File 1 - base.py:

```
1: (0)              from typing import Any
2: (0)              from PIL.Image import Image
3: (0)              class BaseImage:
4: (4)                  """
5: (4)                  Base class for atlas images.
6: (4)                  Must be hashable for use as dict keys and in sets
7: (4)                  and decides the uniqueness of the image.
8: (4)                  """
9: (4)                  def __init__(self, image: Any):
10: (8)                     self._image = image
11: (4)                 @property
12: (4)                 def width(self) -> int:
13: (8)                     """int: Width of the image in pixels"""
14: (8)                     raise NotImplementedError
15: (4)                 @property
16: (4)                 def height(self) -> int:
17: (8)                     """int: Height of the image in pixels"""
18: (8)                     raise NotImplementedError
19: (4)                 @property
20: (4)                 def size(self) -> tuple[int, int]:
21: (8)                     """tuple[int, int]: Size of the image in pixels (width, height)"""
22: (8)                     raise NotImplementedError
23: (4)                 def get_pixel_data(self, components: int = 4) -> bytes:
24: (8)                     """
25: (8)                     Get the raw pixel data from the image.
26: (8)                     Keyword Args:
27: (12)                        components: Number of components to get
28: (8)                     """
29: (8)                     raise NotImplementedError
30: (4)                 def __hash__(self) -> int:
31: (8)                     return id(self)
32: (0)             class AtlasImage(BaseImage):
33: (4)                 """An atlas image using Pillow"""
34: (4)                 def __init__(self, image: Image):
35: (8)                     self._image = image
36: (4)                 @property
37: (4)                 def width(self) -> int:
38: (8)                     return self._image.width
39: (4)                 @property
40: (4)                 def height(self) -> int:
41: (8)                     return self._image.height
42: (4)                 @property
43: (4)                 def size(self) -> tuple[int, int]:
44: (8)                     return self._image.size
45: (4)                 def get_pixel_data(self, components: int = 4) -> bytes:
46: (8)                     """
47: (8)                     Get the raw pixel data from the image.
48: (8)                     Keyword Args:
49: (12)                        components: Number of components to get
50: (8)                     """
51: (8)                     if components == 4:
52: (12)                        return self._image.convert("RGBA").tobytes()
53: (8)                     elif components == 3:
54: (12)                        return self._image.covert("RGB").tobytes()
55: (8)                     else:
56: (12)                        raise ValueError("Only supports 3 or 4 components")
```

File 2 - base.py:

```
1: (0)              import datetime
2: (0)              import os
3: (0)              from typing import Any, Optional, Union
4: (0)              import moderngl
5: (0)              from moderngl_window.timers.clock import Timer
6: (0)              class BaseVideoCapture:
```

```
7: (4)                  """
8: (4)                  ``BaseVideoCapture`` is a base class to video capture
9: (4)                  Args:
10: (8)                     source (moderngl.Texture, moderngl.Framebuffer): the source of the capture
11: (8)                     framerate (int, float) : the framerate of the video, by thefault is 60 fps
12: (4)                 if the source is texture there are some requirements:
13: (8)                     - dtype = 'f1';
14: (8)                     - components >= 3.
15: (4)                 """
16: (4)                 def __init__(
17: (8)                     self,
18: (8)                     source: Union[moderngl.Texture, moderngl.Framebuffer],
19: (8)                     framerate: Union[int, float] = 60,
20: (4)                 ):
21: (8)                     self._source = source
22: (8)                     self._framerate = framerate
23: (8)                     self._recording: Optional[bool] = False
24: (8)                     self._last_time: float = 0.0
25: (8)                     self._filename: str = ""
26: (8)                     self._width: Optional[int] = None
27: (8)                     self._height: Optional[int] = None
28: (8)                     self._timer = Timer()
29: (8)                     self._components: int = 0  # for textures
30: (8)                     if isinstance(self._source, moderngl.Texture):
31: (12)                        self._components = self._source.components
32: (4)                 def _dump_frame(self, frame: Any) -> None:
33: (8)                     """
34: (8)                     custom function called during self.save()
35: (8)                     Args:
36: (12)                        frame: frame data in bytes
37: (8)                     """
38: (8)                     raise NotImplementedError("override this function")
39: (4)                 def _start_func(self) -> bool:
40: (8)                     """
41: (8)                     custom function called during self.start_capture()
42: (8)                     must return a True if this function complete without errors
43: (8)                     """
44: (8)                     raise NotImplementedError("override this function")
45: (4)                 def _release_func(self) -> None:
46: (8)                     """
47: (8)                     custom function called during self.release()
48: (8)                     """
49: (8)                     raise NotImplementedError("override this function")
50: (4)                 def _get_wh(self) -> tuple[int, int]:
51: (8)                     """
52: (8)                     Return a tuple of the width and the height of the source
53: (8)                     """
54: (8)                     return self._source.width, self._source.height
55: (4)                 def _remove_file(self) -> None:
56: (8)                     """Remove the filename of the video is it exist"""
57: (8)                     if os.path.exists(self._filename):
58: (12)                        os.remove(self._filename)
59: (4)                 def start_capture(
60: (8)                     self, filename: Optional[str] = None, framerate: Union[int, float] = 60
61: (4)                 ) -> None:
62: (8)                     """
63: (8)                     Start the capturing process
64: (8)                     Args:
65: (12)                        filename (str): name of the output file
66: (12)                        framerate (int, float): framerate of the video
67: (8)                     if filename is not specified it will be generated based
68: (8)                     on the datetime.
69: (8)                     """
70: (8)                     if self._recording:
71: (12)                        print("Capturing is already started")
72: (12)                        return
```

```
        if isinstance(self._source, moderngl.Texture):
            if self._source.dtype != "f1":
                print("source type: moderngl.Texture must be type `f1` ")
                return
            if self._components < 3:
                print("source type: moderngl.Texture must have at least 3 components")
                return
        if self._source is None:
            print("No source defined, there is nothing to record")
            return
        if not filename:
            now = datetime.datetime.now()
            filename = f"video_{now:%Y%m%d_%H%M%S}.mp4"
        self._filename = filename
        self._framerate = framerate
        self._width, self._height = self._get_wh()
        if not self._start_func():
            self.release()
            print("Capturing failed")
            return
        self._timer.start()
        self._last_time = self._timer.time
        self._recording = True
    def save(self) -> None:
        """
        Save function to call at the end of render function
        """
        if not self._recording:
            return
        if self._source is None:
            return
        dt = 1.0 / self._framerate
        if self._timer.time - self._last_time > dt:
            self._last_time = self._timer.time
            if isinstance(self._source, moderngl.Framebuffer):
                frame = self._source.read(components=3)
                self._dump_frame(frame)
            else:
                frame = self._source.read()
                self._dump_frame(frame)
    def release(self) -> None:
        """
        Stop the recording process
        """
        if self._recording:
            self._release_func()
            self._timer.stop()
            print(f"Video file succesfully saved as {self._filename}")
        self._recording = None
```

File 3 - keys.py:

```
from typing import Any
class KeyModifiers:
    """Namespace for storing key modifiers"""
    shift: Any = False
    ctrl: Any = False
    alt: Any = False
    def __repr__(self) -> str:
        return str(self)
    def __str__(self) -> str:
        return "<KeyModifiers shift={} ctrl={} alt={}>".format(self.shift, self.ctrl, self.alt)
class BaseKeys:
    """
    Namespace for mapping key constants.
```

```
14: (4)                 This is simply a template for what keys should be mapped for all window
libraries
15: (4)                 """
16: (4)                 ACTION_PRESS: Any = "ACTION_PRESS"
17: (4)                 ACTION_RELEASE: Any = "ACTION_RELEASE"
18: (4)                 ESCAPE: Any = "undefined"
19: (4)                 SPACE: Any = "undefined"
20: (4)                 ENTER: Any = "undefined"
21: (4)                 PAGE_UP: Any = "undefined"
22: (4)                 PAGE_DOWN: Any = "undefined"
23: (4)                 LEFT: Any = "undefined"
24: (4)                 RIGHT: Any = "undefined"
25: (4)                 UP: Any = "undefined"
26: (4)                 DOWN: Any = "undefined"
27: (4)                 LEFT_SHIFT: Any = "undefined"
28: (4)                 RIGHT_SHIFT: Any = "undefined"
29: (4)                 LEFT_CTRL: Any = "undefined"
30: (4)                 TAB: Any = "undefined"
31: (4)                 COMMA: Any = "undefined"
32: (4)                 MINUS: Any = "undefined"
33: (4)                 PERIOD: Any = "undefined"
34: (4)                 SLASH: Any = "undefined"
35: (4)                 SEMICOLON: Any = "undefined"
36: (4)                 EQUAL: Any = "undefined"
37: (4)                 LEFT_BRACKET: Any = "undefined"
38: (4)                 RIGHT_BRACKET: Any = "undefined"
39: (4)                 BACKSLASH: Any = "undefined"
40: (4)                 BACKSPACE: Any = "undefined"
41: (4)                 INSERT: Any = "undefined"
42: (4)                 DELETE: Any = "undefined"
43: (4)                 HOME: Any = "undefined"
44: (4)                 END: Any = "undefined"
45: (4)                 CAPS_LOCK: Any = "undefined"
46: (4)                 F1: Any = "undefined"
47: (4)                 F2: Any = "undefined"
48: (4)                 F3: Any = "undefined"
49: (4)                 F4: Any = "undefined"
50: (4)                 F5: Any = "undefined"
51: (4)                 F6: Any = "undefined"
52: (4)                 F7: Any = "undefined"
53: (4)                 F8: Any = "undefined"
54: (4)                 F9: Any = "undefined"
55: (4)                 F10: Any = "undefined"
56: (4)                 F11: Any = "undefined"
57: (4)                 F12: Any = "undefined"
58: (4)                 NUMBER_0: Any = "undefined"
59: (4)                 NUMBER_1: Any = "undefined"
60: (4)                 NUMBER_2: Any = "undefined"
61: (4)                 NUMBER_3: Any = "undefined"
62: (4)                 NUMBER_4: Any = "undefined"
63: (4)                 NUMBER_5: Any = "undefined"
64: (4)                 NUMBER_6: Any = "undefined"
65: (4)                 NUMBER_7: Any = "undefined"
66: (4)                 NUMBER_8: Any = "undefined"
67: (4)                 NUMBER_9: Any = "undefined"
68: (4)                 NUMPAD_0: Any = "undefined"
69: (4)                 NUMPAD_1: Any = "undefined"
70: (4)                 NUMPAD_2: Any = "undefined"
71: (4)                 NUMPAD_3: Any = "undefined"
72: (4)                 NUMPAD_4: Any = "undefined"
73: (4)                 NUMPAD_5: Any = "undefined"
74: (4)                 NUMPAD_6: Any = "undefined"
75: (4)                 NUMPAD_7: Any = "undefined"
76: (4)                 NUMPAD_8: Any = "undefined"
77: (4)                 NUMPAD_9: Any = "undefined"
78: (4)                 A: Any = "undefined"
79: (4)                 B: Any = "undefined"
80: (4)                 C: Any = "undefined"
81: (4)                 D: Any = "undefined"
```

```
82: (4)                 E: Any = "undefined"
83: (4)                 F: Any = "undefined"
84: (4)                 G: Any = "undefined"
85: (4)                 H: Any = "undefined"
86: (4)                 I: Any = "undefined"
87: (4)                 J: Any = "undefined"
88: (4)                 K: Any = "undefined"
89: (4)                 L: Any = "undefined"
90: (4)                 M: Any = "undefined"
91: (4)                 N: Any = "undefined"
92: (4)                 O: Any = "undefined"
93: (4)                 P: Any = "undefined"
94: (4)                 Q: Any = "undefined"
95: (4)                 R: Any = "undefined"
96: (4)                 S: Any = "undefined"
97: (4)                 T: Any = "undefined"
98: (4)                 U: Any = "undefined"
99: (4)                 V: Any = "undefined"
100: (4)                W: Any = "undefined"
101: (4)                X: Any = "undefined"
102: (4)                Y: Any = "undefined"
103: (4)                Z: Any = "undefined"

----------------------------------------

File 4 - keys.py:

1: (0)              import glfw
2: (0)              from moderngl_window.context.base import BaseKeys
3: (0)              GLFW_key = int
4: (0)              class Keys(BaseKeys):
5: (4)                  """
6: (4)                  Namespace defining glfw specific keys constants
7: (4)                  """
8: (4)                  ACTION_PRESS = glfw.PRESS
9: (4)                  ACTION_RELEASE = glfw.RELEASE
10: (4)                 ESCAPE = glfw.KEY_ESCAPE
11: (4)                 SPACE = glfw.KEY_SPACE
12: (4)                 ENTER = glfw.KEY_ENTER
13: (4)                 PAGE_UP = glfw.KEY_PAGE_UP
14: (4)                 PAGE_DOWN = glfw.KEY_PAGE_DOWN
15: (4)                 LEFT = glfw.KEY_LEFT
16: (4)                 RIGHT = glfw.KEY_RIGHT
17: (4)                 UP = glfw.KEY_UP
18: (4)                 DOWN = glfw.KEY_DOWN
19: (4)                 TAB = glfw.KEY_TAB
20: (4)                 COMMA = glfw.KEY_COMMA
21: (4)                 MINUS = glfw.KEY_MINUS
22: (4)                 PERIOD = glfw.KEY_PERIOD
23: (4)                 SLASH = glfw.KEY_SLASH
24: (4)                 SEMICOLON = glfw.KEY_SEMICOLON
25: (4)                 EQUAL = glfw.KEY_EQUAL
26: (4)                 LEFT_BRACKET = glfw.KEY_LEFT_BRACKET
27: (4)                 RIGHT_BRACKET = glfw.KEY_RIGHT_BRACKET
28: (4)                 BACKSLASH = glfw.KEY_BACKSLASH
29: (4)                 BACKSPACE = glfw.KEY_BACKSPACE
30: (4)                 INSERT = glfw.KEY_INSERT
31: (4)                 DELETE = glfw.KEY_DELETE
32: (4)                 HOME = glfw.KEY_HOME
33: (4)                 END = glfw.KEY_END
34: (4)                 CAPS_LOCK = glfw.KEY_CAPS_LOCK
35: (4)                 F1 = glfw.KEY_F1
36: (4)                 F2 = glfw.KEY_F2
37: (4)                 F3 = glfw.KEY_F3
38: (4)                 F4 = glfw.KEY_F4
39: (4)                 F5 = glfw.KEY_F5
40: (4)                 F6 = glfw.KEY_F6
41: (4)                 F7 = glfw.KEY_F7
42: (4)                 F8 = glfw.KEY_F8
```

```
43: (4)                 F9 = glfw.KEY_F9
44: (4)                 F10 = glfw.KEY_F10
45: (4)                 F11 = glfw.KEY_F11
46: (4)                 F12 = glfw.KEY_F12
47: (4)                 NUMBER_0 = glfw.KEY_0
48: (4)                 NUMBER_1 = glfw.KEY_1
49: (4)                 NUMBER_2 = glfw.KEY_2
50: (4)                 NUMBER_3 = glfw.KEY_3
51: (4)                 NUMBER_4 = glfw.KEY_4
52: (4)                 NUMBER_5 = glfw.KEY_5
53: (4)                 NUMBER_6 = glfw.KEY_6
54: (4)                 NUMBER_7 = glfw.KEY_7
55: (4)                 NUMBER_8 = glfw.KEY_8
56: (4)                 NUMBER_9 = glfw.KEY_9
57: (4)                 NUMPAD_0 = glfw.KEY_KP_0
58: (4)                 NUMPAD_1 = glfw.KEY_KP_1
59: (4)                 NUMPAD_2 = glfw.KEY_KP_2
60: (4)                 NUMPAD_3 = glfw.KEY_KP_3
61: (4)                 NUMPAD_4 = glfw.KEY_KP_4
62: (4)                 NUMPAD_5 = glfw.KEY_KP_5
63: (4)                 NUMPAD_6 = glfw.KEY_KP_6
64: (4)                 NUMPAD_7 = glfw.KEY_KP_7
65: (4)                 NUMPAD_8 = glfw.KEY_KP_8
66: (4)                 NUMPAD_9 = glfw.KEY_KP_9
67: (4)                 A = glfw.KEY_A
68: (4)                 B = glfw.KEY_B
69: (4)                 C = glfw.KEY_C
70: (4)                 D = glfw.KEY_D
71: (4)                 E = glfw.KEY_E
72: (4)                 F = glfw.KEY_F
73: (4)                 G = glfw.KEY_G
74: (4)                 H = glfw.KEY_H
75: (4)                 I = glfw.KEY_I
76: (4)                 J = glfw.KEY_J
77: (4)                 K = glfw.KEY_K
78: (4)                 L = glfw.KEY_L
79: (4)                 M = glfw.KEY_M
80: (4)                 N = glfw.KEY_N
81: (4)                 O = glfw.KEY_O
82: (4)                 P = glfw.KEY_P
83: (4)                 Q = glfw.KEY_Q
84: (4)                 R = glfw.KEY_R
85: (4)                 S = glfw.KEY_S
86: (4)                 T = glfw.KEY_T
87: (4)                 U = glfw.KEY_U
88: (4)                 V = glfw.KEY_V
89: (4)                 W = glfw.KEY_W
90: (4)                 X = glfw.KEY_X
91: (4)                 Y = glfw.KEY_Y
92: (4)                 Z = glfw.KEY_Z

----------------------------------------

File 5 - keys.py:

1: (0)              from moderngl_window.context.base import BaseKeys
2: (0)              class Keys(BaseKeys):
3: (4)                  pass

----------------------------------------

File 6 - ffmpeg.py:

1: (0)              import subprocess
2: (0)              from typing import Any, Optional
3: (0)              import moderngl
4: (0)              from .base import BaseVideoCapture
5: (0)              class FFmpegCapture(BaseVideoCapture):
6: (4)                  """
```

```
7: (4)                  ``FFmpegCapture`` it's an utility class to capture runtime render
8: (4)                  and save it as video.
9: (4)                  Args:
10: (4)                 Example:
11: (4)                 .. code:: python
12: (8)                     import moderngl_window
13: (8)                     from moderngl_window.capture.ffmpeg import FFmpegCapture
14: (8)                     class CaptureTest(modenrgl_window.WindowConfig):
15: (12)                        def __init__(self, **kwargs):
16: (16)                            super().__init__(**kwargs)
17: (16)                            self.cap = FFmpegCapture(source=self.wnd.fbo)
18: (16)                            self.cap.start_capture(
19: (20)                                filename="video.mp4",
20: (20)                                framerate=30
21: (16)                            )
22: (12)                        def render(self, time, frametime):
23: (16)                            self.cap.save()
24: (12)                        def close(self):
25: (16)                            self.cap.release()
26: (4)                 """
27: (4)                 def __init__(self, **kwargs: Any) -> None:
28: (8)                     super().__init__(**kwargs)
29: (8)                     self._ffmpeg: Optional[subprocess.Popen[bytes]] = None
30: (4)                 def _start_func(self) -> bool:
31: (8)                     """
32: (8)                     choose the right pixel format based on the number of components
33: (8)                     and start a ffmpeg pipe with a subprocess.
34: (8)                     """
35: (8)                     pix_fmt = "rgb24"  # 3 component, 1 byte per color -> 24 bit
36: (8)                     if isinstance(self._source, moderngl.Texture) and self._components ==
4:
37: (12)                        pix_fmt = "rgba"  # 4 component , 1 byte per color -> 32 bit
38: (8)                     command = [
39: (12)                        "ffmpeg",
40: (12)                        "-hide_banner",
41: (12)                        "-loglevel",
42: (12)                        "error",
43: (12)                        "-stats",  # less verbose, only stats of recording
44: (12)                        "-y",  # (optional) overwrite output file if it exists
45: (12)                        "-f",
46: (12)                        "rawvideo",
47: (12)                        "-vcodec",
48: (12)                        "rawvideo",
49: (12)                        "-s",
50: (12)                        f"{self._width}x{self._height}",  # size of one frame
51: (12)                        "-pix_fmt",
52: (12)                        pix_fmt,
53: (12)                        "-r",
54: (12)                        f"{self._framerate}",  # frames per second
55: (12)                        "-i",
56: (12)                        "-",  # The imput comes from a pipe
57: (12)                        "-vf",
58: (12)                        "vflip",
59: (12)                        "-an",  # Tells FFMPEG not to expect any audio
60: (12)                        self._filename,
61: (8)                     ]
62: (8)                     try:
63: (12)                        self._ffmpeg = subprocess.Popen(command, stdin=subprocess.PIPE,
bufsize=0)
64: (8)                     except FileNotFoundError:
65: (12)                        print("ffmpeg command not found. Be sure to add it to PATH")
66: (12)                        return False
67: (8)                     return True
68: (4)                 def _release_func(self) -> None:
69: (8)                     """
70: (8)                     Safely release the capture
71: (8)                     """
72: (8)                     if (self._ffmpeg is None) or (self._ffmpeg.stdin is None):
73: (12)                        return
```

```
74: (8)                     self._ffmpeg.stdin.close()
75: (8)                     _ = self._ffmpeg.wait()
76: (4)                 def _dump_frame(self, frame: Any) -> None:
77: (8)                     """
78: (8)                     write the frame data in to the ffmpeg pipe
79: (8)                     """
80: (8)                     if (self._ffmpeg is None) or (self._ffmpeg.stdin is None):
81: (12)                        return
82: (8)                     self._ffmpeg.stdin.write(frame)

----------------------------------------

File 7 - window.py:

1: (0)              import logging
2: (0)              import sys
3: (0)              import weakref
4: (0)              from argparse import ArgumentParser, Namespace
5: (0)              from functools import wraps
6: (0)              from pathlib import Path
7: (0)              from typing import Any, Callable, Optional, Union
8: (0)              import moderngl
9: (0)              from moderngl_window import resources
10: (0)             from moderngl_window.context.base import BaseKeys, KeyModifiers
11: (0)             from moderngl_window.geometry.attributes import AttributeNames
12: (0)             from moderngl_window.loaders.texture.icon import IconLoader
13: (0)             from moderngl_window.meta import (
14: (4)                 DataDescription,
15: (4)                 ProgramDescription,
16: (4)                 SceneDescription,
17: (4)                 TextureDescription,
18: (0)             )
19: (0)             from moderngl_window.scene import Scene
20: (0)             from moderngl_window.timers import BaseTimer, Timer
21: (0)             FuncAny = Callable[[Any], Any]
22: (0)             logger = logging.getLogger(__name__)
23: (0)             def require_callable(func: Callable[[Any], Any]) -> Callable[[Any], Any]:
24: (4)                 """Decorator ensuring assigned callbacks are valid callables"""
25: (4)                 @wraps(func)
26: (4)                 def wrapper(*args: Any, **kwargs: Any) -> Any:
27: (8)                     if not callable(args[1]):
28: (12)                        raise ValueError("{} is not a callable".format(args[1]))
29: (8)                     return func(*args, **kwargs)
30: (4)                 return wrapper
31: (0)             class MouseButtons:
32: (4)                 """Maps what button id to a name"""
33: (4)                 left = 1
34: (4)                 right = 2
35: (4)                 middle = 3
36: (0)             class MouseButtonStates:
37: (4)                 """Namespace for storing the current mouse button states"""
38: (4)                 left = False
39: (4)                 right = False
40: (4)                 middle = False
41: (4)                 @property
42: (4)                 def any(self) -> bool:
43: (8)                     """bool: if any mouse buttons are pressed"""
44: (8)                     return self.left or self.right or self.middle
45: (4)                 def __repr__(self) -> str:
46: (8)                     return str(self)
47: (4)                 def __str__(self) -> str:
48: (8)                     return "<MouseButtonStates left={} right={} middle={}".format(
49: (12)                        self.left, self.right, self.middle
50: (8)                     )
51: (0)             class BaseWindow:
52: (4)                 """
53: (4)                 Helper base class for a generic window implementation
54: (4)                 """
55: (4)                 name = "base"
```

```
(4)                     keys = BaseKeys
(4)                     mouse = MouseButtons
(4)                     def __init__(
(8)                         self,
(8)                         title: str = "ModernGL",
(8)                         gl_version: tuple[int, int] = (3, 3),
(8)                         size: tuple[int, int] = (1280, 720),
(8)                         resizable: bool = True,
(8)                         visible: bool = True,
(8)                         fullscreen: bool = False,
(8)                         vsync: bool = True,
(8)                         aspect_ratio: Optional[float] = None,
(8)                         samples: int = 0,
(8)                         cursor: bool = True,
(8)                         backend: Optional[str] = None,
(8)                         context_creation_func: Optional[Callable[[], Optional[moderngl.Context]]] = None,
(8)                         **kwargs: Any,
(4)                     ) -> None:
(8)                         """Initialize a window instance.
(8)                         Keyword Args:
(12)                            title:
(16)                                The window title
(12)                            gl_version:
(16)                                Major and minor version of the opengl context to create
(12)                            size:
(16)                                indow size x, y
(12)                            resizable:
(16)                                Should the window be resizable?
(12)                            visible:
(16)                                Should the window be visible when created?
(12)                            fullscreen:
(16)                                Open window in fullscreen mode
(12)                            vsync:
(16)                                Enable/disable vsync
(12)                            aspect_ratio:
(16)                                The desired fixed aspect ratio. Can be set to ``None`` to make
(16)                                aspect ratio be based on the actual window size.
(12)                            samples:
(16)                                Number of MSAA samples for the default framebuffer
(12)                            cursor:
(16)                                Enable/disable displaying the cursor inside the window
(12)                            backend:
(16)                                The context backend to use. For example ``egl`` for EGL
(12)                            context_creation_func:
(16)                                A callable returning a ModernGL context. This can be used to
(16)                                create a custom context.
(8)                         """
(8)                         self._title = title
(8)                         self._gl_version = gl_version
(8)                         self._width, self._height = int(size[0]), int(size[1])
(8)                         self._resizable = resizable
(8)                         self._visible = visible
(8)                         self._buffer_width, self._buffer_height = size
(8)                         self._fullscreen = fullscreen
(8)                         self._vsync = vsync
(8)                         self._fixed_aspect_ratio = aspect_ratio
(8)                         self._samples = samples
(8)                         self._cursor = cursor
(8)                         self._backend = backend
(8)                         self._headless = False
(8)                         self._context_creation_func = context_creation_func
(8)                         self._exit_key = self.keys.ESCAPE
(8)                         self._fs_key = self.keys.F11
(8)                         self._render_func: Callable[[float, float], None] = dummy_func
(8)                         self._resize_func: Callable[[int, int], None] = dummy_func
(8)                         self._close_func: Callable[[], None] = dummy_func
(8)                         self._iconify_func: Callable[[bool], None] = dummy_func
(8)                         self._key_event_func: Callable[[Union[str, int], int, KeyModifiers],
None] = dummy_func
124: (8)                self._mouse_position_event_func: Callable[[int, int, int, int], None]
= dummy_func
125: (8)                self._mouse_press_event_func: Callable[[int, int, int], None] =
dummy_func
126: (8)                self._mouse_release_event_func: Callable[[int, int, int], None] =
dummy_func
127: (8)                self._mouse_drag_event_func: Callable[[int, int, int, int], None] =
dummy_func
128: (8)                self._mouse_scroll_event_func: Callable[[float, float], None] =
dummy_func
129: (8)                self._unicode_char_entered_func: Callable[[str], None] = dummy_func
130: (8)                self._files_dropped_event_func: Callable[[int, int, list[Union[str,
Path]]], None] = (
131: (12)                   dummy_func
132: (8)                )
133: (8)                self._on_generic_event_func: Callable = dummy_func
134: (8)                self._ctx: moderngl.Context
135: (8)                self._viewport: tuple[int, int, int, int] = (0, 0, 0, 0)
136: (8)                self._position = 0, 0
137: (8)                self._frames = 0  # Frame counter
138: (8)                self._close = False
139: (8)                self._config: Optional[weakref.ReferenceType["WindowConfig"]] = None
140: (8)                self._key_pressed_map: dict[Union[str, int], bool] = {}
141: (8)                self._modifiers = KeyModifiers()
142: (8)                self._mouse_buttons = MouseButtonStates()
143: (8)                self._mouse_pos = 0, 0
144: (8)                self._mouse_exclusivity = False
145: (8)                if self._fullscreen:
146: (12)                   self._resizable = False
147: (8)                if self.keys is None:
148: (12)                   raise ValueError("Window class {} missing keys
attribute".format(self.__class__))
149: (4)            def init_mgl_context(self) -> None:
150: (8)                """
151: (8)                Create or assign a ModernGL context. If no context is supplied a
context will be
152: (8)                created using the window's ``gl_version``.
153: (8)                Keyword Args:
154: (12)                   ctx: An optional custom ModernGL context
155: (8)                """
156: (8)                ctx: Optional[moderngl.Context] = None
157: (8)                if self._context_creation_func:
158: (12)                   ctx = self._context_creation_func()
159: (8)                if ctx is None:
160: (12)                   ctx = moderngl.create_context(require=self.gl_version_code)
161: (8)                self._ctx = ctx
162: (8)                err = self._ctx.error
163: (8)                if err != "GL_NO_ERROR":
164: (12)                   logger.info("Consumed the following error during context creation:
%s", err)
165: (4)            @property
166: (4)            def ctx(self) -> moderngl.Context:
167: (8)                """moderngl.Context: The ModernGL context for the window"""
168: (8)                return self._ctx
169: (4)            @property
170: (4)            def backend(self) -> Optional[str]:
171: (8)                """
172: (8)                Name of the context backend.
173: (8)                This is ``None`` unless a backend is explicitly specified
174: (8)                during context creation. The main use case for this is to
175: (8)                enable EGL in headless mode.
176: (8)                """
177: (8)                return self._backend
178: (4)            @property
179: (4)            def headless(self) -> bool:
180: (8)                """bool: Is the window headless?"""
181: (8)                return self._headless
182: (4)            @property
```

```
(4)                 def fbo(self) -> moderngl.Framebuffer:
(8)                     """moderngl.Framebuffer: The default framebuffer"""
(8)                     return self._ctx.screen
(4)                 @property
(4)                 def title(self) -> str:
(8)                     """str: Window title.
(8)                     This property can also be set::
(12)                        window.title = "New Title"
(8)                     """
(8)                     return self._title
(4)                 @title.setter
(4)                 def title(self, value: str) -> None:
(8)                     self._title = value
(4)                 @property
(4)                 def fullscreen_key(self) -> Any:
(8)                     """Get or set the fullscreen toggle key for the window.
(8)                     Pressing this key will toggle fullscreen for the window.
(8)                     By default this is set to ``F11``, but this can be overridden or disabled::
(12)                        window.fullscreen_key = window.keys.F11
(12)                        window.fullscreen_key = window.keys.F
(12)                        window.fullscreen_key = None
(8)                     """
(8)                     return self._fs_key
(4)                 @fullscreen_key.setter
(4)                 def fullscreen_key(self, value: Any) -> None:
(8)                     self._fs_key = value
(4)                 @property
(4)                 def exit_key(self) -> Any:
(8)                     """Get or set the exit/close key for the window.
(8)                     Pressing this key will close the window.
(8)                     By default the ``ESCAPE`` is set, but this can be overridden or disabled::
(12)                        window.exit_key = window.keys.ESCAPE
(12)                        window.exit_key = window.keys.Q
(12)                        window.exit_key = None
(8)                     """
(8)                     return self._exit_key
(4)                 @exit_key.setter
(4)                 def exit_key(self, value: Any) -> None:
(8)                     self._exit_key = value
(4)                 @property
(4)                 def gl_version(self) -> tuple[int, int]:
(8)                     """tuple[int, int]: (major, minor) required OpenGL version"""
(8)                     return self._gl_version
(4)                 @property
(4)                 def width(self) -> int:
(8)                     """int: The current window width"""
(8)                     return self._width
(4)                 @property
(4)                 def height(self) -> int:
(8)                     """int: The current window height"""
(8)                     return self._height
(4)                 @property
(4)                 def size(self) -> tuple[int, int]:
(8)                     """tuple[int, int]: current window size.
(8)                     This property also support assignment::
(12)                        window.size = 1000, 1000
(8)                     """
(8)                     return self._width, self._height
(4)                 @size.setter
(4)                 def size(self, value: tuple[int, int]) -> None:
(8)                     self._width, self._height = int(value[0]), int(value[1])
(4)                 @property
(4)                 def buffer_width(self) -> int:
(8)                     """int: the current window buffer width"""
(8)                     return self._buffer_width
(4)                 @property
(4)                 def buffer_height(self) -> int:
```

```
(8)                        """int: the current window buffer height"""
(8)                        return self._buffer_height
(4)                    @property
(4)                    def buffer_size(self) -> tuple[int, int]:
(8)                        """tuple[int, int]: tuple with the current window buffer size"""
(8)                        return self._buffer_width, self._buffer_height
(4)                    @property
(4)                    def position(self) -> tuple[int, int]:
(8)                        """tuple[int, int]: The current window position.
(8)                        This property can also be set to move the window::
(12)                           window.position = 100, 100
(8)                        """
(8)                        return self._position
(4)                    @position.setter
(4)                    def position(self, value: tuple[int, int]) -> None:
(8)                        self._position = int(value[0]), int(value[1])
(4)                    @property
(4)                    def pixel_ratio(self) -> float:
(8)                        """float: The framebuffer/window size ratio"""
(8)                        return self.buffer_size[0] / self.size[0]
(4)                    @property
(4)                    def viewport(self) -> tuple[int, int, int, int]:
(8)                        """tuple[int, int, int, int]: current window viewport"""
(8)                        return self._viewport
(4)                    @property
(4)                    def viewport_size(self) -> tuple[int, int]:
(8)                        """tuple[int,int]: Size of the viewport.
(8)                        Equivalent to ``self.viewport[2], self.viewport[3]``
(8)                        """
(8)                        return self._viewport[2], self._viewport[3]
(4)                    @property
(4)                    def viewport_width(self) -> int:
(8)                        """int: The width of the viewport.
(8)                        Equivalent to ``self.viewport[2]``.
(8)                        """
(8)                        return self.viewport[2]
(4)                    @property
(4)                    def viewport_height(self) -> int:
(8)                        """int: The height of the viewport
(8)                        Equivalent to ``self.viewport[3]``.
(8)                        """
(8)                        return self.viewport[3]
(4)                    @property
(4)                    def frames(self) -> int:
(8)                        """int: Number of frames rendered"""
(8)                        return self._frames
(4)                    @property
(4)                    def resizable(self) -> bool:
(8)                        """bool: Window is resizable"""
(8)                        return self._resizable
(4)                    @resizable.setter
(4)                    def resizable(self, value: bool) -> None:
(8)                        self._resizable = value
(4)                    @property
(4)                    def visible(self) -> bool:
(8)                        """bool: Window is visible"""
(8)                        return self._visible
(4)                    @visible.setter
(4)                    def visible(self, value: bool) -> None:
(8)                        self._visible = value
(4)                    @property
(4)                    def hidden(self) -> bool:
(8)                        """bool: Window is hidden"""
(8)                        return not self._visible
(4)                    @hidden.setter
(4)                    def hidden(self, value: bool) -> None:
(8)                        self._visible = not value
(4)                    def hide(self) -> None:
(8)                        """Hide the window"""
```

```
(8)                     self.visible = False
(4)                 def show(self) -> None:
(8)                     """Show the window"""
(8)                     self.visible = True
(4)                 @property
(4)                 def fullscreen(self) -> bool:
(8)                     """bool: Window is in fullscreen mode"""
(8)                     return self._fullscreen
(4)                 @fullscreen.setter
(4)                 def fullscreen(self, value: bool) -> None:
(8)                     self._set_fullscreen(value)
(8)                     self._fullscreen = value
(4)                 @property
(4)                 def config(self) -> Optional["WindowConfig"]:
(8)                     """Get or det the current WindowConfig instance
(8)                     Assigning a WindowConfig instance will automatically
(8)                     set up the necessary event callback methods::
(12)                        window.config = window_config_instance
(8)                     """
(8)                     if self._config is not None:
(12)                        return self._config()
(8)                     return None
(4)                 @config.setter
(4)                 def config(self, config: "WindowConfig") -> None:
(8)                     config.assign_event_callbacks()
(8)                     self._config = weakref.ref(config)
(4)                 @property
(4)                 def vsync(self) -> bool:
(8)                     """bool: vertical sync enabled/disabled"""
(8)                     return self._vsync
(4)                 @vsync.setter
(4)                 def vsync(self, value: bool) -> None:
(8)                     self._set_vsync(value)
(8)                     self._vsync = value
(4)                 @property
(4)                 def aspect_ratio(self) -> float:
(8)                     """float: The current aspect ratio of the window.
(8)                     If a fixed aspect ratio was passed to the window
(8)                     initializer this value will always be returned.
(8)                     Otherwise ``width / height`` will be returned.
(8)                     This property is read only.
(8)                     """
(8)                     if self._fixed_aspect_ratio:
(12)                        return self._fixed_aspect_ratio
(8)                     return self.width / self.height
(4)                 @property
(4)                 def fixed_aspect_ratio(self) -> Optional[float]:
(8)                     """float: The fixed aspect ratio for the window.
(8)                     Can be set to ``None`` to disable fixed aspect ratio
(8)                     making the aspect ratio adjust to the actual window size
(8)                     This will affects how the viewport is calculated and
(8)                     the reported value from the ``aspect_ratio`` property::
(12)                        window.fixed_aspect_ratio = 16 / 9
(12)                        window.fixed_aspect_ratio = None
(8)                     """
(8)                     return self._fixed_aspect_ratio
(4)                 @fixed_aspect_ratio.setter
(4)                 def fixed_aspect_ratio(self, value: float) -> None:
(8)                     self._fixed_aspect_ratio = value
(4)                 @property
(4)                 def samples(self) -> int:
(8)                     """float: Number of Multisample anti-aliasing (MSAA) samples"""
(8)                     return self._samples
(4)                 @property
(4)                 def cursor(self) -> bool:
(8)                     """bool: Should the mouse cursor be visible inside the window?
(8)                     This property can also be assigned to::
(12)                        window.cursor = False
(8)                     """
```

```
(8)                     return self._cursor
(4)                 @cursor.setter
(4)                 def cursor(self, value: bool) -> None:
(8)                     self._cursor = value
(4)                 @property
(4)                 def mouse_exclusivity(self) -> bool:
(8)                     """bool: If mouse exclusivity is enabled.
(8)                     When you enable mouse-exclusive mode, the mouse cursor is no longer
(8)                     available. It is not merely hidden – no amount of mouse movement
(8)                     will make it leave your application. This is for example useful
(8)                     when you don't want the mouse leaving the screen when rotating
(8)                     a 3d scene.
(8)                     This property can also be set::
(12)                        window.mouse_exclusivity = True
(8)                     """
(8)                     return self._mouse_exclusivity
(4)                 @mouse_exclusivity.setter
(4)                 def mouse_exclusivity(self, value: bool) -> None:
(8)                     self._mouse_exclusivity = value
(4)                 @property
(4)                 def render_func(self) -> Callable[[float, float], None]:
(8)                     """callable: The render callable
(8)                     This property can also be used to assign a callable.
(8)                     """
(8)                     return self._render_func
(4)                 @render_func.setter
(4)                 @require_callable
(4)                 def render_func(self, func: Callable[[float, float], None]) -> None:
(8)                     self._render_func = func
(4)                 @property
(4)                 def resize_func(self) -> Callable[[int, int], None]:
(8)                     """callable: Get or set the resize callable"""
(8)                     return self._resize_func
(4)                 @resize_func.setter
(4)                 @require_callable
(4)                 def resize_func(self, func: Callable[[int, int], None]) -> None:
(8)                     self._resize_func = func
(4)                 @property
(4)                 def close_func(self) -> Callable[[], None]:
(8)                     """callable: Get or set the close callable"""
(8)                     return self._close_func
(4)                 @close_func.setter
(4)                 @require_callable
(4)                 def close_func(self, func: Callable[[], None]) -> None:
(8)                     self._close_func = func
(4)                 @property
(4)                 def files_dropped_event_func(self) -> Callable[[int, int, list[Union[str,
Path]]], None]:
(8)                     """callable: Get or set the files_dropped callable"""
(8)                     return self._files_dropped_event_func
(4)                 @files_dropped_event_func.setter
(4)                 @require_callable
(4)                 def files_dropped_event_func(
(8)                     self, func: Callable[[int, int, list[Union[str, Path]]], None]
(4)                 ) -> None:
(8)                     self._files_dropped_event_func = func
(4)                 @property
(4)                 def iconify_func(self) -> Callable[[bool], None]:
(8)                     """callable: Get or set ehe iconify/show/hide callable"""
(8)                     return self._iconify_func
(4)                 @iconify_func.setter
(4)                 @require_callable
(4)                 def iconify_func(self, func: Callable[[bool], None]) -> None:
(8)                     self._iconify_func = func
(4)                 @property
(4)                 def key_event_func(self) -> Callable[[Union[str, int], int, KeyModifiers],
None]:
(8)                     """callable: Get or set the key_event callable"""
(8)                     return self._key_event_func
```

```
(4)                 @key_event_func.setter
(4)                 @require_callable
(4)                 def key_event_func(self, func: Callable[[Union[str, int], int, KeyModifiers], None]) -> None:
(8)                     self._key_event_func = func
(4)                 @property
(4)                 def mouse_position_event_func(self) -> Callable[[int, int, int, int], None]:
(8)                     """callable: Get or set the mouse_position callable"""
(8)                     return self._mouse_position_event_func
(4)                 @mouse_position_event_func.setter
(4)                 @require_callable
(4)                 def mouse_position_event_func(self, func: Callable[[int, int, int, int], None]) -> None:
(8)                     self._mouse_position_event_func = func
(4)                 @property
(4)                 def mouse_drag_event_func(self) -> Callable[[int, int, int, int], None]:
(8)                     """callable: Get or set the mouse_drag callable"""
(8)                     return self._mouse_drag_event_func
(4)                 @mouse_drag_event_func.setter
(4)                 @require_callable
(4)                 def mouse_drag_event_func(self, func: Callable[[int, int, int, int], None]) -> None:
(8)                     self._mouse_drag_event_func = func
(4)                 @property
(4)                 def mouse_press_event_func(self) -> Callable[[int, int, int], None]:
(8)                     """callable: Get or set the mouse_press callable"""
(8)                     return self._mouse_press_event_func
(4)                 @mouse_press_event_func.setter
(4)                 @require_callable
(4)                 def mouse_press_event_func(self, func: Callable[[int, int, int], None]) -> None:
(8)                     self._mouse_press_event_func = func
(4)                 @property
(4)                 def mouse_release_event_func(self) -> Callable[[int, int, int], None]:
(8)                     """callable: Get or set the mouse_release callable"""
(8)                     return self._mouse_release_event_func
(4)                 @mouse_release_event_func.setter
(4)                 @require_callable
(4)                 def mouse_release_event_func(self, func: Callable[[int, int, int], None]) -> None:
(8)                     self._mouse_release_event_func = func
(4)                 @property
(4)                 def unicode_char_entered_func(self) -> Callable[[str], None]:
(8)                     """callable: Get or set the unicode_char_entered callable"""
(8)                     return self._unicode_char_entered_func
(4)                 @unicode_char_entered_func.setter
(4)                 @require_callable
(4)                 def unicode_char_entered_func(self, func: Callable[[str], None]) -> None:
(8)                     self._unicode_char_entered_func = func
(4)                 @property
(4)                 def mouse_scroll_event_func(self) -> Callable[[float, float], None]:
(8)                     """callable: Get or set the mouse_scroll_event calable"""
(8)                     return self._mouse_scroll_event_func
(4)                 @mouse_scroll_event_func.setter
(4)                 @require_callable
(4)                 def mouse_scroll_event_func(self, func: Callable[[float, float], None]) -> None:
(8)                     self._mouse_scroll_event_func = func
(4)                 @property
(4)                 def modifiers(self) -> KeyModifiers:
(8)                     """(KeyModifiers) The current keyboard modifiers"""
(8)                     return self._modifiers
(4)                 @property
(4)                 def mouse_states(self) -> MouseButtonStates:
(8)                     """MouseButtonStates: Mouse button state structure.
(8)                     The current mouse button states.
(8)                     .. code::
(12)                        window.mouse_buttons.left
```

```
517: (12)                       window.mouse_buttons.right
518: (12)                       window.mouse_buttons.middle
519: (8)                    """
520: (8)                    return self._mouse_buttons
521: (4)                def _handle_mouse_button_state_change(self, button: int, pressed: bool) ->
None:
522: (8)                    """Updates the internal mouse button state object.
523: (8)                    Args:
524: (12)                       button (int): The button number [1, 2 or 3]
525: (12)                       pressed (bool): Pressed (True) or released (False)
526: (8)                    """
527: (8)                    if button == self.mouse.left:
528: (12)                       self._mouse_buttons.left = pressed
529: (8)                    elif button == self.mouse.right:
530: (12)                       self._mouse_buttons.right = pressed
531: (8)                    elif button == self.mouse.middle:
532: (12)                       self._mouse_buttons.middle = pressed
533: (8)                    else:
534: (12)                       raise ValueError("Incompatible mouse button number:
{}".format(button))
535: (4)                def convert_window_coordinates(
536: (8)                    self, x: int, y: int, x_flipped: bool = False, y_flipped: bool = False
537: (4)                ) -> tuple[int, int]:
538: (8)                    """
539: (8)                    Convert window coordinates to top-left coordinate space.
540: (8)                    The default origin is the top left corner of the window.
541: (8)                    - If you are converting from bottom origin coordinates use
x_flipped=True
542: (8)                    - If you are converting from right origin coordinates use
y_flipped=True
543: (8)                    Args:
544: (12)                       x_flipped (bool) - if the input x origin is flipped
545: (12)                       y_flipped (bool) - if the input y origin is flipped
546: (8)                    Returns:
547: (12)                       tuple (x, y) of converted window coordinates
548: (8)                    """
549: (8)                    if not y_flipped and not x_flipped:
550: (12)                       return (x, y)
551: (8)                    elif y_flipped and not x_flipped:
552: (12)                       return (x, self.height - y)
553: (8)                    else:
554: (12)                       return (self.width - x, self.height - y)
555: (4)                def is_key_pressed(self, key: str) -> bool:
556: (8)                    """Returns: The press state of a key"""
557: (8)                    return self._key_pressed_map.get(key) is True
558: (4)                @property
559: (4)                def is_closing(self) -> bool:
560: (8)                    """bool: Is the window about to close?"""
561: (8)                    return self._close
562: (4)                @is_closing.setter
563: (4)                def is_closing(self, value: bool) -> None:
564: (8)                    self._close = value
565: (4)                def close(self) -> None:
566: (8)                    """Signal for the window to close"""
567: (8)                    self.is_closing = True
568: (8)                    self.close_func()
569: (4)                def use(self) -> None:
570: (8)                    """Bind the window's framebuffer"""
571: (8)                    self._ctx.screen.use()
572: (4)                def clear(
573: (8)                    self,
574: (8)                    red: float = 0.0,
575: (8)                    green: float = 0.0,
576: (8)                    blue: float = 0.0,
577: (8)                    alpha: float = 0.0,
578: (8)                    depth: float = 1.0,
579: (8)                    viewport: Optional[tuple[int, int, int, int]] = None,
580: (4)                ) -> None:
581: (8)                    """
```

```
582: (8)                    Binds and clears the default framebuffer
583: (8)                    Args:
584: (12)                       red (float): color component
585: (12)                       green (float): color component
586: (12)                       blue (float): color component
587: (12)                       alpha (float): alpha component
588: (12)                       depth (float): depth value
589: (12)                       viewport (tuple): The viewport
590: (8)                    """
591: (8)                    self.use()
592: (8)                    self._ctx.clear(
593: (12)                       red=red, green=green, blue=blue, alpha=alpha, depth=depth,
viewport=viewport
594: (8)                    )
595: (4)                def render(self, time: float = 0.0, frame_time: float = 0.0) -> None:
596: (8)                    """
597: (8)                    Renders a frame by calling the configured render callback
598: (8)                    Keyword Args:
599: (12)                       time (float): Current time in seconds
600: (12)                       frame_time (float): Delta time from last frame in seconds
601: (8)                    """
602: (8)                    self.render_func(time, frame_time)
603: (4)                def swap_buffers(self) -> None:
604: (8)                    """
605: (8)                    Library specific buffer swap method. Must be overridden.
606: (8)                    """
607: (8)                    raise NotImplementedError()
608: (4)                def resize(self, width: int, height: int) -> None:
609: (8)                    """
610: (8)                    Should be called every time window is resized
611: (8)                    so the example can adapt to the new size if needed
612: (8)                    """
613: (8)                    if self._resize_func is not dummy_func:
614: (12)                       self._resize_func(width, height)
615: (4)                def set_icon(self, icon_path: str) -> None:
616: (8)                    """
617: (8)                    Sets the window icon to the given path
618: (8)                    Args:
619: (12)                       icon_path (str): path to the icon
620: (8)                    """
621: (8)                    loader = IconLoader(TextureDescription(path=icon_path))
622: (8)                    resolved_path = loader.find_icon()
623: (8)                    self._set_icon(resolved_path)
624: (4)                def _set_icon(self, icon_path: Path) -> None:
625: (8)                    """
626: (8)                    A library specific destroy method is required.
627: (8)                    """
628: (8)                    raise NotImplementedError(
629: (12)                       "Setting an icon is currently not supported by Window-type:
{}".format(self.name)
630: (8)                    )
631: (4)                def _set_fullscreen(self, value: bool) -> None:
632: (8)                    """
633: (8)                    A library specific destroy method is required
634: (8)                    """
635: (8)                    raise NotImplementedError(
636: (12)                       "Toggling fullscreen is currently not supported by Window-type:
{}".format(self.name)
637: (8)                    )
638: (4)                def _set_vsync(self, value: bool) -> None:
639: (8)                    raise NotImplementedError(
640: (12)                       "Toggling vsync is currently not supported by Window-type:
{}".format(self.name)
641: (8)                    )
642: (4)                def destroy(self) -> None:
643: (8)                    """
644: (8)                    A library specific destroy method is required
645: (8)                    """
646: (8)                    raise NotImplementedError()
```

```
(4)       def set_default_viewport(self) -> None:
(8)           """
(8)           Calculates the and sets the viewport based on window configuration.
(8)           The viewport will based on the configured fixed aspect ratio if set.
(8)           If no fixed aspect ratio is set the viewport will be scaled
(8)           to the entire window size regardless of size.
(8)           Will add black borders and center the viewport if the window
(8)           do not match the configured viewport (fixed only)
(8)           """
(8)           if self._fixed_aspect_ratio:
(12)              expected_width = int(self._buffer_height * self._fixed_aspect_ratio)
(12)              expected_height = int(expected_width / self._fixed_aspect_ratio)
(12)              if expected_width > self._buffer_width:
(16)                  expected_width = self._buffer_width
(16)                  expected_height = int(expected_width / self._fixed_aspect_ratio)
(12)              blank_space_x = self._buffer_width - expected_width
(12)              blank_space_y = self._buffer_height - expected_height
(12)              self._viewport = (
(16)                  blank_space_x // 2,
(16)                  blank_space_y // 2,
(16)                  expected_width,
(16)                  expected_height,
(12)              )
(8)           else:
(12)              self._viewport = (0, 0, self._buffer_width, self._buffer_height)
(8)           self.fbo.viewport = self._viewport
(4)       @property
(4)       def gl_version_code(self) -> int:
(8)           """int: Generates the version code integer for the selected OpenGL version.
(8)           gl_version (4, 1) returns 410
(8)           """
(8)           return self.gl_version[0] * 100 + self.gl_version[1] * 10
(4)       def print_context_info(self) -> None:
(8)           """Prints moderngl context info."""
(8)           logger.info("Context Version:")
(8)           logger.info("ModernGL: %s", moderngl.__version__)
(8)           logger.info("vendor: %s", self._ctx.info["GL_VENDOR"])
(8)           logger.info("renderer: %s", self._ctx.info["GL_RENDERER"])
(8)           logger.info("version: %s", self._ctx.info["GL_VERSION"])
(8)           logger.info("python: %s", sys.version)
(8)           logger.info("platform: %s", sys.platform)
(8)           logger.info("code: %s", self._ctx.version_code)
(8)           err = self._ctx.error
(8)           if err != "GL_NO_ERROR":
(12)              logger.warning("glerror consumed after getting context info: %s", err)
(4)       def _calc_mouse_delta(self, xpos: int, ypos: int) -> tuple[int, int]:
(8)           """Calculates the mouse position delta for events not support this.
(8)           Args:
(12)              xpos (int): current mouse x
(12)              ypos (int): current mouse y
(8)           Returns:
(12)              tuple[int, int]: The x, y delta values
(8)           """
(8)           dx, dy = xpos - self._mouse_pos[0], ypos - self._mouse_pos[1]
(8)           self._mouse_pos = xpos, ypos
(8)           return dx, dy
(4)       @property
(4)       def on_generic_event_func(
(8)           self,
(4)       ) -> Union[Callable[[int, int, int, int], None], None]:
(8)           """
(8)           callable: Get or set the on_generic_event callable
(8)           used to funnel all non-processed events
(8)           """
(8)           return self._mouse_position_event_func
```

```
(4)      @on_generic_event_func.setter
(4)      @require_callable
(4)      def on_generic_event_func(self, func: Callable) -> None:
(8)          self._on_generic_event_func = func
(0)  class WindowConfig:
(4)      """
(4)      Creating a ``WindowConfig`` instance is the simplest interface
(4)      this library provides to open and window, handle inputs and provide simple
(4)      shortcut method for loading basic resources. It's appropriate
(4)      for projects with basic needs.
(4)      Example:
(4)      .. code:: python
(8)          import moderngl_window
(8)          class MyConfig(moderngl_window.WindowConfig):
(12)             gl_version = (3, 3)
(12)             window_size = (1920, 1080)
(12)             aspect_ratio = 16 / 9
(12)             title = "My Config"
(12)             resizable = False
(12)             samples = 8
(12)             def __init__(self, **kwargs):
(16)                 super().__init__(**kwargs)
(12)             def on_render(self, time: float, frametime: float):
(12)             def on_resize(self, width: int, height: int):
(16)                 print("Window was resized. buffer size is {} x {}".format(width, height))
(12)             def on_mouse_position_event(self, x, y, dx, dy):
(16)                 print("Mouse position:", x, y)
(12)             def on_mouse_press_event(self, x, y, button):
(16)                 print("Mouse button {} pressed at {}, {}".format(button, x, y))
(12)             def on_mouse_release_event(self, x: int, y: int, button: int):
(16)                 print("Mouse button {} released at {}, {}".format(button, x, y))
(12)             def on_key_event(self, key, action, modifiers):
(16)                 print(key, action, modifiers)
(4)      """
(4)      window_size = (1280, 720)
(4)      """
(4)      Size of the window.
(4)      .. code:: python
(8)          window_size = (1280, 720)
(4)      """
(4)      vsync = True
(4)      """
(4)      Enable or disable vsync.
(4)      .. code:: python
(8)          vsync = True
(4)      """
(4)      fullscreen = False
(4)      """
(4)      Open the window in fullscreen mode.
(4)      .. code:: python
(8)          fullscreen = False
(4)      """
(4)      resizable = True
(4)      """
(4)      Determines of the window should be resizable
(4)      .. code:: python
(8)          resizable = True
(4)      """
(4)      visible = True
(4)      """
(4)      Determines if the window should be visible when created
(4)      .. code:: python
(8)          visible = True
(4)      """
(4)      gl_version = (3, 3)
(4)      """
```

```
778: (4)                The minimum required OpenGL version required
779: (4)                .. code:: python
780: (8)                    gl_version = (3, 3)
781: (4)                """
782: (4)                title = "Example"
783: (4)                """
784: (4)                Title of the window
785: (4)                .. code:: python
786: (8)                    title = "Example"
787: (4)                """
788: (4)                aspect_ratio = 16 / 9
789: (4)                """
790: (4)                The enforced aspect ratio of the viewport. When specified back borders
791: (4)                will be calculated both vertically and horizontally if needed.
792: (4)                This property can be set to ``None`` to disable the fixed viewport system.
793: (4)                .. code:: python
794: (8)                    aspect_ratio = 16 / 9
795: (4)                """
796: (4)                clear_color = (0.0, 0.0, 0.0, 0.0)
797: (4)                """
798: (4)                The color the active framebuffer is cleared with.
799: (4)                This attribute is expected to be in the form of ``(r, g, b, a)`` in the
range ``[0.0, 1.0]``
800: (4)                If the value is `None` the screen will not be cleared every frame.
801: (4)                .. code:: python
802: (8)                    clear_color = (0.0, 0.0, 0.0, 0.0)
803: (8)                    clear_color = None
804: (4)                """
805: (4)                cursor = True
806: (4)                """
807: (4)                Determines if the mouse cursor should be visible inside the window.
808: (4)                If enabled on some platforms
809: (4)                .. code:: python
810: (8)                    cursor = True
811: (4)                """
812: (4)                samples = 0
813: (4)                """
814: (4)                Number of samples to use in multisampling.
815: (4)                .. code:: python
816: (8)                    samples = 4
817: (4)                """
818: (4)                resource_dir = None
819: (4)                """
820: (4)                Absolute path to your resource directory containing textures, scenes,
821: (4)                shaders/programs or data files. The ``load_`` methods in this class will
822: (4)                look for resources in this path. This attribute can be a ``str`` or
823: (4)                a ``pathlib.Path``.
824: (4)                .. code:: python
825: (8)                    resource_dir = None
826: (4)                """
827: (4)                hidden_window_framerate_limit = 30
828: (4)                """
829: (4)                The framerate limit for hidden windows. This is useful for windows that
830: (4)                should not render at full speed when hidden. On some platforms the
831: (4)                render loop can spike to thousands of frames per second when hidden
832: (4)                eating up battery life on laptops.
833: (4)                A value less than 0 will disable the framerate limit. Otherwise the
834: (4)                the value is a suggested limit in frames per second.
835: (4)                """
836: (4)                log_level = logging.INFO
837: (4)                """
838: (4)                Sets the log level for this library using the standard `logging` module.
839: (4)                .. code:: python
840: (8)                    log_level = logging.INFO
841: (4)                """
842: (4)                argv: Optional[Namespace] = None
843: (4)                """
844: (4)                The parsed command line arguments.
845: (4)                """
```

```
846: (4)                def __init__(
847: (8)                    self,
848: (8)                    ctx: Optional[moderngl.Context] = None,
849: (8)                    wnd: Optional[BaseWindow] = None,
850: (8)                    timer: Optional[BaseTimer] = None,
851: (8)                    **kwargs: Any,
852: (4)                ) -> None:
853: (8)                    """Initialize the window config
854: (8)                    Keyword Args:
855: (12)                       ctx (moderngl.Context): The moderngl context
856: (12)                       wnd: The window instance
857: (12)                       timer: The timer instance
858: (8)                    """
859: (8)                    if self.resource_dir:
860: (12)                       resources.register_dir(Path(self.resource_dir).resolve())
861: (8)                    if ctx is None or not isinstance(ctx, moderngl.Context):
862: (12)                       raise ValueError("WindowConfig requires a moderngl context. ctx=
{}".format(ctx))
863: (8)                    if wnd is None or not isinstance(wnd, BaseWindow):
864: (12)                       raise ValueError("WindowConfig requires a window. wnd=
{}".format(wnd))
865: (8)                    self.ctx = ctx
866: (8)                    self.wnd = wnd
867: (8)                    self.timer: BaseTimer = timer or Timer()
868: (8)                    self.assign_event_callbacks()
869: (4)                def assign_event_callbacks(self) -> None:
870: (8)                    """
871: (8)                    Look for methods in the class instance and assign them to callbacks.
872: (8)                    This method is call by ``__init__``.
873: (8)                    """
874: (8)                    self.wnd.render_func = getattr(self, "on_render", dummy_func)
875: (8)                    self.wnd.resize_func = getattr(self, "on_resize", dummy_func)
876: (8)                    self.wnd.close_func = getattr(self, "on_close", dummy_func)
877: (8)                    self.wnd.iconify_func = getattr(self, "on_iconify", dummy_func)
878: (8)                    self.wnd.key_event_func = getattr(self, "on_key_event", dummy_func)
879: (8)                    self.wnd.mouse_position_event_func = getattr(self,
"on_mouse_position_event", dummy_func)
880: (8)                    self.wnd.mouse_press_event_func = getattr(self,
"on_mouse_press_event", dummy_func)
881: (8)                    self.wnd.mouse_release_event_func = getattr(self,
"on_mouse_release_event", dummy_func)
882: (8)                    self.wnd.mouse_drag_event_func = getattr(self, "on_mouse_drag_event",
dummy_func)
883: (8)                    self.wnd.mouse_scroll_event_func = getattr(self,
"on_mouse_scroll_event", dummy_func)
884: (8)                    self.wnd.unicode_char_entered_func = getattr(self,
"on_unicode_char_entered", dummy_func)
885: (8)                    self.wnd.files_dropped_event_func = getattr(self,
"on_files_dropped_event", dummy_func)
886: (4)                @classmethod
887: (4)                def run(cls: type["WindowConfig"]) -> None:
888: (8)                    """Shortcut for running a ``WindowConfig``.
889: (8)                    This executes the following code::
890: (12)                       import moderngl_window
891: (12)                       moderngl_window.run_window_config(cls)
892: (8)                    """
893: (8)                    import moderngl_window
894: (8)                    moderngl_window.run_window_config(cls)
895: (4)                @classmethod
896: (4)                def add_arguments(cls: type["WindowConfig"], parser: ArgumentParser) ->
None:
897: (8)                    """Add arguments to default argument parser.
898: (8)                    Add arguments using ``add_argument(..)``.
899: (8)                    Args:
900: (12)                       parser (ArgumentParser): The default argument parser.
901: (8)                    """
902: (8)                    pass
903: (4)                @classmethod
904: (4)                def init_mgl_context(cls) -> Optional[moderngl.Context]:
```

```
905: (8)                    """
906: (8)                    Can be implemented to control the creation of the moderngl context.
907: (8)                    The window calls this method first during context creation.
908: (8)                    If not context is returned the window will create its own.
909: (8)                    """
910: (8)                    return None
911: (4)                def on_render(self, time: float, frame_time: float) -> None:
912: (8)                    """Renders the assigned effect
913: (8)                    Args:
914: (12)                       time (float): Current time in seconds
915: (12)                       frame_time (float): Delta time from last frame in seconds
916: (8)                    """
917: (8)                    raise NotImplementedError("WindowConfig.on_render not implemented")
918: (4)                def on_resize(self, width: int, height: int) -> None:
919: (8)                    """
920: (8)                    Called every time the window is resized
921: (8)                    in case the we need to do internal adjustments.
922: (8)                    Args:
923: (12)                       width (int): width in buffer size (not window size)
924: (12)                       height (int): height in buffer size (not window size)
925: (8)                    """
926: (4)                def on_close(self) -> None:
927: (8)                    """Called when the window is about to close"""
928: (4)                def on_files_dropped_event(self, x: int, y: int, paths: list[str]) ->
None:
929: (8)                    """
930: (8)                    Called when files dropped onto the window
931: (8)                    Args:
932: (12)                       x (int): X location in window where file was dropped
933: (12)                       y (int): Y location in window where file was dropped
934: (12)                       paths (list): List of file paths dropped
935: (8)                    """
936: (4)                def on_iconify(self, iconified: bool) -> None:
937: (8)                    """
938: (8)                    Called when the window is minimized/iconified
939: (8)                    or restored from this state
940: (8)                    Args:
941: (12)                       iconified (bool): If ``True`` the window is iconified/minimized.
Otherwise restored.
942: (8)                    """
943: (4)                def on_key_event(self, key: Any, action: Any, modifiers: KeyModifiers) ->
None:
944: (8)                    """
945: (8)                    Called for every key press and release.
946: (8)                    Depending on the library used, key events may
947: (8)                    trigger repeating events during the pressed duration
948: (8)                    based on the configured key repeat on the users
949: (8)                    operating system.
950: (8)                    Args:
951: (12)                       key: The key that was press. Compare with self.wnd.keys.
952: (12)                       action: self.wnd.keys.ACTION_PRESS or ACTION_RELEASE
953: (12)                       modifiers: Modifier state for shift, ctrl and alt
954: (8)                    """
955: (4)                def on_mouse_position_event(self, x: int, y: int, dx: int, dy: int) ->
None:
956: (8)                    """Reports the current mouse cursor position in the window
957: (8)                    Args:
958: (12)                       x (int): X position of the mouse cursor
959: (12)                       y (int): Y position of the mouse cursor
960: (12)                       dx (int): X delta position
961: (12)                       dy (int): Y delta position
962: (8)                    """
963: (4)                def on_mouse_drag_event(self, x: int, y: int, dx: int, dy: int) -> None:
964: (8)                    """Called when the mouse is moved while a button is pressed.
965: (8)                    Args:
966: (12)                       x (int): X position of the mouse cursor
967: (12)                       y (int): Y position of the mouse cursor
968: (12)                       dx (int): X delta position
969: (12)                       dy (int): Y delta position
```

```
970: (8)                    """
971: (4)                def on_mouse_press_event(self, x: int, y: int, button: int) -> None:
972: (8)                    """Called when a mouse button in pressed
973: (8)                    Args:
974: (12)                       x (int): X position the press occurred
975: (12)                       y (int): Y position the press occurred
976: (12)                       button (int): 1 = Left button, 2 = right button
977: (8)                    """
978: (4)                def on_mouse_release_event(self, x: int, y: int, button: int) -> None:
979: (8)                    """Called when a mouse button in released
980: (8)                    Args:
981: (12)                       x (int): X position the release occurred
982: (12)                       y (int): Y position the release occurred
983: (12)                       button (int): 1 = Left button, 2 = right button
984: (8)                    """
985: (4)                def on_mouse_scroll_event(self, x_offset: float, y_offset: float) -> None:
986: (8)                    """Called when the mouse wheel is scrolled.
987: (8)                    Some input devices also support horizontal scrolling,
988: (8)                    but vertical scrolling is fairly universal.
989: (8)                    Args:
990: (12)                       x_offset (int): X scroll offset
991: (12)                       y_offset (int): Y scroll offset
992: (8)                    """
993: (4)                def on_unicode_char_entered(self, char: str) -> None:
994: (8)                    """Called when the user entered a unicode character.
995: (8)                    Args:
996: (12)                       char (str): The character entered
997: (8)                    """
998: (4)                def load_texture_2d(
999: (8)                    self,
1000: (8)                   path: str,
1001: (8)                   flip: bool = True,
1002: (8)                   flip_x: bool = False,
1003: (8)                   flip_y: bool = True,
1004: (8)                   mipmap: bool = False,
1005: (8)                   mipmap_levels: Optional[tuple[int, int]] = None,
1006: (8)                   anisotropy: float = 1.0,
1007: (8)                   **kwargs: Any,
1008: (4)               ) -> moderngl.Texture:
1009: (8)                   """Loads a 2D texture.
1010: (8)                   If the path is relative the resource system is used expecting one or more
1011: (8)                   resource directories to be registered first. Absolute paths will attempt
1012: (8)                   to load the file directly.
1013: (8)                   Args:
1014: (12)                      path (str): Path to the texture relative to search directories
1015: (8)                   Keyword Args:
1016: (12)                      flip (boolean): (Use ```flip_y``) Flip the image vertically (top to bottom)
1017: (12)                      flip_x (boolean): Flip the image horizontally (left to right)
1018: (12)                      flip_y (boolean): Flip the image vertically (top to bottom)
1019: (12)                      mipmap (bool): Generate mipmaps. Will generate max possible levels unless
1020: (27)                                     `mipmap_levels` is defined.
1021: (12)                      mipmap_levels (tuple): (base, max_level) controlling mipmap generation.
1022: (35)                                             When defined the `mipmap` parameter is automatically `True`
1023: (12)                      anisotropy (float): Number of samples for anisotropic filtering
1024: (12)                      **kwargs: Additional parameters to TextureDescription
1025: (8)                   Returns:
1026: (12)                      moderngl.Texture: Texture instance
1027: (8)                   """
1028: (8)                   return resources.textures.load(
1029: (12)                      TextureDescription(
1030: (16)                          path=path,
1031: (16)                          flip=flip,
1032: (16)                          flip_x=flip_x,
```

```
1033: (16)                          flip_y=flip_y,
1034: (16)                          mipmap=mipmap,
1035: (16)                          mipmap_levels=mipmap_levels,
1036: (16)                          anisotropy=anisotropy,
1037: (16)                          **kwargs,
1038: (12)                      )
1039: (8)                   )
1040: (4)               def load_texture_array(
1041: (8)                   self,
1042: (8)                   path: str,
1043: (8)                   layers: int = 0,
1044: (8)                   flip: bool = True,
1045: (8)                   mipmap: bool = False,
1046: (8)                   mipmap_levels: Optional[tuple[int, int]] = None,
1047: (8)                   anisotropy: float = 1.0,
1048: (8)                   **kwargs: Any,
1049: (4)               ) -> moderngl.TextureArray:
1050: (8)                   """Loads a texture array.
1051: (8)                   If the path is relative the resource system is used expecting one or
more
1052: (8)                   resource directories to be registered first. Absolute paths will
attempt
1053: (8)                   to load the file directly.
1054: (8)                   Args:
1055: (12)                      path (str): Path to the texture relative to search directories
1056: (8)                   Keyword Args:
1057: (12)                      layers (int): How many layers to split the texture into vertically
1058: (12)                      flip (boolean): Flip the image horizontally
1059: (12)                      mipmap (bool): Generate mipmaps. Will generate max possible levels
unless
1060: (27)                                     `mipmap_levels` is defined.
1061: (12)                      mipmap_levels (tuple): (base, max_level) controlling mipmap
generation.
1062: (35)                                             When defined the `mipmap` parameter is
automatically `True`
1063: (12)                      anisotropy (float): Number of samples for anisotropic filtering
1064: (12)                      **kwargs: Additional parameters to TextureDescription
1065: (8)                   Returns:
1066: (12)                      moderngl.TextureArray: The texture instance
1067: (8)                   """
1068: (8)                   if kwargs is None:
1069: (12)                      kwargs = {}
1070: (8)                   if "kind" not in kwargs:
1071: (12)                      kwargs["kind"] = "array"
1072: (8)                   return resources.textures.load(
1073: (12)                      TextureDescription(
1074: (16)                          path=path,
1075: (16)                          layers=layers,
1076: (16)                          flip=flip,
1077: (16)                          mipmap=mipmap,
1078: (16)                          mipmap_levels=mipmap_levels,
1079: (16)                          anisotropy=anisotropy,
1080: (16)                          **kwargs,
1081: (12)                      )
1082: (8)                   )
1083: (4)               def load_texture_cube(
1084: (8)                   self,
1085: (8)                   pos_x: str = "",
1086: (8)                   pos_y: str = "",
1087: (8)                   pos_z: str = "",
1088: (8)                   neg_x: str = "",
1089: (8)                   neg_y: str = "",
1090: (8)                   neg_z: str = "",
1091: (8)                   flip: bool = False,
1092: (8)                   flip_x: bool = False,
1093: (8)                   flip_y: bool = False,
1094: (8)                   mipmap: bool = False,
1095: (8)                   mipmap_levels: Optional[tuple[int, int]] = None,
1096: (8)                   anisotropy: float = 1.0,
1097: (8)                   **kwargs: Any,
1098: (4)               ) -> moderngl.TextureCube:
1099: (8)                   """Loads a texture cube.
1100: (8)                   If the path is relative the resource system is used expecting one or
more
1101: (8)                   resource directories to be registered first. Absolute paths will
attempt
1102: (8)                   to load the file directly.
1103: (8)                   Keyword Args:
1104: (12)                      pos_x (str): Path to texture representing positive x face
1105: (12)                      pos_y (str): Path to texture representing positive y face
1106: (12)                      pos_z (str): Path to texture representing positive z face
1107: (12)                      neg_x (str): Path to texture representing negative x face
1108: (12)                      neg_y (str): Path to texture representing negative y face
1109: (12)                      neg_z (str): Path to texture representing negative z face
1110: (12)                      flip (boolean): (Use ``flip_y``)Flip the image vertically (top to
bottom)
1111: (12)                      flip_x (boolean): Flip the image horizontally (left to right)
1112: (12)                      flip_y (boolean): Flip the image vertically (top to bottom)
1113: (12)                      mipmap (bool): Generate mipmaps. Will generate max possible levels
unless
1114: (27)                                     `mipmap_levels` is defined.
1115: (12)                      mipmap_levels (tuple): (base, max_level) controlling mipmap
generation.
1116: (35)                                             When defined the `mipmap` parameter is
automatically `True`
1117: (12)                      anisotropy (float): Number of samples for anisotropic filtering
1118: (12)                      **kwargs: Additional parameters to TextureDescription
1119: (8)                   Returns:
1120: (12)                      moderngl.TextureCube: Texture instance
1121: (8)                   """
1122: (8)                   return resources.textures.load(
1123: (12)                      TextureDescription(
1124: (16)                          pos_x=pos_x,
1125: (16)                          pos_y=pos_y,
1126: (16)                          pos_z=pos_z,
1127: (16)                          neg_x=neg_x,
1128: (16)                          neg_y=neg_y,
1129: (16)                          neg_z=neg_z,
1130: (16)                          flip=flip,
1131: (16)                          flip_x=flip_x,
1132: (16)                          flip_y=flip_y,
1133: (16)                          mipmap=mipmap,
1134: (16)                          mipmap_levels=mipmap_levels,
1135: (16)                          anisotropy=anisotropy,
1136: (16)                          kind="cube",
1137: (16)                          **kwargs,
1138: (12)                      )
1139: (8)                   )
1140: (4)               def load_program(
1141: (8)                   self,
1142: (8)                   path: Optional[str] = None,
1143: (8)                   vertex_shader: Optional[str] = None,
1144: (8)                   geometry_shader: Optional[str] = None,
1145: (8)                   fragment_shader: Optional[str] = None,
1146: (8)                   tess_control_shader: Optional[str] = None,
1147: (8)                   tess_evaluation_shader: Optional[str] = None,
1148: (8)                   defines: Optional[dict[str, Any]] = None,
1149: (8)                   varyings: Optional[list[str]] = None,
1150: (4)               ) -> moderngl.Program:
1151: (8)                   """Loads a shader program.
1152: (8)                   Note that `path` should only be used if all shaders are defined
1153: (8)                   in the same glsl file separated by defines.
1154: (8)                   If the path is relative the resource system is used expecting one or
more
1155: (8)                   resource directories to be registered first. Absolute paths will
attempt
1156: (8)                   to load the file directly.
1157: (8)                   Keyword Args:
```

```
            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'}``.
            varyings (list[str]): Out attribute names for transform shaders
        Returns:
            moderngl.Program: The program instance
        """
        return resources.programs.load(
            ProgramDescription(
                path=path,
                vertex_shader=vertex_shader,
                geometry_shader=geometry_shader,
                fragment_shader=fragment_shader,
                tess_control_shader=tess_control_shader,
                tess_evaluation_shader=tess_evaluation_shader,
                defines=defines,
                varyings=varyings,
            )
        )
    def load_compute_shader(
        self, path: str, defines: Optional[dict[str, Any]] = None, **kwargs: Any
    ) -> moderngl.ComputeShader:
        """Loads a compute shader.
        Args:
            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:
            moderngl.ComputeShader: The compute shader
        """
        return resources.programs.load(
            ProgramDescription(compute_shader=path, defines=defines, **kwargs)
        )
    def load_text(self, path: str, **kwargs: Any) -> str:
        """Load a text file.
        If the path is relative the resource system is used expecting one or more
        resource directories to be registered first. Absolute paths will attempt
        to load the file directly.
        Args:
            path (str): Path to the file relative to search directories
            **kwargs: Additional parameters to DataDescription
        Returns:
            str: Contents of the text file
        """
        if kwargs is None:
            kwargs = {}
        if "kind" not in kwargs:
            kwargs["kind"] = "text"
        return resources.data.load(DataDescription(path=path, **kwargs))
    def load_json(self, path: str, **kwargs: Any) -> dict[str, Any]:
        """Load a json file
        If the path is relative the resource system is used expecting one or more
        resource directories to be registered first. Absolute paths will attempt
        to load the file directly.
        Args:
            path (str): Path to the file relative to search directories
            **kwargs: Additional parameters to DataDescription
```

```
1220: (8)                   Returns:
1221: (12)                      dict: Contents of the json file
1222: (8)                   """
1223: (8)                   if kwargs is not None:
1224: (12)                      kwargs = {}
1225: (8)                   if "kind" not in kwargs:
1226: (12)                      kwargs["kind"] = "json"
1227: (8)                   return resources.data.load(DataDescription(path=path, **kwargs))
1228: (4)               def load_binary(self, path: str, **kwargs: Any) -> bytes:
1229: (8)                   """Load a file in binary mode.
1230: (8)                   If the path is relative the resource system is used expecting one or
more
1231: (8)                   resource directories to be registered first. Absolute paths will
attempt
1232: (8)                   to load the file directly.
1233: (8)                   Args:
1234: (12)                      path (str): Path to the file relative to search directories
1235: (12)                      **kwargs: Additional parameters to DataDescription
1236: (8)                   Returns:
1237: (12)                      bytes: The byte data of the file
1238: (8)                   """
1239: (8)                   if kwargs is not None:
1240: (12)                      kwargs = {}
1241: (8)                   if "kind" not in kwargs:
1242: (12)                      kwargs["kind"] = "binary"
1243: (8)                   return resources.data.load(DataDescription(path=path, kind="binary"))
1244: (4)               def load_scene(
1245: (8)                   self,
1246: (8)                   path: str,
1247: (8)                   cache: bool = False,
1248: (8)                   attr_names: type[AttributeNames] = AttributeNames,
1249: (8)                   kind: Optional[str] = None,
1250: (8)                   **kwargs: Any,
1251: (4)               ) -> Scene:
1252: (8)                   """Loads a scene.
1253: (8)                   If the path is relative the resource system is used expecting one or
more
1254: (8)                   resource directories to be registered first. Absolute paths will
attempt
1255: (8)                   to load the file directly.
1256: (8)                   Keyword Args:
1257: (12)                      path (str): Path to the file relative to search directories
1258: (12)                      cache (str): Use the loader caching system if present
1259: (12)                      attr_names (AttributeNames): Attrib name config
1260: (12)                      kind (str): Override loader kind
1261: (12)                      **kwargs: Additional parameters to SceneDescription
1262: (8)                   Returns:
1263: (12)                      Scene: The scene instance
1264: (8)                   """
1265: (8)                   return resources.scenes.load(
1266: (12)                      SceneDescription(
1267: (16)                          path=path,
1268: (16)                          cache=cache,
1269: (16)                          attr_names=attr_names,
1270: (16)                          kind=kind,
1271: (16)                          **kwargs,
1272: (12)                      )
1273: (8)                   )
1274: (0)           def dummy_func(*args: Any, **kwargs: Any) -> None:
1275: (4)               """Dummy function used as the default for callbacks"""
1276: (4)               pass
```

