

---

# **pyrender Documentation**

***Release 0.1.29***

**Matthew Matl**

**Aug 04, 2019**



---

## Contents

---

<b>1</b>	<b>Installation Guide</b>	<b>3</b>
1.1	Python Installation . . . . .	3
1.2	Getting Pyrender Working with OSMesa . . . . .	3
1.3	Building Documentation . . . . .	5
<b>2</b>	<b>User Guide</b>	<b>7</b>
2.1	Quickstart . . . . .	7
2.2	Loading and Configuring Models . . . . .	10
2.3	Creating Lights . . . . .	13
2.4	Creating Cameras . . . . .	14
2.5	Creating Scenes . . . . .	14
2.6	Offscreen Rendering . . . . .	15
2.7	Live Scene Viewer . . . . .	18
<b>3</b>	<b>Pyrender API Documentation</b>	<b>19</b>
3.1	Constants . . . . .	19
3.2	Cameras . . . . .	23
3.3	Lighting . . . . .	28
3.4	Objects . . . . .	33
3.5	Scenes . . . . .	48
3.6	On-Screen Viewer . . . . .	54
3.7	Off-Screen Rendering . . . . .	60
<b>4</b>	<b>Indices and tables</b>	<b>63</b>
	<b>Index</b>	<b>65</b>



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](#).





### 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-xvnc --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 git@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`.



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

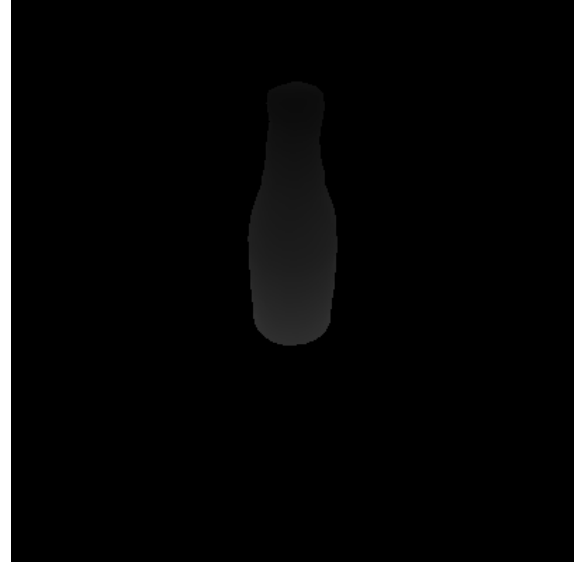
```
>>> fuze_trimesh = trimesh.load('examples/models/fuze.obj')
>>> mesh = pyrender.Mesh.from_trimesh(fuze_trimesh)
>>> scene = pyrender.Scene()
>>> scene.add(mesh)
>>> camera = pyrender.PerspectiveCamera(yfov=np.pi / 3.0, aspectRatio=1.0)
>>> s = np.sqrt(2)/2
>>> camera_pose = np.array([
...     [0.0,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

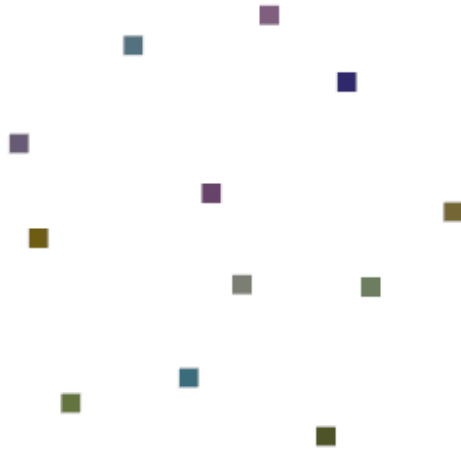
### Point Sprites

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 instantiating 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(m, poses=poses)
```



