Fresnel Documentation

Release 0.13.1

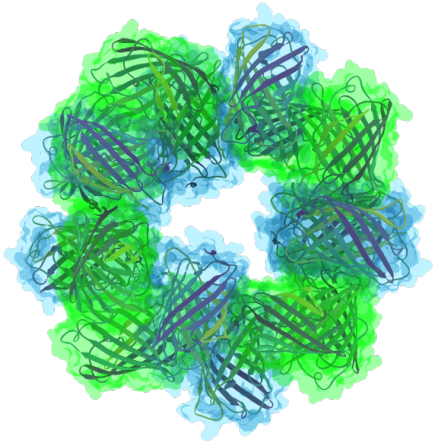
The Regents of the University of Michigan

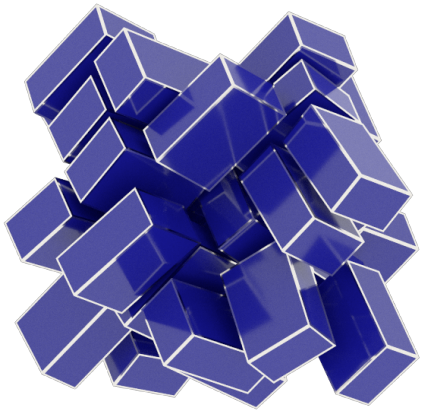
Mar 11, 2021

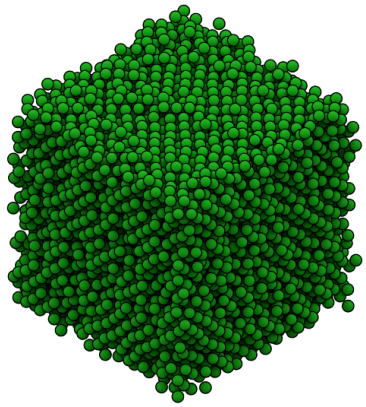
EXAMPLES

fresnel is a python library for path tracing publication quality images of soft matter simulations in real time. The fastest render performance is possible on NVIDIA GPUs using their OptiX ray tracing engine. **fresnel** also supports multi-core CPUs using Intel's Embree ray tracing kernels. Path tracing enables high quality global illumination and advanced rendering effects. **Fresnel** offers intuitive material parameters (like *roughness*, *specular*, and *metal*) and simple predefined lighting setups (like *cloudy* and *lightbox*).

Here are a few samples of what **fresnel** can do:



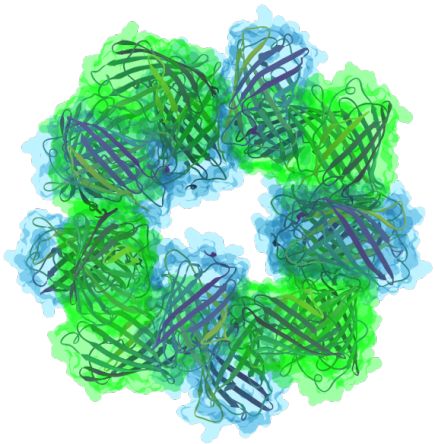


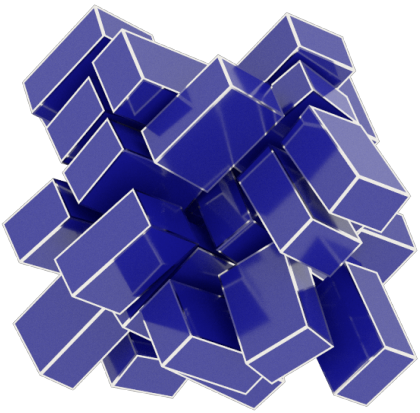


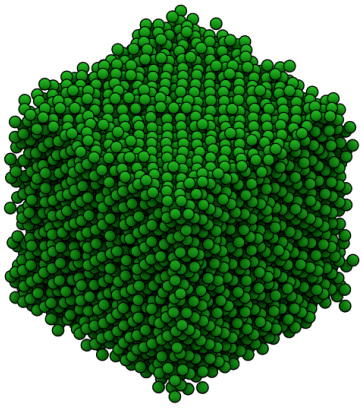
CHAPTER

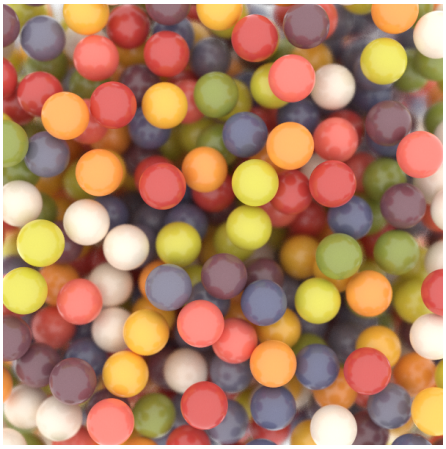
ONE

GALLERY







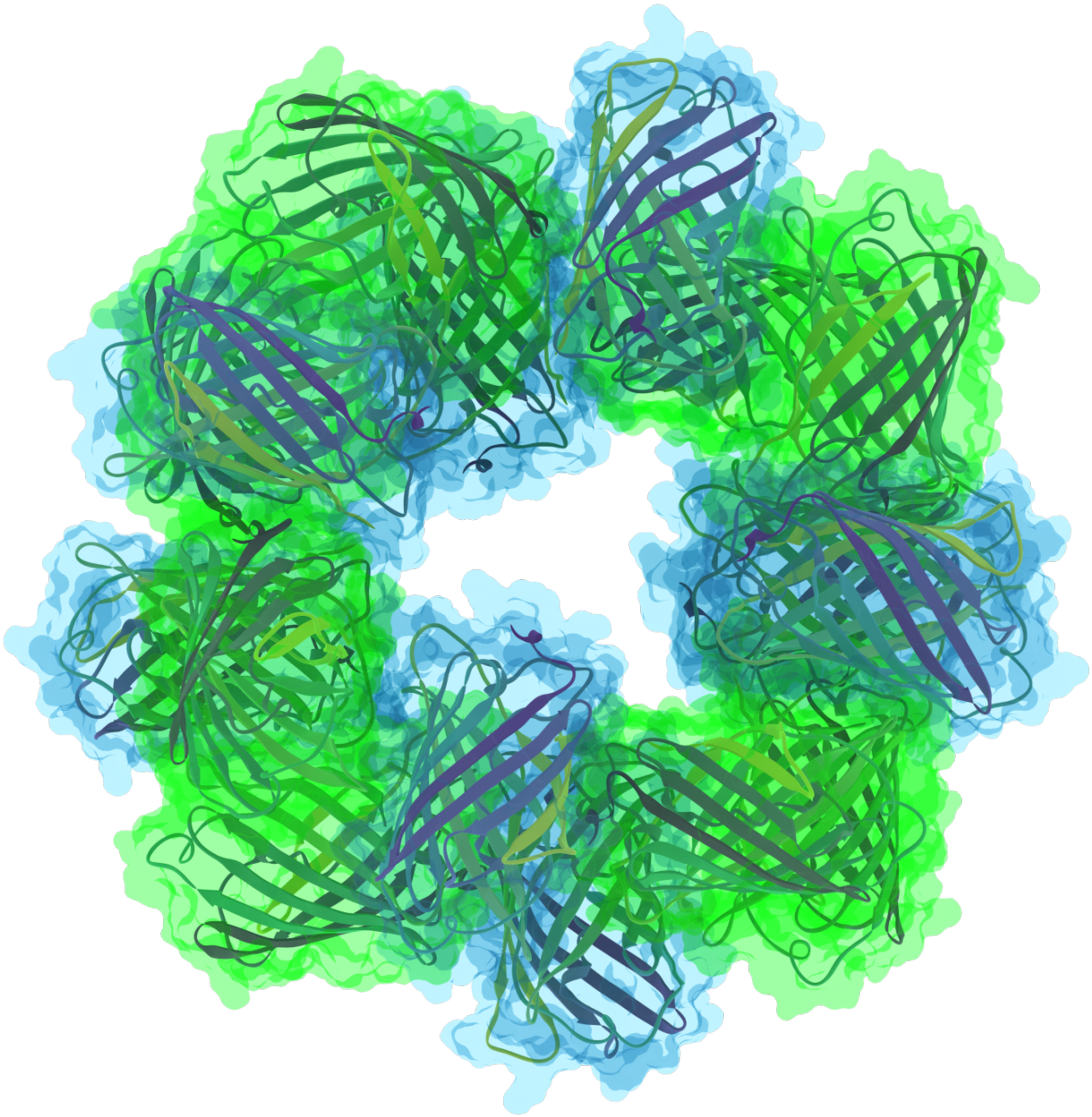


CHAPTER

TWO

RESEARCH

2.1 Protomer



Protomer on the cover of Nature Chemistry volume 11, issue 3:

- Ribbon geometry: `geometry.Mesh`
  - `material`: *roughness* = 1.0, *specular* = 1.0, *metal* = 0, *spec_trans* = 0
  - Generated with: ribbon
- Molecular surface: `geometry.Mesh`
  - `material`: *roughness* = 2.0, *specular* = 0.95, *metal* = 0, *spec_trans* = 0.95
  - Generated with MSMS
- Lighting: `light.lightbox` with background light
- Rendered with: `tracer.Path`: *samples* = 64, *light_samples* = 32 on the GPU