File 8 - window.py:

```
1: (0)              from pathlib import Path
2: (0)              from typing import Any
3: (0)              import glfw
```

```
from PIL import Image
from moderngl_window.context.base import BaseWindow
from moderngl_window.context.glfw.keys import GLFW_key, Keys
class Window(BaseWindow):
    """
    Window based on GLFW
    """
    name = "glfw"
    keys = Keys
    _mouse_button_map = {
        0: 1,
        1: 2,
        2: 3,
    }
    def __init__(self, **kwargs: Any):
        super().__init__(**kwargs)
        if not glfw.init():
            raise ValueError("Failed to initialize glfw")
        glfw.window_hint(glfw.CONTEXT_CREATION_API, glfw.NATIVE_CONTEXT_API)
        glfw.window_hint(glfw.CLIENT_API, glfw.OPENGL_API)
        glfw.window_hint(glfw.CONTEXT_VERSION_MAJOR, self.gl_version[0])
        glfw.window_hint(glfw.CONTEXT_VERSION_MINOR, self.gl_version[1])
        glfw.window_hint(glfw.OPENGL_PROFILE, glfw.OPENGL_CORE_PROFILE)
        glfw.window_hint(glfw.OPENGL_FORWARD_COMPAT, True)
        glfw.window_hint(glfw.RESIZABLE, self.resizable)
        glfw.window_hint(glfw.VISIBLE, self.visible)
        glfw.window_hint(glfw.DOUBLEBUFFER, True)
        glfw.window_hint(glfw.DEPTH_BITS, 24)
        glfw.window_hint(glfw.STENCIL_BITS, 8)
        glfw.window_hint(glfw.SAMPLES, self.samples)
        glfw.window_hint(glfw.SCALE_TO_MONITOR, glfw.TRUE)
        monitor = None
        self._window = glfw.create_window(self.width, self.height, self.title, monitor, None)
        self._has_focus = True
        if self.fullscreen:
            self._set_fullscreen(True)
        if not self._window:
            glfw.terminate()
            raise ValueError("Failed to create window")
        self.cursor = self._cursor
        self._buffer_width, self._buffer_height = glfw.get_framebuffer_size(self._window)
        glfw.make_context_current(self._window)
        if self.vsync:
            glfw.swap_interval(1)
        else:
            glfw.swap_interval(0)
        glfw.set_key_callback(self._window, self.glfw_key_event_callback)
        glfw.set_cursor_pos_callback(self._window, self.glfw_mouse_event_callback)
        glfw.set_mouse_button_callback(self._window, self.glfw_mouse_button_callback)
        glfw.set_scroll_callback(self._window, self.glfw_mouse_scroll_callback)
        glfw.set_window_size_callback(self._window, self.glfw_window_resize_callback)
        glfw.set_char_callback(self._window, self.glfw_char_callback)
        glfw.set_window_focus_callback(self._window, self.glfw_window_focus)
        glfw.set_cursor_enter_callback(self._window, self.glfw_cursor_enter)
        glfw.set_window_iconify_callback(self._window, self.glfw_window_iconify)
        glfw.set_window_close_callback(self._window, self.glfw_window_close)
        self.init_mgl_context()
        self.set_default_viewport()
    def _set_fullscreen(self, value: bool) -> None:
        monitor = glfw.get_primary_monitor()
        mode = glfw.get_video_mode(monitor)
        refresh_rate = mode.refresh_rate if self.vsync else glfw.DONT_CARE
        self.resizable = not value
        glfw.window_hint(glfw.RESIZABLE, self.resizable)
        if value:
            self._non_fullscreen_size = self.width, self.height
            self._non_fullscreen_position = self.position
            glfw.set_window_monitor(
                self._window,
                monitor,
                0,
                0,
                mode.size.width,
                mode.size.height,
                refresh_rate,
            )
            glfw.window_hint(glfw.RED_BITS, mode.bits.red)
            glfw.window_hint(glfw.GREEN_BITS, mode.bits.green)
            glfw.window_hint(glfw.BLUE_BITS, mode.bits.blue)
            glfw.window_hint(glfw.REFRESH_RATE, mode.refresh_rate)
        else:
            glfw.set_window_monitor(
                self._window,
                None,
                *self._non_fullscreen_position,
                *self._non_fullscreen_size,
                refresh_rate,
            )
        if self.vsync:
            glfw.swap_interval(1)
        else:
            glfw.swap_interval(0)
    def _set_vsync(self, value: bool) -> None:
        glfw.swap_interval(value)
    @property
    def size(self) -> tuple[int, int]:
        """tuple[int, int]: current window size.
        This property also support assignment::
            window.size = 1000, 1000
        """
        return self._width, self._height
    @size.setter
    def size(self, value: tuple[int, int]) -> None:
        glfw.set_window_size(self._window, value[0], value[1])
    @property
    def position(self) -> tuple[int, int]:
        """tuple[int, int]: The current window position.
        This property can also be set to move the window::
            window.position = 100, 100
        """
        return glfw.get_window_pos(self._window)
    @position.setter
    def position(self, value: tuple[int, int]) -> None:
        self._position = glfw.set_window_pos(self._window, value[0], value[1])
    @property
    def visible(self) -> bool:
        """bool: Is the window visible?
        This property can also be set::
            window.visible = False
        """
        return self._visible
    @visible.setter
    def visible(self, value: bool) -> None:
        self._visible = value
        if value:
            glfw.show_window(self._window)
        else:
            glfw.hide_window(self._window)
    @property
    def cursor(self) -> bool:
        """bool: Should the mouse cursor be visible inside the window?
```

```
135: (8)                        This property can also be assigned to::
136: (12)                           window.cursor = False
137: (8)                        """
138: (8)                        return self._cursor
139: (4)                    @cursor.setter
140: (4)                    def cursor(self, value: bool) -> None:
141: (8)                        if not self.mouse_exclusivity:
142: (12)                           if value is True:
143: (16)                               glfw.set_input_mode(self._window, glfw.CURSOR,
glfw.CURSOR_NORMAL)
144: (12)                           elif value is False:
145: (16)                               glfw.set_input_mode(self._window, glfw.CURSOR,
glfw.CURSOR_HIDDEN)
146: (8)                        self._cursor = value
147: (4)                    @property
148: (4)                    def mouse_exclusivity(self) -> bool:
149: (8)                        """bool: If mouse exclusivity is enabled.
150: (8)                        When you enable mouse-exclusive mode, the mouse cursor is no longer
151: (8)                        available. It is not merely hidden – no amount of mouse movement
152: (8)                        will make it leave your application. This is for example useful
153: (8)                        when you don't want the mouse leaving the screen when rotating
154: (8)                        a 3d scene.
155: (8)                        This property can also be set::
156: (12)                           window.mouse_exclusivity = True
157: (8)                        """
158: (8)                        return self._mouse_exclusivity
159: (4)                    @mouse_exclusivity.setter
160: (4)                    def mouse_exclusivity(self, value: bool) -> None:
161: (8)                        self._mouse_exclusivity = value
162: (8)                        if value is True:
163: (12)                           self._mouse_pos = glfw.get_cursor_pos(self._window)
164: (12)                           glfw.set_input_mode(self._window, glfw.CURSOR,
glfw.CURSOR_DISABLED)
165: (8)                        else:
166: (12)                           self.cursor = self._cursor
167: (4)                    @property
168: (4)                    def title(self) -> str:
169: (8)                        """str: Window title.
170: (8)                        This property can also be set::
171: (12)                           window.title = "New Title"
172: (8)                        """
173: (8)                        return self._title
174: (4)                    @title.setter
175: (4)                    def title(self, value: str) -> None:
176: (8)                        glfw.set_window_title(self._window, value)
177: (8)                        self._title = value
178: (4)                    def close(self) -> None:
179: (8)                        """Suggest to glfw the window should be closed soon"""
180: (8)                        self.is_closing = True
181: (8)                        self._close_func()
182: (4)                    @property
183: (4)                    def is_closing(self) -> bool:
184: (8)                        """bool: Checks if the window is scheduled for closing"""
185: (8)                        return glfw.window_should_close(self._window)
186: (4)                    @is_closing.setter
187: (4)                    def is_closing(self, value: bool) -> None:
188: (8)                        glfw.set_window_should_close(self._window, value)
189: (4)                    def swap_buffers(self) -> None:
190: (8)                        """Swap buffers, increment frame counter and pull events"""
191: (8)                        glfw.swap_buffers(self._window)
192: (8)                        self._frames += 1
193: (8)                        glfw.poll_events()
194: (4)                    def _handle_modifiers(self, mods: GLFW_key) -> None:
195: (8)                        """Checks key modifiers"""
196: (8)                        self._modifiers.shift = mods & 1 == 1
197: (8)                        self._modifiers.ctrl = mods & 2 == 2
198: (8)                        self._modifiers.alt = mods & 4 == 4
199: (4)                    def _set_icon(self, icon_path: Path) -> None:
200: (8)                        image = Image.open(icon_path)
```

```
(8)             glfw.set_window_icon(self._window, 1, image)
(4)         def glfw_key_event_callback(
(8)             self, window: Any, key: GLFW_key, scancode: int, action: GLFW_key, mods: GLFW_key
(4)         ) -> None:
(8)             """Key event callback for glfw.
(8)             Translates and forwards keyboard event to :py:func:`keyboard_event`
(8)             Args:
(12)                window: Window event origin
(12)                key: The key that was pressed or released.
(12)                scancode: The system-specific scancode of the key.
(12)                action: ``GLFW_PRESS``, ``GLFW_RELEASE`` or ``GLFW_REPEAT``
(12)                mods: Bit field describing which modifier keys were held down.
(8)             """
(8)             if self.exit_key is not None and key == self._exit_key:
(12)                self.close()
(8)             if action == self.keys.ACTION_PRESS and self._fs_key is not None and key == self._fs_key:
(12)                self.fullscreen = not self.fullscreen
(8)             self._handle_modifiers(mods)
(8)             if action == self.keys.ACTION_PRESS:
(12)                self._key_pressed_map[key] = True
(8)             elif action == self.keys.ACTION_RELEASE:
(12)                self._key_pressed_map[key] = False
(8)             self._key_event_func(key, action, self._modifiers)
(4)         def glfw_mouse_event_callback(self, window: Any, xpos: float, ypos: float) -> None:
(8)             """Mouse position event callback from glfw.
(8)             Translates the events forwarding them to :py:func:`cursor_event`.
(8)             Screen coordinates relative to the top-left corner
(8)             Args:
(12)                window: The window
(12)                xpos: viewport x pos
(12)                ypos: viewport y pos
(8)             """
(8)             xpos, ypos = int(xpos), int(ypos)
(8)             dx, dy = self._calc_mouse_delta(xpos, ypos)
(8)             if self.mouse_states.any:
(12)                self._mouse_drag_event_func(xpos, ypos, dx, dy)
(8)             else:
(12)                self._mouse_position_event_func(xpos, ypos, dx, dy)
(4)         def glfw_mouse_button_callback(
(8)             self, window: Any, button: GLFW_key, action: GLFW_key, mods: GLFW_key
(4)         ) -> None:
(8)             """Handle mouse button events and forward them to the example
(8)             Args:
(12)                window: The window
(12)                button: The button creating the event
(12)                action: Button action (press or release)
(12)                mods: They modifiers such as ctrl or shift
(8)             """
(8)             self._handle_modifiers(mods)
(8)             button = self._mouse_button_map.get(button, -1)
(8)             if button == -1:
(12)                return
(8)             xpos, ypos = glfw.get_cursor_pos(self._window)
(8)             if action == glfw.PRESS:
(12)                self._handle_mouse_button_state_change(button, True)
(12)                self._mouse_press_event_func(xpos, ypos, button)
(8)             else:
(12)                self._handle_mouse_button_state_change(button, False)
(12)                self._mouse_release_event_func(xpos, ypos, button)
(4)         def glfw_mouse_scroll_callback(self, window: Any, x_offset: float, y_offset: float) -> None:
(8)             """Handle mouse scroll events and forward them to the example
(8)             Args:
(12)                window: The window
(12)                x_offset (float): x wheel offset
(12)                y_offest (float): y wheel offset
```

```
266: (8)                    """
267: (8)                    self._mouse_scroll_event_func(x_offset, y_offset)
268: (4)                def glfw_char_callback(self, window: Any, codepoint: int) -> None:
269: (8)                    """Handle text input (only unicode charaters)
270: (8)                    Args:
271: (12)                       window: The glfw window
272: (12)                       codepoint (int): The unicode codepoint
273: (8)                    """
274: (8)                    self._unicode_char_entered_func(chr(codepoint))
275: (4)                def glfw_window_resize_callback(self, window: Any, width: int, height: int) -> None:
276: (8)                    """
277: (8)                    Window resize callback for glfw
278: (8)                    Args:
279: (12)                       window: The window
280: (12)                       width: New width
281: (12)                       height: New height
282: (8)                    """
283: (8)                    self._width, self._height = width, height
284: (8)                    self._buffer_width, self._buffer_height = glfw.get_framebuffer_size(self._window)
285: (8)                    self.set_default_viewport()
286: (8)                    super().resize(self._buffer_width, self._buffer_height)
287: (4)                def glfw_window_focus(self, window: Any, focused: int) -> None:
288: (8)                    """Called when the window focus is changed.
289: (8)                    Args:
290: (12)                       window: The window instance
291: (12)                       focused (int): 0: de-focus, 1: focused
292: (8)                    """
293: (8)                    self._has_focus = True if focused == 1 else False
294: (4)                def glfw_cursor_enter(self, window: Any, enter: int) -> None:
295: (8)                    """called when the cursor enters or leaves the content area of the window.
296: (8)                    Args:
297: (12)                       window: the window instance
298: (12)                       enter (int): 0: leave, 1: enter
299: (8)                    """
300: (8)                    pass
301: (4)                def glfw_window_iconify(self, window: Any, iconified: int) -> None:
302: (8)                    """Called when the window is minimized or restored.
303: (8)                    Args:
304: (12)                       window: The window
305: (12)                       iconified (int): 1 = minimized, 0 = restored.
306: (8)                    """
307: (8)                    self._visible = iconified == 0
308: (8)                    self._iconify_func(True if iconified == 1 else False)
309: (4)                def glfw_window_close(self, window: Any) -> None:
310: (8)                    """Called when the window is closed"""
311: (8)                    self.close()
312: (4)                def destroy(self) -> None:
313: (8)                    """Gracefully terminate GLFW"""
314: (8)                    glfw.terminate()

----------------------------------------

File 9 - window.py:

1: (0)              from pathlib import Path
2: (0)              from typing import Any, Optional
3: (0)              import moderngl
4: (0)              from moderngl_window.context.base import BaseWindow
5: (0)              from moderngl_window.context.headless.keys import Keys
6: (0)              class Window(BaseWindow):
7: (4)                  """Headless window.
8: (4)                  Do not currently support any form window events or key input.
9: (4)                  """
10: (4)                 name = "headless"
11: (4)                 keys = Keys
12: (4)                 def __init__(self, **kwargs: Any):
        super().__init__(**kwargs)
        self._fbo: Optional[moderngl.Framebuffer] = None
        self._vsync = False  # We don't care about vsync in headless mode
        self._resizable = False  # headless window is not resizable
        self._cursor = False  # Headless don't have a cursor
        self._headless = True
        self.init_mgl_context()
        self.set_default_viewport()
    @property
    def fbo(self) -> moderngl.Framebuffer:
        """moderngl.Framebuffer: The default framebuffer"""
        if self._fbo is None:
            raise RuntimeError("No framebuffer created yet")
        return self._fbo
    def init_mgl_context(self) -> None:
        """Create an standalone context and framebuffer"""
        if self._backend is not None:
            self._ctx = moderngl.create_standalone_context(
                require=self.gl_version_code,
                backend=self._backend,
            )
        else:
            self._ctx = moderngl.create_standalone_context(
                require=self.gl_version_code,
            )
        self._create_fbo()
        self.use()
    def _create_fbo(self) -> None:
        if self._fbo:
            for attachment in self._fbo.color_attachments:
                attachment.release()
            if self._fbo.depth_attachment:
                self._fbo.depth_attachment.release()
            self._fbo.release()
        self._fbo = self.ctx.framebuffer(
            color_attachments=self.ctx.texture(self.size, 4, samples=self._samples),
            depth_attachment=self.ctx.depth_texture(self.size, samples=self._samples),
        )
    @property
    def size(self) -> tuple[int, int]:
        """tuple[int, int]: current window size.
        This property also support assignment::
            window.size = 1000, 1000
        """
        return self._width, self._height
    @size.setter
    def size(self, value: tuple[int, int]) -> None:
        if value == (self._width, self._height):
            return
        self._width, self._height = value
        self._create_fbo()
    def use(self) -> None:
        """Bind the window's framebuffer"""
        if self._fbo is None:
            raise RuntimeError("No framebuffer created yet")
        self._fbo.use()
    def clear(
        self,
        red: float = 0.0,
        green: float = 0.0,
        blue: float = 0.0,
        alpha: float = 0.0,
        depth: float = 1.0,
        viewport: Optional[tuple[int, int, int, int]] = None,
    ) -> None:
        """
        Binds and clears the default framebuffer
```

```
80: (8)                         Args:
81: (12)                            red (float): color component
82: (12)                            green (float): color component
83: (12)                            blue (float): color component
84: (12)                            alpha (float): alpha component
85: (12)                            depth (float): depth value
86: (12)                            viewport (tuple): The viewport
87: (8)                         """
88: (8)                         self.use()
89: (8)                         self._ctx.clear(
90: (12)                            red=red, green=green, blue=blue, alpha=alpha, depth=depth,
viewport=viewport
91: (8)                         )
92: (4)                     def swap_buffers(self) -> None:
93: (8)                         """
94: (8)                         Placeholder. We currently don't do double buffering in headless mode.
95: (8)                         This may change in the future.
96: (8)                         """
97: (8)                         self._frames += 1
98: (8)                         self._ctx.finish()
99: (4)                     def _set_icon(self, icon_path: Path) -> None:
100: (8)                        """Do nothing when icon is set"""
101: (8)                        pass
102: (4)                    def _set_fullscreen(self, value: bool) -> None:
103: (8)                        """Do nothing when fullscreen is toggled"""
104: (8)                        pass
105: (4)                    def _set_vsync(self, value: bool) -> None:
106: (8)                        pass
107: (4)                    def destroy(self) -> None:
108: (8)                        """Destroy the context"""
109: (8)                        self._ctx.release()

----------------------------------------

File 10 - default.py:

1: (0)                  WINDOW = {
2: (4)                      "gl_version": (3, 3),
3: (4)                      "class": "moderngl_window.context.pyglet.Window",
4: (4)                      "size": (1280, 720),
5: (4)                      "aspect_ratio": 16 / 9,
6: (4)                      "fullscreen": False,
7: (4)                      "resizable": True,
8: (4)                      "title": "ModernGL Window",
9: (4)                      "vsync": True,
10: (4)                     "cursor": True,
11: (4)                     "samples": 0,
12: (0)                 }
13: (0)                 SCREENSHOT_PATH = None
14: (0)                 PROGRAM_FINDERS = [
15: (4)                     "moderngl_window.finders.program.FilesystemFinder",
16: (0)                 ]
17: (0)                 TEXTURE_FINDERS = [
18: (4)                     "moderngl_window.finders.texture.FilesystemFinder",
19: (0)                 ]
20: (0)                 SCENE_FINDERS = [
21: (4)                     "moderngl_window.finders.scene.FilesystemFinder",
22: (0)                 ]
23: (0)                 DATA_FINDERS = [
24: (4)                     "moderngl_window.finders.data.FilesystemFinder",
25: (0)                 ]
26: (0)                 PROGRAM_DIRS: list[str] = []
27: (0)                 TEXTURE_DIRS: list[str] = []
28: (0)                 SCENE_DIRS: list[str] = []
29: (0)                 DATA_DIRS: list[str] = []
30: (0)                 PROGRAM_LOADERS = [
31: (4)                     "moderngl_window.loaders.program.single.Loader",
32: (4)                     "moderngl_window.loaders.program.separate.Loader",
33: (0)                 ]
```

```
34: (0)             TEXTURE_LOADERS = [
35: (4)                 "moderngl_window.loaders.texture.t2d.Loader",
36: (4)                 "moderngl_window.loaders.texture.array.Loader",
37: (4)                 "moderngl_window.loaders.texture.cube.Loader",
38: (0)             ]
39: (0)             SCENE_LOADERS = [
40: (4)                 "moderngl_window.loaders.scene.gltf2.Loader",
41: (4)                 "moderngl_window.loaders.scene.wavefront.Loader",
42: (4)                 "moderngl_window.loaders.scene.stl.Loader",
43: (0)             ]
44: (0)             DATA_LOADERS = [
45: (4)                 "moderngl_window.loaders.data.binary.Loader",
46: (4)                 "moderngl_window.loaders.data.text.Loader",
47: (4)                 "moderngl_window.loaders.data.json.Loader",
48: (0)             ]
```

File 11 - __init__.py:

```
1: (0)              """
2: (0)              General helper functions aiding in the boostrapping of this library.
3: (0)              """
4: (0)              import argparse
5: (0)              import logging
6: (0)              import os
7: (0)              import sys
8: (0)              import time
9: (0)              import weakref
10: (0)             from pathlib import Path
11: (0)             from typing import Any, Optional
12: (0)             import moderngl
13: (0)             from moderngl_window.conf import settings
14: (0)             from moderngl_window.context.base import BaseWindow, WindowConfig
15: (0)             from moderngl_window.timers.clock import Timer
16: (0)             from moderngl_window.utils.keymaps import AZERTY, QWERTY, KeyMap,
KeyMapFactory  # noqa
17: (0)             from moderngl_window.utils.module_loading import import_string
18: (0)             __version__ = "3.0.3"
19: (0)             IGNORE_DIRS = [
20: (4)                 "__pycache__",
21: (4)                 "base",
22: (0)             ]
23: (0)             WINDOW_CLASSES = [
24: (4)                 "glfw",
25: (4)                 "headless",
26: (4)                 "pygame2",
27: (4)                 "pyglet",
28: (4)                 "pyqt5",
29: (4)                 "pyside2",
30: (4)                 "sdl2",
31: (4)                 "tk",
32: (0)             ]
33: (0)             OPTIONS_TRUE = ["yes", "on", "true", "t", "y", "1"]
34: (0)             OPTIONS_FALSE = ["no", "off", "false", "f", "n", "0"]
35: (0)             OPTIONS_ALL = OPTIONS_TRUE + OPTIONS_FALSE
36: (0)             logger = logging.getLogger(__name__)
37: (0)             def setup_basic_logging(level: int) -> None:
38: (4)                 """Set up basic logging
39: (4)                 Args:
40: (8)                     level (int): The log level
41: (4)                 """
42: (4)                 if level is None:
43: (8)                     return
44: (4)                 if not logger.handlers:
45: (8)                     logger.propagate = False
46: (8)                     logger.setLevel(level)
47: (8)                     ch = logging.StreamHandler()
48: (8)                     ch.setLevel(logging.DEBUG)
```

```
49: (8)                     ch.setFormatter(logging.Formatter("%(asctime)s - %(name)s - %
(levelname)s - %(message)s"))
50: (8)                     logger.addHandler(ch)
51: (0)             class ContextRefs:
52: (4)                 """Namespace for window/context references"""
53: (4)                 WINDOW: Optional[BaseWindow] = None
54: (4)                 CONTEXT: Optional[moderngl.Context] = None
55: (0)             def activate_context(
56: (4)                 window: Optional[BaseWindow] = None, ctx: Optional[moderngl.Context] =
None
57: (0)             ) -> None:
58: (4)                 """
59: (4)                 Register the active window and context.
60: (4)                 If only a window is supplied the context is taken from the window.
61: (4)                 Only a context can also be passed in.
62: (4)                 Keyword Args:
63: (8)                     window (window): The window to activate
64: (8)                     ctx (moderngl.Context): The moderngl context to activate
65: (4)                 """
66: (4)                 ContextRefs.WINDOW = window
67: (4)                 ContextRefs.CONTEXT = ctx
68: (4)                 if ctx is None:
69: (8)                     ContextRefs.CONTEXT = window.ctx
70: (0)             def window() -> BaseWindow:
71: (4)                 """Obtain the active window"""
72: (4)                 if ContextRefs.WINDOW:
73: (8)                     return ContextRefs.WINDOW
74: (4)                 raise ValueError("No active window and context. Call activate_window.")
75: (0)             def ctx() -> moderngl.Context:
76: (4)                 """Obtain the active context"""
77: (4)                 if ContextRefs.CONTEXT:
78: (8)                     return ContextRefs.CONTEXT
79: (4)                 raise ValueError("No active window and context. Call activate_window.")
80: (0)             def get_window_cls(window: str = "") -> type[BaseWindow]:
81: (4)                 """
82: (4)                 Attempt to obtain a window class using the full dotted
83: (4)                 python path. This can be used to import custom or modified
84: (4)                 window classes.
85: (4)                 Args:
86: (8)                     window (str): Name of the window
87: (4)                 Returns:
88: (8)                     A reference to the requested window class. Raises exception if not
found.
89: (4)                 """
90: (4)                 logger.info("Attempting to load window class: %s", window)
91: (4)                 win = import_string(window)
92: (4)                 return win
93: (0)             def get_local_window_cls(window: Optional[str] = None) -> type[BaseWindow]:
94: (4)                 """
95: (4)                 Attempt to obtain a window class in the moderngl_window package
96: (4)                 using short window names such as ``pyglet`` or ``glfw``.
97: (4)                 Args:
98: (8)                     window (str): Name of the window
99: (4)                 Returns:
100: (8)                    A reference to the requested window class. Raises exception if not
found.
101: (4)                """
102: (4)                window = os.environ.get("MODERNGL_WINDOW") or window
103: (4)                if window is None:
104: (8)                    window = "pyglet"
105: (4)                return get_window_cls("moderngl_window.context.{}.Window".format(window))
106: (0)            def find_window_classes() -> list[str]:
107: (4)                """
108: (4)                Find available window packages
109: (4)                Returns:
110: (8)                    A list of available window packages
111: (4)                """
112: (4)                try:
113: (8)                    return [
114: (12)                       path.parts[-1]
115: (12)                       for path in Path(__file__).parent.joinpath("context").iterdir()
116: (12)                       if path.is_dir() and path.parts[-1] not in IGNORE_DIRS
117: (8)                    ]
118: (4)                except Exception:
119: (8)                    return WINDOW_CLASSES
120: (0)            def create_window_from_settings() -> BaseWindow:
121: (4)                """
122: (4)                Creates a window using configured values in
:py:attr:`moderngl_window.conf.Settings.WINDOW`.
123: (4)                This will also activate the window/context.
124: (4)                Returns:
125: (8)                    The Window instance
126: (4)                """
127: (4)                window_cls = import_string(settings.WINDOW["class"])
128: (4)                window = window_cls(**settings.WINDOW)
129: (4)                assert isinstance(
130: (8)                    window, BaseWindow
131: (4)                ), f"{type(window)} is not derived from
moderngl_window.context.base.BaseWindow"
132: (4)                activate_context(window=window)
133: (4)                return window
134: (0)            def run_window_config(
135: (4)                config_cls: type[WindowConfig], timer: Optional[Timer] = None, args: Any =
None
136: (0)            ) -> None:
137: (4)                """
138: (4)                Run an WindowConfig entering a blocking main loop
139: (4)                Args:
140: (8)                    config_cls: The WindowConfig class to render
141: (4)                Keyword Args:
142: (8)                    timer: A custom timer instance
143: (8)                    args: Override sys.args
144: (4)                """
145: (4)                config = create_window_config_instance(config_cls, timer=timer, args=args)
146: (4)                run_window_config_instance(config)
147: (0)            def create_window_config_instance(
148: (4)                config_cls: type[WindowConfig], timer: Optional[Timer] = None, args: Any =
None
149: (0)            ) -> WindowConfig:
150: (4)                """
151: (4)                Create and initialize a instance of a WindowConfig class.
152: (4)                Quite a bit of boilerplate is required to create a WindowConfig instance
153: (4)                and this function aims to simplify that.
154: (4)                Args:
155: (8)                    window_config: The WindowConfig class to create an instance of
156: (4)                Keyword Args:
157: (8)                    kwargs: Arguments to pass to the WindowConfig constructor
158: (4)                Returns:
159: (8)                    An instance of the WindowConfig class
160: (4)                """
161: (4)                setup_basic_logging(config_cls.log_level)
162: (4)                parser = create_parser()
163: (4)                config_cls.add_arguments(parser)
164: (4)                values = parse_args(args=args, parser=parser)
165: (4)                config_cls.argv = values
166: (4)                window_cls = get_local_window_cls(values.window)
167: (4)                size = values.size or config_cls.window_size
168: (4)                size = int(size[0] * values.size_mult), int(size[1] * values.size_mult)
169: (4)                show_cursor = values.cursor
170: (4)                if show_cursor is None:
171: (8)                    show_cursor = config_cls.cursor
172: (4)                window = window_cls(
173: (8)                    title=config_cls.title,
174: (8)                    size=size,
175: (8)                    fullscreen=config_cls.fullscreen or values.fullscreen,
176: (8)                    resizable=(values.resizable if values.resizable is not None else
config_cls.resizable),
177: (8)                    visible=config_cls.visible,
        gl_version=config_cls.gl_version,
        aspect_ratio=config_cls.aspect_ratio,
        vsync=values.vsync if values.vsync is not None else config_cls.vsync,
        samples=values.samples if values.samples is not None else config_cls.samples,
        cursor=show_cursor if show_cursor is not None else True,
        backend=values.backend,
        context_creation_func=config_cls.init_mgl_context,
    )
    window.print_context_info()
    activate_context(window=window)
    if timer is None:
        timer = Timer()
    config = config_cls(ctx=window.ctx, wnd=window, timer=timer)
    window._config = weakref.ref(config)
    window.swap_buffers()
    window.set_default_viewport()
    return config
def run_window_config_instance(config: WindowConfig) -> None:
    """
    Run an WindowConfig instance entering a blocking main loop.
    Args:
        window_config: The WindowConfig instance
    """
    window = config.wnd
    timer = config.timer
    timer.start()
    while not window.is_closing:
        current_time, delta = timer.next_frame()
        if not window.visible and config.hidden_window_framerate_limit > 0:
            expected_delta_time = 1.0 / config.hidden_window_framerate_limit
            sleep_time = expected_delta_time - delta
            if sleep_time > 0:
                time.sleep(sleep_time)
        if config.clear_color is not None:
            window.clear(*config.clear_color)
        window.use()
        window.render(current_time, delta)
        if not window.is_closing:
            window.swap_buffers()
    _, duration = timer.stop()
    window.destroy()
    if duration > 0:
        logger.info("Duration: {0:.2f}s @ {1:.2f} FPS".format(duration, timer.fps_average))
def create_parser() -> argparse.ArgumentParser:
    """Create an argparser parsing the standard arguments for WindowConfig"""
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-wnd",
        "--window",
        choices=find_window_classes(),
        help="Name for the window type to use",
    )
    parser.add_argument(
        "-fs",
        "--fullscreen",
        action="store_true",
        help="Open the window in fullscreen mode",
    )
    parser.add_argument(
        "-vs",
        "--vsync",
        type=valid_bool,
        help="Enable or disable vsync",
    )
    parser.add_argument(
        "-r",
        "--resizable",
245: (8)                    type=valid_bool,
246: (8)                    default=None,
247: (8)                    help="Enable/disable window resize",
248: (4)                )
249: (4)                parser.add_argument(
250: (8)                    "-hd",
251: (8)                    "--hidden",
252: (8)                    type=valid_bool,
253: (8)                    default=False,
254: (8)                    help="Start the window in hidden mode",
255: (4)                )
256: (4)                parser.add_argument(
257: (8)                    "-s",
258: (8)                    "--samples",
259: (8)                    type=int,
260: (8)                    help="Specify the desired number of samples to use for multisampling",
261: (4)                )
262: (4)                parser.add_argument(
263: (8)                    "-c",
264: (8)                    "--cursor",
265: (8)                    type=valid_bool,
266: (8)                    help="Enable or disable displaying the mouse cursor",
267: (4)                )
268: (4)                parser.add_argument(
269: (8)                    "--size",
270: (8)                    type=valid_window_size,
271: (8)                    help="Window size",
272: (4)                )
273: (4)                parser.add_argument(
274: (8)                    "--size_mult",
275: (8)                    type=valid_window_size_multiplier,
276: (8)                    default=1.0,
277: (8)                    help="Multiplier for the window size making it easy scale the window",
278: (4)                )
279: (4)                parser.add_argument(
280: (8)                    "--backend",
281: (8)                    help="Specify context backend. This is mostly used to enable EGL in
headless mode",
282: (4)                )
283: (4)                return parser
284: (0)            def parse_args(
285: (4)                args: Optional[Any] = None, parser: Optional[argparse.ArgumentParser] =
None
286: (0)            ) -> argparse.Namespace:
287: (4)                """Parse arguments from sys.argv
288: (4)                Passing in your own argparser can be user to extend the parser.
289: (4)                Keyword Args:
290: (8)                    args: override for sys.argv
291: (8)                    parser: Supply your own argparser instance
292: (4)                """
293: (4)                parser = parser or create_parser()
294: (4)                return parser.parse_args(args or sys.argv[1:])
295: (0)            def valid_bool(value: Optional[str]) -> Optional[bool]:
296: (4)                """Validator for bool values"""
297: (4)                if value is None:
298: (8)                    return None
299: (4)                value = value.lower()
300: (4)                if value in OPTIONS_TRUE:
301: (8)                    return True
302: (4)                if value in OPTIONS_FALSE:
303: (8)                    return False
304: (4)                raise argparse.ArgumentTypeError(f"Boolean value expected. Options:
{OPTIONS_ALL}")
305: (0)            def valid_window_size(value: str) -> tuple[int, int]:
306: (4)                """
307: (4)                Validator for window size parameter.
308: (4)                Valid format is "[int]x[int]". For example "1920x1080".
309: (4)                """
310: (4)                try:
```

```
311: (8)                        width, height = value.split("x")
312: (8)                        return int(width), int(height)
313: (4)                    except ValueError:
314: (8)                        pass
315: (4)                    raise argparse.ArgumentTypeError(
316: (8)                        "Valid size format: int]x[int]. Example '1920x1080'",
317: (4)                    )
318: (0)                def valid_window_size_multiplier(value: str) -> float:
319: (4)                    """Validates window size multiplier
320: (4)                    Must be an integer or float greater than 0
321: (4)                    """
322: (4)                    try:
323: (8)                        val = float(value)
324: (8)                        if val > 0:
325: (12)                           return val
326: (4)                    except ValueError:
327: (8)                        pass
328: (4)                    raise argparse.ArgumentTypeError(
329: (8)                        "Must be a positive int or float",
330: (4)                    )

----------------------------------------

File 12 - __init__.py:

1: (0)

----------------------------------------

File 13 - __init__.py:

1: (0)                  from .base import BaseVideoCapture  # noqa
2: (0)                  from .ffmpeg import FFmpegCapture  # noqa

----------------------------------------

File 14 - __init__.py:

1: (0)                  """
2: (0)                  Bag of settings values
3: (0)                  """
4: (0)                  import importlib
5: (0)                  import os
6: (0)                  import pathlib
7: (0)                  from collections.abc import Generator, Iterable
8: (0)                  from pprint import pformat
9: (0)                  from types import ModuleType as Module
10: (0)                 from typing import Any, Optional, Union
11: (0)                 from moderngl_window.conf import default
12: (0)                 from moderngl_window.exceptions import ImproperlyConfigured
13: (0)                 SETTINGS_ENV_VAR = "MODERNGL_WINDOW_SETTINGS_MODULE"
14: (0)                 class Settings:
15: (4)                     """
16: (4)                     Bag of settings values. New attributes can be freely added runtime.
17: (4)                     Various apply* methods are supplied so the user have full control over how
18: (4)                     settings values are initialized. This is especially useful for more custom
usage.
19: (4)                     And instance of the `Settings` class is created when the `conf` module is
imported.
20: (4)                     Attribute names must currently be in upper case to be recognized.
21: (4)                     Some examples of usage::
22: (8)                         from moderngl_window.conf import settings
23: (8)                         try:
24: (12)                            value = settings.VALUE
25: (8)                         except KeyError:
26: (12)                            raise ValueError("This settings value is required")
27: (8)                         value = getattr(settings, 'VALUE', 'default_value')
28: (8)                         print(settings)
29: (4)                     """
```

```
(4)                 WINDOW: dict[str, Any] = dict()
(4)                 """
(4)                 Window/screen properties. Most importantly the ``class`` attribute
(4)                 decides what class should be used to handle the window.
(4)                 .. code:: python
(8)                     WINDOW = {
(12)                        "gl_version": (3, 3),
(12)                        "class": "moderngl_window.context.pyglet.Window",
(12)                        "size": (1280, 720),
(12)                        "aspect_ratio": 16 / 9,
(12)                        "fullscreen": False,
(12)                        "resizable": True,
(12)                        "title": "ModernGL Window",
(12)                        "vsync": True,
(12)                        "cursor": True,
(12)                        "samples": 0,
(8)                     }
(4)                 Other Properties:
(4)                 - ``gl_version``: The minimum required major/minor OpenGL version
(4)                 - ``size``: The window size to open.
(4)                 - ``aspect_ratio`` is the enforced aspect ratio of the viewport.
(4)                 - ``fullscreen``: True if you want to create a context in fullscreen mode
(4)                 - ``resizable``: If the window should be resizable. This only applies in
(6)                   windowed mode.
(4)                 - ``vsync``: Only render one frame per screen refresh
(4)                 - ``title``: The visible title on the window in windowed mode
(4)                 - ``cursor``: Should the mouse cursor be visible on the screen? Disabling
(6)                   this is also useful in windowed mode when controlling the camera on some
(6)                   platforms as moving the mouse outside the window can cause issues.
(4)                 - ``Samples``: Number if samples used in multisampling. Values above 1
(6)                   enables multisampling.
(4)                 The created window frame buffer will by default use:
(4)                 - RGBA8 (32 bit per pixel)
(4)                 - 24 bit depth buffer
(4)                 - Double buffering
(4)                 - color and depth buffer is cleared for every frame
(4)                 """
(4)                 SCREENSHOT_PATH: Optional[str] = None
(4)                 """
(4)                 Absolute path to the directory screenshots will be saved by the screenshot
module.
(4)                 Screenshots will end up in the project root of not defined.
(4)                 If a path is configured, the directory will be auto-created.
(4)                 """
(4)                 PROGRAM_FINDERS: list[str] = []
(4)                 """
(4)                 Finder classes for locating programs/shaders.
(4)                 .. code:: python
(8)                     PROGRAM_FINDERS = [
(12)                        "moderngl_window.finders.program.FileSystemFinder",
(8)                     ]
(4)                 """
(4)                 TEXTURE_FINDERS: list[str] = []
(4)                 """
(4)                 Finder classes for locating textures.
(4)                 .. code:: python
(8)                     TEXTURE_FINDERS = [
(12)                        "moderngl_window.finders.texture.FileSystemFinder",
(8)                     ]
(4)                 """
(4)                 SCENE_FINDERS: list[str] = []
(4)                 """
(4)                 Finder classes for locating scenes.
(4)                 .. code:: python
(8)                     SCENE_FINDERS = [
(12)                        "moderngl_window.finders.scene.FileSystemFinder",
(8)                     ]
(4)                 """
(4)                 DATA_FINDERS: list[str] = []
```

```
 98: (4)                """
 99: (4)                Finder classes for locating data files.
100: (4)                .. code:: python
101: (8)                    DATA_FINDERS = [
102: (12)                       "moderngl_window.finders.data.FileSystemFinder",
103: (8)                    ]
104: (4)                """
105: (4)                PROGRAM_DIRS: list[Union[str, pathlib.Path]] = []
106: (4)                """
107: (4)                Lists of `str` or `pathlib.Path` used by ``FileSystemFinder``
108: (4)                to looks for programs/shaders.
109: (4)                """
110: (4)                TEXTURE_DIRS: list[Union[str, pathlib.Path]] = []
111: (4)                """
112: (4)                Lists of `str` or `pathlib.Path` used by ``FileSystemFinder``
113: (4)                to looks for textures.
114: (4)                """
115: (4)                SCENE_DIRS: list[Union[str, pathlib.Path]] = []
116: (4)                """
117: (4)                Lists of `str` or `pathlib.Path` used by ``FileSystemFinder``
118: (4)                to looks for scenes (obj, gltf, stl etc).
119: (4)                """
120: (4)                DATA_DIRS: list[Union[str, pathlib.Path]] = []
121: (4)                """
122: (4)                Lists of `str` or `pathlib.Path` used by ``FileSystemFinder``
123: (4)                to looks for data files.
124: (4)                """
125: (4)                PROGRAM_LOADERS: list[str] = []
126: (4)                """
127: (4)                Classes responsible for loading programs/shaders.
128: (4)                .. code:: python
129: (8)                    PROGRAM_LOADERS = [
130: (12)                       'moderngl_window.loaders.program.single.Loader',
131: (12)                       'moderngl_window.loaders.program.separate.Loader',
132: (8)                    ]
133: (4)                """
134: (4)                TEXTURE_LOADERS: list[str] = []
135: (4)                """
136: (4)                Classes responsible for loading textures.
137: (4)                .. code:: python
138: (8)                    TEXTURE_LOADERS = [
139: (12)                       'moderngl_window.loaders.texture.t2d.Loader',
140: (12)                       'moderngl_window.loaders.texture.array.Loader',
141: (8)                    ]
142: (4)                """
143: (4)                SCENE_LOADERS: list[str] = []
144: (4)                """
145: (4)                Classes responsible for loading scenes.
146: (4)                .. code:: python
147: (8)                    SCENE_LOADERS = [
148: (12)                       "moderngl_window.loaders.scene.gltf.GLTF2",
149: (12)                       "moderngl_window.loaders.scene.wavefront.ObjLoader",
150: (12)                       "moderngl_window.loaders.scene.stl_loader.STLLoader",
151: (8)                    ]
152: (4)                """
153: (4)                DATA_LOADERS: list[str] = []
154: (4)                """
155: (4)                Classes responsible for loading data files.
156: (4)                .. code:: python
157: (8)                    DATA_LOADERS = [
158: (12)                       'moderngl_window.loaders.data.binary.Loader',
159: (12)                       'moderngl_window.loaders.data.text.Loader',
160: (12)                       'moderngl_window.loaders.data.json.Loader',
161: (8)                    ]
162: (4)                """
163: (4)                def __init__(self) -> None:
164: (8)                    """Initialize settings with default values"""
165: (8)                    self.apply_default_settings()
166: (4)                def apply_default_settings(self) -> None:
```

```
167: (8)                    """
168: (8)                    Apply keys and values from the default settings module
169: (8)                    located in this package. This is to ensure we always
170: (8)                    have the minimal settings for the system to run.
171: (8)                    If replacing or customizing the settings class
172: (8)                    you must always apply default settings to ensure
173: (8)                    compatibility when new settings are added.
174: (8)                    """
175: (8)                    self.apply_from_module(default)
176: (4)                def apply_settings_from_env(self) -> None:
177: (8)                    """
178: (8)                    Apply settings from ``MODERNGL_WINDOW_SETTINGS_MODULE`` environment
variable.
179: (8)                    If the environment variable is undefined no action will be taken.
180: (8)                    Normally this would be used to easily be able to switch between
181: (8)                    different configuration by setting env vars before executing the
program.
182: (8)                    Example::
183: (12)                       import os
184: (12)                       from moderngl_window.conf import settings
185: (12)                       os.environ['MODERNGL_WINDOW_SETTINGS_MODULE'] =
'python.path.to.module'
186: (12)                       settings.apply_settings_from_env()
187: (8)                    Raises:
188: (12)                       ImproperlyConfigured if the module was not found
189: (8)                    """
190: (8)                    name = os.environ.get(SETTINGS_ENV_VAR)
191: (8)                    if name:
192: (12)                       self.apply_from_module_name(name)
193: (4)                def apply_from_module_name(self, settings_module_name: str) -> None:
194: (8)                    """
195: (8)                    Apply settings from a python module by supplying the full
196: (8)                    pythonpath to the module.
197: (8)                    Args:
198: (12)                       settings_module_name (str): Full python path to the module
199: (8)                    Raises:
200: (12)                       ImproperlyConfigured if the module was not found
201: (8)                    """
202: (8)                    try:
203: (12)                       module = importlib.import_module(settings_module_name)
204: (8)                    except ImportError as ex:
205: (12)                       raise ImproperlyConfigured(
206: (16)                           "Settings module '{}' not found. From importlib: {}".format(
207: (20)                               settings_module_name,
208: (20)                               ex,
209: (16)                           )
210: (12)                       )
211: (8)                    self.apply_from_module(module)
212: (4)                def apply_from_dict(self, data: dict[str, Any]) -> None:
213: (8)                    """
214: (8)                    Apply settings values from a dictionary
215: (8)                    Example::
216: (12)                       >> from moderngl_window.conf import settings
217: (12)                       >> settings.apply_dict({'SOME_VALUE': 1})
218: (12)                       >> settings.SOME_VALUE
219: (12)                       1
220: (8)                    """
221: (8)                    self.apply_from_iterable(data.items())
222: (4)                def apply_from_module(self, module: Module) -> None:
223: (8)                    """
224: (8)                    Apply settings values from a python module
225: (8)                    Example::
226: (12)                       my_settings.py module containing the following line:
227: (12)                       SOME_VALUE = 1
228: (12)                       >> from moderngl_window.conf import settings
229: (12)                       >> import my_settings
230: (12)                       >> settings.apply_module(my_settings)
231: (12)                       >> settings.SOME_VALUE
232: (12)                       1
```

```
233: (8)                    """
234: (8)                    self.apply_from_iterable(module.__dict__.items())
235: (4)                def apply_from_cls(self, cls: Any) -> None:
236: (8)                    """
237: (8)                    Apply settings values from a class namespace
238: (8)                    Example::
239: (12)                       >> from moderngl_window.conf import settings
240: (12)                       >> class MySettings:
241: (12)                       >>    SOME_VALUE = 1
242: (12)                       >>
243: (12)                       >> settings.apply(MySettings)
244: (12)                       >> settings.SOME_VALUE
245: (12)                       1
246: (8)                    """
247: (8)                    self.apply_from_iterable(cls.__dict__.items())
248: (4)                def apply_from_iterable(self, iterable: Iterable[tuple[str, Any]]) ->
None:
249: (8)                    """
250: (8)                    Apply (key, value) pairs from an iterable or generator
251: (8)                    """
252: (8)                    if not isinstance(iterable, Iterable) and not isinstance(self,
Generator):
253: (12)                       raise ValueError(
254: (16)                           "Input value is not a generator or iterable, but of type:
{}".format(type(iterable))
255: (12)                       )
256: (8)                    for name, value in iterable:
257: (12)                       if name.isupper():
258: (16)                           setattr(self, name, value)
259: (4)                def to_dict(self) -> dict[str, Any]:
260: (8)                    """Create a dict representation of the settings
261: (8)                    Only uppercase attributes are included
262: (8)                    Returns:
263: (12)                       dict: dict representation
264: (8)                    """
265: (8)                    return {k: v for k, v in self.__dict__.items() if k.upper()}
266: (4)                def __repr__(self) -> str:
267: (8)                    return "\n".join(
268: (12)                       "{}={}".format(k, pformat(v, indent=2)) for k, v in
self.__dict__.items() if k.isupper()
269: (8)                    )
270: (0)            settings = Settings()

----------------------------------------

File 15 - __init__.py:

1: (0)

----------------------------------------

File 16 - __init__.py:

1: (0)          from moderngl_window.context.base.keys import BaseKeys as BaseKeys
2: (0)          from moderngl_window.context.base.keys import KeyModifiers as KeyModifiers
3: (0)          from moderngl_window.context.base.window import BaseWindow as BaseWindow
4: (0)          from moderngl_window.context.base.window import WindowConfig as WindowConfig
5: (0)          __all__ = [
6: (4)              "BaseKeys",
7: (4)              "KeyModifiers",
8: (4)              "BaseWindow",
9: (4)              "WindowConfig",
10: (0)         ]

----------------------------------------

File 17 - __init__.py:

1: (0)          from .keys import GLFW_key, Keys  # noqa
```

```
2: (0)              from .window import Window  # noqa

--------------------------------------

File 18 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

--------------------------------------

File 19 - exceptions.py:

1: (0)              """
2: (0)              Custom exceptions
3: (0)              """
4: (0)              class ImproperlyConfigured(Exception):
5: (4)                  """Raised when finding faulty configuration"""
6: (4)                  pass

--------------------------------------

File 20 - screenshot.py:

1: (0)              import logging
2: (0)              import os
3: (0)              from datetime import datetime
4: (0)              from typing import Optional, Union
5: (0)              import moderngl
6: (0)              from PIL import Image
7: (0)              from moderngl_window.conf import settings
8: (0)              logger = logging.getLogger(__name__)
9: (0)              TEXTURE_MODES = [None, "L", None, "RGB", "RGBA"]
10: (0)             def create(
11: (4)                 source: Union[moderngl.Framebuffer, moderngl.Texture],
12: (4)                 file_format: str = "png",
13: (4)                 name: Optional[str] = None,
14: (4)                 mode: str = "RGB",
15: (4)                 alignment: int = 1,
16: (0)             ) -> None:
17: (4)                 """
18: (4)                 Create a screenshot from a ``moderngl.Framebuffer`` or
``moderngl.Texture``.
19: (4)                 The screenshot will be written to
:py:attr:`~moderngl_window.conf.Settings.SCREENSHOT_PATH`
20: (4)                 if set or ``cwd`` or an absolute path can be used.
21: (4)                 Args:
22: (8)                     source: The framebuffer or texture to screenshot
23: (8)                     file_format (str): formats supported by PIL (png, jpeg etc)
24: (8)                     name (str): Optional file name with relative or absolute path
25: (8)                     mode (str): Components/mode to use
26: (8)                     alignment (int): Buffer alignment
27: (4)                 """
28: (4)                 dest = ""
29: (4)                 if settings.SCREENSHOT_PATH:
30: (8)                     if not os.path.exists(str(settings.SCREENSHOT_PATH)):
31: (12)                        logger.debug("SCREENSHOT_PATH does not exist. creating: %s",
settings.SCREENSHOT_PATH)
32: (12)                        os.makedirs(str(settings.SCREENSHOT_PATH))
33: (8)                     dest = settings.SCREENSHOT_PATH
34: (4)                 else:
35: (8)                     logger.info("SCREENSHOT_PATH not defined in settings. Using cwd as
fallback.")
36: (4)                 if not name:
37: (8)                     name = "{}.{}".format(datetime.now().strftime("%Y-%m-%d-%H-%M-%S-%f"),
file_format)
38: (4)                 logger.debug(
39: (8)                     "Creating screenshot: source=%s file_format=%s name=%s mode=%s
alignment=%s",
40: (8)                     source,
41: (8)                     file_format,
42: (8)                     name,
43: (8)                     mode,
44: (8)                     alignment,
45: (4)                 )
46: (4)                 if isinstance(source, moderngl.Framebuffer):
47: (8)                     image = Image.frombytes(
48: (12)                        mode,
49: (12)                        (
50: (16)                            source.viewport[2] - source.viewport[0],
51: (16)                            source.viewport[3] - source.viewport[1],
52: (12)                        ),
53: (12)                        source.read(viewport=source.viewport, alignment=alignment),
54: (8)                     )
55: (4)                 elif isinstance(source, moderngl.Texture):
56: (8)                     image = Image.frombytes(
57: (12)                        TEXTURE_MODES[source.components], source.size,
source.read(alignment=1)
58: (8)                     )
59: (4)                 else:
60: (8)                     raise ValueError("Source needs to be a FrameBuffer or Texture, not a
%s", type(source))
61: (4)                 image = image.transpose(Image.Transpose.FLIP_TOP_BOTTOM)
62: (4)                 dest = os.path.join(str(dest), name)
63: (4)                 logger.info("Creating screenshot: %s", dest)
64: (4)                 image.save(dest, format=file_format)

----------------------------------------

File 21 - simple_atlas.py:

1: (0)              """
2: (0)              Simple row based texture atlas created for fast runtime allocation.
3: (0)              * This atlas is partly based on the texture atlas in the Arcade project
4: (0)              * The allocator is based on Pyglet's row allocator
5: (0)              https://github.com/pyglet/pyglet/blob/master/pyglet/image/atlas.py
6: (0)
https://github.com/pythonarcade/arcade/blob/development/arcade/texture_atlas.py
7: (0)              """
8: (0)              import moderngl
9: (0)              from .base import BaseImage
10: (0)             class AllocatorException(Exception):
11: (4)                 pass
12: (0)             class _Row:
13: (4)                 """
14: (4)                 A row in the texture atlas.
15: (4)                 """
16: (4)                 __slots__ = ("x", "y", "y2", "max_height")
17: (4)                 def __init__(self, y: int, max_height: int) -> None:
18: (8)                     self.x = 0
19: (8)                     self.y = y
20: (8)                     self.max_height = max_height
21: (8)                     self.y2 = y
22: (4)                 def add(self, width: int, height: int) -> tuple[int, int]:
23: (8)                     """Add a region to the row and return the position"""
24: (8)                     if width <= 0 or height <= 0:
25: (12)                        raise AllocatorException("Cannot allocate size: [{},
{}]".format(width, height))
26: (8)                     if height > self.max_height:
27: (12)                        raise AllocatorException("Cannot allocate past the max height")
28: (8)                     x, y = self.x, self.y
29: (8)                     self.x += width
30: (8)                     self.y2 = max(self.y + height, self.y2)
31: (8)                     return x, y
32: (4)                 def compact(self) -> None:
33: (8)                     """
34: (8)                     Compacts the row to the smallest height.
35: (8)                     Should only be done once when the row is filled before adding a new
```

```
row.
36: (8)                             """
37: (8)                             self.max_height = self.y2 - self.y
38: (0)                     class Allocator:
39: (4)                         """Row based allocator"""
40: (4)                         def __init__(self, width: int, height: int):
41: (8)                             self.width = width
42: (8)                             self.height = height
43: (8)                             self.rows = [_Row(0, self.height)]
44: (4)                         def alloc(self, width: int, height: int) -> tuple[int, int]:
45: (8)                             """
46: (8)                             Allocate a region.
47: (8)                             Returns:
48: (12)                                tuple[int, int]: The x,y location
49: (8)                             Raises:
50: (12)                                AllocatorException: if no more space
51: (8)                             """
52: (8)                             for row in self.rows:
53: (12)                                if self.width - row.x >= width and row.max_height >= height:
54: (16)                                    return row.add(width, height)
55: (8)                             if self.width >= width and self.height - row.y2 >= height:
56: (12)                                row.compact()
57: (12)                                new_row = _Row(row.y2, self.height - row.y2)
58: (12)                                self.rows.append(new_row)
59: (12)                                return new_row.add(width, height)
60: (8)                             raise AllocatorException("No more space in {} for box [{},
{}]".format(self, width, height))
61: (0)                     class TextureAtlas:
62: (4)                         """
63: (4)                         A simple texture atlas using a row based allocation.
64: (4)                         There are more efficient ways to pack textures, but this
65: (4)                         is normally sufficient for dynamic atlases were textures
66: (4)                         are added on the fly runtime.
67: (4)                         """
68: (4)                         def __init__(
69: (8)                             self,
70: (8)                             ctx: moderngl.Context,
71: (8)                             width: int,
72: (8)                             height: int,
73: (8)                             components: int = 4,
74: (8)                             border: int = 1,
75: (8)                             auto_resize: bool = True,
76: (4)                         ):
77: (8)                             self._ctx = ctx
78: (8)                             self._width = width
79: (8)                             self._height = height
80: (8)                             self._components = components
81: (8)                             self._border = border
82: (8)                             self._auto_resize = auto_resize
83: (8)                             self._max_size: tuple[int, int] =
self._ctx.info["GL_MAX_VIEWPORT_DIMS"]
84: (8)                             self._texture = self._ctx.texture(self.size,
components=self._components)
85: (8)                             self._fbo = self._ctx.framebuffer(color_attachments=[self._texture])
86: (8)                             self._allocator = Allocator(width, height)
87: (4)                         @property
88: (4)                         def ctx(self) -> moderngl.Context:
89: (8)                             """The moderngl contex this atlas belongs to"""
90: (8)                             return self._ctx
91: (4)                         @property
92: (4)                         def textrue(self) -> moderngl.Texture:
93: (8)                             """The moderngl texture with the atlas contents"""
94: (8)                             return self._texture
95: (4)                         @property
96: (4)                         def width(self) -> int:
97: (8)                             """int: Width of the atlas in pixels"""
98: (8)                             return self._width
99: (4)                         @property
100: (4)                        def height(self) -> int:
```

```
(8)                 """int: Height of the atlas in pixels"""
(8)                 return self._height
(4)             @property
(4)             def size(self) -> tuple[int, int]:
(8)                 """tuple[int, int]: The size of he atlas (width, height)"""
(8)                 return self._width, self._height
(4)             @property
(4)             def max_size(self) -> tuple[int, int]:
(8)                 """
(8)                 tuple[int,int]: The maximum size of the atlas in pixels (x, y)
(8)                 """
(8)                 return self._max_size
(4)             def add(self, image: BaseImage) -> None:
(8)                 pass
(4)             def remove(self, image: BaseImage) -> None:
(8)                 pass
(4)             def resize(self, width: int, height: int) -> None:
(8)                 pass
(4)             def rebuild(self) -> None:
(8)                 pass
```

