pyrender Documentation

Release 0.1.45

Matthew Matl

Jun 28, 2021

Contents

Pyrender is a pure Python (2.7, 3.4, 3.5, 3.6) library for physically-based rendering and visualization. It is designed to meet the glTF 2.0 specification from Khronos

Pyrender is lightweight, easy to install, and simple to use. It comes packaged with both an intuitive scene viewer and a headache-free offscreen renderer with support for GPU-accelerated rendering on headless servers, which makes it perfect for machine learning applications. Check out the *User Guide* for a full tutorial, or fork me on Github.

CHAPTER 1

Installation Guide

1.1 Python Installation

This package is available via `pip`.

```
pip install pyrender
```

If you're on MacOS, you'll need to pre-install my fork of `pyglet`, as the version on PyPI hasn't yet included my change that enables OpenGL contexts on MacOS.

```
git clone https://github.com/mmatl/pyglet.git
cd pyglet
pip install .
```

1.2 Getting Pyrender Working with OSMesa

If you want to render scenes offscreen but don't want to have to install a display manager or deal with the pains of trying to get OpenGL to work over SSH, you have two options.

The first (and preferred) option is using EGL, which enables you to perform GPU-accelerated rendering on headless servers. However, you'll need EGL 1.5 to get modern OpenGL contexts. This comes packaged with NVIDIA's current drivers, but if you are having issues getting EGL to work with your hardware, you can try using OSMesa, a software-based offscreen renderer that is included with any Mesa install.

If you want to use OSMesa with pyrender, you'll have to perform two additional installation steps:

- *Installing OSMesa*
- *Installing a Compatible Fork of PyOpenGL*

Then, read the offscreen rendering tutorial. See *Offscreen Rendering*.

1.2.1 Installing OSMesa

As a first step, you'll need to rebuild and re-install Mesa with support for fast offscreen rendering and OpenGL 3+ contexts. I'd recommend installing from source, but you can also try my `.deb` for Ubuntu 16.04 and up.

1.2.2 Installing from a Debian Package

If you're running Ubuntu 16.04 or newer, you should be able to install the required version of Mesa from my `.deb` file.

```
sudo apt update
sudo wget https://github.com/mmatl/travis_debs/raw/master/xenial/mesa_18.3.3-0.deb
sudo dpkg -i ./mesa_18.3.3-0.deb || true
sudo apt install -f
```

If this doesn't work, try building from source.

1.2.3 Building From Source

First, install build dependencies via *apt* or your system's package manager.

```
sudo apt-get install llvm-6.0 freeglut3 freeglut3-dev
```

Then, download the current release of Mesa from here. Unpack the source and go to the source folder:

```
tar xfv mesa-18.3.3.tar.gz
cd mesa-18.3.3
```

Replace `PREFIX` with the path you want to install Mesa at. If you're not worried about overwriting your default Mesa install, a good place is at `/usr/local`.

Now, configure the installation by running the following command:

```
./configure --prefix=PREFIX                                    \
            --enable-opengl --disable-gles1 --disable-gles2    \
            --disable-va --disable-xvmc --disable-vdpau        \
            --enable-shared-glapi                              \
            --disable-texture-float                            \
            --enable-gallium-llvm --enable-llvm-shared-libs    \
            --with-gallium-drivers=swrast,swr                  \
            --disable-dri --with-dri-drivers=                  \
            --disable-egl --with-egl-platforms= --disable-gbm  \
            --disable-glx                                      \
            --disable-osmesa --enable-gallium-osmesa           \
            ac_cv_path_LLVM_CONFIG=llvm-config-6.0
```

Finally, build and install Mesa.

```
make -j8
make install
```

Finally, if you didn't install Mesa in the system path, add the following lines to your `~/.bashrc` file after changing `MESA_HOME` to your mesa installation path (i.e. what you used as `PREFIX` during the configure command).

```
MESA_HOME=/path/to/your/mesa/installation
export LIBRARY_PATH=$LIBRARY_PATH:$MESA_HOME/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MESA_HOME/lib
export C_INCLUDE_PATH=$C_INCLUDE_PATH:$MESA_HOME/include/
export CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH:$MESA_HOME/include/
```

1.2.4 Installing a Compatible Fork of PyOpenGL

Next, install and use my fork of `PyOpenGL`. This fork enables getting modern OpenGL contexts with OSMesa. My patch has been included in `PyOpenGL`, but it has not yet been released on PyPI.

```
git clone https://github.com/mmatl/pyopengl.git
pip install ./pyopengl
```

1.3 Building Documentation

The online documentation for `pyrender` is automatically built by Read The Docs. Building `pyrender`'s documentation locally requires a few extra dependencies – specifically, sphinx and a few plugins.

To install the dependencies required, simply change directories into the *pyrender* source and run

```
$ pip install .[docs]
```

Then, go to the `docs` directory and run `make` with the appropriate target. For example,

```
$ cd docs/
$ make html
```

will generate a set of web pages. Any documentation files generated in this manner can be found in `docs/build`.

CHAPTER 2

User Guide

This section contains guides on how to use Pyrender to quickly visualize your 3D data, including a quickstart guide and more detailed descriptions of each part of the rendering pipeline.

2.1 Quickstart

2.1.1 Minimal Example for 3D Viewer

Here is a minimal example of loading and viewing a triangular mesh model in pyrender.

```
>>> import trimesh
>>> import pyrender
>>> fuze_trimesh = trimesh.load('examples/models/fuze.obj')
>>> mesh = pyrender.Mesh.from_trimesh(fuze_trimesh)
>>> scene = pyrender.Scene()
>>> scene.add(mesh)
>>> pyrender.Viewer(scene, use_raymond_lighting=True)
```

2.1.2 Minimal Example for Offscreen Rendering

Note: If you're using a headless server, make sure that you followed the guide for installing OSMesa. See *Getting Pyrender Working with OSMesa*.

Here is a minimal example of rendering a mesh model offscreen in pyrender. The only additional necessities are that you need to add lighting and a camera.

```
>>> import numpy as np
>>> import trimesh
>>> import pyrender
>>> import matplotlib.pyplot as plt
```

```
>>> fuze_trimesh = trimesh.load('examples/models/fuze.obj')
>>> mesh = pyrender.Mesh.from_trimesh(fuze_trimesh)
>>> scene = pyrender.Scene()
>>> scene.add(mesh)
>>> camera = pyrender.PerspectiveCamera(yfov=np.pi / 3.0, aspectRatio=1.0)
>>> s = np.sqrt(2)/2
>>> camera_pose = np.array([
...    [0.0, -s,   s,   0.3],
```

(continues on next page)

(continued from previous page)

```
...     [1.0,  0.0, 0.0, 0.0],
...     [0.0,  s,   s,   0.35],
...     [0.0,  0.0, 0.0, 1.0],
... ])
>>> scene.add(camera, pose=camera_pose)
>>> light = pyrender.SpotLight(color=np.ones(3), intensity=3.0,
...                            innerConeAngle=np.pi/16.0,
...                            outerConeAngle=np.pi/6.0)
>>> scene.add(light, pose=camera_pose)
>>> r = pyrender.OffscreenRenderer(400, 400)
>>> color, depth = r.render(scene)
>>> plt.figure()
>>> plt.subplot(1,2,1)
>>> plt.axis('off')
>>> plt.imshow(color)
>>> plt.subplot(1,2,2)
>>> plt.axis('off')
>>> plt.imshow(depth, cmap=plt.cm.gray_r)
>>> plt.show()
```

2.2 Loading and Configuring Models

The first step to any rendering application is loading your models. Pyrender implements the GLTF 2.0 specification, which means that all models are composed of a hierarchy of objects.

At the top level, we have a *Mesh*. The *Mesh* is basically a wrapper of any number of *Primitive* types, which actually represent geometry that can be drawn to the screen.

Primitives are composed of a variety of parameters, including vertex positions, vertex normals, color and texture information, and triangle indices if smooth rendering is desired. They can implement point clouds, triangular meshes, or lines depending on how you configure their data and set their *Primitive.mode* parameter.

Although you can create primitives yourself if you want to, it's probably easier to just use the utility functions provided in the *Mesh* class.

2.2.1 Creating Triangular Meshes

Simple Construction

Pyrender allows you to create a *Mesh* containing a triangular mesh model directly from a `Trimesh` object using the *Mesh.from_trimesh()* static method.

```
>>> import trimesh
>>> import pyrender
>>> import numpy as np
>>> tm = trimesh.load('examples/models/fuze.obj')
>>> m = pyrender.Mesh.from_trimesh(tm)
>>> m.primitives
[<pyrender.primitive.Primitive at 0x7fbb0af60e50>]
```

You can also create a single *Mesh* from a list of `Trimesh` objects:

```
>>> tms = [trimesh.creation.icosahedron(), trimesh.creation.cylinder()]
>>> m = pyrender.Mesh.from_trimesh(tms)
```

(continues on next page)

(continued from previous page)

```
[<pyrender.primitive.Primitive at 0x7fbb0c2b74d0>,
 <pyrender.primitive.Primitive at 0x7fbb0c2b7550>]
```

Vertex Smoothing

The *Mesh.from_trimesh()* method has a few additional optional parameters. If you want to render the mesh without interpolating face normals, which can be useful for meshes that are supposed to be angular (e.g. a cube), you can specify `smooth=False`.

```
>>> m = pyrender.Mesh.from_trimesh(tm, smooth=False)
```

Per-Face or Per-Vertex Coloration

If you have an untextured trimesh, you can color it in with per-face or per-vertex colors:

```
>>> tm.visual.vertex_colors = np.random.uniform(size=tm.vertices.shape)
>>> tm.visual.face_colors = np.random.uniform(size=tm.faces.shape)
>>> m = pyrender.Mesh.from_trimesh(tm)
```

Instancing

If you want to render many copies of the same mesh at different poses, you can statically create a vast array of them in an efficient manner. Simply specify the `poses` parameter to be a list of `N` 4x4 homogenous transformation matrics that position the meshes relative to their common base frame:

```
>>> tfs = np.tile(np.eye(4), (3,1,1))
>>> tfs[1,:3,3] = [0.1, 0.0, 0.0]
>>> tfs[2,:3,3] = [0.2, 0.0, 0.0]
>>> tfs
array([[[1. , 0. , 0. , 0. ],
        [0. , 1. , 0. , 0. ],
        [0. , 0. , 1. , 0. ],
        [0. , 0. , 0. , 1. ]],
       [[1. , 0. , 0. , 0.1],
        [0. , 1. , 0. , 0. ],
        [0. , 0. , 1. , 0. ],
        [0. , 0. , 0. , 1. ]],
       [[1. , 0. , 0. , 0.2],
        [0. , 1. , 0. , 0. ],
        [0. , 0. , 1. , 0. ],
        [0. , 0. , 0. , 1. ]]])
```

```
>>> m = pyrender.Mesh.from_trimesh(tm, poses=tfs)
```

Custom Materials

You can also specify a custom material for any triangular mesh you create in the `material` parameter of *Mesh.from_trimesh()*. The main material supported by Pyrender is the *MetallicRoughnessMaterial*. The metallic-roughness model supports rendering highly-realistic objects across a wide gamut of materials.

For more information, see the documentation of the *MetallicRoughnessMaterial* constructor or look at the Khronos documentation for more information.