## 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

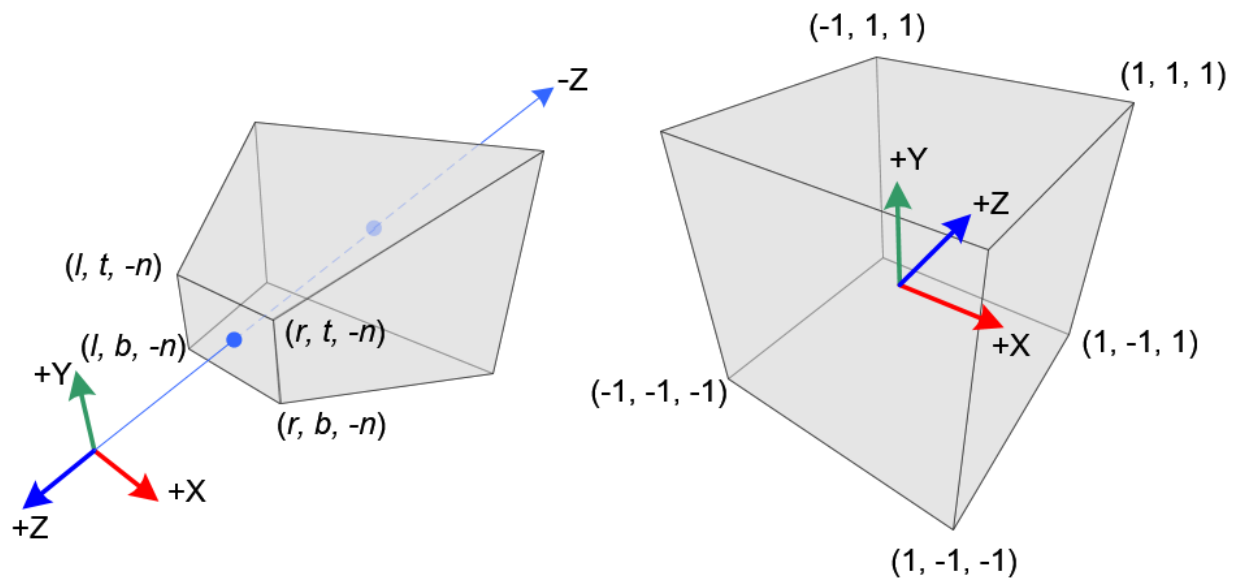
Pyrender supports three camera types – *PerspectiveCamera* and *IntrinsicsCamera* types, which render scenes as a human would see them, and *OrthographicCamera* types, which preserve distances between points.

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 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`.