File 22 - keys.py:

```
(0)             import pygame
(0)             from moderngl_window.context.base import BaseKeys
(0)             class Keys(BaseKeys):
(4)                 """
(4)                 Namespace mapping pygame2 specific key constants
(4)                 """
(4)                 ACTION_PRESS = pygame.KEYDOWN
(4)                 ACTION_RELEASE = pygame.KEYUP
(4)                 ESCAPE = pygame.K_ESCAPE
(4)                 SPACE = pygame.K_SPACE
(4)                 ENTER = pygame.K_RETURN
(4)                 PAGE_UP = pygame.K_PAGEUP
(4)                 PAGE_DOWN = pygame.K_PAGEDOWN
(4)                 LEFT = pygame.K_LEFT
(4)                 RIGHT = pygame.K_RIGHT
(4)                 UP = pygame.K_UP
(4)                 DOWN = pygame.K_DOWN
(4)                 LEFT_SHIFT = pygame.K_LSHIFT
(4)                 RIGHT_SHIFT = pygame.K_RSHIFT
(4)                 LEFT_CTRL = pygame.K_LCTRL
(4)                 TAB = pygame.K_TAB
(4)                 COMMA = pygame.K_COMMA
(4)                 MINUS = pygame.K_MINUS
(4)                 PERIOD = pygame.K_PERIOD
(4)                 SLASH = pygame.K_SLASH
(4)                 SEMICOLON = pygame.K_SEMICOLON
(4)                 EQUAL = pygame.K_EQUALS
(4)                 LEFT_BRACKET = pygame.K_LEFTBRACKET
(4)                 RIGHT_BRACKET = pygame.K_RIGHTBRACKET
(4)                 BACKSLASH = pygame.K_BACKSLASH
(4)                 BACKSPACE = pygame.K_BACKSPACE
(4)                 INSERT = pygame.K_INSERT
(4)                 DELETE = pygame.K_DELETE
(4)                 HOME = pygame.K_HOME
(4)                 END = pygame.K_END
(4)                 CAPS_LOCK = pygame.K_CAPSLOCK
(4)                 F1 = pygame.K_F1
(4)                 F2 = pygame.K_F2
(4)                 F3 = pygame.K_F3
(4)                 F4 = pygame.K_F4
(4)                 F5 = pygame.K_F5
(4)                 F6 = pygame.K_F6
(4)                 F7 = pygame.K_F7
(4)                 F8 = pygame.K_F8
```

```
45: (4)                 F9 = pygame.K_F9
46: (4)                 F10 = pygame.K_F10
47: (4)                 F11 = pygame.K_F11
48: (4)                 F12 = pygame.K_F12
49: (4)                 NUMBER_0 = pygame.K_0
50: (4)                 NUMBER_1 = pygame.K_1
51: (4)                 NUMBER_2 = pygame.K_2
52: (4)                 NUMBER_3 = pygame.K_3
53: (4)                 NUMBER_4 = pygame.K_4
54: (4)                 NUMBER_5 = pygame.K_5
55: (4)                 NUMBER_6 = pygame.K_6
56: (4)                 NUMBER_7 = pygame.K_7
57: (4)                 NUMBER_8 = pygame.K_8
58: (4)                 NUMBER_9 = pygame.K_9
59: (4)                 NUMPAD_0 = pygame.K_KP_0
60: (4)                 NUMPAD_1 = pygame.K_KP_1
61: (4)                 NUMPAD_2 = pygame.K_KP_2
62: (4)                 NUMPAD_3 = pygame.K_KP_3
63: (4)                 NUMPAD_4 = pygame.K_KP_4
64: (4)                 NUMPAD_5 = pygame.K_KP_5
65: (4)                 NUMPAD_6 = pygame.K_KP_6
66: (4)                 NUMPAD_7 = pygame.K_KP_7
67: (4)                 NUMPAD_8 = pygame.K_KP_8
68: (4)                 NUMPAD_9 = pygame.K_KP_9
69: (4)                 A = pygame.K_a
70: (4)                 B = pygame.K_b
71: (4)                 C = pygame.K_c
72: (4)                 D = pygame.K_d
73: (4)                 E = pygame.K_e
74: (4)                 F = pygame.K_f
75: (4)                 G = pygame.K_g
76: (4)                 H = pygame.K_h
77: (4)                 I = pygame.K_i
78: (4)                 J = pygame.K_j
79: (4)                 K = pygame.K_k
80: (4)                 L = pygame.K_l
81: (4)                 M = pygame.K_m
82: (4)                 N = pygame.K_n
83: (4)                 O = pygame.K_o
84: (4)                 P = pygame.K_p
85: (4)                 Q = pygame.K_q
86: (4)                 R = pygame.K_r
87: (4)                 S = pygame.K_s
88: (4)                 T = pygame.K_t
89: (4)                 U = pygame.K_u
90: (4)                 V = pygame.K_v
91: (4)                 W = pygame.K_w
92: (4)                 X = pygame.K_x
93: (4)                 Y = pygame.K_y
94: (4)                 Z = pygame.K_z

----------------------------------------

File 23 - keys.py:

1: (0)              import platform
2: (0)              import pyglet
3: (0)              if platform.system() == "Darwin":
4: (4)                  pyglet.options["shadow_window"] = False
5: (0)              pyglet.options["debug_gl"] = False
6: (0)              from pyglet.window import key
7: (0)              from moderngl_window.context.base import BaseKeys
8: (0)              class Keys(BaseKeys):
9: (4)                  """
10: (4)                 Namespace mapping pyglet specific key constants
11: (4)                 """
12: (4)                 ESCAPE = key.ESCAPE
13: (4)                 SPACE = key.SPACE
14: (4)                 ENTER = key.ENTER
```

```
15: (4)                 PAGE_UP = key.PAGEUP
16: (4)                 PAGE_DOWN = key.PAGEDOWN
17: (4)                 LEFT = key.LEFT
18: (4)                 RIGHT = key.RIGHT
19: (4)                 UP = key.UP
20: (4)                 DOWN = key.DOWN
21: (4)                 LEFT_SHIFT = key.LSHIFT
22: (4)                 RIGHT_SHIFT = key.RSHIFT
23: (4)                 LEFT_CTRL = key.LCTRL
24: (4)                 TAB = key.TAB
25: (4)                 COMMA = key.COMMA
26: (4)                 MINUS = key.MINUS
27: (4)                 PERIOD = key.PERIOD
28: (4)                 SLASH = key.SLASH
29: (4)                 SEMICOLON = key.SEMICOLON
30: (4)                 EQUAL = key.EQUAL
31: (4)                 LEFT_BRACKET = key.BRACELEFT
32: (4)                 RIGHT_BRACKET = key.BRACERIGHT
33: (4)                 BACKSLASH = key.BACKSLASH
34: (4)                 BACKSPACE = key.BACKSPACE
35: (4)                 INSERT = key.INSERT
36: (4)                 DELETE = key.DELETE
37: (4)                 HOME = key.HOME
38: (4)                 END = key.END
39: (4)                 CAPS_LOCK = key.CAPSLOCK
40: (4)                 F1 = key.F1
41: (4)                 F2 = key.F2
42: (4)                 F3 = key.F3
43: (4)                 F4 = key.F4
44: (4)                 F5 = key.F5
45: (4)                 F6 = key.F6
46: (4)                 F7 = key.F7
47: (4)                 F8 = key.F8
48: (4)                 F9 = key.F9
49: (4)                 F10 = key.F10
50: (4)                 F11 = key.F11
51: (4)                 F12 = key.F12
52: (4)                 NUMBER_0 = key._0
53: (4)                 NUMBER_1 = key._1
54: (4)                 NUMBER_2 = key._2
55: (4)                 NUMBER_3 = key._3
56: (4)                 NUMBER_4 = key._4
57: (4)                 NUMBER_5 = key._5
58: (4)                 NUMBER_6 = key._6
59: (4)                 NUMBER_7 = key._7
60: (4)                 NUMBER_8 = key._8
61: (4)                 NUMBER_9 = key._9
62: (4)                 NUMPAD_0 = key.NUM_0
63: (4)                 NUMPAD_1 = key.NUM_1
64: (4)                 NUMPAD_2 = key.NUM_2
65: (4)                 NUMPAD_3 = key.NUM_3
66: (4)                 NUMPAD_4 = key.NUM_4
67: (4)                 NUMPAD_5 = key.NUM_5
68: (4)                 NUMPAD_6 = key.NUM_6
69: (4)                 NUMPAD_7 = key.NUM_7
70: (4)                 NUMPAD_8 = key.NUM_8
71: (4)                 NUMPAD_9 = key.NUM_9
72: (4)                 A = key.A
73: (4)                 B = key.B
74: (4)                 C = key.C
75: (4)                 D = key.D
76: (4)                 E = key.E
77: (4)                 F = key.F
78: (4)                 G = key.G
79: (4)                 H = key.H
80: (4)                 I = key.I
81: (4)                 J = key.J
82: (4)                 K = key.K
83: (4)                 L = key.L
```

```
84: (4)                 M = key.M
85: (4)                 N = key.N
86: (4)                 O = key.O
87: (4)                 P = key.P
88: (4)                 Q = key.Q
89: (4)                 R = key.R
90: (4)                 S = key.S
91: (4)                 T = key.T
92: (4)                 U = key.U
93: (4)                 V = key.V
94: (4)                 W = key.W
95: (4)                 X = key.X
96: (4)                 Y = key.Y
97: (4)                 Z = key.Z

----------------------------------------

File 24 - keys.py:

1: (0)              from PyQt5.QtCore import Qt
2: (0)              from moderngl_window.context.base import BaseKeys
3: (0)              class Keys(BaseKeys):
4: (4)                  """
5: (4)                  Namespace mapping pyqt specific key constants
6: (4)                  """
7: (4)                  ESCAPE = Qt.Key_Escape
8: (4)                  SPACE = Qt.Key_Space
9: (4)                  ENTER = Qt.Key_Return
10: (4)                 PAGE_UP = Qt.Key_PageUp
11: (4)                 PAGE_DOWN = Qt.Key_PageDown
12: (4)                 LEFT = Qt.Key_Left
13: (4)                 RIGHT = Qt.Key_Right
14: (4)                 UP = Qt.Key_Up
15: (4)                 DOWN = Qt.Key_Down
16: (4)                 TAB = Qt.Key_Tab
17: (4)                 COMMA = Qt.Key_Comma
18: (4)                 MINUS = Qt.Key_Minus
19: (4)                 PERIOD = Qt.Key_Period
20: (4)                 SLASH = Qt.Key_Slash
21: (4)                 SEMICOLON = Qt.Key_Semicolon
22: (4)                 EQUAL = Qt.Key_Equal
23: (4)                 LEFT_BRACKET = Qt.Key_BracketLeft
24: (4)                 RIGHT_BRACKET = Qt.Key_BracketRight
25: (4)                 BACKSLASH = Qt.Key_Backslash
26: (4)                 BACKSPACE = Qt.Key_Backspace
27: (4)                 INSERT = Qt.Key_Insert
28: (4)                 DELETE = Qt.Key_Delete
29: (4)                 HOME = Qt.Key_Home
30: (4)                 END = Qt.Key_End
31: (4)                 CAPS_LOCK = Qt.Key_CapsLock
32: (4)                 F1 = Qt.Key_F1
33: (4)                 F2 = Qt.Key_F2
34: (4)                 F3 = Qt.Key_F3
35: (4)                 F4 = Qt.Key_F4
36: (4)                 F5 = Qt.Key_F5
37: (4)                 F6 = Qt.Key_F6
38: (4)                 F7 = Qt.Key_F7
39: (4)                 F8 = Qt.Key_F8
40: (4)                 F9 = Qt.Key_F9
41: (4)                 F10 = Qt.Key_F10
42: (4)                 F11 = Qt.Key_F11
43: (4)                 F12 = Qt.Key_F12
44: (4)                 NUMBER_0 = Qt.Key_0
45: (4)                 NUMBER_1 = Qt.Key_1
46: (4)                 NUMBER_2 = Qt.Key_2
47: (4)                 NUMBER_3 = Qt.Key_3
48: (4)                 NUMBER_4 = Qt.Key_4
49: (4)                 NUMBER_5 = Qt.Key_5
50: (4)                 NUMBER_6 = Qt.Key_6
```

```
51: (4)                 NUMBER_7 = Qt.Key_7
52: (4)                 NUMBER_8 = Qt.Key_8
53: (4)                 NUMBER_9 = Qt.Key_9
54: (4)                 NUMPAD_0 = Qt.Key_0
55: (4)                 NUMPAD_1 = Qt.Key_1
56: (4)                 NUMPAD_2 = Qt.Key_2
57: (4)                 NUMPAD_3 = Qt.Key_3
58: (4)                 NUMPAD_4 = Qt.Key_4
59: (4)                 NUMPAD_5 = Qt.Key_5
60: (4)                 NUMPAD_6 = Qt.Key_6
61: (4)                 NUMPAD_7 = Qt.Key_7
62: (4)                 NUMPAD_8 = Qt.Key_8
63: (4)                 NUMPAD_9 = Qt.Key_9
64: (4)                 A = Qt.Key_A
65: (4)                 B = Qt.Key_B
66: (4)                 C = Qt.Key_C
67: (4)                 D = Qt.Key_D
68: (4)                 E = Qt.Key_E
69: (4)                 F = Qt.Key_F
70: (4)                 G = Qt.Key_G
71: (4)                 H = Qt.Key_H
72: (4)                 I = Qt.Key_I
73: (4)                 J = Qt.Key_J
74: (4)                 K = Qt.Key_K
75: (4)                 L = Qt.Key_L
76: (4)                 M = Qt.Key_M
77: (4)                 N = Qt.Key_N
78: (4)                 O = Qt.Key_O
79: (4)                 P = Qt.Key_P
80: (4)                 Q = Qt.Key_Q
81: (4)                 R = Qt.Key_R
82: (4)                 S = Qt.Key_S
83: (4)                 T = Qt.Key_T
84: (4)                 U = Qt.Key_U
85: (4)                 V = Qt.Key_V
86: (4)                 W = Qt.Key_W
87: (4)                 X = Qt.Key_X
88: (4)                 Y = Qt.Key_Y
89: (4)                 Z = Qt.Key_Z

----------------------------------------

File 25 - keys.py:

1: (0)              from PySide2.QtCore import Qt
2: (0)              from moderngl_window.context.base import BaseKeys
3: (0)              class Keys(BaseKeys):
4: (4)                  """
5: (4)                  Namespace mapping pyside2 specific key constants
6: (4)                  """
7: (4)                  ESCAPE = Qt.Key_Escape
8: (4)                  SPACE = Qt.Key_Space
9: (4)                  ENTER = Qt.Key_Enter
10: (4)                 PAGE_UP = Qt.Key_PageUp
11: (4)                 PAGE_DOWN = Qt.Key_PageDown
12: (4)                 LEFT = Qt.Key_Left
13: (4)                 RIGHT = Qt.Key_Right
14: (4)                 UP = Qt.Key_Up
15: (4)                 DOWN = Qt.Key_Down
16: (4)                 TAB = Qt.Key_Tab
17: (4)                 COMMA = Qt.Key_Comma
18: (4)                 MINUS = Qt.Key_Minus
19: (4)                 PERIOD = Qt.Key_Period
20: (4)                 SLASH = Qt.Key_Slash
21: (4)                 SEMICOLON = Qt.Key_Semicolon
22: (4)                 EQUAL = Qt.Key_Equal
23: (4)                 LEFT_BRACKET = Qt.Key_BracketLeft
24: (4)                 RIGHT_BRACKET = Qt.Key_BracketRight
25: (4)                 BACKSLASH = Qt.Key_Backslash
```

```
26: (4)                 BACKSPACE = Qt.Key_Backspace
27: (4)                 INSERT = Qt.Key_Insert
28: (4)                 DELETE = Qt.Key_Delete
29: (4)                 HOME = Qt.Key_Home
30: (4)                 END = Qt.Key_End
31: (4)                 CAPS_LOCK = Qt.Key_CapsLock
32: (4)                 F1 = Qt.Key_F1
33: (4)                 F2 = Qt.Key_F2
34: (4)                 F3 = Qt.Key_F3
35: (4)                 F4 = Qt.Key_F4
36: (4)                 F5 = Qt.Key_F5
37: (4)                 F6 = Qt.Key_F6
38: (4)                 F7 = Qt.Key_F7
39: (4)                 F8 = Qt.Key_F8
40: (4)                 F9 = Qt.Key_F9
41: (4)                 F10 = Qt.Key_F10
42: (4)                 F11 = Qt.Key_F11
43: (4)                 F12 = Qt.Key_F12
44: (4)                 NUMBER_0 = Qt.Key_0
45: (4)                 NUMBER_1 = Qt.Key_1
46: (4)                 NUMBER_2 = Qt.Key_2
47: (4)                 NUMBER_3 = Qt.Key_3
48: (4)                 NUMBER_4 = Qt.Key_4
49: (4)                 NUMBER_5 = Qt.Key_5
50: (4)                 NUMBER_6 = Qt.Key_6
51: (4)                 NUMBER_7 = Qt.Key_7
52: (4)                 NUMBER_8 = Qt.Key_8
53: (4)                 NUMBER_9 = Qt.Key_9
54: (4)                 NUMPAD_0 = Qt.Key_0
55: (4)                 NUMPAD_1 = Qt.Key_1
56: (4)                 NUMPAD_2 = Qt.Key_2
57: (4)                 NUMPAD_3 = Qt.Key_3
58: (4)                 NUMPAD_4 = Qt.Key_4
59: (4)                 NUMPAD_5 = Qt.Key_5
60: (4)                 NUMPAD_6 = Qt.Key_6
61: (4)                 NUMPAD_7 = Qt.Key_7
62: (4)                 NUMPAD_8 = Qt.Key_8
63: (4)                 NUMPAD_9 = Qt.Key_9
64: (4)                 A = Qt.Key_A
65: (4)                 B = Qt.Key_B
66: (4)                 C = Qt.Key_C
67: (4)                 D = Qt.Key_D
68: (4)                 E = Qt.Key_E
69: (4)                 F = Qt.Key_F
70: (4)                 G = Qt.Key_G
71: (4)                 H = Qt.Key_H
72: (4)                 I = Qt.Key_I
73: (4)                 J = Qt.Key_J
74: (4)                 K = Qt.Key_K
75: (4)                 L = Qt.Key_L
76: (4)                 M = Qt.Key_M
77: (4)                 N = Qt.Key_N
78: (4)                 O = Qt.Key_O
79: (4)                 P = Qt.Key_P
80: (4)                 Q = Qt.Key_Q
81: (4)                 R = Qt.Key_R
82: (4)                 S = Qt.Key_S
83: (4)                 T = Qt.Key_T
84: (4)                 U = Qt.Key_U
85: (4)                 V = Qt.Key_V
86: (4)                 W = Qt.Key_W
87: (4)                 X = Qt.Key_X
88: (4)                 Y = Qt.Key_Y
89: (4)                 Z = Qt.Key_Z

----------------------------------------

File 26 - keys.py:
```

```
(0)              import sdl2
(0)              from moderngl_window.context.base import BaseKeys
(0)              class Keys(BaseKeys):
(4)                  """
(4)                  Namespace mapping SDL2 specific key constants
(4)                  """
(4)                  ACTION_PRESS = sdl2.SDL_KEYDOWN
(4)                  ACTION_RELEASE = sdl2.SDL_KEYUP
(4)                  ESCAPE = sdl2.SDLK_ESCAPE
(4)                  SPACE = sdl2.SDLK_SPACE
(4)                  ENTER = sdl2.SDLK_RETURN
(4)                  PAGE_UP = sdl2.SDLK_PAGEUP
(4)                  PAGE_DOWN = sdl2.SDLK_PAGEDOWN
(4)                  LEFT = sdl2.SDLK_LEFT
(4)                  RIGHT = sdl2.SDLK_RIGHT
(4)                  UP = sdl2.SDLK_UP
(4)                  DOWN = sdl2.SDLK_DOWN
(4)                  TAB = sdl2.SDLK_TAB
(4)                  COMMA = sdl2.SDLK_COMMA
(4)                  MINUS = sdl2.SDLK_MINUS
(4)                  PERIOD = sdl2.SDLK_PERIOD
(4)                  SLASH = sdl2.SDLK_SLASH
(4)                  SEMICOLON = sdl2.SDLK_SEMICOLON
(4)                  EQUAL = sdl2.SDLK_EQUALS
(4)                  LEFT_BRACKET = sdl2.SDLK_LEFTBRACKET
(4)                  RIGHT_BRACKET = sdl2.SDLK_RIGHTBRACKET
(4)                  BACKSLASH = sdl2.SDLK_BACKSLASH
(4)                  BACKSPACE = sdl2.SDLK_BACKSPACE
(4)                  INSERT = sdl2.SDLK_INSERT
(4)                  DELETE = sdl2.SDLK_DELETE
(4)                  HOME = sdl2.SDLK_HOME
(4)                  END = sdl2.SDLK_END
(4)                  CAPS_LOCK = sdl2.SDLK_CAPSLOCK
(4)                  F1 = sdl2.SDLK_F1
(4)                  F2 = sdl2.SDLK_F2
(4)                  F3 = sdl2.SDLK_F3
(4)                  F4 = sdl2.SDLK_F4
(4)                  F5 = sdl2.SDLK_F5
(4)                  F6 = sdl2.SDLK_F6
(4)                  F7 = sdl2.SDLK_F7
(4)                  F8 = sdl2.SDLK_F8
(4)                  F9 = sdl2.SDLK_F9
(4)                  F10 = sdl2.SDLK_F10
(4)                  F11 = sdl2.SDLK_F11
(4)                  F12 = sdl2.SDLK_F12
(4)                  NUMBER_0 = sdl2.SDLK_0
(4)                  NUMBER_1 = sdl2.SDLK_1
(4)                  NUMBER_2 = sdl2.SDLK_2
(4)                  NUMBER_3 = sdl2.SDLK_3
(4)                  NUMBER_4 = sdl2.SDLK_4
(4)                  NUMBER_5 = sdl2.SDLK_5
(4)                  NUMBER_6 = sdl2.SDLK_6
(4)                  NUMBER_7 = sdl2.SDLK_7
(4)                  NUMBER_8 = sdl2.SDLK_8
(4)                  NUMBER_9 = sdl2.SDLK_9
(4)                  NUMPAD_0 = sdl2.SDLK_KP_0
(4)                  NUMPAD_1 = sdl2.SDLK_KP_1
(4)                  NUMPAD_2 = sdl2.SDLK_KP_2
(4)                  NUMPAD_3 = sdl2.SDLK_KP_3
(4)                  NUMPAD_4 = sdl2.SDLK_KP_4
(4)                  NUMPAD_5 = sdl2.SDLK_KP_5
(4)                  NUMPAD_6 = sdl2.SDLK_KP_6
(4)                  NUMPAD_7 = sdl2.SDLK_KP_7
(4)                  NUMPAD_8 = sdl2.SDLK_KP_8
(4)                  NUMPAD_9 = sdl2.SDLK_KP_9
(4)                  A = sdl2.SDLK_a
(4)                  B = sdl2.SDLK_b
(4)                  C = sdl2.SDLK_c
(4)                  D = sdl2.SDLK_d
```

```
70: (4)                 E = sdl2.SDLK_e
71: (4)                 F = sdl2.SDLK_f
72: (4)                 G = sdl2.SDLK_g
73: (4)                 H = sdl2.SDLK_h
74: (4)                 I = sdl2.SDLK_i
75: (4)                 J = sdl2.SDLK_j
76: (4)                 K = sdl2.SDLK_k
77: (4)                 L = sdl2.SDLK_l
78: (4)                 M = sdl2.SDLK_m
79: (4)                 N = sdl2.SDLK_n
80: (4)                 O = sdl2.SDLK_o
81: (4)                 P = sdl2.SDLK_p
82: (4)                 Q = sdl2.SDLK_q
83: (4)                 R = sdl2.SDLK_r
84: (4)                 S = sdl2.SDLK_s
85: (4)                 T = sdl2.SDLK_t
86: (4)                 U = sdl2.SDLK_u
87: (4)                 V = sdl2.SDLK_v
88: (4)                 W = sdl2.SDLK_w
89: (4)                 X = sdl2.SDLK_x
90: (4)                 Y = sdl2.SDLK_y
91: (4)                 Z = sdl2.SDLK_z

----------------------------------------

File 27 - keys.py:

1: (0)              from moderngl_window.context.base import BaseKeys
2: (0)              class Keys(BaseKeys):
3: (4)                  """
4: (4)                  Namespace mapping tkinter keys.
5: (4)                  Maps the keysym strings provided in tk.Events
6: (4)                  """
7: (4)                  ESCAPE = "Escape"
8: (4)                  SPACE = " "
9: (4)                  ENTER = "Return"
10: (4)                 PAGE_UP = "Prior"
11: (4)                 PAGE_DOWN = "Next"
12: (4)                 LEFT = "Left"
13: (4)                 RIGHT = "Right"
14: (4)                 UP = "Up"
15: (4)                 DOWN = "Down"
16: (4)                 TAB = "Tab"
17: (4)                 COMMA = "comma"
18: (4)                 MINUS = "minus"
19: (4)                 PERIOD = "period"
20: (4)                 SLASH = "slash"
21: (4)                 SEMICOLON = "semicolon"
22: (4)                 EQUAL = "equal"
23: (4)                 LEFT_BRACKET = "bracketleft"
24: (4)                 RIGHT_BRACKET = "bracketright"
25: (4)                 BACKSLASH = "backslash"
26: (4)                 BACKSPACE = "BackSpace"
27: (4)                 INSERT = "Insert"
28: (4)                 DELETE = "Delete"
29: (4)                 HOME = "Home"
30: (4)                 END = "End"
31: (4)                 CAPS_LOCK = "Caps_Lock"
32: (4)                 F1 = "F1"
33: (4)                 F2 = "F2"
34: (4)                 F3 = "F3"
35: (4)                 F4 = "F4"
36: (4)                 F5 = "F5"
37: (4)                 F6 = "F6"
38: (4)                 F7 = "F7"
39: (4)                 F8 = "F8"
40: (4)                 F9 = "F9"
41: (4)                 F10 = "F10"
42: (4)                 F11 = "F11"
```

```
43: (4)                 F12 = "F12"
44: (4)                 NUMBER_0 = "0"
45: (4)                 NUMBER_1 = "1"
46: (4)                 NUMBER_2 = "2"
47: (4)                 NUMBER_3 = "3"
48: (4)                 NUMBER_4 = "4"
49: (4)                 NUMBER_5 = "5"
50: (4)                 NUMBER_6 = "6"
51: (4)                 NUMBER_7 = "7"
52: (4)                 NUMBER_8 = "8"
53: (4)                 NUMBER_9 = "9"
54: (4)                 NUMPAD_0 = "0"
55: (4)                 NUMPAD_1 = "1"
56: (4)                 NUMPAD_2 = "2"
57: (4)                 NUMPAD_3 = "3"
58: (4)                 NUMPAD_4 = "4"
59: (4)                 NUMPAD_5 = "5"
60: (4)                 NUMPAD_6 = "6"
61: (4)                 NUMPAD_7 = "7"
62: (4)                 NUMPAD_8 = "8"
63: (4)                 NUMPAD_9 = "9"
64: (4)                 A = "a"
65: (4)                 B = "b"
66: (4)                 C = "c"
67: (4)                 D = "d"
68: (4)                 E = "e"
69: (4)                 F = "f"
70: (4)                 G = "g"
71: (4)                 H = "h"
72: (4)                 I = "i"
73: (4)                 J = "j"
74: (4)                 K = "k"
75: (4)                 L = "l"
76: (4)                 M = "m"
77: (4)                 N = "n"
78: (4)                 O = "o"
79: (4)                 P = "p"
80: (4)                 Q = "q"
81: (4)                 R = "r"
82: (4)                 S = "s"
83: (4)                 T = "t"
84: (4)                 U = "u"
85: (4)                 V = "v"
86: (4)                 W = "w"
87: (4)                 X = "x"
88: (4)                 Y = "y"
89: (4)                 Z = "z"
```

File 28 - window.py:

```
1: (0)              from pathlib import Path
2: (0)              from typing import Any
3: (0)              import pygame
4: (0)              import pygame._sdl2
5: (0)              import pygame.display
6: (0)              import pygame.event
7: (0)              from moderngl_window.context.base import BaseWindow
8: (0)              from moderngl_window.context.pygame2.keys import Keys
9: (0)              class Window(BaseWindow):
10: (4)                 """
11: (4)                 Basic window implementation using pygame2.
12: (4)                 """
13: (4)                 name = "pygame2"
14: (4)                 keys = Keys
15: (4)                 _mouse_button_map = {
16: (8)                     1: 1,
17: (8)                     3: 2,
18: (8)                     2: 3,
19: (4)                 }
20: (4)                 def __init__(self, **kwargs: Any):
21: (8)                     super().__init__(**kwargs)
22: (8)                     pygame.display.init()
23: (8)                     pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MAJOR_VERSION,
self.gl_version[0])
24: (8)                     pygame.display.gl_set_attribute(pygame.GL_CONTEXT_MINOR_VERSION,
self.gl_version[1])
25: (8)                     pygame.display.gl_set_attribute(
26: (12)                        pygame.GL_CONTEXT_PROFILE_MASK, pygame.GL_CONTEXT_PROFILE_CORE
27: (8)                     )
28: (8)
pygame.display.gl_set_attribute(pygame.GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, 1)
29: (8)                     pygame.display.gl_set_attribute(pygame.GL_DOUBLEBUFFER, 1)
30: (8)                     pygame.display.gl_set_attribute(pygame.GL_DEPTH_SIZE, 24)
31: (8)                     pygame.display.gl_set_attribute(pygame.GL_STENCIL_SIZE, 8)
32: (8)                     if self.samples > 1:
33: (12)                        pygame.display.gl_set_attribute(pygame.GL_MULTISAMPLEBUFFERS, 1)
34: (12)                        pygame.display.gl_set_attribute(pygame.GL_MULTISAMPLESAMPLES,
self.samples)
35: (8)                     self._depth = 24
36: (8)                     self._flags = pygame.OPENGL | pygame.DOUBLEBUF
37: (8)                     if self.resizable:
38: (12)                        self._flags |= pygame.RESIZABLE
39: (8)                     if not self._visible:
40: (12)                        self._flags |= pygame.HIDDEN
41: (8)                     self._set_mode()
42: (8)                     self.title = self._title
43: (8)                     self.cursor = self._cursor
44: (8)                     self._sdl_window = pygame._sdl2.video.Window.from_display_module()
45: (8)                     if self.fullscreen:
46: (12)                        self._set_fullscreen(True)
47: (8)                     self.init_mgl_context()
48: (8)                     self.set_default_viewport()
49: (4)                 def _set_mode(self) -> None:
50: (8)                     self._surface = pygame.display.set_mode(
51: (12)                        size=(self._width, self._height),
52: (12)                        flags=self._flags,
53: (12)                        depth=self._depth,
54: (12)                        vsync=self._vsync,
55: (8)                     )
56: (4)                 def _set_fullscreen(self, value: bool) -> None:
57: (8)                     if value:
58: (12)                        self._sdl_window.set_fullscreen(True)
59: (8)                     else:
60: (12)                        self._sdl_window.set_windowed()
61: (4)                 def _set_vsync(self, value: bool) -> None:
62: (8)                     self._vsync = value
63: (8)                     self._set_mode()
64: (4)                 @property
65: (4)                 def size(self) -> tuple[int, int]:
66: (8)                     """tuple[int, int]: current window size.
67: (8)                     This property also support assignment::
68: (12)                        window.size = 1000, 1000
69: (8)                     """
70: (8)                     return self._width, self._height
71: (4)                 @size.setter
72: (4)                 def size(self, value: tuple[int, int]) -> None:
73: (8)                     self._width, self._height = value
74: (8)                     self._set_mode()
75: (8)                     self.resize(value[0], value[1])
76: (4)                 @property
77: (4)                 def position(self) -> tuple[int, int]:
78: (8)                     """tuple[int, int]: The current window position.
79: (8)                     This property can also be set to move the window::
80: (12)                        window.position = 100, 100
81: (8)                     """
82: (8)                     return self._sdl_window.position
```

```
(4)                 @position.setter
(4)                 def position(self, value: tuple[int, int]) -> None:
(8)                     self._sdl_window.position = value
(4)                 @property
(4)                 def visible(self) -> bool:
(8)                     """bool: Is the window visible?
(8)                     This property can also be set::
(12)                        window.visible = False
(8)                     """
(8)                     return self._visible
(4)                 @visible.setter
(4)                 def visible(self, value: bool) -> None:
(8)                     self._visible = value
(8)                     if value:
(12)                        self._sdl_window.show()
(8)                     else:
(12)                        self._sdl_window.hide()
(4)                 @property
(4)                 def cursor(self) -> bool:
(8)                     """bool: Should the mouse cursor be visible inside the window?
(8)                     This property can also be assigned to::
(12)                        window.cursor = False
(8)                     """
(8)                     return self._cursor
(4)                 @cursor.setter
(4)                 def cursor(self, value: bool) -> None:
(8)                     pygame.mouse.set_visible(value)
(8)                     self._cursor = value
(4)                 @property
(4)                 def mouse_exclusivity(self) -> bool:
(8)                     """bool: If mouse exclusivity is enabled.
(8)                     When you enable mouse-exclusive mode, the mouse cursor is no longer
(8)                     available. It is not merely hidden – no amount of mouse movement
(8)                     will make it leave your application. This is for example useful
(8)                     when you don't want the mouse leaving the screen when rotating
(8)                     a 3d scene.
(8)                     This property can also be set::
(12)                        window.mouse_exclusivity = True
(8)                     """
(8)                     return self._mouse_exclusivity
(4)                 @mouse_exclusivity.setter
(4)                 def mouse_exclusivity(self, value: bool) -> None:
(8)                     if self._cursor:
(12)                        self.cursor = False
(8)                     pygame.event.set_grab(value)
(8)                     self._mouse_exclusivity = value
(4)                 @property
(4)                 def title(self) -> str:
(8)                     """str: Window title.
(8)                     This property can also be set::
(12)                        window.title = "New Title"
(8)                     """
(8)                     return self._title
(4)                 @title.setter
(4)                 def title(self, value: str) -> None:
(8)                     pygame.display.set_caption(value)
(8)                     self._title = value
(4)                 def swap_buffers(self) -> None:
(8)                     """Swap buffers, set viewport, trigger events and increment frame
counter"""
(8)                     pygame.display.flip()
(8)                     self.set_default_viewport()
(8)                     self.process_events()
(8)                     self._frames += 1
(4)                 def _set_icon(self, icon_path: Path) -> None:
(8)                     icon = pygame.image.load(icon_path)
(8)                     pygame.display.set_icon(icon)
(4)                 def resize(self, width: int, height: int) -> None:
(8)                     """Resize callback
```

```
151: (8)                    Args:
152: (12)                       width: New window width
153: (12)                       height: New window height
154: (8)                    """
155: (8)                    self._width = width
156: (8)                    self._height = height
157: (8)                    self._buffer_width, self._buffer_height = self._width, self._height
158: (8)                    self.set_default_viewport()
159: (8)                    super().resize(self._buffer_width, self._buffer_height)
160: (4)                def close(self) -> None:
161: (8)                    """Close the window"""
162: (8)                    super().close()
163: (8)                    self._close_func()
164: (4)                def _handle_mods(self) -> None:
165: (8)                    """Update key mods"""
166: (8)                    mods = pygame.key.get_mods()
167: (8)                    self._modifiers.shift = mods & pygame.KMOD_SHIFT
168: (8)                    self._modifiers.ctrl = mods & pygame.KMOD_CTRL
169: (8)                    self._modifiers.alt = mods & pygame.KMOD_ALT
170: (4)                def process_events(self) -> None:
171: (8)                    """Handle all queued events in pygame2 dispatching events to standard
methods"""
172: (8)                    for event in pygame.event.get():
173: (12)                       if event.type == pygame.MOUSEMOTION:
174: (16)                           self._handle_mods()
175: (16)                           if self.mouse_states.any:
176: (20)                               self._mouse_drag_event_func(
177: (24)                                   event.pos[0],
178: (24)                                   event.pos[1],
179: (24)                                   event.rel[0],
180: (24)                                   event.rel[1],
181: (20)                               )
182: (16)                           else:
183: (20)                               self._mouse_position_event_func(
184: (24)                                   event.pos[0],
185: (24)                                   event.pos[1],
186: (24)                                   event.rel[0],
187: (24)                                   event.rel[1],
188: (20)                               )
189: (12)                       elif event.type == pygame.MOUSEBUTTONDOWN:
190: (16)                           self._handle_mods()
191: (16)                           button = self._mouse_button_map.get(event.button, None)
192: (16)                           if button is not None:
193: (20)                               self._handle_mouse_button_state_change(button, True)
194: (20)                               self._mouse_press_event_func(
195: (24)                                   event.pos[0],
196: (24)                                   event.pos[1],
197: (24)                                   button,
198: (20)                               )
199: (12)                       elif event.type == pygame.MOUSEBUTTONUP:
200: (16)                           self._handle_mods()
201: (16)                           button = self._mouse_button_map.get(event.button, None)
202: (16)                           if button is not None:
203: (20)                               self._handle_mouse_button_state_change(button, False)
204: (20)                               self._mouse_release_event_func(
205: (24)                                   event.pos[0],
206: (24)                                   event.pos[1],
207: (24)                                   button,
208: (20)                               )
209: (12)                       elif event.type in [pygame.KEYDOWN, pygame.KEYUP]:
210: (16)                           self._handle_mods()
211: (16)                           if self._exit_key is not None and event.key == self._exit_key:
212: (20)                               self.close()
213: (16)                           if (
214: (20)                               event.type == pygame.KEYUP
215: (20)                               and self._fs_key is not None
216: (20)                               and event.key == self._fs_key
217: (16)                           ):
218: (20)                               self.fullscreen = not self.fullscreen
```

```
219: (16)                           if event.type == pygame.KEYDOWN:
220: (20)                               self._key_pressed_map[event.key] = True
221: (16)                           elif event.type == pygame.KEYUP:
222: (20)                               self._key_pressed_map[event.key] = False
223: (16)                           self._key_event_func(event.key, event.type, self._modifiers)
224: (12)                       elif event.type == pygame.TEXTINPUT:
225: (16)                           self._handle_mods()
226: (16)                           self._unicode_char_entered_func(event.text)
227: (12)                       elif event.type == pygame.MOUSEWHEEL:
228: (16)                           self._handle_mods()
229: (16)                           self._mouse_scroll_event_func(float(event.x), float(event.y))
230: (12)                       elif event.type == pygame.QUIT:
231: (16)                           self.close()
232: (12)                       elif event.type == pygame.VIDEORESIZE:
233: (16)                           self.resize(event.size[0], event.size[1])
234: (12)                       elif event.type == pygame.ACTIVEEVENT:
235: (16)                           if getattr(event, "state", None) == 2:
236: (20)                               if event.gain:
237: (24)                                   self._visible = True
238: (24)                                   self._iconify_func(False)
239: (20)                               else:
240: (24)                                   self._visible = False
241: (24)                                   self._iconify_func(True)
242: (12)                       elif event.type == pygame.USEREVENT:
243: (16)                           self._on_generic_event_func(event)
244: (4)                def destroy(self) -> None:
245: (8)                    """Gracefully close the window"""
246: (8)                    pygame.quit()

----------------------------------------

File 29 - window.py:

1: (0)              import platform
2: (0)              import pyglet
3: (0)              if platform.system() == "Darwin":
4: (4)                  pyglet.options["shadow_window"] = False
5: (0)              pyglet.options["debug_gl"] = False
6: (0)              from pathlib import Path  # noqa
7: (0)              from typing import Any, Union  # noqa
8: (0)              from moderngl_window.context.base import BaseWindow  # noqa: E402
9: (0)              from moderngl_window.context.pyglet.keys import Keys  # noqa: E402
10: (0)             class Window(BaseWindow):
11: (4)                 """
12: (4)                 Window based on Pyglet 1.4.x
13: (4)                 """
14: (4)                 name = "pyglet"
15: (4)                 keys = Keys
16: (4)                 _mouse_button_map = {
17: (8)                     1: 1,
18: (8)                     4: 2,
19: (8)                     2: 3,
20: (4)                 }
21: (4)                 def __init__(self, **kwargs: Any):
22: (8)                     super().__init__(**kwargs)
23: (8)                     config = pyglet.gl.Config(
24: (12)                        major_version=self.gl_version[0],
25: (12)                        minor_version=self.gl_version[1],
26: (12)                        forward_compatible=True,
27: (12)                        red_size=8,
28: (12)                        green_size=8,
29: (12)                        blue_size=8,
30: (12)                        alpha_size=8,
31: (12)                        stencil_size=8,
32: (12)                        depth_size=24,
33: (12)                        double_buffer=True,
34: (12)                        sample_buffers=1 if self.samples > 1 else 0,
35: (12)                        samples=self.samples,
36: (8)                     )
```

```
(8)           if self.fullscreen:
(12)              display = pyglet.canvas.get_display()
(12)              screen = display.get_default_screen()
(12)              self._width, self._height = screen.width, screen.height
(8)           self._window = PygletWrapper(
(12)              width=self._width,
(12)              height=self._height,
(12)              caption=self._title,
(12)              resizable=self._resizable,
(12)              visible=self._visible,
(12)              vsync=self._vsync,
(12)              fullscreen=self._fullscreen,
(12)              config=config,
(12)              file_drops=True and platform.system() != "Darwin",
(8)           )
(8)           self.cursor = self._cursor
(8)           self._window.event(self.on_key_press)
(8)           self._window.event(self.on_key_release)
(8)           self._window.event(self.on_mouse_motion)
(8)           self._window.event(self.on_mouse_drag)
(8)           self._window.event(self.on_resize)
(8)           self._window.event(self.on_close)
(8)           self._window.event(self.on_mouse_press)
(8)           self._window.event(self.on_mouse_release)
(8)           self._window.event(self.on_mouse_scroll)
(8)           self._window.event(self.on_text)
(8)           self._window.event(self.on_show)
(8)           self._window.event(self.on_hide)
(8)           self._window.event(self.on_file_drop)
(8)           self.init_mgl_context()
(8)           self._buffer_width, self._buffer_height =
self._window.get_framebuffer_size()
(8)           self.set_default_viewport()
(4)       def _set_fullscreen(self, value: bool) -> None:
(8)           self._window.set_fullscreen(value)
(4)       @property
(4)       def size(self) -> tuple[int, int]:
(8)           """tuple[int, int]: current window size.
(8)           This property also support assignment::
(12)              window.size = 1000, 1000
(8)           """
(8)           return self._width, self._height
(4)       @size.setter
(4)       def size(self, value: tuple[int, int]) -> None:
(8)           self._window.set_size(value[0], value[1])
(4)       @property
(4)       def position(self) -> tuple[int, int]:
(8)           """tuple[int, int]: The current window position.
(8)           This property can also be set to move the window::
(12)              window.position = 100, 100
(8)           """
(8)           return self._window.get_location()
(4)       @position.setter
(4)       def position(self, value: tuple[int, int]) -> None:
(8)           self._window.set_location(value[0], value[1])
(4)       @property
(4)       def visible(self) -> bool:
(8)           """bool: Is the window visible?
(8)           This property can also be set::
(12)              window.visible = False
(8)           """
(8)           return self._visible
(4)       @visible.setter
(4)       def visible(self, value: bool) -> None:
(8)           self._visible = value
(8)           self._window.set_visible(value)
(4)       @property
(4)       def cursor(self) -> bool:
(8)           """bool: Should the mouse cursor be visible inside the window?
```

```
(8)                 This property can also be assigned to::
(12)                    window.cursor = False
(8)                 """
(8)                 return self._cursor
(4)             @cursor.setter
(4)             def cursor(self, value: bool) -> None:
(8)                 self._window.set_mouse_visible(value)
(8)                 self._cursor = value
(4)             @property
(4)             def mouse_exclusivity(self) -> bool:
(8)                 """bool: If mouse exclusivity is enabled.
(8)                 When you enable mouse-exclusive mode, the mouse cursor is no longer
(8)                 available. It is not merely hidden – no amount of mouse movement
(8)                 will make it leave your application. This is for example useful
(8)                 when you don't want the mouse leaving the screen when rotating
(8)                 a 3d scene.
(8)                 This property can also be set::
(12)                    window.mouse_exclusivity = True
(8)                 """
(8)                 return self._mouse_exclusivity
(4)             @mouse_exclusivity.setter
(4)             def mouse_exclusivity(self, value: bool) -> None:
(8)                 self._window.set_exclusive_mouse(value)
(8)                 self._mouse_exclusivity = value
(4)             @property
(4)             def title(self) -> str:
(8)                 """str: Window title.
(8)                 This property can also be set::
(12)                    window.title = "New Title"
(8)                 """
(8)                 return self._title
(4)             @title.setter
(4)             def title(self, value: str) -> None:
(8)                 self._window.set_caption(value)
(8)                 self._title = value
(4)             @property
(4)             def is_closing(self) -> bool:
(8)                 """Check pyglet's internal exit state"""
(8)                 return self._window.has_exit or super().is_closing
(4)             @is_closing.setter
(4)             def is_closing(self, value: bool) -> None:
(8)                 self._close = value
(4)             def close(self) -> None:
(8)                 """Close the pyglet window directly"""
(8)                 self.is_closing = True
(8)                 self._window.close()
(8)                 super().close()
(4)             def swap_buffers(self) -> None:
(8)                 """Swap buffers, increment frame counter and pull events"""
(8)                 self._window.flip()
(8)                 self._frames += 1
(8)                 self._window.dispatch_events()
(4)             def _handle_modifiers(self, mods: int) -> None:
(8)                 """Update key modifier states"""
(8)                 self._modifiers.shift = mods & 1 == 1
(8)                 self._modifiers.ctrl = mods & 2 == 2
(8)                 self._modifiers.alt = mods & 4 == 4
(4)             def _set_icon(self, icon_path: Path) -> None:
(8)                 icon = pyglet.image.load(icon_path)
(8)                 self._window.set_icon(icon)
(4)             def _set_vsync(self, value: bool) -> None:
(8)                 self._window.set_vsync(value)
(4)             def on_key_press(self, symbol: int, modifiers: int) -> bool:
(8)                 """Pyglet specific key press callback.
(8)                 Forwards and translates the events to the standard methods.
(8)                 Args:
(12)                    symbol: The symbol of the pressed key
(12)                    modifiers: Modifier state (shift, ctrl etc.)
(8)                 """
```

```
(8)             if self._exit_key is not None and symbol == self._exit_key:
(12)                self.close()
(8)             if self._fs_key is not None and symbol == self._fs_key:
(12)                self.fullscreen = not self.fullscreen
(8)             self._key_pressed_map[symbol] = True
(8)             self._handle_modifiers(modifiers)
(8)             self._key_event_func(symbol, self.keys.ACTION_PRESS, self._modifiers)
(8)             return pyglet.event.EVENT_HANDLED
(4)         def on_text(self, text: str) -> None:
(8)             """Pyglet specific text input callback
(8)             Forwards and translates the events to the standard methods.
(8)             Args:
(12)                text (str): The unicode character entered
(8)             """
(8)             self._unicode_char_entered_func(text)
(4)         def on_key_release(self, symbol: int, modifiers: int) -> None:
(8)             """Pyglet specific key release callback.
(8)             Forwards and translates the events to standard methods.
(8)             Args:
(12)                symbol: The symbol of the pressed key
(12)                modifiers: Modifier state (shift, ctrl etc.)
(8)             """
(8)             self._key_pressed_map[symbol] = False
(8)             self._handle_modifiers(modifiers)
(8)             self._key_event_func(symbol, self.keys.ACTION_RELEASE,
self._modifiers)
(4)         def on_mouse_motion(self, x: int, y: int, dx: int, dy: int) -> None:
(8)             """Pyglet specific mouse motion callback.
(8)             Forwards and translates the event to the standard methods.
(8)             Args:
(12)                x: x position of the mouse
(12)                y: y position of the mouse
(12)                dx: delta x position
(12)                dy: delta y position of the mouse
(8)             """
(8)             self._mouse_position_event_func(x, self._height - y, dx, -dy)
(4)         def on_mouse_drag(self, x: int, y: int, dx: int, dy: int, buttons: int,
modifiers: int) -> None:
(8)             """Pyglet specific mouse drag event.
(8)             When a mouse button is pressed this is the only way
(8)             to capture mouse position events
(8)             """
(8)             self._handle_modifiers(modifiers)
(8)             self._mouse_drag_event_func(x, self._height - y, dx, -dy)
(4)         def on_mouse_press(self, x: int, y: int, button: int, mods: int) -> None:
(8)             """Handle mouse press events and forward to standard methods
(8)             Args:
(12)                x: x position of the mouse when pressed
(12)                y: y position of the mouse when pressed
(12)                button: The pressed button
(12)                mods: Modifiers
(8)             """
(8)             self._handle_modifiers(mods)
(8)             button = self._mouse_button_map.get(button, -1)
(8)             if button != -1:
(12)                self._handle_mouse_button_state_change(button, True)
(12)                self._mouse_press_event_func(
(16)                    x,
(16)                    self._height - y,
(16)                    button,
(12)                )
(4)         def on_mouse_release(self, x: int, y: int, button: int, mods: int) ->
None:
(8)             """Handle mouse release events and forward to standard methods
(8)             Args:
(12)                x: x position when mouse button was released
(12)                y: y position when mouse button was released
(12)                button: The button pressed
(12)                mods: Modifiers
```

```
(8)                 """
(8)                 button = self._mouse_button_map.get(button, -1)
(8)                 if button != -1:
(12)                    self._handle_mouse_button_state_change(button, False)
(12)                    self._mouse_release_event_func(
(16)                        x,
(16)                        self._height - y,
(16)                        button,
(12)                    )
(4)             def on_mouse_scroll(self, x: int, y: int, x_offset: float, y_offset: float) -> None:
(8)                 """Handle mouse wheel.
(8)                 Args:
(12)                    x_offset (float): X scroll offset
(12)                    y_offset (float): Y scroll offset
(8)                 """
(8)                 self._handle_modifiers(0)  # No modifiers available
(8)                 self.mouse_scroll_event_func(x_offset, y_offset)
(4)             def on_resize(self, width: int, height: int) -> None:
(8)                 """Pyglet specific callback for window resize events forwarding to standard methods
(8)                 Args:
(12)                    width: New window width
(12)                    height: New window height
(8)                 """
(8)                 self._width, self._height = width, height
(8)                 self._buffer_width, self._buffer_height = self._window.get_framebuffer_size()
(8)                 self.set_default_viewport()
(8)                 super().resize(self._buffer_width, self._buffer_height)
(4)             def on_close(self) -> None:
(8)                 """Pyglet specific window close callback"""
(8)                 self._close_func()
(4)             def on_show(self) -> None:
(8)                 """Called when window first appear or restored from hidden state"""
(8)                 self._visible = True
(8)                 self._iconify_func(False)
(4)             def on_hide(self) -> None:
(8)                 """Called when window is minimized"""
(8)                 self._visible = False
(8)                 self._iconify_func(True)
(4)             def on_file_drop(self, x: int, y: int, paths: list[Union[str, Path]]) -> None:
(8)                 """Called when files dropped onto the window
(8)                 Args:
(12)                    x (int): X location in window where file was dropped
(12)                    y (int): Y location in window where file was dropped
(12)                    paths (list): List of file paths dropped
(8)                 """
(8)                 (x, y) = self.convert_window_coordinates(x, y, y_flipped=True)
(8)                 self._files_dropped_event_func(x, y, paths)
(4)             def destroy(self) -> None:
(8)                 """Destroy the pyglet window"""
(8)                 pass
(0)         class PygletWrapper(pyglet.window.Window):
(4)             """Block out some window methods so pyglet don't trigger GL errors"""
(4)             def on_resize(self, width: int, height: int) -> None:
(8)                 """Block out the resize method.
(8)                 For some reason pyglet calls this triggering errors.
(8)                 """
(8)                 pass
(4)             def on_draw(self) -> None:
(8)                 """Block out the default draw method to avoid GL errors"""
(8)                 pass
```