2.2.2 Creating Point Clouds

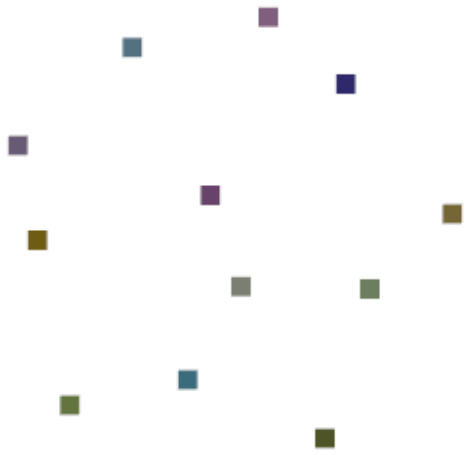
Point Sprites

Pyrender also allows you to create a *Mesh* containing a point cloud directly from `numpy.ndarray` instances using the *Mesh.from_points()* static method.

Simply provide a list of points and optional per-point colors and normals.

```
>>> pts = tm.vertices.copy()
>>> colors = np.random.uniform(size=pts.shape)
>>> m = pyrender.Mesh.from_points(pts, colors=colors)
```

Point clouds created in this way will be rendered as square point sprites.



Point Spheres

If you have a monochromatic point cloud and would like to render it with spheres, you can render it by instancing a spherical trimesh:

```
>>> sm = trimesh.creation.uv_sphere(radius=0.1)
>>> sm.visual.vertex_colors = [1.0, 0.0, 0.0]
>>> tfs = np.tile(np.eye(4), (len(pts), 1, 1))
>>> tfs[:,:3,3] = pts
>>> m = pyrender.Mesh.from_trimesh(sm, poses=tfs)
```

2.3 Creating Lights

Pyrender supports three types of punctual light:

- *PointLight*: Point-based light sources, such as light bulbs.
- *SpotLight*: A conical light source, like a flashlight.
- *DirectionalLight*: A general light that does not attenuate with distance.

Creating lights is easy – just specify their basic attributes:

```
>>> pl = pyrender.PointLight(color=[1.0, 1.0, 1.0], intensity=2.0)
>>> sl = pyrender.SpotLight(color=[1.0, 1.0, 1.0], intensity=2.0,
...                         innerConeAngle=0.05, outerConeAngle=0.5)
>>> dl = pyrender.DirectionalLight(color=[1.0, 1.0, 1.0], intensity=2.0)
```

For more information about how these lighting models are implemented, see their class documentation.

2.4 Creating Cameras

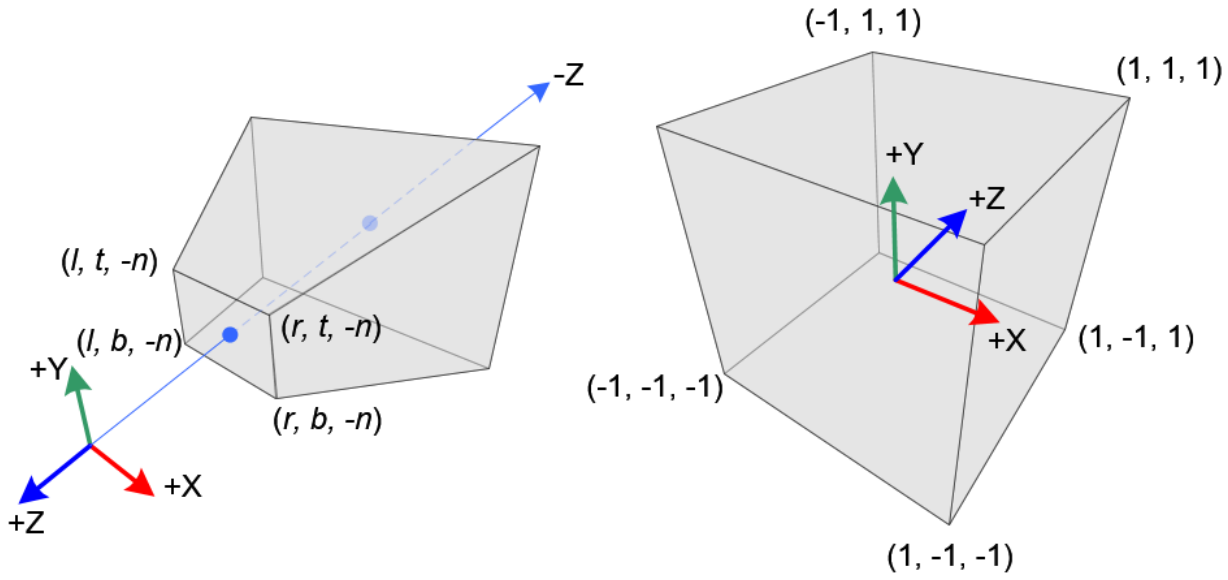
Pyrender supports three camera types – *PerspectiveCamera* and *IntrinsicsCamera* types, which render scenes as a human would see them, and *OrthographicCamera* types, which preserve distances between points.

Creating cameras is easy – just specify their basic attributes:

```
>>> pc = pyrender.PerspectiveCamera(yfov=np.pi / 3.0, aspectRatio=1.414)
>>> oc = pyrender.OrthographicCamera(xmag=1.0, ymag=1.0)
```

For more information, see the Khronos group's documentation here:

When you add cameras to the scene, make sure that you're using OpenGL camera coordinates to specify their pose. See the illustration below for details. Basically, the camera z-axis points away from the scene, the x-axis points right in image space, and the y-axis points up in image space.



2.5 Creating Scenes

Before you render anything, you need to put all of your lights, cameras, and meshes into a scene. The *Scene* object keeps track of the relative poses of these primitives by inserting them into *Node* objects and keeping them in a directed acyclic graph.

2.5.1 Adding Objects

To create a *Scene*, simply call the constructor. You can optionally specify an ambient light color and a background color:

```
>>> scene = pyrender.Scene(ambient_light=[0.02, 0.02, 0.02],
...                        bg_color=[1.0, 1.0, 1.0])
```

You can add objects to a scene by first creating a *Node* object and adding the object and its pose to the *Node*. Poses are specified as 4x4 homogenous transformation matrices that are stored in the node's *Node.matrix* attribute. Note that the *Node* constructor requires you to specify whether you're adding a mesh, light, or camera.

```
>>> mesh = pyrender.Mesh.from_trimesh(tm)
>>> light = pyrender.PointLight(color=[1.0, 1.0, 1.0], intensity=2.0)
>>> cam = pyrender.PerspectiveCamera(yfov=np.pi / 3.0, aspectRatio=1.414)
>>> nm = pyrender.Node(mesh=mesh, matrix=np.eye(4))
>>> nl = pyrender.Node(light=light, matrix=np.eye(4))
>>> nc = pyrender.Node(camera=cam, matrix=np.eye(4))
>>> scene.add_node(nm)
>>> scene.add_node(nl)
>>> scene.add_node(nc)
```

You can also add objects directly to a scene with the *Scene.add()* function, which takes care of creating a *Node* for you.

```
>>> scene.add(mesh, pose=np.eye(4))
>>> scene.add(light, pose=np.eye(4))
>>> scene.add(cam, pose=np.eye(4))
```

Nodes can be hierarchical, in which case the node's *Node.matrix* specifies that node's pose relative to its parent frame. You can add nodes to a scene hierarchically by specifying a parent node in your calls to *Scene.add()* or *Scene.add_node()*:

```
>>> scene.add_node(nl, parent_node=nc)
>>> scene.add(cam, parent_node=nm)
```

If you add multiple cameras to a scene, you can specify which one to render from by setting the *Scene.main_camera_node* attribute.

2.5.2 Updating Objects

You can update the poses of existing nodes with the *Scene.set_pose()* function. Simply call it with a *Node* that is already in the scene and the new pose of that node with respect to its parent as a 4x4 homogenous transformation matrix:

```
>>> scene.set_pose(nl, pose=np.eye(4))
```

If you want to get the local pose of a node, you can just access its *Node.matrix* attribute. However, if you want to the get the pose of a node *with respect to the world frame*, you can call the *Scene.get_pose()* method.

```
>>> tf = scene.get_pose(nl)
```

2.5.3 Removing Objects

Finally, you can remove a *Node* and all of its children from the scene with the *Scene.remove_node()* function:

```
>>> scene.remove_node(nl)
```

2.6 Offscreen Rendering

Note: If you're using a headless server, you'll need to use either EGL (for GPU-accelerated rendering) or OSMesa (for CPU-only software rendering). If you're using OSMesa, be sure that you've installed it properly. See *Getting*

Pyrender Working with OSMesa for details.

2.6.1 Choosing a Backend

Once you have a scene set up with its geometry, cameras, and lights, you can render it using the *OffscreenRenderer*. Pyrender supports three backends for offscreen rendering:

- Pyglet, the same engine that runs the viewer. This requires an active display manager, so you can't run it on a headless server. This is the default option.
- OSMesa, a software renderer.
- EGL, which allows for GPU-accelerated rendering without a display manager.

If you want to use OSMesa or EGL, you need to set the `PYOPENGL_PLATFORM` environment variable before importing pyrender or any other OpenGL library. You can do this at the command line:

```
PYOPENGL_PLATFORM=osmesa python render.py
```

or at the top of your Python script:

```
# Top of main python script
import os
os.environ['PYOPENGL_PLATFORM'] = 'egl'
```

The handle for EGL is `egl`, and the handle for OSMesa is `osmesa`.

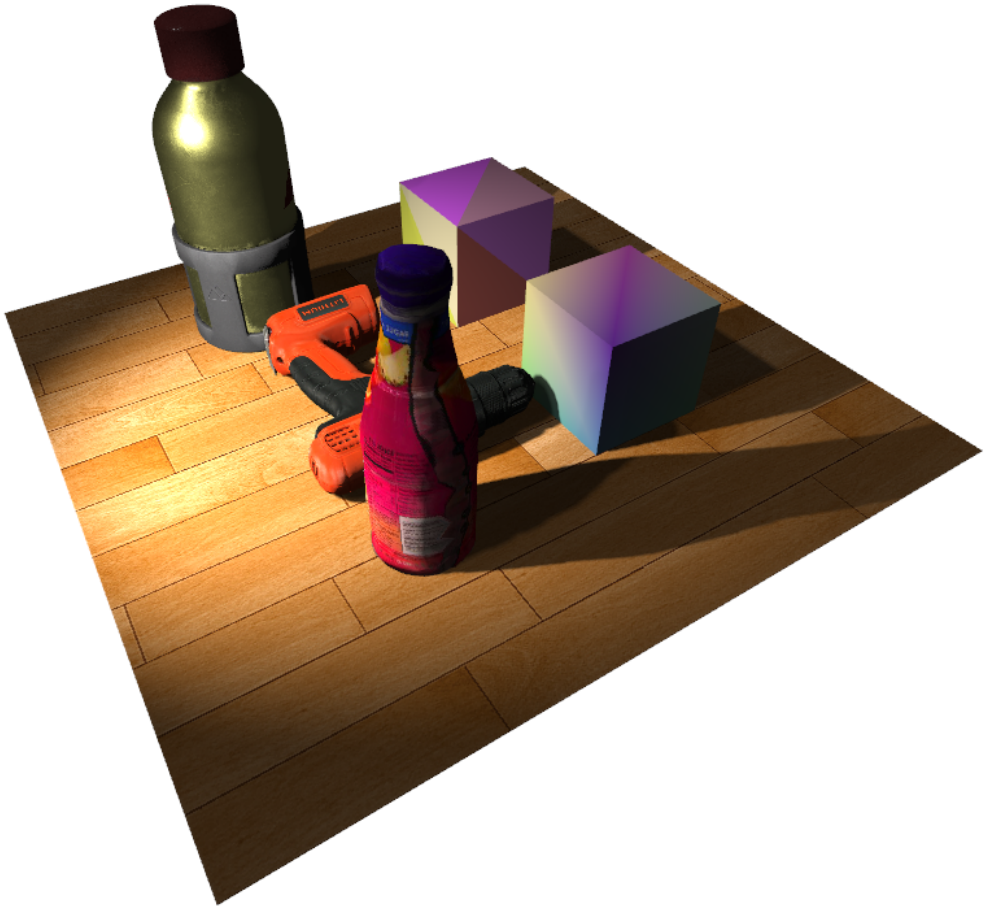
2.6.2 Running the Renderer

Once you've set your environment variable appropriately, create your scene and then configure the *OffscreenRenderer* object with a window width, a window height, and a size for point-cloud points:

```
>>> r = pyrender.OffscreenRenderer(viewport_width=640,
...                                viewport_height=480,
...                                point_size=1.0)
```

Then, just call the *OffscreenRenderer.render()* function:

```
>>> color, depth = r.render(scene)
```



This will return a `(w,h,3)` channel floating-point color image and a `(w,h)` floating-point depth image rendered from the scene's main camera.

