texture representing positive z face
- **neg_x** (*str*) – Path to texture representing negative x face
- **neg_y** (*str*) – Path to texture representing negative y face
- **neg_z** (*str*) – Path to texture representing negative z face
- **flip** (*boolean*) – (Use `flip_y`)Flip the image vertically (top to bottom)

- **flip_x** (*boolean*) – Flip the image horizontally (left to right)
- **flip_y** (*boolean*) – Flip the image vertically (top to bottom)
- **mipmap** (*bool*) – Generate mipmaps. Will generate max possible levels unless *mipmap_levels* is defined.
- **mipmap_levels** (*tuple*) – (base, max_level) controlling mipmap generation. When defined the *mipmap* parameter is automatically *True*
- **anisotropy** (*float*) – Number of samples for anisotropic filtering
- ****kwargs** – Additional parameters to TextureDescription

Returns Texture instance

Return type moderngl.TextureCube

WindowConfig.**load_program**(*path=None*, *vertex_shader=None*, *geometry_shader=None*, *fragment_shader=None*, *tess_control_shader=None*, *tess_evaluation_shader=None*, *defines: dict = None*) → moderngl.program.Program

Loads a shader program.

Note that *path* should only be used if all shaders are defined in the same glsl file separated by defines.

Keyword Arguments

- **path** (*str*) – Path to a single glsl file
- **vertex_shader** (*str*) – Path to vertex shader
- **geometry_shader** (*str*) – Path to geometry shader
- **fragment_shader** (*str*) – Path to fragment shader
- **tess_control_shader** (*str*) – Path to tessellation control shader
- **tess_evaluation_shader** (*str*) – Path to tessellation eval shader
- **defines** (*dict*) – `#define` values to replace in the shader source. Example: `{'VALUE1': 10, 'VALUE2': '3.1415'}`.

Returns The program instance

Return type moderngl.Program

WindowConfig.**load_compute_shader**(*path*, *defines: dict = None*, ***kwargs*) → moderngl.compute_shader.ComputeShader

Loads a compute shader.

Parameters

- **path** (*str*) – Path to a single glsl file
- **defines** (*dict*) – `#define` values to replace in the shader source. Example: `{'VALUE1': 10, 'VALUE2': '3.1415'}`.

Returns The compute shader

Return type moderngl.ComputeShader

WindowConfig.**load_text**(*path: str*, ***kwargs*) → str

Load a text file.

Parameters

- **path** (*str*) – Path to the file relative to search directories

- ****kwargs** – Additional parameters to DataDescription

Returns Contents of the text file

Return type str

WindowConfig.**load_json**(*path: str*, ***kwargs*) → dict
Load a json file

Parameters

- **path** (*str*) – Path to the file relative to search directories
- ****kwargs** – Additional parameters to DataDescription

Returns Contents of the json file

Return type dict

WindowConfig.**load_binary**(*path: str*, ***kwargs*) → bytes
Load a file in binary mode.

Parameters

- **path** (*str*) – Path to the file relative to search directories
- ****kwargs** – Additional parameters to DataDescription

Returns The byte data of the file

Return type bytes

WindowConfig.**load_scene**(*path: str*, *cache=False*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*, *kind=None*, ***kwargs*) → moderngl_window.scene.scene.Scene
Loads a scene.

Keyword Arguments

- **path** (*str*) – Path to the file relative to search directories
- **cache** (*str*) – Use the loader caching system if present
- **attr_names** (*AttributeNames*) – Attrib name config
- **kind** (*str*) – Override loader kind
- ****kwargs** – Additional parameters to SceneDescription

Returns The scene instance

Return type Scene

10.1.2 Attributes

WindowConfig.**window_size**
Size of the window.

```
# Default value
window_size = (1280, 720)
```

WindowConfig.**vsync**
Enable or disable vsync.

```
# Default value
vsync = True
```

WindowConfig.**fullscreen**
Open the window in fullscreen mode.

```
# Default value
fullscreen = False
```

WindowConfig.**resizable**
Determines of the window should be resizable

```
# Default value
resizable = True
```

WindowConfig.**gl_version**
The minimum required OpenGL version required

```
# Default value
gl_version = (3, 3)
```

WindowConfig.**title**
Title of the window

```
# Default value
title = "Example"
```

WindowConfig.**aspect_ratio**
The enforced aspect ratio of the viewport. When specified back borders will be calculated both vertically and horizontally if needed.

This property can be set to `None` to disable the fixed viewport system.

```
# Default value
aspect_ratio = 16 / 9
```

WindowConfig.**cursor**
Determines if the mouse cursor should be visible inside the window. If enabled on some platforms

```
# Default value
cursor = True
```

WindowConfig.**clear_color**
The color the active framebuffer is cleared with. This attribute is expected to be in the form of `(r, g, b, a)` in the range `[0.0, 1.0]`

If the value is *None* the screen will not be cleared every frame.

```
# Default value
clear_color = (0.0, 0.0, 0.0, 0.0)
# Disable screen clearing
clear_color = None
```

WindowConfig.**samples**
Number of samples to use in multisampling.

```
# Default value
samples = 4
```

WindowConfig.**resource_dir**
Absolute path to your resource directory containing textures, scenes, shaders/programs or data files. The `load_` methods in this class will look for resources in this path. This attribute can be a `str` or a `pathlib.Path`.

```
# Default value
resource_dir = None
```

WindowConfig.**log_level**
Sets the log level for this library using the standard *logging* module.

```
# Default value
log_level = logging.INFO
```

WindowConfig.**argv**
The parsed command line arguments.

10.2 base.BaseWindow

10.2.1 Methods

BaseWindow.**__init__**(*title='ModernGL', gl_version=(3, 3), size=(1280, 720), resizable=True, fullscreen=False, vsync=True, aspect_ratio: float = None, samples=0, cursor=True, **kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

BaseWindow.**init_mgl_context**() → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's `gl_version`.

Keyword Arguments **ctx** – An optional custom ModernGL context

BaseWindow.**is_key_pressed**(*key*) → bool
Returns: The press state of a key

BaseWindow.**set_icon**(*icon_path: str*) → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

BaseWindow.**close**() → None
Signal for the window to close

BaseWindow.**use**()
Bind the window's framebuffer

BaseWindow.**clear**(*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*)
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

BaseWindow.**render**(*time=0.0*, *frame_time=0.0*) → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

BaseWindow.**swap_buffers**() → None
Library specific buffer swap method. Must be overridden.

BaseWindow.**resize**(*width*, *height*) → None
Should be called every time window is resized so the example can adapt to the new size if needed

BaseWindow.**destroy**() → None
A library specific destroy method is required

BaseWindow.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

BaseWindow.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

BaseWindow.**print_context_info**()
Prints moderngl context info.

10.2.2 Attributes

BaseWindow.**name = None**
Name of the window. For example pyglet, glfw

BaseWindow.**keys**
Window specific key constants

BaseWindow.**ctx**
The ModernGL context for the window

Type moderngl.Context

BaseWindow.**fbo**
The default framebuffer

Type moderngl.Framebuffer

BaseWindow.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

BaseWindow.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

BaseWindow.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

BaseWindow.**width**
The current window width

Type int

BaseWindow.**height**
The current window height

Type int

BaseWindow.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

BaseWindow.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

BaseWindow.**buffer_width**
the current window buffer width

Type int

BaseWindow.**buffer_height**
the current window buffer height

Type int

BaseWindow.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

BaseWindow.**pixel_ratio**
The framebuffer/window size ratio

Type float

BaseWindow.**viewport**
current window viewport

Type Tuple[int, int, int, int]

BaseWindow.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

BaseWindow.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

BaseWindow.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

BaseWindow.**frames**
Number of frames rendered

Type int

BaseWindow.**resizable**
Window is resizable

Type bool

BaseWindow.**fullscreen**
Window is in fullscreen mode

Type bool

BaseWindow.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

BaseWindow.**vsync**
vertical sync enabled/disabled

Type bool

BaseWindow.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

BaseWindow.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

BaseWindow.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

BaseWindow.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

BaseWindow.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

BaseWindow.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

BaseWindow.**resize_func**
Get or set the resize callable

Type callable

BaseWindow.**close_func**
Get or set the close callable

Type callable

BaseWindow.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

BaseWindow.**key_event_func**
Get or set the key_event callable

Type callable

BaseWindow.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

BaseWindow.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

BaseWindow.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

BaseWindow.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

BaseWindow.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

BaseWindow.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

BaseWindow.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

BaseWindow.**is_closing**
Is the window about to close?

Type bool

BaseWindow.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**
Mouse button enum

BaseWindow.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

BaseWindow.**modifiers**
(KeyModifiers) The current keyboard modifiers

BaseWindow.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.3 glfw.Window

10.3.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync

- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

`Window.`**`init_mgl_context`**`()` → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's `gl_version`.

Keyword Arguments **ctx** – An optional custom ModernGL context

`Window.`**`is_key_pressed`**`(`*key*`)` → bool
Returns: The press state of a key

`Window.`**`set_icon`**`(`*icon_path: str*`)` → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

`Window.`**`close`**`()` → None
Suggest to glfw the window should be closed soon

`Window.`**`use`**`()`
Bind the window's framebuffer

`Window.`**`clear`**`(`*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*`)`
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

`Window.`**`render`**`(`*time=0.0*, *frame_time=0.0*`)` → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

`Window.`**`swap_buffers`**`()`
Swap buffers, increment frame counter and pull events

`Window.`**`resize`**`(`*width*, *height*`)` → None
Should be called every time window is resized so the example can adapt to the new size if needed

`Window.`**`destroy`**`()`
Gracefully terminate GLFW

`Window.`**`set_default_viewport`**`()` → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**print_context_info**()
Prints moderngl context info.

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

10.3.2 Attributes

Window.**name = 'glfw'**
Name of the window

Window.**keys**
GLFW specific key constants

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**close_func**
Get or set the close callable

Type callable

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9
```

(continues on next page)

(continued from previous page)

```
# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Checks if the window is scheduled for closing

Type bool

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.3.3 Window Specific Methods

Window.**glfw_window_resize_callback**(*window*, *width*, *height*)
Window resize callback for glfw

Parameters

- **window** – The window
- **width** – New width
- **height** – New height

Window.**glfw_mouse_event_callback**(*window*, *xpos*, *ypos*)
Mouse position event callback from glfw. Translates the events forwarding them to `cursor_event()`.

Screen coordinates relative to the top-left corner

Parameters

- **window** – The window
- **xpos** – viewport x pos
- **ypos** – viewport y pos

Window.**glfw_mouse_button_callback**(*window*, *button*, *action*, *mods*)
Handle mouse button events and forward them to the example

Parameters

- **window** – The window
- **button** – The button creating the event
- **action** – Button action (press or release)
- **mods** – They modifiers such as ctrl or shift

Window.**glfw_mouse_scroll_callback**(*window*, *x_offset: float*, *y_offset: float*)
Handle mouse scroll events and forward them to the example

Parameters

- **window** – The window
- **x_offset** (*float*) – x wheel offset
- **y_offest** (*float*) – y wheel offset

Window.**glfw_key_event_callback**(*window*, *key*, *scancode*, *action*, *mods*)
Key event callback for glfw. Translates and forwards keyboard event to `keyboard_event()`

Parameters

- **window** – Window event origin
- **key** – The key that was pressed or released.
- **scancode** – The system-specific scancode of the key.
- **action** – `GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`
- **mods** – Bit field describing which modifier keys were held down.