File 30 - window.py:

```
(0)             from pathlib import Path
(0)             from typing import Any
(0)             from PyQt5 import QtCore, QtGui, QtOpenGL, QtWidgets
(0)             from moderngl_window.context.base import BaseWindow
(0)             from moderngl_window.context.pyqt5.keys import Keys
(0)             class Window(BaseWindow):
(4)                 """
(4)                 A basic window implementation using PyQt5 with the goal of
(4)                 creating an OpenGL context and handle keyboard and mouse input.
(4)                 This window bypasses Qt's own event loop to make things as flexible as
possible.
(4)                 If you need to use the event loop and are using other features
(4)                 in Qt as well, this example can still be useful as a reference
(4)                 when creating your own window.
(4)                 """
(4)                 name = "pyqt5"
(4)                 keys = Keys
(4)                 _mouse_button_map = {
(8)                     1: 1,
(8)                     2: 2,
(8)                     4: 3,
(4)                 }
(4)                 def __init__(self, **kwargs: Any):
(8)                     super().__init__(**kwargs)
(8)                     gl = QtOpenGL.QGLFormat()
(8)                     gl.setVersion(self.gl_version[0], self.gl_version[1])
(8)                     gl.setProfile(QtOpenGL.QGLFormat.CoreProfile)
(8)                     gl.setDepthBufferSize(24)
(8)                     gl.setStencilBufferSize(8)
(8)                     gl.setDoubleBuffer(True)
(8)                     gl.setSwapInterval(1 if self._vsync else 0)
(8)                     if self.samples > 1:
(12)                        gl.setSampleBuffers(True)
(12)                        gl.setSamples(int(self.samples))
(8)                     self._app = QtWidgets.QApplication([])
(8)                     self._widget = QtOpenGL.QGLWidget(gl)
(8)                     self.title = self._title
(8)                     if self.fullscreen:
(12)                        rect = QtWidgets.QDesktopWidget().screenGeometry()
(12)                        self._width = rect.width()
(12)                        self._height = rect.height()
(12)                        self._buffer_width = rect.width() *
self._widget.devicePixelRatio()
(12)                        self._buffer_height = rect.height() *
self._widget.devicePixelRatio()
(8)                     if self.resizable:
(12)                        size_policy = QtWidgets.QSizePolicy(
(16)                            QtWidgets.QSizePolicy.Expanding,
(16)                            QtWidgets.QSizePolicy.Expanding,
(12)                        )
(12)                        self._widget.setSizePolicy(size_policy)
(12)                        self._widget.resize(self.width, self.height)
(8)                     else:
(12)                        self._widget.setFixedSize(self.width, self.height)
(8)                     if not self.visible:
(12)                        self._widget.hide()
(8)                     if not self.fullscreen:
(12)                        center_window_position = (
(16)                            int(self.position[0] - self.width / 2),
(16)                            int(self.position[1] - self.height / 2),
(12)                        )
(12)                        self._widget.move(*center_window_position)
(8)                     self._widget.resizeGL = self.resize
(8)                     self.cursor = self._cursor
(8)                     if self.fullscreen:
(12)                        self._widget.showFullScreen()
(8)                     else:
(12)                        self._widget.show()
(8)                     self._widget.setMouseTracking(True)
```

```
(8)             self._widget.keyPressEvent = self.key_pressed_event
(8)             self._widget.keyReleaseEvent = self.key_release_event
(8)             self._widget.mouseMoveEvent = self.mouse_move_event
(8)             self._widget.mousePressEvent = self.mouse_press_event
(8)             self._widget.mouseReleaseEvent = self.mouse_release_event
(8)             self._widget.wheelEvent = self.mouse_wheel_event
(8)             self._widget.closeEvent = self.close_event
(8)             self._widget.showEvent = self.show_event
(8)             self._widget.hideEvent = self.hide_event
(8)             self.init_mgl_context()
(8)             self._buffer_width = self._width * self._widget.devicePixelRatio()
(8)             self._buffer_height = self._height * self._widget.devicePixelRatio()
(8)             self.set_default_viewport()
(4)         def _set_fullscreen(self, value: bool) -> None:
(8)             if value:
(12)                self._widget.showFullScreen()
(8)             else:
(12)                self._widget.showNormal()
(4)         def _set_vsync(self, value: bool) -> None:
(8)             pass
(4)         @property
(4)         def size(self) -> tuple[int, int]:
(8)             """tuple[int, int]: current window size.
(8)             This property also support assignment::
(12)                window.size = 1000, 1000
(8)             """
(8)             return self._width, self._height
(4)         @size.setter
(4)         def size(self, value: tuple[int, int]) -> None:
(8)             pos = self.position
(8)             self._widget.setGeometry(pos[0], pos[1], value[0], value[1])
(4)         @property
(4)         def position(self) -> tuple[int, int]:
(8)             """tuple[int, int]: The current window position.
(8)             This property can also be set to move the window::
(12)                window.position = 100, 100
(8)             """
(8)             geo = self._widget.geometry()
(8)             return geo.x(), geo.y()
(4)         @position.setter
(4)         def position(self, value: tuple[int, int]) -> None:
(8)             self._widget.setGeometry(value[0], value[1], self._width, self._height)
(4)         @property
(4)         def visible(self) -> bool:
(8)             """bool: Is the window visible?
(8)             This property can also be set::
(12)                window.visible = False
(8)             """
(8)             return self._visible
(4)         @visible.setter
(4)         def visible(self, value: bool) -> None:
(8)             self._visible = value
(8)             if value:
(12)                self._widget.show()
(8)             else:
(12)                self._widget.hide()
(4)         def swap_buffers(self) -> None:
(8)             """Swap buffers, set viewport, trigger events and increment frame counter"""
(8)             self._widget.swapBuffers()
(8)             self.set_default_viewport()
(8)             self._app.processEvents()
(8)             self._frames += 1
(4)         @property
(4)         def cursor(self) -> bool:
(8)             """bool: Should the mouse cursor be visible inside the window?
(8)             This property can also be assigned to::
(12)                window.cursor = False
```

```
(8)             """
(8)             return self._cursor
(4)         @cursor.setter
(4)         def cursor(self, value: bool) -> None:
(8)             if value is True:
(12)                self._widget.setCursor(QtCore.Qt.ArrowCursor)
(8)             else:
(12)                self._widget.setCursor(QtCore.Qt.BlankCursor)
(8)             self._cursor = value
(4)         @property
(4)         def title(self) -> str:
(8)             """str: Window title.
(8)             This property can also be set::
(12)                window.title = "New Title"
(8)             """
(8)             return self._title
(4)         @title.setter
(4)         def title(self, value: str) -> None:
(8)             self._widget.setWindowTitle(value)
(8)             self._title = value
(4)         def resize(self, width: int, height: int) -> None:
(8)             """Replacement for Qt's ``resizeGL`` method.
(8)             Args:
(12)                width: New window width
(12)                height: New window height
(8)             """
(8)             self._width = width // self._widget.devicePixelRatio()
(8)             self._height = height // self._widget.devicePixelRatio()
(8)             self._buffer_width = width
(8)             self._buffer_height = height
(8)             if self._ctx:
(12)                self.set_default_viewport()
(8)             super().resize(self._buffer_width, self._buffer_height)
(4)         def _handle_modifiers(self, mods: int) -> None:
(8)             """Update modifiers"""
(8)             self._modifiers.shift = bool(mods & QtCore.Qt.ShiftModifier)
(8)             self._modifiers.ctrl = bool(mods & QtCore.Qt.ControlModifier)
(8)             self._modifiers.alt = bool(mods & QtCore.Qt.AltModifier)
(4)         def _set_icon(self, icon_path: Path) -> None:
(8)             self._widget.setWindowIcon(QtGui.QIcon(icon_path))
(4)         def key_pressed_event(self, event: QtCore.QEvent) -> None:
(8)             """Process Qt key press events forwarding them to standard methods
(8)             Args:
(12)                event: The qtevent instance
(8)             """
(8)             if self._exit_key is not None and event.key() == self._exit_key:
(12)                self.close()
(8)             if self._fs_key is not None and event.key() == self._fs_key:
(12)                self.fullscreen = not self.fullscreen
(8)             self._handle_modifiers(event.modifiers())
(8)             self._key_pressed_map[event.key()] = True
(8)             self._key_event_func(event.key(), self.keys.ACTION_PRESS,
self._modifiers)
(8)             text = event.text()
(8)             if text.strip() or event.key() == self.keys.SPACE:
(12)                self._unicode_char_entered_func(text)
(4)         def key_release_event(self, event: QtCore.QEvent) -> None:
(8)             """Process Qt key release events forwarding them to standard methods
(8)             Args:
(12)                event: The qtevent instance
(8)             """
(8)             self._handle_modifiers(event.modifiers())
(8)             self._key_pressed_map[event.key()] = False
(8)             self._key_event_func(event.key(), self.keys.ACTION_RELEASE,
self._modifiers)
(4)         def mouse_move_event(self, event: QtCore.QEvent) -> None:
(8)             """Forward mouse cursor position events to standard methods
(8)             Args:
(12)                event: The qtevent instance
```

```
(8)                 """
(8)                 x, y = event.x(), event.y()
(8)                 dx, dy = self._calc_mouse_delta(x, y)
(8)                 if self.mouse_states.any:
(12)                    self._mouse_drag_event_func(x, y, dx, dy)
(8)                 else:
(12)                    self._mouse_position_event_func(x, y, dx, dy)
(4)             def mouse_press_event(self, event: QtCore.QEvent) -> None:
(8)                 """Forward mouse press events to standard methods
(8)                 Args:
(12)                    event: The qtevent instance
(8)                 """
(8)                 self._handle_modifiers(event.modifiers())
(8)                 button = self._mouse_button_map.get(event.button())
(8)                 if button is None:
(12)                    return
(8)                 self._handle_mouse_button_state_change(button, True)
(8)                 self._mouse_press_event_func(event.x(), event.y(), button)
(4)             def mouse_release_event(self, event: QtCore.QEvent) -> None:
(8)                 """Forward mouse release events to standard methods
(8)                 Args:
(12)                    event: The qtevent instance
(8)                 """
(8)                 self._handle_modifiers(event.modifiers())
(8)                 button = self._mouse_button_map.get(event.button())
(8)                 if button is None:
(12)                    return
(8)                 self._handle_mouse_button_state_change(button, False)
(8)                 self._mouse_release_event_func(event.x(), event.y(), button)
(4)             def mouse_wheel_event(self, event: QtCore.QEvent) -> None:
(8)                 """Forward mouse wheel events to standard metods.
(8)                 From Qt docs:
(8)                 Returns the distance that the wheel is rotated, in eighths of a degree.
(8)                 A positive value indicates that the wheel was rotated forwards away from the user;
(8)                 a negative value indicates that the wheel was rotated backwards toward the user.
(8)                 Most mouse types work in steps of 15 degrees, in which case the delta value is a
(8)                 multiple of 120; i.e., 120 units * 1/8 = 15 degrees.
(8)                 However, some mice have finer-resolution wheels and send delta values that are less
(8)                 than 120 units (less than 15 degrees). To support this possibility, you can either
(8)                 cumulatively add the delta values from events until the value of 120 is reached,
(8)                 then scroll the widget, or you can partially scroll the widget in response to each
(8)                 wheel event.
(8)                 Args:
(12)                    event (QWheelEvent): Mouse wheel event
(8)                 """
(8)                 self._handle_modifiers(event.modifiers())
(8)                 point = event.angleDelta()
(8)                 self._mouse_scroll_event_func(point.x() / 120.0, point.y() / 120.0)
(4)             def close_event(self, event: QtCore.QEvent) -> None:
(8)                 """The standard PyQt close events
(8)                 Args:
(12)                    event: The qtevent instance
(8)                 """
(8)                 self.close()
(4)             def close(self) -> None:
(8)                 """Close the window"""
(8)                 super().close()
(8)                 self._close_func()
(4)             def show_event(self, event: QtCore.QEvent) -> None:
(8)                 """The standard Qt show event"""
(8)                 self._visible = True
```

```
262: (8)                        self._iconify_func(False)
263: (4)                    def hide_event(self, event: QtCore.QEvent) -> None:
264: (8)                        """The standard Qt hide event"""
265: (8)                        self._visible = False
266: (8)                        self._iconify_func(True)
267: (4)                    def destroy(self) -> None:
268: (8)                        """Quit the Qt application to exit the window gracefully"""
269: (8)                        QtCore.QCoreApplication.instance().quit()

----------------------------------------

File 31 - window.py:

1: (0)                  from pathlib import Path
2: (0)                  from typing import Any
3: (0)                  from PySide2 import QtCore, QtGui, QtOpenGL, QtWidgets
4: (0)                  from moderngl_window.context.base import BaseWindow
5: (0)                  from moderngl_window.context.pyside2.keys import Keys
6: (0)                  class Window(BaseWindow):
7: (4)                      """
8: (4)                      A basic window implementation using PySide2 with the goal of
9: (4)                      creating an OpenGL context and handle keyboard and mouse input.
10: (4)                     This window bypasses Qt's own event loop to make things as flexible as
possible.
11: (4)                     If you need to use the event loop and are using other features
12: (4)                     in Qt as well, this example can still be useful as a reference
13: (4)                     when creating your own window.
14: (4)                     """
15: (4)                     name = "pyside2"
16: (4)                     keys = Keys
17: (4)                     _mouse_button_map = {
18: (8)                         1: 1,
19: (8)                         2: 2,
20: (8)                         4: 3,
21: (4)                     }
22: (4)                     def __init__(self, **kwargs: Any):
23: (8)                         super().__init__(**kwargs)
24: (8)                         gl = QtOpenGL.QGLFormat()
25: (8)                         gl.setVersion(self.gl_version[0], self.gl_version[1])
26: (8)                         gl.setProfile(QtOpenGL.QGLFormat.CoreProfile)
27: (8)                         gl.setDepthBufferSize(24)
28: (8)                         gl.setStencilBufferSize(8)
29: (8)                         gl.setDoubleBuffer(True)
30: (8)                         gl.setSwapInterval(1 if self.vsync else 0)
31: (8)                         if self.samples > 1:
32: (12)                            gl.setSampleBuffers(True)
33: (12)                            gl.setSamples(self.samples)
34: (8)                         self._app = QtWidgets.QApplication([])
35: (8)                         self._widget = QtOpenGL.QGLWidget(gl)
36: (8)                         self.title = self._title
37: (8)                         if self.fullscreen:
38: (12)                            rect = QtWidgets.QDesktopWidget().screenGeometry()
39: (12)                            self._width = rect.width()
40: (12)                            self._height = rect.height()
41: (12)                            self._buffer_width = rect.width() *
self._widget.devicePixelRatio()
42: (12)                            self._buffer_height = rect.height() *
self._widget.devicePixelRatio()
43: (8)                         if self.resizable:
44: (12)                            size_policy = QtWidgets.QSizePolicy(
45: (16)                                QtWidgets.QSizePolicy.Expanding,
46: (16)                                QtWidgets.QSizePolicy.Expanding,
47: (12)                            )
48: (12)                            self._widget.setSizePolicy(size_policy)
49: (12)                            self._widget.resize(self.width, self.height)
50: (8)                         else:
51: (12)                            self._widget.setFixedSize(self.width, self.height)
52: (8)                         if not self.visible:
53: (12)                            self._widget.hide()
```

```
(8)          if not self.fullscreen:
(12)             center_window_position = (
(16)                 self.position[0] - self.width / 2,
(16)                 self.position[1] - self.height / 2,
(12)             )
(12)             self._widget.move(*center_window_position)
(8)          self._widget.resizeGL = self.resize
(8)          self.cursor = self._cursor
(8)          if self.fullscreen:
(12)             self._widget.showFullScreen()
(8)          else:
(12)             self._widget.show()
(8)          self._widget.setMouseTracking(True)
(8)          self._widget.keyPressEvent = self.key_pressed_event
(8)          self._widget.keyReleaseEvent = self.key_release_event
(8)          self._widget.mouseMoveEvent = self.mouse_move_event
(8)          self._widget.mousePressEvent = self.mouse_press_event
(8)          self._widget.mouseReleaseEvent = self.mouse_release_event
(8)          self._widget.wheelEvent = self.mouse_wheel_event
(8)          self._widget.closeEvent = self.close_event
(8)          self._widget.showEvent = self.show_event
(8)          self._widget.hideEvent = self.hide_event
(8)          self.init_mgl_context()
(8)          self._buffer_width = self._width * self._widget.devicePixelRatio()
(8)          self._buffer_height = self._height * self._widget.devicePixelRatio()
(8)          self.set_default_viewport()
(4)      def _set_fullscreen(self, value: bool) -> None:
(8)          if value:
(12)             self._widget.showFullScreen()
(8)          else:
(12)             self._widget.showNormal()
(4)      def _set_vsync(self, value: bool) -> None:
(8)          pass
(4)      @property
(4)      def size(self) -> tuple[int, int]:
(8)          """tuple[int, int]: current window size.
(8)          This property also support assignment::
(12)             window.size = 1000, 1000
(8)          """
(8)          return self._width, self._height
(4)      @size.setter
(4)      def size(self, value: tuple[int, int]) -> None:
(8)          pos = self.position
(8)          self._widget.setGeometry(pos[0], pos[1], value[0], value[1])
(4)      @property
(4)      def visible(self) -> bool:
(8)          """bool: Is the window visible?
(8)          This property can also be set::
(12)             window.visible = False
(8)          """
(8)          return self._visible
(4)      @visible.setter
(4)      def visible(self, value: bool) -> None:
(8)          self._visible = value
(8)          if value:
(12)             self._widget.show()
(8)          else:
(12)             self._widget.hide()
(4)      @property
(4)      def position(self) -> tuple[int, int]:
(8)          """tuple[int, int]: The current window position.
(8)          This property can also be set to move the window::
(12)             window.position = 100, 100
(8)          """
(8)          geo = self._widget.geometry()
(8)          return geo.x(), geo.y()
(4)      @position.setter
(4)      def position(self, value: tuple[int, int]) -> None:
(8)          self._widget.setGeometry(value[0], value[1], self._width,
self._height)
123: (4)                @property
124: (4)                def cursor(self) -> bool:
125: (8)                    """bool: Should the mouse cursor be visible inside the window?
126: (8)                    This property can also be assigned to::
127: (12)                       window.cursor = False
128: (8)                    """
129: (8)                    return self._cursor
130: (4)                @cursor.setter
131: (4)                def cursor(self, value: bool) -> None:
132: (8)                    if value is True:
133: (12)                       self._widget.setCursor(QtCore.Qt.ArrowCursor)
134: (8)                    else:
135: (12)                       self._widget.setCursor(QtCore.Qt.BlankCursor)
136: (8)                    self._cursor = value
137: (4)                @property
138: (4)                def title(self) -> str:
139: (8)                    """str: Window title.
140: (8)                    This property can also be set::
141: (12)                       window.title = "New Title"
142: (8)                    """
143: (8)                    return self._title
144: (4)                @title.setter
145: (4)                def title(self, value: str) -> None:
146: (8)                    self._widget.setWindowTitle(value)
147: (8)                    self._title = value
148: (4)                def swap_buffers(self) -> None:
149: (8)                    """Swap buffers, set viewport, trigger events and increment frame counter"""
150: (8)                    self._widget.swapBuffers()
151: (8)                    self.set_default_viewport()
152: (8)                    self._app.processEvents()
153: (8)                    self._frames += 1
154: (4)                def resize(self, width: int, height: int) -> None:
155: (8)                    """Replacement for Qt's ``resizeGL`` method.
156: (8)                    Args:
157: (12)                       width: New window width
158: (12)                       height: New window height
159: (8)                    """
160: (8)                    self._width = width // self._widget.devicePixelRatio()
161: (8)                    self._height = height // self._widget.devicePixelRatio()
162: (8)                    self._buffer_width = width
163: (8)                    self._buffer_height = height
164: (8)                    if self._ctx:
165: (12)                       self.set_default_viewport()
166: (8)                    super().resize(self._buffer_width, self._buffer_height)
167: (4)                def _handle_modifiers(self, mods: QtCore.Qt.KeyboardModifier) -> None:
168: (8)                    """Update modifiers"""
169: (8)                    self._modifiers.shift = bool(mods & QtCore.Qt.ShiftModifier)
170: (8)                    self._modifiers.ctrl = bool(mods & QtCore.Qt.ControlModifier)
171: (8)                    self._modifiers.alt = bool(mods & QtCore.Qt.AltModifier)
172: (4)                def _set_icon(self, icon_path: Path) -> None:
173: (8)                    self._widget.setWindowIcon(QtGui.QIcon(icon_path))
174: (4)                def key_pressed_event(self, event: QtCore.QEvent) -> None:
175: (8)                    """Process Qt key press events forwarding them to standard methods
176: (8)                    Args:
177: (12)                       event: The qtevent instance
178: (8)                    """
179: (8)                    if self._exit_key is not None and event.key() == self._exit_key:
180: (12)                       self.close()
181: (8)                    if self._fs_key is not None and event.key() == self._fs_key:
182: (12)                       self.fullscreen = not self.fullscreen
183: (8)                    self._handle_modifiers(event.modifiers())
184: (8)                    self._key_pressed_map[event.key()] = True
185: (8)                    self.key_event_func(event.key(), self.keys.ACTION_PRESS, self._modifiers)
186: (8)                    text = event.text()
187: (8)                    if text.strip() or event.key() == self.keys.SPACE:
188: (12)                       self._unicode_char_entered_func(text)
```

```
    def key_release_event(self, event: QtCore.QEvent) -> None:
        """Process Qt key release events forwarding them to standard methods
        Args:
            event: The qtevent instance
        """
        self._handle_modifiers(event.modifiers())
        self._key_pressed_map[event.key()] = False
        self.key_event_func(event.key(), self.keys.ACTION_RELEASE, self._modifiers)
    def mouse_move_event(self, event: QtCore.QEvent) -> None:
        """Forward mouse cursor position events to standard methods
        Args:
            event: The qtevent instance
        """
        x, y = event.x(), event.y()
        dx, dy = self._calc_mouse_delta(x, y)
        if self.mouse_states.any:
            self._mouse_drag_event_func(x, y, dx, dy)
        else:
            self._mouse_position_event_func(x, y, dx, dy)
    def mouse_press_event(self, event: QtCore.QEvent) -> None:
        """Forward mouse press events to standard methods
        Args:
            event: The qtevent instance
        """
        self._handle_modifiers(event.modifiers())
        button = self._mouse_button_map.get(event.button())
        if button is None:
            return
        self._handle_mouse_button_state_change(button, True)
        self.mouse_press_event_func(event.x(), event.y(), button)
    def mouse_release_event(self, event: QtCore.QEvent) -> None:
        """Forward mouse release events to standard methods
        Args:
            event: The qtevent instance
        """
        self._handle_modifiers(event.modifiers())
        button = self._mouse_button_map.get(event.button())
        if button is None:
            return
        self._handle_mouse_button_state_change(button, False)
        self.mouse_release_event_func(event.x(), event.y(), button)
    def mouse_wheel_event(self, event: QtCore.QEvent) -> None:
        """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.
        Args:
            event (QWheelEvent): Mouse wheel event
        """
        self._handle_modifiers(event.modifiers())
        point = event.angleDelta()
        self._mouse_scroll_event_func(point.x() / 120.0, point.y() / 120.0)
```

```
249: (4)                def close_event(self, event: QtCore.QEvent) -> None:
250: (8)                    """The standard PyQt close events
251: (8)                    Args:
252: (12)                       event: The qtevent instance
253: (8)                    """
254: (8)                    self.close()
255: (4)                def close(self) -> None:
256: (8)                    """Close the window"""
257: (8)                    super().close()
258: (8)                    self._close_func()
259: (4)                def show_event(self, event: QtCore.QEvent) -> None:
260: (8)                    """The standard Qt show event"""
261: (8)                    self._visible = True
262: (8)                    self._iconify_func(False)
263: (4)                def hide_event(self, event: QtCore.QEvent) -> None:
264: (8)                    """The standard Qt hide event"""
265: (8)                    self._visible = False
266: (8)                    self._iconify_func(True)
267: (4)                def destroy(self) -> None:
268: (8)                    """Quit the Qt application to exit the window gracefully"""
269: (8)                    QtCore.QCoreApplication.instance().quit()

----------------------------------------

File 32 - window.py:

1: (0)              from ctypes import c_char_p, c_int
2: (0)              from pathlib import Path
3: (0)              from typing import Any
4: (0)              import sdl2
5: (0)              import sdl2.ext
6: (0)              import sdl2.video
7: (0)              from moderngl_window.context.base import BaseWindow
8: (0)              from moderngl_window.context.sdl2.keys import Keys
9: (0)              class Window(BaseWindow):
10: (4)                 """
11: (4)                 Basic window implementation using SDL2.
12: (4)                 """
13: (4)                 name = "sdl2"
14: (4)                 keys = Keys
15: (4)                 _mouse_button_map = {
16: (8)                     1: 1,
17: (8)                     3: 2,
18: (8)                     2: 3,
19: (4)                 }
20: (4)                 def __init__(self, **kwargs: Any):
21: (8)                     super().__init__(**kwargs)
22: (8)                     if sdl2.SDL_Init(sdl2.SDL_INIT_VIDEO) != 0:
23: (12)                        raise ValueError("Failed to initialize sdl2")
24: (8)                     sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_CONTEXT_MAJOR_VERSION,
self.gl_version[0])
25: (8)                     sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_CONTEXT_MINOR_VERSION,
self.gl_version[1])
26: (8)                     sdl2.video.SDL_GL_SetAttribute(
27: (12)                        sdl2.SDL_GL_CONTEXT_PROFILE_MASK, sdl2.SDL_GL_CONTEXT_PROFILE_CORE
28: (8)                     )
29: (8)
sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_CONTEXT_FORWARD_COMPATIBLE_FLAG, 1)
30: (8)                     sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_DOUBLEBUFFER, 1)
31: (8)                     sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_DEPTH_SIZE, 24)
32: (8)                     sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_STENCIL_SIZE, 8)
33: (8)                     self.cursor = self._cursor
34: (8)                     if self.samples > 1:
35: (12)                        sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_MULTISAMPLEBUFFERS, 1)
36: (12)                        sdl2.video.SDL_GL_SetAttribute(sdl2.SDL_GL_MULTISAMPLESAMPLES,
self.samples)
37: (8)                     flags = sdl2.SDL_WINDOW_OPENGL | sdl2.SDL_WINDOW_ALLOW_HIGHDPI
38: (8)                     if self.fullscreen:
39: (12)                        flags |= sdl2.SDL_WINDOW_FULLSCREEN_DESKTOP
```

```
        else:
            if self.resizable:
                flags |= sdl2.SDL_WINDOW_RESIZABLE
        if not self._visible:
            flags |= sdl2.SDL_WINDOW_HIDDEN
        self._window = sdl2.SDL_CreateWindow(
            self.title.encode(),
            sdl2.SDL_WINDOWPOS_UNDEFINED,
            sdl2.SDL_WINDOWPOS_UNDEFINED,
            self.width,
            self.height,
            flags,
        )
        if not self._window:
            raise ValueError("Failed to create window:", sdl2.SDL_GetError())
        self._context = sdl2.SDL_GL_CreateContext(self._window)
        sdl2.video.SDL_GL_SetSwapInterval(1 if self.vsync else 0)
        self._buffer_width, self._buffer_height = self._get_drawable_size()
        self.init_mgl_context()
        self.set_default_viewport()
    def _set_fullscreen(self, value: bool) -> None:
        sdl2.SDL_SetWindowFullscreen(
            self._window, sdl2.SDL_WINDOW_FULLSCREEN_DESKTOP if value else 0
        )
    def _set_vsync(self, value: bool) -> None:
        sdl2.video.SDL_GL_SetSwapInterval(1 if value else 0)
    def _get_drawable_size(self) -> tuple[int, int]:
        x = c_int()
        y = c_int()
        sdl2.video.SDL_GL_GetDrawableSize(self._window, x, y)
        return x.value, y.value
    @property
    def size(self) -> tuple[int, int]:
        """tuple[int, int]: current window size.
        This property also support assignment::
            window.size = 1000, 1000
        """
        return self._width, self._height
    @size.setter
    def size(self, value: tuple[int, int]) -> None:
        sdl2.SDL_SetWindowSize(self._window, value[0], value[1])
        self.resize(value[0], value[1])
    @property
    def position(self) -> tuple[int, int]:
        """tuple[int, int]: The current window position.
        This property can also be set to move the window::
            window.position = 100, 100
        """
        x = c_int(0)
        y = c_int(0)
        sdl2.SDL_GetWindowPosition(self._window, x, y)
        return x.value, y.value
    @position.setter
    def position(self, value: tuple[int, int]) -> None:
        sdl2.SDL_SetWindowPosition(self._window, value[0], value[1])
    @property
    def visible(self) -> bool:
        """bool: Is the window visible?
        This property can also be set::
            window.visible = False
        """
        return self._visible
    @visible.setter
    def visible(self, value: bool) -> None:
        self._visible = value
        if value:
            sdl2.SDL_ShowWindow(self._window)
        else:
            sdl2.SDL_HideWindow(self._window)
    @property
    def cursor(self) -> bool:
        """bool: Should the mouse cursor be visible inside the window?
        This property can also be assigned to::
            window.cursor = False
        """
        return self._cursor
    @cursor.setter
    def cursor(self, value: bool) -> None:
        sdl2.SDL_ShowCursor(sdl2.SDL_ENABLE if value else sdl2.SDL_DISABLE)
        self._cursor = value
    @property
    def mouse_exclusivity(self) -> bool:
        """bool: 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
        """
        return self._mouse_exclusivity
    @mouse_exclusivity.setter
    def mouse_exclusivity(self, value: bool) -> None:
        if value is True:
            sdl2.SDL_SetRelativeMouseMode(sdl2.SDL_TRUE)
        else:
            sdl2.SDL_SetRelativeMouseMode(sdl2.SDL_FALSE)
        self._mouse_exclusivity = value
    @property
    def title(self) -> str:
        """str: Window title.
        This property can also be set::
            window.title = "New Title"
        """
        return self._title
    @title.setter
    def title(self, value: str) -> None:
        data = c_char_p(value.encode())
        sdl2.SDL_SetWindowTitle(self._window, data)
        self._title = value
    def swap_buffers(self) -> None:
        """Swap buffers, set viewport, trigger events and increment frame counter"""
        sdl2.SDL_GL_SwapWindow(self._window)
        self.set_default_viewport()
        self.process_events()
        self._frames += 1
    def resize(self, width: int, height: int) -> None:
        """Resize callback.
        Args:
            width: New window width
            height: New window height
        """
        self._width = width
        self._height = height
        self._buffer_width, self._buffer_height = self._get_drawable_size()
        self.set_default_viewport()
        super().resize(self._buffer_width, self._buffer_height)
    def _handle_mods(self) -> None:
        """Update key mods"""
        mods = sdl2.SDL_GetModState()
        self._modifiers.shift = mods & sdl2.KMOD_SHIFT
        self._modifiers.ctrl = mods & sdl2.KMOD_CTRL
        self._modifiers.alt = mods & sdl2.KMOD_ALT
    def _set_icon(self, icon_path: Path) -> None:
        sdl2.SDL_SetWindowIcon(self._window, sdl2.ext.load_image(icon_path))
    def process_events(self) -> None:
        """Handle all queued events in sdl2 dispatching events to standard methods"""
        for event in sdl2.ext.get_events():
            if event.type == sdl2.SDL_MOUSEMOTION:
                if self.mouse_states.any:
                    self._mouse_drag_event_func(
                        event.motion.x,
                        event.motion.y,
                        event.motion.xrel,
                        event.motion.yrel,
                    )
                else:
                    self._mouse_position_event_func(
                        event.motion.x,
                        event.motion.y,
                        event.motion.xrel,
                        event.motion.yrel,
                    )
            elif event.type == sdl2.SDL_MOUSEBUTTONDOWN:
                self._handle_mods()
                button = self._mouse_button_map.get(event.button.button, None)
                if button is not None:
                    self._handle_mouse_button_state_change(button, True)
                    self._mouse_press_event_func(
                        event.motion.x,
                        event.motion.y,
                        button,
                    )
            elif event.type == sdl2.SDL_MOUSEBUTTONUP:
                self._handle_mods()
                button = self._mouse_button_map.get(event.button.button, None)
                if button is not None:
                    self._handle_mouse_button_state_change(button, False)
                    self._mouse_release_event_func(
                        event.motion.x,
                        event.motion.y,
                        button,
                    )
            elif event.type in [sdl2.SDL_KEYDOWN, sdl2.SDL_KEYUP]:
                self._handle_mods()
                if self._exit_key is not None and event.key.keysym.sym == self._exit_key:
                    self.close()
                if (
                    self._fs_key is not None
                    and event.key.keysym.sym == self._fs_key
                    and event.type == sdl2.SDL_KEYDOWN
                ):
                    self.fullscreen = not self.fullscreen
                if event.type == sdl2.SDL_KEYDOWN:
                    self._key_pressed_map[event.key.keysym.sym] = True
                elif event.type == sdl2.SDL_KEYUP:
                    self._key_pressed_map[event.key.keysym.sym] = False
                self._key_event_func(event.key.keysym.sym, event.type, self._modifiers)
            elif event.type == sdl2.SDL_TEXTINPUT:
                self._unicode_char_entered_func(event.text.text.decode())
            elif event.type == sdl2.SDL_MOUSEWHEEL:
                self._handle_mods()
                self._mouse_scroll_event_func(float(event.wheel.x), float(event.wheel.y))
            elif event.type == sdl2.SDL_QUIT:
                self.close()
            elif event.type == sdl2.SDL_WINDOWEVENT:
                if event.window.event in [
                    sdl2.SDL_WINDOWEVENT_RESIZED,
                    sdl2.SDL_WINDOWEVENT_SIZE_CHANGED,
                ]:
                    self.resize(event.window.data1, event.window.data2)
```

```
242: (16)                           elif event.window.event == sdl2.SDL_WINDOWEVENT_MINIMIZED:
243: (20)                               self._visible = False
244: (20)                               self._iconify_func(True)
245: (16)                           elif event.window.event == sdl2.SDL_WINDOWEVENT_RESTORED:
246: (20)                               self._visible = True
247: (20)                               self._iconify_func(False)
248: (4)                    def close(self) -> None:
249: (8)                        """Close the window"""
250: (8)                        super().close()
251: (8)                        self._close_func()
252: (4)                    def destroy(self) -> None:
253: (8)                        """Gracefully close the window"""
254: (8)                        sdl2.SDL_GL_DeleteContext(self._context)
255: (8)                        sdl2.SDL_DestroyWindow(self._window)
256: (8)                        sdl2.SDL_Quit()

----------------------------------------

File 33 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

----------------------------------------

File 34 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

----------------------------------------

File 35 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

----------------------------------------

File 36 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

----------------------------------------

File 37 - __init__.py:

1: (0)              from .keys import Keys  # noqa
2: (0)              from .window import Window  # noqa

----------------------------------------

File 38 - __init__.py:

1: (0)              from moderngl_window.context.tk.keys import Keys  # noqa
2: (0)              from moderngl_window.context.tk.window import Window  # noqa

----------------------------------------

File 39 - window.py:

1: (0)              import tkinter
2: (0)              from pathlib import Path
3: (0)              from typing import Any
4: (0)              from pyopengltk import OpenGLFrame
5: (0)              from moderngl_window.context.base import BaseWindow
6: (0)              from moderngl_window.context.tk.keys import Keys
7: (0)              class Window(BaseWindow):
    name = "tk"
    keys = Keys
    _mouse_button_map = {
        1: 1,
        3: 2,
        2: 3,
    }
    def __init__(self, **kwargs: Any):
        super().__init__(**kwargs)
        self._tk = tkinter.Tk()
        self._gl_widget = ModernglTkWindow(self._tk, width=self.width, height=self.height)
        self._gl_widget.pack(fill=tkinter.BOTH, expand=tkinter.YES)
        self._tk.resizable(self._resizable, self._resizable)
        if self._fullscreen:
            self._tk.attributes("-fullscreen", True)
        self.cursor = self._cursor
        self._gl_widget.bind("<Configure>", self.tk_resize)
        self._tk.bind("<KeyPress>", self.tk_key_press)
        self._tk.bind("<KeyRelease>", self.tk_key_release)
        self._tk.bind("<Motion>", self.tk_mouse_motion)
        self._tk.bind("<Button>", self.tk_mouse_button_press)
        self._tk.bind("<ButtonRelease>", self.tk_mouse_button_release)
        self._tk.bind("<MouseWheel>", self.tk_mouse_wheel)
        self._tk.bind("<Map>", self.tk_map)
        self._tk.bind("<Unmap>", self.tk_unmap)
        self._tk.protocol("WM_DELETE_WINDOW", self.tk_close_window)
        self.title = self._title
        self._tk.update()
        self._gl_widget.tkMakeCurrent()
        self.init_mgl_context()
        self.set_default_viewport()
    def _set_fullscreen(self, value: bool) -> None:
        self._tk.attributes("-fullscreen", value)
    def _set_vsync(self, value: bool) -> None:
        pass
    @property
    def size(self) -> tuple[int, int]:
        """tuple[int, int]: current window size.
        This property also support assignment::
            window.size = 1000, 1000
        """
        return self._width, self._height
    @size.setter
    def size(self, value: tuple[int, int]) -> None:
        self._tk.geometry("{}x{}".format(value[0], value[1]))
    @property
    def position(self) -> tuple[int, int]:
        """tuple[int, int]: The current window position.
        This property can also be set to move the window::
            window.position = 100, 100
        """
        _, x, y = self._tk.geometry().split("+")
        return int(x), int(y)
    @position.setter
    def position(self, value: tuple[int, int]) -> None:
        self._tk.geometry("+{}+{}".format(value[0], value[1]))
    @property
    def visible(self) -> bool:
        """bool: Is the window visible?
        This property can also be set::
            window.visible = False
        """
        return self._visible
    @visible.setter
    def visible(self, value: bool) -> None:
        self._visible = value
        if value:
            self._tk.deiconify()
```

```
76: (8)                         else:
77: (12)                            self._tk.withdraw()
78: (4)                     @property
79: (4)                     def cursor(self) -> bool:
80: (8)                         """bool: Should the mouse cursor be visible inside the window?
81: (8)                         This property can also be assigned to::
82: (12)                            window.cursor = False
83: (8)                         """
84: (8)                         return self._cursor
85: (4)                     @cursor.setter
86: (4)                     def cursor(self, value: bool) -> None:
87: (8)                         if value is True:
88: (12)                            self._tk.config(cursor="arrow")
89: (8)                         else:
90: (12)                            self._tk.config(cursor="none")
91: (8)                         self._cursor = value
92: (4)                     @property
93: (4)                     def title(self) -> str:
94: (8)                         """str: Window title.
95: (8)                         This property can also be set::
96: (12)                            window.title = "New Title"
97: (8)                         """
98: (8)                         return self._title
99: (4)                     @title.setter
100: (4)                    def title(self, value: str) -> None:
101: (8)                        self._tk.title(value)
102: (8)                        self._title = value
103: (4)                    def swap_buffers(self) -> None:
104: (8)                        """Swap buffers, set viewport, trigger events and increment frame
counter"""
105: (8)                        err = self._ctx.error
106: (8)                        if err != "GL_NO_ERROR":
107: (12)                           print(err)
108: (8)                        self._tk.update_idletasks()
109: (8)                        self._tk.update()
110: (8)                        self._gl_widget.tkSwapBuffers()
111: (8)                        self._frames += 1
112: (4)                    def _set_icon(self, icon_path: Path) -> None:
113: (8)                        self._tk.iconphoto(False, tkinter.PhotoImage(file=icon_path))
114: (4)                    def tk_key_press(self, event: tkinter.Event) -> None:
115: (8)                        """Handle all queued key press events in tkinter dispatching events to
standard methods"""
116: (8)                        self._key_event_func(event.keysym, self.keys.ACTION_PRESS,
self._modifiers)
117: (8)                        self._handle_modifiers(event, True)
118: (8)                        if event.char:
119: (12)                           self._unicode_char_entered_func(event.char)
120: (8)                        if self._exit_key is not None and event.keysym == self._exit_key:
121: (12)                           self.close()
122: (8)                        if self._fs_key is not None and event.keysym == self._fs_key:
123: (12)                           self.fullscreen = not self.fullscreen
124: (4)                    def tk_key_release(self, event: tkinter.Event) -> None:
125: (8)                        """Handle all queued key release events in tkinter dispatching events
to standard methods
126: (8)                        Args:
127: (12)                           event (tkinter.Event): The key release event
128: (8)                        """
129: (8)                        self._handle_modifiers(event, False)
130: (8)                        self._key_event_func(event.keysym, self.keys.ACTION_RELEASE,
self._modifiers)
131: (4)                    def tk_mouse_motion(self, event: tkinter.Event) -> None:
132: (8)                        """Handle and translate tkinter mouse position events
133: (8)                        Args:
134: (12)                           event (tkinter.Event): The mouse motion event
135: (8)                        """
136: (8)                        x, y = event.x, event.y
137: (8)                        dx, dy = self._calc_mouse_delta(x, y)
138: (8)                        if self._mouse_buttons.any:
139: (12)                           self._mouse_drag_event_func(x, y, dx, dy)
```

```
(8)               else:
(12)                  self._mouse_position_event_func(x, y, dx, dy)
(4)           def tk_mouse_button_press(self, event: tkinter.Event) -> None:
(8)               """Handle tkinter mouse press events.
(8)               Args:
(12)                  event (tkinter.Event): The mouse button press event
(8)               """
(8)               self._handle_modifiers(event, True)
(8)               button = self._mouse_button_map.get(event.num)
(8)               if not button:
(12)                  return
(8)               self._handle_mouse_button_state_change(button, True)
(8)               self._mouse_press_event_func(event.x, event.y, button)
(4)           def tk_mouse_button_release(self, event: tkinter.Event) -> None:
(8)               """Handle tkinter mouse press events.
(8)               Args:
(12)                  event (tkinter.Event): The mouse button release event
(8)               """
(8)               self._handle_modifiers(event, True)
(8)               button = self._mouse_button_map.get(event.num)
(8)               if not button:
(12)                  return
(8)               self._handle_mouse_button_state_change(button, False)
(8)               self._mouse_release_event_func(event.x, event.y, button)
(4)           def tk_mouse_wheel(self, event: tkinter.Event) -> None:
(8)               """Handle mouse wheel event.
(8)               Args:
(12)                  event (tkinter.Event): The mouse wheel event
(8)               """
(8)               self._handle_modifiers(event, True)
(8)               self._mouse_scroll_event_func(0, event.delta / 120.0)
(4)           def _handle_modifiers(self, event: tkinter.Event, press: bool) -> None:
(8)               """Update internal key modifiers
(8)               Args:
(12)                  event (tkinter.Event): The key event
(12)                  press (bool): Press or release event
(8)               """
(8)               if event.keysym in ["Shift_L", "Shift_R"]:
(12)                  self._modifiers.shift = press
(8)               elif event.keysym in ["Control_L", "Control_R"]:
(12)                  self._modifiers.ctrl = press
(8)               elif event.keysym in ["Alt_L", "Alt_R"]:
(12)                  self._modifiers.alt = press
(4)           def tk_resize(self, event: tkinter.Event) -> None:
(8)               """tkinter specific window resize event.
(8)               Forwards resize events to the configured resize function.
(8)               Args:
(12)                  event (tkinter.Event): The resize event
(8)               """
(8)               self._width, self._height = event.width, event.height
(8)               self._buffer_width, self._buffer_height = event.width, event.height
(8)               if not self._ctx:
(12)                  return
(8)               self.set_default_viewport()
(8)               self._resize_func(event.width, event.height)
(4)           def tk_close_window(self) -> None:
(8)               """tkinter close window callback"""
(8)               self._close_func()
(8)               self._close = True
(4)           def tk_map(self, event: tkinter.Event) -> None:
(8)               self._visible = True
(8)               self._iconify_func(False)
(4)           def tk_unmap(self, event: tkinter.Event) -> None:
(8)               self._visible = False
(8)               self._iconify_func(True)
(4)           def destroy(self) -> None:
(8)               """Destroy logic for tkinter window."""
(8)               self._tk.destroy()
(0)       class ModernglTkWindow(OpenGLFrame):
```

```
209: (4)                def __init__(self, *args: Any, **kwargs: Any):
210: (8)                    super().__init__(*args, **kwargs)
211: (4)                def redraw(self) -> None:
212: (8)                    """pyopengltk's own render method."""
213: (8)                    pass
214: (4)                def initgl(self) -> None:
215: (8)                    """pyopengltk's user code for initialization."""
216: (8)                    pass
217: (4)                def tkResize(self, event: tkinter.Event) -> None:
218: (8)                    """Should never be called. Event overridden."""
219: (8)                    raise ValueError("tkResize should never be called. The event is
overridden.")
220: (4)                def tkMap(self, event: tkinter.Event) -> None:
221: (8)                    """Called when frame goes onto the screen"""
222: (8)                    if not getattr(self, "_wid", None):
223: (12)                       super().tkMap(event)

----------------------------------------

File 40 - base.py:

1: (0)              """
2: (0)              Base finders
3: (0)              """
4: (0)              import functools
5: (0)              import logging
6: (0)              from collections import namedtuple
7: (0)              from pathlib import Path
8: (0)              from typing import Optional
9: (0)              from moderngl_window.conf import settings
10: (0)             from moderngl_window.exceptions import ImproperlyConfigured
11: (0)             from moderngl_window.utils.module_loading import import_string
12: (0)             FinderEntry = namedtuple("FinderEntry", ["path", "abspath", "exists"])
13: (0)             logger = logging.getLogger(__name__)
14: (0)             class BaseFilesystemFinder:
15: (4)                 """Base class for searching filesystem directories"""
16: (4)                 settings_attr = ""
17: (4)                 """str: Name of the attribute in
:py:class:`~moderngl_window.conf.Settings`
18: (4)                 containing a list of paths the finder should search in.
19: (4)                 """
20: (4)                 def __init__(self) -> None:
21: (8)                     """Initialize finder class by looking up the paths referenced in
``settings_attr``."""
22: (8)                     if not hasattr(settings, self.settings_attr):
23: (12)                        raise ImproperlyConfigured(
24: (16)                            "Settings doesn't define {}. "
25: (16)                            "This is required when using a
FileSystemFinder.".format(self.settings_attr)
26: (12)                        )
27: (8)                     self.paths = getattr(settings, self.settings_attr)
28: (4)                 def find(self, path: Path) -> Optional[Path]:
29: (8)                     """Finds a file in the configured paths returning its absolute path.
30: (8)                     Args:
31: (12)                        path (pathlib.Path): The path to find
32: (8)                     Returns:
33: (12)                        The absolute path to the file or None if not found
34: (8)                     """
35: (8)                     if getattr(self, "settings_attr", None):
36: (12)                        self.paths = getattr(settings, self.settings_attr)
37: (8)                     if not isinstance(path, Path):
38: (12)                        raise ValueError(
39: (16)                            "FilesystemFinders only take Path instances, not
{}".format(type(path))
40: (12)                        )
41: (8)                     logger.debug("find %s", path)
42: (8)                     if path.is_absolute():
43: (12)                        logger.debug("Ignoring absolute path: %s", path)
44: (12)                        return None
```

```
45: (8)                     logger.debug("paths: %s", self.paths)
46: (8)                     for search_path in self.paths:
47: (12)                        search_path = Path(search_path)
48: (12)                        if not search_path.is_absolute():
49: (16)                            raise ImproperlyConfigured("Search search path '{}' is not an
absolute path")
50: (12)                        abspath = search_path / path
51: (12)                        logger.debug("abspath %s", abspath)
52: (12)                        if abspath.exists():
53: (16)                            logger.debug("found %s", abspath)
54: (16)                            return Path(abspath)  # Needed to please mypy, but is already
be a path
55: (8)                     return None
56: (0)             @functools.lru_cache(maxsize=None)
57: (0)             def get_finder(import_path: str) -> BaseFilesystemFinder:
58: (4)                 """
59: (4)                 Get a finder class from an import path.
60: (4)                 This function uses an lru cache.
61: (4)                 Args:
62: (8)                     import_path: string representing an import path
63: (4)                 Return:
64: (8)                     An instance of the finder
65: (4)                 Raises:
66: (8)                     ImproperlyConfigured is the finder is not found
67: (4)                 """
68: (4)                 Finder = import_string(import_path)
69: (4)                 find = Finder()
70: (4)                 if not isinstance(find, BaseFilesystemFinder):
71: (8)                     raise ImproperlyConfigured(
72: (12)                        "Finder {} is not a subclass of
.finders.FileSystemFinder".format(import_path)
73: (8)                     )
74: (4)                 return find
```

File 41 - data.py:

```
1: (0)              from collections.abc import Iterable
2: (0)              from moderngl_window.conf import settings
3: (0)              from moderngl_window.finders import base
4: (0)              class FilesystemFinder(base.BaseFilesystemFinder):
5: (4)                  """Find data in ``settings.DATA_DIRS``"""
6: (4)                  settings_attr = "DATA_DIRS"
7: (0)              def get_finders() -> Iterable[base.BaseFilesystemFinder]:
8: (4)                  for finder in settings.DATA_FINDERS:
9: (8)                      yield base.get_finder(finder)
```

File 42 - bbox.py:

```
1: (0)              from typing import Optional
2: (0)              import moderngl
3: (0)              import numpy
4: (0)              from moderngl_window.geometry import AttributeNames
5: (0)              from moderngl_window.opengl.vao import VAO
6: (0)              def bbox(
7: (4)                  size: tuple[float, float, float] = (1.0, 1.0, 1.0),
8: (4)                  name: Optional[str] = None,
9: (4)                  attr_names: type[AttributeNames] = AttributeNames,
10: (0)             ) -> VAO:
11: (4)                 """
12: (4)                 Generates a bounding box with (0.0, 0.0, 0.0) as the center.
13: (4)                 This is simply a box with ``LINE_STRIP`` as draw mode.
14: (4)                 Keyword Args:
15: (8)                     size (tuple): x, y, z size of the box
16: (8)                     name (str): Optional name for the VAO
17: (8)                     attr_names (AttributeNames): Attribute names
```

```
(4)                 Returns:
(8)                     A :py:class:`moderngl_window.opengl.vao.VAO` instance
(4)                 """
(4)                 width, height, depth = size[0] / 2.0, size[1] / 2.0, size[2] / 2.0
(4)                 pos = numpy.array([
(8)                     width, -height, depth,
(8)                     width, height, depth,
(8)                     -width, -height, depth,
(8)                     width, height, depth,
(8)                     -width, height, depth,
(8)                     -width, -height, depth,
(8)                     width, -height, -depth,
(8)                     width, height, -depth,
(8)                     width, -height, depth,
(8)                     width, height, -depth,
(8)                     width, height, depth,
(8)                     width, -height, depth,
(8)                     width, -height, -depth,
(8)                     width, -height, depth,
(8)                     -width, -height, depth,
(8)                     width, -height, -depth,
(8)                     -width, -height, depth,
(8)                     -width, -height, -depth,
(8)                     -width, -height, depth,
(8)                     -width, height, depth,
(8)                     -width, height, -depth,
(8)                     -width, -height, depth,
(8)                     -width, height, -depth,
(8)                     -width, -height, -depth,
(8)                     width, height, -depth,
(8)                     width, -height, -depth,
(8)                     -width, -height, -depth,
(8)                     width, height, -depth,
(8)                     -width, -height, -depth,
(8)                     -width, height, -depth,
(8)                     width, height, -depth,
(8)                     -width, height, -depth,
(8)                     width, height, depth,
(8)                     -width, height, -depth,
(8)                     -width, height, depth,
(8)                     width, height, depth,
(4)                 ], dtype=numpy.float32)
(4)                 vao = VAO(name or "geometry:cube", mode=moderngl.LINE_STRIP)
(4)                 vao.buffer(pos, "3f", [attr_names.POSITION])
(4)                 return vao
```

File 43 - cube.py:

```
(0)             from typing import Optional
(0)             import numpy
(0)             from moderngl_window.geometry import AttributeNames
(0)             from moderngl_window.opengl.vao import VAO
(0)             def cube(
(4)                 size: tuple[float, float, float] = (1.0, 1.0, 1.0),
(4)                 center: tuple[float, float, float] = (0.0, 0.0, 0.0),
(4)                 normals: bool = True,
(4)                 uvs: bool = True,
(4)                 name: Optional[str] = None,
(4)                 attr_names: type[AttributeNames] = AttributeNames,
(0)             ) -> VAO:
(4)                 """Creates a cube VAO with normals and texture coordinates
(4)                 Keyword Args:
(8)                     width (float): Width of the cube
(8)                     height (float): Height of the cube
(8)                     depth (float): Depth of the cube
(8)                     center: center of the cube as a 3-component tuple
(8)                     normals: (bool) Include normals
```

```
20: (8)                     uvs: (bool) include uv coordinates
21: (8)                     name (str): Optional name for the VAO
22: (8)                     attr_names (AttributeNames): Attribute names
23: (4)                 Returns:
24: (8)                     A :py:class:`moderngl_window.opengl.vao.VAO` instance
25: (4)                 """
26: (4)                 width, height, depth = size
27: (4)                 width, height, depth = width / 2.0, height / 2.0, depth / 2.0
28: (4)                 pos = numpy.array([
29: (8)                     center[0] + width, center[1] - height, center[2] + depth,
30: (8)                     center[0] + width, center[1] + height, center[2] + depth,
31: (8)                     center[0] - width, center[1] - height, center[2] + depth,
32: (8)                     center[0] + width, center[1] + height, center[2] + depth,
33: (8)                     center[0] - width, center[1] + height, center[2] + depth,
34: (8)                     center[0] - width, center[1] - height, center[2] + depth,
35: (8)                     center[0] + width, center[1] - height, center[2] - depth,
36: (8)                     center[0] + width, center[1] + height, center[2] - depth,
37: (8)                     center[0] + width, center[1] - height, center[2] + depth,
38: (8)                     center[0] + width, center[1] + height, center[2] - depth,
39: (8)                     center[0] + width, center[1] + height, center[2] + depth,
40: (8)                     center[0] + width, center[1] - height, center[2] + depth,
41: (8)                     center[0] + width, center[1] - height, center[2] - depth,
42: (8)                     center[0] + width, center[1] - height, center[2] + depth,
43: (8)                     center[0] - width, center[1] - height, center[2] + depth,
44: (8)                     center[0] + width, center[1] - height, center[2] - depth,
45: (8)                     center[0] - width, center[1] - height, center[2] + depth,
46: (8)                     center[0] - width, center[1] - height, center[2] - depth,
47: (8)                     center[0] - width, center[1] - height, center[2] + depth,
48: (8)                     center[0] - width, center[1] + height, center[2] + depth,
49: (8)                     center[0] - width, center[1] + height, center[2] - depth,
50: (8)                     center[0] - width, center[1] - height, center[2] + depth,
51: (8)                     center[0] - width, center[1] + height, center[2] - depth,
52: (8)                     center[0] - width, center[1] - height, center[2] - depth,
53: (8)                     center[0] + width, center[1] + height, center[2] - depth,
54: (8)                     center[0] + width, center[1] - height, center[2] - depth,
55: (8)                     center[0] - width, center[1] - height, center[2] - depth,
56: (8)                     center[0] + width, center[1] + height, center[2] - depth,
57: (8)                     center[0] - width, center[1] - height, center[2] - depth,
58: (8)                     center[0] - width, center[1] + height, center[2] - depth,
59: (8)                     center[0] + width, center[1] + height, center[2] - depth,
60: (8)                     center[0] - width, center[1] + height, center[2] - depth,
61: (8)                     center[0] + width, center[1] + height, center[2] + depth,
62: (8)                     center[0] - width, center[1] + height, center[2] - depth,
63: (8)                     center[0] - width, center[1] + height, center[2] + depth,
64: (8)                     center[0] + width, center[1] + height, center[2] + depth,
65: (4)                 ], dtype=numpy.float32)
66: (4)                 if normals:
67: (8)                     normal_data = numpy.array([
68: (12)                        -0, 0, 1,
69: (12)                        -0, 0, 1,
70: (12)                        -0, 0, 1,
71: (12)                        0, 0, 1,
72: (12)                        0, 0, 1,
73: (12)                        0, 0, 1,
74: (12)                        1, 0, 0,
75: (12)                        1, 0, 0,
76: (12)                        1, 0, 0,
77: (12)                        1, 0, 0,
78: (12)                        1, 0, 0,
79: (12)                        1, 0, 0,
80: (12)                        0, -1, 0,
81: (12)                        0, -1, 0,
82: (12)                        0, -1, 0,
83: (12)                        0, -1, 0,
84: (12)                        0, -1, 0,
85: (12)                        0, -1, 0,
86: (12)                        -1, -0, 0,
87: (12)                        -1, -0, 0,
88: (12)                        -1, -0, 0,
            -1, -0, 0,
            -1, -0, 0,
            -1, -0, 0,
            0, 0, -1,
            0, 0, -1,
            0, 0, -1,
            0, 0, -1,
            0, 0, -1,
            0, 0, -1,
            0, 1, 0,
            0, 1, 0,
            0, 1, 0,
            0, 1, 0,
            0, 1, 0,
            0, 1, 0,
        ], dtype=numpy.float32)
    if uvs:
        uvs_data = numpy.array([
            1, 0,
            1, 1,
            0, 0,
            1, 1,
            0, 1,
            0, 0,
            1, 0,
            1, 1,
            0, 0,
            1, 1,
            0, 1,
            0, 0,
            1, 1,
            0, 1,
            0, 0,
            1, 1,
            0, 0,
            1, 0,
            0, 1,
            0, 0,
            1, 0,
            0, 1,
            1, 0,
            1, 1,
            1, 0,
            1, 1,
            0, 1,
            1, 0,
            0, 1,
            0, 0,
            1, 1,
            0, 1,
            1, 0,
            0, 1,
            0, 0,
            1, 0
        ], dtype=numpy.float32)
    vao = VAO(name or "geometry:cube")
    vao.buffer(pos, "3f", [attr_names.POSITION])
    if normals:
        vao.buffer(normal_data, "3f", [attr_names.NORMAL])
    if uvs:
        vao.buffer(uvs_data, "2f", [attr_names.TEXCOORD_0])
    return vao
```

File 44 - quad.py:

```
from typing import Optional
import moderngl
```

```
3: (0)              import numpy
4: (0)              from moderngl_window.geometry.attributes import AttributeNames
5: (0)              from moderngl_window.opengl.vao import VAO
6: (0)              def quad_fs(
7: (4)                  attr_names: type[AttributeNames] = AttributeNames,
8: (4)                  normals: bool = True,
9: (4)                  uvs: bool = True,
10: (4)                 name: Optional[str] = None,
11: (0)             ) -> VAO:
12: (4)                 """
13: (4)                 Creates a screen aligned quad using two triangles with normals and texture coordinates.
14: (4)                 Keyword Args:
15: (8)                     attr_names (AttributeNames): Attrib name config
16: (8)                     normals (bool): Include normals in VAO
17: (8)                     uvs (bool): Include texture coordinates in VAO
18: (8)                     name (str): Optional name for the VAO
19: (4)                 Returns:
20: (8)                     A :py:class:`~moderngl_window.opengl.vao.VAO` instance.
21: (4)                 """
22: (4)                 return quad_2d(
23: (8)                     size=(2.0, 2.0),
24: (8)                     normals=normals,
25: (8)                     uvs=uvs,
26: (8)                     attr_names=attr_names,
27: (8)                     name=name,
28: (4)                 )
29: (0)             def quad_2d(
30: (4)                 size: tuple[float, float] = (1.0, 1.0),
31: (4)                 pos: tuple[float, float] = (0.0, 0.0),
32: (4)                 normals: bool = True,
33: (4)                 uvs: bool = True,
34: (4)                 attr_names: type[AttributeNames] = AttributeNames,
35: (4)                 name: Optional[str] = None,
36: (0)             ) -> VAO:
37: (4)                 """
38: (4)                 Creates a 2D quad VAO using 2 triangles with normals and texture coordinates.
39: (4)                 Keyword Args:
40: (8)                     size (tuple): width and height
41: (8)                     pos (float): Center position x and y
42: (8)                     normals (bool): Include normals in VAO
43: (8)                     uvs (bool): Include texture coordinates in VAO
44: (8)                     attr_names (AttributeNames): Attrib name config
45: (8)                     name (str): Optional name for the VAO
46: (4)                 Returns:
47: (8)                     A :py:class:`~moderngl_window.opengl.vao.VAO` instance.
48: (4)                 """
49: (4)                 width, height = size
50: (4)                 xpos, ypos = pos
51: (4)                 pos_data = numpy.array([
52: (8)                     xpos - width / 2.0, ypos + height / 2.0, 0.0,
53: (8)                     xpos - width / 2.0, ypos - height / 2.0, 0.0,
54: (8)                     xpos + width / 2.0, ypos - height / 2.0, 0.0,
55: (8)                     xpos - width / 2.0, ypos + height / 2.0, 0.0,
56: (8)                     xpos + width / 2.0, ypos - height / 2.0, 0.0,
57: (8)                     xpos + width / 2.0, ypos + height / 2.0, 0.0,
58: (4)                 ], dtype=numpy.float32)
59: (4)                 normal_data = numpy.array([
60: (8)                     0.0, 0.0, 1.0,
61: (8)                     0.0, 0.0, 1.0,
62: (8)                     0.0, 0.0, 1.0,
63: (8)                     0.0, 0.0, 1.0,
64: (8)                     0.0, 0.0, 1.0,
65: (8)                     0.0, 0.0, 1.0,
66: (4)                 ], dtype=numpy.float32)
67: (4)                 uv_data = numpy.array([
68: (8)                     0.0, 1.0,
69: (8)                     0.0, 0.0,
70: (8)                     1.0, 0.0,
71: (8)                     0.0, 1.0,
72: (8)                     1.0, 0.0,
73: (8)                     1.0, 1.0,
74: (4)                 ], dtype=numpy.float32)
75: (4)                 vao = VAO(name or "geometry:quad", mode=moderngl.TRIANGLES)
76: (4)                 vao.buffer(pos_data, "3f", [attr_names.POSITION])
77: (4)                 if normals:
78: (8)                     vao.buffer(normal_data, "3f", [attr_names.NORMAL])
79: (4)                 if uvs:
80: (8)                     vao.buffer(uv_data, "2f", [attr_names.TEXCOORD_0])
81: (4)                 return vao
```

File 45 - scene.py:

```
1: (0)              from collections.abc import Iterator
2: (0)              from moderngl_window.conf import settings
3: (0)              from moderngl_window.finders import base
4: (0)              class FilesystemFinder(base.BaseFilesystemFinder):
5: (4)                  """Find scenes in ``settings.SCENE_DIRS``"""
6: (4)                  settings_attr = "SCENE_DIRS"
7: (0)              def get_finders() -> Iterator[base.BaseFilesystemFinder]:
8: (4)                  for finder in settings.SCENE_FINDERS:
9: (8)                      yield base.get_finder(finder)
```

File 46 - program.py:

```
1: (0)              from collections.abc import Iterator
2: (0)              from moderngl_window.conf import settings
3: (0)              from moderngl_window.finders import base
4: (0)              class FilesystemFinder(base.BaseFilesystemFinder):
5: (4)                  """Find shaders in ``settings.PROGRAM_DIRS``"""
6: (4)                  settings_attr = "PROGRAM_DIRS"
7: (0)              def get_finders() -> Iterator[base.BaseFilesystemFinder]:
8: (4)                  for finder in settings.PROGRAM_FINDERS:
9: (8)                      yield base.get_finder(finder)
```

File 47 - texture.py:

```
1: (0)              from collections.abc import Iterator
2: (0)              from moderngl_window.conf import settings
3: (0)              from moderngl_window.finders import base
4: (0)              class FilesystemFinder(base.BaseFilesystemFinder):
5: (4)                  """Find textures in ``settings.TEXTURE_DIRS``"""
6: (4)                  settings_attr = "TEXTURE_DIRS"
7: (0)              def get_finders() -> Iterator[base.BaseFilesystemFinder]:
8: (4)                  for finder in settings.TEXTURE_FINDERS:
9: (8)                      yield base.get_finder(finder)
```

File 48 - __init__.py:

```
1: (0)
```

File 49 - __init__.py:

```
1: (0)              from moderngl_window.geometry.attributes import AttributeNames as
AttributeNames
2: (0)              from moderngl_window.geometry.bbox import bbox as bbox
3: (0)              from moderngl_window.geometry.cube import cube as cube
```

```
4: (0)              from moderngl_window.geometry.quad import quad_2d as quad_2d
5: (0)              from moderngl_window.geometry.quad import quad_fs as quad_fs
6: (0)              from moderngl_window.geometry.sphere import sphere as sphere
7: (0)              __all__ = [
8: (4)                  "AttributeNames",
9: (4)                  "bbox",
10: (4)                 "cube",
11: (4)                 "quad_2d",
12: (4)                 "quad_fs",
13: (4)                 "sphere",
14: (0)             ]

----------------------------------------

File 50 - attributes.py:

1: (0)              """
2: (0)              Follows the standard attributes from GLFT2.0
3: (0)
https://github.com/KhronosGroup/glTF/blob/master/specification/2.0/README.md#meshes
4: (0)              """
5: (0)              from typing import Any, Optional
6: (0)              class AttributeNames:
7: (4)                  """Standard buffer/attribute names.
8: (4)                  This works as a lookup for buffer names when creating VAO instances.
9: (4)                  This class can be used directly or an instance of the class can be used
with overrides.
10: (4)                 Optionally it can be extended into a new class.
11: (4)                 """
12: (4)                 POSITION = "in_position"
13: (4)                 NORMAL = "in_normal"
14: (4)                 TANGENT = "in_tangent"
15: (4)                 TEXCOORD_0 = "in_texcoord_0"
16: (4)                 TEXCOORD_1 = "in_texcoord_1"
17: (4)                 COLOR_0 = "in_color0"
18: (4)                 JOINTS_0 = "in_joints_0"
19: (4)                 WEIGHTS_0 = "in_weights_0"
20: (4)                 def __init__(
21: (8)                     self,
22: (8)                     position: Optional[str] = None,
23: (8)                     normal: Optional[str] = None,
24: (8)                     tangent: Optional[str] = None,
25: (8)                     texcoord_0: Optional[str] = None,
26: (8)                     texcoord_1: Optional[str] = None,
27: (8)                     color_0: Optional[str] = None,
28: (8)                     joints_0: Optional[str] = None,
29: (8)                     weights: Optional[str] = None,
30: (8)                     **kwargs: Any,
31: (4)                 ):
32: (8)                     """Override default values.
33: (8)                     All attributes will be set on the instance as upper case strings
34: (8)                     Keyword Args:
35: (16)                            position (str): Name for position buffers/attribute
36: (16)                            normal (str): Name for normal buffer/attribute
37: (16)                            tangent (str): name for tangent buffer/attribute
38: (16)                            texcoord_0 (str): Name for texcoord 0 buffer/attribute
39: (16)                            texcoord_1 (str): Name for texcoord 1 buffer/attribute
40: (16)                            color_0 (str): name for vertex color buffer/attribute
41: (16)                            joints_0 (str): Name for joints buffer/attribute
42: (16)                            weights (str): Name for weights buffer/attribute
43: (8)                     """
44: (8)                     self.apply_values(
45: (12)                        {
46: (16)                            "position": position,
47: (16)                            "normal": normal,
48: (16)                            "tangent": tangent,
49: (16)                            "texcoord_0": texcoord_0,
50: (16)                            "texcoord_1": texcoord_1,
51: (16)                            "color_0": color_0,
(16)                        "joints_0": joints_0,
(16)                        "weights": weights,
(16)                        **kwargs,
(12)                    }
(8)                 )
(4)             def apply_values(self, kwargs: dict[str, Any]) -> None:
(8)                 """Only applies attribute values not None"""
(8)                 for key, value in kwargs.items():
(12)                    if value:
(16)                        setattr(self, key.upper(), value)
```