You can customize the rendering process by using flag options from *RenderFlags* and bitwise or-ing them together. For example, the following code renders a color image with an alpha channel and enables shadow mapping for all directional lights:

```
>>> flags = RenderFlags.RGBA | RenderFlags.SHADOWS_DIRECTIONAL
>>> color, depth = r.render(scene, flags=flags)
```

Once you're done with the offscreen renderer, you need to close it before you can run a different renderer or open the viewer for the same scene:

```
>>> r.delete()
```

2.6.3 Google CoLab Examples

For a minimal working example of offscreen rendering using OSMesa, see the OSMesa Google CoLab notebook.

For a minimal working example of offscreen rendering using EGL, see the EGL Google CoLab notebook.

2.7 Live Scene Viewer

2.7.1 Standard Usage

In addition to the offscreen renderer, Pyrender comes with a live scene viewer. In its standard invocation, calling the *Viewer*'s constructor will immediately pop a viewing window that you can navigate around in.

```
>>> pyrender.Viewer(scene)
```

By default, the viewer uses your scene's lighting. If you'd like to start with some additional lighting that moves around with the camera, you can specify that with:

```
>>> pyrender.Viewer(scene, use_raymond_lighting=True)
```

For a full list of the many options that the *Viewer* supports, check out its documentation.

2.7.2 Running the Viewer in a Separate Thread

If you'd like to animate your models, you'll want to run the viewer in a separate thread so that you can update the scene while the viewer is running. To do this, first pop the viewer in a separate thread by calling its constructor with the `run_in_thread` option set:

```
>>> v = pyrender.Viewer(scene, run_in_thread=True)
```

Then, you can manipulate the *Scene* while the viewer is running to animate things. However, be careful to acquire the viewer's *Viewer.render_lock* before editing the scene to prevent data corruption:

```
>>> i = 0
>>> while True:
...     pose = np.eye(4)
...     pose[:3,3] = [i, 0, 0]
...     v.render_lock.acquire()
...     scene.set_pose(mesh_node, pose)
...     v.render_lock.release()
...     i += 0.01
```

You can wait on the viewer to be closed manually:

```
>>> while v.is_active:
...     pass
```

Or you can close it from the main thread forcibly. Make sure to still loop and block for the viewer to actually exit before using the scene object again.

```
>>> v.close_external()
>>> while v.is_active:
...     pass
```

CHAPTER 3

Pyrender API Documentation

3.1 Constants

3.1.1 Classes

RenderFlags	Flags for rendering in the scene.
TextAlign	Text alignment options for captions.
GLTF	Options for GL objects.

RenderFlags

class pyrender.constants.**RenderFlags**

Bases: object

Flags for rendering in the scene.

Combine them with the bitwise or. For example,

```
>>> flags = OFFSCREEN | SHADOWS_DIRECTIONAL | VERTEX_NORMALS
```

would result in an offscreen render with directional shadows and vertex normals enabled.

Attributes Summary

ALL_SOLID	Render all meshes as solids.
ALL_WIREFRAME	Render all meshes as wireframes.
DEPTH_ONLY	Only render the depth buffer.
FACE_NORMALS	Render face normals.
FLAT	Render the color buffer flat, with no lighting computations.

Continued on next page

Table 2 – continued from previous page

FLIP_WIREFRAME	Invert the status of wireframe rendering for each mesh.
NONE	Normal PBR Render.
OFFSCREEN	Render offscreen and return the depth and (optionally) color buffers.
RGBA	Render the color buffer with the alpha channel enabled.
SEG	
SHADOWS_ALL	Render shadows for all lights.
SHADOWS_DIRECTIONAL	Render shadows for directional lights.
SHADOWS_POINT	Render shadows for point lights.
SHADOWS_SPOT	Render shadows for spot lights.
SKIP_CULL_FACES	Do not cull back faces.
VERTEX_NORMALS	Render vertex normals.

Attributes Documentation

ALL_SOLID = 16
Render all meshes as solids.

ALL_WIREFRAME = 8
Render all meshes as wireframes.

DEPTH_ONLY = 1
Only render the depth buffer.

FACE_NORMALS = 512
Render face normals.

FLAT = 4096
Render the color buffer flat, with no lighting computations.

FLIP_WIREFRAME = 4
Invert the status of wireframe rendering for each mesh.

NONE = 0
Normal PBR Render.

OFFSCREEN = 2
Render offscreen and return the depth and (optionally) color buffers.

RGBA = 2048
Render the color buffer with the alpha channel enabled.

SEG = 8192

SHADOWS_ALL = 224
Render shadows for all lights.

SHADOWS_DIRECTIONAL = 32
Render shadows for directional lights.

SHADOWS_POINT = 64
Render shadows for point lights.

SHADOWS_SPOT = 128
Render shadows for spot lights.

SKIP_CULL_FACES = 1024
Do not cull back faces.

VERTEX_NORMALS = 256
Render vertex normals.

TextAlign

class pyrender.constants.**TextAlign**
Bases: object

Text alignment options for captions.

Only use one at a time.

Attributes Summary

BOTTOM_CENTER	Center the text by width and fix it to the bottom.
BOTTOM_LEFT	Put the text in the bottom-left corner.
BOTTOM_RIGHT	Put the text in the bottom-right corner.
CENTER	Center the text by width and height.
CENTER_LEFT	Center the text by height and left-align it.
CENTER_RIGHT	Center the text by height and right-align it.
TOP_CENTER	Center the text by width and fix it to the top.
TOP_LEFT	Put the text in the top-left corner.
TOP_RIGHT	Put the text in the top-right corner.

Attributes Documentation

BOTTOM_CENTER = 5
Center the text by width and fix it to the bottom.

BOTTOM_LEFT = 3
Put the text in the bottom-left corner.

BOTTOM_RIGHT = 4
Put the text in the bottom-right corner.

CENTER = 0
Center the text by width and height.

CENTER_LEFT = 1
Center the text by height and left-align it.

CENTER_RIGHT = 2
Center the text by height and right-align it.

TOP_CENTER = 8
Center the text by width and fix it to the top.

TOP_LEFT = 6
Put the text in the top-left corner.

TOP_RIGHT = 7
Put the text in the top-right corner.

GLTF

class pyrender.constants.**GLTF**
Bases: object

Options for GL objects.

Attributes Summary

CLAMP_TO_EDGE	Clamp to the edge of the texture.
LINEAR	Linear interpolation.
LINEAR_MIPMAP_LINEAR	Linear mipmapping.
LINEAR_MIPMAP_NEAREST	Linear mipmapping.
LINES	Render as lines.
LINE_LOOP	Render as a line loop.
LINE_STRIP	Render as a line strip.
MIRRORED_REPEAT	Mirror the texture.
NEAREST	Nearest neighbor interpolation.
NEAREST_MIPMAP_LINEAR	Nearest mipmapping.
NEAREST_MIPMAP_NEAREST	Nearest mipmapping.
POINTS	Render as points.
REPEAT	Repeat the texture.
TRIANGLES	Render as triangles.
TRIANGLE_FAN	Render as a triangle fan.
TRIANGLE_STRIP	Render as a triangle strip.

Attributes Documentation

CLAMP_TO_EDGE = 33071
Clamp to the edge of the texture.

LINEAR = 9729
Linear interpolation.

LINEAR_MIPMAP_LINEAR = 9987
Linear mipmapping.

LINEAR_MIPMAP_NEAREST = 9985
Linear mipmapping.

LINES = 1
Render as lines.

LINE_LOOP = 2
Render as a line loop.

LINE_STRIP = 3
Render as a line strip.

MIRRORED_REPEAT = 33648
Mirror the texture.

NEAREST = 9728
Nearest neighbor interpolation.

NEAREST_MIPMAP_LINEAR = 9986
Nearest mipmapping.

NEAREST_MIPMAP_NEAREST = 9984
Nearest mipmapping.

POINTS = 0
Render as points.

REPEAT = 10497
Repeat the texture.

TRIANGLES = 4
Render as triangles.

TRIANGLE_FAN = 6
Render as a triangle fan.

TRIANGLE_STRIP = 5
Render as a triangle strip.

3.2 Cameras

3.2.1 Classes

Camera([znear, zfar, name])	Abstract base class for all cameras.
PerspectiveCamera(yfov[, znear, zfar, ...])	A perspective camera for perspective projection.
OrthographicCamera(xmag, ymag[, znear, ...])	An orthographic camera for orthographic projection.
IntrinsicsCamera(fx, fy, cx, cy[, znear, ...])	A perspective camera with custom intrinsics.

Camera

class pyrender.camera.**Camera**(*znear=0.05*, *zfar=100.0*, *name=None*)
Bases: object

Abstract base class for all cameras.

Note: Camera poses are specified in the OpenGL format, where the z axis points away from the view direction and the x and y axes point to the right and up in the image plane, respectively.

Parameters

- **znear** (*float*) – The floating-point distance to the near clipping plane.
- **zfar** (*float*) – The floating-point distance to the far clipping plane. `zfar` must be greater than `znear`.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

name	The user-defined name of this object.
zfar	The distance to the far clipping plane.
znear	The distance to the near clipping plane.

Methods Summary

get_projection_matrix([width, height])	Return the OpenGL projection matrix for this camera.

Attributes Documentation

name
The user-defined name of this object.

Type str

zfar
The distance to the far clipping plane.

Type float

znear
The distance to the near clipping plane.

Type float

Methods Documentation

get_projection_matrix(*width=None*, *height=None*)
Return the OpenGL projection matrix for this camera.

Parameters

- **width** (*int*) – Width of the current viewport, in pixels.
- **height** (*int*) – Height of the current viewport, in pixels.

PerspectiveCamera

class pyrender.camera.**PerspectiveCamera**(*yfov*, *znear=0.05*, *zfar=None*, *aspectRatio=None*, *name=None*)

Bases: *pyrender.camera.Camera*

A perspective camera for perspective projection.