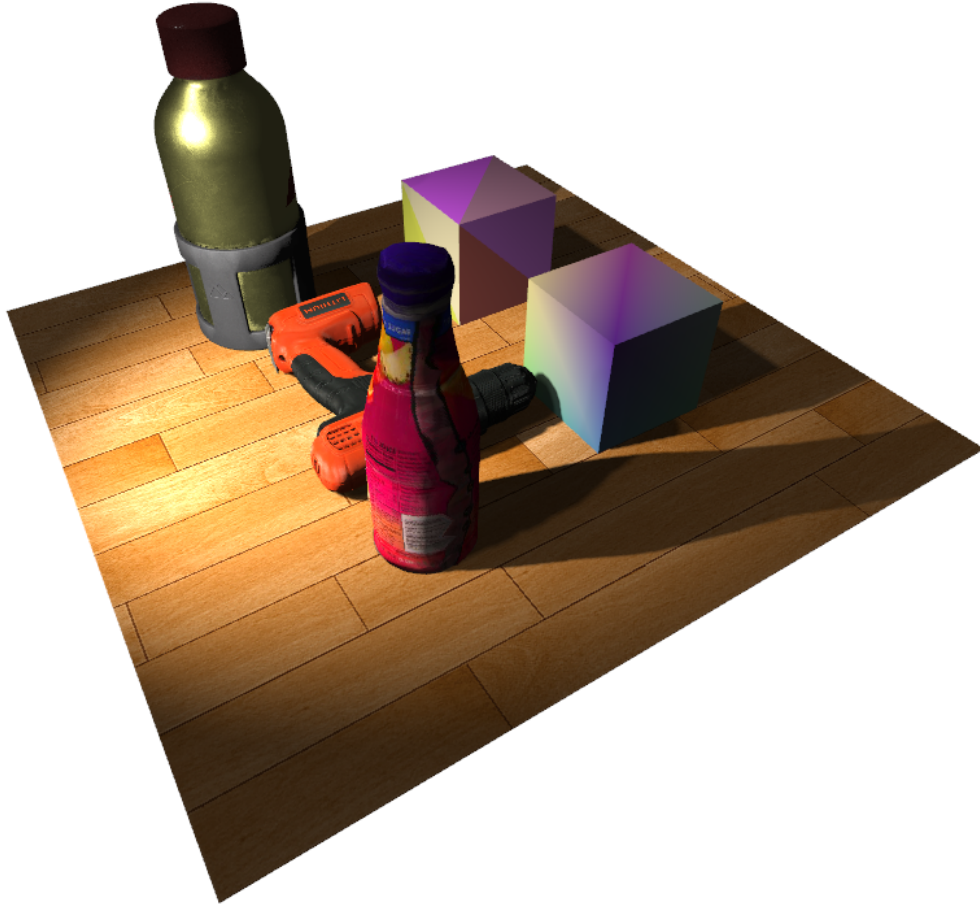
## 2.6.2 Running the Renderer

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
```

## 3.1 Constants

### 3.1.1 Classes

<i>RenderFlags</i>	Flags for rendering in the scene.
<i>TextAlign</i>	Text alignment options for captions.
<i>GLTF</i>	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

<i>ALL_SOLID</i>	Render all meshes as solids.
<i>ALL_WIREFRAME</i>	Render all meshes as wireframes.
<i>DEPTH_ONLY</i>	Only render the depth buffer.
<i>FACE_NORMALS</i>	Render face normals.
<i>FLIP_WIREFRAME</i>	Invert the status of wireframe rendering for each mesh.

Continued on next page

Table 2 – continued from previous page

<i>NONE</i>	Normal PBR Render.
<i>OFFSCREEN</i>	Render offscreen and return the depth and (optionally) color buffers.
<i>RGBA</i>	Render the color buffer with the alpha channel enabled.
<i>SHADOWS_ALL</i>	Render shadows for all lights.
<i>SHADOWS_DIRECTIONAL</i>	Render shadows for directional lights.
<i>SHADOWS_POINT</i>	Render shadows for point lights.
<i>SHADOWS_SPOT</i>	Render shadows for spot lights.
<i>SKIP_CULL_FACES</i>	Do not cull back faces.
<i>VERTEX_NORMALS</i>	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.

**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.

**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

Text alignment options for captions.

Only use one at a time.

### Attributes Summary

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

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

<i>Camera</i> ([znear, zfar, name])	Abstract base class for all cameras.
<i>PerspectiveCamera</i> (yfov[, znear, zfar, ...])	A perspective camera for perspective projection.
<i>OrthographicCamera</i> (xmag, ymag[, znear, ...])	A perspective camera for perspective projection.
<i>IntrinsicsCamera</i> (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

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

#### Methods Summary

---

<code>get_projection_matrix([width, height])</code>	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

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

## Methods Summary

<code>get_projection_matrix([width, height])</code>	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`

A perspective camera for perspective 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

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

### Methods Summary

<i>get_projection_matrix</i> ([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 meters.
- **fy** (*float*) – Y-axis focal length in meters.
- **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

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

## Methods Summary

<code>get_projection_matrix(width, height)</code>	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

<i>Light</i> ([color, intensity, name])	Base class for all light objects.
<i>DirectionalLight</i> ([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.
<i>SpotLight</i> ([color, intensity, range, ...])	Spot lights emit light in a cone in the direction of the local -z axis.
<i>PointLight</i> ([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

<i>color</i>	The light’s color.
<i>intensity</i>	The light’s intensity in candela or lux.
<i>name</i>	The user-defined name of this object.
<i>shadow_texture</i>	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<sup>2</sup>). Defaults to 1.0
- **name** (*str*, *optional*) – Name of the light.

### Attributes Summary

<i>color</i>	The light’s color.
<i>intensity</i>	The light’s intensity in candela or lux.
<i>name</i>	The user-defined name of this object.
<i>shadow_texture</i>	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  $\text{PI} / 2.0$ . Defaults to  $\text{PI} / 4.0$ .
- **name** (*str*, *optional*) – Name of the light.

### Attributes Summary

<i>color</i>	The light’s color.
<i>innerConeAngle</i>	The inner cone angle in radians.
<i>intensity</i>	The light’s intensity in candela or lux.
<i>name</i>	The user-defined name of this object.
<i>outerConeAngle</i>	The outer cone angle in radians.
<i>range</i>	The cutoff distance for the light.
<i>shadow_texture</i>	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

<i>color</i>	The light's color.
<i>intensity</i>	The light's intensity in candela or lux.
<i>name</i>	The user-defined name of this object.
<i>range</i>	The cutoff distance for the light.
<i>shadow_texture</i>	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

<i>IntrinsicsCamera</i> (fx, fy, cx, cy[, znear, ...])	A perspective camera with custom intrinsics.
<i>Sampler</i> ([name, magFilter, minFilter, wrapS, ...])	Texture sampler properties for filtering and wrapping modes.
<i>Texture</i> ([name, sampler, source, ...])	A texture and its sampler.
<i>Material</i> ([name, normalTexture, ...])	Base for standard glTF 2.0 materials.
<i>MetallicRoughnessMaterial</i> ([name, ...])	A material based on the metallic-roughness material model from Physically-Based Rendering (PBR) methodology.
<i>Primitive</i> (positions[, normals, tangents, ...])	A primitive object which can be rendered.
<i>Mesh</i> (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 meters.
- **fy** (*float*) – Y-axis focal length in meters.
- **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

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

##### Methods Summary

---

<code>get_projection_matrix</code> (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

<i>magFilter</i>	Magnification filter type.
<i>minFilter</i>	Minification filter type.
<i>name</i>	The user-defined name of this object.
<i>wrapS</i>	S (U) wrapping mode.
<i>wrapT</i>	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_FLOAT)
```