Author

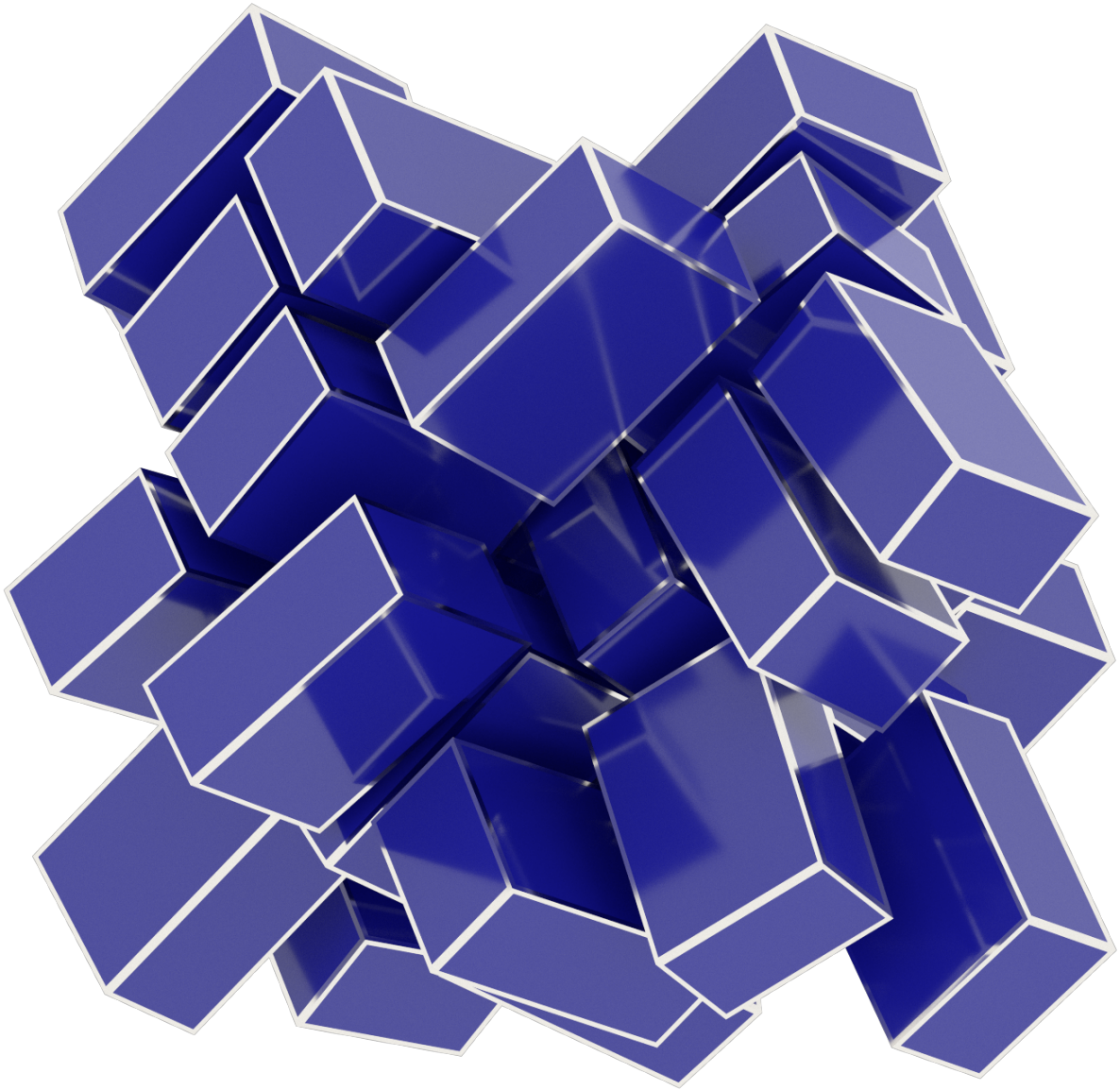
Jens Glaser

CHAPTER

THREE

FEATURES

3.1 Cuboids



Cuboid example script:

- Geometry: *geometry.ConvexPolyhedron*: *outline_width* = 0.015
  - *material*: *roughness* = 0.1, *specular* = 1, *metal* = 0, *spec_trans* = 0
  - *outline_material*: *roughness* = 0.1, *metal* = 1, *spec_trans* = 0, *color* = (0.95,0.93,0.88)
  - position, orientation: output of a HOOMD simulation
- Lighting: *light.lightbox*
- Rendered with: *tracer.Path*: *samples* = 256, *light_samples* = 16

Source code

```
"""Cuboid example scene."""

import fresnel
import numpy
import PIL
import sys

data = numpy.load('cuboids.npz')

scene = fresnel.Scene()
scene.lights = fresnel.light.lightbox()
W, H, D = data['width']
poly_info = fresnel.util.convex_polyhedron_from_vertices([
    [-W, -H, -D],
    [-W, -H, D],
    [-W, H, -D],
    [-W, H, D],
    [W, -H, -D],
    [W, -H, D],
    [W, H, -D],
    [W, H, D],
])

geometry = fresnel.geometry.ConvexPolyhedron(scene,
                                             poly_info,
                                             position=data['position'],
                                             orientation=data['orientation'],
                                             outline_width=0.015)
geometry.material = fresnel.material.Material(color=fresnel.color.linear(
    [0.1, 0.1, 0.6]),
                                              roughness=0.1,
                                              specular=1)
geometry.outline_material = fresnel.material.Material(color=(0.95, 0.93, 0.88),
                                                      roughness=0.1,
                                                      metal=1.0)

scene.camera = fresnel.camera.Orthographic.fit(scene, view='front')
out = fresnel.pathtrace(scene, samples=64, light_samples=32, w=580, h=580)
PIL.Image.fromarray(out[:], mode='RGBA').save('cuboid.png')

if len(sys.argv) > 1 and sys.argv[1] == 'hires':
    out = fresnel.pathtrace(scene,
                            samples=256,
```

(continues on next page)