Parameters

- **yfov** (*float*) – The floating-point vertical field of view in radians.
- **znear** (*float*) – The floating-point distance to the near clipping plane. If not specified, defaults to 0.05.
- **zfar** (*float, optional*) – The floating-point distance to the far clipping plane. `zfar` must be greater than `znear`. If None, the camera uses an infinite projection matrix.
- **aspectRatio** (*float, optional*) – The floating-point aspect ratio of the field of view. If not specified, the camera uses the viewport's aspect ratio.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

aspectRatio	The ratio of the width to the height of the field of view.
name	The user-defined name of this object.
yfov	The vertical field of view in radians.
zfar	The distance to the far clipping plane.
znear	The distance to the near clipping plane.

Methods Summary

get_projection_matrix([width, height])	Return the OpenGL projection matrix for this camera.

Attributes Documentation

aspectRatio
The ratio of the width to the height of the field of view.

Type float

name
The user-defined name of this object.

Type str

yfov
The vertical field of view in radians.

Type float

zfar
The distance to the far clipping plane.

Type float

znear
The distance to the near clipping plane.

Type float

Methods Documentation

get_projection_matrix(*width=None*, *height=None*)
Return the OpenGL projection matrix for this camera.

Parameters

- **width** (*int*) – Width of the current viewport, in pixels.
- **height** (*int*) – Height of the current viewport, in pixels.

OrthographicCamera

class pyrender.camera.**OrthographicCamera**(*xmag*, *ymag*, *znear=0.05*, *zfar=100.0*, *name=None*)
Bases: *pyrender.camera.Camera*

An orthographic camera for orthographic projection.

Parameters

- **xmag** (*float*) – The floating-point horizontal magnification of the view.
- **ymag** (*float*) – The floating-point vertical magnification of the view.
- **znear** (*float*) – The floating-point distance to the near clipping plane. If not specified, defaults to 0.05.
- **zfar** (*float*) – The floating-point distance to the far clipping plane. `zfar` must be greater than `znear`. If not specified, defaults to 100.0.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

name	The user-defined name of this object.
xmag	The horizontal magnification of the view.
ymag	The vertical magnification of the view.
zfar	The distance to the far clipping plane.
znear	The distance to the near clipping plane.

Methods Summary

get_projection_matrix([width, height])	Return the OpenGL projection matrix for this camera.

Attributes Documentation

name
The user-defined name of this object.

Type str

xmag
The horizontal magnification of the view.

Type float

ymag
The vertical magnification of the view.

Type float

zfar
The distance to the far clipping plane.

Type float

znear
The distance to the near clipping plane.

Type float

Methods Documentation

get_projection_matrix(*width=None*, *height=None*)
Return the OpenGL projection matrix for this camera.

Parameters

- **width** (*int*) – Width of the current viewport, in pixels. Unused in this function.
- **height** (*int*) – Height of the current viewport, in pixels. Unused in this function.

IntrinsicsCamera

class pyrender.camera.**IntrinsicsCamera**(*fx*, *fy*, *cx*, *cy*, *znear=0.05*, *zfar=100.0*, *name=None*)
Bases: *pyrender.camera.Camera*

A perspective camera with custom intrinsics.

Parameters

- **fx** (*float*) – X-axis focal length in pixels.
- **fy** (*float*) – Y-axis focal length in pixels.
- **cx** (*float*) – X-axis optical center in pixels.
- **cy** (*float*) – Y-axis optical center in pixels.
- **znear** (*float*) – The floating-point distance to the near clipping plane. If not specified, defaults to 0.05.
- **zfar** (*float*) – The floating-point distance to the far clipping plane. `zfar` must be greater than `znear`. If not specified, defaults to 100.0.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

cx	X-axis optical center in pixels.
cy	Y-axis optical center in pixels.
fx	X-axis focal length in meters.
fy	Y-axis focal length in meters.
name	The user-defined name of this object.
zfar	The distance to the far clipping plane.
znear	The distance to the near clipping plane.

Methods Summary

get_projection_matrix(width, height)	Return the OpenGL projection matrix for this camera.

Attributes Documentation

cx
X-axis optical center in pixels.

Type float

cy
Y-axis optical center in pixels.

Type float

fx
X-axis focal length in meters.

Type float

fy
Y-axis focal length in meters.

Type float

name
The user-defined name of this object.

Type str

zfar
The distance to the far clipping plane.

Type float

znear
The distance to the near clipping plane.

Type float

Methods Documentation

get_projection_matrix(*width*, *height*)
Return the OpenGL projection matrix for this camera.

Parameters

- **width** (*int*) – Width of the current viewport, in pixels.
- **height** (*int*) – Height of the current viewport, in pixels.

3.3 Lighting

3.3.1 Classes

Light([color, intensity, name])	Base class for all light objects.
DirectionalLight([color, intensity, name])	Directional lights are light sources that act as though they are infinitely far away and emit light in the direction of the local -z axis.
SpotLight([color, intensity, range, ...])	Spot lights emit light in a cone in the direction of the local -z axis.
PointLight([color, intensity, range, name])	Point lights emit light in all directions from their position in space; rotation and scale are ignored except for their effect on the inherited node position.

Light

class pyrender.light.**Light**(*color=None*, *intensity=None*, *name=None*)
Bases: object

Base class for all light objects.

Parameters

- **color** (*(3,) float*) – RGB value for the light's color in linear space.
- **intensity** (*float*) – Brightness of light. The units that this is defined in depend on the type of light. Point and spot lights use luminous intensity in candela (lm/sr), while directional lights use illuminance in lux (lm/m2).
- **name** (*str, optional*) – Name of the light.

Attributes Summary

color	The light's color.
intensity	The light's intensity in candela or lux.
name	The user-defined name of this object.
shadow_texture	A texture used to hold shadow maps for this light.

Attributes Documentation

color
The light's color.

Type (3,) float

intensity
The light's intensity in candela or lux.

Type float

name
The user-defined name of this object.

Type str

shadow_texture
A texture used to hold shadow maps for this light.

Type *Texture*

DirectionalLight

class pyrender.light.**DirectionalLight**(*color=None*, *intensity=None*, *name=None*)
Bases: *pyrender.light.Light*

Directional lights are light sources that act as though they are infinitely far away and emit light in the direction of the local -z axis. This light type inherits the orientation of the node that it belongs to; position and scale are ignored except for their effect on the inherited node orientation. Because it is at an infinite distance, the light is not attenuated. Its intensity is defined in lumens per metre squared, or lux (lm/m2).

Parameters

- **color** (*(3,) float, optional*) – RGB value for the light's color in linear space. Defaults to white (i.e. [1.0, 1.0, 1.0]).
- **intensity** (*float, optional*) – Brightness of light, in lux (lm/m^2). Defaults to 1.0
- **name** (*str, optional*) – Name of the light.

Attributes Summary

color	The light's color.
intensity	The light's intensity in candela or lux.
name	The user-defined name of this object.
shadow_texture	A texture used to hold shadow maps for this light.

Attributes Documentation

color
The light's color.

Type (3,) float

intensity
The light's intensity in candela or lux.

Type float

name
The user-defined name of this object.

Type str

shadow_texture
A texture used to hold shadow maps for this light.

Type *Texture*

SpotLight

class pyrender.light.**SpotLight**(*color=None*, *intensity=None*, *range=None*, *innerConeAngle=0.0*, *outerConeAngle=0.7853981633974483*, *name=None*)

Bases: *pyrender.light.Light*

Spot lights emit light in a cone in the direction of the local -z axis. The angle and falloff of the cone is defined using two numbers, the `innerConeAngle` and `outerConeAngle`. As with point lights, the brightness also attenuates in a physically correct manner as distance increases from the light's position (i.e. brightness goes like the inverse square of the distance). Spot light intensity refers to the brightness inside the `innerConeAngle` (and at the location of the light) and is defined in candela, which is lumens per square radian (lm/sr). A spot light's position and orientation are inherited from its node transform. Inherited scale does not affect cone shape, and is ignored except for its effect on position and orientation.

Parameters

- **color** (*(3,) float*) – RGB value for the light's color in linear space.
- **intensity** (*float*) – Brightness of light in candela (lm/sr).

- **range** (*float*) – Cutoff distance at which light's intensity may be considered to have reached zero. If None, the range is assumed to be infinite.
- **innerConeAngle** (*float*) – Angle, in radians, from centre of spotlight where falloff begins. Must be greater than or equal to `0` and less than `outerConeAngle`. Defaults to `0`.
- **outerConeAngle** (*float*) – Angle, in radians, from centre of spotlight where falloff ends. Must be greater than `innerConeAngle` and less than or equal to `PI / 2.0`. Defaults to `PI / 4.0`.
- **name** (*str, optional*) – Name of the light.

Attributes Summary

color	The light's color.
innerConeAngle	The inner cone angle in radians.
intensity	The light's intensity in candela or lux.
name	The user-defined name of this object.
outerConeAngle	The outer cone angle in radians.
range	The cutoff distance for the light.
shadow_texture	A texture used to hold shadow maps for this light.

Attributes Documentation

color
The light's color.

Type (3,) float

innerConeAngle
The inner cone angle in radians.

Type float

intensity
The light's intensity in candela or lux.

Type float

name
The user-defined name of this object.

Type str

outerConeAngle
The outer cone angle in radians.

Type float

range
The cutoff distance for the light.

Type float

shadow_texture
A texture used to hold shadow maps for this light.

Type *Texture*

PointLight

class pyrender.light.**PointLight**(*color=None*, *intensity=None*, *range=None*, *name=None*)
Bases: *pyrender.light.Light*

Point lights emit light in all directions from their position in space; rotation and scale are ignored except for their effect on the inherited node position. The brightness of the light attenuates in a physically correct manner as distance increases from the light's position (i.e. brightness goes like the inverse square of the distance). Point light intensity is defined in candela, which is lumens per square radian (lm/sr).

Parameters

- **color** (*(3,) float*) – RGB value for the light's color in linear space.
- **intensity** (*float*) – Brightness of light in candela (lm/sr).
- **range** (*float*) – Cutoff distance at which light's intensity may be considered to have reached zero. If None, the range is assumed to be infinite.
- **name** (*str, optional*) – Name of the light.

Attributes Summary

color	The light's color.
intensity	The light's intensity in candela or lux.
name	The user-defined name of this object.
range	The cutoff distance for the light.
shadow_texture	A texture used to hold shadow maps for this light.

Attributes Documentation

color
The light's color.

Type (3,) float

intensity
The light's intensity in candela or lux.

Type float

name
The user-defined name of this object.

Type str

range
The cutoff distance for the light.

Type float

shadow_texture
A texture used to hold shadow maps for this light.

Type *Texture*

3.4 Objects

3.4.1 Classes

IntrinsicsCamera(fx, fy, cx, cy[, znear, ...])	A perspective camera with custom intrinsics.
Sampler([name, magFilter, minFilter, wrapS, ...])	Texture sampler properties for filtering and wrapping modes.
Texture([name, sampler, source, ...])	A texture and its sampler.
Material([name, normalTexture, ...])	Base for standard glTF 2.0 materials.
MetallicRoughnessMaterial([name, ...])	A material based on the metallic-roughness material model from Physically-Based Rendering (PBR) methodology.
Primitive(positions[, normals, tangents, ...])	A primitive object which can be rendered.
Mesh(primitives[, name, weights, is_visible])	A set of primitives to be rendered.