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

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



## Methods Summary

<code>delete()</code>	Remove this texture from the OpenGL context.
<code>is_transparent([cutoff])</code>	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

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

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

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

Continued on next page



Table 27 – continued from previous page

<i>targets</i>	Morph target indices.
<i>texcoord_0</i>	The first set of UV texture coordinates.
<i>texcoord_1</i>	The second set of UV texture coordinates.
<i>weights_0</i>	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 ABB.

**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

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

## Methods Summary

<i>from_points</i> (points[, colors, normals, ...])	Create a Mesh from a set of points.
<i>from_trimesh</i> (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, optional) – 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

<i>IntrinsicsCamera</i> (fx, fy, cx, cy[, znear, ...])	A perspective camera with custom intrinsics.
<i>Node</i> ([name, camera, children, skin, matrix, ...])	A node in the node hierarchy.
<i>Scene</i> ([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

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

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

## Methods Summary

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

Get 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

---

<code>Viewer(scene[, viewport_size, render_flags, ...])</code>	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

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

## Methods Summary

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

---

<code>OffscreenRenderer</code> (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

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

#### Methods Summary

<code>delete()</code>	Free all OpenGL resources.
<code>render</code> (scene[, flags])	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*)

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`.

### 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`



## A

[add\(\)](#) (*pyrender.Scene* method), 53  
[add\\_node\(\)](#) (*pyrender.Scene* method), 53  
[ALL\\_SOLID](#) (*pyrender.constants.RenderFlags* attribute), 20  
[ALL\\_WIREFRAME](#) (*pyrender.constants.RenderFlags* attribute), 20  
[alphaCutoff](#) (*pyrender.Material* attribute), 39  
[alphaCutoff](#) (*pyrender.MetallicRoughnessMaterial* attribute), 42  
[alphaMode](#) (*pyrender.Material* attribute), 39  
[alphaMode](#) (*pyrender.MetallicRoughnessMaterial* attribute), 42  
[ambient\\_light](#) (*pyrender.Scene* attribute), 51  
[aspectRatio](#) (*pyrender.camera.PerspectiveCamera* attribute), 25

## B

[baseColorFactor](#) (*pyrender.MetallicRoughnessMaterial* attribute), 42  
[baseColorTexture](#) (*pyrender.MetallicRoughnessMaterial* attribute), 42  
[bg\\_color](#) (*pyrender.Scene* attribute), 51  
[BOTTOM\\_CENTER](#) (*pyrender.constants.TextAlign* attribute), 21  
[BOTTOM\\_LEFT](#) (*pyrender.constants.TextAlign* attribute), 21  
[BOTTOM\\_RIGHT](#) (*pyrender.constants.TextAlign* attribute), 21  
[bounds](#) (*pyrender.Mesh* attribute), 47  
[bounds](#) (*pyrender.Primitive* attribute), 45  
[bounds](#) (*pyrender.Scene* attribute), 51  
[buf\\_flags](#) (*pyrender.Primitive* attribute), 45