File 51 - sphere.py:

```
(0)             import math
(0)             from typing import Optional
(0)             import moderngl as mlg
(0)             import numpy
(0)             from moderngl_window.geometry import AttributeNames
(0)             from moderngl_window.opengl.vao import VAO
(0)             def sphere(
(4)                 radius: float = 0.5,
(4)                 sectors: int = 32,
(4)                 rings: int = 16,
(4)                 normals: bool = True,
(4)                 uvs: bool = True,
(4)                 name: Optional[str] = None,
(4)                 attr_names: type[AttributeNames] = AttributeNames,
(0)             ) -> VAO:
(4)                 """Creates a sphere.
(4)                 Keyword Args:
(8)                     radius (float): Radius or the sphere
(8)                     rings (int): number or horizontal rings
(8)                     sectors (int): number of vertical segments
(8)                     normals (bool): Include normals in the VAO
(8)                     uvs (bool): Include texture coordinates in the VAO
(8)                     name (str): An optional name for the VAO
(8)                     attr_names (AttributeNames): Attribute names
(4)                 Returns:
(8)                     A :py:class:`VAO` instance
(4)                 """
(4)                 R = 1.0 / (rings - 1)
(4)                 S = 1.0 / (sectors - 1)
(4)                 vertices_l = [0.0] * (rings * sectors * 3)
(4)                 normals_l = [0.0] * (rings * sectors * 3)
(4)                 uvs_l = [0.0] * (rings * sectors * 2)
(4)                 v, n, t = 0, 0, 0
(4)                 for r in range(rings):
(8)                     for s in range(sectors):
(12)                        y = math.sin(-math.pi / 2 + math.pi * r * R)
(12)                        x = math.cos(2 * math.pi * s * S) * math.sin(math.pi * r * R)
(12)                        z = math.sin(2 * math.pi * s * S) * math.sin(math.pi * r * R)
(12)                        uvs_l[t] = s * S
(12)                        uvs_l[t + 1] = r * R
(12)                        vertices_l[v] = x * radius
(12)                        vertices_l[v + 1] = y * radius
(12)                        vertices_l[v + 2] = z * radius
(12)                        normals_l[n] = x
(12)                        normals_l[n + 1] = y
(12)                        normals_l[n + 2] = z
(12)                        t += 2
(12)                        v += 3
(12)                        n += 3
(4)                 indices = [0] * rings * sectors * 6
(4)                 i = 0
(4)                 for r in range(rings - 1):
(8)                     for s in range(sectors - 1):
(12)                        indices[i] = r * sectors + s
```

```
55: (12)                        indices[i + 1] = (r + 1) * sectors + (s + 1)
56: (12)                        indices[i + 2] = r * sectors + (s + 1)
57: (12)                        indices[i + 3] = r * sectors + s
58: (12)                        indices[i + 4] = (r + 1) * sectors + s
59: (12)                        indices[i + 5] = (r + 1) * sectors + (s + 1)
60: (12)                        i += 6
61: (4)                 vao = VAO(name or "sphere", mode=mlg.TRIANGLES)
62: (4)                 vbo_vertices = numpy.array(vertices_l, dtype=numpy.float32)
63: (4)                 vao.buffer(vbo_vertices, "3f", [attr_names.POSITION])
64: (4)                 if normals:
65: (8)                     vbo_normals = numpy.array(normals_l, dtype=numpy.float32)
66: (8)                     vao.buffer(vbo_normals, "3f", [attr_names.NORMAL])
67: (4)                 if uvs:
68: (8)                     vbo_uvs = numpy.array(uvs_l, dtype=numpy.float32)
69: (8)                     vao.buffer(vbo_uvs, "2f", [attr_names.TEXCOORD_0])
70: (4)                 vbo_elements = numpy.array(indices, dtype=numpy.uint32)
71: (4)                 vao.index_buffer(vbo_elements, index_element_size=4)
72: (4)                 return vao
```

File 52 - stl.py:

```
1: (0)                import gzip
2: (0)                from pathlib import Path
3: (0)                import moderngl
4: (0)                import numpy
5: (0)                import trimesh
6: (0)                from moderngl_window.exceptions import ImproperlyConfigured
7: (0)                from moderngl_window.loaders.base import BaseLoader
8: (0)                from moderngl_window.opengl.vao import VAO
9: (0)                from moderngl_window.scene import Material, Mesh, Node, Scene
10: (0)               class Loader(BaseLoader):
11: (4)                   kind = "stl"
12: (4)                   file_extensions = [
13: (8)                       [".stl"],
14: (8)                       [".stl", ".gz"],
15: (4)                   ]
16: (4)                   def load(self) -> Scene:
17: (8)                       """Loads and stl scene/file
18: (8)                       Returns:
19: (12)                          Scene: The Scene instance
20: (8)                       """
21: (8)                       path = self.find_scene(self.meta.path)
22: (8)                       if not path:
23: (12)                          raise ImproperlyConfigured("Scene '{}' not
found".format(self.meta.path))
24: (8)                       file_obj = str(path)
25: (8)                       if file_obj.endswith(".gz"):
26: (12)                          file_obj = gzip.GzipFile(file_obj)
27: (8)                       stl_mesh = trimesh.load(file_obj, file_type="stl")
28: (8)                       path = self.meta.resolved_path
29: (8)                       if isinstance(path, Path):
30: (12)                          resolved = path.as_posix()
31: (8)                       else:
32: (12)                          resolved = None
33: (8)                       scene = Scene(resolved)
34: (8)                       scene_mesh = Mesh("mesh")
35: (8)                       scene_mesh.material = Material("default")
36: (8)                       vao = VAO("mesh", mode=moderngl.TRIANGLES)
37: (8)                       vao.buffer(numpy.array(stl_mesh.vertices, dtype="f4"), "3f",
["in_position"])
38: (8)                       vao.buffer(numpy.array(stl_mesh.vertex_normals, dtype="f4"), "3f",
["in_normal"])
39: (8)                       vao.index_buffer(numpy.array(stl_mesh.faces, dtype="u4"))
40: (8)                       scene_mesh.vao = vao
41: (8)                       scene_mesh.add_attribute("POSITION", "in_position", 3)
42: (8)                       scene_mesh.add_attribute("NORMAL", "in_normal", 3)
43: (8)                       scene.meshes.append(scene_mesh)
```

```
44: (8)                 scene.root_nodes.append(Node(mesh=scene_mesh))
45: (8)                 scene.prepare()
46: (8)                 return scene
```

File 53 - t2d.py:

```
1: (0)               import logging
2: (0)               import moderngl
3: (0)               from moderngl_window.loaders.texture.pillow import PillowLoader, image_data
4: (0)               logger = logging.getLogger(__name__)
5: (0)               class Loader(PillowLoader):
6: (4)                   kind = "2d"
7: (4)                   def load(self) -> moderngl.Texture:
8: (8)                       """Load a 2d texture as configured in the supplied
``TextureDescription``
9: (8)                       Returns:
10: (12)                          moderngl.Texture: The Texture instance
11: (8)                       """
12: (8)                       self._open_image()
13: (8)                       components, data = image_data(self.image)
14: (8)                       texture = self.ctx.texture(
15: (12)                          self.image.size,
16: (12)                          components,
17: (12)                          data,
18: (8)                       )
19: (8)                       texture.extra = {"meta": self.meta}
20: (8)                       if self.meta.mipmap_levels is not None:
21: (12)                          self.meta.mipmap = True
22: (8)                       if self.meta.mipmap:
23: (12)                          if isinstance(self.meta.mipmap_levels, tuple):
24: (16)                              texture.build_mipmaps(*self.meta.mipmap_levels)
25: (12)                          else:
26: (16)                              texture.build_mipmaps()
27: (12)                          if self.meta.anisotropy:
28: (16)                              texture.anisotropy = self.meta.anisotropy
29: (8)                       self._close_image()
30: (8)                       return texture
```

File 54 - vao.py:

```
1: (0)               from typing import Any, Optional, Union
2: (0)               import moderngl
3: (0)               import numpy
4: (0)               import numpy.typing as npt
5: (0)               import moderngl_window as mglw
6: (0)               from moderngl_window.opengl import types
7: (0)               DRAW_MODES = {
8: (4)                   moderngl.TRIANGLES: "TRIANGLES",
9: (4)                   moderngl.TRIANGLE_FAN: "TRIANGLE_FAN",
10: (4)                  moderngl.TRIANGLE_STRIP: "TRIANGLE_STRIP",
11: (4)                  moderngl.TRIANGLES_ADJACENCY: "TRIANGLES_ADJACENCY",
12: (4)                  moderngl.TRIANGLE_STRIP_ADJACENCY: "TRIANGLE_STRIP_ADJACENCY",
13: (4)                  moderngl.POINTS: "POINTS",
14: (4)                  moderngl.LINES: "LINES",
15: (4)                  moderngl.LINE_STRIP: "LINE_STRIP",
16: (4)                  moderngl.LINE_LOOP: "LINE_LOOP",
17: (4)                  moderngl.LINES_ADJACENCY: "LINES_ADJACENCY",
18: (0)              }
19: (0)              class BufferInfo:
20: (4)                  """Container for a vbo with additional information"""
21: (4)                  def __init__(
22: (8)                      self,
23: (8)                      buffer: moderngl.Buffer,
24: (8)                      buffer_format: str,
25: (8)                      attributes: list[str] = [],
(8)          per_instance: bool = False,
(4)      ):
(8)          """
(8)          :param buffer: The vbo object
(8)          :param format: The format of the buffer
(8)          """
(8)          self.buffer = buffer
(8)          self.attrib_formats = types.parse_attribute_formats(buffer_format)
(8)          self.attributes = attributes
(8)          self.per_instance = per_instance
(8)          if self.buffer.size % self.vertex_size != 0:
(12)             raise VAOError(
(16)                 "Buffer with type {} has size not aligning with {}. Remainder: {}".format(
(20)                     buffer_format, self.vertex_size, self.buffer.size % self.vertex_size
(16)                 )
(12)             )
(8)          self.vertices = self.buffer.size // self.vertex_size
(4)      @property
(4)      def vertex_size(self) -> int:
(8)          return sum(f.bytes_total for f in self.attrib_formats)
(4)      def content(self, attributes: list[str]) -> Optional[tuple[object, ...]]:
(8)          """Build content tuple for the buffer
(8)          Returns
(12)             The first value is the moderngl buffer
(12)             From the third to the end, it is the attributes of the class"""
(8)          formats = []
(8)          attrs = []
(8)          for attrib_format, attrib in zip(self.attrib_formats, self.attributes):
(12)             if attrib not in attributes:
(16)                 formats.append(attrib_format.pad_str())
(16)                 continue
(12)             formats.append(attrib_format.format)
(12)             attrs.append(attrib)
(12)             attributes.remove(attrib)
(8)          if len(attrs) == 0:
(12)             return None
(8)          return (
(12)             self.buffer,
(12)             "{}{}".format(" ".join(formats), "/i" if self.per_instance else ""),
(12)             *attrs,
(8)          )
(4)      def has_attribute(self, name: str) -> bool:
(8)          return name in self.attributes
(0)  class VAO:
(4)      """
(4)      Represents a vertex array object.
(4)      This is a wrapper class over ``moderngl.VertexArray`` to make interactions
(4)      with programs/shaders simpler. Named buffers are added corresponding with
(4)      attribute names in a vertex shader. When rendering the VAO an internal
(4)      ``moderngl.VertextArray`` is created automatically mapping the named buffers
(4)      compatible with the supplied program. This program is cached internally.
(4)      The shader program doesn't need to use all the buffers registered in
(4)      this wrapper. When a subset is used only the used buffers are mapped
(4)      and the appropriate padding is calculated when interleaved data is used.
(4)      You are not required to use this class, but most methods in the
(4)      system creating vertexbuffers will return this type. You can obtain
(4)      a single ``moderngl.VertexBuffer`` instance by calling :py:meth:`VAO.instance`
(4)      method if you prefer to work directly on moderngl instances.
(4)      Example::
(8)          vao = VAO(name="test", mode=moderngl.POINTS)
(8)          vao.buffer(positions, '3f', ['in_position'])
(8)          vao.buffer(velocities, '3f', ['in_velocities'])
(8)          vao = VAO(name="test", mode=moderngl.POINTS)
```

```
89: (8)                     vao.buffer(interleaved_data, '3f 3f', ['in_position', 'in_velocities'])
90: (4)                 .. code:: glsl
91: (8)                     in vec3 in_position;
92: (8)                     in vec3 in_velocities;
93: (4)                 """
94: (4)                 def __init__(self, name: str = "", mode: int = moderngl.TRIANGLES):
95: (8)                     """Create and empty VAO with a name and default render mode.
96: (8)                     Example::
97: (12)                        VAO(name="cube", mode=moderngl.TRIANGLES)
98: (8)                     Keyword Args:
99: (12)                        name (str): Optional name for debug purposes
100: (12)                       mode (int): Default draw mode
101: (8)                    """
102: (8)                    self.name = name
103: (8)                    self.mode = mode
104: (8)                    try:
105: (12)                       DRAW_MODES[self.mode]
106: (8)                    except KeyError:
107: (12)                       raise VAOError("Invalid draw mode. Options are {}".format(DRAW_MODES.values()))
108: (8)                    self._buffers: list[BufferInfo] = []
109: (8)                    self._index_buffer: Optional[moderngl.Buffer] = None
110: (8)                    self._index_element_size: Optional[int] = None
111: (8)                    self.vertex_count = 0
112: (8)                    self.vaos: dict[Any, moderngl.VertexArray] = {}
113: (4)                @property
114: (4)                def ctx(self) -> moderngl.Context:
115: (8)                    """moderngl.Context: The active moderngl context"""
116: (8)                    return mglw.ctx()
117: (4)                def render(
118: (8)                    self,
119: (8)                    program: moderngl.Program,
120: (8)                    mode: Optional[int] = None,
121: (8)                    vertices: int = -1,
122: (8)                    first: int = 0,
123: (8)                    instances: int = 1,
124: (4)                ) -> None:
125: (8)                    """Render the VAO.
126: (8)                    An internal ``moderngl.VertexBuffer`` with compatible buffer bindings
127: (8)                    is automatically created on the fly and cached internally.
128: (8)                    Args:
129: (12)                       program: The ``moderngl.Program``
130: (8)                    Keyword Args:
131: (12)                       mode: Override the draw mode (``TRIANGLES`` etc)
132: (12)                       vertices (int): The number of vertices to transform
133: (12)                       first (int): The index of the first vertex to start with
134: (12)                       instances (int): The number of instances
135: (8)                    """
136: (8)                    vao = self.instance(program)
137: (8)                    if mode is None:
138: (12)                       mode = self.mode
139: (8)                    vao.render(mode, vertices=vertices, first=first, instances=instances)
140: (4)                def render_indirect(
141: (8)                    self,
142: (8)                    program: moderngl.Program,
143: (8)                    buffer: moderngl.Buffer,
144: (8)                    mode: Optional[int] = None,
145: (8)                    count: int = -1,
146: (8)                    *,
147: (8)                    first: int = 0,
148: (4)                ) -> None:
149: (8)                    """
150: (8)                    The render primitive (mode) must be the same as the input primitive of the
151: (8)                    GeometryShader.
152: (8)                    The draw commands are 5 integers:
153: (8)                    (count, instanceCount, firstIndex, baseVertex, baseInstance).
154: (8)                    Args:
```

```
155: (12)                       program: The ``moderngl.Program``
156: (12)                       buffer: The ``moderngl.Buffer`` containing indirect draw commands
157: (8)                    Keyword Args:
158: (12)                       mode (int): By default :py:data:`TRIANGLES` will be used.
159: (12)                       count (int): The number of draws.
160: (12)                       first (int): The index of the first indirect draw command.
161: (8)                    """
162: (8)                    vao = self.instance(program)
163: (8)                    if mode is None:
164: (12)                       mode = self.mode
165: (8)                    vao.render_indirect(buffer, mode=mode, count=count, first=first)
166: (4)                def transform(
167: (8)                    self,
168: (8)                    program: moderngl.Program,
169: (8)                    buffer: moderngl.Buffer,
170: (8)                    mode: Optional[int] = None,
171: (8)                    vertices: int = -1,
172: (8)                    first: int = 0,
173: (8)                    instances: int = 1,
174: (4)                ) -> None:
175: (8)                    """Transform vertices. Stores the output in a single buffer.
176: (8)                    Args:
177: (12)                       program: The ``moderngl.Program``
178: (12)                       buffer: The ``moderngl.buffer`` to store the output
179: (8)                    Keyword Args:
180: (12)                       mode: Draw mode (for example ``moderngl.POINTS``)
181: (12)                       vertices (int): The number of vertices to transform
182: (12)                       first (int): The index of the first vertex to start with
183: (12)                       instances (int): The number of instances
184: (8)                    """
185: (8)                    vao = self.instance(program)
186: (8)                    if mode is None:
187: (12)                       mode = self.mode
188: (8)                    vao.transform(buffer, mode=mode, vertices=vertices, first=first,
instances=instances)
189: (4)                def buffer(
190: (8)                    self,
191: (8)                    buffer: Union[moderngl.Buffer, npt.NDArray[Any], bytes],
192: (8)                    buffer_format: str,
193: (8)                    attribute_names: Union[list[str], str],
194: (4)                ) -> moderngl.Buffer:
195: (8)                    """Register a buffer/vbo for the VAO. This can be called multiple
times.
196: (8)                    adding multiple buffers (interleaved or not).
197: (8)                    Args:
198: (12)                       buffer:
199: (16)                           The buffer data. Can be ``numpy.array``, ``moderngl.Buffer``
or ``bytes``.
200: (12)                       buffer_format (str):
201: (16)                           The format of the buffer. (eg. ``3f 3f`` for interleaved
positions and normals).
202: (12)                       attribute_names:
203: (16)                           A list of attribute names this buffer should map to.
204: (8)                    Returns:
205: (12)                       The ``moderngl.Buffer`` instance object. This is handy when
providing ``bytes``
206: (12)                       and ``numpy.array``.
207: (8)                    """
208: (8)                    if not isinstance(attribute_names, list):
209: (12)                       attribute_names = [
210: (16)                           attribute_names,
211: (12)                       ]
212: (8)                    if type(buffer) not in [moderngl.Buffer, numpy.ndarray, bytes]:
213: (12)                       raise VAOError(
214: (16)                           (
215: (20)                               "buffer parameter must be a moderngl.Buffer, numpy.ndarray
or bytes instance"
216: (20)                               "(not {})".format(type(buffer))
217: (16)                           )
```

```
218: (12)                       )
219: (8)                    if isinstance(buffer, numpy.ndarray):
220: (12)                       buffer = self.ctx.buffer(buffer.tobytes())
221: (8)                    if isinstance(buffer, bytes):
222: (12)                       buffer = self.ctx.buffer(data=buffer)
223: (8)                    formats = buffer_format.split()
224: (8)                    if len(formats) != len(attribute_names):
225: (12)                       raise VAOError(
226: (16)                           "Format '{}' does not describe attributes
{}".format(buffer_format, attribute_names)
227: (12)                       )
228: (8)                    self._buffers.append(BufferInfo(buffer, buffer_format,
attribute_names))
229: (8)                    self.vertex_count = self._buffers[-1].vertices
230: (8)                    return buffer
231: (4)                def index_buffer(
232: (8)                    self, buffer: Union[moderngl.Buffer, npt.NDArray[Any], bytes],
index_element_size: int = 4
233: (4)                ) -> None:
234: (8)                    """Set the index buffer for this VAO.
235: (8)                    Args:
236: (12)                       buffer: ``moderngl.Buffer``, ``numpy.array`` or ``bytes``
237: (8)                    Keyword Args:
238: (12)                       index_element_size (int): Byte size of each element. 1, 2 or 4
239: (8)                    """
240: (8)                    if type(buffer) not in [moderngl.Buffer, numpy.ndarray, bytes]:
241: (12)                       raise VAOError(
242: (16)                           "buffer parameter must be a moderngl.Buffer, numpy.ndarray or
bytes instance"
243: (12)                       )
244: (8)                    if isinstance(buffer, numpy.ndarray):
245: (12)                       buffer = self.ctx.buffer(buffer.tobytes())
246: (8)                    if isinstance(buffer, bytes):
247: (12)                       buffer = self.ctx.buffer(data=buffer)
248: (8)                    self._index_buffer = buffer
249: (8)                    self._index_element_size = index_element_size
250: (4)                def instance(self, program: moderngl.Program) -> moderngl.VertexArray:
251: (8)                    """Obtain the ``moderngl.VertexArray`` instance for the program.
252: (8)                    The instance is only created once and cached internally.
253: (8)                    Args:
254: (12)                       program (moderngl.Program): The program
255: (8)                    Returns:
256: (12)                       ``moderngl.VertexArray``: instance
257: (8)                    """
258: (8)                    vao = self.vaos.get(program.glo)
259: (8)                    if vao is not None and isinstance(vao, moderngl.VertexArray):
260: (12)                       return vao
261: (8)                    program_attributes = [
262: (12)                       name
263: (12)                       for name, attr in program._members.items()
264: (12)                       if isinstance(attr, moderngl.Attribute) and not
attr.name.startswith("gl_")
265: (8)                    ]
266: (8)                    for attrib_name in program_attributes:
267: (12)                       if not sum(buffer.has_attribute(attrib_name) for buffer in
self._buffers):
268: (16)                           raise VAOError(
269: (20)                               (
270: (24)                                   "VAO {} doesn't have attribute {} for program {}.\n"
271: (24)                                   "Program attributes: {}.\n"
272: (24)                                   "VAO attributes: {}"
273: (20)                               ).format(
274: (24)                                   self.name,
275: (24)                                   attrib_name,
276: (24)                                   program,
277: (24)                                   program_attributes,
278: (24)                                   [attr for buff in self._buffers for attr in
buff.attributes],
279: (20)                               )
```

```
(16)                        )
(8)                 vao_content = []
(8)                 for buffer in self._buffers:
(12)                    content = buffer.content(program_attributes)
(12)                    if content:
(16)                        vao_content.append(content)
(8)                 if program_attributes:
(12)                    raise VAOError(
(16)                        "Did not find a buffer mapping for {}".format([n for n in
program_attributes])
(12)                    )
(8)                 if self._index_buffer:
(12)                    vao = self.ctx.vertex_array(
(16)                        program,
(16)                        vao_content,
(16)                        self._index_buffer,
(16)                        self._index_element_size,
(12)                    )
(8)                 else:
(12)                    vao = self.ctx.vertex_array(program, vao_content)
(8)                 self.vaos[program.glo] = vao
(8)                 return vao
(4)             def release(self, buffer: bool = True) -> None:
(8)                 """Destroy all internally cached vaos and release all buffers.
(8)                 Keyword Args:
(12)                    buffers (bool): also release buffers
(8)                 """
(8)                 for _, vao in self.vaos.items():
(12)                    vao.release()
(8)                 self.vaos = {}
(8)                 if buffer:
(12)                    for buff in self._buffers:
(16)                        buff.buffer.release()
(12)                    if self._index_buffer:
(16)                        self._index_buffer.release()
(8)                 self._buffers = []
(4)             def get_buffer_by_name(self, name: str) -> Optional[BufferInfo]:
(8)                 """Get the BufferInfo associated with a specific attribute name
(8)                 If no buffer is associated with the name `None` will be returned.
(8)                 Args:
(12)                    name (str): Name of the mapped attribute
(8)                 Returns:
(12)                    BufferInfo: BufferInfo instance
(8)                 """
(8)                 for buffer in self._buffers:
(12)                    if name in buffer.attributes:
(16)                        return buffer
(8)                 return None
(0)         class VAOError(Exception):
(4)             pass
```

File 55 - base.py:

```
(0)         import logging
(0)         from collections.abc import Iterable
(0)         from pathlib import Path
(0)         from typing import Any, Optional, Union
(0)         import moderngl
(0)         import moderngl_window as mglw
(0)         from moderngl_window.finders import data, program, scene, texture
(0)         from moderngl_window.finders.base import BaseFilesystemFinder
(0)         from moderngl_window.meta.base import ResourceDescription
(0)         logger = logging.getLogger(__name__)
(0)         class BaseLoader:
(4)             """Base loader class for all resources"""
(4)             kind = "unknown"
(4)             """
```

```
15: (4)                 The kind of resource this loaded supports.
16: (4)                 This can be used when file extensions is not enough
17: (4)                 to decide what loader should be selected.
18: (4)                 """
19: (4)                 file_extensions: list[list[str]] = []
20: (4)                 """
21: (4)                 A list defining the file extensions accepted by this loader.
22: (4)                 Example::
23: (8)                     file_extensions = [
24: (12)                        ['.xyz'],
25: (12)                        ['.xyz', '.gz'],
26: (8)                     ]
27: (4)                 """
28: (4)                 def __init__(self, meta: ResourceDescription) -> None:
29: (8)                     """Initialize loader.
30: (8)                     Loaders take a ResourceDescription instance
31: (8)                     containing all the parameters needed to load and initialize
32: (8)                     this data.
33: (8)                     Args:
34: (12)                        meta (ResourceDescription): The resource to load
35: (8)                     """
36: (8)                     self.meta = meta
37: (8)                     if self.kind is None:
38: (12)                        raise ValueError("Loader {} doesn't have a
kind".format(self.__class__))
39: (4)                 @classmethod
40: (4)                 def supports_file(cls: type["BaseLoader"], meta: ResourceDescription) ->
bool:
41: (8)                     """Check if the loader has a supported file extension.
42: (8)                     What extensions are supported can be defined in the
43: (8)                     :py:attr:`file_extensions` class attribute.
44: (8)                     """
45: (8)                     path = Path(meta.path if meta.path is not None else "")
46: (8)                     for ext in cls.file_extensions:
47: (12)                        if path.suffixes[: len(ext)] == ext:
48: (16)                            return True
49: (8)                     return False
50: (4)                 def load(self) -> Any:
51: (8)                     """Loads a resource.
52: (8)                     When creating a loader this is the only
53: (8)                     method that needs to be implemented.
54: (8)                     Returns:
55: (12)                        The loaded resource
56: (8)                     """
57: (8)                     raise NotImplementedError()
58: (4)                 def find_data(self, path: Optional[Union[str, Path]]) -> Optional[Path]:
59: (8)                     """Find resource using data finders.
60: (8)                     This is mainly a shortcut method to simplify the task.
61: (8)                     Args:
62: (12)                        path: Path to resource
63: (8)                     """
64: (8)                     return self._find(path, data.get_finders())
65: (4)                 def find_program(self, path: Optional[Union[str, Path]]) ->
Optional[Path]:
66: (8)                     """Find resource using program finders.
67: (8)                     This is mainly a shortcut method to simplify the task.
68: (8)                     Args:
69: (12)                        path: Path to resource
70: (8)                     """
71: (8)                     return self._find(path, program.get_finders())
72: (4)                 def find_texture(self, path: Optional[Union[str, Path]]) ->
Optional[Path]:
73: (8)                     """Find resource using texture finders.
74: (8)                     This is mainly a shortcut method to simplify the task.
75: (8)                     Args:
76: (12)                        path: Path to resource
77: (8)                     """
78: (8)                     return self._find(path, texture.get_finders())
79: (4)                 def find_scene(self, path: Optional[Union[str, Path]]) -> Optional[Path]:
```

```
80: (8)                         """Find resource using scene finders.
81: (8)                         This is mainly a shortcut method to simplify the task.
82: (8)                         Args:
83: (12)                            path: Path to resource
84: (8)                         """
85: (8)                         return self._find(path, scene.get_finders())
86: (4)                     def _find(
87: (8)                         self, path: Optional[Union[str, Path]], finders:
Iterable[BaseFilesystemFinder]
88: (4)                     ) -> Optional[Path]:
89: (8)                         """Find the first occurrance of this path in all finders.
90: (8)                         If the incoming path is an absolute path we assume this
91: (8)                         path exist and return it.
92: (8)                         Args:
93: (12)                            path (str): The path to find
94: (8)                         """
95: (8)                         if not path:
96: (12)                            return None
97: (8)                         if isinstance(path, str):
98: (12)                            path = Path(path)
99: (8)                         if path.is_absolute():
100: (12)                           return path
101: (8)                        for finder in finders:
102: (12)                           result = finder.find(path)
103: (12)                           if result:
104: (16)                               return result
105: (8)                        logger.debug("No finder was able to locate: %s", path)
106: (8)                        return None
107: (4)                    @property
108: (4)                    def ctx(self) -> moderngl.Context:
109: (8)                        """moderngl.Context: ModernGL context"""
110: (8)                        return mglw.ctx()

----------------------------------------

File 56 - json.py:

1: (0)              import json
2: (0)              import logging
3: (0)              from typing import Any
4: (0)              from moderngl_window.exceptions import ImproperlyConfigured
5: (0)              from moderngl_window.loaders.base import BaseLoader
6: (0)              logger = logging.getLogger(__name__)
7: (0)              class Loader(BaseLoader):
8: (4)                  kind = "json"
9: (4)                  file_extensions = [
10: (8)                     [".json"],
11: (4)                 ]
12: (4)                 def load(self) -> dict[Any, Any]:
13: (8)                     """Load a file as json
14: (8)                     Returns:
15: (12)                        dict: The json contents
16: (8)                     """
17: (8)                     assert self.meta.path is not None, "the path is empty for this loader"
18: (8)                     self.meta.resolved_path = self.find_data(self.meta.path)
19: (8)                     if not self.meta.resolved_path:
20: (12)                        raise ImproperlyConfigured("Data file '{}' not
found".format(self.meta.path))
21: (8)                     logger.info("Loading: %s", self.meta.path)
22: (8)                     with open(str(self.meta.resolved_path), "r") as fd:
23: (12)                        return json.loads(fd.read())

----------------------------------------

File 57 - text.py:

1: (0)              import logging
2: (0)              from moderngl_window.exceptions import ImproperlyConfigured
3: (0)              from moderngl_window.loaders.base import BaseLoader
```

```
4: (0)              logger = logging.getLogger(__name__)
5: (0)              class Loader(BaseLoader):
6: (4)                  kind = "text"
7: (4)                  file_extensions = [
8: (8)                      [".txt"],
9: (4)                  ]
10: (4)                 def load(self) -> str:
11: (8)                     """Load a file in text mode.
12: (8)                     Returns:
13: (12)                        str: The string contents of the file
14: (8)                     """
15: (8)                     assert self.meta.path is not None, "the path is empty for this loader"
16: (8)                     self.meta.resolved_path = self.find_data(self.meta.path)
17: (8)                     if not self.meta.resolved_path:
18: (12)                        raise ImproperlyConfigured("Data file '{}' not
found".format(self.meta.path))
19: (8)                     logger.info("Loading: %s", self.meta.path)
20: (8)                     with open(str(self.meta.resolved_path), "r") as fd:
21: (12)                        return fd.read()
```

File 58 - cube.py:

```
1: (0)              from collections import namedtuple
2: (0)              from typing import Any, Optional
3: (0)              import moderngl
4: (0)              from moderngl_window.exceptions import ImproperlyConfigured
5: (0)              from moderngl_window.loaders.texture.pillow import PillowLoader, image_data
6: (0)              from moderngl_window.meta.base import ResourceDescription
7: (0)              from moderngl_window.meta.texture import TextureDescription
8: (0)              FaceInfo = namedtuple("FaceInfo", ["width", "height", "data", "components"])
9: (0)              class Loader(PillowLoader):
10: (4)                 kind = "cube"
11: (4)                 meta: TextureDescription
12: (4)                 def __init__(self, meta: ResourceDescription):
13: (8)                     super().__init__(meta)
14: (4)                 def load(self) -> moderngl.TextureCube:
15: (8)                     """Load a texture cube as described by the supplied
``TextureDescription```
16: (8)                     Returns:
17: (12)                        moderngl.TextureCube: The TextureArray instance
18: (8)                     """
19: (8)                     pos_x = self._load_face(self.meta.pos_x, face_name="pos_x")
20: (8)                     pos_y = self._load_face(self.meta.pos_y, face_name="pos_y")
21: (8)                     pos_z = self._load_face(self.meta.pos_z, face_name="pos_z")
22: (8)                     neg_x = self._load_face(self.meta.neg_x, face_name="neg_x")
23: (8)                     neg_y = self._load_face(self.meta.neg_y, face_name="neg_y")
24: (8)                     neg_z = self._load_face(self.meta.neg_z, face_name="neg_z")
25: (8)                     self._validate([pos_x, pos_y, pos_z, neg_x, neg_y, neg_z])
26: (8)                     texture = self.ctx.texture_cube(
27: (12)                        (pos_x.width, pos_x.height),
28: (12)                        pos_x.components,
29: (12)                        pos_x.data + neg_x.data + pos_y.data + neg_y.data + pos_z.data +
neg_z.data,
30: (8)                     )
31: (8)                     texture.extra = {"meta": self.meta}
32: (8)                     if self.meta.mipmap_levels is not None:
33: (12)                        self.meta.mipmap = True
34: (8)                     if self.meta.mipmap:
35: (12)                        if isinstance(self.meta.mipmap_levels, tuple):
36: (16)                            texture.build_mipmaps(*self.meta.mipmap_levels)
37: (12)                        else:
38: (16)                            texture.build_mipmaps()
39: (12)                        if self.meta.anisotropy:
40: (16)                            texture.anisotropy = self.meta.anisotropy
41: (8)                     return texture
42: (4)                 def _load_face(self, path: Optional[str], face_name: Optional[str] = None)
-> FaceInfo:
```

```
43: (8)                     """Obtain raw byte data for a face
44: (8)                     Returns:
45: (12)                        tuple[int, bytes]: number of components, byte data
46: (8)                     """
47: (8)                     if not path:
48: (12)                        raise ImproperlyConfigured(f"{face_name} texture face not supplied")
49: (8)                     image = self._load_texture(path)
50: (8)                     components, data = image_data(image)
51: (8)                     return FaceInfo(width=image.size[0], height=image.size[1], data=data, components=components)
52: (4)                 def _validate(self, faces: list[FaceInfo]) -> Any:
53: (8)                     """Validates each face ensuring components and size it the same"""
54: (8)                     components = faces[0].components
55: (8)                     data_size = len(faces[0].data)
56: (8)                     for face in faces:
57: (12)                        if face.components != components:
58: (16)                            raise ImproperlyConfigured(
59: (20)                                "Cubemap face textures have different number of components"
60: (16)                            )
61: (12)                        if len(face.data) != data_size:
62: (16)                            raise ImproperlyConfigured("Cubemap face textures must all have the same size")
63: (8)                     return components
```

File 59 - icon.py:

```
1: (0)                from pathlib import Path
2: (0)                from moderngl_window.finders import texture
3: (0)                from moderngl_window.loaders.base import BaseLoader
4: (0)                from moderngl_window.meta.base import ResourceDescription
5: (0)                from moderngl_window.meta.texture import TextureDescription
6: (0)                class IconLoader(BaseLoader):
7: (4)                    kind = "icon"
8: (4)                    meta: TextureDescription
9: (4)                    def __init__(self, meta: ResourceDescription) -> None:
10: (8)                       super().__init__(meta)
11: (4)                   def find_icon(self) -> Path:
12: (8)                       """Find resource using texture finders.
13: (8)                       This is mainly a shortcut method to simplify the task.
14: (8)                       Args:
15: (12)                          path: Path to resource
16: (8)                       """
17: (8)                       abs_path = self._find(self.meta.path, texture.get_finders())
18: (8)                       if abs_path is None:
19: (12)                          raise ValueError("Could not find the icon specified. {}".format(self.meta.path))
20: (8)                       return abs_path
```

File 60 - base.py:

```
1: (0)                from pathlib import Path
2: (0)                from typing import Any, Optional
3: (0)                class ResourceDescription:
4: (4)                    """Description of any resource.
5: (4)                    Resource descriptions are required to load a resource.
6: (4)                    This class can be extended to add more specific properties.
7: (4)                    """
8: (4)                    default_kind = ""  # The default kind of loader
9: (4)                    """str: The default kind for this resource type"""
10: (4)                   resource_type = ""  # What resource type is described
11: (4)                   """str: A unique identifier for the resource type"""
12: (4)                   def __init__(self, **kwargs: Any):
13: (8)                       """Initialize a resource description
```

```
(8)                     Args:
(12)                        **kwargs: Attributes describing the resource to load
(8)                     """
(8)                     self._kwargs = kwargs
(4)                 @property
(4)                 def path(self) -> Optional[str]:
(8)                     """str: The path to a resource when a single file is specified"""
(8)                     return self._kwargs.get("path")
(4)                 @property
(4)                 def label(self) -> Optional[str]:
(8)                     """str: optional name for the resource
(8)                     Assigning a label is not mandatory but can help
(8)                     when aliasing resources. Some prefer to preload
(8)                     all needed resources and fetch them later by the label.
(8)                     This can he a lot less chaotic in larger applications.
(8)                     """
(8)                     return self._kwargs.get("label")
(4)                 @property
(4)                 def kind(self) -> str:
(8)                     """str: default resource kind.
(8)                     The resource ``kind`` is directly matched
(8)                     with the ``kind`` in loader classes.
(8)                     This property also supports assignment
(8)                     and is useful if the ``kind`` is detected
(8)                     based in the the attribute values.
(8)                     .. code:: python
(12)                        description.kind = 'something'
(8)                     """
(8)                     k = self._kwargs.get("kind")
(8)                     if k is None:
(12)                        k = self.default_kind
(8)                     return k
(4)                 @kind.setter
(4)                 def kind(self, value: str) -> None:
(8)                     self._kwargs["kind"] = value
(4)                 @property
(4)                 def loader_cls(self) -> Optional[type]:
(8)                     """type: The loader class for this resource.
(8)                     This property is assigned to during the loading
(8)                     stage were a loader class is assigned based on
(8)                     the `kind`.
(8)                     """
(8)                     return self._kwargs.get("loader_cls")
(4)                 @loader_cls.setter
(4)                 def loader_cls(self, value: type) -> None:
(8)                     self._kwargs["loader_cls"] = value
(4)                 @property
(4)                 def resolved_path(self) -> Optional[Path]:
(8)                     """pathlib.Path: The resolved path by a finder.
(8)                     The absolute path to the resource can optionally
(8)                     be assigned by a loader class.
(8)                     """
(8)                     return self._kwargs.get("resolved_path")
(4)                 @resolved_path.setter
(4)                 def resolved_path(self, value: Path) -> None:
(8)                     self._kwargs["resolved_path"] = value
(4)                 @property
(4)                 def attrs(self) -> dict[str, Any]:
(8)                     """dict: All keywords arguments passed to the resource"""
(8)                     return self._kwargs
(4)                 def __str__(self) -> str:
(8)                     return str(self._kwargs)
(4)                 def __repr__(self) -> str:
(8)                     return str(self)
```

File 61 - data.py:

```
1: (0)              from typing import Any, Optional
2: (0)              from moderngl_window.meta.base import ResourceDescription
3: (0)              class DataDescription(ResourceDescription):
4: (4)                  """Describes data file to load.
5: (4)                  This is a generic resource description type
6: (4)                  for loading resources that are not textures, programs and scenes.
7: (4)                  That loaded class is used depends on the ``kind`` or the file extension.
8: (4)                  Currently used to load:
9: (4)                  - text files
10: (4)                 - json files
11: (4)                 - binary files
12: (4)                 .. code:: python
13: (8)                     DataDescription(path='data/text.txt')
14: (8)                     DataDescription(path='data/data.json')
15: (8)                     DataDescription(path='data/data.bin', kind='binary')
16: (4)                 """
17: (4)                 default_kind: str = ""
18: (4)                 resource_type = "data"
19: (4)                 def __init__(
20: (8)                     self, path: Optional[str] = None, kind: Optional[str] = None,
**kwargs: Any
21: (4)                 ) -> None:
22: (8)                     """Initialize the resource description.
23: (8)                     Keyword Args:
24: (12)                        path (str): Relative path to the resource
25: (12)                        kind (str): The resource kind deciding loader class
26: (12)                        **kwargs: Additional custom attributes
27: (8)                     """
28: (8)                     kwargs.update({"path": path, "kind": kind})
29: (8)                     super().__init__(**kwargs)
```

File 62 - imgui.py:

```
1: (0)              import ctypes
2: (0)              import imgui
3: (0)              import moderngl
4: (0)              from imgui.integrations import compute_fb_scale
5: (0)              from imgui.integrations.base import BaseOpenGLRenderer
6: (0)              class ModernglWindowMixin:
7: (4)                  def resize(self, width: int, height: int):
8: (8)                      self.io.display_size = self.wnd.size
9: (8)                      self.io.display_fb_scale = compute_fb_scale(self.wnd.size,
self.wnd.buffer_size)
10: (4)                 def key_event(self, key, action, modifiers):
11: (8)                     keys = self.wnd.keys
12: (8)                     if action == keys.ACTION_PRESS:
13: (12)                        if key in self.REVERSE_KEY_MAP:
14: (16)                            self.io.keys_down[self.REVERSE_KEY_MAP[key]] = True
15: (8)                     else:
16: (12)                        if key in self.REVERSE_KEY_MAP:
17: (16)                            self.io.keys_down[self.REVERSE_KEY_MAP[key]] = False
18: (4)                 def _mouse_pos_viewport(self, x, y):
19: (8)                     """Make sure mouse coordinates are correct with black borders"""
20: (8)                     return (
21: (12)                        int(x - (self.wnd.width - self.wnd.viewport_width /
self.wnd.pixel_ratio) / 2),
22: (12)                        int(y - (self.wnd.height - self.wnd.viewport_height /
self.wnd.pixel_ratio) / 2),
23: (8)                     )
24: (4)                 def mouse_position_event(self, x, y, dx, dy):
25: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
26: (4)                 def mouse_drag_event(self, x, y, dx, dy):
27: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
28: (8)                     if self.wnd.mouse_states.left:
29: (12)                        self.io.mouse_down[0] = 1
30: (8)                     if self.wnd.mouse_states.middle:
31: (12)                        self.io.mouse_down[2] = 1
```

```
32: (8)                     if self.wnd.mouse_states.right:
33: (12)                        self.io.mouse_down[1] = 1
34: (4)                 def mouse_scroll_event(self, x_offset, y_offset):
35: (8)                     self.io.mouse_wheel = y_offset
36: (4)                 def mouse_press_event(self, x, y, button):
37: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
38: (8)                     if button == self.wnd.mouse.left:
39: (12)                        self.io.mouse_down[0] = 1
40: (8)                     if button == self.wnd.mouse.middle:
41: (12)                        self.io.mouse_down[2] = 1
42: (8)                     if button == self.wnd.mouse.right:
43: (12)                        self.io.mouse_down[1] = 1
44: (4)                 def mouse_release_event(self, x: int, y: int, button: int):
45: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
46: (8)                     if button == self.wnd.mouse.left:
47: (12)                        self.io.mouse_down[0] = 0
48: (8)                     if button == self.wnd.mouse.middle:
49: (12)                        self.io.mouse_down[2] = 0
50: (8)                     if button == self.wnd.mouse.right:
51: (12)                        self.io.mouse_down[1] = 0
52: (4)                 def unicode_char_entered(self, char):
53: (8)                     io = imgui.get_io()
54: (8)                     io.add_input_character(ord(char))
55: (0)             class ModernGLRenderer(BaseOpenGLRenderer):
56: (4)                 VERTEX_SHADER_SRC = """
57: (8)                     uniform mat4 ProjMtx;
58: (8)                     in vec2 Position;
59: (8)                     in vec2 UV;
60: (8)                     in vec4 Color;
61: (8)                     out vec2 Frag_UV;
62: (8)                     out vec4 Frag_Color;
63: (8)                     void main() {
64: (12)                        Frag_UV = UV;
65: (12)                        Frag_Color = Color;
66: (12)                        gl_Position = ProjMtx * vec4(Position.xy, 0, 1);
67: (8)                     }
68: (4)                 """
69: (4)                 FRAGMENT_SHADER_SRC = """
70: (8)                     uniform sampler2D Texture;
71: (8)                     in vec2 Frag_UV;
72: (8)                     in vec4 Frag_Color;
73: (8)                     out vec4 Out_Color;
74: (8)                     void main() {
75: (12)                        Out_Color = (Frag_Color * texture(Texture, Frag_UV.st));
76: (8)                     }
77: (4)                 """
78: (4)                 def __init__(self, *args, **kwargs):
79: (8)                     self._prog = None
80: (8)                     self._fbo = None
81: (8)                     self._font_texture = None
82: (8)                     self._vertex_buffer = None
83: (8)                     self._index_buffer = None
84: (8)                     self._vao = None
85: (8)                     self._textures = {}
86: (8)                     self.wnd = kwargs.get("wnd")
87: (8)                     self.ctx = self.wnd.ctx if self.wnd and self.wnd.ctx else
kwargs.get("ctx")
88: (8)                     if not self.ctx:
89: (12)                        raise ValueError("Missing moderngl context")
90: (8)                     super().__init__()
91: (8)                     if hasattr(self, "wnd") and self.wnd:
92: (12)                        self.resize(*self.wnd.buffer_size)
93: (8)                     elif "display_size" in kwargs:
94: (12)                        self.io.display_size = kwargs.get("display_size")
95: (4)                 def register_texture(self, texture: moderngl.Texture):
96: (8)                     """Make the imgui renderer aware of the texture"""
97: (8)                     self._textures[texture.glo] = texture
98: (4)                 def remove_texture(self, texture: moderngl.Texture):
99: (8)                     """Remove the texture from the imgui renderer"""
```

```
        del self._textures[texture.glo]
    def refresh_font_texture(self):
        width, height, pixels = self.io.fonts.get_tex_data_as_rgba32()
        if self._font_texture:
            self.remove_texture(self._font_texture)
            self._font_texture.release()
        self._font_texture = self.ctx.texture((width, height), 4, data=pixels)
        self.register_texture(self._font_texture)
        self.io.fonts.texture_id = self._font_texture.glo
        self.io.fonts.clear_tex_data()
    def _create_device_objects(self):
        self._prog = self.ctx.program(
            vertex_shader=self.VERTEX_SHADER_SRC,
            fragment_shader=self.FRAGMENT_SHADER_SRC,
        )
        self.projMat = self._prog["ProjMtx"]
        self._prog["Texture"].value = 0
        self._vertex_buffer = self.ctx.buffer(reserve=imgui.VERTEX_SIZE * 65536)
        self._index_buffer = self.ctx.buffer(reserve=imgui.INDEX_SIZE * 65536)
        self._vao = self.ctx.vertex_array(
            self._prog,
            [(self._vertex_buffer, "2f 2f 4f1", "Position", "UV", "Color")],
            index_buffer=self._index_buffer,
            index_element_size=imgui.INDEX_SIZE,
        )
    def render(self, draw_data):
        io = self.io
        display_width, display_height = io.display_size
        fb_width = int(display_width * io.display_fb_scale[0])
        fb_height = int(display_height * io.display_fb_scale[1])
        if fb_width == 0 or fb_height == 0:
            return
        self.projMat.value = (
            2.0 / display_width,
            0.0,
            0.0,
            0.0,
            0.0,
            2.0 / -display_height,
            0.0,
            0.0,
            0.0,
            0.0,
            -1.0,
            0.0,
            -1.0,
            1.0,
            0.0,
            1.0,
        )
        draw_data.scale_clip_rects(*io.display_fb_scale)
        self.ctx.enable_only(moderngl.BLEND)
        self.ctx.blend_equation = moderngl.FUNC_ADD
        self.ctx.blend_func = moderngl.SRC_ALPHA, moderngl.ONE_MINUS_SRC_ALPHA
        self._font_texture.use()
        for commands in draw_data.commands_lists:
            vtx_type = ctypes.c_byte * commands.vtx_buffer_size * imgui.VERTEX_SIZE
            idx_type = ctypes.c_byte * commands.idx_buffer_size * imgui.INDEX_SIZE
            vtx_arr = (vtx_type).from_address(commands.vtx_buffer_data)
            idx_arr = (idx_type).from_address(commands.idx_buffer_data)
            self._vertex_buffer.write(vtx_arr)
            self._index_buffer.write(idx_arr)
            idx_pos = 0
            for command in commands.commands:
                texture = self._textures.get(command.texture_id)
                if texture is None:
```

```
166: (20)                               raise ValueError(
167: (24)                                   (
168: (28)                                       "Texture {} is not registered. Please add to renderer using "
169: (28)                                       "register_texture(..). "
170: (28)                                       "Current textures: {}".format(command.texture_id, list(self._textures))
171: (24)                                   )
172: (20)                               )
173: (16)                           texture.use(0)
174: (16)                           x, y, z, w = command.clip_rect
175: (16)                           self.ctx.scissor = int(x), int(fb_height - w), int(z - x), int(w - y)
176: (16)                           self._vao.render(moderngl.TRIANGLES, vertices=command.elem_count, first=idx_pos)
177: (16)                           idx_pos += command.elem_count
178: (8)                    self.ctx.scissor = None
179: (4)                def _invalidate_device_objects(self):
180: (8)                    if self._font_texture:
181: (12)                       self._font_texture.release()
182: (8)                    if self._vertex_buffer:
183: (12)                       self._vertex_buffer.release()
184: (8)                    if self._index_buffer:
185: (12)                       self._index_buffer.release()
186: (8)                    if self._vao:
187: (12)                       self._vao.release()
188: (8)                    if self._prog:
189: (12)                       self._prog.release()
190: (8)                    self.io.fonts.texture_id = 0
191: (8)                    self._font_texture = None
192: (0)            class ModernglWindowRenderer(ModernGLRenderer, ModernglWindowMixin):
193: (4)                def __init__(self, window):
194: (8)                    super().__init__(wnd=window)
195: (8)                    self.wnd = window
196: (8)                    self._init_key_maps()
197: (8)                    self.io.display_size = self.wnd.size
198: (8)                    self.io.display_fb_scale = self.wnd.pixel_ratio, self.wnd.pixel_ratio
199: (4)                def _init_key_maps(self):
200: (8)                    keys = self.wnd.keys
201: (8)                    self.REVERSE_KEY_MAP = {
202: (12)                       keys.TAB: imgui.KEY_TAB,
203: (12)                       keys.LEFT: imgui.KEY_LEFT_ARROW,
204: (12)                       keys.RIGHT: imgui.KEY_RIGHT_ARROW,
205: (12)                       keys.UP: imgui.KEY_UP_ARROW,
206: (12)                       keys.DOWN: imgui.KEY_DOWN_ARROW,
207: (12)                       keys.PAGE_UP: imgui.KEY_PAGE_UP,
208: (12)                       keys.PAGE_DOWN: imgui.KEY_PAGE_DOWN,
209: (12)                       keys.HOME: imgui.KEY_HOME,
210: (12)                       keys.END: imgui.KEY_END,
211: (12)                       keys.DELETE: imgui.KEY_DELETE,
212: (12)                       keys.SPACE: imgui.KEY_SPACE,
213: (12)                       keys.BACKSPACE: imgui.KEY_BACKSPACE,
214: (12)                       keys.ENTER: imgui.KEY_ENTER,
215: (12)                       keys.ESCAPE: imgui.KEY_ESCAPE,
216: (12)                       keys.A: imgui.KEY_A,
217: (12)                       keys.C: imgui.KEY_C,
218: (12)                       keys.V: imgui.KEY_V,
219: (12)                       keys.X: imgui.KEY_X,
220: (12)                       keys.Y: imgui.KEY_Y,
221: (12)                       keys.Z: imgui.KEY_Z,
222: (8)                    }
223: (8)                    for value in self.REVERSE_KEY_MAP.values():
224: (12)                       self.io.key_map[value] = value
```