Window.**glfw_char_callback**(*window*, *codepoint: int*)
Handle text input (only unicode charaters)

Parameters

- **window** – The glfw window
- **codepoint** (*int*) – The unicode codepoint

Window.**glfw_cursor_enter**(*window*, *enter: int*)
called when the cursor enters or leaves the content area of the window.

Parameters

- **window** – the window instance
- **enter** (*int*) – 0: leave, 1: enter

Window.**glfw_window_focus**(*window*, *focused: int*)
Called when the window focus is changed.

Parameters

- **window** – The window instance
- **focused** (*int*) – 0: de-focus, 1: focused

Window.**glfw_window_iconify**(*window*, *iconified: int*)
Called when the window is minimized or restored.

Parameters

- **window** – The window
- **iconified** (*int*) – 1 = minimized, 0 = restored.

Window.**glfw_window_close**(*window*)
Called when the window is closed

10.4 headless.Window

10.4.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

Window.**init_mgl_context**() → None
Create an standalone context and framebuffer

Window.**is_key_pressed**(*key*) → bool
Returns: The press state of a key

Window.**set_icon**(*icon_path: str*) → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

Window.**close**() → None
Signal for the window to close

Window.**use**()
Bind the window's framebuffer

Window.**clear**(*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*)
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

Window.**render**(*time=0.0*, *frame_time=0.0*) → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

Window.**swap_buffers**() → None
Placeholder. We currently don't do double buffering in headless mode. This may change in the future.

Window.**resize**(*width*, *height*) → None
Should be called every time window is resized so the example can adapt to the new size if needed

Window.**destroy**() → None
Destroy the context

Window.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

Window.**print_context_info**()
Prints moderngl context info.

10.4.2 Attributes

Window.**name = 'headless'**
Name of the window

Window.**keys**

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**close_func**
Get or set the close callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Is the window about to close?

Type bool

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.5 pyglet.Window

10.5.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.

- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

Window.**init_mgl_context**() → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's gl_version.

Keyword Arguments **ctx** – An optional custom ModernGL context

Window.**is_key_pressed**(*key*) → bool
Returns: The press state of a key

Window.**set_icon**(*icon_path: str*) → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

Window.**close**() → None
Close the pyglet window directly

Window.**use**()
Bind the window's framebuffer

Window.**clear**(*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*)
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

Window.**render**(*time=0.0*, *frame_time=0.0*) → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

Window.**swap_buffers**() → None
Swap buffers, increment frame counter and pull events

Window.**resize**(*width*, *height*) → None
Should be called every time window is resized so the example can adapt to the new size if needed

Window.**destroy**()
Destroy the pyglet window

Window.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

Window.**print_context_info**()
Prints moderngl context info.

10.5.2 Window Specific Methods

Window.**on_mouse_press**(*x: int*, *y: int*, *button*, *mods*)
Handle mouse press events and forward to standard methods

Parameters

- **x** – x position of the mouse when pressed
- **y** – y position of the mouse when pressed
- **button** – The pressed button
- **mods** – Modifiers

Window.**on_key_release**(*symbol*, *modifiers*)
Pyglet specific key release callback.

Forwards and translates the events to standard methods.

Parameters

- **symbol** – The symbol of the pressed key
- **modifiers** – Modifier state (shift, ctrl etc.)

Window.**on_mouse_drag**(*x*, *y*, *dx*, *dy*, *buttons*, *modifiers*)
Pyglet specific mouse drag event.

When a mouse button is pressed this is the only way to capture mouse position events

Window.**on_key_press**(*symbol*, *modifiers*)
Pyglet specific key press callback.

Forwards and translates the events to the standard methods.

Parameters

- **symbol** – The symbol of the pressed key
- **modifiers** – Modifier state (shift, ctrl etc.)

Window.**on_mouse_release**(*x: int*, *y: int*, *button*, *mods*)
Handle mouse release events and forward to standard methods

Parameters

- **x** – x position when mouse button was released
- **y** – y position when mouse button was released
- **button** – The button pressed

- **mods** – Modifiers

Window.**on_mouse_motion**(*x*, *y*, *dx*, *dy*)
Pyglet specific mouse motion callback.

Forwards and translates the event to the standard methods.

Parameters

- **x** – x position of the mouse
- **y** – y position of the mouse
- **dx** – delta x position
- **dy** – delta y position of the mouse

Window.**on_mouse_scroll**(*x*, *y*, *x_offset: float*, *y_offset: float*)
Handle mouse wheel.

Parameters

- **x_offset** (*float*) – X scroll offset
- **y_offset** (*float*) – Y scroll offset

Window.**on_text**(*text*)
Pyglet specific text input callback

Forwards and translates the events to the standard methods.

Parameters **text** (*str*) – The unicode character entered

Window.**on_resize**(*width: int*, *height: int*)
Pyglet specific callback for window resize events forwarding to standard methods

Parameters

- **width** – New window width
- **height** – New window height

Window.**on_show**()
Called when window first appear or restored from hidden state

Window.**on_hide**()
Called when window is minimized

Window.**on_close**()
Pyglet specific window close callback

Window.**on_file_drop**(*x*, *y*, *paths*)
Called when files dropped onto the window

Parameters

- **x** (*int*) – X location in window where file was dropped
- **y** (*int*) – Y location in window where file was dropped
- **paths** (*list*) – List of file paths dropped

10.5.3 Attributes

Window.**name = 'pyglet'**
Name of the window

Window.**keys**
Pyglet specific key constants

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**close_func**
Get or set the close callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Check pyglet's internal exit state

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.6 pyqt5.Window

10.6.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

Window.**init_mgl_context**() → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's gl_version.

Keyword Arguments **ctx** – An optional custom ModernGL context

Window.**is_key_pressed**(*key*) → bool
Returns: The press state of a key

Window.**set_icon**(*icon_path: str*) → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

Window.**close**()
Close the window

Window.**use**()
Bind the window's framebuffer

Window.**clear**(*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*)
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

Window.**render**(*time=0.0*, *frame_time=0.0*) → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

Window.**swap_buffers**() → None
Swap buffers, set viewport, trigger events and increment frame counter

Window.**resize**(*width: int*, *height: int*) → None
Replacement for Qt's resizeGL method.

Parameters

- **width** – New window width
- **height** – New window height

Window.**destroy**() → None
Quit the Qt application to exit the window gracefully

Window.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

Window.**print_context_info**()
Prints moderngl context info.

10.6.2 Window Specific Methods

Window.**close_event**(*event*) → None
The standard PyQt close events

Parameters **event** – The qtevent instance

Window.**mouse_release_event**(*event*) → None
Forward mouse release events to standard methods

Parameters **event** – The qtevent instance

Window.**key_release_event**(*event*) → None
Process Qt key release events forwarding them to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_move_event**(*event*) → None
Forward mouse cursor position events to standard methods

Parameters **event** – The qtevent instance

Window.**key_pressed_event**(*event*) → None
Process Qt key press events forwarding them to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_press_event**(*event*) → None
Forward mouse press events to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_wheel_event**(*event*)
Forward mouse wheel events to standard metods.

From Qt docs:

Returns the distance that the wheel is rotated, in eighths of a degree. A positive value indicates that the wheel was rotated forwards away from the user; a negative value indicates that the wheel was rotated backwards toward the user.

Most mouse types work in steps of 15 degrees, in which case the delta value is a multiple of 120; i.e., 120 units * 1/8 = 15 degrees.

However, some mice have finer-resolution wheels and send delta values that are less than 120 units (less than 15 degrees). To support this possibility, you can either cumulatively add the delta values from events until the value of 120 is reached, then scroll the widget, or you can partially scroll the widget in response to each wheel event.

Parameters **event** (*QWheelEvent*) – Mouse wheel event

Window.**show_event**(*event*)
The standard Qt show event

Window.**hide_event**(*event*)
The standard Qt hide event

10.6.3 Attributes

Window.**name = 'pyqt5'**
Name of the window

Window.**keys**
PyQt5 specific key constants

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**close_func**
Get or set the close callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Is the window about to close?

Type bool

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.7 pyside2.Window

10.7.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title

- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

`Window.`**init_mgl_context**`()` → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's `gl_version`.

Keyword Arguments **ctx** – An optional custom ModernGL context

`Window.`**is_key_pressed**`(`*key*`)` → bool
Returns: The press state of a key

`Window.`**set_icon**`(`*icon_path: str*`)` → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

`Window.`**close**`()`
Close the window

`Window.`**use**`()`
Bind the window's framebuffer

`Window.`**clear**`(`*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*`)`
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component
- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

`Window.`**render**`(`*time=0.0*, *frame_time=0.0*`)` → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

`Window.`**swap_buffers**`()` → None
Swap buffers, set viewport, trigger events and increment frame counter

`Window.`**resize**`(`*width: int*, *height: int*`)` → None
Replacement for Qt's `resizeGL` method.

Parameters

- **width** – New window width
- **height** – New window height

Window.**destroy**() → None
Quit the Qt application to exit the window gracefully

Window.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

Window.**print_context_info**()
Prints moderngl context info.

10.7.2 Window Specific Methods

Window.**close_event**(*event*) → None
The standard PyQt close events

Parameters **event** – The qtevent instance

Window.**mouse_release_event**(*event*) → None
Forward mouse release events to standard methods

Parameters **event** – The qtevent instance

Window.**key_release_event**(*event*)
Process Qt key release events forwarding them to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_move_event**(*event*) → None
Forward mouse cursor position events to standard methods

Parameters **event** – The qtevent instance

Window.**key_pressed_event**(*event*)
Process Qt key press events forwarding them to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_press_event**(*event*) → None
Forward mouse press events to standard methods

Parameters **event** – The qtevent instance

Window.**mouse_wheel_event**(*event*)
Forward mouse wheel events to standard metods.

From Qt docs:

Returns the distance that the wheel is rotated, in eighths of a degree. A positive value indicates that the wheel was rotated forwards away from the user; a negative value indicates that the wheel was rotated backwards toward the user.

Most mouse types work in steps of 15 degrees, in which case the delta value is a multiple of 120; i.e., 120 units * 1/8 = 15 degrees.

However, some mice have finer-resolution wheels and send delta values that are less than 120 units (less than 15 degrees). To support this possibility, you can either cumulatively add the delta values from events until the value of 120 is reached, then scroll the widget, or you can partially scroll the widget in response to each wheel event.

Parameters **event** (*QWheelEvent*) – Mouse wheel event

Window.**show_event**(*event*)
The standard Qt show event

Window.**hide_event**(*event*)
The standard Qt hide event

10.7.3 Attributes

Window.**name = 'pyside2'**
Name of the window

Window.**keys**
PySide2 specific key constants

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**close_func**
Get or set the close callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Is the window about to close?

Type bool

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

10.8 sdl2.Window

10.8.1 Methods

Window.**__init__**(***kwargs*)
Initialize a window instance.

Parameters

- **title** (*str*) – The window title
- **gl_version** (*tuple*) – Major and minor version of the opengl context to create
- **size** (*tuple*) – Window size x, y
- **resizable** (*bool*) – Should the window be resizable?
- **fullscreen** (*bool*) – Open window in fullsceeen mode
- **vsync** (*bool*) – Enable/disable vsync
- **aspect_ratio** (*float*) – The desired fixed aspect ratio. Can be set to `None` to make aspect ratio be based on the actual window size.
- **samples** (*int*) – Number of MSAA samples for the default framebuffer
- **cursor** (*bool*) – Enable/disable displaying the cursor inside the window

Window.**init_mgl_context**() → None
Create or assign a ModernGL context. If no context is supplied a context will be created using the window's `gl_version`.