## C

[Camera](#) (class in *pyrender.camera*), 23  
[camera](#) (*pyrender.Node* attribute), 49

[camera\\_nodes](#) (*pyrender.Scene* attribute), 52  
[cameras](#) (*pyrender.Scene* attribute), 52  
[CENTER](#) (*pyrender.constants.TextAlign* attribute), 21  
[CENTER\\_LEFT](#) (*pyrender.constants.TextAlign* attribute), 21  
[CENTER\\_RIGHT](#) (*pyrender.constants.TextAlign* attribute), 21  
[centroid](#) (*pyrender.Mesh* attribute), 47  
[centroid](#) (*pyrender.Primitive* attribute), 45  
[centroid](#) (*pyrender.Scene* attribute), 52  
[children](#) (*pyrender.Node* attribute), 49  
[CLAMP\\_TO\\_EDGE](#) (*pyrender.constants.GLTF* attribute), 22  
[clear\(\)](#) (*pyrender.Scene* method), 53  
[close\\_external\(\)](#) (*pyrender.viewer.Viewer* method), 59  
[color](#) (*pyrender.light.DirectionalLight* attribute), 30  
[color](#) (*pyrender.light.Light* attribute), 29  
[color](#) (*pyrender.light.PointLight* attribute), 32  
[color](#) (*pyrender.light.SpotLight* attribute), 31  
[color\\_0](#) (*pyrender.Primitive* attribute), 45  
[cx](#) (*pyrender.camera.IntrinsicsCamera* attribute), 27  
[cx](#) (*pyrender.IntrinsicsCamera* attribute), 34  
[cy](#) (*pyrender.camera.IntrinsicsCamera* attribute), 27  
[cy](#) (*pyrender.IntrinsicsCamera* attribute), 34

## D

[data\\_format](#) (*pyrender.Texture* attribute), 37  
[delete\(\)](#) (*pyrender.offscreen.OffscreenRenderer* method), 61  
[delete\(\)](#) (*pyrender.Primitive* method), 46  
[delete\(\)](#) (*pyrender.Texture* method), 37  
[DEPTH\\_ONLY](#) (*pyrender.constants.RenderFlags* attribute), 20  
[directional\\_light\\_nodes](#) (*pyrender.Scene* attribute), 52  
[directional\\_lights](#) (*pyrender.Scene* attribute), 52  
[DirectionalLight](#) (class in *pyrender.light*), 29  
[doubleSided](#) (*pyrender.Material* attribute), 39

doubleSided (*pyrender.MetallicRoughnessMaterial attribute*), 42

## E

emissiveFactor (*pyrender.Material attribute*), 39  
emissiveFactor (*pyrender.MetallicRoughnessMaterial attribute*), 42

emissiveTexture (*pyrender.Material attribute*), 39  
emissiveTexture (*pyrender.MetallicRoughnessMaterial attribute*), 42

extents (*pyrender.Mesh attribute*), 47  
extents (*pyrender.Primitive attribute*), 45  
extents (*pyrender.Scene attribute*), 52

## F

FACE\_NORMALS (*pyrender.constants.RenderFlags attribute*), 20  
FLIP\_WIREFRAME (*pyrender.constants.RenderFlags attribute*), 20  
from\_points() (*pyrender.Mesh static method*), 48  
from\_trimesh() (*pyrender.Mesh static method*), 48  
from\_trimesh\_scene() (*pyrender.Scene static method*), 53  
fx (*pyrender.camera.IntrinsicsCamera attribute*), 28  
fx (*pyrender.IntrinsicsCamera attribute*), 34  
fy (*pyrender.camera.IntrinsicsCamera attribute*), 28  
fy (*pyrender.IntrinsicsCamera attribute*), 34