IntrinsicsCamera

class pyrender.**IntrinsicsCamera**(*fx*, *fy*, *cx*, *cy*, *znear=0.05*, *zfar=100.0*, *name=None*)

Bases: *pyrender.camera.Camera*

A perspective camera with custom intrinsics.

Parameters

- **fx** (*float*) – X-axis focal length in pixels.
- **fy** (*float*) – Y-axis focal length in pixels.
- **cx** (*float*) – X-axis optical center in pixels.
- **cy** (*float*) – Y-axis optical center in pixels.
- **znear** (*float*) – The floating-point distance to the near clipping plane. If not specified, defaults to 0.05.
- **zfar** (*float*) – The floating-point distance to the far clipping plane. `zfar` must be greater than `znear`. If not specified, defaults to 100.0.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

cx	X-axis optical center in pixels.
cy	Y-axis optical center in pixels.
fx	X-axis focal length in meters.
fy	Y-axis focal length in meters.
name	The user-defined name of this object.
zfar	The distance to the far clipping plane.
znear	The distance to the near clipping plane.

Methods Summary

get_projection_matrix(width, height)	Return the OpenGL projection matrix for this camera.

Attributes Documentation

cx
X-axis optical center in pixels.

Type float

cy
Y-axis optical center in pixels.

Type float

fx
X-axis focal length in meters.

Type float

fy
Y-axis focal length in meters.

Type float

name
The user-defined name of this object.

Type str

zfar
The distance to the far clipping plane.

Type float

znear
The distance to the near clipping plane.

Type float

Methods Documentation

get_projection_matrix(*width*, *height*)
Return the OpenGL projection matrix for this camera.

Parameters

- **width** (*int*) – Width of the current viewport, in pixels.
- **height** (*int*) – Height of the current viewport, in pixels.

Sampler

class pyrender.**Sampler**(*name=None*, *magFilter=None*, *minFilter=None*, *wrapS=10497*, *wrapT=10497*)
Bases: object

Texture sampler properties for filtering and wrapping modes.

Parameters

- **name** (*str, optional*) – The user-defined name of this object.
- **magFilter** (*int, optional*) –

  Magnification filter. Valid values:

  - *GLTF.NEAREST*
  - *GLTF.LINEAR*
- **minFilter** (*int, optional*) –

  Minification filter. Valid values:

  - *GLTF.NEAREST*
  - *GLTF.LINEAR*
  - *GLTF.NEAREST_MIPMAP_NEAREST*
  - *GLTF.LINEAR_MIPMAP_NEAREST*
  - *GLTF.NEAREST_MIPMAP_LINEAR*
  - *GLTF.LINEAR_MIPMAP_LINEAR*
- **wrapS** (*int, optional*) –

  S (U) wrapping mode. Valid values:

  - *GLTF.CLAMP_TO_EDGE*
  - *GLTF.MIRRORED_REPEAT*
  - *GLTF.REPEAT*
- **wrapT** (*int, optional*) –

  T (V) wrapping mode. Valid values:

  - *GLTF.CLAMP_TO_EDGE*
  - *GLTF.MIRRORED_REPEAT*
  - *GLTF.REPEAT*

Attributes Summary

magFilter	Magnification filter type.
minFilter	Minification filter type.
name	The user-defined name of this object.
wrapS	S (U) wrapping mode.
wrapT	T (V) wrapping mode.

Attributes Documentation

magFilter
Magnification filter type.

Type int

minFilter
Minification filter type.

Type int

name
The user-defined name of this object.

Type str

wrapS
S (U) wrapping mode.

Type int

wrapT
T (V) wrapping mode.

Type int

Texture

class pyrender.**Texture**(*name=None*, *sampler=None*, *source=None*, *source_channels=None*, *width=None*, *height=None*, *tex_type=GL_TEXTURE_2D*, *data_format=GL_UNSIGNED_BYTE*)

Bases: object

A texture and its sampler.

Parameters

- **name** (*str, optional*) – The user-defined name of this object.
- **sampler** (*Sampler*) – The sampler used by this texture.
- **source** ((h,w,c) uint8 or (h,w,c) float or PIL.Image.Image) – The image used by this texture. If None, the texture is created empty and width and height must be specified.
- **source_channels** (*str*) – Either *D*, *R*, *RG*, *GB*, *RGB*, or *RGBA*. Indicates the channels to extract from *source*. Any missing channels will be filled with *1.0*.
- **width** (*int, optional*) – For empty textures, the width of the texture buffer.
- **height** (*int, optional*) – For empty textures, the height of the texture buffer.
- **tex_type** (*int*) – Either GL_TEXTURE_2D or GL_TEXTURE_CUBE.
- **data_format** (*int*) – For now, just GL_FLOAT.

Attributes Summary

data_format	The format of the texture data.
height	The height of the texture buffer.
name	The user-defined name of this object.
sampler	The sampler used by this texture.
source	The image used in this texture.
source_channels	The channels that were extracted from the original source.
tex_type	The type of the texture.
width	The width of the texture buffer.

Methods Summary

delete()	Remove this texture from the OpenGL context.
is_transparent([cutoff])	bool : If True, the texture is partially transparent.

Attributes Documentation

data_format
The format of the texture data.

Type int

height
The height of the texture buffer.

Type int

name
The user-defined name of this object.

Type str

sampler
The sampler used by this texture.

Type *Sampler*

source
The image used in this texture.

Type (h,w,c) uint8 or float or `PIL.Image.Image`

source_channels
The channels that were extracted from the original source.

Type str

tex_type
The type of the texture.

Type int

width
The width of the texture buffer.

Type int

Methods Documentation

delete()
Remove this texture from the OpenGL context.

is_transparent(*cutoff=1.0*)
bool : If True, the texture is partially transparent.

Material

class pyrender.**Material**(*name=None*, *normalTexture=None*, *occlusionTexture=None*, *emissiveTexture=None*, *emissiveFactor=None*, *alphaMode=None*, *alphaCutoff=None*, *doubleSided=False*, *smooth=True*, *wireframe=False*)

Bases: object

Base for standard glTF 2.0 materials.

Parameters

- **name** (*str, optional*) – The user-defined name of this object.
- **normalTexture** ((n,n,3) float or *Texture*, optional) – A tangent space normal map. The texture contains RGB components in linear space. Each texel represents the XYZ components of a normal vector in tangent space. Red [0 to 255] maps to X [-1 to 1]. Green [0 to 255] maps to Y [-1 to 1]. Blue [128 to 255] maps to Z [1/255 to 1]. The normal vectors use OpenGL conventions where +X is right and +Y is up. +Z points toward the viewer.
- **occlusionTexture** ((n,n,1) float or *Texture*, optional) – The occlusion map texture. The occlusion values are sampled from the R channel. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. These values are linear. If other channels are present (GBA), they are ignored for occlusion calculations.
- **emissiveTexture** ((n,n,3) float or *Texture*, optional) – The emissive map controls the color and intensity of the light being emitted by the material. This texture contains RGB components in sRGB color space. If a fourth component (A) is present, it is ignored.
- **emissiveFactor** (*(3,) float, optional*) – The RGB components of the emissive color of the material. These values are linear. If an emissiveTexture is specified, this value is multiplied with the texel values.
- **alphaMode** (*str, optional*) – The material's alpha rendering mode enumeration specifying the interpretation of the alpha value of the main factor and texture. Allowed Values:
  - *"OPAQUE"* The alpha value is ignored and the rendered output is fully opaque.
  - *"MASK"* The rendered output is either fully opaque or fully transparent depending on the alpha value and the specified alpha cutoff value.
  - *"BLEND"* The alpha value is used to composite the source and destination areas. The rendered output is combined with the background using the normal painting operation (i.e. the Porter and Duff over operator).
- **alphaCutoff** (*float, optional*) – Specifies the cutoff threshold when in MASK mode. If the alpha value is greater than or equal to this value then it is rendered as fully opaque, otherwise, it is rendered as fully transparent. A value greater than 1.0 will render the entire material as fully transparent. This value is ignored for other modes.
- **doubleSided** (*bool, optional*) – Specifies whether the material is double sided. When this value is false, back-face culling is enabled. When this value is true, back-face culling is disabled and double sided lighting is enabled.
- **smooth** (*bool, optional*) – If True, the material is rendered smoothly by using only one normal per vertex and face indexing.
- **wireframe** (*bool, optional*) – If True, the material is rendered in wireframe mode.

Attributes Summary

alphaCutoff	The cutoff threshold in MASK mode.
alphaMode	The mode for blending.
doubleSided	Whether the material is double-sided.
emissiveFactor	Base multiplier for emission colors.
emissiveTexture	The emission map.
is_transparent	If True, the object is partially transparent.
name	The user-defined name of this object.
normalTexture	The tangent-space normal map.
occlusionTexture	The ambient occlusion map.
smooth	Whether to render the mesh smoothly by interpolating vertex normals.
tex_flags	Texture availability flags.
textures	The textures associated with this material.
wireframe	Whether to render the mesh in wireframe mode.

Attributes Documentation

alphaCutoff
The cutoff threshold in MASK mode.

Type float

alphaMode
The mode for blending.

Type str

doubleSided
Whether the material is double-sided.

Type bool

emissiveFactor
Base multiplier for emission colors.

Type (3,) float

emissiveTexture
The emission map.

Type (n,n,3) float or *Texture*

is_transparent
If True, the object is partially transparent.

Type bool

name
The user-defined name of this object.

Type str

normalTexture
The tangent-space normal map.

Type (n,n,3) float or *Texture*

occlusionTexture
The ambient occlusion map.

Type (n,n,1) float or *Texture*

smooth
Whether to render the mesh smoothly by interpolating vertex normals.

Type bool

tex_flags
Texture availability flags.

Type int

textures
The textures associated with this material.

Type list of *Texture*

wireframe
Whether to render the mesh in wireframe mode.

Type bool

MetallicRoughnessMaterial

class pyrender.**MetallicRoughnessMaterial**(*name=None*, *normalTexture=None*, *occlusionTexture=None*, *emissiveTexture=None*, *emissiveFactor=None*, *alphaMode=None*, *alphaCutoff=None*, *doubleSided=False*, *smooth=True*, *wireframe=False*, *baseColorFactor=None*, *baseColorTexture=None*, *metallicFactor=1.0*, *roughnessFactor=1.0*, *metallicRoughnessTexture=None*)

Bases: *pyrender.Material*

A material based on the metallic-roughness material model from Physically-Based Rendering (PBR) methodology.

Parameters

- **name** (*str, optional*) – The user-defined name of this object.
- **normalTexture** ((n,n,3) float or *Texture*, optional) – A tangent space normal map. The texture contains RGB components in linear space. Each texel represents the XYZ components of a normal vector in tangent space. Red [0 to 255] maps to X [-1 to 1]. Green [0 to 255] maps to Y [-1 to 1]. Blue [128 to 255] maps to Z [1/255 to 1]. The normal vectors use OpenGL conventions where +X is right and +Y is up. +Z points toward the viewer.
- **occlusionTexture** ((n,n,1) float or *Texture*, optional) – The occlusion map texture. The occlusion values are sampled from the R channel. Higher values indicate areas that should receive full indirect lighting and lower values indicate no indirect lighting. These values are linear. If other channels are present (GBA), they are ignored for occlusion calculations.
- **emissiveTexture** ((n,n,3) float or *Texture*, optional) – The emissive map controls the color and intensity of the light being emitted by the material. This texture contains RGB components in sRGB color space. If a fourth component (A) is present, it is ignored.