Keyword Arguments **ctx** – An optional custom ModernGL context

Window.**is_key_pressed**(*key*) → bool
Returns: The press state of a key

Window.**set_icon**(*icon_path: str*) → None
Sets the window icon to the given path

Parameters **icon_path** (*str*) – path to the icon

Window.**close**()
Close the window

Window.**use**()
Bind the window's framebuffer

Window.**clear**(*red=0.0*, *green=0.0*, *blue=0.0*, *alpha=0.0*, *depth=1.0*, *viewport=None*)
Binds and clears the default framebuffer

Parameters

- **red** (*float*) – color component
- **green** (*float*) – color component

- **blue** (*float*) – color component
- **alpha** (*float*) – alpha component
- **depth** (*float*) – depth value
- **viewport** (*tuple*) – The viewport

Window.**render**(*time=0.0*, *frame_time=0.0*) → None
Renders a frame by calling the configured render callback

Keyword Arguments

- **time** (*float*) – Current time in seconds
- **frame_time** (*float*) – Delta time from last frame in seconds

Window.**swap_buffers**() → None
Swap buffers, set viewport, trigger events and increment frame counter

Window.**resize**(*width*, *height*) → None
Resize callback.

Parameters

- **width** – New window width
- **height** – New window height

Window.**destroy**() → None
Gracefully close the window

Window.**set_default_viewport**() → None
Calculates the and sets the viewport based on window configuration.

The viewport will based on the configured fixed aspect ratio if set. If no fixed aspect ratio is set the viewport will be scaled to the entire window size regardless of size.

Will add black borders and center the viewport if the window do not match the configured viewport (fixed only)

Window.**convert_window_coordinates**(*x*, *y*, *x_flipped=False*, *y_flipped=False*)
Convert window coordinates to top-left coordinate space. The default origin is the top left corner of the window.

Args : x_flipped (bool) - if the input x origin is flipped y_flipped (bool) - if the input y origin is flipped

Returns tuple (x, y) of converted window coordinates

If you are converting from bottom origin coordinates use x_flipped=True If you are converting from right origin coordinates use y_flipped=True

Window.**print_context_info**()
Prints moderngl context info.

10.8.2 Window Specific Methods

Window.**process_events**() → None
Handle all queued events in sdl2 dispatching events to standard methods

10.8.3 Attributes

Window.**name = 'sdl2'**
Name of the window

Window.**keys**
SDL2 specific key constants

Window.**ctx**
The ModernGL context for the window

Type moderngl.Context

Window.**fbo**
The default framebuffer

Type moderngl.Framebuffer

Window.**title**
Window title.

This property can also be set:

```
window.title = "New Title"
```

Type str

Window.**exit_key**
Get or set the exit/close key for the window.

Pressing this key will close the window.

By default the ESCAPE is set, but this can be overridden or disabled:

```
# Default exit key
window.exit_key = window.keys.ESCAPE

# Set some other random exit key
window.exit_key = window.keys.Q

# Disable the exit key
window.exit_key = None
```

Window.**gl_version**
(major, minor) required OpenGL version

Type Tuple[int, int]

Window.**width**
The current window width

Type int

Window.**height**
The current window height

Type int

Window.**size**
current window size.

This property also support assignment:

```
# Resize the window to 1000 x 1000
window.size = 1000, 1000
```

Type Tuple[int, int]

Window.**position**
The current window position.

This property can also be set to move the window:

```
# Move window to 100, 100
window.position = 100, 100
```

Type Tuple[int, int]

Window.**buffer_width**
the current window buffer width

Type int

Window.**buffer_height**
the current window buffer height

Type int

Window.**buffer_size**
tuple with the current window buffer size

Type Tuple[int, int]

Window.**pixel_ratio**
The framebuffer/window size ratio

Type float

Window.**viewport**
current window viewport

Type Tuple[int, int, int, int]

Window.**viewport_size**
Size of the viewport.

Equivalent to `self.viewport[2], self.viewport[3]`

Type Tuple[int,int]

Window.**viewport_width**
The width of the viewport.

Equivalent to `self.viewport[2]`.

Type int

Window.**viewport_height**
The height of the viewport

Equivalent to `self.viewport[3]`.

Type int

Window.**frames**
Number of frames rendered

Type int

Window.**resizable**
Window is resizable

Type bool

Window.**fullscreen**
Window is in fullscreen mode

Type bool

Window.**config**
Get the current WindowConfig instance

This property can also be set. Assinging a WindowConfig instance will automatically set up the necessary event callback methods:

```
window.config = window_config_instance
```

Window.**vsync**
vertical sync enabled/disabled

Type bool

Window.**aspect_ratio**
The current aspect ratio of the window. If a fixed aspect ratio was passed to the window initializer this value will always be returned. Otherwise `width / height` will be returned.

This property is read only.

Type float

Window.**fixed_aspect_ratio**
The fixed aspect ratio for the window.

Can be set to `None` to disable fixed aspect ratio making the aspect ratio adjust to the actual window size

This will affects how the viewport is calculated and the reported value from the `aspect_ratio` property:

```
# Enabled fixed aspect ratio
window.fixed_aspect_ratio = 16 / 9

# Disable fixed aspect ratio
window.fixed_aspect_ratio = None
```

Type float

Window.**samples**
Number of Multisample anti-aliasing (MSAA) samples

Type float

Window.**cursor**
Should the mouse cursor be visible inside the window?

This property can also be assigned to:

```
# Disable cursor
window.cursor = False
```

Type bool

Window.**mouse_exclusivity**
If mouse exclusivity is enabled.

When you enable mouse-exclusive mode, the mouse cursor is no longer available. It is not merely hidden – no amount of mouse movement will make it leave your application. This is for example useful when you don't want the mouse leaving the screen when rotating a 3d scene.

This property can also be set:

```
window.mouse_exclusivity = True
```

Type bool

Window.**render_func**
The render callable

This property can also be used to assign a callable.

Type callable

Window.**resize_func**
Get or set the resize callable

Type callable

Window.**close_func**
Get or set the close callable

Type callable

Window.**iconify_func**
Get or set ehe iconify/show/hide callable

Type callable

Window.**key_event_func**
Get or set the key_event callable

Type callable

Window.**mouse_position_event_func**
Get or set the mouse_position callable

Type callable

Window.**mouse_press_event_func**
Get or set the mouse_press callable

Type callable

Window.**mouse_release_event_func**
Get or set the mouse_release callable

Type callable

Window.**mouse_drag_event_func**
Get or set the mouse_drag callable

Type callable

Window.**unicode_char_entered_func**
Get or set the unicode_char_entered callable

Type callable

Window.**mouse_scroll_event_func**
Get or set the mouse_scroll_event calable

Type callable

Window.**files_dropped_event_func**
Get or set the files_dropped callable

Type callable

Window.**is_closing**
Is the window about to close?

Type bool

Window.**mouse = <class 'moderngl_window.context.base.window.MouseButtons'>**

Window.**mouse_states**
Mouse button state structure.

The current mouse button states.

```
window.mouse_buttons.left
window.mouse_buttons.right
window.mouse_buttons.middle
```

Type MouseButtonStates

Window.**modifiers**
(KeyModifiers) The current keyboard modifiers

Window.**gl_version_code**
Generates the version code integer for the selected OpenGL version.

gl_version (4, 1) returns 410

Type int

CHAPTER

ELEVEN

MODERNGL_WINDOW.GEOMETRY

moderngl_window.geometry.**bbox**(*size=(1.0, 1.0, 1.0)*, *name=None*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*)

Generates a bounding box with (0.0, 0.0, 0.0) as the center. This is simply a box with `LINE_STRIP` as draw mode.

Keyword Arguments

- **size** (*tuple*) – x, y, z size of the box
- **name** (*str*) – Optional name for the VAO
- **attr_names** (*AttributeNames*) – Attribute names

Returns A *moderngl_window.opengl.vao.VAO* instance

moderngl_window.geometry.**quad_fs**(*attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*, *normals=True*, *uvs=True*, *name=None*) → moderngl_window.opengl.vao.VAO

Creates a screen aligned quad using two triangles with normals and texture coordinates.

Keyword Arguments

- **attr_names** (*AttributeNames*) – Attrib name config
- **normals** (*bool*) – Include normals in VAO
- **uvs** (*bool*) – Include normals in VAO
- **name** (*str*) – Optional name for the VAO

Returns A *VAO* instance.

moderngl_window.geometry.**quad_2d**(*size=(1.0, 1.0)*, *pos=(0.0, 0.0)*, *normals=True*, *uvs=True*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*, *name=None*) → moderngl_window.opengl.vao.VAO

Creates a 2D quad VAO using 2 triangles with normals and texture coordinates.

Keyword Arguments

- **size** (*tuple*) – width and height
- **pos** (*float*) – Center position x and y
- **normals** (*bool*) – Include normals in VAO
- **uvs** (*bool*) – Include normals in VAO
- **attr_names** (*AttributeNames*) – Attrib name config
- **name** (*str*) – Optional name for the VAO

Returns A *VAO* instance.

moderngl_window.geometry.**cube**(*size=(1.0, 1.0, 1.0)*, *center=(0.0, 0.0, 0.0)*, *normals=True*, *uvs=True*, *name=None*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*) → moderngl_window.opengl.vao.VAO

Creates a cube VAO with normals and texture coordinates

Keyword Arguments

- **width** (*float*) – Width of the cube
- **height** (*float*) – Height of the cube
- **depth** (*float*) – Depth of the cube
- **center** – center of the cube as a 3-component tuple
- **normals** – (bool) Include normals
- **uvs** – (bool) include uv coordinates
- **name** (*str*) – Optional name for the VAO
- **attr_names** (*AttributeNames*) – Attribute names

Returns A *moderngl_window.opengl.vao.VAO* instance

moderngl_window.geometry.**sphere**(*radius=0.5*, *sectors=32*, *rings=16*, *normals=True*, *uvs=True*, *name: str = None*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*) → moderngl_window.opengl.vao.VAO

Creates a sphere.

Keyword Arguments

- **radius** (*float*) – Radius or the sphere
- **rings** (*int*) – number or horizontal rings
- **sectors** (*int*) – number of vertical segments
- **normals** (*bool*) – Include normals in the VAO
- **uvs** (*bool*) – Include texture coordinates in the VAO
- **name** (*str*) – An optional name for the VAO
- **attr_names** (*AttributeNames*) – Attribute names

Returns A VAO instance

CHAPTER

TWELVE

MODERNGL_WINDOW.LOADERS

12.1 base.BaseLoader

12.1.1 Method

BaseLoader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod BaseLoader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

BaseLoader.**load**() → Any
Loads a resource.

When creating a loader this is the only method that needs to be implemented.

Returns The loaded resource

BaseLoader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

BaseLoader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

BaseLoader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

BaseLoader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.1.2 Attributes

`BaseLoader.`**`kind = None`**
The kind of resource this loaded supports. This can be used when file extensions is not enough to decide what loader should be selected.

`BaseLoader.`**`file_extensions = []`**
A list defining the file extensions accepted by this loader.

Example:

```
# Loader will match .xyz and .xyz.gz files.
file_extensions = [
    ['.xyz'],
    ['.xyz', '.gz'],
]
```

`BaseLoader.`**`ctx`**
ModernGL context

Type moderngl.Context

12.2 texture.t2d.Loader

12.2.1 Method

`Loader.`**`__init__`**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod `Loader.`**`supports_file`**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

`Loader.`**`load`**()
Load a 2d texture as configured in the supplied `TextureDescription`

Returns The Texture instance

Return type moderngl.Texture

`Loader.`**`find_data`**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

`Loader.`**`find_program`**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.2.2 Attributes

Loader.**kind = '2d'**

Loader.**file_extensions = []**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.3 program.single.Loader

12.3.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → moderngl.program.Program
Loads a shader program from a single glsl file.