## G

get\_nodes() (*pyrender.Scene method*), 54  
get\_pose() (*pyrender.Scene method*), 54  
get\_projection\_matrix() (*pyrender.camera.Camera method*), 24  
get\_projection\_matrix() (*pyrender.camera.IntrinsicsCamera method*), 28  
get\_projection\_matrix() (*pyrender.camera.OrthographicCamera method*), 27  
get\_projection\_matrix() (*pyrender.camera.PerspectiveCamera method*), 25  
get\_projection\_matrix() (*pyrender.camera.IntrinsicsCamera method*), 34  
GLTF (*class in pyrender.constants*), 21

## H

has\_node() (*pyrender.Scene method*), 54  
height (*pyrender.Texture attribute*), 37

## I

indices (*pyrender.Primitive attribute*), 45

innerConeAngle (*pyrender.light.SpotLight attribute*), 31

intensity (*pyrender.light.DirectionLight attribute*), 30

intensity (*pyrender.light.Light attribute*), 29  
intensity (*pyrender.light.PointLight attribute*), 32  
intensity (*pyrender.light.SpotLight attribute*), 31

IntrinsicsCamera (*class in pyrender*), 33  
IntrinsicsCamera (*class in pyrender.camera*), 27  
is\_active (*pyrender.viewer.Viewer attribute*), 57  
is\_transparent (*pyrender.Material attribute*), 39  
is\_transparent (*pyrender.Mesh attribute*), 47  
is\_transparent (*pyrender.MetallicRoughnessMaterial attribute*), 42

is\_transparent (*pyrender.Primitive attribute*), 45  
is\_transparent() (*pyrender.Texture method*), 37  
is\_visible (*pyrender.Mesh attribute*), 47

## J

joints\_0 (*pyrender.Primitive attribute*), 45

## L

Light (*class in pyrender.light*), 29  
light (*pyrender.Node attribute*), 50  
light\_nodes (*pyrender.Scene attribute*), 52  
lights (*pyrender.Scene attribute*), 52  
LINE\_LOOP (*pyrender.constants.GLTF attribute*), 22  
LINE\_STRIP (*pyrender.constants.GLTF attribute*), 22  
LINEAR (*pyrender.constants.GLTF attribute*), 22  
LINEAR\_MIPMAP\_LINEAR (*pyrender.constants.GLTF attribute*), 22  
LINEAR\_MIPMAP\_NEAREST (*pyrender.constants.GLTF attribute*), 22  
LINES (*pyrender.constants.GLTF attribute*), 22

## M

magFilter (*pyrender.Sampler attribute*), 35  
main\_camera\_node (*pyrender.Scene attribute*), 52  
Material (*class in pyrender*), 38  
material (*pyrender.Primitive attribute*), 45  
matrix (*pyrender.Node attribute*), 50  
Mesh (*class in pyrender*), 46  
mesh (*pyrender.Node attribute*), 50  
mesh\_nodes (*pyrender.Scene attribute*), 52  
meshes (*pyrender.Scene attribute*), 52  
metallicFactor (*pyrender.MetallicRoughnessMaterial attribute*), 43  
MetallicRoughnessMaterial (*class in pyrender*), 40  
metallicRoughnessTexture (*pyrender.MetallicRoughnessMaterial attribute*), 43

minFilter (*pyrender.Sampler attribute*), 35  
 MIRRORED\_REPEAT (*pyrender.constants.GLTF attribute*), 22  
 mode (*pyrender.Primitive attribute*), 45

## N

name (*pyrender.camera.Camera attribute*), 24  
 name (*pyrender.camera.IntrinsicsCamera attribute*), 28  
 name (*pyrender.camera.OrthographicCamera attribute*), 26  
 name (*pyrender.camera.PerspectiveCamera attribute*), 25  
 name (*pyrender.IntrinsicsCamera attribute*), 34  
 name (*pyrender.light.DirectionalLight attribute*), 30  
 name (*pyrender.light.Light attribute*), 29  
 name (*pyrender.light.PointLight attribute*), 32  
 name (*pyrender.light.SpotLight attribute*), 31  
 name (*pyrender.Material attribute*), 39  
 name (*pyrender.Mesh attribute*), 47  
 name (*pyrender.MetallicRoughnessMaterial attribute*), 43  
 name (*pyrender.Node attribute*), 50  
 name (*pyrender.Sampler attribute*), 36  
 name (*pyrender.Scene attribute*), 52  
 name (*pyrender.Texture attribute*), 37  
 NEAREST (*pyrender.constants.GLTF attribute*), 22  
 NEAREST\_MIPMAP\_LINEAR (*pyrender.constants.GLTF attribute*), 22  
 NEAREST\_MIPMAP\_NEAREST (*pyrender.constants.GLTF attribute*), 22  
 Node (*class in pyrender*), 49  
 nodes (*pyrender.Scene attribute*), 52  
 NONE (*pyrender.constants.RenderFlags attribute*), 20  
 normals (*pyrender.Primitive attribute*), 45  
 normalTexture (*pyrender.Material attribute*), 39  
 normalTexture (*pyrender.MetallicRoughnessMaterial attribute*), 43