File 63 - gltf2.py:

```
1: (0)            from __future__ import annotations
```

```
import base64
import io
import json
import logging
import struct
from collections import namedtuple
from pathlib import Path
from typing import Any, Optional, Union
import glm
import moderngl
import numpy
import numpy.typing as npt
from PIL import Image
import moderngl_window
from moderngl_window.exceptions import ImproperlyConfigured
from moderngl_window.loaders.base import BaseLoader
from moderngl_window.loaders.texture import t2d
from moderngl_window.meta import SceneDescription, TextureDescription
from moderngl_window.opengl.vao import VAO
from moderngl_window.scene import Material, MaterialTexture, Mesh, Node, Scene
logger = logging.getLogger(__name__)
GLTF_MAGIC_HEADER = b"glTF"
REPEAT = 10497
CLAMP_TO_EDGE = 33071
MIRRORED_REPEAT = 33648
NP_COMPONENT_DTYPE = {
    5121: numpy.uint8,  # GL_UNSIGNED_BYTE
    5123: numpy.uint16,  # GL_UNSIGNED_SHORT
    5125: numpy.uint32,  # GL_UNSIGNED_INT
    5126: numpy.float32,  # GL_FLOAT
}
ComponentType = namedtuple("ComponentType", ["name", "value", "size"])
COMPONENT_TYPE = {
    5120: ComponentType("BYTE", 5120, 1),
    5121: ComponentType("UNSIGNED_BYTE", 5121, 1),
    5122: ComponentType("SHORT", 5122, 2),
    5123: ComponentType("UNSIGNED_SHORT", 5123, 2),
    5125: ComponentType("UNSIGNED_INT", 5125, 4),
    5126: ComponentType("FLOAT", 5126, 4),
}
DTYPE_BUFFER_TYPE = {
    numpy.uint8: "u1",  # GL_UNSIGNED_BYTE
    numpy.uint16: "u2",  # GL_UNSIGNED_SHORT
    numpy.uint32: "u4",  # GL_UNSIGNED_INT
    numpy.float32: "f4",  # GL_FLOAT
}
ACCESSOR_TYPE = {
    "SCALAR": 1,
    "VEC2": 2,
    "VEC3": 3,
    "VEC4": 4,
}
class Loader(BaseLoader):
    """Loader for GLTF 2.0 files"""
    kind = "gltf"
    file_extensions = [
        [".gltf"],
        [".glb"],
    ]
    supported_extensions: list[str] = []
    meta: SceneDescription
    def __init__(self, meta: SceneDescription):
        """Initialize loading GLTF 2 scene.
        Supported formats:
        - gltf json format with external resources
        - gltf embedded buffers
        - glb Binary format
        """
        super().__init__(meta)
```

```
(8)           self.scenes: list[Scene] = []
(8)           self.nodes: list[Node] = []
(8)           self.meshes: list[list[Mesh]] = []
(8)           self.materials: list[Material] = []
(8)           self.images: list[moderngl.Texture] = []
(8)           self.samplers: list[moderngl.Sampler] = []
(8)           self.textures: list[MaterialTexture] = []
(8)           self.path: Optional[Path] = None
(8)           self.scene: Scene
(8)           self.gltf: GLTFMeta
(4)       def load(self) -> Scene:
(8)           """Load a GLTF 2 scene including referenced textures.
(8)           Returns:
(12)              Scene: The scene instance
(8)           """
(8)           assert self.meta.path is not None, "The path to this resource is empty"
(8)           self.path = self.find_scene(self.meta.path)
(8)           if not self.path:
(12)              raise ImproperlyConfigured("Scene '{}' not found".format(self.meta.path))
(8)           self.scene = Scene(str(self.path))
(8)           if self.path.suffix == ".gltf":
(12)              self.load_gltf()
(8)           if self.path.suffix == ".glb":
(12)              self.load_glb()
(8)           assert self.gltf is not None, "There is a problem with your file, could not load gltf"
(8)           self.gltf.check_version()
(8)           self.gltf.check_extensions(self.supported_extensions)
(8)           self.load_images()
(8)           self.load_samplers()
(8)           self.load_textures()
(8)           self.load_materials()
(8)           self.load_meshes()
(8)           self.load_nodes()
(8)           self.scene.calc_scene_bbox()
(8)           self.scene.prepare()
(8)           return self.scene
(4)       def load_gltf(self) -> None:
(8)           """Loads a gltf json file parsing its contents"""
(8)           with open(str(self.path)) as fd:
(12)              self.gltf = GLTFMeta(str(self.path), json.load(fd), self.meta)
(4)       def load_glb(self) -> None:
(8)           """Loads a binary gltf file parsing its contents"""
(8)           with open(str(self.path), "rb") as fd:
(12)              magic = fd.read(4)
(12)              if magic != GLTF_MAGIC_HEADER:
(16)                  raise ValueError(
(20)                      "{} has incorrect header {!r} != {!r}".format(
(24)                          self.path, magic, GLTF_MAGIC_HEADER
(20)                      )
(16)                  )
(12)              version = struct.unpack("<I", fd.read(4))[0]
(12)              if version != 2:
(16)                  raise ValueError(f"{self.path} has unsupported version {version}")
(12)              _ = struct.unpack("<I", fd.read(4))[0]  # noqa
(12)              chunk_0_length = struct.unpack("<I", fd.read(4))[0]
(12)              chunk_0_type = fd.read(4)
(12)              if chunk_0_type != b"JSON":
(16)                  raise ValueError(
(20)                      "Expected JSON chunk, not {!r} in file {}".format(chunk_0_type, self.path)
(16)                  )
(12)              json_meta = fd.read(chunk_0_length).decode()
(12)              chunk_1_length = struct.unpack("<I", fd.read(4))[0]
(12)              chunk_1_type = fd.read(4)
(12)              if chunk_1_type != b"BIN\x00":
```

```
135: (16)                               raise ValueError(
136: (20)                                   "Expected BIN chunk, not {!r} in file
{}".format(chunk_1_type, self.path)
137: (16)                               )
138: (12)                           self.gltf = GLTFMeta(
139: (16)                               str(self.path),
140: (16)                               json.loads(json_meta),
141: (16)                               self.meta,
142: (16)                               binary_buffer=fd.read(chunk_1_length),
143: (12)                           )
144: (4)                    def load_images(self) -> None:
145: (8)                        """Load images referenced in gltf metadata"""
146: (8)                        for image in self.gltf.images:
147: (12)                           self.images.append(image.load(self.path.parent))
148: (4)                    def load_samplers(self) -> None:
149: (8)                        """Load samplers referenced in gltf metadata"""
150: (8)                        for sampler in self.gltf.samplers:
151: (12)                           if sampler.minFilter is sampler.magFilter is None:
152: (16)                               self.samplers.append(
153: (20)                                   self.ctx.sampler(
154: (24)                                       filter=(moderngl.LINEAR_MIPMAP_LINEAR,
moderngl.LINEAR),
155: (24)                                       repeat_x=False,
156: (24)                                       repeat_y=False,
157: (24)                                       anisotropy=16.0,
158: (20)                                   )
159: (16)                               )
160: (12)                           else:
161: (16)                               self.samplers.append(
162: (20)                                   self.ctx.sampler(
163: (24)                                       filter=(sampler.minFilter, sampler.magFilter),
164: (24)                                       repeat_x=sampler.wrapS in [REPEAT, MIRRORED_REPEAT],
165: (24)                                       repeat_y=sampler.wrapT in [REPEAT, MIRRORED_REPEAT],
166: (24)                                       anisotropy=16.0,
167: (20)                                   )
168: (16)                               )
169: (4)                    def load_textures(self) -> None:
170: (8)                        """Load textures referenced in gltf metadata"""
171: (8)                        for texture_meta in self.gltf.textures:
172: (12)                           texture = MaterialTexture()
173: (12)                           if texture_meta.source is not None:
174: (16)                               texture.texture = self.images[texture_meta.source]
175: (12)                           if texture_meta.sampler is not None:
176: (16)                               texture.sampler = self.samplers[texture_meta.sampler]
177: (12)                           self.textures.append(texture)
178: (4)                    def load_meshes(self) -> None:
179: (8)                        """Load meshes referenced in gltf metadata"""
180: (8)                        for meta_mesh in self.gltf.meshes:
181: (12)                           meshes = meta_mesh.load(self.materials)
182: (12)                           self.meshes.append(meshes)
183: (12)                           for mesh in meshes:
184: (16)                               self.scene.meshes.append(mesh)
185: (4)                    def load_materials(self) -> None:
186: (8)                        """Load materials referenced in gltf metadata"""
187: (8)                        for meta_mat in self.gltf.materials:
188: (12)                           mat = Material(meta_mat.name)
189: (12)                           mat.color = meta_mat.baseColorFactor or (1.0, 1.0, 1.0, 1.0)
190: (12)                           mat.double_sided = meta_mat.doubleSided
191: (12)                           if meta_mat.baseColorTexture is not None:
192: (16)                               mat.mat_texture =
self.textures[meta_mat.baseColorTexture["index"]]
193: (12)                           self.materials.append(mat)
194: (12)                           self.scene.materials.append(mat)
195: (4)                    def load_nodes(self) -> None:
196: (8)                        """Load nodes referenced in gltf metadata"""
197: (8)                        for node_id in self.gltf.scenes[0].nodes:
198: (12)                           node = self.load_node(self.gltf.nodes[node_id])
199: (12)                           self.scene.root_nodes.append(node)
200: (4)                    def load_node(self, meta: GLTFNode, parent: Optional[Node] = None) ->
```

```
Node:
201: (8)                    """Load a single node"""
202: (8)                    node = Node(name=meta.name, matrix=meta.matrix)
203: (8)                    self.scene.nodes.append(node)
204: (8)                    if meta.mesh is not None:
205: (12)                       if len(self.meshes[meta.mesh]) == 1:
206: (16)                           node.mesh = self.meshes[meta.mesh][0]
207: (12)                       elif len(self.meshes[meta.mesh]) > 1:
208: (16)                           for mesh in self.meshes[meta.mesh]:
209: (20)                               node.add_child(Node(mesh=mesh))
210: (8)                    if meta.camera is not None:
211: (12)                       node.camera = self.gltf.cameras[meta.camera]
212: (8)                    if parent:
213: (12)                       parent.add_child(node)
214: (8)                    if meta.has_children:
215: (12)                       for node_id in meta.children:
216: (16)                           self.load_node(self.gltf.nodes[node_id], parent=node)
217: (8)                    return node
218: (0)            class GLTFMeta:
219: (4)                """Container for gltf metadata"""
220: (4)                def __init__(
221: (8)                    self,
222: (8)                    path: Union[Path, str],
223: (8)                    data: dict[Any, Any],
224: (8)                    meta: SceneDescription,
225: (8)                    binary_buffer: Optional[bytes] = None,
226: (4)                ) -> None:
227: (8)                    """
228: (8)                    :param file: GLTF file name loaded
229: (8)                    :param data: Metadata (json loaded)
230: (8)                    :param binary_buffer: Binary buffer when loading glb files
231: (8)                    """
232: (8)                    self.path = Path(path) if isinstance(path, str) else path
233: (8)                    self.data = data
234: (8)                    self.meta = meta
235: (8)                    self.asset = GLTFAsset(data["asset"])
236: (8)                    self.materials = (
237: (12)                       [GLTFMaterial(m) for m in data["materials"]] if
data.get("materials") else []
238: (8)                    )
239: (8)                    self.images = [GLTFImage(i) for i in data["images"]] if
data.get("images") else []
240: (8)                    self.samplers = [GLTFSampler(s) for s in data["samplers"]] if
data.get("samplers") else []
241: (8)                    self.textures = [GLTFTexture(t) for t in data["textures"]] if
data.get("textures") else []
242: (8)                    self.scenes = [GLTFScene(s) for s in data["scenes"]] if
data.get("scenes") else []
243: (8)                    self.nodes = [GLTFNode(n) for n in data["nodes"]] if data.get("nodes")
else []
244: (8)                    self.meshes = [GLTFMesh(m, self.meta) for m in data["meshes"]] if
data.get("meshes") else []
245: (8)                    self.cameras = [GLTFCamera(c) for c in data["cameras"]] if
data.get("cameras") else []
246: (8)                    self.buffer_views = (
247: (12)                       [GLTFBufferView(i, v) for i, v in enumerate(data["bufferViews"])]
248: (12)                       if data.get("bufferViews")
249: (12)                       else []
250: (8)                    )
251: (8)                    self.buffers = (
252: (12)                       [GLTFBuffer(i, b, self.path.parent) for i, b in
enumerate(data["buffers"])]
253: (12)                       if data.get("buffers")
254: (12)                       else []
255: (8)                    )
256: (8)                    self.accessors = (
257: (12)                       [GLTFAccessor(i, a) for i, a in enumerate(data["accessors"])]
258: (12)                       if data.get("accessors")
259: (12)                       else []
```

```
        )
        if binary_buffer:
            self.buffers[0].data = binary_buffer
        self._link_data()
        self.buffers_exist()
        self.images_exist()
    def _link_data(self) -> None:
        """Add references"""
        for acc in self.accessors:
            acc.bufferView = self.buffer_views[acc.bufferViewId]
        for buffer_view in self.buffer_views:
            buffer_view.buffer = self.buffers[buffer_view.bufferId]
        for mesh in self.meshes:
            for primitive in mesh.primitives:
                if primitive.indices is not None:
                    primitive.accessor = self.accessors[primitive.indices]
                for name, value in primitive.attributes.items():
                    primitive.attributes[name] = self.accessors[value]
        for image in self.images:
            if image.bufferViewId is not None:
                image.bufferView = self.buffer_views[image.bufferViewId]
    @property
    def version(self) -> str:
        return self.asset.version
    def check_version(self, required: str = "2.0") -> None:
        if not self.version == required:
            msg = (
                f"GLTF Format version is not 2.0. Version states '{self.version}' "
                f"in file {self.path}"
            )
            raise ValueError(msg)
    def check_extensions(self, supported: list[str]) -> None:
        """
        "extensionsRequired": ["KHR_draco_mesh_compression"],
        "extensionsUsed": ["KHR_draco_mesh_compression"]
        """
        extReq = self.data.get("extensionsRequired")
        if extReq is not None:
            for ext in extReq:
                if ext not in supported:
                    raise ValueError(f"Extension {ext} not supported")
        extUse = self.data.get("extensionsUsed")
        if extUse is not None:
            for ext in extUse:
                if ext not in supported:
                    raise ValueError("Extension {ext} not supported")
    def buffers_exist(self) -> None:
        """Checks if the bin files referenced exist"""
        for buff in self.buffers:
            if not buff.is_separate_file:
                continue
            path = self.path.parent / buff.uri
            if not path.exists():
                raise FileNotFoundError(
                    "Buffer {} referenced in {} not found".format(path, self.path)
                )
    def images_exist(self) -> None:
        """checks if the images references in textures exist"""
        pass
class GLTFAsset:
    """Asset Information"""
    def __init__(self, data: dict[str, Any]):
        self.version = data.get("version")
        self.generator = data.get("generator")
        self.copyright = data.get("copyright")
class GLTFMesh:
    class Primitives:
```

```
(8)              mode: int | None
(8)              accessor: GLTFAccessor | None
(8)              def __init__(self, data: dict[str, Any]):
(12)                 self.attributes: dict[str, Any] = data.get("attributes")
(12)                 self.indices = data.get("indices")
(12)                 self.mode = data.get("mode")
(12)                 self.material = data.get("material")
(12)                 self.accessor = None
(4)          def __init__(self, data: dict[str, Any], meta: SceneDescription):
(8)              self.meta = meta
(8)              self.name = data.get("name", "")
(8)              self.primitives = [GLTFMesh.Primitives(p) for p in data["primitives"]]
(4)          def load(self, materials: list[Material]) -> list[Mesh]:
(8)              name_map = {
(12)                 "POSITION": self.meta.attr_names.POSITION,
(12)                 "NORMAL": self.meta.attr_names.NORMAL,
(12)                 "TEXCOORD_0": self.meta.attr_names.TEXCOORD_0,
(12)                 "TANGENT": self.meta.attr_names.TANGENT,
(12)                 "JOINTS_0": self.meta.attr_names.JOINTS_0,
(12)                 "WEIGHTS_0": self.meta.attr_names.WEIGHTS_0,
(12)                 "COLOR_0": self.meta.attr_names.COLOR_0,
(8)              }
(8)              meshes = []
(8)              for primitive in self.primitives:
(12)                 vao = VAO(self.name, mode=primitive.mode or moderngl.TRIANGLES)
(12)                 component_type, index_vbo = self.load_indices(primitive)
(12)                 if index_vbo is not None:
(16)                     vao.index_buffer(
(20)                         moderngl_window.ctx().buffer(index_vbo.tobytes()),
(20)                         index_element_size=component_type.size,
(16)                     )
(12)                 attributes = {}
(12)                 vbos = self.prepare_attrib_mapping(primitive)
(12)                 for vbo_info in vbos:
(16)                     dtype, buffer = vbo_info.create()
(16)                     vao.buffer(
(20)                         buffer,
(20)                         " ".join(
(24)                             [
(28)                                 "{}{}".format(attr[1], DTYPE_BUFFER_TYPE[dtype])
(28)                                 for attr in vbo_info.attributes
(24)                             ]
(20)                         ),
(20)                         [name_map[attr[0]] for attr in vbo_info.attributes],
(16)                     )
(16)                     for attr in vbo_info.attributes:
(20)                         attributes[attr[0]] = {
(24)                             "name": name_map[attr[0]],
(24)                             "components": attr[1],
(24)                             "type": vbo_info.component_type.value,
(20)                         }
(12)                 bbox_min, bbox_max = self.get_bbox(primitive)
(12)                 meshes.append(
(16)                     Mesh(
(20)                         self.name,
(20)                         vao=vao,
(20)                         attributes=attributes,
(20)                         material=(
(24)                             materials[primitive.material] if primitive.material is
not None else None
(20)                         ),
(20)                         bbox_min=bbox_min,
(20)                         bbox_max=bbox_max,
(16)                     )
(12)                 )
(8)              return meshes
(4)          def load_indices(
(8)              self, primitive: Primitives
(4)          ) -> tuple[ComponentType, npt.NDArray[Any]] | tuple[None, None]:
```

```
395: (8)                    """Loads the index buffer / polygon list for a primitive"""
396: (8)                    if primitive.indices is None or primitive.accessor is None:
397: (12)                       return None, None
398: (8)                    _, component_type, buffer = primitive.accessor.read()
399: (8)                    return component_type, buffer
400: (4)                def prepare_attrib_mapping(self, primitive: Primitives) -> list[VBOInfo]:
401: (8)                    """Pre-parse buffer mappings for each VBO to detect interleaved data for a primitive"""
402: (8)                    buffer_info: list[VBOInfo] = []
403: (8)                    for name, accessor in primitive.attributes.items():
404: (12)                       info = VBOInfo(*accessor.info())
405: (12)                       info.attributes.append((name, info.components))
406: (12)                       if buffer_info and buffer_info[-1].buffer_view == info.buffer_view:
407: (16)                           if buffer_info[-1].interleaves(info):
408: (20)                               buffer_info[-1].merge(info)
409: (20)                               continue
410: (12)                       buffer_info.append(info)
411: (8)                    return buffer_info
412: (4)                def get_bbox(self, primitive: Primitives) -> tuple[glm.vec3, glm.vec3]:
413: (8)                    """Get the bounding box for the mesh"""
414: (8)                    accessor = primitive.attributes.get("POSITION")
415: (8)                    return glm.vec3(accessor.min), glm.vec3(accessor.max)
416: (0)            class VBOInfo:
417: (4)                """Resolved data about each VBO"""
418: (4)                def __init__(
419: (8)                    self,
420: (8)                    buffer: Optional[GLTFBuffer] = None,
421: (8)                    buffer_view: Optional[GLTFBuffer] = None,
422: (8)                    byte_length: int = 0,
423: (8)                    byte_offset: int = 0,
424: (8)                    component_type: ComponentType = ComponentType("", 0, 0),
425: (8)                    components: int = 0,
426: (8)                    count: int = 0,
427: (4)                ):
428: (8)                    self.buffer = buffer  # reference to the buffer
429: (8)                    self.buffer_view = buffer_view
430: (8)                    self.byte_length = byte_length  # Raw byte buffer length
431: (8)                    self.byte_offset = byte_offset  # Raw byte offset
432: (8)                    self.component_type = component_type  # Datatype of each component
433: (8)                    self.components = components
434: (8)                    self.count = count  # number of elements of the component type size
435: (8)                    self.attributes: list[Any] = []
436: (4)                def interleaves(self, info: VBOInfo) -> bool:
437: (8)                    """Does the buffer interleave with this one?"""
438: (8)                    return bool(info.byte_offset == (self.component_type.size * self.components))
439: (4)                def merge(self, info: VBOInfo) -> None:
440: (8)                    self.components += info.components
441: (8)                    self.attributes += info.attributes
442: (4)                def create(self) -> tuple[type[object], npt.NDArray[Any]]:
443: (8)                    """Create the VBO"""
444: (8)                    assert self.buffer is not None, "No buffer defined"
445: (8)                    dtype = NP_COMPONENT_DTYPE[self.component_type.value]
446: (8)                    data = numpy.frombuffer(
447: (12)                       self.buffer.read(byte_length=self.byte_length, byte_offset=self.byte_offset),
448: (12)                       count=self.count * self.components,
449: (12)                       dtype=dtype,
450: (8)                    )
451: (8)                    return dtype, data
452: (4)                def __str__(self) -> str:
453: (8)                    assert self.buffer is not None, "No buffer defined"
454: (8)                    assert self.buffer_view is not None, "No buffer_view defined"
455: (8)                    return (
456: (12)                       "VBOInfo<buffer={}, buffer_view={},\n"
457: (12)                       "        length={}, offset={}, count={}\n"
458: (12)                       "        component_type={}, components={}, \n"
459: (12)                       "        attribs={}".format(
460: (16)                           self.buffer.id,
461: (16)                           self.buffer_view.id,
462: (16)                           self.byte_length,
463: (16)                           self.byte_offset,
464: (16)                           self.count,
465: (16)                           self.component_type.value,
466: (16)                           self.components,
467: (16)                           self.attributes,
468: (12)                       )
469: (8)                    )
470: (4)                def __repr__(self) -> str:
471: (8)                    return str(self)
472: (0)            class GLTFAccessor:
473: (4)                def __init__(self, accessor_id: int, data: dict[str, Any]):
474: (8)                    self.id = accessor_id
475: (8)                    self.bufferViewId = data.get("bufferView", 0)
476: (8)                    self.bufferView: GLTFBufferView
477: (8)                    self.byteOffset = data.get("byteOffset", 0)
478: (8)                    self.componentType = COMPONENT_TYPE[data["componentType"]]
479: (8)                    self.count = data.get("count", 1)
480: (8)                    self.min = numpy.array(data.get("min") or [-0.5, -0.5, -0.5],
dtype="f4")
481: (8)                    self.max = numpy.array(data.get("max") or [0.5, 0.5, 0.5], dtype="f4")
482: (8)                    self.type = data.get("type", "")
483: (4)                def read(self) -> tuple[int, ComponentType, npt.NDArray[Any]]:
484: (8)                    """
485: (8)                    Reads buffer data
486: (8)                    :return: component count, component type, data
487: (8)                    """
488: (8)                    dtype = NP_COMPONENT_DTYPE[self.componentType.value]
489: (8)                    return (
490: (12)                       ACCESSOR_TYPE[self.type],
491: (12)                       self.componentType,
492: (12)                       self.bufferView.read(
493: (16)                           byte_offset=self.byteOffset,
494: (16)                           dtype=dtype,
495: (16)                           count=self.count * ACCESSOR_TYPE[self.type],
496: (12)                       ),
497: (8)                    )
498: (4)                def info(self) -> tuple[GLTFBuffer, GLTFBufferView, int, int,
ComponentType, int, int]:
499: (8)                    """
500: (8)                    Get underlying buffer info for this accessor
501: (8)                    :return: buffer, byte_length, byte_offset, component_type, count
502: (8)                    """
503: (8)                    buffer, byte_length, byte_offset =
self.bufferView.info(byte_offset=self.byteOffset)
504: (8)                    return (
505: (12)                       buffer,
506: (12)                       self.bufferView,
507: (12)                       byte_length,
508: (12)                       byte_offset,
509: (12)                       self.componentType,
510: (12)                       ACCESSOR_TYPE[self.type],
511: (12)                       self.count,
512: (8)                    )
513: (0)            class GLTFBufferView:
514: (4)                def __init__(self, view_id: int, data: dict[str, Any]):
515: (8)                    self.id = view_id
516: (8)                    self.bufferId = data.get("buffer", 0)
517: (8)                    self.buffer: GLTFBuffer
518: (8)                    self.byteOffset = data.get("byteOffset", 0)
519: (8)                    self.byteLength = data.get("byteLength", 0)
520: (8)                    self.byteStride = data.get("byteStride", 0)
521: (4)                def read(
522: (8)                    self, byte_offset: int = 0, dtype: Optional[type[object]] = None,
count: int = 0
523: (4)                ) -> npt.NDArray[Any]:
524: (8)                    data = self.buffer.read(
(12)                byte_offset=byte_offset + self.byteOffset,
(12)                byte_length=self.byteLength,
(8)             )
(8)             vbo = numpy.frombuffer(data, count=count, dtype=dtype)
(8)             return vbo
(4)         def read_raw(self) -> bytes:
(8)             return self.buffer.read(byte_length=self.byteLength, byte_offset=self.byteOffset)
(4)         def info(self, byte_offset: int = 0) -> tuple[GLTFBuffer, int, int]:
(8)             """
(8)             Get the underlying buffer info
(8)             :param byte_offset: byte offset from accessor
(8)             :return: buffer, byte_length, byte_offset
(8)             """
(8)             return self.buffer, self.byteLength, byte_offset + self.byteOffset
(0)     class GLTFBuffer:
(4)         def __init__(self, buffer_id: int, data: dict[str, str], path: Path):
(8)             self.id = buffer_id
(8)             self.path = path
(8)             self.byteLength = data.get("byteLength")
(8)             uri = data.get("uri")
(8)             if uri is None:
(12)                uri = ""
(8)             self.uri = uri
(8)             self.data = b""
(4)         @property
(4)         def has_data_uri(self) -> bool:
(8)             """Is data embedded in json?"""
(8)             if self.uri == "":
(12)                return False
(8)             return self.uri.startswith("data:")
(4)         @property
(4)         def is_separate_file(self) -> bool:
(8)             """Buffer represents an independent bin file?"""
(8)             return self.uri is not None and not self.has_data_uri
(4)         def open(self) -> None:
(8)             if self.data != b"":
(12)                return
(8)             if self.has_data_uri:
(12)                self.data = base64.b64decode(self.uri[self.uri.find(",") + 1 :])
(12)                return
(8)             with open(str(self.path / (self.uri if self.uri is not None else "")), "rb") as fd:
(12)                self.data = fd.read()
(4)         def read(self, byte_offset: int = 0, byte_length: int = 0) -> bytes:
(8)             self.open()
(8)             return self.data[byte_offset : byte_offset + byte_length]
(0)     class GLTFScene:
(4)         def __init__(self, data: dict[str, list[int]]):
(8)             self.nodes = data["nodes"]
(0)     class GLTFNode:
(4)         def __init__(self, data: dict[str, Any]) -> None:
(8)             self.name = data.get("name")
(8)             self.children = data.get("children")
(8)             self.mesh = data.get("mesh")
(8)             self.camera = data.get("camera")
(8)             _matrix = data.get("matrix")
(8)             self.matrix = glm.mat4(*_matrix) if _matrix is not None else glm.mat4()
(8)             self.translation = data.get("translation")
(8)             self.rotation = data.get("rotation")
(8)             self.scale = data.get("scale")
(8)             trans_mat = (
(12)                glm.translate(glm.vec3(*self.translation))
(12)                if self.translation is not None
(12)                else glm.mat4()
(8)             )
(8)             if self.rotation is not None:
(12)                quat = glm.quat(
(16)                 x=self.rotation[0],
(16)                 y=self.rotation[1],
(16)                 z=self.rotation[2],
(16)                 w=self.rotation[3],
(12)             )
(12)             rot_mat = glm.mat4_cast(quat)
(8)         else:
(12)             rot_mat = glm.mat4()
(8)         scale_mat = glm.scale(self.scale) if self.scale is not None else glm.mat4()
(8)         self.matrix = self.matrix * trans_mat * rot_mat * scale_mat
(4)     @property
(4)     def has_children(self) -> bool:
(8)         return self.children is not None and len(self.children) > 0
(4)     @property
(4)     def is_resource_node(self) -> bool:
(8)         """Is this just a reference node to a resource?"""
(8)         return self.camera is not None or self.mesh is not None
(0) class GLTFMaterial:
(4)     def __init__(self, data: dict[str, Any]):
(8)         self.name = data.get("name")
(8)         self.doubleSided = data.get("doubleSided") or True
(8)         pbr = data["pbrMetallicRoughness"]
(8)         self.baseColorFactor = pbr.get("baseColorFactor")
(8)         self.baseColorTexture = pbr.get("baseColorTexture")
(8)         self.metallicFactor = pbr.get("metallicFactor")
(8)         self.emissiveFactor = data.get("emissiveFactor")
(0) class GLTFImage:
(4)     """
(4)     Represent texture data.
(4)     May be a file, embedded data or pointer to data in bufferview
(4)     """
(4)     def __init__(self, data: dict[str, Any]):
(8)         self.uri = data.get("uri")
(8)         self.bufferViewId = data.get("bufferView")
(8)         self.bufferView = None
(8)         self.mimeType = data.get("mimeType")
(4)     def load(self, path: Path) -> moderngl.Texture:
(8)         if self.bufferView is not None:
(12)             image = Image.open(io.BytesIO(self.bufferView.read_raw()))
(8)         elif self.uri and self.uri.startswith("data:"):
(12)             data = self.uri[self.uri.find(",") + 1 :]
(12)             image = Image.open(io.BytesIO(base64.b64decode(data)))
(12)             logger.info("Loading embedded image")
(8)         else:
(12)             path = path / Path(self.uri if self.uri is not None else "")
(12)             logger.info("Loading: %s", self.uri)
(12)             image = Image.open(path)
(8)         texture = t2d.Loader(
(12)             TextureDescription(
(16)                 label="gltf",
(16)                 image=image,
(16)                 flip=False,
(16)                 mipmap=True,
(16)                 anisotropy=16.0,
(12)             )
(8)         ).load()
(8)         return texture
(0) class GLTFTexture:
(4)     def __init__(self, data: dict[str, int]):
(8)         self.sampler: Optional[int] = data.get("sampler")
(8)         self.source: Optional[int] = data.get("source")
(0) class GLTFSampler:
(4)     def __init__(self, data):
(8)         self.magFilter = data.get("magFilter")
(8)         self.minFilter = data.get("minFilter")
(8)         self.wrapS = data.get("wrapS")
(8)         self.wrapT = data.get("wrapT")
(0) class GLTFCamera:
```

````
659: (4)                def __init__(self, data: dict[str, str]):
660: (8)                    self.data = data

----------------------------------------

File 64 - array.py:

1: (0)              import moderngl
2: (0)              from moderngl_window.exceptions import ImproperlyConfigured
3: (0)              from moderngl_window.loaders.texture.pillow import PillowLoader, image_data
4: (0)              from moderngl_window.meta.base import ResourceDescription
5: (0)              from moderngl_window.meta.texture import TextureDescription
6: (0)              class Loader(PillowLoader):
7: (4)                  kind = "array"
8: (4)                  meta: TextureDescription
9: (4)                  def __init__(self, meta: ResourceDescription):
10: (8)                     super().__init__(meta)
11: (8)                     self.layers = self.meta.layers
12: (8)                     if self.layers is None:
13: (12)                        raise ImproperlyConfigured("TextureArray requires layers
parameter")
14: (4)                 def load(self) -> moderngl.TextureArray:
15: (8)                     """Load a texture array as described by the supplied
``TextureDescription```
16: (8)                     Returns:
17: (12)                        moderngl.TextureArray: The TextureArray instance
18: (8)                     """
19: (8)                     self._open_image()
20: (8)                     width, height, depth = (
21: (12)                        self.image.size[0],
22: (12)                        self.image.size[1] // self.layers,
23: (12)                        self.layers,
24: (8)                     )
25: (8)                     components, data = image_data(self.image)
26: (8)                     texture = self.ctx.texture_array(
27: (12)                        (width, height, depth),
28: (12)                        components,
29: (12)                        data,
30: (8)                     )
31: (8)                     texture.extra = {"meta": self.meta}
32: (8)                     if self.meta.mipmap_levels is not None:
33: (12)                        self.meta.mipmap = True
34: (8)                     if self.meta.mipmap:
35: (12)                        if isinstance(self.meta.mipmap_levels, tuple):
36: (16)                            texture.build_mipmaps(*self.meta.mipmap_levels)
37: (12)                        else:
38: (16)                            texture.build_mipmaps()
39: (12)                        if self.meta.anisotropy:
40: (16)                            texture.anisotropy = self.meta.anisotropy
41: (8)                     self._close_image()
42: (8)                     return texture

----------------------------------------

File 65 - scene.py:

1: (0)              from typing import Any, Optional
2: (0)              from moderngl_window.geometry.attributes import AttributeNames
3: (0)              from moderngl_window.meta.base import ResourceDescription
4: (0)              class SceneDescription(ResourceDescription):
5: (4)                  """Describes a scene to load.
6: (4)                  The correct loader is resolved by looking at the file extension.
7: (4)                  This can be overridden by specifying a ``kind`` that maps directly
8: (4)                  to a specific loader class.
9: (4)                  .. code:: python
10: (8)                     SceneDescription(path='scenes/cube.obj')
11: (8)                     SceneDescription(path='scenes/crater.stl')
12: (8)                     SceneDescription(path='scenes/sponza.gltf')
13: (4)                 The user can also override what buffer/attribute names
````

```
14: (4)                 should be used by specifying ``attr_names``.
15: (4)                 A ``cache`` option is also available as some scene loaders
16: (4)                 supports converting the file into a different format
17: (4)                 on the fly to speed up loading.
18: (4)                 """
19: (4)                 default_kind = ""
20: (4)                 resource_type = "scenes"
21: (4)                 def __init__(
22: (8)                     self,
23: (8)                     path: Optional[str] = None,
24: (8)                     kind: Optional[str] = None,
25: (8)                     cache: bool = False,
26: (8)                     attr_names: type[AttributeNames] = AttributeNames,
27: (8)                     **kwargs: Any,
28: (4)                 ):
29: (8)                     """Create a scene description.
30: (8)                     Keyword Args:
31: (12)                        path (str): Path to resource
32: (12)                        kind (str): Loader kind
33: (12)                        cache (str): Use the loader caching system if present
34: (12)                        attr_names (AttributeNames): Attrib name config
35: (12)                        **kwargs: Optional custom attributes
36: (8)                     """
37: (8)                     if attr_names is None:
38: (12)                        attr_names = AttributeNames
39: (8)                     kwargs.update({"path": path, "kind": kind, "cache": cache,
"attr_names": attr_names})
40: (8)                     super().__init__(**kwargs)
41: (4)                 @property
42: (4)                 def cache(self) -> bool:
43: (8)                     """bool: Use cache feature in scene loader"""
44: (8)                     return bool(self._kwargs["cache"])
45: (4)                 @property
46: (4)                 def attr_names(self) -> AttributeNames:
47: (8)                     """AttributeNames: Attribute name config"""
48: (8)                     return self._kwargs["attr_names"]
```

File 66 - types.py:

```
1: (0)              """
2: (0)              Notes from moderngl:
3: (0)              The vao_content is a list of 3-tuples (buffer, format, attribs)
4: (0)              the format can have an empty or '/v', '/i', '/r' ending.
5: (0)              '/v' attributes are the default
6: (0)              '/i` attributes are per instance attributes
7: (0)              '/r' attributes are per render (like a uniform)
8: (0)              Example:
9: (4)                  vao_content = [
10: (8)                     (self.position_vertex_buffer, '2f', 'in_vert'),
11: (8)                     (self.color_buffer, '3f', 'in_color'),
12: (8)                     (self.pos_scale_buffer, '2f 1f/i', 'in_pos', 'in_scale'),
13: (4)                 ]
14: (0)             """
15: (0)             import re
16: (0)             from functools import lru_cache
17: (0)             VALID_DIVISORS = ["v", "i", "r"]
18: (0)             class BufferFormat:
19: (4)                 def __init__(
20: (8)                     self,
21: (8)                     format_string: str,
22: (8)                     components: int,
23: (8)                     bytes_per_component: int,
24: (8)                     per_instance: bool = False,
25: (4)                 ):
26: (8)                     """
27: (8)                     Args:
28: (12)                        format_string (str): moderngl format string
```

```
29: (12)                        components (int): components
30: (12)                        byte_size (int): bytes per component
31: (12)                        per_instance (bool): Instanced attribute
32: (8)                     """
33: (8)                     self.format = format_string
34: (8)                     self.components = components
35: (8)                     self.bytes_per_component = bytes_per_component
36: (8)                     self.per_instance = per_instance
37: (4)                 @property
38: (4)                 def bytes_total(self) -> int:
39: (8)                     """int: total byte size if this type"""
40: (8)                     return self.components * self.bytes_per_component
41: (4)                 def pad_str(self) -> str:
42: (8)                     """Padding string used my moderngl in interleaved buffers"""
43: (8)                     return "{}x{}".format(self.components, self.bytes_per_component)
44: (4)                 def __str__(self) -> str:
45: (8)                     return "<BufferFormat {} components={} bytes_per_component={}>".format(
46: (12)                        self.format, self.components, self.bytes_per_component
47: (8)                     )
48: (4)                 def __repr__(self) -> str:
49: (8)                     return str(self)
50: (0)             @lru_cache(maxsize=500)
51: (0)             def attribute_format(attr_format: str) -> BufferFormat:
52: (4)                 """Look up info about an attribute format.
53: (4)                 Translate the format into a BufferFormat instance
54: (4)                 containing things like byte size and components
55: (4)                 Args:
56: (8)                     buffer_format (str): Format of an attribute
57: (4)                 Returns:
58: (8)                     BufferFormat instance
59: (4)                 """
60: (4)                 if not attr_format:
61: (8)                     raise ValueError("Cannot resolve buffer format: '{}'".format(attr_format))
62: (4)                 parts = attr_format.split("/")
63: (4)                 fmt = parts[0]
64: (4)                 divisor = ""
65: (4)                 if len(parts) > 1:
66: (8)                     divisor = parts[1]
67: (8)                     if divisor not in VALID_DIVISORS:
68: (12)                        raise ValueError(
69: (16)                            "Invalid attribute divisor '{}' in '{}'".format(divisor, buffer_format)
70: (12)                        )
71: (4)                 parts = re.split(r"([fiudn])", fmt)
72: (4)                 components = 1
73: (4)                 if parts[0].isalnum():
74: (8)                     components = int(parts[0])
75: (8)                     bformat = fmt[len(parts[0]) :]
76: (4)                 else:
77: (8)                     bformat = fmt
78: (4)                 fmt_info = buffer_format(bformat)
79: (4)                 return BufferFormat(
80: (8)                     "{}{}{}".format(components, bformat, "/{}".format(divisor) if divisor else ""),
81: (8)                     components,
82: (8)                     fmt_info.bytes_per_component,
83: (8)                     per_instance=divisor == "i",
84: (4)                 )
85: (0)             def parse_attribute_formats(frmt: str) -> list[BufferFormat]:
86: (4)                 return [attribute_format(attr) for attr in frmt.split()]
87: (0)             def buffer_format(frmt: str) -> BufferFormat:
88: (4)                 """Look up info about a buffer format type
89: (4)                 Args:
90: (8)                     frmt (str): format string such as 'f', 'i' and 'u'
91: (4)                 Returns:
92: (8)                     BufferFormat instance
93: (4)                 """
```

```
94: (4)                 try:
95: (8)                     return BUFFER_FORMATS[frmt]
96: (4)                 except KeyError:
97: (8)                     raise ValueError(
98: (12)                        "Buffer format '{}' unknown. Valid formats: {}".format(frmt,
BUFFER_FORMATS.keys())
99: (8)                     )
100: (0)            BUFFER_FORMATS = {
101: (4)                "f": BufferFormat("f", 1, 4),
102: (4)                "f1": BufferFormat("f1", 1, 1),
103: (4)                "f2": BufferFormat("f2", 1, 2),
104: (4)                "f4": BufferFormat("f4", 1, 4),
105: (4)                "f8": BufferFormat("f8", 1, 8),
106: (4)                "u": BufferFormat("u", 1, 4),
107: (4)                "u1": BufferFormat("u1", 1, 1),
108: (4)                "u2": BufferFormat("u2", 1, 2),
109: (4)                "u4": BufferFormat("u4", 1, 4),
110: (4)                "i": BufferFormat("i", 1, 4),
111: (4)                "i1": BufferFormat("i1", 1, 1),
112: (4)                "i2": BufferFormat("i2", 1, 2),
113: (4)                "i4": BufferFormat("i4", 1, 4),
114: (4)                "nf": BufferFormat("nf", 1, 4),
115: (4)                "nf1": BufferFormat("nf1", 1, 1),
116: (4)                "nf2": BufferFormat("nf2", 1, 2),
117: (4)                "nf4": BufferFormat("nf4", 1, 4),
118: (4)                "nu": BufferFormat("nu", 1, 4),
119: (4)                "nu1": BufferFormat("nu1", 1, 1),
120: (4)                "nu2": BufferFormat("nu2", 1, 2),
121: (4)                "nu4": BufferFormat("nu4", 1, 4),
122: (4)                "ni": BufferFormat("ni", 1, 4),
123: (4)                "ni1": BufferFormat("ni1", 1, 1),
124: (4)                "ni2": BufferFormat("ni2", 1, 2),
125: (4)                "ni4": BufferFormat("ni4", 1, 4),
126: (0)            }
```

File 67 - binary.py:

```
1: (0)              import logging
2: (0)              from moderngl_window.exceptions import ImproperlyConfigured
3: (0)              from moderngl_window.loaders.base import BaseLoader
4: (0)              logger = logging.getLogger(__name__)
5: (0)              class Loader(BaseLoader):
6: (4)                  kind = "binary"
7: (4)                  def load(self) -> bytes:
8: (8)                      """Load a file in binary mode
9: (8)                      Returns:
10: (12)                        bytes: The bytes contents of the file
11: (8)                     """
12: (8)                     self.meta.resolved_path = self.find_data(self.meta.path)
13: (8)                     if not self.meta.resolved_path:
14: (12)                        raise ImproperlyConfigured("Data file '{}' not
found".format(self.meta.path))
15: (8)                     logger.info("Loading: %s", self.meta.path)
16: (8)                     with open(str(self.meta.resolved_path), "rb") as fd:
17: (12)                        return fd.read()
```

File 68 - single.py:

```
1: (0)              import logging
2: (0)              from pathlib import Path
3: (0)              from typing import Union
4: (0)              import moderngl
5: (0)              from moderngl_window.exceptions import ImproperlyConfigured
6: (0)              from moderngl_window.loaders.base import BaseLoader
7: (0)              from moderngl_window.opengl import program
```

```
8: (0)              logger = logging.getLogger(__name__)
9: (0)              class Loader(BaseLoader):
10: (4)                 kind = "single"
11: (4)                 meta: program.ProgramDescription
12: (4)                 def load(self) -> moderngl.Program:
13: (8)                     """Loads a shader program from a single glsl file.
14: (8)                     Each shader type is separated by preprocessors
15: (8)                     - VERTEX_SHADER
16: (8)                     - FRAGMENT_SHADER
17: (8)                     - GEOMETRY_SHADER
18: (8)                     - TESS_CONTROL_SHADER
19: (8)                     - TESS_EVALUATION_SHADER
20: (8)                     Example:
21: (8)                     .. code:: glsl
22: (12)                        in vec3 in_position;
23: (12)                        in vec2 in_texcoord_0;
24: (12)                        out vec2 uv0;
25: (12)                        void main() {
26: (16)                            gl_Position = vec4(in_position, 1);
27: (16)                            uv0 = in_texcoord_0;
28: (12)                        }
29: (12)                        out vec4 fragColor;
30: (12)                        uniform sampler2D texture0;
31: (12)                        in vec2 uv0;
32: (12)                        void main() {
33: (16)                            fragColor = texture(texture0, uv0);
34: (12)                        }
35: (8)                     Returns:
36: (12)                        moderngl.Program: The Program instance
37: (8)                     """
38: (8)                     prog: Union[moderngl.Program, program.ReloadableProgram]
39: (8)                     assert self.meta.path is not None, "There is no path for the resource"
40: (8)                     assert self.meta.path is not None, "There is no path for the resource"
41: (8)                     self.meta.resolved_path, source = self._load_source(self.meta.path)
42: (8)                     shaders = program.ProgramShaders.from_single(self.meta, source)
43: (8)                     shaders.handle_includes(self._load_source)
44: (8)                     prog = shaders.create()
45: (8)                     if self.meta.reloadable:
46: (12)                        self.meta.reloadable = False
47: (12)                        prog = program.ReloadableProgram(self.meta, prog)
48: (8)                     return prog
49: (4)                 def _load_source(self, path: Union[Path, str]) -> tuple[Path, str]:
50: (8)                     """Finds and loads a single source file.
51: (8)                     Args:
52: (12)                        path: Path to resource
53: (8)                     Returns:
54: (12)                        tuple[resolved_path, source]: The resolved path and the source
55: (8)                     """
56: (8)                     resolved_path = self.find_program(path)
57: (8)                     if not resolved_path:
58: (12)                        raise ImproperlyConfigured("Cannot find program
'{}'".format(path))
59: (8)                     logger.info("Loading: %s", path)
60: (8)                     with open(str(resolved_path), "r") as fd:
61: (12)                        return resolved_path, fd.read()

----------------------------------------

File 69 - pillow.py:

1: (0)              import logging
2: (0)              from pathlib import Path
3: (0)              from typing import Optional, Union
4: (0)              try:
5: (4)                  from PIL import Image
6: (0)              except ImportError as ex:
7: (4)                  raise ImportError("Texture loader 'PillowLoader' requires Pillow:
{}".format(ex))
8: (0)              from moderngl_window.exceptions import ImproperlyConfigured
```

```
9: (0)              from moderngl_window.loaders.base import BaseLoader
10: (0)             from moderngl_window.meta.base import ResourceDescription
11: (0)             from moderngl_window.meta.texture import TextureDescription
12: (0)             from moderngl_window.resources.textures import TextureAny
13: (0)             logger = logging.getLogger(__name__)
14: (0)             class PillowLoader(BaseLoader):
15: (4)                 """Base loader using PIL/Pillow"""
16: (4)                 kind = "__unknown__"
17: (4)                 image: Image.Image
18: (4)                 meta: TextureDescription
19: (4)                 def __init__(self, meta: ResourceDescription):
20: (8)                     super().__init__(meta)
21: (4)                 def load(self) -> TextureAny:
22: (8)                     raise NotImplementedError()
23: (4)                 def _open_image(self) -> Image.Image:
24: (8)                     if self.meta.image:
25: (12)                        self.image = self.meta.image
26: (8)                     else:
27: (12)                        self.meta.resolved_path = self.find_texture(self.meta.path)
28: (12)                        logger.info("loading %s", self.meta.resolved_path)
29: (12)                        if not self.meta.resolved_path:
30: (16)                            raise ImproperlyConfigured("Cannot find texture:
{}".format(self.meta.path))
31: (12)                        self.image = Image.open(self.meta.resolved_path)
32: (12)                        if (
33: (16)                            hasattr(self.image, "is_animated")
34: (16)                            and self.image.is_animated
35: (16)                            and hasattr(self.image, "n_frames")
36: (12)                        ):
37: (16)                            self.layers = self.image.n_frames
38: (16)                            anim = Image.new(
39: (20)                                self.image.palette.mode if self.image.palette is not None
else "L",
40: (20)                                (self.image.width, self.image.height *
self.image.n_frames),
41: (16)                            )
42: (16)                            anim.putalpha(0)
43: (16)                            for frame_number in range(self.image.n_frames):
44: (20)                                self.image.seek(frame_number)
45: (20)                                frame = self._palette_to_raw(self.image, mode="RGBA")
46: (20)                                anim.paste(frame, (0, frame_number * self.image.height))
47: (16)                            self.image = anim
48: (8)                     self.image = self._apply_modifiers(self.image)
49: (8)                     return self.image
50: (4)                 def _load_texture(self, path: Union[str, Path]) -> Image.Image:
51: (8)                     """Find and load separate texture. Useful when multiple textue files
needs to be loaded"""
52: (8)                     resolved_path = self.find_texture(path)
53: (8)                     logger.info("loading %s", resolved_path)
54: (8)                     if not resolved_path:
55: (12)                        raise ImproperlyConfigured("Cannot find texture: {}".format(path))
56: (8)                     image = Image.open(resolved_path)
57: (8)                     return self._apply_modifiers(image)
58: (4)                 def _apply_modifiers(self, image: Image.Image) -> Image.Image:
59: (8)                     if self.meta.flip_x:
60: (12)                        image = image.transpose(Image.Transpose.FLIP_LEFT_RIGHT)
61: (8)                     if self.meta.flip_y:
62: (12)                        image = image.transpose(Image.Transpose.FLIP_TOP_BOTTOM)
63: (8)                     return self._palette_to_raw(image)
64: (4)                 def _palette_to_raw(self, image: Image.Image, mode: Optional[str] = None)
-> Image.Image:
65: (8)                     """Converts image to raw if palette is present"""
66: (8)                     if image.palette and image.palette.mode.lower() in ["rgb", "rgba"]:
67: (12)                        mode = mode or image.palette.mode
68: (12)                        logger.debug("Converting P image to %s using palette", mode)
69: (12)                        return image.convert(mode)
70: (8)                     return image
71: (4)                 def _close_image(self) -> None:
72: (8)                     self.image.close()
```

```
73: (0)              def image_data(image: Image.Image) -> tuple[int, bytes]:
74: (4)                  """Get components and bytes for an image.
75: (4)                  The number of components is assumed by image
76: (4)                  size and the byte length of the raw data.
77: (4)                  Returns:
78: (8)                      tuple[int, bytes]: Number of components, byte data
79: (4)                  """
80: (4)                  data = image.tobytes()
81: (4)                  components = len(data) // (image.size[0] * image.size[1])
82: (4)                  logger.debug(
83: (8)                      "image_data size=[%s, %s] components=%s bytes=%s",
84: (8)                      image.size[0],
85: (8)                      image.size[1],
86: (8)                      components,
87: (8)                      len(data),
88: (4)                  )
89: (4)                  return components, data
```

File 70 - program.py:

```
1: (0)               from typing import Any, Optional
2: (0)               from moderngl_window.meta.base import ResourceDescription
3: (0)               class ProgramDescription(ResourceDescription):
4: (4)                   """Describes a program to load
5: (4)                   By default a program can be loaded in the following ways:
6: (4)                   - By supplying a `path` to s single glsl file containing all shaders
7: (4)                   - By supplying several paths to separate files containing each shader
type.
8: (6)                     For example ``vertex_shader``, ``fragment_shader`` .. etc.
9: (4)                   .. code:: python
10: (8)                      ProgramDescription(path='programs/myprogram.glsl')
11: (8)                      ProgramDescription(
12: (12)                         vertex_shader='programs/myprogram_vs.glsl'.
13: (12)                         fragment_shader='programs/myprogram_fs.glsl'.
14: (12)                         geometry_shader='programs/myprogram_gs.glsl'.
15: (8)                      )
16: (4)                  """
17: (4)                  default_kind = ""
18: (4)                  resource_type = "programs"
19: (4)                  def __init__(
20: (8)                      self,
21: (8)                      path: Optional[str] = None,
22: (8)                      kind: Optional[str] = None,
23: (8)                      reloadable: bool = False,
24: (8)                      vertex_shader: Optional[str] = None,
25: (8)                      geometry_shader: Optional[str] = None,
26: (8)                      fragment_shader: Optional[str] = None,
27: (8)                      tess_control_shader: Optional[str] = None,
28: (8)                      tess_evaluation_shader: Optional[str] = None,
29: (8)                      compute_shader: Optional[str] = None,
30: (8)                      defines: Optional[dict[str, Any]] = None,
31: (8)                      varyings: Optional[list[str]] = None,
32: (8)                      **kwargs: Any,
33: (4)                  ):
34: (8)                      """Create a program description
35: (8)                      Keyword Args:
36: (12)                         path (str): path to the resource relative to search directories
37: (12)                         kind (str): The kind of loader to use
38: (12)                         reloadable (bool): Should this program be reloadable
39: (12)                         vertex_shader (str): Path to vertex shader file
40: (12)                         geometry_shader (str): Path to geometry shader
41: (12)                         fragment_shader (str): Path to fragmet shader
42: (12)                         tess_control_shader (str) Path to tess control shader
43: (12)                         tess_evaluation_shader (str): Path to tess eval shader
44: (12)                         compute_shader (str): Path to compute shader
45: (12)                         defines (dict): Dictionary with define values to replace in the
source
```

```
46: (12)                            varyings (list): List of varying names for transform shader
47: (12)                            **kwargs: Optional custom attributes
48: (8)                         """
49: (8)                         kwargs.update(
50: (12)                            {
51: (16)                                "path": path,
52: (16)                                "kind": kind,
53: (16)                                "reloadable": reloadable,
54: (16)                                "vertex_shader": vertex_shader,
55: (16)                                "geometry_shader": geometry_shader,
56: (16)                                "fragment_shader": fragment_shader,
57: (16)                                "tess_control_shader": tess_control_shader,
58: (16)                                "tess_evaluation_shader": tess_evaluation_shader,
59: (16)                                "compute_shader": compute_shader,
60: (16)                                "defines": defines,
61: (16)                                "varyings": varyings,
62: (12)                            }
63: (8)                         )
64: (8)                         super().__init__(**kwargs)
65: (4)                     @property
66: (4)                     def reloadable(self) -> Optional[bool]:
67: (8)                         """bool: if this program is reloadable"""
68: (8)                         return self._kwargs.get("reloadable")
69: (4)                     @reloadable.setter
70: (4)                     def reloadable(self, value: Any) -> None:
71: (8)                         self._kwargs["reloadable"] = value
72: (4)                     @property
73: (4)                     def vertex_shader(self) -> Optional[str]:
74: (8)                         """str: Relative path to vertex shader"""
75: (8)                         return self._kwargs.get("vertex_shader")
76: (4)                     @property
77: (4)                     def geometry_shader(self) -> Optional[str]:
78: (8)                         """str: Relative path to geometry shader"""
79: (8)                         return self._kwargs.get("geometry_shader")
80: (4)                     @property
81: (4)                     def fragment_shader(self) -> Optional[str]:
82: (8)                         """str: Relative path to fragment shader"""
83: (8)                         return self._kwargs.get("fragment_shader")
84: (4)                     @property
85: (4)                     def tess_control_shader(self) -> Optional[str]:
86: (8)                         """str: Relative path to tess control shader"""
87: (8)                         return self._kwargs.get("tess_control_shader")
88: (4)                     @property
89: (4)                     def tess_evaluation_shader(self) -> Optional[str]:
90: (8)                         """str: Relative path to tessellation evaluation shader"""
91: (8)                         return self._kwargs.get("tess_evaluation_shader")
92: (4)                     @property
93: (4)                     def compute_shader(self) -> Optional[str]:
94: (8)                         """str: Relative path to compute shader"""
95: (8)                         return self._kwargs.get("compute_shader")
96: (4)                     @property
97: (4)                     def defines(self) -> dict[str, Any]:
98: (8)                         """dict: Dictionary with define values to replace in the source"""
99: (8)                         return self._kwargs.get("defines", {})
100: (4)                    @property
101: (4)                    def varyings(self) -> list[str]:
102: (8)                        """list: List of varying names for transform shaders"""
103: (8)                        return self._kwargs.get("varyings", [])

----------------------------------------

File 71 - texture.py:

1: (0)                  from typing import Any, Optional
2: (0)                  from PIL.Image import Image
3: (0)                  from moderngl_window.meta.base import ResourceDescription
4: (0)                  class TextureDescription(ResourceDescription):
5: (4)                      """Describes a texture to load.
6: (4)                      Example:
```

```
7: (4)                  .. code:: python
8: (8)                      TextureDescription(path='textures/wood.png')
9: (8)                      TextureDescription(path='textures/wood.png', mipmap=True, anisotropy=16.0)
10: (8)                     TextureDescription(path='textures/tiles.png', layers=10, kind='array')
11: (4)                 """
12: (4)                 default_kind = "2d"
13: (4)                 resource_type = "textures"
14: (4)                 def __init__(
15: (8)                     self,
16: (8)                     path: Optional[str] = None,
17: (8)                     kind: Optional[str] = None,
18: (8)                     flip: bool = True,
19: (8)                     flip_x: bool = False,
20: (8)                     flip_y: bool = True,
21: (8)                     mipmap: bool = False,
22: (8)                     mipmap_levels: Optional[tuple[int, int]] = None,
23: (8)                     anisotropy: float = 1.0,
24: (8)                     image: Optional[Image] = None,
25: (8)                     layers: Optional[int] = None,
26: (8)                     pos_x: Optional[str] = None,
27: (8)                     pos_y: Optional[str] = None,
28: (8)                     pos_z: Optional[str] = None,
29: (8)                     neg_x: Optional[str] = None,
30: (8)                     neg_y: Optional[str] = None,
31: (8)                     neg_z: Optional[str] = None,
32: (8)                     **kwargs: Any,
33: (4)                 ):
34: (8)                     """Describes a texture resource
35: (8)                     Args:
36: (12)                        path (str): path to resource relative to search directories
37: (12)                        kind (str): The kind of loader to use
38: (12)                        flip (boolean): (use flip_y) Flip the image vertically (top to bottom)
39: (12)                        flip_x (boolean): Flip the image horizontally (left to right)
40: (12)                        flip_y (boolean): Flip the image vertically (top to bottom)
41: (12)                        mipmap (bool): Generate mipmaps. Will generate max possible levels unless
42: (27)                                       `mipmap_levels` is defined.
43: (12)                        mipmap_levels (tuple): (base, max_level) controlling mipmap generation.
44: (35)                                               When defined the `mipmap` parameter is automatically `True`.
45: (12)                        anisotropy (float): Number of samples for anisotropic filtering
46: (12)                        image: PIL image for when loading embedded resources
47: (12)                        layers: (int): Number of layers for texture arrays
48: (12)                        neg_x (str): Path to negative x texture in a cube map
49: (12)                        neg_y (str): Path to negative y texture in a cube map
50: (12)                        neg_z (str): Path to negative z texture in a cube map
51: (12)                        pos_x (str): Path to positive x texture in a cube map
52: (12)                        pop_y (str): Path to positive y texture in a cube map
53: (12)                        pos_z (str): Path to positive z texture in a cube map
54: (12)                        **kwargs: Any optional/custom attributes
55: (8)                     """
56: (8)                     kwargs.update(
57: (12)                        {
58: (16)                            "path": path,
59: (16)                            "kind": kind,
60: (16)                            "flip_x": flip_x,
61: (16)                            "flip_y": flip and flip_y,
62: (16)                            "mipmap": mipmap,
63: (16)                            "mipmap_levels": mipmap_levels,
64: (16)                            "anisotropy": anisotropy,
65: (16)                            "layers": layers,
66: (16)                            "image": image,
67: (16)                            "neg_x": neg_x,
68: (16)                            "neg_y": neg_y,
69: (16)                            "neg_z": neg_z,
70: (16)                            "pos_x": pos_x,
71: (16)                            "pos_y": pos_y,
72: (16)                            "pos_z": pos_z,
73: (12)                        }
74: (8)                     )
75: (8)                     super().__init__(**kwargs)
76: (4)                 @property
77: (4)                 def flip_x(self) -> Optional[bool]:
78: (8)                     """bool: If the image should be flipped horizontally (left to
right)"""
79: (8)                     return self._kwargs.get("flip_x")
80: (4)                 @property
81: (4)                 def flip_y(self) -> Optional[bool]:
82: (8)                     """bool: If the image should be flipped vertically (top to bottom)"""
83: (8)                     return self._kwargs.get("flip_y")
84: (4)                 @property
85: (4)                 def mipmap(self) -> Optional[bool]:
86: (8)                     """bool: If mipmaps should be generated"""
87: (8)                     return self._kwargs.get("mipmap")
88: (4)                 @mipmap.setter
89: (4)                 def mipmap(self, value: float) -> None:
90: (8)                     self._kwargs["mipmap"] = value
91: (4)                 @property
92: (4)                 def mipmap_levels(self) -> Optional[tuple[int, int]]:
93: (8)                     """tuple[int, int]: base, max_level for mipmap generation"""
94: (8)                     return self._kwargs.get("mipmap_levels")
95: (4)                 @property
96: (4)                 def layers(self) -> Optional[int]:
97: (8)                     """int: Number of layers in texture array"""
98: (8)                     return self._kwargs.get("layers")
99: (4)                 @property
100: (4)                def anisotropy(self) -> Optional[float]:
101: (8)                    """float: Number of samples for anisotropic filtering"""
102: (8)                    return self._kwargs.get("anisotropy")
103: (4)                @property
104: (4)                def image(self) -> Optional[Image]:
105: (8)                    """Image: PIL image when loading embedded resources"""
106: (8)                    return self._kwargs.get("image")
107: (4)                @property
108: (4)                def pos_x(self) -> Optional[str]:
109: (8)                    """str: Path to positive x in a cubemap texture"""
110: (8)                    return self._kwargs.get("pos_x")
111: (4)                @property
112: (4)                def pos_y(self) -> Optional[str]:
113: (8)                    """str: Path to positive y in a cubemap texture"""
114: (8)                    return self._kwargs.get("pos_y")
115: (4)                @property
116: (4)                def pos_z(self) -> Optional[str]:
117: (8)                    """str: Path to positive z in a cubemap texture"""
118: (8)                    return self._kwargs.get("pos_z")
119: (4)                @property
120: (4)                def neg_x(self) -> Optional[str]:
121: (8)                    """str: Path to negative x in a cubemap texture"""
122: (8)                    return self._kwargs.get("neg_x")
123: (4)                @property
124: (4)                def neg_y(self) -> Optional[str]:
125: (8)                    """str: Path to negative y in a cubemap texture"""
126: (8)                    return self._kwargs.get("neg_y")
127: (4)                @property
128: (4)                def neg_z(self) -> Optional[str]:
129: (8)                    """str: Path to negative z in a cubemap texture"""
130: (8)                    return self._kwargs.get("neg_z")

----------------------------------------

File 72 - program.py:

1: (0)              """
2: (0)              Helper classes for loading shader
3: (0)              """
```

```
(0)             import re
(0)             from typing import Any, Callable, Optional, Union
(0)             import moderngl
(0)             import moderngl_window
(0)             from moderngl_window.meta import ProgramDescription as ProgramDescription
(0)             VERTEX_SHADER = "VERTEX_SHADER"
(0)             GEOMETRY_SHADER = "GEOMETRY_SHADER"
(0)             FRAGMENT_SHADER = "FRAGMENT_SHADER"
(0)             TESS_CONTROL_SHADER = "TESS_CONTROL_SHADER"
(0)             TESS_EVALUATION_SHADER = "TESS_EVALUATION_SHADER"
(0)             COMPUTE_SHADER = "COMPUTE_SHADER"
(0)             class ProgramShaders:
(4)                 """Helper class preparing shader source strings for a program"""
(4)                 def __init__(self, meta: ProgramDescription):
(8)                     self.meta = meta
(8)                     self.vertex_source: Optional[ShaderSource] = None
(8)                     self.geometry_source: Optional[ShaderSource] = None
(8)                     self.fragment_source: Optional[ShaderSource] = None
(8)                     self.tess_control_source: Optional[ShaderSource] = None
(8)                     self.tess_evaluation_source: Optional[ShaderSource] = None
(8)                     self.compute_shader_source: Optional[ShaderSource] = None
(4)                 @property
(4)                 def ctx(self) -> moderngl.Context:
(8)                     """The moderngl context"""
(8)                     return moderngl_window.ctx()
(4)                 @classmethod
(4)                 def from_single(
(8)                     cls: type["ProgramShaders"], meta: ProgramDescription, source: str
(4)                 ) -> "ProgramShaders":
(8)                     """Initialize a single glsl string containing all shaders"""
(8)                     instance = cls(meta)
(8)                     instance.vertex_source = ShaderSource(
(12)                        VERTEX_SHADER,
(12)                        meta.path or meta.vertex_shader,
(12)                        source,
(12)                        defines=meta.defines,
(8)                     )
(8)                     if GEOMETRY_SHADER in source:
(12)                        instance.geometry_source = ShaderSource(
(16)                            GEOMETRY_SHADER,
(16)                            meta.path or meta.geometry_shader,
(16)                            source,
(16)                            defines=meta.defines,
(12)                        )
(8)                     if FRAGMENT_SHADER in source:
(12)                        instance.fragment_source = ShaderSource(
(16)                            FRAGMENT_SHADER,
(16)                            meta.path or meta.fragment_shader,
(16)                            source,
(16)                            defines=meta.defines,
(12)                        )
(8)                     if TESS_CONTROL_SHADER in source:
(12)                        instance.tess_control_source = ShaderSource(
(16)                            TESS_CONTROL_SHADER,
(16)                            meta.path or meta.tess_control_shader,
(16)                            source,
(16)                            defines=meta.defines,
(12)                        )
(8)                     if TESS_EVALUATION_SHADER in source:
(12)                        instance.tess_evaluation_source = ShaderSource(
(16)                            TESS_EVALUATION_SHADER,
(16)                            meta.path or meta.tess_evaluation_shader,
(16)                            source,
(16)                            defines=meta.defines,
(12)                        )
(8)                     return instance
(4)                 @classmethod
(4)                 def from_separate(
(8)                     cls: type["ProgramShaders"],
(8)          meta: ProgramDescription,
(8)          vertex_source: str,
(8)          geometry_source: Optional[str] = None,
(8)          fragment_source: Optional[str] = None,
(8)          tess_control_source: Optional[str] = None,
(8)          tess_evaluation_source: Optional[str] = None,
(4)      ) -> "ProgramShaders":
(8)          """Initialize multiple shader strings"""
(8)          instance = cls(meta)
(8)          instance.vertex_source = ShaderSource(
(12)             VERTEX_SHADER,
(12)             meta.path or meta.vertex_shader,
(12)             vertex_source,
(12)             defines=meta.defines,
(8)          )
(8)          if geometry_source is not None:
(12)             instance.geometry_source = ShaderSource(
(16)                 GEOMETRY_SHADER,
(16)                 meta.path or meta.geometry_shader,
(16)                 geometry_source,
(16)                 defines=meta.defines,
(12)             )
(8)          if fragment_source is not None:
(12)             instance.fragment_source = ShaderSource(
(16)                 FRAGMENT_SHADER,
(16)                 meta.path or meta.fragment_shader,
(16)                 fragment_source,
(16)                 defines=meta.defines,
(12)             )
(8)          if tess_control_source is not None:
(12)             instance.tess_control_source = ShaderSource(
(16)                 TESS_CONTROL_SHADER,
(16)                 meta.path or meta.tess_control_shader,
(16)                 tess_control_source,
(16)                 defines=meta.defines,
(12)             )
(8)          if tess_evaluation_source is not None:
(12)             instance.tess_evaluation_source = ShaderSource(
(16)                 TESS_EVALUATION_SHADER,
(16)                 meta.path or meta.tess_control_shader,
(16)                 tess_evaluation_source,
(16)                 defines=meta.defines,
(12)             )
(8)          return instance
(4)      @classmethod
(4)      def compute_shader(
(8)          cls: type["ProgramShaders"], meta: ProgramDescription, compute_shader_source: str = ""
(4)      ) -> "ProgramShaders":
(8)          instance = cls(meta)
(8)          instance.compute_shader_source = ShaderSource(
(12)             COMPUTE_SHADER,
(12)             "" if meta.compute_shader is None else meta.compute_shader,
(12)             compute_shader_source,
(12)             defines=meta.defines,
(8)          )
(8)          return instance
(4)      def create_compute_shader(self) -> moderngl.ComputeShader:
(8)          assert self.compute_shader_source is not None, "There is not compute_shader to create"
(8)          return self.ctx.compute_shader(self.compute_shader_source.source)
(4)      def create(self) -> moderngl.Program:
(8)          """
(8)          Creates a shader program.
(8)          Returns:
(12)             ModernGL Program instance
(8)          """
(8)          out_attribs = []
(8)          assert self.vertex_source is not None, "There is no vertex_source to
```

```
use"
140: (8)                    if not self.fragment_source:
141: (12)                       if self.geometry_source:
142: (16)                           out_attribs = self.meta.varyings or
self.geometry_source.find_out_attribs()
143: (12)                       else:
144: (16)                           out_attribs = self.meta.varyings or
self.vertex_source.find_out_attribs()
145: (8)                    program = self.ctx.program(
146: (12)                       vertex_shader=self.vertex_source.source,
147: (12)                       geometry_shader=(self.geometry_source.source if
self.geometry_source else None),
148: (12)                       fragment_shader=(self.fragment_source.source if
self.fragment_source else None),
149: (12)                       tess_control_shader=(
150: (16)                           self.tess_control_source.source if self.tess_control_source
else None
151: (12)                       ),
152: (12)                       tess_evaluation_shader=(
153: (16)                           self.tess_evaluation_source.source if
self.tess_evaluation_source else None
154: (12)                       ),
155: (12)                       varyings=tuple(out_attribs),
156: (8)                    )
157: (8)                    program.extra = {"meta": self.meta}
158: (8)                    return program
159: (4)                def handle_includes(self, load_source_func: Callable[[Any], Any]) -> None:
160: (8)                    """Resolves ``#include`` preprocessors
161: (8)                    Args:
162: (12)                       load_source_func (func): A function for finding and loading a
source
163: (8)                    """
164: (8)                    if self.vertex_source:
165: (12)                       self.vertex_source.handle_includes(load_source_func)
166: (8)                    if self.geometry_source:
167: (12)                       self.geometry_source.handle_includes(load_source_func)
168: (8)                    if self.fragment_source:
169: (12)                       self.fragment_source.handle_includes(load_source_func)
170: (8)                    if self.tess_control_source:
171: (12)                       self.tess_control_source.handle_includes(load_source_func)
172: (8)                    if self.tess_evaluation_source:
173: (12)                       self.tess_evaluation_source.handle_includes(load_source_func)
174: (8)                    if self.compute_shader_source:
175: (12)                       self.compute_shader_source.handle_includes(load_source_func)
176: (0)            class ShaderSource:
177: (4)                """
178: (4)                Helper class representing a single shader type.
179: (4)                It ensures the source has the right format, injects ``#define`` pre-
processors,
180: (4)                resolves ``#include`` pre-processors etc.
181: (4)                A ``ShaderSource`` can be the base/root shader or a source referenced in
an ``#include``.
182: (4)                """
183: (4)                def __init__(
184: (8)                    self,
185: (8)                    shader_type: Optional[str],
186: (8)                    name: Optional[str],
187: (8)                    source: str,
188: (8)                    defines: Optional[dict[str, str]] = None,
189: (8)                    id: int = 0,
190: (8)                    root: bool = True,
191: (4)                ):
192: (8)                    """Create shader source.
193: (8)                    Args:
194: (12)                       shader_type (str):
195: (16)                           A preprocessor name for setting the shader type
196: (12)                       name (str):
197: (16)                           A string (usually the path) so we can give useful error
messages to the user
```

```
198: (12)                       source (str):
199: (16)                           The raw source for the shader
200: (8)                    Keyword Args:
201: (12)                       id (int):
202: (16)                           The source number. Used when shader consists of multiple sources through includes
203: (12)                       root (bool):
204: (16)                           If this shader source is the root shader (Not an include)
205: (8)                    """
206: (8)                    self._id = id
207: (8)                    self._root = root
208: (8)                    self._source_list = [
209: (12)                       self
210: (8)                    ]  # List of sources this shader consists of (original source + includes)
211: (8)                    self._type = shader_type
212: (8)                    self._name = name
213: (8)                    self._defines = {} if defines is None else defines
214: (8)                    if root:
215: (12)                       source = source.strip()
216: (8)                    self._lines = source.split("\n")
217: (8)                    if self._root and not self._lines[0].startswith("#version"):
218: (12)                       self.print()
219: (12)                       raise ShaderError(
220: (16)                           f"Missing #version in {self._name}. A version must be defined in the first line"
221: (12)                       )
222: (8)                    self.apply_defines(self._defines)
223: (8)                    if self._root:
224: (12)                       self._lines.insert(1, f"#define {self._type} 1")
225: (12)                       self._lines.insert(2, "#line 2")
226: (4)                @property
227: (4)                def id(self) -> int:
228: (8)                    """int: The shader number/id"""
229: (8)                    return self._id
230: (4)                @property
231: (4)                def source(self) -> str:
232: (8)                    """str: The source lines as a string"""
233: (8)                    return "\n".join(self._lines)
234: (4)                @property
235: (4)                def source_list(self) -> list["ShaderSource"]:
236: (8)                    """list[ShaderSource]: List of all shader sources"""
237: (8)                    return self._source_list
238: (4)                @property
239: (4)                def name(self) -> Optional[str]:
240: (8)                    """str: a path or name for this shader"""
241: (8)                    return self._name
242: (4)                @property
243: (4)                def lines(self) -> list[str]:
244: (8)                    """list[str]: The lines in this shader"""
245: (8)                    return self._lines
246: (4)                @property
247: (4)                def line_count(self) -> int:
248: (8)                    """int: Number of lines in this source (stripped)"""
249: (8)                    return len(self._lines)
250: (4)                @property
251: (4)                def defines(self) -> dict[str, str]:
252: (8)                    """dict: Defines configured for this shader"""
253: (8)                    return self._defines
254: (4)                def handle_includes(
255: (8)                    self, load_source_func: Callable[[Any], Any], depth: int = 0, source_id: int = 0
256: (4)                ) -> None:
257: (8)                    """Inject includes into the shader source.
258: (8)                    This happens recursively up to a max level in case the users has
259: (8)                    circular includes. We also build up a list of all the included
260: (8)                    sources in the root shader.
261: (8)                    Args:
262: (12)                       load_source_func (func): A function for finding and loading a
```

```
source
263: (12)                       depth (int): The current include depth (increase by 1 for every
call)
264: (8)                    """
265: (8)                    if depth > 100:
266: (12)                       raise ShaderError(
267: (16)                           "Reaching an include depth of 100. You probably have circular
includes"
268: (12)                       )
269: (8)                    current_id = source_id
270: (8)                    while True:
271: (12)                       for nr, line in enumerate(self._lines):
272: (16)                           line = line.strip()
273: (16)                           if line.startswith("#include"):
274: (20)                               match = re.search(r'#include\s+"?([^"]+)', line)
275: (20)                               if match is None:
276: (24)                                   raise ShaderError(
277: (28)                                       f"Could not match '#include\\s+\"?([^\"]+)' in
line {line}"
278: (24)                                   )
279: (20)                               path = match[1]
280: (20)                               current_id += 1
281: (20)                               _, source = load_source_func(path)
282: (20)                               source = ShaderSource(
283: (24)                                   None,
284: (24)                                   path,
285: (24)                                   source,
286: (24)                                   defines=self._defines,
287: (24)                                   id=current_id,
288: (24)                                   root=False,
289: (20)                               )
290: (20)                               source.handle_includes(load_source_func, depth=depth + 1,
source_id=current_id)
291: (20)                               self._lines = self.lines[:nr] + source.lines +
self.lines[nr + 1 :]
292: (20)                               self._source_list += source.source_list
293: (20)                               current_id = self._source_list[-1].id
294: (20)                               break
295: (12)                       else:
296: (16)                           break
297: (4)                def apply_defines(self, defines: dict[str, str]) -> None:
298: (8)                    """Apply the configured define values"""
299: (8)                    if not defines:
300: (12)                       return
301: (8)                    for nr, line in enumerate(self._lines):
302: (12)                       line = line.strip()
303: (12)                       if line.startswith("#define"):
304: (16)                           try:
305: (20)                               name = line.split()[1]
306: (20)                               value = defines.get(name)
307: (20)                               if not value:
308: (24)                                   continue
309: (20)                               self.lines[nr] = f"#define {name} {value}"
310: (16)                           except IndexError:
311: (20)                               pass
312: (4)                def find_out_attribs(self) -> list[str]:
313: (8)                    """
314: (8)                    Get all out attributes in the shader source.
315: (8)                    Returns:
316: (12)                       list[str]: List of out attribute names
317: (8)                    """
318: (8)                    names = []
319: (8)                    for line in self.lines:
320: (12)                       res = re.match(r"(layout(.+)\))?(\s+)?(out)(\s+)(\w+)(\s+)(\w+)",
line.strip())
321: (12)                       if res:
322: (16)                           names.append(res.groups()[-1])
323: (8)                    return names
324: (4)                def print(self) -> None:
```

```
(8)             """Print the shader lines (for debugging)"""
(8)             print(f"---[ START {self.name} ]---")
(8)             for i, line in enumerate(self.lines):
(12)                print(f"{str(i).zfill(3)}: {line}")
(8)             print("---[ END {self.name} ]---")
(4)         def __repr__(self) -> str:
(8)             return f"<ShaderSource: {self.name} id={self.id}>"
(0)     class ShaderError(Exception):
(4)         """Generic shader related error"""
(0)     class ReloadableProgram:
(4)         """
(4)         Programs we want to be reloadable must be created with this wrapper.
(4)         """
(4)         def __init__(self, meta: ProgramDescription, program: moderngl.Program):
(8)             """
(8)             Create a shader using either a file path or a name.
(8)             Args:
(12)                meta: The program meta
(12)                program: The program instance
(8)             """
(8)             self.program = program
(8)             self.meta = meta
(4)         @property
(4)         def name(self) -> Optional[str]:
(8)             return self.meta.path or self.meta.vertex_shader
(4)         @property
(4)         def _members(self) -> dict[Any, Any]:
(8)             return self.program._members
(4)         @property
(4)         def ctx(self) -> moderngl.Context:
(8)             return self.program.ctx
(4)         def __getitem__(
(8)             self, key: Any
(4)         ) -> Union[
(8)             moderngl.Uniform,
(8)             moderngl.UniformBlock,
(8)             moderngl.Subroutine,
(8)             moderngl.Attribute,
(8)             moderngl.Varying,
(4)         ]:
(8)             return self.program[key]
(4)         def get(self, key: Any, default: Any) -> Any:
(8)             return self.program.get(key, default)
(4)         @property
(4)         def extra(self) -> Any:
(8)             return self.program.extra
(4)         @property
(4)         def mglo(self) -> moderngl.Program:
(8)             """The ModernGL Program object"""
(8)             return self.program.mglo
(4)         @property
(4)         def glo(self) -> int:
(8)             """
(8)             int: The internal OpenGL object.
(8)             This values is provided for debug purposes only.
(8)             """
(8)             return self.program.glo
(4)         @property
(4)         def subroutines(self) -> tuple[str, ...]:
(8)             """
(8)             tuple: The subroutine uniforms.
(8)             """
(8)             return self.program.subroutines
(4)         @property
(4)         def geometry_input(self) -> int:
(8)             """
(8)             int: The geometry input primitive.
(8)             The GeometryShader's input primitive if the GeometryShader exists.
(8)             The geometry input primitive will be used for validation.
```

```
394: (8)                    """
395: (8)                    return self.program.geometry_input
396: (4)                @property
397: (4)                def geometry_output(self) -> int:
398: (8)                    """
399: (8)                    int: The geometry output primitive.
400: (8)                    The GeometryShader's output primitive if the GeometryShader exists.
401: (8)                    """
402: (8)                    return self.program.geometry_output
403: (4)                @property
404: (4)                def geometry_vertices(self) -> int:
405: (8)                    """
406: (8)                    int: The maximum number of vertices that
407: (8)                    the geometry shader will output.
408: (8)                    """
409: (8)                    return self.program.geometry_vertices
410: (4)                def __repr__(self) -> str:
411: (8)                    return f"<ReloadableProgram: {self.name} id={self.glo}>"

----------------------------------------

File 73 - __init__.py:

1: (0)

----------------------------------------

File 74 - __init__.py:

1: (0)

----------------------------------------

File 75 - __init__.py:

1: (0)

----------------------------------------

File 76 - separate.py:

1: (0)              import logging
2: (0)              from pathlib import Path
3: (0)              from typing import Optional, Union
4: (0)              import moderngl
5: (0)              from moderngl_window.exceptions import ImproperlyConfigured
6: (0)              from moderngl_window.loaders.base import BaseLoader
7: (0)              from moderngl_window.opengl import program
8: (0)              logger = logging.getLogger(__name__)
9: (0)              class Loader(BaseLoader):
10: (4)                 kind = "separate"
11: (4)                 meta: program.ProgramDescription
12: (4)                 def load(
13: (8)                     self,
14: (4)                 ) -> Union[moderngl.Program, moderngl.ComputeShader,
program.ReloadableProgram]:
15: (8)                     """Loads a shader program were each shader is a separate file.
16: (8)                     This detected and dictated by the ``kind`` in the
``ProgramDescription``.
17: (8)                     Returns:
18: (12)                        moderngl.Program: The Program instance
19: (8)                     """
20: (8)                     prog: Union[moderngl.Program, moderngl.ComputeShader,
program.ReloadableProgram]
21: (8)                     vs_source = self._load_shader("vertex", self.meta.vertex_shader)
22: (8)                     geo_source = self._load_shader("geometry", self.meta.geometry_shader)
23: (8)                     fs_source = self._load_shader("fragment", self.meta.fragment_shader)
24: (8)                     tc_source = self._load_shader("tess_control",
self.meta.tess_control_shader)
```

```
25: (8)                     te_source = self._load_shader("tess_evaluation",
self.meta.tess_evaluation_shader)
26: (8)                     cs_source = self._load_shader("compute", self.meta.compute_shader)
27: (8)                     if vs_source:
28: (12)                        shaders = program.ProgramShaders.from_separate(
29: (16)                            self.meta,
30: (16)                            vs_source,
31: (16)                            geometry_source=geo_source,
32: (16)                            fragment_source=fs_source,
33: (16)                            tess_control_source=tc_source,
34: (16)                            tess_evaluation_source=te_source,
35: (12)                        )
36: (12)                        shaders.handle_includes(self._load_source)
37: (12)                        prog = shaders.create()
38: (12)                        if self.meta.reloadable:
39: (16)                            self.meta.reloadable = False
40: (16)                            prog = program.ReloadableProgram(self.meta, prog)
41: (8)                     elif cs_source:
42: (12)                        shaders = program.ProgramShaders.compute_shader(self.meta,
cs_source)
43: (12)                        shaders.handle_includes(self._load_source)
44: (12)                        prog = shaders.create_compute_shader()
45: (8)                     else:
46: (12)                        raise ImproperlyConfigured("Cannot find a shader source to load")
47: (8)                     return prog
48: (4)                 def _load_shader(self, shader_type: str, path: Optional[str]) ->
Optional[str]:
49: (8)                     """Load a single shader source"""
50: (8)                     if path is not None:
51: (12)                        resolved_path = self.find_program(path)
52: (12)                        if not resolved_path:
53: (16)                            raise ImproperlyConfigured("Cannot find {} shader
'{}'".format(shader_type, path))
54: (12)                        logger.info("Loading: %s", resolved_path)
55: (12)                        with open(str(resolved_path), "r") as fd:
56: (16)                            return fd.read()
57: (8)                     return None
58: (4)                 def _load_source(self, path: Union[Path, str]) -> tuple[Path, str]:
59: (8)                     """Finds and loads a single source file.
60: (8)                     Args:
61: (12)                        path: Path to resource
62: (8)                     Returns:
63: (12)                        tuple[resolved_path, source]: The resolved path and the source
64: (8)                     """
65: (8)                     resolved_path = self.find_program(path)
66: (8)                     if resolved_path is None:
67: (12)                        raise ImproperlyConfigured("Cannot find program
'{}'".format(path))
68: (8)                     logger.info("Loading: %s", path)
69: (8)                     with open(str(resolved_path), "r") as fd:
70: (12)                        return resolved_path, fd.read()

----------------------------------------

File 77 - __init__.py:

1: (0)

----------------------------------------

File 78 - __init__.py:

1: (0)

----------------------------------------

File 79 - __init__.py:

1: (0)
```

```
----------------------------------------

File 80 - __init__.py:

1: (0)              from .base import ResourceDescription as ResourceDescription
2: (0)              from .data import DataDescription as DataDescription
3: (0)              from .program import ProgramDescription as ProgramDescription
4: (0)              from .scene import SceneDescription as SceneDescription
5: (0)              from .texture import TextureDescription as TextureDescription

----------------------------------------

File 81 - __init__.py:

1: (0)

----------------------------------------

File 82 - wavefront.py:

1: (0)              import io
2: (0)              import logging
3: (0)              import os
4: (0)              from pathlib import Path
5: (0)              import moderngl
6: (0)              import numpy
7: (0)              import pywavefront
8: (0)              from pywavefront import cache
9: (0)              from pywavefront.obj import ObjParser
10: (0)             from moderngl_window import resources
11: (0)             from moderngl_window.exceptions import ImproperlyConfigured
12: (0)             from moderngl_window.geometry.attributes import AttributeNames
13: (0)             from moderngl_window.loaders.base import BaseLoader
14: (0)             from moderngl_window.meta import SceneDescription, TextureDescription
15: (0)             from moderngl_window.opengl.vao import VAO
16: (0)             from moderngl_window.resources.decorators import texture_dirs
17: (0)             from moderngl_window.scene import Material, MaterialTexture, Mesh, Node, Scene
18: (0)             logger = logging.getLogger(__name__)
19: (0)             def translate_buffer_format(
20: (4)                 vertex_format: str, attr_names: AttributeNames
21: (0)             ) -> tuple[str, list[str], list[tuple[str, str, int]]]:
22: (4)                 """Translate the buffer format"""
23: (4)                 buffer_format = []
24: (4)                 attributes = []
25: (4)                 mesh_attributes = []
26: (4)                 if "T2F" in vertex_format:
27: (8)                     buffer_format.append("2f")
28: (8)                     attributes.append(attr_names.TEXCOORD_0)
29: (8)                     mesh_attributes.append(("TEXCOORD_0", attr_names.TEXCOORD_0, 2))
30: (4)                 if "C3F" in vertex_format:
31: (8)                     buffer_format.append("3f")
32: (8)                     attributes.append(attr_names.COLOR_0)
33: (8)                     mesh_attributes.append(("COLOR_0", attr_names.COLOR_0, 3))
34: (4)                 if "N3F" in vertex_format:
35: (8)                     buffer_format.append("3f")
36: (8)                     attributes.append(attr_names.NORMAL)
37: (8)                     mesh_attributes.append(("NORMAL", attr_names.NORMAL, 3))
38: (4)                 buffer_format.append("3f")
39: (4)                 attributes.append(attr_names.POSITION)
40: (4)                 mesh_attributes.append(("POSITION", attr_names.POSITION, 3))
41: (4)                 return " ".join(buffer_format), attributes, mesh_attributes
42: (0)             class VAOCacheLoader(cache.CacheLoader):
43: (4)                 """Load geometry data directly into vaos"""
44: (4)                 attr_names: AttributeNames
45: (4)                 def load_vertex_buffer(
46: (8)                     self, fd: io.TextIOWrapper, material: pywavefront.material.Material,
length: int
47: (4)                 ) -> None:
```

```
(8)             buffer_format, attributes, mesh_attributes = translate_buffer_format(
(12)                material.vertex_format, self.attr_names
(8)             )
(8)             vao = VAO(material.name, mode=moderngl.TRIANGLES)
(8)             vao.buffer(fd.read(length), buffer_format, attributes)
(8)             setattr(material, "vao", vao)
(8)             setattr(material, "buffer_format", buffer_format)
(8)             setattr(material, "attributes", attributes)
(8)             setattr(material, "mesh_attributes", mesh_attributes)
(0)     ObjParser.cache_loader_cls = VAOCacheLoader
(0)     class Loader(BaseLoader):
(4)         """Load wavefront/obj files"""
(4)         kind = "wavefront"
(4)         file_extensions = [
(8)             [".obj"],
(8)             [".obj", ".gz"],
(8)             [".bin"],
(4)         ]
(4)         meta: SceneDescription
(4)         def __init__(self, meta: SceneDescription):
(8)             super().__init__(meta)
(4)         def load(self) -> Scene:
(8)             """Loads a wavefront/obj file including materials and textures
(8)             Returns:
(12)                Scene: The Scene instance
(8)             """
(8)             path = self.find_scene(Path(self.meta.path if self.meta.path is not None else ""))
(8)             logger.info("loading %s", path)
(8)             if not path:
(12)                raise ImproperlyConfigured("Scene '{}' not found".format(self.meta.path))
(8)             if path.suffix == ".bin":
(12)                path = path.parent / path.stem
(8)             VAOCacheLoader.attr_names = self.meta.attr_names
(8)             data = pywavefront.Wavefront(str(path), create_materials=True, cache=self.meta.cache)
(8)             scene = Scene(
(12)                self.meta.resolved_path.as_posix() if self.meta.resolved_path is not None else ""
(8)             )
(8)             texture_cache: dict[str, pywavefront.material.Material] = {}
(8)             for _, mat in data.materials.items():
(12)                mesh = Mesh(mat.name)
(12)                if mat.vertices:
(16)                    buffer_format, attributes, mesh_attributes = translate_buffer_format(
(20)                        mat.vertex_format, self.meta.attr_names
(16)                    )
(16)                    vbo = numpy.array(mat.vertices, dtype="f4")
(16)                    vao = VAO(mat.name, mode=moderngl.TRIANGLES)
(16)                    vao.buffer(vbo, buffer_format, attributes)
(16)                    mesh.vao = vao
(16)                    for attrs in mesh_attributes:
(20)                        mesh.add_attribute(*attrs)
(12)                elif hasattr(mat, "vao"):
(16)                    mesh = Mesh(mat.name)
(16)                    mesh.vao = mat.vao
(16)                    for attrs in mat.mesh_attributes:
(20)                        mesh.add_attribute(*attrs)
(12)                else:
(16)                    continue
(12)                scene.meshes.append(mesh)
(12)                mesh.material = Material(mat.name)
(12)                scene.materials.append(mesh.material)
(12)                mesh.material.color = mat.diffuse
(12)                if mat.texture:
(16)                    texture = texture_cache.get(mat.texture.path)
(16)                    if not texture:
```

```
112: (20)                               rel_path = os.path.relpath(mat.texture.find(),
str(path.parent))
113: (20)                               logger.info("Loading: %s", rel_path)
114: (20)                               with texture_dirs([path.parent]):
115: (24)                                   texture = resources.textures.load(
116: (28)                                       TextureDescription(
117: (32)                                           label=rel_path,
118: (32)                                           path=rel_path,
119: (32)                                           mipmap=True,
120: (32)                                           anisotropy=16.0,
121: (28)                                       )
122: (24)                                   )
123: (20)                               texture_cache[rel_path] = texture
124: (16)                           mesh.material.mat_texture = MaterialTexture(
125: (20)                               texture=texture,
126: (20)                               sampler=None,
127: (16)                           )
128: (12)                       node = Node(mesh=mesh)
129: (12)                       scene.root_nodes.append(node)
130: (8)                    scene.prepare()
131: (8)                    return scene

----------------------------------------

File 83 - projection.py:

1: (0)              from typing import Optional
2: (0)              import glm
3: (0)              class Projection3D:
4: (4)                  """3D Projection"""
5: (4)                  def __init__(
6: (8)                      self, aspect_ratio: float = 16 / 9, fov: float = 75.0, near: float =
1.0, far: float = 100.0
7: (4)                  ):
8: (8)                      """Create a 3D projection
9: (8)                      Keyword Args:
10: (12)                        aspect_ratio (float): Aspect ratio
11: (12)                        fov (float): Field of view
12: (12)                        near (float): Near plane value
13: (12)                        far (float): Far plane value
14: (8)                      """
15: (8)                      self._aspect_ratio = aspect_ratio
16: (8)                      self._fov = fov
17: (8)                      self._near = near
18: (8)                      self._far = far
19: (8)                      self._matrix = glm.mat4(0)
20: (8)                      self._matrix_bytes = bytes(0)
21: (8)                      self.update()
22: (4)                  @property
23: (4)                  def aspect_ratio(self) -> float:
24: (8)                      """float: The projection's aspect ratio"""
25: (8)                      return self._aspect_ratio
26: (4)                  @property
27: (4)                  def fov(self) -> float:
28: (8)                      """float: Current field of view"""
29: (8)                      return self._fov
30: (4)                  @property
31: (4)                  def near(self) -> float:
32: (8)                      """float: Current near plane value"""
33: (8)                      return self._near
34: (4)                  @property
35: (4)                  def far(self) -> float:
36: (8)                      """float : Current far plane value"""
37: (8)                      return self._far
38: (4)                  @property
39: (4)                  def matrix(self) -> glm.mat4:
40: (8)                      """glm.mat4x4: Current projection matrix"""
41: (8)                      return self._matrix
42: (4)                  def update(
        self,
        aspect_ratio: Optional[float] = None,
        fov: Optional[float] = None,
        near: Optional[float] = None,
        far: Optional[float] = None,
    ) -> None:
        """Update the projection matrix
        Keyword Args:
            aspect_ratio (float): Aspect ratio
            fov (float): Field of view
            near (float): Near plane value
            far (float): Far plane value
        """
        if aspect_ratio is not None:
            self._aspect_ratio = aspect_ratio
        if fov is not None:
            self._fov = fov
        if near is not None:
            self._near = near
        if far is not None:
            self._far = far
        self._matrix = glm.perspective(
            glm.radians(self._fov), self._aspect_ratio, self._near, self._far
        )
        self._matrix_bytes = self._matrix.to_bytes()
    def tobytes(self) -> bytes:
        """Get the byte representation of the projection matrix
        Returns:
            bytes: byte representation of the projection matrix
        """
        return self._matrix_bytes
    @property
    def projection_constants(self) -> tuple[float, float]:
        """
        (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.
        """
        return (
            self._far / (self._far - self._near),
            (self._far * self._near) / (self._near - self._far),
        )
```