Each shader type is separated by preprocessors

- VERTEX_SHADER
- FRAGMENT_SHADER
- GEOMETRY_SHADER
- TESS_CONTROL_SHADER
- TESS_EVALUATION_SHADER

Example:

```
#version 330

#if defined VERTEX_SHADER

in vec3 in_position;
in vec2 in_texcoord_0;
out vec2 uv0;

void main() {
    gl_Position = vec4(in_position, 1);
    uv0 = in_texcoord_0;
}

#elif defined FRAGMENT_SHADER

out vec4 fragColor;
uniform sampler2D texture0;
in vec2 uv0;

void main() {
    fragColor = texture(texture0, uv0);
}
#endif
```

Returns The Program instance

Return type moderngl.Program

`Loader.`**`find_data`**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **`path`** – Path to resource

`Loader.`**`find_program`**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **`path`** – Path to resource

`Loader.`**`find_texture`**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **`path`** – Path to resource

`Loader.`**`find_scene`**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **`path`** – Path to resource

12.3.2 Attributes

`Loader.`**`kind = 'single'`**

Loader.**file_extensions = []**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.4 program.separate.Loader

12.4.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → Union[moderngl.program.Program, moderngl.compute_shader.ComputeShader, moderngl_window.opengl.program.ReloadableProgram]
Loads a shader program were each shader is a separate file.

This detected and dictated by the `kind` in the `ProgramDescription`.

Returns The Program instance

Return type moderngl.Program

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.4.2 Attributes

Loader.**kind = 'separate'**

Loader.**file_extensions = []**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.5 texture.array.Loader

12.5.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**()
Load a texture array as described by the supplied `TextureDescription`

Returns The TextureArray instance

Return type moderngl.TextureArray

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.5.2 Attributes

Loader.**kind = 'array'**

Loader.**file_extensions = []**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.6 scene.wavefront.Loader

12.6.1 Method

Loader.**__init__**(*meta: moderngl_window.meta.scene.SceneDescription*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**()
Loads a wavefront/obj file including materials and textures

Returns The Scene instance

Return type Scene

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.6.2 Attributes

Loader.**kind = 'wavefront'**

Loader.**file_extensions = [['.obj'], ['.obj', '.gz'], ['.bin']]**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.7 scene.gltf2.Loader

12.7.1 Method

Loader.**__init__**(*meta: moderngl_window.meta.scene.SceneDescription*)
Initialize loading GLTF 2 scene.

Supported formats:

- gltf json format with external resources
- gltf embedded buffers
- glb Binary format

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → moderngl_window.scene.scene.Scene
Load a GLTF 2 scene including referenced textures.

Returns The scene instance

Return type Scene

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.7.2 Loader Specific Methods

Loader.**load_gltf**()
Loads a gltf json file parsing its contents

Loader.**load_glb**()
Loads a binary gltf file parsing its contents

Loader.**load_materials**()
Load materials referenced in gltf metadata

Loader.**load_nodes**()
Load nodes referenced in gltf metadata

Loader.**load_node**(*meta*, *parent=None*)
Load a single node

Loader.**load_images**()
Load images referenced in gltf metadata

Loader.**load_textures**()
Load textures referenced in gltf metadata

Loader.**load_samplers**()
Load samplers referenced in gltf metadata

Loader.**load_meshes**()
Load meshes referenced in gltf metadata

12.7.3 Attributes

Loader.**kind = 'gltf'**

Loader.**file_extensions = [['.gltf'], ['.glb']]**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.7.4 Loader Specific Attributes

Loader.**supported_extensions = []**
Supported GLTF extensions https://github.com/KhronosGroup/glTF/tree/master/extensions

12.8 scene.stl.Loader

12.8.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → moderngl_window.scene.scene.Scene
Loads and stl scene/file

Returns The Scene instance

Return type Scene

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.8.2 Attributes

Loader.**kind = 'stl'**

Loader.**file_extensions = [['.stl'], ['.stl', '.gz']]**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.9 data.json.Loader

12.9.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → dict
Load a file as json

Returns The json contents

Return type dict

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.9.2 Attributes

Loader.**kind = 'json'**

Loader.**file_extensions = [['.json']]**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.10 data.text.Loader

12.10.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → str
Load a file in text mode.

Returns The string contents of the file

Return type str

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.10.2 Attributes

Loader.**kind = 'text'**

Loader.**file_extensions = [['.txt']]**

Loader.**ctx**
ModernGL context

Type moderngl.Context

12.11 data.binary.Loader

12.11.1 Method

Loader.**__init__**(*meta*)
Initialize loader.

Loaders take a ResourceDescription instance containing all the parameters needed to load and initialize this data.

Parameters **meta** (*ResourceDescription*) – The resource to load

classmethod Loader.**supports_file**(*meta*)
Check if the loader has a supported file extension.

What extensions are supported can be defined in the *file_extensions* class attribute.

Loader.**load**() → bytes
Load a file in binary mode

Returns The bytes contents of the file

Return type bytes

Loader.**find_data**(*path*)
Find resource using data finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_program**(*path*)
Find resource using program finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_texture**(*path*)
Find resource using texture finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

Loader.**find_scene**(*path*)
Find resource using scene finders.

This is mainly a shortcut method to simplify the task.

Parameters **path** – Path to resource

12.11.2 Attributes

Loader.**kind = 'binary'**

Loader.**file_extensions = []**

Loader.**ctx**
ModernGL context

Type moderngl.Context

CHAPTER

THIRTEEN

MODERNGL_WINDOW.META

13.1 base.ResourceDescription

moderngl_window.meta.base.**ResourceDescription**
Description of any resource. Resource descriptions are required to load a resource. This class can be extended to add more specific properties.

13.1.1 Methods

ResourceDescription.**__init__**(***kwargs*)
Initialize a resource description

Parameters ****kwargs** – Attributes describing the resource to load

13.1.2 Attributes

ResourceDescription.**path**
The path to a resource when a single file is specified

Type str

ResourceDescription.**resolved_path**
The resolved path by a finder.

The absolute path to the resource can optionally be assigned by a loader class.

Type pathlib.Path

ResourceDescription.**attrs**
All keywords arguments passed to the resource

Type dict

ResourceDescription.**label**
optional name for the resource

Assigning a label is not mandatory but can help when aliasing resources. Some prefer to preload all needed resources and fetch them later by the label. This can he a lot less chaotic in larger applications.

Type str

ResourceDescription.**kind**
default resource kind.

The resource `kind` is directly matched with the `kind` in loader classes.

This property also supports assignment and is useful if the `kind` is detected based in the the attribute values.

```
description.kind = 'something'
```

Type str

`ResourceDescription.`**`loader_cls`**
The loader class for this resource.

This property is assigned to during the loading stage were a loader class is assigned based on the *kind*.

Type Type

`ResourceDescription.`**`default_kind = None`**
The default kind for this resource type

Type str

`ResourceDescription.`**`resource_type = None`**
A unique identifier for the resource type

Type str

13.2 texture.TextureDescription

`moderngl_window.meta.texture.`**`TextureDescription`**
Describes a texture to load.

Example:

```
# Loading a 2d texture
TextureDescription(path='textures/wood.png')

# Loading a 2d texture with mimpmaps with anisotropy
TextureDescription(path='textures/wood.png', mipmap=True, anisotropy=16.0)

# Loading texture array containing 10 layers
TextureDescription(path='textures/tiles.png', layers=10, kind='array')
```

13.2.1 Methods

`TextureDescription.`**`__init__`**(*path: str = None*, *kind: str = None*, *flip=True*, *flip_x=False*, *flip_y=True*, *mipmap=False*, *mipmap_levels: Tuple[int, int] = None*, *anisotropy=1.0*, *image=None*, *layers=None*, *pos_x: str = None*, *pos_y: str = None*, *pos_z: str = None*, *neg_x: str = None*, *neg_y: str = None*, *neg_z: str = None*, ***kwargs*)
Describes a texture resource

Parameters

- **`path`** (*`str`*) – path to resource relative to search directories
- **`kind`** (*`str`*) – The kind of loader to use
- **`flip`** (*`boolean`*) – (use flip_y) Flip the image vertically (top to bottom)
- **`flip_x`** (*`boolean`*) – Flip the image horizontally (left to right)

- **flip_y** (*boolean*) – Flip the image vertically (top to bottom)
- **mipmap** (*bool*) – Generate mipmaps. Will generate max possible levels unless *mipmap_levels* is defined.
- **mipmap_levels** (*tuple*) – (base, max_level) controlling mipmap generation. When defined the *mipmap* parameter is automatically *True*.
- **anisotropy** (*float*) – Number of samples for anisotropic filtering
- **image** – PIL image for when loading embedded resources
- **layers** – (int): Number of layers for texture arrays
- **neg_x** (*str*) – Path to negative x texture in a cube map
- **neg_y** (*str*) – Path to negative y texture in a cube map
- **neg_z** (*str*) – Path to negative z texture in a cube map
- **pos_x** (*str*) – Path to positive x texture in a cube map
- **pop_y** (*str*) – Path to positive y texture in a cube map
- **pos_z** (*str*) – Path to positive z texture in a cube map
- ****kwargs** – Any optional/custom attributes

13.2.2 Attributes

TextureDescription.**mipmap**
If mipmaps should be generated

Type bool

TextureDescription.**image**
PIL image when loading embedded resources

Type Image

TextureDescription.**layers**
Number of layers in texture array

Type int

TextureDescription.**anisotropy**
Number of samples for anisotropic filtering

Type float

TextureDescription.**mipmap_levels**
base, max_level for mipmap generation

Type Tuple[int, int]

TextureDescription.**flip_x**
If the image should be flipped horizontally (left to right)

Type bool

TextureDescription.**flip_y**
If the image should be flipped vertically (top to bottom)

Type bool

TextureDescription.**pos_x**
Path to positive x in a cubemap texture

Type str

TextureDescription.**pos_y**
Path to positive y in a cubemap texture

Type str

TextureDescription.**pos_z**
Path to positive z in a cubemap texture

Type str

TextureDescription.**neg_x**
Path to negative x in a cubemap texture

Type str

TextureDescription.**neg_y**
Path to negative y in a cubemap texture

Type str

TextureDescription.**neg_z**
Path to negative z in a cubemap texture

Type str

13.2.3 Inherited Attributes

TextureDescription.**path**
The path to a resource when a single file is specified

Type str

TextureDescription.**resolved_path**
The resolved path by a finder.

The absolute path to the resource can optionally be assigned by a loader class.

Type pathlib.Path

TextureDescription.**attrs**
All keywords arguments passed to the resource

Type dict

TextureDescription.**label**
optional name for the resource

Assigning a label is not mandatory but can help when aliasing resources. Some prefer to preload all needed resources and fetch them later by the label. This can he a lot less chaotic in larger applications.

Type str

TextureDescription.**kind**
default resource kind.

The resource `kind` is directly matched with the `kind` in loader classes.

This property also supports assignment and is useful if the `kind` is detected based in the the attribute values.

```
description.kind = 'something'
```

Type str

TextureDescription.**loader_cls**
The loader class for this resource.

This property is assigned to during the loading stage were a loader class is assigned based on the *kind*.