- **emissiveFactor** (*(3,) float, optional*) – The RGB components of the emissive color of the material. These values are linear. If an emissiveTexture is specified, this value is multiplied with the texel values.
- **alphaMode** (*str, optional*) – The material's alpha rendering mode enumeration specifying the interpretation of the alpha value of the main factor and texture. Allowed Values:
  - *"OPAQUE"* The alpha value is ignored and the rendered output is fully opaque.
  - *"MASK"* The rendered output is either fully opaque or fully transparent depending on the alpha value and the specified alpha cutoff value.
  - *"BLEND"* The alpha value is used to composite the source and destination areas. The rendered output is combined with the background using the normal painting operation (i.e. the Porter and Duff over operator).
- **alphaCutoff** (*float, optional*) – Specifies the cutoff threshold when in MASK mode. If the alpha value is greater than or equal to this value then it is rendered as fully opaque, otherwise, it is rendered as fully transparent. A value greater than 1.0 will render the entire material as fully transparent. This value is ignored for other modes.
- **doubleSided** (*bool, optional*) – Specifies whether the material is double sided. When this value is false, back-face culling is enabled. When this value is true, back-face culling is disabled and double sided lighting is enabled.
- **smooth** (*bool, optional*) – If True, the material is rendered smoothly by using only one normal per vertex and face indexing.
- **wireframe** (*bool, optional*) – If True, the material is rendered in wireframe mode.
- **baseColorFactor** (*(4,) float, optional*) – The RGBA components of the base color of the material. The fourth component (A) is the alpha coverage of the material. The alphaMode property specifies how alpha is interpreted. These values are linear. If a baseColorTexture is specified, this value is multiplied with the texel values.
- **baseColorTexture** ((n,n,4) float or *Texture*, optional) – The base color texture. This texture contains RGB(A) components in sRGB color space. The first three components (RGB) specify the base color of the material. If the fourth component (A) is present, it represents the alpha coverage of the material. Otherwise, an alpha of 1.0 is assumed. The alphaMode property specifies how alpha is interpreted. The stored texels must not be premultiplied.
- **metallicFactor** (*float*) – The metalness of the material. A value of 1.0 means the material is a metal. A value of 0.0 means the material is a dielectric. Values in between are for blending between metals and dielectrics such as dirty metallic surfaces. This value is linear. If a metallicRoughnessTexture is specified, this value is multiplied with the metallic texel values.
- **roughnessFactor** (*float*) – The roughness of the material. A value of 1.0 means the material is completely rough. A value of 0.0 means the material is completely smooth. This value is linear. If a metallicRoughnessTexture is specified, this value is multiplied with the roughness texel values.
- **metallicRoughnessTexture** ((n,n,2) float or *Texture*, optional) – The metallic-roughness texture. The metalness values are sampled from the B channel. The roughness values are sampled from the G channel. These values are linear. If other channels are present (R or A), they are ignored for metallic-roughness calculations.

Attributes Summary

alphaCutoff	The cutoff threshold in MASK mode.
alphaMode	The mode for blending.
baseColorFactor	The RGBA base color multiplier.
baseColorTexture	The diffuse texture.
doubleSided	Whether the material is double-sided.
emissiveFactor	Base multiplier for emission colors.
emissiveTexture	The emission map.
is_transparent	If True, the object is partially transparent.
metallicFactor	The metalness of the material.
metallicRoughnessTexture	The metallic-roughness texture.
name	The user-defined name of this object.
normalTexture	The tangent-space normal map.
occlusionTexture	The ambient occlusion map.
roughnessFactor	The roughness of the material.
smooth	Whether to render the mesh smoothly by interpolating vertex normals.
tex_flags	Texture availability flags.
textures	The textures associated with this material.
wireframe	Whether to render the mesh in wireframe mode.

Attributes Documentation

alphaCutoff
The cutoff threshold in MASK mode.

Type float

alphaMode
The mode for blending.

Type str

baseColorFactor
The RGBA base color multiplier.

Type (4,) float or *Texture*

baseColorTexture
The diffuse texture.

Type (n,n,4) float or *Texture*

doubleSided
Whether the material is double-sided.

Type bool

emissiveFactor
Base multiplier for emission colors.

Type (3,) float

emissiveTexture
The emission map.

Type (n,n,3) float or *Texture*

is_transparent
If True, the object is partially transparent.

Type bool

metallicFactor
The metalness of the material.

Type float

metallicRoughnessTexture
The metallic-roughness texture.

Type (n,n,2) float or *Texture*

name
The user-defined name of this object.

Type str

normalTexture
The tangent-space normal map.

Type (n,n,3) float or *Texture*

occlusionTexture
The ambient occlusion map.

Type (n,n,1) float or *Texture*

roughnessFactor
The roughness of the material.

Type float

smooth
Whether to render the mesh smoothly by interpolating vertex normals.

Type bool

tex_flags
Texture availability flags.

Type int

textures
The textures associated with this material.

Type list of *Texture*

wireframe
Whether to render the mesh in wireframe mode.

Type bool

Primitive

class pyrender.**Primitive**(*positions*, *normals=None*, *tangents=None*, *texcoord_0=None*, *texcoord_1=None*, *color_0=None*, *joints_0=None*, *weights_0=None*, *indices=None*, *material=None*, *mode=None*, *targets=None*, *poses=None*)

Bases: object

A primitive object which can be rendered.

Parameters

- **positions** (*(n, 3) float*) – XYZ vertex positions.
- **normals** (*(n, 3) float*) – Normalized XYZ vertex normals.
- **tangents** (*(n, 4) float*) – XYZW vertex tangents where the w component is a sign value (either +1 or -1) indicating the handedness of the tangent basis.
- **texcoord_0** (*(n, 2) float*) – The first set of UV texture coordinates.
- **texcoord_1** (*(n, 2) float*) – The second set of UV texture coordinates.
- **color_0** (*(n, 4) float*) – RGBA vertex colors.
- **joints_0** (*(n, 4) float*) – Joint information.
- **weights_0** (*(n, 4) float*) – Weight information for morphing.
- **indices** (*(m, 3) int*) – Face indices for triangle meshes or fans.
- **material** (*Material*) – The material to apply to this primitive when rendering.
- **mode** (*int*) – The type of primitives to render, one of the following:
  - 0: POINTS
  - 1: LINES
  - 2: LINE_LOOP
  - 3: LINE_STRIP
  - 4: TRIANGLES
  - 5: TRIANGLES_STRIP
  - 6: TRIANGLES_FAN
- **targets** (*(k,) int*) – Morph target indices.
- **poses** (*(x,4,4), float*) – Array of 4x4 transformation matrices for instancing this object.

Attributes Summary

bounds	
buf_flags	The flags for the render buffer.
centroid	The centroid of the primitive's AABB.
color_0	RGBA vertex colors.
extents	The lengths of the axes of the primitive's AABB.
indices	Face indices for triangle meshes or fans.
is_transparent	If True, the mesh is partially-transparent.
joints_0	Joint information.
material	The material for this primitive.
mode	The type of primitive to render.
normals	Normalized XYZ vertex normals.
poses	Homogenous transforms for instancing this primitive.
positions	XYZ vertex positions.
scale	The length of the diagonal of the primitive's AABB.
tangents	XYZW vertex tangents.

Continued on next page

Table 27 – continued from previous page

targets	Morph target indices.
texcoord_0	The first set of UV texture coordinates.
texcoord_1	The second set of UV texture coordinates.
weights_0	Weight information for morphing.

Methods Summary

delete()	

Attributes Documentation

bounds

buf_flags
The flags for the render buffer.

Type int

centroid
The centroid of the primitive's AABB.

Type (3,) float

color_0
RGBA vertex colors.

Type (n,4) float

extents
The lengths of the axes of the primitive's AABB.

Type (3,) float

indices
Face indices for triangle meshes or fans.

Type (m,3) int

is_transparent
If True, the mesh is partially-transparent.

Type bool

joints_0
Joint information.

Type (n,4) float

material
The material for this primitive.

Type *Material*

mode
The type of primitive to render.

Type int

normals
Normalized XYZ vertex normals.

Type (n,3) float

poses
Homogenous transforms for instancing this primitive.

Type (x,4,4) float

positions
XYZ vertex positions.

Type (n,3) float

scale
The length of the diagonal of the primitive's AABB.

Type (3,) float

tangents
XYZW vertex tangents.

Type (n,4) float

targets
Morph target indices.

Type (k,) int

texcoord_0
The first set of UV texture coordinates.

Type (n,2) float

texcoord_1
The second set of UV texture coordinates.

Type (n,2) float

weights_0
Weight information for morphing.

Type (n,4) float

Methods Documentation

delete()

Mesh

class pyrender.**Mesh**(*primitives*, *name=None*, *weights=None*, *is_visible=True*)
Bases: object

A set of primitives to be rendered.

Parameters

- **name** (*str*) – The user-defined name of this object.
- **primitives** (list of *Primitive*) – The primitives associated with this mesh.
- **weights** (*(k,) float*) – Array of weights to be applied to the Morph Targets.
- **is_visible** (*bool*) – If False, the mesh will not be rendered.

Attributes Summary

bounds	The axis-aligned bounds of the mesh.
centroid	The centroid of the mesh's axis-aligned bounding box (AABB).
extents	The lengths of the axes of the mesh's AABB.
is_transparent	If True, the mesh is partially-transparent.
is_visible	Whether the mesh is visible.
name	The user-defined name of this object.
primitives	The primitives associated with this mesh.
scale	The length of the diagonal of the mesh's AABB.
weights	Weights to be applied to morph targets.

Methods Summary

from_points(points[, colors, normals, ...])	Create a Mesh from a set of points.
from_trimesh(mesh[, material, is_visible, ...])	Create a Mesh from a `Trimesh`.

Attributes Documentation

bounds
The axis-aligned bounds of the mesh.

Type (2,3) float

centroid
The centroid of the mesh's axis-aligned bounding box (AABB).

Type (3,) float

extents
The lengths of the axes of the mesh's AABB.

Type (3,) float

is_transparent
If True, the mesh is partially-transparent.

Type bool

is_visible
Whether the mesh is visible.

Type bool

name
The user-defined name of this object.

Type str

primitives
The primitives associated with this mesh.

Type list of *Primitive*

scale
The length of the diagonal of the mesh's AABB.

Type (3,) float

weights
Weights to be applied to morph targets.

Type (k,) float

Methods Documentation

static from_points(*points*, *colors=None*, *normals=None*, *is_visible=True*, *poses=None*)
Create a Mesh from a set of points.

Parameters

- **points** (*(n,3) float*) – The point positions.
- **colors** (*(n,3) or (n,4) float, optional*) – RGB or RGBA colors for each point.
- **normals** (*(n,3) float, optionals*) – The normal vectors for each point.
- **is_visible** (*bool*) – If False, the points will not be rendered.
- **poses** (*(x,4,4)*) – Array of 4x4 transformation matrices for instancing this object.

Returns **mesh** – The created mesh.