## O

occlusionTexture (*pyrender.Material attribute*), 39  
 occlusionTexture (*pyrender.MetallicRoughnessMaterial attribute*), 43  
 OFFSCREEN (*pyrender.constants.RenderFlags attribute*), 20  
 OffscreenRenderer (*class in pyrender.offscreen*), 60  
 on\_close() (*pyrender.viewer.Viewer method*), 59  
 on\_draw() (*pyrender.viewer.Viewer method*), 59  
 on\_key\_press() (*pyrender.viewer.Viewer method*), 59  
 on\_mouse\_drag() (*pyrender.viewer.Viewer method*), 59

on\_mouse\_press() (*pyrender.viewer.Viewer method*), 59  
 on\_mouse\_release() (*pyrender.viewer.Viewer method*), 59  
 on\_mouse\_scroll() (*pyrender.viewer.Viewer method*), 59  
 on\_resize() (*pyrender.viewer.Viewer method*), 59  
 OrthographicCamera (*class in pyrender.camera*), 25  
 outerConeAngle (*pyrender.light.SpotLight attribute*), 31

## P

PerspectiveCamera (*class in pyrender.camera*), 24  
 point\_light\_nodes (*pyrender.Scene attribute*), 52  
 point\_lights (*pyrender.Scene attribute*), 53  
 point\_size (*pyrender.offscreen.OffscreenRenderer attribute*), 60  
 PointLight (*class in pyrender.light*), 32  
 POINTS (*pyrender.constants.GLTF attribute*), 22  
 poses (*pyrender.Primitive attribute*), 46  
 positions (*pyrender.Primitive attribute*), 46  
 Primitive (*class in pyrender*), 43  
 primitives (*pyrender.Mesh attribute*), 47

## R

range (*pyrender.light.PointLight attribute*), 32  
 range (*pyrender.light.SpotLight attribute*), 31  
 registered\_keys (*pyrender.viewer.Viewer attribute*), 57  
 remove\_node() (*pyrender.Scene method*), 54  
 render() (*pyrender.offscreen.OffscreenRenderer method*), 61  
 render\_flags (*pyrender.viewer.Viewer attribute*), 58  
 render\_lock (*pyrender.viewer.Viewer attribute*), 58  
 RenderFlags (*class in pyrender.constants*), 19  
 REPEAT (*pyrender.constants.GLTF attribute*), 22  
 RGBA (*pyrender.constants.RenderFlags attribute*), 20  
 rotation (*pyrender.Node attribute*), 50  
 roughnessFactor (*pyrender.MetallicRoughnessMaterial attribute*), 43  
 run\_in\_thread (*pyrender.viewer.Viewer attribute*), 58

## S

Sampler (*class in pyrender*), 34  
 sampler (*pyrender.Texture attribute*), 37  
 save\_gif() (*pyrender.viewer.Viewer method*), 59  
 scale (*pyrender.Mesh attribute*), 47  
 scale (*pyrender.Node attribute*), 50  
 scale (*pyrender.Primitive attribute*), 46  
 scale (*pyrender.Scene attribute*), 53  
 Scene (*class in pyrender*), 50  
 scene (*pyrender.viewer.Viewer attribute*), 58  
 set\_pose() (*pyrender.Scene method*), 54