Type Type

TextureDescription.**default_kind = '2d'**

TextureDescription.**resource_type = 'textures'**

13.3 program.ProgramDescription

moderngl_window.meta.program.**ProgramDescription**
Describes a program to load

By default a program can be loaded in the following ways:

- By supplying a *path* to s single glsl file containing all shaders
- By supplying several paths to separate files containing each shader type. For example `vertex_shader`, `fragment_shader` .. etc.

```
# Single glsl file containing all shaders
ProgramDescription(path='programs/myprogram.glsl')

# Multiple shader files
ProgramDescription(
    vertex_shader='programs/myprogram_vs.glsl',
    fragment_shader='programs/myprogram_fs.glsl',
    geometry_shader='programs/myprogram_gs.glsl',
)
```

13.3.1 Methods

ProgramDescription.**__init__**(*path: str = None*, *kind: str = None*, *reloadable=False*, *vertex_shader: str = None*, *geometry_shader: str = None*, *fragment_shader: str = None*, *tess_control_shader: str = None*, *tess_evaluation_shader: str = None*, *compute_shader: str = None*, *defines: dict = None*, ***kwargs*)
Create a program description

Keyword Arguments

- **path** (*str*) – path to the resource relative to search directories
- **kind** (*str*) – The kind of loader to use
- **reloadable** (*bool*) – Should this program be reloadable
- **vertex_shader** (*str*) – Path to vertex shader file
- **geometry_shader** (*str*) – Path to geometry shader

- **fragment_shader** (*str*) – Path to fragmet shader
- **tess_control_shader** (*str*) –
- **tess_evaluation_shader** (*str*) – Path to tess eval shader
- **compute_shader** (*str*) – Path to compute shader
- **defines** (*dict*) – Dictionary with define values to replace in the source
- ****kwargs** – Optional custom attributes

13.3.2 Attributes

ProgramDescription.**tess_evaluation_shader**
Relative path to tessellation evaluation shader

Type str

ProgramDescription.**vertex_shader**
Relative path to vertex shader

Type str

ProgramDescription.**geometry_shader**
Relative path to geometry shader

Type str

ProgramDescription.**reloadable**
if this program is reloadable

Type bool

ProgramDescription.**fragment_shader**
Relative path to fragment shader

Type str

ProgramDescription.**tess_control_shader**
Relative path to tess control shader

Type str

ProgramDescription.**compute_shader**
Relative path to compute shader

Type str

ProgramDescription.**defines**
Dictionary with define values to replace in the source

Type dict

13.3.3 Inherited Attributes

ProgramDescription.**path**
The path to a resource when a single file is specified

Type str

ProgramDescription.**resolved_path**
The resolved path by a finder.

The absolute path to the resource can optionally be assigned by a loader class.

Type pathlib.Path

ProgramDescription.**attrs**
All keywords arguments passed to the resource

Type dict

ProgramDescription.**label**
optional name for the resource

Assigning a label is not mandatory but can help when aliasing resources. Some prefer to preload all needed resources and fetch them later by the label. This can he a lot less chaotic in larger applications.

Type str

ProgramDescription.**kind**
default resource kind.

The resource `kind` is directly matched with the `kind` in loader classes.

This property also supports assignment and is useful if the `kind` is detected based in the the attribute values.

```
description.kind = 'something'
```

Type str

ProgramDescription.**loader_cls**
The loader class for this resource.

This property is assigned to during the loading stage were a loader class is assigned based on the *kind*.

Type Type

ProgramDescription.**default_kind = None**

ProgramDescription.**resource_type = 'programs'**

13.4 scene.SceneDescription

moderngl_window.meta.scene.**SceneDescription**
Describes a scene to load.

The correct loader is resolved by looking at the file extension. This can be overridden by specifying a `kind` that maps directly to a specific loader class.

```
# Wavefront/obj file
SceneDescription(path='scenes/cube.obj')

# stl file
SceneDescription(path='scenes/crater.stl')

# GLTF 2 file
SceneDescription(path='scenes/sponza.gltf')
```

The user can also override what buffer/attribute names should be used by specifying `attr_names`.

A `cache` option is also available as some scene loaders supports converting the file into a different format on the fly to speed up loading.

13.4.1 Methods

`SceneDescription.__init__`(*path=None*, *kind=None*, *cache=False*, *attr_names=<class 'moderngl_window.geometry.attributes.AttributeNames'>*, ***kwargs*)

Create a scene description.

Keyword Arguments

- **path** (*str*) – Path to resource
- **kind** (*str*) – Loader kind
- **cache** (*str*) – Use the loader caching system if present
- **attr_names** (*AttributeNames*) – Attrib name config
- ****kwargs** – Optional custom attributes

13.4.2 Attributes

`SceneDescription.attr_names`

Attribute name config

Type AttributeNames

`SceneDescription.cache`

Use cache feature in scene loader

Type bool

13.4.3 Inherited Attributes

`SceneDescription.path`

The path to a resource when a single file is specified

Type str

`SceneDescription.resolved_path`

The resolved path by a finder.

The absolute path to the resource can optionally be assigned by a loader class.

Type pathlib.Path

`SceneDescription.attrs`

All keywords arguments passed to the resource

Type dict

`SceneDescription.label`

optional name for the resource

Assigning a label is not mandatory but can help when aliasing resources. Some prefer to preload all needed resources and fetch them later by the label. This can he a lot less chaotic in larger applications.

Type str

SceneDescription.**kind**
default resource kind.

The resource `kind` is directly matched with the `kind` in loader classes.

This property also supports assignment and is useful if the `kind` is detected based in the the attribute values.

```
description.kind = 'something'
```

Type str

SceneDescription.**loader_cls**
The loader class for this resource.

This property is assigned to during the loading stage were a loader class is assigned based on the *kind*.

Type Type

SceneDescription.**default_kind = None**

SceneDescription.**resource_type = 'scenes'**

13.5 data.DataDescription

moderngl_window.meta.data.**DataDescription**
Describes data file to load.

This is a generic resource description type for loading resources that are not textures, programs and scenes. That loaded class is used depends on the `kind` or the file extension.

Currently used to load:

- text files
- json files
- binary files

```
# Describe a text file. Text loader is used based on file extension
DataDescription(path='data/text.txt')

# Describe a json file. Json loader is used based on file extension
DataDescription(path='data/data.json')

# Describe a binary file. Specify a binary loader should be used.
DataDescription(path='data/data.bin', kind='binary')
```

13.5.1 Methods

DataDescription.**__init__**(*path=None*, *kind=None*, ***kwargs*)
Initialize the resource description.

Keyword Arguments

- **path** (*str*) – Relative path to the resource
- **kind** (*str*) – The resource kind deciding loader class
- ****kwargs** – Additional custom attributes

13.5.2 Attributes

DataDescription.**path**
The path to a resource when a single file is specified

Type str

DataDescription.**resolved_path**
The resolved path by a finder.

The absolute path to the resource can optionally be assigned by a loader class.

Type pathlib.Path

DataDescription.**attrs**
All keywords arguments passed to the resource

Type dict

DataDescription.**label**
optional name for the resource

Assigning a label is not mandatory but can help when aliasing resources. Some prefer to preload all needed resources and fetch them later by the label. This can he a lot less chaotic in larger applications.

Type str

DataDescription.**kind**
default resource kind.

The resource `kind` is directly matched with the `kind` in loader classes.

This property also supports assignment and is useful if the `kind` is detected based in the the attribute values.

```
description.kind = 'something'
```

Type str

DataDescription.**loader_cls**
The loader class for this resource.

This property is assigned to during the loading stage were a loader class is assigned based on the *kind*.

Type Type

DataDescription.**default_kind = None**

DataDescription.**resource_type = 'data'**

CHAPTER

FOURTEEN

MODERNGL_WINDOW.FINDERS

14.1 base.BaseFilesystemFinder

moderngl_window.finders.base.**BaseFilesystemFinder**
Base class for searching filesystem directories

14.1.1 Methods

BaseFilesystemFinder.**__init__**()
Initialize finder class by looking up the paths referenced in settings_attr.

BaseFilesystemFinder.**find**(*path: pathlib.Path*) → pathlib.Path
Finds a file in the configured paths returning its absolute path.

Parameters **path** (*pathlib.Path*) – The path to find

Returns The absolute path to the file or None if not found

14.1.2 Attributes

BaseFilesystemFinder.**settings_attr = None**
Name of the attribute in *Settings* containing a list of paths the finder should search in.

Type str

14.2 texture.FilesystemFinder

moderngl_window.finders.texture.**FilesystemFinder**
Find textures in settings.TEXTURE_DIRS

14.2.1 Methods

FilesystemFinder.**__init__**()
Initialize finder class by looking up the paths referenced in settings_attr.

FilesystemFinder.**find**(*path: pathlib.Path*) → pathlib.Path
Finds a file in the configured paths returning its absolute path.

Parameters **path** (*pathlib.Path*) – The path to find

Returns The absolute path to the file or None if not found

14.2.2 Attributes

`FilesystemFinder.`**`settings_attr = 'TEXTURE_DIRS'`**

14.3 program.FilesystemFinder

`moderngl_window.finders.program.`**`FilesystemFinder`**
Find shaders in `settings.PROGRAM_DIRS`

14.3.1 Methods

`FilesystemFinder.`**`__init__`**`()`
Initialize finder class by looking up the paths referenced in `settings_attr`.

`FilesystemFinder.`**`find`**`(`*path: pathlib.Path*`)` → pathlib.Path
Finds a file in the configured paths returning its absolute path.

Parameters **`path`** (*`pathlib.Path`*) – The path to find

Returns The absolute path to the file or None if not found

14.3.2 Attributes

`FilesystemFinder.`**`settings_attr = 'PROGRAM_DIRS'`**

14.4 scene.FilesystemFinder

`moderngl_window.finders.scene.`**`FilesystemFinder`**
Find scenes in `settings.SCENE_DIRS`

14.4.1 Methods

`FilesystemFinder.`**`__init__`**`()`
Initialize finder class by looking up the paths referenced in `settings_attr`.

`FilesystemFinder.`**`find`**`(`*path: pathlib.Path*`)` → pathlib.Path
Finds a file in the configured paths returning its absolute path.

Parameters **`path`** (*`pathlib.Path`*) – The path to find

Returns The absolute path to the file or None if not found

14.4.2 Attributes

`FilesystemFinder.`**`settings_attr = 'SCENE_DIRS'`**

14.5 data.FilesystemFinder

moderngl_window.finders.data.**FilesystemFinder**
Find data in settings.DATA_DIRS

14.5.1 Methods

FilesystemFinder.**__init__**()
Initialize finder class by looking up the paths referenced in settings_attr.

FilesystemFinder.**find**(*path: pathlib.Path*) → pathlib.Path
Finds a file in the configured paths returning its absolute path.