Return type *Mesh*

static from_trimesh(*mesh*, *material=None*, *is_visible=True*, *poses=None*, *wireframe=False*, *smooth=True*)
Create a Mesh from a `Trimesh`.

Parameters

- **mesh** (`Trimesh` or list of them) – A triangular mesh or a list of meshes.
- **material** (*Material*) – The material of the object. Overrides any mesh material. If not specified and the mesh has no material, a default material will be used.
- **is_visible** (*bool*) – If False, the mesh will not be rendered.
- **poses** (*(n,4,4) float*) – Array of 4x4 transformation matrices for instancing this object.
- **wireframe** (*bool*) – If *True*, the mesh will be rendered as a wireframe object
- **smooth** (*bool*) – If *True*, the mesh will be rendered with interpolated vertex normals. Otherwise, the mesh edges will stay sharp.

Returns **mesh** – The created mesh.

Return type *Mesh*

3.5 Scenes

3.5.1 Classes

IntrinsicsCamera(fx, fy, cx, cy[, znear, ...])	A perspective camera with custom intrinsics.
Node([name, camera, children, skin, matrix, ...])	A node in the node hierarchy.
Scene([nodes, bg_color, ambient_light, name])	A hierarchical scene graph.

Node

class pyrender.**Node**(*name=None*, *camera=None*, *children=None*, *skin=None*, *matrix=None*, *mesh=None*, *rotation=None*, *scale=None*, *translation=None*, *weights=None*, *light=None*)

Bases: object

A node in the node hierarchy.

Parameters

- **name** (*str, optional*) – The user-defined name of this object.
- **camera** (Camera, optional) – The camera in this node.
- **children** (list of *Node*) – The children of this node.
- **skin** (*int, optional*) – The index of the skin referenced by this node.
- **matrix** (*(4,4) float, optional*) – A floating-point 4x4 transformation matrix.
- **mesh** (*Mesh*, optional) – The mesh in this node.
- **rotation** (*(4,) float, optional*) – The node's unit quaternion in the order (x, y, z, w), where w is the scalar.
- **scale** (*(3,) float, optional*) – The node's non-uniform scale, given as the scaling factors along the x, y, and z axes.
- **translation** (*(3,) float, optional*) – The node's translation along the x, y, and z axes.
- **weights** (*(n,) float*) – The weights of the instantiated Morph Target. Number of elements must match number of Morph Targets of used mesh.
- **light** (Light, optional) – The light in this node.

Attributes Summary

camera	The camera in this node.
children	The children of this node.
light	The light in this node.
matrix	The homogenous transform matrix for this node.
mesh	The mesh in this node.
name	The user-defined name of this object.
rotation	The xyzw quaternion for this node.
scale	The scale for this node.
skin	The skin index for this node.
translation	The translation for this node.

Attributes Documentation

camera
The camera in this node.

Type Camera

children
The children of this node.

Type list of *Node*

light
The light in this node.

Type Light

matrix
The homogenous transform matrix for this node.

Note that this matrix's elements are not settable, it's just a copy of the internal matrix. You can set the whole matrix, but not an individual element.

Type (4,4) float

mesh
The mesh in this node.

Type *Mesh*

name
The user-defined name of this object.

Type str

rotation
The xyzw quaternion for this node.

Type (4,) float

scale
The scale for this node.

Type (3,) float

skin
The skin index for this node.

Type int

translation
The translation for this node.

Type (3,) float

Scene

class pyrender.**Scene**(*nodes=None*, *bg_color=None*, *ambient_light=None*, *name=None*)
Bases: object

A hierarchical scene graph.

Parameters

- **nodes** (list of *Node*) – The set of all nodes in the scene.
- **bg_color** (*(4,) float, optional*) – Background color of scene.
- **ambient_light** (*(3,) float, optional*) – Color of ambient light. Defaults to no ambient light.
- **name** (*str, optional*) – The user-defined name of this object.

Attributes Summary

ambient_light	The ambient light in the scene.
bg_color	The scene background color.
bounds	The axis-aligned bounds of the scene.
camera_nodes	The nodes containing cameras in the scene.
cameras	The cameras in the scene.
centroid	The centroid of the scene's axis-aligned bounding box (AABB).
directional_light_nodes	The nodes containing directional lights.
directional_lights	The directional lights in the scene.
extents	The lengths of the axes of the scene's AABB.
light_nodes	The nodes containing lights.
lights	The lights in the scene.
main_camera_node	The node containing the main camera in the scene.
mesh_nodes	The nodes containing meshes.
meshes	The meshes in the scene.
name	The user-defined name of this object.
nodes	Set of nodes in the scene.
point_light_nodes	The nodes containing point lights.
point_lights	The point lights in the scene.
scale	The length of the diagonal of the scene's AABB.
spot_light_nodes	The nodes containing spot lights.
spot_lights	The spot lights in the scene.

Methods Summary

add(obj[, name, pose, parent_node, parent_name])	Add an object (mesh, light, or camera) to the scene.
add_node(node[, parent_node])	Add a Node to the scene.
clear()	Clear out all nodes to form an empty scene.
from_trimesh_scene(trimesh_scene[, ...])	Create a *Scene* from a `trimesh.scene.scene.Scene`.
get_nodes([node, name, obj, obj_name])	Search for existing nodes.
get_pose(node)	Get the world-frame pose of a node in the scene.
has_node(node)	Check if a node is already in the scene.
remove_node(node)	Remove a node and all its children from the scene.
set_pose(node, pose)	Set the local-frame pose of a node in the scene.

Attributes Documentation

ambient_light
The ambient light in the scene.

Type (3,) float

bg_color
The scene background color.

Type (3,) float

bounds
The axis-aligned bounds of the scene.

Type (2,3) float

camera_nodes
The nodes containing cameras in the scene.

Type set of *Node*

cameras
The cameras in the scene.

Type set of Camera

centroid
The centroid of the scene's axis-aligned bounding box (AABB).

Type (3,) float

directional_light_nodes
The nodes containing directional lights.

Type set of *Node*

directional_lights
The directional lights in the scene.

Type set of DirectionalLight

extents
The lengths of the axes of the scene's AABB.

Type (3,) float

light_nodes
The nodes containing lights.

Type set of *Node*

lights
The lights in the scene.

Type set of Light

main_camera_node
The node containing the main camera in the scene.

Type set of *Node*

mesh_nodes
The nodes containing meshes.

Type set of *Node*

meshes
The meshes in the scene.

Type set of *Mesh*

name
The user-defined name of this object.

Type str

nodes
Set of nodes in the scene.

Type set of *Node*

point_light_nodes
The nodes containing point lights.

Type set of *Node*

point_lights
The point lights in the scene.

Type set of PointLight

scale
The length of the diagonal of the scene's AABB.

Type (3,) float

spot_light_nodes
The nodes containing spot lights.

Type set of *Node*

spot_lights
The spot lights in the scene.

Type set of SpotLight

Methods Documentation

add(*obj*, *name=None*, *pose=None*, *parent_node=None*, *parent_name=None*)
Add an object (mesh, light, or camera) to the scene.

Parameters

- **obj** (*Mesh*, Light, or Camera) – The object to add to the scene.
- **name** (*str*) – A name for the new node to be created.
- **pose** (*(4, 4) float*) – The local pose of this node relative to its parent node.
- **parent_node** (*Node*) – The parent of this Node. If None, the new node is a root node.
- **parent_name** (*str*) – The name of the parent node, can be specified instead of *parent_node*.

Returns **node** – The newly-created and inserted node.

Return type *Node*

add_node(*node*, *parent_node=None*)
Add a Node to the scene.

Parameters

- **node** (*Node*) – The node to be added.
- **parent_node** (*Node*) – The parent of this Node. If None, the new node is a root node.

clear()
Clear out all nodes to form an empty scene.

static from_trimesh_scene(*trimesh_scene*, *bg_color=None*, *ambient_light=None*)
Create a *Scene* from a trimesh.scene.scene.Scene.

Parameters

- **trimesh_scene** (trimesh.scene.scene.Scene) – Scene with :class:~'trimesh.base.Trimesh' objects.

- **bg_color** (*(4,) float*) – Background color for the created scene.
- **ambient_light** (*(3,) float or None*) – Ambient light in the scene.

Returns **scene_pr** – A scene containing the same geometry as the trimesh scene.

Return type *Scene*

get_nodes(*node=None*, *name=None*, *obj=None*, *obj_name=None*)
Search for existing nodes. Only nodes matching all specified parameters is returned, or None if no such node exists.

Parameters

- **node** (*Node*, optional) – If present, returns this node if it is in the scene.
- **name** (*str*) – A name for the Node.
- **obj** (*Mesh*, Light, or Camera) – An object that is attached to the node.
- **obj_name** (*str*) – The name of an object that is attached to the node.

Returns **nodes** – The nodes that match all query terms.

Return type set of *Node*

get_pose(*node*)
Get the world-frame pose of a node in the scene.

Parameters **node** (*Node*) – The node to find the pose of.

Returns **pose** – The transform matrix for this node.

Return type (4,4) float

has_node(*node*)
Check if a node is already in the scene.

Parameters **node** (*Node*) – The node to be checked.

Returns **has_node** – True if the node is already in the scene and false otherwise.

Return type bool

remove_node(*node*)
Remove a node and all its children from the scene.

Parameters **node** (*Node*) – The node to be removed.

set_pose(*node*, *pose*)
Set the local-frame pose of a node in the scene.

Parameters

- **node** (*Node*) – The node to set the pose of.
- **pose** (*(4, 4) float*) – The pose to set the node to.

3.6 On-Screen Viewer

3.6.1 Classes

Viewer(scene[, viewport_size, render_flags, ...])	An interactive viewer for 3D scenes.

Viewer

class pyrender.viewer.**Viewer**(*scene*, *viewport_size=None*, *render_flags=None*, *viewer_flags=None*, *registered_keys=None*, *run_in_thread=False*, ***kwargs*)

Bases: pyglet.window.Window

An interactive viewer for 3D scenes.

The viewer's camera is separate from the scene's, but will take on the parameters of the scene's main view camera and start in the same pose. If the scene does not have a camera, a suitable default will be provided.

Parameters

- **scene** (Scene) – The scene to visualize.
- **viewport_size** (*(2,) int*) – The width and height of the initial viewing window.
- **render_flags** (*dict*) – A set of flags for rendering the scene. Described in the note below.
- **viewer_flags** (*dict*) – A set of flags for controlling the viewer's behavior. Described in the note below.
- **registered_keys** (*dict*) – A map from ASCII key characters to tuples containing:
  - A function to be called whenever the key is pressed, whose first argument will be the viewer itself.
  - (Optionally) A list of additional positional arguments to be passed to the function.
  - (Optionally) A dict of keyword arguments to be passed to the function.
- **kwargs** (*dict*) – Any keyword arguments left over will be interpreted as belonging to either the *Viewer.render_flags* or *Viewer.viewer_flags* dictionaries. Those flag sets will be updated appropriately.

Note: The basic commands for moving about the scene are given as follows:

- **Rotating about the scene**: Hold the left mouse button and drag the cursor.
- **Rotating about the view axis**: Hold CTRL and the left mouse button and drag the cursor.
- **Panning**:
  - Hold SHIFT, then hold the left mouse button and drag the cursor, or
  - Hold the middle mouse button and drag the cursor.
- **Zooming**:
  - Scroll the mouse wheel, or
  - Hold the right mouse button and drag the cursor.