File 84 - imgui_bundle.py:

```
import ctypes
import moderngl
from imgui_bundle import imgui
from imgui_bundle.python_backends import compute_fb_scale
class ModernglWindowMixin:
    io: imgui.IO
    def resize(self, width: int, height: int):
        self.io.display_size = self.wnd.size
        self.io.display_framebuffer_scale = compute_fb_scale(self.wnd.size, self.wnd.buffer_size)
    def key_event(self, key, action, modifiers):
        keys = self.wnd.keys
        if key in self.REVERSE_KEYMAP:
            down = action == keys.ACTION_PRESS
            self.io.add_key_event(self.REVERSE_KEYMAP[key], down=down)
    def _mouse_pos_viewport(self, x, y):
        """Make sure mouse coordinates are correct with black borders"""
        return (
            int(x - (self.wnd.width - self.wnd.viewport_width / self.wnd.pixel_ratio) / 2),
            int(y - (self.wnd.height - self.wnd.viewport_height / self.wnd.pixel_ratio) / 2),
20: (8)                     )
21: (4)                 def mouse_position_event(self, x, y, dx, dy):
22: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
23: (4)                 def mouse_drag_event(self, x, y, dx, dy):
24: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
25: (8)                     if self.wnd.mouse_states.left:
26: (12)                        self.io.mouse_down[0] = 1
27: (8)                     if self.wnd.mouse_states.middle:
28: (12)                        self.io.mouse_down[2] = 1
29: (8)                     if self.wnd.mouse_states.right:
30: (12)                        self.io.mouse_down[1] = 1
31: (4)                 def mouse_scroll_event(self, x_offset, y_offset):
32: (8)                     self.io.mouse_wheel = y_offset
33: (4)                 def mouse_press_event(self, x, y, button):
34: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
35: (8)                     if button == self.wnd.mouse.left:
36: (12)                        self.io.mouse_down[0] = 1
37: (8)                     if button == self.wnd.mouse.middle:
38: (12)                        self.io.mouse_down[2] = 1
39: (8)                     if button == self.wnd.mouse.right:
40: (12)                        self.io.mouse_down[1] = 1
41: (4)                 def mouse_release_event(self, x: int, y: int, button: int):
42: (8)                     self.io.mouse_pos = self._mouse_pos_viewport(x, y)
43: (8)                     if button == self.wnd.mouse.left:
44: (12)                        self.io.mouse_down[0] = 0
45: (8)                     if button == self.wnd.mouse.middle:
46: (12)                        self.io.mouse_down[2] = 0
47: (8)                     if button == self.wnd.mouse.right:
48: (12)                        self.io.mouse_down[1] = 0
49: (4)                 def unicode_char_entered(self, char):
50: (8)                     io = imgui.get_io()
51: (8)                     io.add_input_character(ord(char))
52: (0)             class BaseOpenGLRenderer(object):
53: (4)                 def __init__(self):
54: (8)                     if not imgui.get_current_context():
55: (12)                        raise RuntimeError(
56: (16)                            "No valid ImGui context. Use imgui.create_context() first
and/or "
57: (16)                            "imgui.set_current_context()."
58: (12)                        )
59: (8)                     self.io = imgui.get_io()
60: (8)                     self._font_texture = None
61: (8)                     self.io.delta_time = 1.0 / 60.0
62: (8)                     self._create_device_objects()
63: (8)                     self.refresh_font_texture()
64: (4)                 def render(self, draw_data):
65: (8)                     raise NotImplementedError
66: (4)                 def refresh_font_texture(self):
67: (8)                     raise NotImplementedError
68: (4)                 def _create_device_objects(self):
69: (8)                     raise NotImplementedError
70: (4)                 def _invalidate_device_objects(self):
71: (8)                     raise NotImplementedError
72: (4)                 def shutdown(self):
73: (8)                     self._invalidate_device_objects()
74: (0)             class ModernGLRenderer(BaseOpenGLRenderer):
75: (4)                 VERTEX_SHADER_SRC = """
76: (8)                     uniform mat4 ProjMtx;
77: (8)                     in vec2 Position;
78: (8)                     in vec2 UV;
79: (8)                     in vec4 Color;
80: (8)                     out vec2 Frag_UV;
81: (8)                     out vec4 Frag_Color;
82: (8)                     void main() {
83: (12)                        Frag_UV = UV;
84: (12)                        Frag_Color = Color;
85: (12)                        gl_Position = ProjMtx * vec4(Position.xy, 0, 1);
86: (8)                     }
87: (4)                 """
```

```
    FRAGMENT_SHADER_SRC = """
        uniform sampler2D Texture;
        in vec2 Frag_UV;
        in vec4 Frag_Color;
        out vec4 Out_Color;
        void main() {
            Out_Color = (Frag_Color * texture(Texture, Frag_UV.st));
        }
    """
    def __init__(self, *args, **kwargs):
        self._prog = None
        self._fbo = None
        self._font_texture = None
        self._vertex_buffer = None
        self._index_buffer = None
        self._vao = None
        self._textures = {}
        self.wnd = kwargs.get("wnd")
        self.ctx: moderngl.Context = (
            self.wnd.ctx if self.wnd and self.wnd.ctx else kwargs.get("ctx")
        )
        if not self.ctx:
            raise RuntimeError("Missing moderngl context")
        super().__init__()
        if hasattr(self, "wnd") and self.wnd:
            self.resize(*self.wnd.buffer_size)
        elif "display_size" in kwargs:
            self.io.display_size = kwargs.get("display_size")
    def register_texture(self, texture: moderngl.Texture):
        """Make the imgui renderer aware of the texture"""
        self._textures[texture.glo] = texture
    def remove_texture(self, texture: moderngl.Texture):
        """Remove the texture from the imgui renderer"""
        del self._textures[texture.glo]
    def refresh_font_texture(self):
        font_matrix = self.io.fonts.get_tex_data_as_rgba32()
        width = font_matrix.shape[1]
        height = font_matrix.shape[0]
        pixels = font_matrix.data
        if self._font_texture:
            self.remove_texture(self._font_texture)
            self._font_texture.release()
        self._font_texture = self.ctx.texture((width, height), 4, data=pixels)
        self.register_texture(self._font_texture)
        self.io.fonts.tex_id = self._font_texture.glo
        self.io.fonts.clear_tex_data()
    def _create_device_objects(self):
        self._prog = self.ctx.program(
            vertex_shader=self.VERTEX_SHADER_SRC,
            fragment_shader=self.FRAGMENT_SHADER_SRC,
        )
        self.projMat = self._prog["ProjMtx"]
        self._prog["Texture"].value = 0
        self._vertex_buffer = self.ctx.buffer(reserve=imgui.VERTEX_SIZE * 65536)
        self._index_buffer = self.ctx.buffer(reserve=imgui.INDEX_SIZE * 65536)
        self._vao = self.ctx.vertex_array(
            self._prog,
            [(self._vertex_buffer, "2f 2f 4f1", "Position", "UV", "Color")],
            index_buffer=self._index_buffer,
            index_element_size=imgui.INDEX_SIZE,
        )
    def render(self, draw_data: imgui.ImDrawData):
        io = self.io
        display_width, display_height = io.display_size
        fb_width = int(display_width * io.display_framebuffer_scale[0])
        fb_height = int(display_height * io.display_framebuffer_scale[1])
        if fb_width == 0 or fb_height == 0:
            return
```

```
(8)                    self.projMat.value = (
(12)                       2.0 / display_width,
(12)                       0.0,
(12)                       0.0,
(12)                       0.0,
(12)                       0.0,
(12)                       2.0 / -display_height,
(12)                       0.0,
(12)                       0.0,
(12)                       0.0,
(12)                       0.0,
(12)                       -1.0,
(12)                       0.0,
(12)                       -1.0,
(12)                       1.0,
(12)                       0.0,
(12)                       1.0,
(8)                    )
(8)                    draw_data.scale_clip_rects(imgui.ImVec2(*io.display_framebuffer_scale))
(8)                    self.ctx.enable_only(moderngl.BLEND)
(8)                    self.ctx.blend_equation = moderngl.FUNC_ADD
(8)                    self.ctx.blend_func = moderngl.SRC_ALPHA, moderngl.ONE_MINUS_SRC_ALPHA
(8)                    self._font_texture.use()
(8)                    for commands in draw_data.cmd_lists:
(12)                       vtx_type = ctypes.c_byte * commands.vtx_buffer.size() * imgui.VERTEX_SIZE
(12)                       idx_type = ctypes.c_byte * commands.idx_buffer.size() * imgui.INDEX_SIZE
(12)                       vtx_arr = (vtx_type).from_address(commands.vtx_buffer.data_address())
(12)                       idx_arr = (idx_type).from_address(commands.idx_buffer.data_address())
(12)                       self._vertex_buffer.write(vtx_arr)
(12)                       self._index_buffer.write(idx_arr)
(12)                       idx_pos = 0
(12)                       for command in commands.cmd_buffer:
(16)                           texture = self._textures.get(command.texture_id)
(16)                           if texture is None:
(20)                               raise ValueError(
(24)                                   (
(28)                                       "Texture {} is not registered. Please add to renderer using "
(28)                                       "register_texture(..). "
(28)                                       "Current textures: {}".format(command.texture_id, list(self._textures))
(24)                                   )
(20)                               )
(16)                           texture.use(0)
(16)                           x, y, z, w = command.clip_rect
(16)                           self.ctx.scissor = int(x), int(fb_height - w), int(z - x), int(w - y)
(16)                           self._vao.render(moderngl.TRIANGLES, vertices=command.elem_count, first=idx_pos)
(16)                           idx_pos += command.elem_count
(8)                    self.ctx.scissor = None
(4)                def _invalidate_device_objects(self):
(8)                    if self._font_texture:
(12)                       self._font_texture.release()
(8)                    if self._vertex_buffer:
(12)                       self._vertex_buffer.release()
(8)                    if self._index_buffer:
(12)                       self._index_buffer.release()
(8)                    if self._vao:
(12)                       self._vao.release()
(8)                    if self._prog:
(12)                       self._prog.release()
(8)                    self.io.fonts.tex_id = 0
(8)                    self._font_texture = None
```

```
216: (0)            class ModernglWindowRenderer(ModernGLRenderer, ModernglWindowMixin):
217: (4)                def __init__(self, window):
218: (8)                    super().__init__(wnd=window)
219: (8)                    self.wnd = window
220: (8)                    self._init_key_maps()
221: (8)                    self.io.display_size = self.wnd.size
222: (8)                    self.io.display_framebuffer_scale = self.wnd.pixel_ratio,
self.wnd.pixel_ratio
223: (4)                def _init_key_maps(self):
224: (8)                    keys = self.wnd.keys
225: (8)                    self.REVERSE_KEYMAP = {
226: (12)                       keys.TAB: imgui.Key.tab,
227: (12)                       keys.LEFT: imgui.Key.left_arrow,
228: (12)                       keys.RIGHT: imgui.Key.right_arrow,
229: (12)                       keys.UP: imgui.Key.up_arrow,
230: (12)                       keys.DOWN: imgui.Key.down_arrow,
231: (12)                       keys.PAGE_UP: imgui.Key.page_up,
232: (12)                       keys.PAGE_DOWN: imgui.Key.page_down,
233: (12)                       keys.HOME: imgui.Key.home,
234: (12)                       keys.END: imgui.Key.end,
235: (12)                       keys.DELETE: imgui.Key.delete,
236: (12)                       keys.SPACE: imgui.Key.space,
237: (12)                       keys.BACKSPACE: imgui.Key.backspace,
238: (12)                       keys.ENTER: imgui.Key.enter,
239: (12)                       keys.ESCAPE: imgui.Key.escape,
240: (8)                    }

----------------------------------------

File 85 - base.py:

1: (0)              """
2: (0)              Base registry class
3: (0)              """
4: (0)              import inspect
5: (0)              from functools import lru_cache
6: (0)              from typing import Any, Generator
7: (0)              from moderngl_window.conf import settings
8: (0)              from moderngl_window.exceptions import ImproperlyConfigured
9: (0)              from moderngl_window.loaders.base import BaseLoader
10: (0)             from moderngl_window.meta.base import ResourceDescription
11: (0)             from moderngl_window.utils.module_loading import import_string
12: (0)             class BaseRegistry:
13: (4)                 """Base class for all resource pools"""
14: (4)                 settings_attr = ""
15: (4)                 """str: The name of the attribute in
:py:class:`~moderngl_window.conf.Settings`
16: (4)                 containting a list of loader classes.
17: (4)                 """
18: (4)                 def __init__(self) -> None:
19: (8)                     """Initialize internal attributes"""
20: (8)                     self._resources: list[ResourceDescription] = []
21: (4)                 @property
22: (4)                 def count(self) -> int:
23: (8)                     """int: The number of resource descriptions added.
24: (8)                     This is only relevant when using `add` and `load_pool`.
25: (8)                     """
26: (8)                     return len(self._resources)
27: (4)                 @property
28: (4)                 def loaders(self) -> Generator[type[BaseLoader], None, None]:
29: (8)                     """Generator: Loader classes for this resource type"""
30: (8)                     for loader in getattr(settings, self.settings_attr):
31: (12)                        yield self._loader_cls(loader)
32: (4)                 @lru_cache(maxsize=None)
33: (4)                 def _loader_cls(self, python_path: str) -> type[BaseLoader]:
34: (8)                     cls = import_string(python_path)
35: (8)                     assert issubclass(cls, BaseLoader), f"{python_path} does not lead to a
Loader"
36: (8)                     return cls
```

```
37: (4)                 def load(self, meta: ResourceDescription) -> Any:
38: (8)                     """
39: (8)                     Loads a resource using the configured finders and loaders.
40: (8)                     Args:
41: (12)                        meta (ResourceDescription): The resource description
42: (8)                     """
43: (8)                     self._check_meta(meta)
44: (8)                     self.resolve_loader(meta)
45: (8)                     cls = meta.loader_cls(meta)
46: (8)                     assert cls is not None, f"Could not load {meta}, no arributes named
'loader_cls'"
47: (8)                     return cls.load()
48: (4)                 def add(self, meta: ResourceDescription) -> None:
49: (8)                     """
50: (8)                     Adds a resource description without loading it.
51: (8)                     The resource is loaded and returned when ``load_pool()`` is called.
52: (8)                     Args:
53: (12)                        meta (ResourceDescription): The resource description
54: (8)                     """
55: (8)                     self._check_meta(meta)
56: (8)                     self.resolve_loader(meta)
57: (8)                     self._resources.append(meta)
58: (4)                 def load_pool(self) -> Generator[tuple[ResourceDescription, Any], None,
None]:
59: (8)                     """
60: (8)                     Loads all the data files using the configured finders.
61: (8)                     This is only relevant when resource have been added to this
62: (8)                     pool using ``add()``.
63: (8)                     Returns:
64: (12)                        Generator of (meta, resource) tuples
65: (8)                     """
66: (8)                     for meta in self._resources:
67: (12)                        resource = self.load(meta)
68: (12)                        yield meta, resource
69: (8)                     self._resources = []
70: (4)                 def resolve_loader(self, meta: ResourceDescription) -> None:
71: (8)                     """
72: (8)                     Attempts to assign a loader class to a ResourceDescription.
73: (8)                     Args:
74: (12)                        meta (:py:class:`~moderngl_window.meta.base.ResourceDescription`):
75: (12)                        The resource description instance
76: (8)                     """
77: (8)                     if meta.kind:
78: (12)                        for loader_cls in self.loaders:
79: (16)                            if loader_cls.kind == meta.kind:
80: (20)                                meta.loader_cls = loader_cls
81: (20)                                return
82: (12)                        raise ImproperlyConfigured(
83: (16)                            "Resource has invalid loader kind '{}': {}\nAvailable loaders:
{}".format(
84: (20)                                meta.kind, meta, [loader.kind for loader in self.loaders]
85: (16)                            )
86: (12)                        )
87: (8)                     for loader_cls in self.loaders:
88: (12)                        if loader_cls.supports_file(meta):
89: (16)                            meta.loader_cls = loader_cls
90: (16)                            return
91: (8)                     raise ImproperlyConfigured("Could not find a loader for:
{}".format(meta))
92: (4)                 def _check_meta(self, meta: Any) -> None:
93: (8)                     """Check is the instance is a resource description
94: (8)                     Raises:
95: (12)                        ImproperlyConfigured if not a ResourceDescription instance
96: (8)                     """
97: (8)                     if inspect.isclass(type(meta)):
98: (12)                        if issubclass(meta.__class__, ResourceDescription):
99: (16)                            return
100: (8)                    raise ImproperlyConfigured(
101: (12)                       "Resource loader got type {}, not a resource
description".format(type(meta))
102: (8)                    )
```

File 86 - data.py:

```
1: (0)              """
2: (0)              Registry general data files
3: (0)              """
4: (0)              from typing import Any
5: (0)              from moderngl_window.meta import DataDescription, ResourceDescription
6: (0)              from moderngl_window.resources.base import BaseRegistry
7: (0)              class DataFiles(BaseRegistry):
8: (4)                  """Registry for requested data files"""
9: (4)                  settings_attr = "DATA_LOADERS"
10: (4)                 meta: DataDescription
11: (4)                 def load(self, meta: ResourceDescription) -> Any:
12: (8)                     """Load data file with the configured loaders.
13: (8)                     Args:
14: (12)                        meta (:py:class:`~moderngl_window.meta.data.DataDescription`): the
resource description
15: (8)                     Returns:
16: (12)                        Any: The loaded resource
17: (8)                     """
18: (8)                     return super().load(meta)
19: (0)             data = DataFiles()
```

File 87 - mesh.py:

```
1: (0)              from typing import TYPE_CHECKING, Any, Optional
2: (0)              import glm
3: (0)              import moderngl
4: (0)              from moderngl_window.opengl.vao import VAO
5: (0)              from .material import Material
6: (0)              if TYPE_CHECKING:
7: (4)                  from .programs import MeshProgram
8: (0)              class Mesh:
9: (4)                  """Mesh info and geometry"""
10: (4)                 def __init__(
11: (8)                     self,
12: (8)                     name: str,
13: (8)                     vao: Optional[VAO] = None,
14: (8)                     material: Optional[Material] = None,
15: (8)                     attributes: Optional[dict[str, Any]] = None,
16: (8)                     bbox_min: glm.vec3 = glm.vec3(),
17: (8)                     bbox_max: glm.vec3 = glm.vec3(),
18: (4)                 ) -> None:
19: (8)                     """Initialize mesh.
20: (8)                     Args:
21: (12)                        name (str): name of the mesh
22: (8)                     Keyword Args:
23: (12)                        vao (VAO): geometry
24: (12)                        material (Material): material for the mesh
25: (12)                        attributes (dict): Details info about each mesh attribute (dict)
26: (12)                        bbox_min: xyz min values
27: (12)                        bbox_max: xyz max values
28: (8)                     Attributes example::
29: (12)                        {
30: (16)                            "NORMAL": {"name": "in_normal", "components": 3, "type":
GL_FLOAT},
31: (16)                            "POSITION": {"name": "in_position", "components": 3, "type":
GL_FLOAT}
32: (12)                        }
33: (8)                     """
34: (8)                     self.name = name
35: (8)                     self.vao = vao
```

```
36: (8)                         self.material = material
37: (8)                         self.attributes = attributes or {}
38: (8)                         self.bbox_min = bbox_min
39: (8)                         self.bbox_max = bbox_max
40: (8)                         self.mesh_program: Optional["MeshProgram"] = None
41: (4)                     def draw(
42: (8)                         self,
43: (8)                         projection_matrix: Optional[glm.mat4] = None,
44: (8)                         model_matrix: Optional[glm.mat4] = None,
45: (8)                         camera_matrix: Optional[glm.mat4] = None,
46: (8)                         time: float = 0.0,
47: (4)                     ) -> None:
48: (8)                         """Draw the mesh using the assigned mesh program
49: (8)                         Keyword Args:
50: (12)                            projection_matrix (bytes): projection_matrix
51: (12)                            view_matrix (bytes): view_matrix
52: (12)                            camera_matrix (bytes): camera_matrix
53: (8)                         """
54: (8)                         if self.mesh_program is not None:
55: (12)                            assert (
56: (16)                                projection_matrix is not None
57: (12)                            ), "Can not draw, there is no projection matrix to use"
58: (12)                            assert model_matrix is not None, "Can not draw, there is no model matrix to use"
59: (12)                            assert camera_matrix is not None, "Can not draw, there is no camera matrix to use"
60: (12)                            self.mesh_program.draw(
61: (16)                                self,
62: (16)                                projection_matrix=projection_matrix,
63: (16)                                model_matrix=model_matrix,
64: (16)                                camera_matrix=camera_matrix,
65: (16)                                time=time,
66: (12)                            )
67: (4)                     def draw_bbox(
68: (8)                         self,
69: (8)                         proj_matrix: glm.mat4,
70: (8)                         model_matrix: glm.mat4,
71: (8)                         cam_matrix: glm.mat4,
72: (8)                         program: moderngl.Program,
73: (8)                         vao: VAO,
74: (4)                     ) -> None:
75: (8)                         """Renders the bounding box for this mesh.
76: (8)                         Args:
77: (12)                            proj_matrix: Projection matrix
78: (12)                            model_matrix: View/model matrix
79: (12)                            cam_matrix: Camera matrix
80: (12)                            program: The moderngl.Program rendering the bounding box
81: (12)                            vao: The vao mesh for the bounding box
82: (8)                         """
83: (8)                         program["m_proj"].write(proj_matrix.to_bytes())
84: (8)                         program["m_model"].write(model_matrix.to_bytes())
85: (8)                         program["m_cam"].write(cam_matrix.to_bytes())
86: (8)                         program["bb_min"].write(self.bbox_min.to_bytes())
87: (8)                         program["bb_max"].write(self.bbox_max.to_bytes())
88: (8)                         vao.render(program)
89: (4)                     def draw_wireframe(
90: (8)                         self, proj_matrix: glm.mat4, model_matrix: glm.mat4, program: moderngl.Program
91: (4)                     ) -> None:
92: (8)                         """Render the mesh as wireframe.
93: (8)                         proj_matrix: Projection matrix
94: (8)                         model_matrix: View/model matrix
95: (8)                         program: The moderngl.Program rendering the wireframe
96: (8)                         """
97: (8)                         assert self.vao is not None, "Can not draw the wireframe, vao is empty"
98: (8)                         program["m_proj"].write(proj_matrix.to_bytes())
99: (8)                         program["m_model"].write(model_matrix.to_bytes())
100: (8)                        self.vao.render(program)
```

```
    def add_attribute(self, attr_type: str, name: str, components: int) -> None:
        """
        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
        """
        self.attributes[attr_type] = {"name": name, "components": components}
    def calc_global_bbox(
        self, view_matrix: glm.mat4, bbox_min: Optional[glm.vec3], bbox_max: Optional[glm.vec3]
    ) -> tuple[glm.vec3, glm.vec3]:
        """Calculates the global bounding.
        Args:
            view_matrix: View matrix
            bbox_min: xyz min
            bbox_max: xyz max
        Returns:
            bbox_min, bbox_max: Combined bbox
        """
        bb1 = glm.vec4(self.bbox_min, 1.0)
        bb2 = glm.vec4(self.bbox_max, 1.0)
        bmin = view_matrix * bb1
        bmax = view_matrix * bb2
        for i in range(3):
            if bmax[i] - bmin[i] < 0:
                bmin[i], bmax[i] = bmax[i], bmin[i]
        if bbox_min is None or bbox_max is None:
            return (glm.vec3(bmin.x, bmin.y, bmin.z), glm.vec3(bmax.x, bmax.y, bmax.z))
        for i in range(3):
            bbox_min[i] = min(bbox_min[i], bmin[i])
        for i in range(3):
            bbox_max[i] = max(bbox_max[i], bmax[i])
        return bbox_min, bbox_max
    def has_normals(self) -> bool:
        """
        Returns:
            bool: Does the mesh have a normals?
        """
        return "NORMAL" in self.attributes
    def has_uvs(self, layer: int = 0) -> bool:
        """
        Returns:
            bool: Does the mesh have texture coordinates?
        """
        return "TEXCOORD_{}".format(layer) in self.attributes
```

File 88 - node.py:

```
"""
Wrapper for a loaded mesh / vao with properties
"""
from typing import Optional
import glm
import moderngl
from moderngl_window.opengl.vao import VAO
from .camera import Camera
from .mesh import Mesh
class 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.
    """
    def __init__(
        self,
17: (8)                         name: Optional[str] = None,
18: (8)                         camera: Optional[Camera] = None,
19: (8)                         mesh: Optional[Mesh] = None,
20: (8)                         matrix: Optional[glm.mat4] = None,
21: (4)                     ):
22: (8)                         """Create a node.
23: (8)                         Keyword Args:
24: (12)                            name: Name of the node
25: (12)                            camera: Camera to store in the node
26: (12)                            mesh: Mesh to store in the node
27: (12)                            matrix: The node's matrix
28: (8)                         """
29: (8)                         self._name = name
30: (8)                         self._camera = camera
31: (8)                         self._mesh = mesh
32: (8)                         self._matrix = matrix
33: (8)                         self._matrix_global = glm.mat4(1.0)
34: (8)                         self._children: list["Node"] = []
35: (4)                     @property
36: (4)                     def name(self) -> Optional[str]:
37: (8)                         """str: Get or set the node name"""
38: (8)                         return self._name
39: (4)                     @name.setter
40: (4)                     def name(self, value: str) -> None:
41: (8)                         self._name = value
42: (4)                     @property
43: (4)                     def mesh(self) -> Optional[Mesh]:
44: (8)                         """:py:class:`~moderngl_window.scene.Mesh`: The mesh if present"""
45: (8)                         return self._mesh
46: (4)                     @mesh.setter
47: (4)                     def mesh(self, value: Mesh) -> None:
48: (8)                         self._mesh = value
49: (4)                     @property
50: (4)                     def camera(self) -> Optional[Camera]:
51: (8)                         """:py:class:`~moderngl_window.scene.Camera`: The camera if present"""
52: (8)                         return self._camera
53: (4)                     @camera.setter
54: (4)                     def camera(self, value: Camera) -> None:
55: (8)                         self._camera = value
56: (4)                     @property
57: (4)                     def matrix(self) -> Optional[glm.mat4]:
58: (8)                         """glm.mat4x4: Note matrix (local)"""
59: (8)                         return self._matrix
60: (4)                     @matrix.setter
61: (4)                     def matrix(self, value: glm.mat4) -> None:
62: (8)                         self._matrix = value
63: (4)                     @property
64: (4)                     def matrix_global(self) -> Optional[glm.mat4]:
65: (8)                         """glm.matx4: The global node matrix containing transformations from
parent nodes"""
66: (8)                         return self._matrix_global
67: (4)                     @matrix_global.setter
68: (4)                     def matrix_global(self, value: glm.mat4) -> None:
69: (8)                         self._matrix_global = value
70: (4)                     @property
71: (4)                     def children(self) -> list["Node"]:
72: (8)                         """list: List of children"""
73: (8)                         return self._children
74: (4)                     def add_child(self, node: "Node") -> None:
75: (8)                         """Add a child to this node
76: (8)                         Args:
77: (12)                            node (Node): Node to add as a child
78: (8)                         """
79: (8)                         self._children.append(node)
80: (4)                     def draw(
81: (8)                         self,
82: (8)                         projection_matrix: Optional[glm.mat4],
83: (8)                         camera_matrix: Optional[glm.mat4],
84: (8)                         time: float = 0.0,
(4)         ) -> None:
(8)             """Draw node and children.
(8)             Keyword Args:
(12)                projection_matrix: projection matrix
(12)                camera_matrix: camera_matrix
(12)                time: The current time
(8)             """
(8)             if self._mesh:
(12)                self._mesh.draw(
(16)                    projection_matrix=projection_matrix,
(16)                    model_matrix=self._matrix_global,
(16)                    camera_matrix=camera_matrix,
(16)                    time=time,
(12)                )
(8)             for child in self._children:
(12)                child.draw(
(16)                    projection_matrix=projection_matrix,
(16)                    camera_matrix=camera_matrix,
(16)                    time=time,
(12)                )
(4)         def draw_bbox(
(8)             self,
(8)             projection_matrix: Optional[glm.mat4],
(8)             camera_matrix: Optional[glm.mat4],
(8)             program: moderngl.Program,
(8)             vao: VAO,
(4)         ) -> None:
(8)             """Draw bounding box around the node and children.
(8)             Keyword Args:
(12)                projection_matrix: projection matrix
(12)                camera_matrix: camera_matrix
(12)                program (moderngl.Program): The program to render the bbox
(12)                vao: The vertex array representing the bounding box
(8)             """
(8)             if self._mesh:
(12)                assert (
(16)                    projection_matrix is not None
(12)                ), "Can not draw bbox, the projection matrix is empty"
(12)                assert self._matrix_global is not None, "Can not draw bbox, the global matrix is empty"
(12)                assert camera_matrix is not None, "Can not draw bbox, the camera matrix is empty"
(12)                self._mesh.draw_bbox(
(16)                    projection_matrix, self._matrix_global, camera_matrix, program, vao
(12)                )
(8)             for child in self.children:
(12)                child.draw_bbox(projection_matrix, camera_matrix, program, vao)
(4)         def draw_wireframe(
(8)             self,
(8)             projection_matrix: Optional[glm.mat4],
(8)             camera_matrix: Optional[glm.mat4],
(8)             program: moderngl.Program,
(4)         ) -> None:
(8)             """Render the node as wireframe.
(8)             Keyword Args:
(12)                projection_matrix (bytes): projection matrix
(12)                camera_matrix (bytes): camera_matrix
(12)                program (moderngl.Program): The program to render wireframe
(8)             """
(8)             if self._mesh:
(12)                assert (
(16)                    projection_matrix is not None
(12)                ), "Can not draw bbox, the projection matrix is empty"
(12)                assert self._matrix_global is not None, "Can not draw bbox, the global matrix is empty"
(12)                self._mesh.draw_wireframe(projection_matrix, self._matrix_global, program)
(8)             for child in self.children:
```

```
149: (12)                       child.draw_wireframe(projection_matrix, self._matrix_global,
program)
150: (4)                def calc_global_bbox(
151: (8)                    self, view_matrix: glm.mat4, bbox_min: Optional[glm.vec3], bbox_max:
Optional[glm.vec3]
152: (4)                ) -> tuple[glm.vec3, glm.vec3]:
153: (8)                    """Recursive calculation of scene bbox.
154: (8)                    Keyword Args:
155: (12)                       view_matrix (numpy.ndarray): view matrix
156: (12)                       bbox_min: min bbox values
157: (12)                       bbox_max: max bbox values
158: (8)                    """
159: (8)                    if self._matrix is not None:
160: (12)                       view_matrix = self._matrix * view_matrix
161: (8)                    if self._mesh:
162: (12)                       bbox_min, bbox_max = self._mesh.calc_global_bbox(view_matrix,
bbox_min, bbox_max)
163: (8)                    for child in self._children:
164: (12)                       bbox_min, bbox_max = child.calc_global_bbox(view_matrix, bbox_min,
bbox_max)
165: (8)                    return bbox_min, bbox_max
166: (4)                def calc_model_mat(self, parent_matrix: glm.mat4) -> None:
167: (8)                    """Calculate the model matrix related to all parents.
168: (8)                    Args:
169: (12)                       parent_matrix: Matrix for parent node
170: (8)                    """
171: (8)                    if self._matrix is not None:
172: (12)                       self._matrix_global = parent_matrix * self._matrix
173: (12)                       for child in self._children:
174: (16)                           child.calc_model_mat(self._matrix_global)
175: (8)                    else:
176: (12)                       self._matrix_global = parent_matrix
177: (12)                       for child in self._children:
178: (16)                           child.calc_model_mat(parent_matrix)
179: (4)                def __repr__(self) -> str:
180: (8)                    return "<Node name={}>".format(self.name)

----------------------------------------

File 89 - scene.py:

1: (0)              """
2: (0)              Wrapper for a loaded scene with properties.
3: (0)              """
4: (0)              import logging
5: (0)              from typing import TYPE_CHECKING, Any, Optional
6: (0)              import glm
7: (0)              import moderngl
8: (0)              import moderngl_window as mglw
9: (0)              from moderngl_window import geometry
10: (0)             from moderngl_window.meta import ProgramDescription
11: (0)             from moderngl_window.resources.programs import programs
12: (0)             from .material import Material
13: (0)             from .node import Node
14: (0)             from .programs import (
15: (4)                 ColorLightProgram,
16: (4)                 FallbackProgram,
17: (4)                 MeshProgram,
18: (4)                 TextureLightProgram,
19: (4)                 TextureProgram,
20: (4)                 TextureVertexColorProgram,
21: (4)                 VertexColorProgram,
22: (0)             )
23: (0)             logger = logging.getLogger(__name__)
24: (0)             if TYPE_CHECKING:
25: (4)                 from moderngl_window.scene import Camera, Material, Mesh, Node
26: (0)             class Scene:
27: (4)                 """Generic scene"""
28: (4)                 def __init__(self, name: Optional[str], **kwargs: Any):
```

```
29: (8)                     """Create a scene with a name.
30: (8)                     Args:
31: (12)                        name (str): Unique name or path for the scene
32: (8)                     """
33: (8)                     self.name = name
34: (8)                     self.root_nodes: list[Node] = []
35: (8)                     self.nodes: list[Node] = []
36: (8)                     self.materials: list[Material] = []
37: (8)                     self.meshes: list[Mesh] = []
38: (8)                     self.cameras: list[Camera] = []
39: (8)                     self.bbox_min: glm.vec3 = glm.vec3()
40: (8)                     self.bbox_max: glm.vec3 = glm.vec3()
41: (8)                     self.diagonal_size = 1.0
42: (8)                     self.bbox_vao = geometry.bbox()
43: (8)                     if self.ctx.extra is None:
44: (12)                        self.ctx.extra = {}
45: (8)                     self.bbox_program = self.ctx.extra.get("DEFAULT_BBOX_PROGRAM")
46: (8)                     if not self.bbox_program:
47: (12)                        self.bbox_program = programs.load(
48: (16)                            ProgramDescription(path="scene_default/bbox.glsl"),
49: (12)                        )
50: (12)                        self.ctx.extra["DEFAULT_BBOX_PROGRAM"] = self.bbox_program
51: (8)                     self.wireframe_program =
self.ctx.extra.get("DEFAULT_WIREFRAME_PROGRAM")
52: (8)                     if not self.wireframe_program:
53: (12)                        self.wireframe_program = programs.load(
54: (16)                            ProgramDescription(path="scene_default/wireframe.glsl"),
55: (12)                        )
56: (12)                        self.ctx.extra["DEFAULT_WIREFRAME_PROGRAM"] =
self.wireframe_program
57: (8)                     self._matrix = glm.mat4()
58: (4)                 @property
59: (4)                 def ctx(self) -> moderngl.Context:
60: (8)                     """moderngl.Context: The current context"""
61: (8)                     return mglw.ctx()
62: (4)                 @property
63: (4)                 def matrix(self) -> glm.mat4:
64: (8)                     """glm.mat4x4: The current model matrix
65: (8)                     This property is settable.
66: (8)                     """
67: (8)                     return self._matrix
68: (4)                 @matrix.setter
69: (4)                 def matrix(self, matrix: glm.mat4) -> None:
70: (8)                     self._matrix = matrix
71: (8)                     for node in self.root_nodes:
72: (12)                        node.calc_model_mat(self._matrix)
73: (4)                 def draw(
74: (8)                     self,
75: (8)                     projection_matrix: Optional[glm.mat4] = None,
76: (8)                     camera_matrix: Optional[glm.mat4] = None,
77: (8)                     time: float = 0.0,
78: (4)                 ) -> None:
79: (8)                     """Draw all the nodes in the scene.
80: (8)                     Args:
81: (12)                        projection_matrix (ndarray): projection matrix (bytes)
82: (12)                        camera_matrix (ndarray): camera_matrix (bytes)
83: (12)                        time (float): The current time
84: (8)                     """
85: (8)                     for node in self.root_nodes:
86: (12)                        node.draw(
87: (16)                            projection_matrix=projection_matrix,
88: (16)                            camera_matrix=camera_matrix,
89: (16)                            time=time,
90: (12)                        )
91: (8)                     self.ctx.clear_samplers(0, 4)
92: (4)                 def draw_bbox(
93: (8)                     self,
94: (8)                     projection_matrix: Optional[glm.mat4] = None,
95: (8)                     camera_matrix: Optional[glm.mat4] = None,
        children: float = True,
        color: tuple[float, float, float] = (0.75, 0.75, 0.75),
    ) -> None:
        """Draw scene and mesh bounding boxes.
        Args:
            projection_matrix (glm.mat4): mat4 projection
            camera_matrix (glm.mat4): mat4 camera matrix
            children (bool): Will draw bounding boxes for meshes as well
            color (tuple): Color of the bounding boxes
        """
        projection_matrix = projection_matrix
        camera_matrix = camera_matrix
        self.bbox_program["m_proj"].write(projection_matrix)
        self.bbox_program["m_model"].write(self._matrix)
        self.bbox_program["m_cam"].write(camera_matrix)
        self.bbox_program["bb_min"].write(self.bbox_min)
        self.bbox_program["bb_max"].write(self.bbox_max)
        self.bbox_program["color"].value = color
        self.bbox_vao.render(self.bbox_program)
        if not children:
            return
        for node in self.root_nodes:
            node.draw_bbox(projection_matrix, camera_matrix, self.bbox_program, self.bbox_vao)
    def draw_wireframe(
        self,
        projection_matrix: Optional[glm.mat4] = None,
        camera_matrix: Optional[glm.mat4] = None,
        color: tuple[float, float, float, float] = (0.75, 0.75, 0.75, 1.0),
    ) -> None:
        """Render the scene in wireframe mode.
        Args:
            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
        """
        projection_matrix = projection_matrix
        camera_matrix = camera_matrix
        self.wireframe_program["m_proj"].write(projection_matrix)
        self.wireframe_program["m_model"].write(self._matrix)
        self.wireframe_program["m_cam"].write(camera_matrix)
        self.wireframe_program["color"] = color
        self.ctx.wireframe = True
        for node in self.root_nodes:
            node.draw_wireframe(projection_matrix, camera_matrix, self.wireframe_program)
        self.ctx.wireframe = False
    def apply_mesh_programs(
        self, mesh_programs: Optional[list[MeshProgram]] = None, clear: bool = True
    ) -> None:
        """Applies mesh programs to meshes.
        If not mesh programs are passed in we assign default ones.
        Args:
            mesh_programs (list): List of mesh programs to assign
            clear (bool): Clear all assigned mesh programs
        """
        global DEFAULT_PROGRAMS
        if clear:
            for mesh in self.meshes:
                mesh.mesh_program = None
        if not mesh_programs:
            mesh_programs = self.ctx.extra.get("DEFAULT_PROGRAMS")
            if not mesh_programs:
                mesh_programs = [
                    TextureLightProgram(),
                    TextureProgram(),
                    VertexColorProgram(),
162: (20)                               TextureVertexColorProgram(),
163: (20)                               ColorLightProgram(),
164: (20)                               FallbackProgram(),
165: (16)                           ]
166: (16)                           self.ctx.extra["DEFAULT_PROGRAMS"] = mesh_programs
167: (8)                    for mesh in self.meshes:
168: (12)                       for mesh_prog in mesh_programs:
169: (16)                           instance = mesh_prog.apply(mesh)
170: (16)                           if instance is not None:
171: (20)                               if isinstance(instance, MeshProgram):
172: (24)                                   mesh.mesh_program = mesh_prog
173: (24)                                   break
174: (20)                               else:
175: (24)                                   raise ValueError(
176: (28)                                       "apply() must return a MeshProgram instance, not
{}".format(
177: (32)                                           type(instance)
178: (28)                                       )
179: (24)                                   )
180: (12)                       if not mesh.mesh_program:
181: (16)                           logger.warning("WARING: No mesh program applied to '%s'",
mesh.name)
182: (4)                def calc_scene_bbox(self) -> None:
183: (8)                    """Calculate scene bbox"""
184: (8)                    bbox_min: Optional[glm.vec3] = None
185: (8)                    bbox_max: Optional[glm.vec3] = None
186: (8)                    for node in self.root_nodes:
187: (12)                       bbox_min, bbox_max = node.calc_global_bbox(glm.mat4(), bbox_min,
bbox_max)
188: (8)                    assert (bbox_max is not None) and (
189: (12)                       bbox_min is not None
190: (8)                    ), "The bounding are not defined, please make sure your code is
correct"
191: (8)                    self.bbox_min = bbox_min
192: (8)                    self.bbox_max = bbox_max
193: (8)                    self.diagonal_size = glm.length(self.bbox_max - self.bbox_min)
194: (4)                def prepare(self) -> None:
195: (8)                    """prepare the scene for rendering.
196: (8)                    Calls ``apply_mesh_programs()`` assigning default meshprograms if
needed
197: (8)                    and sets the model matrix.
198: (8)                    """
199: (8)                    self.apply_mesh_programs()
200: (8)                    self.matrix = glm.mat4()
201: (4)                def find_node(self, name: Optional[str] = None) -> Optional[Node]:
202: (8)                    """Finds a :py:class:`~moderngl_window.scene.Node`
203: (8)                    Keyword Args:
204: (12)                       name (str): Case sensitive name
205: (8)                    Returns:
206: (12)                       A :py:class:`~moderngl_window.scene.Node` or ``None`` if not
found.
207: (8)                    """
208: (8)                    for node in self.nodes:
209: (12)                       if node.name == name:
210: (16)                           return node
211: (8)                    return None
212: (4)                def find_material(self, name: Optional[str] = None) -> Optional[Material]:
213: (8)                    """Finds a :py:class:`~moderngl_window.scene.Material`
214: (8)                    Keyword Args:
215: (12)                       name (str): Case sensitive material name
216: (8)                    Returns:
217: (12)                       A :py:class:`~moderngl_window.scene.Material` or ``None``
218: (8)                    """
219: (8)                    for mat in self.materials:
220: (12)                       if mat.name == name:
221: (16)                           return mat
222: (8)                    return None
223: (4)                def release(self) -> None:
224: (8)                    """Destroys the scene data and vertex buffers"""
```

```
225: (8)                    self.destroy()
226: (4)                def destroy(self) -> None:
227: (8)                    """Destroys the scene data and vertex buffers"""
228: (8)                    for mesh in self.meshes:
229: (12)                       if mesh.vao is not None:
230: (16)                           mesh.vao.release()
231: (8)                    for mat in self.materials:
232: (12)                       mat.release()
233: (8)                    self.meshes = []
234: (8)                    self.root_nodes = []
235: (4)                def __str__(self) -> str:
236: (8)                    return "<Scene: {}>".format(self.name)
237: (4)                def __repr__(self) -> str:
238: (8)                    return str(self)

----------------------------------------

File 90 - scenes.py:

1: (0)              """
2: (0)              Scene Registry
3: (0)              """
4: (0)              from moderngl_window.meta import ResourceDescription, SceneDescription
5: (0)              from moderngl_window.resources.base import BaseRegistry
6: (0)              from moderngl_window.scene import Scene
7: (0)              class Scenes(BaseRegistry):
8: (4)                  """Handles scene loading"""
9: (4)                  settings_attr = "SCENE_LOADERS"
10: (4)                 meta: SceneDescription
11: (4)                 def load(self, meta: ResourceDescription) -> Scene:
12: (8)                     """Load a scene with the configured loaders.
13: (8)                     Args:
14: (12)                        meta (:py:class:`~moderngl_window.meta.scene.SceneDescription`):
15: (12)                        The resource description
16: (8)                     Returns:
17: (12)                        :py:class:`~moderngl_window.scene.Scene`: The loaded scene
18: (8)                     """
19: (8)                     scene = super().load(meta)
20: (8)                     assert isinstance(
21: (12)                        scene, Scene
22: (8)                     ), f"{meta} did not load a moderngl_window.scene.Scene object, please
correct it."
23: (8)                     return scene
24: (0)             scenes = Scenes()

----------------------------------------

File 91 - tracks.py:

1: (0)              """
2: (0)              Registry for rocket tracks
3: (0)              """
4: (0)              from rocket.tracks import Track
5: (0)              class Tracks:
6: (4)                  """Registry for requested rocket tracks"""
7: (4)                  def __init__(self) -> None:
8: (8)                      self.tacks: list[Track] = []
9: (8)                      self.track_map: dict[str, Track] = {}
10: (4)                 def get(self, name: str) -> Track:
11: (8)                     """
12: (8)                     Get or create a Track object.
13: (8)                     :param name: Name of the track
14: (8)                     :return: Track object
15: (8)                     """
16: (8)                     name = name.lower()
17: (8)                     track = self.track_map.get(name)
18: (8)                     if not track:
19: (12)                        track = Track(name)
20: (12)                        self.tacks.append(track)
```

```
(12)                        self.track_map[name] = track
(8)                     return track
(0)             tracks = Tracks()
```