Parameters **path** (*pathlib.Path*) – The path to find

Returns The absolute path to the file or None if not found

14.5.2 Attributes

FilesystemFinder.**settings_attr = 'DATA_DIRS'**

CHAPTER

FIFTEEN

MODERNGL_WINDOW.OPENGL

15.1 opengl.projection.Projection3D

moderngl_window.opengl.projection.**Projection3D**
3D Projection

15.1.1 Methods

Projection3D.**__init__**(*aspect_ratio=1.7777777777777777*, *fov=75.0*, *near=1.0*, *far=100.0*)
Create a 3D projection

Keyword Arguments

- **aspect_ratio** (*float*) – Aspect ratio
- **fov** (*float*) – Field of view
- **near** (*float*) – Near plane value
- **far** (*float*) – Far plane value

Projection3D.**update**(*aspect_ratio: float = None*, *fov: float = None*, *near: float = None*, *far: float = None*) → None
Update the projection matrix

Keyword Arguments

- **aspect_ratio** (*float*) – Aspect ratio
- **fov** (*float*) – Field of view
- **near** (*float*) – Near plane value
- **far** (*float*) – Far plane value

Projection3D.**tobytes**() → bytes
Get the byte representation of the projection matrix

Returns byte representation of the projection matrix

Return type bytes

15.1.2 Attributes

Projection3D.**aspect_ratio**
The projection's aspect ratio

Type float

`Projection3D.`**`fov`**
Current field of view

Type float

`Projection3D.`**`near`**
Current near plane value

Type float

`Projection3D.`**`far`**
Current far plane value

Type float

`Projection3D.`**`matrix`**
Current numpy projection matrix

Type np.ndarray

`Projection3D.`**`projection_constants`**
(x, y) projection constants for the current projection. This is for example useful when reconstructing a view position of a fragment from a linearized depth value.

15.2 opengl.vao.VAO

`moderngl_window.opengl.vao.`**`VAO`**
Represents a vertex array object.

This is a wrapper class over `moderngl.VertexArray` to make interactions with programs/shaders simpler. Named buffers are added correspoding with attribute names in a vertex shader. When rendering the VAO an internal `moderngl.VertextArray` is created automatically mapping the named buffers compatible with the supplied program. This program is cached internally.

The shader program doesn't need to use all the buffers registered in this wrapper. When a subset is used only the used buffers are mapped and the appropriate padding is calculated when interleaved data is used.

You are not required to use this class, but most methods in the system creating vertexbuffers will return this type. You can obtain a single `moderngl.VertexBuffer` instance by calling *`VAO.instance()`* method if you prefer to work directly on moderngl instances.

Example:

```
# Separate buffers
vao = VAO(name="test", mode=moderngl.POINTS)
vao.buffer(positions, '3f', ['in_position'])
vao.buffer(velocities, '3f', ['in_velocities'])

# Interleaved
vao = VAO(name="test", mode=moderngl.POINTS)
vao.buffer(interleaved_data, '3f 3f', ['in_position', 'in_velocities'])
```

```
# GLSL vertex shader in attributes
in vec3 in_position;
in vec3 in_velocities;
```

15.2.1 Methods

VAO.**__init__**(*name=''*, *mode=4*)
Create and empty VAO with a name and default render mode.

Example:

```
VAO(name="cube", mode=moderngl.TRIANGLES)
```

Keyword Arguments

- **name** (*str*) – Optional name for debug purposes
- **mode** (*int*) – Default draw mode

VAO.**render**(*program: moderngl.program.Program*, *mode=None*, *vertices=-1*, *first=0*, *instances=1*)
Render the VAO.

An internal `moderngl.VertexBuffer` with compatible buffer bindings is automatically created on the fly and cached internally.

Parameters **program** – The `moderngl.Program`

Keyword Arguments

- **mode** – Override the draw mode (`TRIANGLES` etc)
- **vertices** (*int*) – The number of vertices to transform
- **first** (*int*) – The index of the first vertex to start with
- **instances** (*int*) – The number of instances

VAO.**render_indirect**(*program: moderngl.program.Program*, *buffer*, *mode=None*, *count=-1*, ***, *first=0*)
The render primitive (mode) must be the same as the input primitive of the GeometryShader. The draw commands are 5 integers: (count, instanceCount, firstIndex, baseVertex, baseInstance).

Parameters

- **program** – The `moderngl.Program`
- **buffer** – The `moderngl.Buffer` containing indirect draw commands

Keyword Arguments

- **mode** (*int*) – By default `TRIANGLES` will be used.
- **count** (*int*) – The number of draws.
- **first** (*int*) – The index of the first indirect draw command.

VAO.**transform**(*program: moderngl.program.Program*, *buffer: moderngl.buffer.Buffer*, *mode=None*, *vertices=-1*, *first=0*, *instances=1*)
Transform vertices. Stores the output in a single buffer.

Parameters

- **program** – The `moderngl.Program`
- **buffer** – The `moderngl.buffer` to store the output

Keyword Arguments

- **mode** – Draw mode (for example `moderngl.POINTS`)
- **vertices** (*int*) – The number of vertices to transform

- **first** (*int*) – The index of the first vertex to start with
- **instances** (*int*) – The number of instances

VAO.**buffer**(*buffer, buffer_format: str, attribute_names: List[str]*)
Register a buffer/vbo for the VAO. This can be called multiple times. adding multiple buffers (interleaved or not).

Parameters

- **buffer** – The buffer data. Can be `numpy.array`, `moderngl.Buffer` or `bytes`.
- **buffer_format** (*str*) – The format of the buffer. (eg. `3f 3f` for interleaved positions and normals).
- **attribute_names** – A list of attribute names this buffer should map to.

Returns The `moderngl.Buffer` instance object. This is handy when providing `bytes` and `numpy.array`.

VAO.**index_buffer**(*buffer*, *index_element_size=4*)
Set the index buffer for this VAO.

Parameters **buffer** – `moderngl.Buffer`, `numpy.array` or `bytes`

Keyword Arguments **index_element_size** (*int*) – Byte size of each element. 1, 2 or 4

VAO.**instance**(*program: moderngl.program.Program*) → moderngl.vertex_array.VertexArray
Obtain the `moderngl.VertexArray` instance for the program.

The instance is only created once and cached internally.

Parameters **program** (*moderngl.Program*) – The program

Returns instance

Return type `moderngl.VertexArray`

VAO.**release**(*buffer=True*)
Destroy all internally cached vaos and release all buffers.

Keyword Arguments **buffers** (*bool*) – also release buffers

VAO.**get_buffer_by_name**(*name: str*) → moderngl_window.opengl.vao.BufferInfo
Get the BufferInfo associated with a specific attribute name

If no buffer is associated with the name *None* will be returned.

Parameters **name** (*str*) – Name of the mapped attribute

Returns BufferInfo instance

Return type BufferInfo

15.2.2 Attributes

VAO.**ctx**
The actite moderngl context

Type moderngl.Context

CHAPTER

SIXTEEN

MODERNGL_WINDOW.RESOURCES

moderngl_window.resources.**register_dir**(*path: Union[pathlib.Path, str]*) → None
Adds a resource directory for all resource types

Parameters **path** (*Union[Path, str]*) – Directory path

moderngl_window.resources.**register_program_dir**(*path: Union[pathlib.Path, str]*) → None
Adds a resource directory specifically for programs

Parameters **path** (*Union[Path, str]*) – Directory path

moderngl_window.resources.**register_texture_dir**(*path: Union[pathlib.Path, str]*) → None
Adds a resource directory specifically for textures

Parameters **path** (*Union[Path, str]*) – Directory path

moderngl_window.resources.**register_scene_dir**(*path: Union[pathlib.Path, str]*) → None
Adds a resource directory specifically for scenes

Parameters **path** (*Union[Path, str]*) – Directory path

moderngl_window.resources.**register_data_dir**(*path: Union[pathlib.Path, str]*) → None
Adds a resource directory specifically for data files

Parameters **path** (*Union[Path, str]*) – Directory path

16.1 base.BaseRegistry

moderngl_window.resources.base.**BaseRegistry**
Base class for all resource pools

16.1.1 Methods

BaseRegistry.**__init__**()
Initialize internal attributes

BaseRegistry.**load**(*meta: moderngl_window.meta.base.ResourceDescription*) → Any
Loads a resource using the configured finders and loaders.

Parameters **meta** (*ResourceDescription*) – The resource description

BaseRegistry.**add**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Adds a resource description without loading it. The resource is loaded and returned when `load_pool()` is called.

Parameters **meta** (*ResourceDescription*) – The resource description

BaseRegistry.**load_pool**() → Generator[Tuple[moderngl_window.meta.base.ResourceDescription, Any], None, None]
Loads all the data files using the configured finders.

This is only relevant when resource have been added to this pool using `add()`.

Returns Generator of (meta, resource) tuples

BaseRegistry.**resolve_loader**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Attempts to assign a loader class to a ResourceDescription.

Parameters **meta** (*ResourceDescription*) – The resource description instance

16.1.2 Attributes

BaseRegistry.**settings_attr = None**
The name of the attribute in *Settings* containting a list of loader classes.

Type str

BaseRegistry.**count**
The number of resource descriptions added. This is only relevant when using *add* and *load_pool*.

Type int

BaseRegistry.**loaders**
Loader classes for this resource type

Type Generator

16.2 textures.Textures

moderngl_window.resources.textures.**Textures**
Handles texture resources

16.2.1 Methods

Textures.**__init__**()
Initialize internal attributes

Textures.**load**(*meta: moderngl_window.meta.texture.TextureDescription*) → Union[moderngl.texture.Texture, moderngl.texture_array.TextureArray]
Loads a texture with the configured loaders.

Parameters **meta** (*TextureDescription*) – The resource description

Returns 2d texture

Return type moderngl.Texture

Returns texture array if `layers` is supplied

Return type moderngl.TextureArray

Textures.**add**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Adds a resource description without loading it. The resource is loaded and returned when `load_pool()` is called.

Parameters **meta** (*ResourceDescription*) – The resource description

Textures.**load_pool**() → Generator[Tuple[moderngl_window.meta.base.ResourceDescription, Any], None, None]
Loads all the data files using the configured finders.

This is only relevant when resource have been added to this pool using `add()`.

Returns Generator of (meta, resource) tuples

Textures.**resolve_loader**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Attempts to assign a loader class to a ResourceDescription.

Parameters **meta** (*ResourceDescription*) – The resource description instance

16.2.2 Attributes

Textures.**settings_attr = 'TEXTURE_LOADERS'**

Textures.**count**
The number of resource descriptions added. This is only relevant when using *add* and *load_pool*.

Type int

Textures.**loaders**
Loader classes for this resource type

Type Generator

16.3 programs.Programs

moderngl_window.resources.programs.**Programs**
Handle program loading

16.3.1 Methods

Programs.**__init__**()
Initialize internal attributes

Programs.**load**(*meta: moderngl_window.meta.program.ProgramDescription*) → moderngl.program.Program
Loads a shader program with the configured loaders

Parameters **meta** (*ProgramDescription*) – The resource description

Returns The shader program

Return type moderngl.Program

Programs.**add**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Adds a resource description without loading it. The resource is loaded and returned when `load_pool()` is called.

Parameters **meta** (*ResourceDescription*) – The resource description

Programs.**load_pool**() → Generator[Tuple[moderngl_window.meta.base.ResourceDescription, Any], None, None]
Loads all the data files using the configured finders.

This is only relevant when resource have been added to this pool using `add()`.

Returns Generator of (meta, resource) tuples

Programs.**resolve_loader**(*meta: moderngl_window.meta.program.ProgramDescription*) → None
Resolve program loader.

Determines if the references resource is a single or multiple glsl files unless `kind` is specified.

Parameters **meta** (*ProgramDescription*) – The resource description

16.3.2 Attributes

Programs.**settings_attr = 'PROGRAM_LOADERS'**

Programs.**count**
The number of resource descriptions added. This is only relevant when using *add* and *load_pool*.

Type int

Programs.**loaders**
Loader classes for this resource type

Type Generator

16.4 scenes.Scenes

moderngl_window.resources.scenes.**Scenes**
Handles scene loading

16.4.1 Methods

Scenes.**__init__**()
Initialize internal attributes

Scenes.**load**(*meta: moderngl_window.meta.scene.SceneDescription*) → moderngl_window.scene.scene.Scene
Load a scene with the configured loaders.