Other keyboard commands are as follows:

- a: Toggles rotational animation mode.
- c: Toggles backface culling.
- f: Toggles fullscreen mode.

- h: Toggles shadow rendering.
- i: Toggles axis display mode (no axes, world axis, mesh axes, all axes).
- l: Toggles lighting mode (scene lighting, Raymond lighting, or direct lighting).
- m: Toggles face normal visualization.
- n: Toggles vertex normal visualization.
- o: Toggles orthographic mode.
- q: Quits the viewer.
- r: Starts recording a GIF, and pressing again stops recording and opens a file dialog.
- s: Opens a file dialog to save the current view as an image.
- w: Toggles wireframe mode (scene default, flip wireframes, all wireframe, or all solid).
- z: Resets the camera to the initial view.

Note: The valid keys for `render_flags` are as follows:

- `flip_wireframe`: *bool*, If *True*, all objects will have their wireframe modes flipped from what their material indicates. Defaults to *False*.
- `all_wireframe`: *bool*, If *True*, all objects will be rendered in wireframe mode. Defaults to *False*.
- `all_solid`: *bool*, If *True*, all objects will be rendered in solid mode. Defaults to *False*.
- `shadows`: *bool*, If *True*, shadows will be rendered. Defaults to *False*.
- `vertex_normals`: *bool*, If *True*, vertex normals will be rendered as blue lines. Defaults to *False*.
- `face_normals`: *bool*, If *True*, face normals will be rendered as blue lines. Defaults to *False*.
- `cull_faces`: *bool*, If *True*, backfaces will be culled. Defaults to *True*.
- `point_size` : float, The point size in pixels. Defaults to 1px.

Note: The valid keys for `viewer_flags` are as follows:

- `rotate`: *bool*, If *True*, the scene's camera will rotate about an axis. Defaults to *False*.
- `rotate_rate`: *float*, The rate of rotation in radians per second. Defaults to *PI / 3.0*.
- `rotate_axis`: *(3,) float*, The axis in world coordinates to rotate about. Defaults to `[0,0,1]`.
- `view_center`: *(3,) float*, The position to rotate the scene about. Defaults to the scene's centroid.
- `use_raymond_lighting`: *bool*, If *True*, an additional set of three directional lights that move with the camera will be added to the scene. Defaults to *False*.
- `use_direct_lighting`: *bool*, If *True*, an additional directional light that moves with the camera and points out of it will be added to the scene. Defaults to *False*.
- `lighting_intensity`: *float*, The overall intensity of the viewer's additional lights (when they're in use). Defaults to 3.0.
- `use_perspective_cam`: *bool*, If *True*, a perspective camera will be used. Otherwise, an orthographic camera is used. Defaults to *True*.
- `save_directory`: *str*, A directory to open the file dialogs in. Defaults to *None*.

- `window_title`: *str*, A title for the viewer's application window. Defaults to *"Scene Viewer"*.
- `refresh_rate`: *float*, A refresh rate for rendering, in Hertz. Defaults to *30.0*.
- `fullscreen`: *bool*, Whether to make viewer fullscreen. Defaults to *False*.
- `show_world_axis`: *bool*, Whether to show the world axis. Defaults to *False*.
- `show_mesh_axes`: *bool*, Whether to show the individual mesh axes. Defaults to *False*.
- `caption`: *list of dict*, Text caption(s) to display on the viewer. Defaults to *None*.

Note: Animation can be accomplished by running the viewer with `run_in_thread` enabled. Then, just run a loop in your main thread, updating the scene as needed. Before updating the scene, be sure to acquire the *Viewer.render_lock*, and release it when your update is done.

Attributes Summary

is_active	*True* if the viewer is active, or *False* if it has been closed.
registered_keys	Map from ASCII key character to a handler function.
render_flags	Flags for controlling the renderer's behavior.
render_lock	If acquired, prevents the viewer from rendering until released.
run_in_thread	Whether the viewer was run in a separate thread.
scene	The scene being visualized.
viewer_flags	Flags for controlling the viewer's behavior.
viewport_size	The width and height of the viewing window.

Methods Summary

close_external()	Close the viewer from another thread.
on_close()	Exit the event loop when the window is closed.
on_draw()	Redraw the scene into the viewing window.
on_key_press(symbol, modifiers)	Record a key press.
on_mouse_drag(x, y, dx, dy, buttons, modifiers)	Record a mouse drag.
on_mouse_press(x, y, buttons, modifiers)	Record an initial mouse press.
on_mouse_release(x, y, button, modifiers)	Record a mouse release.
on_mouse_scroll(x, y, dx, dy)	Record a mouse scroll.
on_resize(width, height)	Resize the camera and trackball when the window is resized.
save_gif([filename])	Save the stored GIF frames to a file.

Attributes Documentation

is_active

True if the viewer is active, or *False* if it has been closed.

Type bool

registered_keys

Map from ASCII key character to a handler function.

This is a map from ASCII key characters to tuples containing:

- A function to be called whenever the key is pressed, whose first argument will be the viewer itself.
- (Optionally) A list of additional positional arguments to be passed to the function.
- (Optionally) A dict of keyword arguments to be passed to the function.

Type dict

render_flags
Flags for controlling the renderer's behavior.

- `flip_wireframe`: *bool*, If *True*, all objects will have their wireframe modes flipped from what their material indicates. Defaults to *False*.
- `all_wireframe`: *bool*, If *True*, all objects will be rendered in wireframe mode. Defaults to *False*.
- `all_solid`: *bool*, If *True*, all objects will be rendered in solid mode. Defaults to *False*.
- `shadows`: *bool*, If *True*, shadows will be rendered. Defaults to *False*.
- `vertex_normals`: *bool*, If *True*, vertex normals will be rendered as blue lines. Defaults to *False*.
- `face_normals`: *bool*, If *True*, face normals will be rendered as blue lines. Defaults to *False*.
- `cull_faces`: *bool*, If *True*, backfaces will be culled. Defaults to *True*.
- `point_size` : float, The point size in pixels. Defaults to 1px.

Type dict

render_lock
If acquired, prevents the viewer from rendering until released.

Run `Viewer.render_lock.acquire()` before making updates to the scene in a different thread, and run `Viewer.render_lock.release()` once you're done to let the viewer continue.

Type `threading.RLock`

run_in_thread
Whether the viewer was run in a separate thread.

Type bool

scene
The scene being visualized.

Type *Scene*

viewer_flags
Flags for controlling the viewer's behavior.

The valid keys for `viewer_flags` are as follows:

- `rotate`: *bool*, If *True*, the scene's camera will rotate about an axis. Defaults to *False*.
- `rotate_rate`: *float*, The rate of rotation in radians per second. Defaults to *PI / 3.0*.
- `rotate_axis`: *(3,) float*, The axis in world coordinates to rotate about. Defaults to `[0,0,1]`.
- `view_center`: *(3,) float*, The position to rotate the scene about. Defaults to the scene's centroid.

- `use_raymond_lighting`: *bool*, If *True*, an additional set of three directional lights that move with the camera will be added to the scene. Defaults to *False*.
- `use_direct_lighting`: *bool*, If *True*, an additional directional light that moves with the camera and points out of it will be added to the scene. Defaults to *False*.
- `lighting_intensity`: *float*, The overall intensity of the viewer's additional lights (when they're in use). Defaults to 3.0.
- `use_perspective_cam`: *bool*, If *True*, a perspective camera will be used. Otherwise, an orthographic camera is used. Defaults to *True*.
- `save_directory`: *str*, A directory to open the file dialogs in. Defaults to *None*.
- `window_title`: *str*, A title for the viewer's application window. Defaults to *"Scene Viewer"*.
- `refresh_rate`: *float*, A refresh rate for rendering, in Hertz. Defaults to *30.0*.
- `fullscreen`: *bool*, Whether to make viewer fullscreen. Defaults to *False*.
- `show_world_axis`: *bool*, Whether to show the world axis. Defaults to *False*.
- `show_mesh_axes`: *bool*, Whether to show the individual mesh axes. Defaults to *False*.
- `caption`: *list of dict*, Text caption(s) to display on the viewer. Defaults to *None*.

Type dict

viewport_size
The width and height of the viewing window.

Type (2,) int

Methods Documentation

close_external()
Close the viewer from another thread.

This function will wait for the actual close, so you immediately manipulate the scene afterwards.

on_close()
Exit the event loop when the window is closed.

on_draw()
Redraw the scene into the viewing window.

on_key_press(*symbol*, *modifiers*)
Record a key press.

on_mouse_drag(*x*, *y*, *dx*, *dy*, *buttons*, *modifiers*)
Record a mouse drag.

on_mouse_press(*x*, *y*, *buttons*, *modifiers*)
Record an initial mouse press.

on_mouse_release(*x*, *y*, *button*, *modifiers*)
Record a mouse release.

on_mouse_scroll(*x*, *y*, *dx*, *dy*)
Record a mouse scroll.

on_resize(*width*, *height*)
Resize the camera and trackball when the window is resized.

save_gif (*filename=None*)
Save the stored GIF frames to a file.

To use this asynchronously, run the viewer with the `record` flag and the `run_in_thread` flags set. Kill the viewer after your desired time with *Viewer.close_external()*, and then call *Viewer.save_gif()*.

Parameters **filename** (*str*) – The file to save the GIF to. If not specified, a file dialog will be opened to ask the user where to save the GIF file.

3.7 Off-Screen Rendering

3.7.1 Classes

OffscreenRenderer(viewport_width, . . . [, . . .])	A wrapper for offscreen rendering.

OffscreenRenderer

class pyrender.offscreen.**OffscreenRenderer**(*viewport_width*, *viewport_height*, *point_size=1.0*)

Bases: object

A wrapper for offscreen rendering.

Parameters

- **viewport_width** (*int*) – The width of the main viewport, in pixels.
- **viewport_height** (*int*) – The height of the main viewport, in pixels.
- **point_size** (*float*) – The size of screen-space points in pixels.

Attributes Summary

point_size	The pixel size of points in point clouds.
viewport_height	The height of the main viewport, in pixels.
viewport_width	The width of the main viewport, in pixels.

Methods Summary

delete()	Free all OpenGL resources.
render(scene[, flags, seg_node_map])	Render a scene with the given set of flags.

Attributes Documentation

point_size
The pixel size of points in point clouds.

Type float

viewport_height
The height of the main viewport, in pixels.

Type int

viewport_width
The width of the main viewport, in pixels.

Type int

Methods Documentation

delete()
Free all OpenGL resources.

render(*scene*, *flags=0*, *seg_node_map=None*)
Render a scene with the given set of flags.

Parameters

- **scene** (Scene) – A scene to render.
- **flags** (*int*) – A bitwise or of one or more flags from *RenderFlags*.
- **seg_node_map** (*dict*) – A map from *Node* objects to (3,) colors for each. If specified along with flags set to *RenderFlags.SEG*, the color image will be a segmentation image.

Returns

- **color_im** (*(h, w, 3) uint8 or (h, w, 4) uint8*) – The color buffer in RGB format, or in RGBA format if *RenderFlags.RGBA* is set. Not returned if flags includes *RenderFlags.DEPTH_ONLY*.
- **depth_im** (*(h, w) float32*) – The depth buffer in linear units.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

N

O

P

R

S

T

V

W

X

Y

Z