`shadow_texture` (`pyrender.light.DirectionalLight` attribute), 30  
`shadow_texture` (`pyrender.light.Light` attribute), 29  
`shadow_texture` (`pyrender.light.PointLight` attribute), 32  
`shadow_texture` (`pyrender.light.SpotLight` attribute), 31  
`SHADOWS_ALL` (`pyrender.constants.RenderFlags` attribute), 20  
`SHADOWS_DIRECTIONAL` (`pyrender.constants.RenderFlags` attribute), 20  
`SHADOWS_POINT` (`pyrender.constants.RenderFlags` attribute), 20  
`SHADOWS_SPOT` (`pyrender.constants.RenderFlags` attribute), 20  
`skin` (`pyrender.Node` attribute), 50  
`SKIP_CULL_FACES` (`pyrender.constants.RenderFlags` attribute), 20  
`smooth` (`pyrender.Material` attribute), 40  
`smooth` (`pyrender.MetallicRoughnessMaterial` attribute), 43  
`source` (`pyrender.Texture` attribute), 37  
`source_channels` (`pyrender.Texture` attribute), 37  
`spot_light_nodes` (`pyrender.Scene` attribute), 53  
`spot_lights` (`pyrender.Scene` attribute), 53  
`SpotLight` (class in `pyrender.light`), 30

## T

`tangents` (`pyrender.Primitive` attribute), 46  
`targets` (`pyrender.Primitive` attribute), 46  
`tex_flags` (`pyrender.Material` attribute), 40  
`tex_flags` (`pyrender.MetallicRoughnessMaterial` attribute), 43  
`tex_type` (`pyrender.Texture` attribute), 37  
`texcoord_0` (`pyrender.Primitive` attribute), 46  
`texcoord_1` (`pyrender.Primitive` attribute), 46  
`TextAlign` (class in `pyrender.constants`), 21  
`Texture` (class in `pyrender`), 36  
`textures` (`pyrender.Material` attribute), 40  
`textures` (`pyrender.MetallicRoughnessMaterial` attribute), 43  
`TOP_CENTER` (`pyrender.constants.TextAlign` attribute), 21  
`TOP_LEFT` (`pyrender.constants.TextAlign` attribute), 21  
`TOP_RIGHT` (`pyrender.constants.TextAlign` attribute), 21  
`translation` (`pyrender.Node` attribute), 50  
`TRIANGLE_FAN` (`pyrender.constants.GLTF` attribute), 23  
`TRIANGLE_STRIP` (`pyrender.constants.GLTF` attribute), 23  
`TRIANGLES` (`pyrender.constants.GLTF` attribute), 23

## V

`VERTEX_NORMALS` (`pyrender.constants.RenderFlags`

attribute), 20

`Viewer` (class in `pyrender.viewer`), 55

`viewer_flags` (`pyrender.viewer.Viewer` attribute), 58

`viewport_height` (`pyrender.offscreen.OffscreenRenderer` attribute), 60

`viewport_size` (`pyrender.viewer.Viewer` attribute), 59

`viewport_width` (`pyrender.offscreen.OffscreenRenderer` attribute), 61

## W

`weights` (`pyrender.Mesh` attribute), 47

`weights_0` (`pyrender.Primitive` attribute), 46

`width` (`pyrender.Texture` attribute), 37

`wireframe` (`pyrender.Material` attribute), 40

`wireframe` (`pyrender.MetallicRoughnessMaterial` attribute), 43

`wrapS` (`pyrender.Sampler` attribute), 36

`wrapT` (`pyrender.Sampler` attribute), 36

## X

`xmag` (`pyrender.camera.OrthographicCamera` attribute), 26

## Y

`yfov` (`pyrender.camera.PerspectiveCamera` attribute), 25

`ymag` (`pyrender.camera.OrthographicCamera` attribute), 26

## Z

`zfar` (`pyrender.camera.Camera` attribute), 24

`zfar` (`pyrender.camera.IntrinsicsCamera` attribute), 28

`zfar` (`pyrender.camera.OrthographicCamera` attribute), 26

`zfar` (`pyrender.camera.PerspectiveCamera` attribute), 25

`zfar` (`pyrender.IntrinsicsCamera` attribute), 34

`znear` (`pyrender.camera.Camera` attribute), 24

`znear` (`pyrender.camera.IntrinsicsCamera` attribute), 28  
`znear` (`pyrender.camera.OrthographicCamera` attribute), 26

`znear` (`pyrender.camera.PerspectiveCamera` attribute), 25

`znear` (`pyrender.IntrinsicsCamera` attribute), 34