Parameters **meta** (*SceneDescription*) – The resource description

Returns The loaded scene

Return type Scene

Scenes.**add**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Adds a resource description without loading it. The resource is loaded and returned when `load_pool()` is called.

Parameters **meta** (*ResourceDescription*) – The resource description

Scenes.**load_pool**() → Generator[Tuple[moderngl_window.meta.base.ResourceDescription, Any], None, None]
Loads all the data files using the configured finders.

This is only relevant when resource have been added to this pool using `add()`.

Returns Generator of (meta, resource) tuples

Scenes.**resolve_loader**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Attempts to assign a loader class to a ResourceDescription.

Parameters **meta** (*ResourceDescription*) – The resource description instance

16.4.2 Attributes

Scenes.**settings_attr = 'SCENE_LOADERS'**

Scenes.**count**
The number of resource descriptions added. This is only relevant when using *add* and *load_pool*.

Type int

Scenes.**loaders**
Loader classes for this resource type

Type Generator

16.5 base.DataFiles

moderngl_window.resources.data.**DataFiles**
Registry for requested data files

16.5.1 Methods

DataFiles.**__init__**()
Initialize internal attributes

DataFiles.**load**(*meta: moderngl_window.meta.data.DataDescription*) → Any
Load data file with the configured loaders.

Parameters **meta** (*DataDescription*) – the resource description

Returns The loaded resource

Return type Any

DataFiles.**add**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Adds a resource description without loading it. The resource is loaded and returned when `load_pool()` is called.

Parameters **meta** (*ResourceDescription*) – The resource description

DataFiles.**load_pool**() → Generator[Tuple[moderngl_window.meta.base.ResourceDescription, Any], None, None]
Loads all the data files using the configured finders.

This is only relevant when resource have been added to this pool using `add()`.

Returns Generator of (meta, resource) tuples

DataFiles.**resolve_loader**(*meta: moderngl_window.meta.base.ResourceDescription*) → None
Attempts to assign a loader class to a ResourceDescription.

Parameters **meta** (*ResourceDescription*) – The resource description instance

16.5.2 Attributes

DataFiles.**settings_attr = 'DATA_LOADERS'**

DataFiles.**count**
The number of resource descriptions added. This is only relevant when using *add* and *load_pool*.

Type int

DataFiles.**loaders**
Loader classes for this resource type

Type Generator

CHAPTER

SEVENTEEN

MODERNGL_WINDOW.TIMERS

17.1 base.BaseTimer

moderngl_window.timers.base.**BaseTimer**
A timer controls the time passed into the the render function. This can be used in creative ways to control the current time such as basing it on current location in an audio file.

All methods must be implemented.

17.1.1 Methods

BaseTimer.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

BaseTimer.**next_frame**() → Tuple[float, float]
Get timer information for the next frame.

Returns The frametime and current time

Return type Tuple[float, float]

BaseTimer.**start**()
Start the timer initially or resume after pause

BaseTimer.**pause**()
Pause the timer

BaseTimer.**toggle_pause**()
Toggle pause state

BaseTimer.**stop**() → Tuple[float, float]
Stop the timer. Should only be called once when stopping the timer.

Returns Tuple[float, float]> Current position in the timer, actual running duration

17.1.2 Attributes

BaseTimer.**is_paused**
The pause state of the timer

Type bool

BaseTimer.**is_running**
Is the timer currently running?

Type bool

`BaseTimer.`**`time`**
Get or set the current time. This can be used to jump around in the timeline.

Returns The current time in seconds

Return type float

17.2 clock.Timer

`moderngl_window.timers.clock.`**`Timer`**
Timer based on python `time`.

17.2.1 Methods

`Timer.`**`__init__`**`(`***kwargs*`)`
Initialize self. See help(type(self)) for accurate signature.

`Timer.`**`next_frame`**`()` → Tuple[float, float]
Get the time and frametime for the next frame. This should only be called once per frame.

Returns current time and frametime

Return type Tuple[float, float]

`Timer.`**`start`**`()`
Start the timer by recoding the current `time.time()` preparing to report the number of seconds since this timestamp.

`Timer.`**`pause`**`()`
Pause the timer by setting the internal pause time using `time.time()`

`Timer.`**`toggle_pause`**`()`
Toggle the paused state

`Timer.`**`stop`**`()` → Tuple[float, float]
Stop the timer. Should only be called once when stopping the timer.

Returns Current position in the timer, actual running duration

Return type Tuple[float, float]

17.2.2 Attributes

`Timer.`**`is_paused`**
The pause state of the timer

Type bool

`Timer.`**`is_running`**
Is the timer currently running?

Type bool

`Timer.`**`time`**
Get or set the current time. This can be used to jump around in the timeline.

Returns The current time in seconds

CHAPTER

EIGHTEEN

MODERNGL_WINDOW.UTILS

18.1 Scheduler

18.1.1 Methods

Scheduler.**__init__**(*timer: moderngl_window.timers.base.BaseTimer*)
Create a Scheduler object to handle events.

Parameters **timer** (*BaseTimer*) – timer to use, subclass of BaseTimer.

Raises **ValueError** – timer is not a valid argument.

Scheduler.**run_once**(*action*, *delay: float*, *, *priority: int = 1*, *arguments=()*, *kwargs={}*) → int
Schedule a function for execution after a delay.

Parameters

- **action** (*callable*) – function to be called.
- **delay** (*float*) – delay in seconds.
- **priority** (*int, optional*) – priority for this event, lower is more important. Defaults to 1.
- **arguments** (*tuple, optional*) – arguments for the action. Defaults to ().
- **kwargs** (*dict, optional*) – keyword arguments for the action. Defaults to dict().

Returns event id that can be canceled.

Return type int

Scheduler.**run_at**(*action*, *time: float*, *, *priority: int = 1*, *arguments=()*, *kwargs={}*) → int
Schedule a function to be executed at a certain time.

Parameters

- **action** (*callable*) – function to be called.
- **time** (*float*) – epoch time at which the function should be called.
- **priority** (*int, optional*) – priority for this event, lower is more important. Defaults to 1.
- **arguments** (*tuple, optional*) – arguments for the action. Defaults to ().
- **kwargs** (*dict, optional*) – keyword arguments for the action. Defaults to dict().

Returns event id that can be canceled.

Return type int

Scheduler.**run_every**(*action*, *delay: float*, *, *priority: int = 1*, *initial_delay: float = 0.0*, *arguments=()*, *kwargs={}*) → int

Schedule a recurring function to be called every *delay* seconds after a initial delay.

Parameters

- **action** (*callable*) – function to be called.
- **delay** (*float*) – delay in seconds.
- **priority** (*int, optional*) – priority for this event, lower is more important. Defaults to 1.
- **initial_delay** (*float, optional*) – initial delay in seconds before executing for the first time.
- **to 0. arguments** (*Defaults*) – arguments for the action. Defaults to ().
- **kwargs** (*dict, optional*) – keyword arguments for the action. Defaults to dict().

Returns event id that can be canceled.

Return type int

Scheduler.**execute**() → None

Run the scheduler without blocking and execute any expired events.

Scheduler.**cancel**(*event_id: int*, *delay: float = 0*) → None

Cancel a previously scheduled event.

Parameters

- **event_id** (*int*) – event to be canceled
- **delay** (*float, optional*) – delay before canceling the event. Defaults to 0.

CHAPTER

NINETEEN

MODERNGL_WINDOW.SCENE

19.1 Camera

moderngl_window.scene.**Camera**
Simple camera class containing projection.

```
# create a camera
camera = Camera(fov=60.0, aspect_ratio=1.0, near=1.0, far=100.0)

# Get the current camera matrix as numpy array
print(camera.matrix)

# Get projection matrix as numpy array
print(camera.projection.matrix)
```

19.1.1 Methods

Camera.**__init__**(*fov=60.0*, *aspect_ratio=1.0*, *near=1.0*, *far=100.0*)
Initialize camera using a specific projection

Keyword Arguments

- **fov** (*float*) – Field of view
- **aspect_ratio** (*float*) – Aspect ratio
- **near** (*float*) – Near plane
- **far** (*float*) – Far plane

Camera.**set_position**(*x*, *y*, *z*) → None
Set the 3D position of the camera.

Parameters

- **x** (*float*) – x position
- **y** (*float*) – y position
- **z** (*float*) – z position

Camera.**set_rotation**(*yaw*, *pitch*) → None
Set the rotation of the camera.

Parameters

- **yaw** (*float*) – yaw rotation

- **pitch** (*float*) – pitch rotation

Camera.**look_at**(*vec=None*, *pos=None*) → numpy.ndarray
Look at a specific point

Either `vec` or `pos` needs to be supplied.

Keyword Arguments

- **vec** (*pyrr.Vector3*) – position
- **pos** (*tuple/list*) – list of tuple `[x, y, x]` / `(x, y, x)`

Returns Camera matrix

Return type numpy.ndarray

19.1.2 Attributes

Camera.**pitch**
The current pitch angle.

Type float

Camera.**yaw**
The current yaw angle.

Type float

Camera.**matrix**
The current view matrix for the camera

Type numpy.ndarray

Camera.**projection**
The 3D projection

Type *Projection3D*

19.2 KeyboardCamera

moderngl_window.scene.**KeyboardCamera**
Camera controlled by mouse and keyboard. The class interacts with the key constants in the built in window types.

Creating a keyboard camera:

```
camera = KeyboardCamera(
    self.wnd.keys,
    fov=75.0,
    aspect_ratio=self.wnd.aspect_ratio,
    near=0.1,
    far=1000.0,
)
```

We can also interact with the belonging *Projection3D* instance.

```
# Update aspect ratio
camera.projection.update(aspect_ratio=1.0)

# Get projection matrix in bytes (f4)
camera.projection.tobytes()
```

19.2.1 Methods

KeyboardCamera.**__init__**(*keys: moderngl_window.context.base.keys.BaseKeys*, *fov=60.0*, *aspect_ratio=1.0*, *near=1.0*, *far=100.0*)

Initialize the camera

Parameters **keys** (*BaseKeys*) – The key constants for the current window type

Keyword Arguments

- **fov** (*float*) – Field of view
- **aspect_ratio** (*float*) – Aspect ratio
- **near** (*float*) – near plane
- **far** (*float*) – far plane

KeyboardCamera.**key_input**(*key*, *action*, *modifiers*) → None

Process key inputs and move camera

Parameters

- **key** – The key
- **action** – key action release/press
- **modifiers** – key modifier states such as ctrl or shit

KeyboardCamera.**set_position**(*x*, *y*, *z*) → None

Set the 3D position of the camera.

Parameters

- **x** (*float*) – x position
- **y** (*float*) – y position
- **z** (*float*) – z position

KeyboardCamera.**set_rotation**(*yaw*, *pitch*) → None

Set the rotation of the camera.

Parameters

- **yaw** (*float*) – yaw rotation
- **pitch** (*float*) – pitch rotation

KeyboardCamera.**look_at**(*vec=None*, *pos=None*) → numpy.ndarray

Look at a specific point

Either `vec` or `pos` needs to be supplied.

Keyword Arguments

- **vec** (*pyrr.Vector3*) – position
- **pos** (*tuple/list*) – list of tuple `[x, y, x]` / `(x, y, x)`

Returns Camera matrix

Return type numpy.ndarray

KeyboardCamera.**move_left**(*activate*) → None
The camera should be continiously moving to the left.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_right**(*activate*) → None
The camera should be continiously moving to the right.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_forward**(*activate*) → None
The camera should be continiously moving forward.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_backward**(*activate*) → None
The camera should be continiously moving backwards.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_up**(*activate*) → None
The camera should be continiously moving up.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_down**(*activate*)
The camera should be continiously moving down.