File 92 - camera.py:

```
(0)             import time
(0)             from typing import Any, Optional, Union
(0)             import glm
(0)             from glm import cos, radians, sin
(0)             from moderngl_window.context.base import BaseKeys
(0)             from moderngl_window.opengl.projection import Projection3D
(0)             from moderngl_window.utils.keymaps import QWERTY, KeyMapFactory
(0)             RIGHT = 1
(0)             LEFT = 2
(0)             FORWARD = 3
(0)             BACKWARD = 4
(0)             UP = 5
(0)             DOWN = 6
(0)             STILL = 0
(0)             POSITIVE = 1
(0)             NEGATIVE = 2
(0)             class Camera:
(4)                 """Simple camera class containing projection.
(4)                 .. code:: python
(8)                     camera = Camera(fov=60.0, aspect_ratio=1.0, near=1.0, far=100.0)
(8)                     print(camera.matrix)
(8)                     print(camera.projection.matrix)
(4)                 """
(4)                 def __init__(
(8)                     self, fov: float = 60.0, aspect_ratio: float = 1.0, near: float = 1.0,
far: float = 100.0
(4)                 ):
(8)                     """Initialize camera using a specific projection
(8)                     Keyword Args:
(12)                        fov (float): Field of view
(12)                        aspect_ratio (float): Aspect ratio
(12)                        near (float): Near plane
(12)                        far (float): Far plane
(8)                     """
(8)                     self.position = glm.vec3(0.0, 0.0, 0.0)
(8)                     self.up = glm.vec3(0.0, 1.0, 0.0)
(8)                     self.right = glm.vec3(1.0, 0.0, 0.0)
(8)                     self.dir = glm.vec3(0.0, 0.0, -1.0)
(8)                     self._yaw = -90.0
(8)                     self._pitch = 0.0
(8)                     self._up = glm.vec3(0.0, 1.0, 0.0)
(8)                     self._projection = Projection3D(aspect_ratio, fov, near, far)
(4)                 @property
(4)                 def projection(self) -> Projection3D:
(8)                     """:py:class:`~moderngl_window.opengl.projection.Projection3D`: The 3D
projection"""
(8)                     return self._projection
(4)                 def set_position(self, x: float, y: float, z: float) -> None:
(8)                     """Set the 3D position of the camera.
(8)                     Args:
(12)                        x (float): x position
(12)                        y (float): y position
(12)                        z (float): z position
(8)                     """
(8)                     self.position = glm.vec3(x, y, z)
(4)                 def set_rotation(self, yaw: float, pitch: float) -> None:
(8)                     """Set the rotation of the camera.
(8)                     Args:
(12)                        yaw (float): yaw rotation
(12)                        pitch (float): pitch rotation
(8)                     """
```

```
60: (8)                         self._pitch = pitch
61: (8)                         self._yaw = yaw
62: (8)                         self._update_yaw_and_pitch()
63: (4)                     @property
64: (4)                     def yaw(self) -> float:
65: (8)                         """float: The current yaw angle."""
66: (8)                         return self._yaw
67: (4)                     @yaw.setter
68: (4)                     def yaw(self, value: float) -> None:
69: (8)                         self._yaw = value
70: (8)                         self._update_yaw_and_pitch()
71: (4)                     @property
72: (4)                     def pitch(self) -> float:
73: (8)                         """float: The current pitch angle."""
74: (8)                         return self._pitch
75: (4)                     @pitch.setter
76: (4)                     def pitch(self, value: float) -> None:
77: (8)                         self._pitch = value
78: (8)                         self._update_yaw_and_pitch()
79: (4)                     @property
80: (4)                     def matrix(self) -> glm.mat4:
81: (8)                         """glm.mat4: The current view matrix for the camera"""
82: (8)                         self._update_yaw_and_pitch()
83: (8)                         return self._gl_look_at(self.position, self.position + self.dir, self._up)
84: (4)                     def _update_yaw_and_pitch(self) -> None:
85: (8)                         """Updates the camera vectors based on the current yaw and pitch"""
86: (8)                         front = glm.vec3(0.0, 0.0, 0.0)
87: (8)                         front.x = cos(radians(self.yaw)) * cos(radians(self.pitch))
88: (8)                         front.y = sin(radians(self.pitch))
89: (8)                         front.z = sin(radians(self.yaw)) * cos(radians(self.pitch))
90: (8)                         self.dir = glm.normalize(front)
91: (8)                         self.right = glm.normalize(glm.cross(self.dir, self._up))
92: (8)                         self.up = glm.normalize(glm.cross(self.right, self.dir))
93: (4)                     def look_at(
94: (8)                         self, vec: Optional[glm.vec3] = None, pos: Optional[tuple[float, float, float]] = None
95: (4)                     ) -> glm.mat4:
96: (8)                         """Look at a specific point
97: (8)                         Either ``vec`` or ``pos`` needs to be supplied.
98: (8)                         Keyword Args:
99: (12)                            vec (glm.vec3): position
100: (12)                           pos (tuple/list): list of tuple ``[x, y, x]`` / ``(x, y, x)``
101: (8)                        Returns:
102: (12)                           glm.mat4x4: Camera matrix
103: (8)                        """
104: (8)                        if pos is not None:
105: (12)                           vec = glm.vec3(pos)
106: (8)                        if vec is None:
107: (12)                           raise ValueError("vector or pos must be set")
108: (8)                        return self._gl_look_at(self.position, vec, self._up)
109: (4)                    def _gl_look_at(self, pos: glm.vec3, target: glm.vec3, up: glm.vec3) -> glm.mat4:
110: (8)                        """The standard lookAt method.
111: (8)                        Args:
112: (12)                           pos: current position
113: (12)                           target: target position to look at
114: (12)                           up: direction up
115: (8)                        Returns:
116: (12)                           glm.mat4: The matrix
117: (8)                        """
118: (8)                        z = glm.normalize(pos - target)
119: (8)                        x = glm.normalize(glm.cross(glm.normalize(up), z))
120: (8)                        y = glm.cross(z, x)
121: (8)                        translate = glm.mat4()
122: (8)                        translate[3][0] = -pos.x
123: (8)                        translate[3][1] = -pos.y
124: (8)                        translate[3][2] = -pos.z
125: (8)                        rotate = glm.mat4()
```

```
(8)              rotate[0][0] = x[0]  # -- X
(8)              rotate[1][0] = x[1]
(8)              rotate[2][0] = x[2]
(8)              rotate[0][1] = y[0]  # -- Y
(8)              rotate[1][1] = y[1]
(8)              rotate[2][1] = y[2]
(8)              rotate[0][2] = z[0]  # -- Z
(8)              rotate[1][2] = z[1]
(8)              rotate[2][2] = z[2]
(8)              return rotate * translate
(0)      class KeyboardCamera(Camera):
(4)          """Camera controlled by mouse and keyboard.
(4)          The class interacts with the key constants in the
(4)          built in window types.
(4)          Creating a keyboard camera:
(4)          .. code:: python
(8)              camera = KeyboardCamera(
(12)                 self.wnd.keys,
(12)                 fov=75.0,
(12)                 aspect_ratio=self.wnd.aspect_ratio,
(12)                 near=0.1,
(12)                 far=1000.0,
(8)              )
(4)          We can also interact with the belonging
(4)          :py:class:`~moderngl_window.opengl.projection.Projection3D` instance.
(4)          .. code:: python
(8)              camera.projection.update(aspect_ratio=1.0)
(8)              camera.projection.tobytes()
(4)          """
(4)          def __init__(
(8)              self,
(8)              keys: BaseKeys,
(8)              keymap: KeyMapFactory = QWERTY,
(8)              fov: float = 60.0,
(8)              aspect_ratio: float = 1.0,
(8)              near: float = 1.0,
(8)              far: float = 100.0,
(4)          ):
(8)              """Initialize the camera
(8)              Args:
(12)                 keys (BaseKeys): The key constants for the current window type
(8)              Keyword Args:
(12)                 keymap (KeyMapFactory) : The keymap to use. By default QWERTY.
(12)                 fov (float): Field of view
(12)                 aspect_ratio (float): Aspect ratio
(12)                 near (float): near plane
(12)                 far (float): far plane
(8)              """
(8)              self.keys = keys
(8)              self.keymap = keymap(keys)
(8)              self._xdir = STILL
(8)              self._zdir = STILL
(8)              self._ydir = STILL
(8)              self._last_time = 0.0
(8)              self._last_rot_time = 0.0
(8)              self._velocity = 10.0
(8)              self._mouse_sensitivity = 0.5
(8)              super().__init__(fov=fov, aspect_ratio=aspect_ratio, near=near,
far=far)
(4)          @property
(4)          def mouse_sensitivity(self) -> float:
(8)              """float: Mouse sensitivity (rotation speed).
(8)              This property can also be set::
(12)                 camera.mouse_sensitivity = 2.5
(8)              """
(8)              return self._mouse_sensitivity
(4)          @mouse_sensitivity.setter
(4)          def mouse_sensitivity(self, value: float) -> None:
(8)              self._mouse_sensitivity = value
```

```
(4)               @property
(4)               def velocity(self) -> float:
(8)                   """float: The speed this camera move based on key inputs
(8)                   The property can also be modified::
(12)                      camera.velocity = 5.0
(8)                   """
(8)                   return self._velocity
(4)               @velocity.setter
(4)               def velocity(self, value: float) -> None:
(8)                   self._velocity = value
(4)               def key_input(self, key: str, action: str, modifiers: Any) -> None:
(8)                   """Process key inputs and move camera
(8)                   Args:
(12)                      key: The key
(12)                      action: key action release/press
(12)                      modifiers: key modifier states such as ctrl or shit
(8)                   """
(8)                   if key == self.keymap.RIGHT:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_right(True)
(12)                      elif action == self.keys.ACTION_RELEASE:
(16)                          self.move_right(False)
(8)                   elif key == self.keymap.LEFT:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_left(True)
(12)                      elif action == self.keys.ACTION_RELEASE:
(16)                          self.move_left(False)
(8)                   elif key == self.keymap.FORWARD:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_forward(True)
(12)                      if action == self.keys.ACTION_RELEASE:
(16)                          self.move_forward(False)
(8)                   elif key == self.keymap.BACKWARD:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_backward(True)
(12)                      if action == self.keys.ACTION_RELEASE:
(16)                          self.move_backward(False)
(8)                   elif key == self.keymap.DOWN:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_down(True)
(12)                      if action == self.keys.ACTION_RELEASE:
(16)                          self.move_down(False)
(8)                   elif key == self.keymap.UP:
(12)                      if action == self.keys.ACTION_PRESS:
(16)                          self.move_up(True)
(12)                      if action == self.keys.ACTION_RELEASE:
(16)                          self.move_up(False)
(4)               def move_left(self, activate: bool) -> None:
(8)                   """The camera should be continiously moving to the left.
(8)                   Args:
(12)                      activate (bool): Activate or deactivate this state
(8)                   """
(8)                   self.move_state(LEFT, activate)
(4)               def move_right(self, activate: bool) -> None:
(8)                   """The camera should be continiously moving to the right.
(8)                   Args:
(12)                      activate (bool): Activate or deactivate this state
(8)                   """
(8)                   self.move_state(RIGHT, activate)
(4)               def move_forward(self, activate: bool) -> None:
(8)                   """The camera should be continiously moving forward.
(8)                   Args:
(12)                      activate (bool): Activate or deactivate this state
(8)                   """
(8)                   self.move_state(FORWARD, activate)
(4)               def move_backward(self, activate: bool) -> None:
(8)                   """The camera should be continiously moving backwards.
(8)                   Args:
(12)                      activate (bool): Activate or deactivate this state
```

```
(8)            """
(8)            self.move_state(BACKWARD, activate)
(4)        def move_up(self, activate: bool) -> None:
(8)            """The camera should be continiously moving up.
(8)            Args:
(12)               activate (bool): Activate or deactivate this state
(8)            """
(8)            self.move_state(UP, activate)
(4)        def move_down(self, activate: bool) -> None:
(8)            """The camera should be continiously moving down.
(8)            Args:
(12)               activate (bool): Activate or deactivate this state
(8)            """
(8)            self.move_state(DOWN, activate)
(4)        def move_state(self, direction: int, activate: bool) -> None:
(8)            """Set the camera position move state.
(8)            Args:
(12)               direction: What direction to update
(12)               activate: Start or stop moving in the direction
(8)            """
(8)            if direction == RIGHT:
(12)               self._xdir = POSITIVE if activate else STILL
(8)            elif direction == LEFT:
(12)               self._xdir = NEGATIVE if activate else STILL
(8)            elif direction == FORWARD:
(12)               self._zdir = NEGATIVE if activate else STILL
(8)            elif direction == BACKWARD:
(12)               self._zdir = POSITIVE if activate else STILL
(8)            elif direction == UP:
(12)               self._ydir = POSITIVE if activate else STILL
(8)            elif direction == DOWN:
(12)               self._ydir = NEGATIVE if activate else STILL
(4)        def rot_state(self, dx: float, dy: float) -> None:
(8)            """Update the rotation of the camera.
(8)            This is done by passing in the relative
(8)            mouse movement change on x and y (delta x, delta y).
(8)            In the past this method took the viewport position
(8)            of the mouse. This does not work well when
(8)            mouse exclusivity mode is enabled.
(8)            Args:
(12)               dx: Relative mouse position change on x
(12)               dy: Relative mouse position change on y
(8)            """
(8)            now = time.time()
(8)            delta = now - self._last_rot_time
(8)            self._last_rot_time = now
(8)            if delta > 0.1 and max(abs(dx), abs(dy)) > 2:
(12)               return
(8)            dx *= self._mouse_sensitivity
(8)            dy *= self._mouse_sensitivity
(8)            self._yaw -= dx
(8)            self._pitch += dy
(8)            if self.pitch > 85.0:
(12)               self.pitch = 85.0
(8)            if self.pitch < -85.0:
(12)               self.pitch = -85.0
(8)            self._update_yaw_and_pitch()
(4)        @property
(4)        def matrix(self) -> glm.mat4:
(8)            """glm.mat4x4: The current view matrix for the camera"""
(8)            now = time.time()
(8)            t = max(now - self._last_time, 0)
(8)            self._last_time = now
(8)            if self._xdir == POSITIVE:
(12)               self.position += self.right * self._velocity * t
(8)            elif self._xdir == NEGATIVE:
(12)               self.position -= self.right * self._velocity * t
(8)            if self._zdir == NEGATIVE:
(12)               self.position += self.dir * self._velocity * t
```

```
        elif self._zdir == POSITIVE:
            self.position -= self.dir * self._velocity * t
        if self._ydir == POSITIVE:
            self.position += self.up * self._velocity * t
        elif self._ydir == NEGATIVE:
            self.position -= self.up * self._velocity * t
        return self._gl_look_at(self.position, self.position + self.dir, self._up)
class OrbitCamera(Camera):
    """Camera controlled by the mouse to pan around the target.
    The functions :py:function:`~camera.OrbitCamera.rot_state` and
    :py:function:`~camera.OrbitCamera.rot_state` are used to update the rotation and zoom.
    Creating a orbit camera:
    .. code:: python
        camera = OrbitCamera(
            target=(0., 0., 0.),
            radius=2.0
            fov=75.0,
            aspect_ratio=self.wnd.aspect_ratio,
            near=0.1,
            far=1000.0,
        )
    We can also interact with the belonging
    :py:class:`~moderngl_window.opengl.projection.Projection3D` instance.
    .. code:: python
        camera.projection.update(aspect_ratio=1.0)
        camera.projection.tobytes()
    """
    def __init__(
        self,
        target: Union[glm.vec3, tuple[float, float, float]] = (0.0, 0.0, 0.0),
        radius: float = 2.0,
        angles: tuple[float, float] = (45.0, -45.0),
        **kwargs: Any,
    ):
        """Initialize the camera
        Keyword Args:
            target (float, float, float): Target point
            radius (float): Radius
            angles (float, float): angle_x and angle_y in degrees
            fov (float): Field of view
            aspect_ratio (float): Aspect ratio
            near (float): near plane
            far (float): far plane
        """
        self.radius = radius  # radius in base units
        self.angle_x, self.angle_y = angles  # angles in degrees
        self.target = glm.vec3(target)  # camera target in base units
        self.up = glm.vec3(0.0, 1.0, 0.0)  # camera up vector
        self._mouse_sensitivity = 1.0
        self._zoom_sensitivity = 1.0
        super().__init__(**kwargs)
    @property
    def matrix(self) -> glm.mat4:
        """glm.mat4: The current view matrix for the camera"""
        position = (
            cos(radians(self.angle_x)) * sin(radians(self.angle_y)) * self.radius + self.target[0],
            cos(radians(self.angle_y)) * self.radius + self.target[1],
            sin(radians(self.angle_x)) * sin(radians(self.angle_y)) * self.radius + self.target[2],
        )
        self.set_position(*position)
        return glm.lookAt(
            position,
            self.target,  # what to look at
            self.up,  # camera up direction (change for rolling the camera)
        )
```

```
(4)                 @property
(4)                 def angle_x(self) -> float:
(8)                     """float: camera angle x in degrees.
(8)                     This property can also be set::
(12)                        camera.angle_x = 45.
(8)                     """
(8)                     return self._angle_x
(4)                 @angle_x.setter
(4)                 def angle_x(self, value: float) -> None:
(8)                     """Set camera rotation_x in degrees."""
(8)                     self._angle_x = value
(4)                 @property
(4)                 def angle_y(self) -> float:
(8)                     """float: camera angle y in degrees.
(8)                     This property can also be set::
(12)                        camera.angle_y = 45.
(8)                     """
(8)                     return self._angle_y
(4)                 @angle_y.setter
(4)                 def angle_y(self, value: float) -> None:
(8)                     """Set camera rotation_y in degrees."""
(8)                     self._angle_y = value
(4)                 @property
(4)                 def mouse_sensitivity(self) -> float:
(8)                     """float: Mouse sensitivity (rotation speed).
(8)                     This property can also be set::
(12)                        camera.mouse_sensitivity = 2.5
(8)                     """
(8)                     return self._mouse_sensitivity
(4)                 @mouse_sensitivity.setter
(4)                 def mouse_sensitivity(self, value: float) -> None:
(8)                     self._mouse_sensitivity = value
(4)                 @property
(4)                 def zoom_sensitivity(self) -> float:
(8)                     """float: Mousewheel zooming sensitivity (zoom speed).
(8)                     This property can also be set::
(12)                        camera.zoom_sensitivity = 2.5
(8)                     """
(8)                     return self._zoom_sensitivity
(4)                 @zoom_sensitivity.setter
(4)                 def zoom_sensitivity(self, value: float) -> None:
(8)                     self._zoom_sensitivity = value
(4)                 def rot_state(self, dx: float, dy: float) -> None:
(8)                     """Update the rotation of the camera around the target point.
(8)                     This is done by passing relative mouse change in the x and y axis
(delta x, delta y)
(8)                     Args:
(12)                        dx: Relative mouse position change on x axis
(12)                        dy: Relative mouse position change on y axis
(8)                     """
(8)                     self.angle_x += dx * self.mouse_sensitivity / 10.0
(8)                     self.angle_y += dy * self.mouse_sensitivity / 10.0
(8)                     self.angle_y = max(min(self.angle_y, -5.0), -175.0)
(4)                 def zoom_state(self, y_offset: float) -> None:
(8)                     self.radius -= y_offset * self._zoom_sensitivity
(8)                     self.radius = max(1.0, self.radius)
```

File 93 - programs.py:

```
(0)             import moderngl
(0)             from moderngl_window.meta import ProgramDescription, ResourceDescription
(0)             from moderngl_window.resources.base import BaseRegistry
(0)             class Programs(BaseRegistry):
(4)                 """Handle program loading"""
(4)                 settings_attr = "PROGRAM_LOADERS"
(4)                 meta: ProgramDescription
(4)                 def resolve_loader(self, meta: ResourceDescription) -> None:
```

```
9: (8)                      """Resolve program loader.
10: (8)                     Determines if the references resource is a single
11: (8)                     or multiple glsl files unless ``kind`` is specified.
12: (8)                     Args:
13: (12)                        meta (ProgramDescription): The resource description
14: (8)                     """
15: (8)                     if meta.kind == "":
16: (12)                        if meta.path is None:
17: (16)                            meta.kind = "separate"
18: (12)                        else:
19: (16)                            meta.kind = "single"
20: (8)                     super().resolve_loader(meta)
21: (4)                 def load(self, meta: ResourceDescription) -> moderngl.Program:
22: (8)                     """Loads a shader program with the configured loaders
23: (8)                     Args:
24: (12)                        meta
(:py:class:`~moderngl_window.meta.program.ProgramDescription`):
25: (12)                        The resource description
26: (8)                     Returns:
27: (12)                        moderngl.Program: The shader program
28: (8)                     """
29: (8)                     return super().load(meta)
30: (0)             programs = Programs()

----------------------------------------

File 94 - textures.py:

1: (0)             """
2: (0)             Shader Registry
3: (0)             """
4: (0)             from typing import Union
5: (0)             import moderngl
6: (0)             from moderngl_window.meta import ResourceDescription, TextureDescription
7: (0)             from moderngl_window.resources.base import BaseRegistry
8: (0)             TextureAny = Union[
9: (4)                 moderngl.Texture,
10: (4)                moderngl.TextureArray,
11: (4)                moderngl.TextureCube,
12: (4)                moderngl.Texture3D,
13: (0)             ]
14: (0)             class Textures(BaseRegistry):
15: (4)                 """Handles texture resources"""
16: (4)                 settings_attr = "TEXTURE_LOADERS"
17: (4)                 meta: TextureDescription
18: (4)                 def load(self, meta: ResourceDescription) -> TextureAny:
19: (8)                     """Loads a texture with the configured loaders.
20: (8)                     Args:
21: (12)                        meta
(:py:class:`~moderngl_window.meta.texture.TextureDescription`):
22: (12)                        The resource description
23: (8)                     Returns:
24: (12)                        moderngl.Texture: 2d texture
25: (8)                     Returns:
26: (12)                        moderngl.TextureArray: texture array if ``layers`` is supplied
27: (8)                     """
28: (8)                     texture = super().load(meta)
29: (8)                     assert (
30: (12)                        isinstance(texture, moderngl.Texture)
31: (12)                        or isinstance(texture, moderngl.TextureArray)
32: (12)                        or isinstance(texture, moderngl.TextureCube)
33: (12)                        or isinstance(texture, moderngl.Texture3D)
34: (8)                     ), f"{meta} did not load a texture. Please correct it"
35: (8)                     return texture
36: (0)             textures = Textures()

----------------------------------------

File 95 - __init__.py:
```

```
(0)     from collections.abc import Iterator
(0)     from contextlib import contextmanager
(0)     from pathlib import Path
(0)     from typing import Union
(0)     from moderngl_window.conf import settings
(0)     from moderngl_window.exceptions import ImproperlyConfigured
(0)     from moderngl_window.resources.data import data as data
(0)     from moderngl_window.resources.programs import programs as programs
(0)     from moderngl_window.resources.scenes import scenes as scenes
(0)     from moderngl_window.resources.textures import TextureAny as TextureAny
(0)     from moderngl_window.resources.textures import textures as textures
(0)     def register_dir(path: Union[Path, str]) -> None:
(4)         """Adds a resource directory for all resource types
(4)         Args:
(8)             path (Union[Path, str]): Directory path
(4)         """
(4)         register_data_dir(path)
(4)         register_program_dir(path)
(4)         register_scene_dir(path)
(4)         register_texture_dir(path)
(0)     def register_program_dir(path: Union[Path, str]) -> None:
(4)         """Adds a resource directory specifically for programs
(4)         Args:
(8)             path (Union[Path, str]): Directory path
(4)         """
(4)         _append_unique_path(path, settings.PROGRAM_DIRS)
(0)     def register_texture_dir(path: Union[Path, str]) -> None:
(4)         """Adds a resource directory specifically for textures
(4)         Args:
(8)             path (Union[Path, str]): Directory path
(4)         """
(4)         _append_unique_path(path, settings.TEXTURE_DIRS)
(0)     def register_scene_dir(path: Union[Path, str]) -> None:
(4)         """Adds a resource directory specifically for scenes
(4)         Args:
(8)             path (Union[Path, str]): Directory path
(4)         """
(4)         _append_unique_path(path, settings.SCENE_DIRS)
(0)     def register_data_dir(path: Union[Path, str]) -> None:
(4)         """Adds a resource directory specifically for data files
(4)         Args:
(8)             path (Union[Path, str]): Directory path
(4)         """
(4)         _append_unique_path(path, settings.DATA_DIRS)
(0)     def _append_unique_path(path: Union[Path, str], dest: list[Union[Path, str]]) -> None:
(4)         path = Path(path)
(4)         if not path.is_absolute():
(8)             raise ImproperlyConfigured("Search path must be absolute: {}".format(path))
(4)         if not path.is_dir():
(8)             raise ImproperlyConfigured("Search path is not a directory: {}".format(path))
(4)         if not path.exists():
(8)             raise ImproperlyConfigured("Search path do not exist: {}".format(path))
(4)         for resource_path in dest:
(8)             if Path(resource_path).samefile(path):
(12)                break
(4)         else:
(8)             dest.append(Path(path).absolute())
(0)     @contextmanager
(0)     def temporary_dirs(dirs: list[Union[Path, str]]) -> Iterator[list[Union[Path, str]]]:
(4)         """Temporarily changes all resource directories
(4)         Example::
(8)             with temporary_dirs([path1, path2, path3]):
(4)         Args:
```

```
64: (8)                     dirs (Union[Path,str]) list of paths to use
65: (4)                 """
66: (4)                 data_dirs = settings.DATA_DIRS
67: (4)                 program_dirs = settings.PROGRAM_DIRS
68: (4)                 scene_dirs = settings.SCENE_DIRS
69: (4)                 textures_dirs = settings.TEXTURE_DIRS
70: (4)                 settings.DATA_DIRS = dirs
71: (4)                 settings.PROGRAM_DIRS = dirs
72: (4)                 settings.SCENE_DIRS = dirs
73: (4)                 settings.TEXTURE_DIRS = dirs
74: (4)                 try:
75: (8)                     yield dirs
76: (4)                 finally:
77: (8)                     settings.DATA_DIRS = data_dirs
78: (8)                     settings.PROGRAM_DIRS = program_dirs
79: (8)                     settings.SCENE_DIRS = scene_dirs
80: (8)                     settings.TEXTURE_DIRS = textures_dirs
```

File 96 - material.py:

```
1: (0)              from typing import Optional
2: (0)              import moderngl
3: (0)              class MaterialTexture:
4: (4)                  """Wrapper for textures used in materials.
5: (4)                  Contains a texture and a sampler object.
6: (4)                  """
7: (4)                  def __init__(
8: (8)                      self, texture: Optional[moderngl.Texture] = None, sampler:
Optional[moderngl.Sampler] = None
9: (4)                  ):
10: (8)                     """Initialize instance.
11: (8)                     Args:
12: (12)                        texture (moderngl.Texture): Texture instance
13: (12)                        sampler (moderngl.Sampler): Sampler instance
14: (8)                     """
15: (8)                     self._texture = texture
16: (8)                     self._sampler = sampler
17: (4)                 @property
18: (4)                 def texture(self) -> Optional[moderngl.Texture]:
19: (8)                     """moderngl.Texture: Texture instance"""
20: (8)                     return self._texture
21: (4)                 @texture.setter
22: (4)                 def texture(self, value: moderngl.Texture) -> None:
23: (8)                     self._texture = value
24: (4)                 @property
25: (4)                 def sampler(self) -> Optional[moderngl.Sampler]:
26: (8)                     """moderngl.Sampler: Sampler instance"""
27: (8)                     return self._sampler
28: (4)                 @sampler.setter
29: (4)                 def sampler(self, value: moderngl.Sampler) -> None:
30: (8)                     self._sampler = value
31: (0)             class Material:
32: (4)                 """Generic material"""
33: (4)                 def __init__(self, name: str = ""):
34: (8)                     """Initialize material.
35: (8)                     Args:
36: (12)                        name (str): Name of the material
37: (8)                     """
38: (8)                     self._name = name or "default"
39: (8)                     self._color = (1.0, 1.0, 1.0, 1.0)
40: (8)                     self._mat_texture: Optional[MaterialTexture] = None
41: (8)                     self._double_sided = True
42: (4)                 @property
43: (4)                 def name(self) -> str:
44: (8)                     """str: Name of the material"""
45: (8)                     return self._name
46: (4)                 @name.setter
```

```
47: (4)                 def name(self, value: str) -> None:
48: (8)                     self._name = value
49: (4)                 @property
50: (4)                 def color(self) -> tuple[float, float, float, float]:
51: (8)                     """tuple[float, float, float, float]: RGBA color"""
52: (8)                     return self._color
53: (4)                 @color.setter
54: (4)                 def color(self, value: tuple[float, float, float, float]) -> None:
55: (8)                     self._color = value
56: (4)                 @property
57: (4)                 def mat_texture(self) -> Optional[MaterialTexture]:
58: (8)                     """MaterialTexture: instance"""
59: (8)                     return self._mat_texture
60: (4)                 @mat_texture.setter
61: (4)                 def mat_texture(self, value: MaterialTexture) -> None:
62: (8)                     self._mat_texture = value
63: (4)                 @property
64: (4)                 def double_sided(self) -> bool:
65: (8)                     """bool: Material surface is double sided?"""
66: (8)                     return self._double_sided
67: (4)                 @double_sided.setter
68: (4)                 def double_sided(self, value: bool) -> None:
69: (8)                     self._double_sided = value
70: (4)                 def release(self) -> None:
71: (8)                     if self._mat_texture:
72: (12)                        if self._mat_texture.texture:
73: (16)                            self._mat_texture.texture.release()
74: (4)                 def __str__(self) -> str:
75: (8)                     return "<Material {}>".format(self.name)
76: (4)                 def __repr__(self) -> str:
77: (8)                     return str(self)
```

File 97 - programs.py:

```
1: (0)             from __future__ import annotations
2: (0)             import os
3: (0)             from typing import Any, Optional
4: (0)             import glm
5: (0)             import moderngl
6: (0)             import moderngl_window
7: (0)             from moderngl_window.conf import settings
8: (0)             from moderngl_window.meta import ProgramDescription
9: (0)             from moderngl_window.resources.programs import programs
10: (0)            from .mesh import Mesh
11: (0)            settings.PROGRAM_DIRS.append(os.path.join(os.path.dirname(__file__),
"programs"))
12: (0)            class MeshProgram:
13: (4)                """
14: (4)                Describes how a mesh is rendered using a specific shader program
15: (4)                """
16: (4)                def __init__(self, program: Optional[moderngl.Program] = None, **kwargs:
Any) -> None:
17: (8)                    """Initialize.
18: (8)                    Args:
19: (12)                       program: The moderngl program
20: (8)                    """
21: (8)                    self.program = program
22: (4)                @property
23: (4)                def ctx(self) -> moderngl.Context:
24: (8)                    """moderngl.Context: The current context"""
25: (8)                    return moderngl_window.ctx()
26: (4)                def draw(
27: (8)                    self,
28: (8)                    mesh: Mesh,
29: (8)                    projection_matrix: glm.mat4,
30: (8)                    model_matrix: glm.mat4,
31: (8)                    camera_matrix: glm.mat4,
        time: float = 0.0,
    ) -> None:
        """Draw code for the mesh
        Args:
            mesh (Mesh): The mesh to render
        Keyword Args:
            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
        """
        assert self.program is not None, "There is no program to draw"
        assert mesh.vao is not None, "There is no vao to render"
        self.program["m_proj"].write(projection_matrix)
        self.program["m_mv"].write(model_matrix)
        self.program["m_cam"].write(camera_matrix)
        mesh.vao.render(self.program)
    def apply(self, mesh: Mesh) -> "MeshProgram" | None:
        """
        Determine if this ``MeshProgram`` should be applied to the mesh.
        Can return self or some ``MeshProgram`` instance to support dynamic ``MeshProgram`` creation
        Args:
            mesh: The mesh to inspect
        """
        raise NotImplementedError(
            "apply is not implemented. Please override the MeshProgram method"
        )
class VertexColorProgram(MeshProgram):
    """Vertex color program"""
    def __init__(self, program: Optional[moderngl.Program] = None, **kwargs: Any) -> None:
        super().__init__(program=None)
        self.program = programs.load(ProgramDescription(path="scene_default/vertex_color.glsl"))
    def draw(
        self,
        mesh: Mesh,
        projection_matrix: glm.mat4,
        model_matrix: glm.mat4,
        camera_matrix: glm.mat4,
        time: float = 0.0,
    ) -> None:
        assert self.program is not None, "There is no program to draw"
        assert mesh.vao is not None, "There is no vao to render"
        self.program["m_proj"].write(projection_matrix)
        self.program["m_model"].write(model_matrix)
        self.program["m_cam"].write(camera_matrix)
        mesh.vao.render(self.program)
    def apply(self, mesh: Mesh) -> Optional[MeshProgram]:
        if not mesh.material:
            return None
        if mesh.attributes.get("TEXCOORD_0"):
            return None
        if mesh.attributes.get("COLOR_0"):
            return self
        return None
class ColorLightProgram(MeshProgram):
    """Simple color program with light"""
    def __init__(self, program: Optional[moderngl.Program] = None, **kwargs: Any) -> None:
        super().__init__(program=None)
        self.program = programs.load(ProgramDescription(path="scene_default/color_light.glsl"))
    def draw(
        self,
        mesh: Mesh,
        projection_matrix: glm.mat4,
        model_matrix: glm.mat4,
96: (8)                     camera_matrix: glm.mat4,
97: (8)                     time: float = 0.0,
98: (4)                 ) -> None:
99: (8)                     assert self.program is not None, "There is no program to draw"
100: (8)                    assert mesh.vao is not None, "There is no vao to render"
101: (8)                    if mesh.material is not None:
102: (12)                       if mesh.material.color:
103: (16)                           self.program["color"].value = tuple(mesh.material.color)
104: (12)                       else:
105: (16)                           self.program["color"].value = (1.0, 1.0, 1.0, 1.0)
106: (8)                    self.program["m_proj"].write(projection_matrix)
107: (8)                    self.program["m_model"].write(model_matrix)
108: (8)                    self.program["m_cam"].write(camera_matrix)
109: (8)                    mesh.vao.render(self.program)
110: (4)                def apply(self, mesh: Mesh) -> "MeshProgram" | None:
111: (8)                    if not mesh.material:
112: (12)                       return None
113: (8)                    if not mesh.attributes.get("NORMAL"):
114: (12)                       return None
115: (8)                    return self
116: (0)            class TextureProgram(MeshProgram):
117: (4)                """Plan textured"""
118: (4)                def __init__(self, program: Optional[moderngl.Program] = None, **kwargs:
Any) -> None:
119: (8)                    super().__init__(program=None)
120: (8)                    self.program =
programs.load(ProgramDescription(path="scene_default/texture.glsl"))
121: (4)                def draw(
122: (8)                    self,
123: (8)                    mesh: Mesh,
124: (8)                    projection_matrix: glm.mat4,
125: (8)                    model_matrix: glm.mat4,
126: (8)                    camera_matrix: glm.mat4,
127: (8)                    time: float = 0.0,
128: (4)                ) -> None:
129: (8)                    assert self.program is not None, "There is no program to draw"
130: (8)                    assert mesh.vao is not None, "There is no vao to render"
131: (8)                    assert mesh.material is not None, "There is no material to render"
132: (8)                    assert (
133: (12)                       mesh.material.mat_texture is not None
134: (8)                    ), "The material does not have a texture to render"
135: (8)                    assert (
136: (12)                       mesh.material.mat_texture.texture is not None
137: (8)                    ), "The material texture is not linked to a texture, so it can not be
rendered"
138: (8)                    mesh.material.mat_texture.texture.use()
139: (8)                    self.program["m_proj"].write(projection_matrix)
140: (8)                    self.program["m_model"].write(model_matrix)
141: (8)                    self.program["m_cam"].write(camera_matrix)
142: (8)                    mesh.vao.render(self.program)
143: (4)                def apply(self, mesh: Mesh) -> Optional[MeshProgram]:
144: (8)                    if not mesh.material:
145: (12)                       return None
146: (8)                    if mesh.attributes.get("NORMAL"):
147: (12)                       return None
148: (8)                    if not mesh.attributes.get("TEXCOORD_0"):
149: (12)                       return None
150: (8)                    if mesh.attributes.get("COLOR_0"):
151: (12)                       return None
152: (8)                    if mesh.material.mat_texture is not None:
153: (12)                       return self
154: (8)                    return None
155: (0)            class TextureVertexColorProgram(MeshProgram):
156: (4)                """textured object with vertex color"""
157: (4)                def __init__(self, program: Optional[moderngl.Program] = None, **kwargs:
Any) -> None:
158: (8)                    super().__init__(program=None)
159: (8)                    self.program = programs.load(
160: (12)                       ProgramDescription(path="scene_default/vertex_color_texture.glsl")
```

```
161: (8)                    )
162: (4)                def draw(
163: (8)                    self,
164: (8)                    mesh: Mesh,
165: (8)                    projection_matrix: glm.mat4,
166: (8)                    model_matrix: glm.mat4,
167: (8)                    camera_matrix: glm.mat4,
168: (8)                    time: float = 0.0,
169: (4)                ) -> None:
170: (8)                    assert self.program is not None, "There is no program to draw"
171: (8)                    assert mesh.vao is not None, "There is no vao to render"
172: (8)                    assert mesh.material is not None, "There is no material to render"
173: (8)                    assert (
174: (12)                       mesh.material.mat_texture is not None
175: (8)                    ), "The material does not have a texture to render"
176: (8)                    assert (
177: (12)                       mesh.material.mat_texture.texture is not None
178: (8)                    ), "The material texture is not linked to a texture, so it can not be
rendered"
179: (8)                    mesh.material.mat_texture.texture.use()
180: (8)                    self.program["m_proj"].write(projection_matrix)
181: (8)                    self.program["m_model"].write(model_matrix)
182: (8)                    self.program["m_cam"].write(camera_matrix)
183: (8)                    mesh.vao.render(self.program)
184: (4)                def apply(self, mesh: Mesh) -> "MeshProgram" | None:
185: (8)                    if not mesh.material:
186: (12)                       return None
187: (8)                    if mesh.attributes.get("NORMAL"):
188: (12)                       return None
189: (8)                    if not mesh.attributes.get("TEXCOORD_0"):
190: (12)                       return None
191: (8)                    if not mesh.attributes.get("COLOR_0"):
192: (12)                       return None
193: (8)                    if mesh.material.mat_texture is not None:
194: (12)                       return self
195: (8)                    return None
196: (0)            class TextureLightProgram(MeshProgram):
197: (4)                """
198: (4)                Simple texture program
199: (4)                """
200: (4)                def __init__(self, program: Optional[moderngl.Program] = None, **kwargs:
Any) -> None:
201: (8)                    super().__init__(program=None)
202: (8)                    self.program =
programs.load(ProgramDescription(path="scene_default/texture_light.glsl"))
203: (4)                def draw(
204: (8)                    self,
205: (8)                    mesh: Mesh,
206: (8)                    projection_matrix: glm.mat4,
207: (8)                    model_matrix: glm.mat4,
208: (8)                    camera_matrix: glm.mat4,
209: (8)                    time: float = 0.0,
210: (4)                ) -> None:
211: (8)                    assert self.program is not None, "There is no program to draw"
212: (8)                    assert mesh.vao is not None, "There is no vao to render"
213: (8)                    assert mesh.material is not None, "There is no material to render"
214: (8)                    assert (
215: (12)                       mesh.material.mat_texture is not None
216: (8)                    ), "The material does not have a texture to render"
217: (8)                    assert (
218: (12)                       mesh.material.mat_texture.texture is not None
219: (8)                    ), "The material texture is not linked to a texture, so it can not be
rendered"
220: (8)                    mesh.material.mat_texture.texture.use()
221: (8)                    self.program["texture0"].value = 0
222: (8)                    self.program["m_proj"].write(projection_matrix)
223: (8)                    self.program["m_model"].write(model_matrix)
224: (8)                    self.program["m_cam"].write(camera_matrix)
225: (8)                    mesh.vao.render(self.program)
```

```
(4)                 def apply(self, mesh: Mesh) -> "MeshProgram" | None:
(8)                     if not mesh.material:
(12)                        return None
(8)                     if not mesh.attributes.get("NORMAL"):
(12)                        return None
(8)                     if not mesh.attributes.get("TEXCOORD_0"):
(12)                        return None
(8)                     if mesh.material.mat_texture is not None:
(12)                        return self
(8)                     return None
(0)             class TextureLightColorProgram:
(4)                 pass
(0)             class FallbackProgram(MeshProgram):
(4)                 """
(4)                 Fallback program only rendering positions in white
(4)                 """
(4)                 def __init__(self, program: Optional[moderngl.Program] = None, **kwargs: Any) -> None:
(8)                     super().__init__(program=None)
(8)                     self.program = programs.load(ProgramDescription(path="scene_default/fallback.glsl"))
(4)                 def draw(
(8)                     self,
(8)                     mesh: Mesh,
(8)                     projection_matrix: glm.mat4,
(8)                     model_matrix: glm.mat4,
(8)                     camera_matrix: glm.mat4,
(8)                     time: float = 0.0,
(4)                 ) -> None:
(8)                     assert self.program is not None, "There is no program to draw"
(8)                     assert mesh.vao is not None, "There is no vao to render"
(8)                     self.program["m_proj"].write(projection_matrix)
(8)                     self.program["m_model"].write(model_matrix)
(8)                     self.program["m_cam"].write(camera_matrix)
(8)                     if mesh.material:
(12)                        self.program["color"].value = tuple(mesh.material.color[0:3])
(8)                     else:
(12)                        self.program["color"].value = (1.0, 1.0, 1.0)
(8)                     mesh.vao.render(self.program)
(4)                 def apply(self, mesh: Mesh) -> "MeshProgram" | None:
(8)                     return self
```

File 98 - __init__.py:

```
(0)             from .camera import Camera as Camera
(0)             from .camera import KeyboardCamera as KeyboardCamera
(0)             from .camera import OrbitCamera as OrbitCamera
(0)             from .material import Material as Material
(0)             from .material import MaterialTexture as MaterialTexture
(0)             from .mesh import Mesh as Mesh
(0)             from .node import Node as Node
(0)             from .programs import MeshProgram as MeshProgram
(0)             from .scene import Scene as Scene
(0)             __all__ = [
(4)                 "Camera",
(4)                 "KeyboardCamera",
(4)                 "OrbitCamera",
(4)                 "Material",
(4)                 "MaterialTexture",
(4)                 "Mesh",
(4)                 "Node",
(4)                 "MeshProgram",
(4)                 "Scene",
(0)             ]
```

```
File 99 - decorators.py:

1: (0)              from contextlib import contextmanager
2: (0)              from pathlib import Path
3: (0)              from typing import Generator, Union
4: (0)              from moderngl_window.conf import settings
5: (0)              @contextmanager
6: (0)              def texture_dirs(paths: list[Union[Path, str]]) -> Generator[None, None,
None]:
7: (4)                  """Context manager temporarily replacing texture paths
8: (4)                  Args:
9: (8)                      paths (list[Union[Path, str]]): list of paths
10: (4)                 """
11: (4)                 original_dirs = settings.DATA_DIRS
12: (4)                 settings.TEXTURE_DIRS = paths
13: (4)                 yield None
14: (4)                 settings.TEXTURE_DIRS = original_dirs

----------------------------------------

File 100 - base.py:

1: (0)              from typing import Any, Generator, Optional, Union
2: (0)              import moderngl_window
3: (0)              class FontMeta:
4: (4)                  """Metdata for texture array"""
5: (4)                  def __init__(self, meta: dict[str, Union[int, list[dict[str, int]]]]):
6: (8)                      self._meta = meta
7: (8)                      assert isinstance(self._meta["characters"], int)
8: (8)                      assert isinstance(self._meta["character_height"], int)
9: (8)                      assert isinstance(self._meta["character_width"], int)
10: (8)                     assert isinstance(self._meta["atlas_height"], int)
11: (8)                     assert isinstance(self._meta["atlas_width"], int)
12: (8)                     assert isinstance(self._meta["character_ranges"], list)
13: (8)                     self.characters = self._meta["characters"]
14: (8)                     self.character_ranges = self._meta["character_ranges"]
15: (8)                     self.character_height = self._meta["character_height"]
16: (8)                     self.character_width = self._meta["character_width"]
17: (8)                     self.atlas_height = self._meta["atlas_height"]
18: (8)                     self.atlas_width = self._meta["atlas_width"]
19: (4)                 @property
20: (4)                 def char_aspect_wh(self) -> float:
21: (8)                     return self.character_width / self.character_height
22: (4)                 def char_aspect_hw(self) -> float:
23: (8)                     return self.character_height / self.character_width
24: (0)             class BaseText:
25: (4)                 """Simple base class for a bitmapped text rendered"""
26: (4)                 def __init__(self) -> None:
27: (8)                     self._meta: Optional[FontMeta] = None
28: (8)                     self._ct: list[int] = []
29: (8)                     self.ctx = moderngl_window.ContextRefs.CONTEXT
30: (4)                 def draw(self, *args: Any, **kwargs: Any) -> None:
31: (8)                     raise NotImplementedError()
32: (4)                 def _translate_string(self, data: str) -> Generator[int, None, None]:
33: (8)                     """Translate string into character texture positions"""
34: (8)                     assert (self._meta is not None) and (
35: (12)                        self._ct is not None
36: (8)                     ), "_meta or _ct (or both) are empty. Did you call _init()?"
37: (8)                     data_bytes = data.encode("iso-8859-1", errors="replace")
38: (8)                     for index, char in enumerate(data_bytes):
39: (12)                        yield self._meta.characters - 1 - self._ct[char]
40: (4)                 def _init(self, meta: FontMeta) -> None:
41: (8)                     self._meta = meta
42: (8)                     if not self._meta.characters * self._meta.character_height ==
self._meta.atlas_height:
43: (12)                        raise ValueError("characters * character_width != atlas_height")
44: (8)                     self._generate_character_map()
45: (4)                 def _generate_character_map(self) -> None:
46: (8)                     """Generate character translation map (latin1 pos to texture pos)"""
```

```
47: (8)                     assert self._meta is not None, "You should not call
_generate_character_map but _init"
48: (8)                     self._ct = [-1] * 256
49: (8)                     index = 0
50: (8)                     for c_range in self._meta.character_ranges:
51: (12)                        for c_pos in range(c_range["min"], c_range["max"] + 1):
52: (16)                            self._ct[c_pos] = index
53: (16)                            index += 1
```

File 101 - base.py:

```
1: (0)                  class BaseTimer:
2: (4)                      """
3: (4)                      A timer controls the time passed into the the render function.
4: (4)                      This can be used in creative ways to control the current time
5: (4)                      such as basing it on current location in an audio file.
6: (4)                      All methods must be implemented.
7: (4)                      """
8: (4)                      @property
9: (4)                      def is_paused(self) -> bool:
10: (8)                         """bool: The pause state of the timer"""
11: (8)                         raise NotImplementedError()
12: (4)                     @property
13: (4)                     def is_running(self) -> bool:
14: (8)                         """bool: Is the timer currently running?"""
15: (8)                         raise NotImplementedError()
16: (4)                     @property
17: (4)                     def time(self) -> float:
18: (8)                         """Get or set the current time.
19: (8)                         This can be used to jump around in the timeline.
20: (8)                         Returns:
21: (12)                            float: The current time in seconds
22: (8)                         """
23: (8)                         raise NotImplementedError()
24: (4)                     @time.setter
25: (4)                     def time(self, value: float) -> None:
26: (8)                         raise NotImplementedError()
27: (4)                     @property
28: (4)                     def fps(self) -> float:
29: (8)                         """Get the current frames per second."""
30: (8)                         raise NotImplementedError()
31: (4)                     @property
32: (4)                     def fps_average(self) -> float:
33: (8)                         """get the average fps since the timer was started"""
34: (8)                         raise NotImplementedError()
35: (4)                     def next_frame(self) -> tuple[float, float]:
36: (8)                         """Get timer information for the next frame.
37: (8)                         Returns:
38: (12)                            tuple[float, float]: The frametime and current time
39: (8)                         """
40: (8)                         raise NotImplementedError()
41: (4)                     def start(self) -> None:
42: (8)                         """Start the timer initially or resume after pause"""
43: (8)                         raise NotImplementedError()
44: (4)                     def pause(self) -> None:
45: (8)                         """Pause the timer"""
46: (8)                         raise NotImplementedError()
47: (4)                     def toggle_pause(self) -> None:
48: (8)                         """Toggle pause state"""
49: (8)                         raise NotImplementedError()
50: (4)                     def stop(self) -> tuple[float, float]:
51: (8)                         """
52: (8)                         Stop the timer. Should only be called once when stopping the timer.
53: (8)                         Returns:
54: (12)                            tuple[float, float]> Current position in the timer, actual running
duration
55: (8)                         """
```

```
56: (8)                     raise NotImplementedError()
```

File 102 - clock.py:

```
1: (0)              import time
2: (0)              from typing import Any, Optional
3: (0)              from moderngl_window.timers.base import BaseTimer
4: (0)              class Timer(BaseTimer):
5: (4)                  """Timer based on python ``time``."""
6: (4)                  def __init__(self, **kwargs: Any) -> None:
7: (8)                      self._start_time: Optional[float] = None
8: (8)                      self._stop_time: Optional[float] = None
9: (8)                      self._pause_time: Optional[float] = None
10: (8)                     self._last_frame = 0.0
11: (8)                     self._offset = 0.0
12: (8)                     self._frames = 0  # similar to ticks
13: (8)                     self._fps = 0.0
14: (4)                 @property
15: (4)                 def is_paused(self) -> bool:
16: (8)                     """bool: The pause state of the timer"""
17: (8)                     return self._pause_time is not None
18: (4)                 @property
19: (4)                 def is_running(self) -> bool:
20: (8)                     """bool: Is the timer currently running?"""
21: (8)                     return self._pause_time is None
22: (4)                 @property
23: (4)                 def time(self) -> float:
24: (8)                     """Get or set the current time.
25: (8)                     This can be used to jump around in the timeline.
26: (8)                     Returns:
27: (12)                        The current time in seconds
28: (8)                     """
29: (8)                     if self._start_time is None:
30: (12)                        return 0.0
31: (8)                     if self.is_paused and self._pause_time is not None:
32: (12)                        return self._pause_time - self._offset - self._start_time
33: (8)                     return time.time() - self._start_time - self._offset
34: (4)                 @time.setter
35: (4)                 def time(self, value: float) -> None:
36: (8)                     if value < 0:
37: (12)                        value = 0.0
38: (8)                     self._offset += self.time - value
39: (4)                 @property
40: (4)                 def fps_average(self) -> float:
41: (8)                     """The average fps since the timer was started"""
42: (8)                     if self._frames == 0:
43: (12)                        return 0.0
44: (8)                     return self._frames / self.time
45: (4)                 @property
46: (4)                 def fps(self) -> float:
47: (8)                     """Get the current frames per second."""
48: (8)                     return self._fps
49: (4)                 def next_frame(self) -> tuple[float, float]:
50: (8)                     """
51: (8)                     Get the time and frametime for the next frame.
52: (8)                     This should only be called once per frame.
53: (8)                     Returns:
54: (12)                        tuple[float, float]: current time and frametime
55: (8)                     """
56: (8)                     self._frames += 1
57: (8)                     current = self.time
58: (8)                     delta, self._last_frame = current - self._last_frame, current
59: (8)                     if delta > 0:
60: (12)                        self._fps = 1.0 / delta
61: (8)                     else:
62: (12)                        self._fps = 0.0
63: (8)                     return current, delta
```

```
64: (4)                 def start(self) -> None:
65: (8)                     """Start the timer by recoding the current ``time.time()``
66: (8)                     preparing to report the number of seconds since this timestamp.
67: (8)                     """
68: (8)                     if self._start_time is None:
69: (12)                        self._start_time = time.time()
70: (12)                        self._last_frame = 0.0
71: (8)                     elif self._pause_time is not None:
72: (12)                        self._offset += time.time() - self._pause_time
73: (12)                        self._pause_time = None
74: (8)                     else:
75: (12)                        print("The timer is already started")
76: (4)                 def pause(self) -> None:
77: (8)                     """Pause the timer by setting the internal pause time using
``time.time()``"""
78: (8)                     self._pause_time = time.time()
79: (4)                 def toggle_pause(self) -> None:
80: (8)                     """Toggle the paused state"""
81: (8)                     if self.is_paused:
82: (12)                        self.start()
83: (8)                     else:
84: (12)                        self.pause()
85: (4)                 def stop(self) -> tuple[float, float]:
86: (8)                     """
87: (8)                     Stop the timer. Should only be called once when stopping the timer.
88: (8)                     Returns:
89: (12)                        tuple[float, float]: Current position in the timer, actual running
duration
90: (8)                     """
91: (8)                     if self._start_time is None:
92: (12)                        return 0.0, 0.0
93: (8)                     self._stop_time = time.time()
94: (8)                     return (
95: (12)                        self._stop_time - self._start_time - self._offset,
96: (12)                        self._stop_time - self._start_time,
97: (8)                     )
```

File 103 - text_2d.py:

```
1: (0)                 from pathlib import Path
2: (0)                 from typing import Optional
3: (0)                 import glm
4: (0)                 import moderngl
5: (0)                 import numpy
6: (0)                 from moderngl_window import resources
7: (0)                 from moderngl_window.meta import DataDescription, ProgramDescription,
TextureDescription
8: (0)                 from moderngl_window.opengl.vao import VAO
9: (0)                 from .base import BaseText, FontMeta
10: (0)                resources.register_dir(Path(__file__).parent.resolve())
11: (0)                class TextWriter2D(BaseText):
12: (4)                    """Simple monospaced bitmapped text renderer"""
13: (4)                    def __init__(self) -> None:
14: (8)                        super().__init__()
15: (8)                        meta =
FontMeta(resources.data.load(DataDescription(path="bitmapped/text/meta.json")))
16: (8)                        self._texture = resources.textures.load(
17: (12)                           TextureDescription(
18: (16)                               path="bitmapped/textures/VeraMono.png",
19: (16)                               kind="array",
20: (16)                               mipmap=True,
21: (16)                               layers=meta.characters,
22: (12)                           )
23: (8)                        )
24: (8)                        self._program = resources.programs.load(
25: (12)                           ProgramDescription(path="bitmapped/programs/text_2d.glsl")
26: (8)                        )
```

```
27: (8)                     self._init(meta)
28: (8)                     assert self.ctx is not None, "There was a problem, we do not have a context"
29: (8)                     self._string_buffer = self.ctx.buffer(reserve=1024 * 4)
30: (8)                     self._string_buffer.clear(chunk=b"\32")
31: (8)                     pos = self.ctx.buffer(data=bytes([0] * 4 * 3))
32: (8)                     self._vao = VAO("textwriter", mode=moderngl.POINTS)
33: (8)                     self._vao.buffer(pos, "3f", "in_position")
34: (8)                     self._vao.buffer(self._string_buffer, "1u/i", "in_char_id")
35: (8)                     self._text: Optional[str] = None
36: (4)                 @property
37: (4)                 def text(self) -> Optional[str]:
38: (8)                     return self._text
39: (4)                 @text.setter
40: (4)                 def text(self, value: str) -> None:
41: (8)                     self._text = value
42: (8)                     self._string_buffer.orphan(size=len(value) * 4)
43: (8)                     self._string_buffer.clear(chunk=b"\32")
44: (8)                     self._write(value)
45: (4)                 def _write(self, text: str) -> None:
46: (8)                     self._string_buffer.clear(chunk=b"\32")
47: (8)                     self._string_buffer.write(
48: (12)                        numpy.fromiter(
49: (16)                            self._translate_string(text),
50: (16)                            dtype=numpy.uint32,
51: (12)                        )
52: (8)                     )
53: (4)                 def draw(self, pos: tuple[float, float, float], length: int = -1, size: float = 24.0) -> None:
54: (8)                     assert self.ctx is not None, "There was a problem, we do not have a context"
55: (8)                     assert self.ctx.fbo is not None, "The current context do not have a framebuffer"
56: (8)                     assert self._meta is not None, "We are missing the information needed to write text"
57: (8)                     vp = self.ctx.fbo.viewport
58: (8)                     w, h = vp[2], vp[3]
59: (8)                     projection = glm.ortho(
60: (12)                        0,  # left
61: (12)                        w,  # right
62: (12)                        0,  # bottom
63: (12)                        h,  # top
64: (12)                        1.0,  # near
65: (12)                        -1.0,  # far
66: (8)                     )
67: (8)                     self._texture.use(location=0)
68: (8)                     self._program["m_proj"].write(projection)
69: (8)                     self._program["text_pos"].value = pos
70: (8)                     self._program["font_texture"].value = 0
71: (8)                     self._program["char_size"].value = self._meta.char_aspect_wh * size, size
72: (8)                     self._vao.render(self._program, instances=len(self._text if self._text is not None else ""))

----------------------------------------

File 104 - keymaps.py:

1: (0)              from collections import namedtuple
2: (0)              from typing import Callable
3: (0)              from moderngl_window.context.base.keys import BaseKeys
4: (0)              KeyMap = namedtuple("KeyMap", ["UP", "DOWN", "LEFT", "RIGHT", "FORWARD", "BACKWARD"])
5: (0)              KeyMapFactory = Callable[[BaseKeys], KeyMap]
6: (0)              AZERTY: KeyMapFactory = lambda keys: KeyMap(  # noqa
7: (4)                  UP=keys.A, DOWN=keys.E, LEFT=keys.Q, RIGHT=keys.D, FORWARD=keys.Z, BACKWARD=keys.S
8: (0)              )
9: (0)              QWERTY: KeyMapFactory = lambda keys: KeyMap(  # noqa
10: (4)                 UP=keys.Q, DOWN=keys.E, LEFT=keys.A, RIGHT=keys.D, FORWARD=keys.W,
BACKWARD=keys.S
11: (0)             )
```

File 105 - __init__.py:

```
1: (0)              from .text_2d import TextWriter2D  # noqa
```

File 106 - __init__.py:

```
1: (0)              from .base import BaseTimer as BaseTimer
2: (0)              from .clock import Timer as Timer
3: (0)              __all__ = ["BaseTimer", "Timer"]
```

File 107 - __init__.py:

```
1: (0)              from .scheduler import Scheduler  # noqa
```

File 108 - scheduler.py:

```
1: (0)              import sched
2: (0)              import time
3: (0)              from typing import Any, Callable
4: (0)              from moderngl_window.timers.base import BaseTimer
5: (0)              class Scheduler:
6: (4)                  def __init__(self, timer: BaseTimer):
7: (8)                      """Create a Scheduler object to handle events.
8: (8)                      Args:
9: (12)                         timer (BaseTimer): timer to use, subclass of BaseTimer.
10: (8)                     Raises:
11: (12)                        ValueError: timer is not a valid argument.
12: (8)                     """
13: (8)                     if not isinstance(timer, BaseTimer):
14: (12)                        raise ValueError(
15: (16)                            "timer, {}, has to be a instance of BaseTimer or a
callable!".format(timer)
16: (12)                        )
17: (8)                     self._events: dict[int, sched.Event] = dict()
18: (8)                     self._event_id = 0
19: (8)                     self._scheduler = sched.scheduler(lambda: timer.time, time.sleep)
20: (4)                 def run_once(
21: (8)                     self,
22: (8)                     action: Callable[[Any], Any],
23: (8)                     delay: float,
24: (8)                     *,
25: (8)                     priority: int = 1,
26: (8)                     arguments: tuple[Any, ...] = (),
27: (8)                     kwargs: dict[Any, Any] = dict(),
28: (4)                 ) -> int:
29: (8)                     """Schedule a function for execution after a delay.
30: (8)                     Args:
31: (12)                        action (callable):
32: (16)                            function to be called.
33: (12)                        delay (float):
34: (16)                            delay in seconds.
35: (12)                        priority (int, optional):
36: (16)                            priority for this event, lower is more important. Defaults to
1.
37: (12)                        arguments (tuple, optional):
38: (16)                            arguments for the action. Defaults to ().
39: (12)                        kwargs (dict, optional):
```

```
40: (16)                        keyword arguments for the action. Defaults to dict().
41: (8)                 Returns:
42: (12)                    int: event id that can be canceled.
43: (8)                 """
44: (8)                 event = self._scheduler.enter(delay, priority, action, arguments, kwargs)
45: (8)                 self._events[self._event_id] = event
46: (8)                 self._event_id += 1
47: (8)                 return self._event_id - 1
48: (4)             def run_at(
49: (8)                 self,
50: (8)                 action: Callable[[Any], Any],
51: (8)                 time: float,
52: (8)                 *,
53: (8)                 priority: int = 1,
54: (8)                 arguments: tuple[Any, ...] = (),
55: (8)                 kwargs: dict[Any, Any] = dict(),
56: (4)             ) -> int:
57: (8)                 """Schedule a function to be executed at a certain time.
58: (8)                 Args:
59: (12)                    action (callable):
60: (16)                        function to be called.
61: (12)                    time (float):
62: (16)                        epoch time at which the function should be called.
63: (12)                    priority (int, optional):
64: (16)                        priority for this event, lower is more important. Defaults to 1.
65: (12)                    arguments (tuple, optional):
66: (16)                        arguments for the action. Defaults to ().
67: (12)                    kwargs (dict, optional):
68: (16)                        keyword arguments for the action. Defaults to dict().
69: (8)                 Returns:
70: (12)                    int: event id that can be canceled.
71: (8)                 """
72: (8)                 event = self._scheduler.enterabs(time, priority, action, arguments, kwargs)
73: (8)                 self._events[self._event_id] = event
74: (8)                 self._event_id += 1
75: (8)                 return self._event_id - 1
76: (4)             def run_every(
77: (8)                 self,
78: (8)                 action: Callable[[Any], Any],
79: (8)                 delay: float,
80: (8)                 *,
81: (8)                 priority: int = 1,
82: (8)                 initial_delay: float = 0.0,
83: (8)                 arguments: tuple[Any, ...] = (),
84: (8)                 kwargs: dict[Any, Any] = dict(),
85: (4)             ) -> int:
86: (8)                 """Schedule a recurring function to be called every `delay` seconds after a initial delay.
87: (8)                 Args:
88: (12)                    action (callable):
89: (16)                        function to be called.
90: (12)                    delay (float):
91: (16)                        delay in seconds.
92: (12)                    priority (int, optional):
93: (16)                        priority for this event, lower is more important. Defaults to 1.
94: (12)                    initial_delay (float, optional):
95: (16)                        initial delay in seconds before executing for the first time.
96: (12)                    Defaults to 0. arguments (tuple, optional):
97: (16)                        arguments for the action. Defaults to ().
98: (12)                    kwargs (dict, optional):
99: (16)                        keyword arguments for the action. Defaults to dict().
100: (8)                Returns:
101: (12)                   int: event id that can be canceled.
102: (8)                """
103: (8)                recurring_event = self._recurring_event_factory(
104: (12)                       action, arguments, kwargs, (delay, priority), self._event_id
105: (8)                    )
106: (8)                    event = self._scheduler.enter(initial_delay, priority, recurring_event)
107: (8)                    self._events[self._event_id] = event
108: (8)                    self._event_id += 1
109: (8)                    return self._event_id - 1
110: (4)                def _recurring_event_factory(
111: (8)                    self,
112: (8)                    function: Callable[[Any], Any],
113: (8)                    arguments: tuple[Any, ...],
114: (8)                    kwargs: dict[Any, Any],
115: (8)                    scheduling_info: tuple[Any, Any],
116: (8)                    id: int,
117: (4)                ) -> Callable[[], None]:
118: (8)                    """Factory for creating recurring events that will reschedule themselves.
119: (8)                    Args:
120: (12)                       function (callable): function to be called.
121: (12)                       arguments (tuple): arguments for the function.
122: (12)                       kwargs (dict): keyword arguments for the function.
123: (12)                       scheduling_info (tuple): tuple of information for scheduling the task.
124: (12)                       id (int): event id this event should be assigned to.
125: (8)                    """
126: (8)                    def _f() -> None:
127: (12)                       function(*arguments, **kwargs)
128: (12)                       event = self._scheduler.enter(*scheduling_info, _f)
129: (12)                       self._events[id] = event
130: (8)                    return _f
131: (4)                def execute(self) -> None:
132: (8)                    """Run the scheduler without blocking and execute any expired events."""
133: (8)                    self._scheduler.run(blocking=False)
134: (4)                def cancel(self, event_id: int, delay: float = 0) -> None:
135: (8)                    """Cancel a previously scheduled event.
136: (8)                    Args:
137: (12)                       event_id (int): event to be canceled
138: (12)                       delay (float, optional): delay before canceling the event. Defaults to 0.
139: (8)                    """
140: (8)                    if delay == 0:
141: (12)                       self._cancel(event_id)
142: (8)                    else:
143: (12)                       self.run_once(self._cancel, delay, priority=0, arguments=(event_id,))
144: (4)                def _cancel(self, event_id: int) -> None:
145: (8)                    if event_id not in self._events:
146: (12)                       raise ValueError("Recurring event with id {} does not exist".format(event_id))
147: (8)                    event = self._events.pop(event_id)
148: (8)                    self._scheduler.cancel(event)

----------------------------------------

File 109 - module_loading.py:

1: (0)              from importlib import import_module
2: (0)              from typing import Any
3: (0)              def import_string(dotted_path: str) -> Any:
4: (4)                  """
5: (4)                  Import a dotted module path and return the attribute/class designated by the
6: (4)                  last name in the path. Raise ImportError if the import failed.
7: (4)                  Args:
8: (8)                      dotted_path: The path to attempt importing
9: (4)                  Returns:
10: (8)                     Imported class/attribute
11: (4)                 """
```

```
12: (4)                 try:
13: (8)                     module_path, class_name = dotted_path.rsplit(".", 1)
14: (4)                 except ValueError as err:
15: (8)                     raise ImportError("%s doesn't look like a module path" % dotted_path) from err
16: (4)                 module = import_module(module_path)
17: (4)                 try:
18: (8)                     return getattr(module, class_name)
19: (4)                 except AttributeError as err:
20: (8)                     raise ImportError(
21: (12)                        'Module "%s" does not define a "%s" attribute/class' % (module_path, class_name)
22: (8)                     ) from err

----------------------------------------

File 110 - SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRYCOMBINER_aligner_20_characters_for_pythons_codes.py:

1: (0)              import os
2: (0)              from datetime import datetime
3: (0)              def get_file_info(root_folder):
4: (4)                  file_info_list = []
5: (4)                  for root, dirs, files in os.walk(root_folder):
6: (8)                      for file in files:
7: (12)                         try:
8: (16)                             if file.endswith('.py'):
9: (20)                                 file_path = os.path.join(root, file)
10: (20)                                creation_time = datetime.fromtimestamp(os.path.getctime(file_path))
11: (20)                                modified_time = datetime.fromtimestamp(os.path.getmtime(file_path))
12: (20)                                file_extension = os.path.splitext(file)[1].lower()
13: (20)                                file_info_list.append([file, file_path, creation_time, modified_time, file_extension, root])
14: (12)                        except Exception as e:
15: (16)                            print(f"Error processing file {file}: {e}")
16: (4)                 file_info_list.sort(key=lambda x: (x[2], x[3], len(x[0]), x[4]))  # Sort by creation, modification time, name length, extension
17: (4)                 return file_info_list
18: (0)             def process_file(file_info_list):
19: (4)                 combined_output = []
20: (4)                 for idx, (file_name, file_path, creation_time, modified_time, file_extension, root) in enumerate(file_info_list):
21: (8)                     with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
22: (12)                        content = f.read()
23: (12)                        content = "\n".join([line for line in content.split('\n') if line.strip() and not line.strip().startswith("#")])
24: (12)                        content = content.replace('\t', '    ')
25: (12)                        processed_lines = []
26: (12)                        for i, line in enumerate(content.split('\n')):
27: (16)                            leading_spaces = len(line) - len(line.lstrip(' '))
28: (16)                            line_number_str = f"{i+1}: ({leading_spaces})"
29: (16)                            padding = ' ' * (20 - len(line_number_str))
30: (16)                            processed_line = f"{line_number_str}{padding}{line}"
31: (16)                            processed_lines.append(processed_line)
32: (12)                        content_with_line_numbers = "\n".join(processed_lines)
33: (12)                        combined_output.append(f"File {idx + 1} - {file_name}:\n")
34: (12)                        combined_output.append(content_with_line_numbers)
35: (12)                        combined_output.append("\n" + "-"*40 + "\n")
36: (4)                 return combined_output
37: (0)             root_folder_path = '.'  # Set this to the desired folder
38: (0)             file_info_list = get_file_info(root_folder_path)
39: (0)             combined_output = process_file(file_info_list)
40: (0)             output_file = 'SANJOYNATHQHENOMENOLOGYGEOMETRIFYINGTRIGONOMETRY_combined_python_files_20_chars.txt'
41: (0)             with open(output_file, 'w', encoding='utf-8') as logfile:
42: (4)                 logfile.write("\n".join(combined_output))
```

```
43: (0)             print(f"Processed file info logged to {output_file}")

----------------------------------------
```