Parameters **activate** (*bool*) – Activate or deactivate this state

KeyboardCamera.**move_state**(*direction*, *activate*) → None
Set the camera position move state.

Parameters

- **direction** – What direction to update
- **activate** – Start or stop moving in the direction

KeyboardCamera.**rot_state**(*dx: int*, *dy: int*) → None
Update the rotation of the camera.

This is done by passing in the relative mouse movement change on x and y (delta x, delta y).

In the past this method took the viewport position of the mouse. This does not work well when mouse exclusivity mode is enabled.

Parameters

- **dx** – Relative mouse position change on x
- **dy** – Relative mouse position change on y

19.2.2 Attributes

KeyboardCamera.**pitch**
The current pitch angle.

Type float

KeyboardCamera.**yaw**
The current yaw angle.

Type float

KeyboardCamera.**matrix**
The current view matrix for the camera

Type numpy.ndarray

KeyboardCamera.**mouse_sensitivity**
Mouse sensitivity (rotation speed).

This property can also be set:

```
camera.mouse_sensitivity = 2.5
```

Type float

KeyboardCamera.**velocity**
The speed this camera move based on key inputs

The property can also be modified:

```
camera.velocity = 5.0
```

Type float

KeyboardCamera.**projection**
The 3D projection

Type *Projection3D*

19.3 Scene

19.3.1 Methods

Scene.**__init__**(*name*, ***kwargs*)
Create a scene with a name.

Parameters **name** (*str*) – Unique name or path for the scene

Scene.**draw**(*projection_matrix: numpy.ndarray = None*, *camera_matrix: numpy.ndarray = None*, *time=0.0*) → None
Draw all the nodes in the scene.

Parameters

- **projection_matrix** (*ndarray*) – projection matrix (bytes)
- **camera_matrix** (*ndarray*) – camera_matrix (bytes)
- **time** (*float*) – The current time

Scene.**draw_bbox**(*projection_matrix=None*, *camera_matrix=None*, *children=True*, *color=(0.75*, *0.75*, *0.75)*) → None
Draw scene and mesh bounding boxes.

Parameters

- **projection_matrix** (*ndarray*) – mat4 projection
- **camera_matrix** (*ndarray*) – mat4 camera matrix

- **children** (*bool*) – Will draw bounding boxes for meshes as well
- **color** (*tuple*) – Color of the bounding boxes

Scene.**draw_wireframe**(*projection_matrix=None*, *camera_matrix=None*, *color=(0.75*, *0.75*, *0.75*, *1.0)*)
Render the scene in wireframe mode.

Parameters

- **projection_matrix** (*ndarray*) – mat4 projection
- **camera_matrix** (*ndarray*) – mat4 camera matrix
- **children** (*bool*) – Will draw bounding boxes for meshes as well
- **color** (*tuple*) – Color of the wireframes

Scene.**apply_mesh_programs**(*mesh_programs=None*, *clear: bool = True*) → None
Applies mesh programs to meshes. If not mesh programs are passed in we assign default ones.

Parameters

- **mesh_programs** (*list*) – List of mesh programs to assign
- **clear** (*bool*) – Clear all assigned mesh programs

Scene.**calc_scene_bbox**() → None
Calculate scene bbox

Scene.**find_material**(*name: str = None*) → Material
Finds a *Material*

Keyword Arguments **name** (*str*) – Case sensitive material name

Returns A *Material* or None

Scene.**find_node**(*name: str = None*) → Node
Finds a *Node*

Keyword Arguments **name** (*str*) – Case sensitive name

Returns A *Node* or None if not found.

Scene.**prepare**() → None
prepare the scene for rendering.

Calls apply_mesh_programs() assigning default meshprograms if needed and sets the model matrix.

Scene.**destroy**() → None
Destroys the scene data and vertex buffers

Scene.**release**()
Destroys the scene data and vertex buffers

19.3.2 Attributes

Scene.**ctx**
The current context

Type moderngl.Context

Scene.**matrix**
The current model matrix

This property is settable.

Type numpy.ndarray

19.4 Node

moderngl_window.scene.**Node**
A generic scene node containing a mesh or camera and/or a container for other nodes. Nodes and their children represents the scene tree.

19.4.1 Methods

Node.**__init__**(*name=None*, *camera=None*, *mesh=None*, *matrix=None*)
Create a node.

Keyword Arguments

- **name** – Name of the node
- **camera** – Camera to store in the node
- **mesh** – Mesh to store in the node
- **matrix** – The node's matrix

Node.**add_child**(*node*)
Add a child to this node

Parameters **node** (*Node*) – Node to add as a child

Node.**draw**(*projection_matrix=None*, *camera_matrix=None*, *time=0*)
Draw node and children.

Keyword Arguments

- **projection_matrix** (*bytes*) – projection matrix
- **camera_matrix** (*bytes*) – camera_matrix
- **time** (*float*) – The current time

Node.**draw_bbox**(*projection_matrix*, *camera_matrix*, *program*, *vao*)
Draw bounding box around the node and children.

Keyword Arguments

- **projection_matrix** (*bytes*) – projection matrix
- **camera_matrix** (*bytes*) – camera_matrix
- **program** (*moderngl.Program*) – The program to render the bbox
- **vao** – The vertex array representing the bounding box

Node.**draw_wireframe**(*projection_matrix*, *camera_matrix*, *program*)
Render the node as wireframe.

Keyword Arguments

- **projection_matrix** (*bytes*) – projection matrix
- **camera_matrix** (*bytes*) – camera_matrix
- **program** (*moderngl.Program*) – The program to render wireframe

Node.**calc_global_bbox**(*view_matrix*, *bbox_min*, *bbox_max*)
Recursive calculation of scene bbox.

Keyword Arguments

- **view_matrix** (*numpy.ndarray*) – view matrix
- **bbox_min** – min bbox values
- **bbox_max** – max bbox values

Node.**calc_model_mat**(*model_matrix*)
Calculate the model matrix related to all parents.

Parameters **model_matrix** (*numpy.ndarray*) – model matrix

19.4.2 Attributes

Node.**name**
Get or set the node name

Type str

Node.**mesh**
The mesh if present

Type *Mesh*

Node.**camera**
The camera if present

Type *Camera*

Node.**matrix**
Note matrix (local)

Type numpy.ndarray

Node.**matrix_global**
The global node matrix containing transformations from parent nodes

Type numpy.ndarray

Node.**children**
List of children

Type list

19.5 Mesh

moderngl_window.scene.**Mesh = <class 'moderngl_window.scene.mesh.Mesh'>**
Mesh info and geometry

19.5.1 Methods

Mesh.**__init__**(*name*, *vao=None*, *material=None*, *attributes=None*, *bbox_min=None*, *bbox_max=None*)
Initialize mesh.

Parameters **name** (*str*) – name of the mesh

Keyword Arguments

- **vao** (VAO) – geometry
- **material** (Material) – material for the mesh
- **attributes** (*dict*) – Details info about each mesh attribute (dict)
- **bbox_min** – xyz min values
- **bbox_max** – xyz max values

Attributes example:

```
{
    "NORMAL": {"name": "in_normal", "components": 3, "type": GL_FLOAT},
    "POSITION": {"name": "in_position", "components": 3, "type": GL_FLOAT}
}
```

Mesh.**draw**(*projection_matrix=None*, *model_matrix=None*, *camera_matrix=None*, *time=0.0*)
Draw the mesh using the assigned mesh program

Keyword Arguments

- **projection_matrix** (*bytes*) – projection_matrix
- **view_matrix** (*bytes*) – view_matrix
- **camera_matrix** (*bytes*) – camera_matrix

Mesh.**draw_bbox**(*proj_matrix*, *model_matrix*, *cam_matrix*, *program*, *vao*)
Renders the bounding box for this mesh.

Parameters

- **proj_matrix** – Projection matrix
- **model_matrix** – View/model matrix
- **cam_matrix** – Camera matrix
- **program** – The moderngl.Program rendering the bounding box
- **vao** – The vao mesh for the bounding box

Mesh.**draw_wireframe**(*proj_matrix*, *model_matrix*, *program*)
Render the mesh as wireframe.

proj_matrix: Projection matrix model_matrix: View/model matrix program: The moderngl.Program rendering the wireframe

Mesh.**add_attribute**(*attr_type*, *name*, *components*)
Add metadata about the mesh :param attr_type: POSITION, NORMAL etc :param name: The attribute name used in the program :param components: Number of floats

Mesh.**calc_global_bbox**(*view_matrix*, *bbox_min*, *bbox_max*)
Calculates the global bounding.

Parameters

- **view_matrix** – View matrix
- **bbox_min** – xyz min
- **bbox_max** – xyz max

Returns Combined bbox

Return type bbox_min, bbox_max

`Mesh.`**`has_normals`**`()` → bool

Returns Does the mesh have a normals?

Return type bool

`Mesh.`**`has_uvs`**`(`*layer=0*`)` → bool

Returns Does the mesh have texture coordinates?

Return type bool

19.6 Material

`moderngl_window.scene.`**`Material`**

Generic material

19.6.1 Methods

`Material.`**`__init__`**`(`*name: str = None*`)`

Initialize material.

Parameters **`name`** (*`str`*) – Name of the material

`Material.`**`release`**`()`

19.6.2 Attributes

`Material.`**`name`**

Name of the material

Type str

`Material.`**`color`**

RGBA color

Type Tuple[float, float, float, float]

`Material.`**`mat_texture`**

instance

Type MaterialTexture

`Material.`**`double_sided`**

Material surface is double sided?

Type bool

19.7 MaterialTexture

`moderngl_window.scene.`**`MaterialTexture`**

Wrapper for textures used in materials. Contains a texture and a sampler object.

19.7.1 Methods

MaterialTexture.**__init__**(*texture: moderngl.texture.Texture = None*, *sampler: moderngl.sampler.Sampler = None*)

Initialize instance.

Parameters

- **texture** (*moderngl.Texture*) – Texture instance
- **sampler** (*moderngl.Sampler*) – Sampler instance

19.7.2 Attributes

MaterialTexture.**texture**

Texture instance

Type moderngl.Texture

MaterialTexture.**sampler**

Sampler instance

Type moderngl.Sampler

19.8 MeshProgram

moderngl_window.scene.**MeshProgram**

Describes how a mesh is rendered using a specific shader program

19.8.1 Methods

MeshProgram.**__init__**(*program: moderngl.program.Program = None*, ***kwargs*)

Initialize.

Parameters **program** – The moderngl program

MeshProgram.**draw**(*mesh*, *projection_matrix: numpy.ndarray = None*, *model_matrix: numpy.ndarray = None*, *camera_matrix: numpy.ndarray = None*, *time=0.0*)

Draw code for the mesh

Parameters **mesh** (*Mesh*) – The mesh to render

Keyword Arguments

- **projection_matrix** (*numpy.ndarray*) – projection_matrix (bytes)
- **model_matrix** (*numpy.ndarray*) – view_matrix (bytes)
- **camera_matrix** (*numpy.ndarray*) – camera_matrix (bytes)
- **time** (*float*) – The current time

MeshProgram.**apply**(*mesh*)

Determine if this MeshProgram should be applied to the mesh. Can return self or some MeshProgram instance to support dynamic MeshProgram creation

Parameters **mesh** – The mesh to inspect

19.8.2 Attributes

`MeshProgram.`**`ctx`**
The current context

Type moderngl.Context

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

S

T

U

V

W

Y