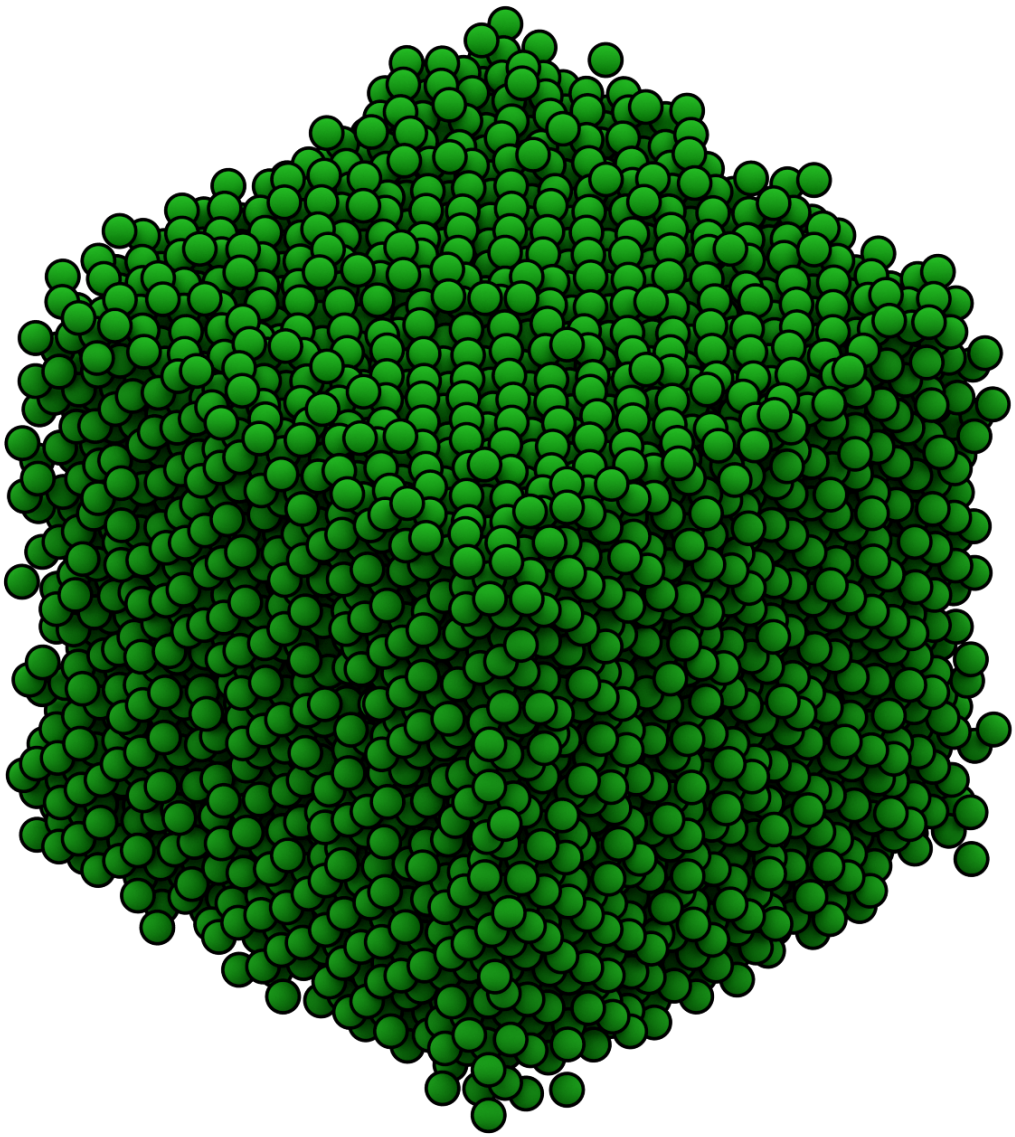
(continued from previous page)

```
                            light_samples=16,
                            w=1380,
                            h=1380)
PIL.Image.fromarray(out[:], mode='RGBA').save('cuboid-hires.png')
```

Author

Joshua A. Anderson

3.2 Spheres



Spheres example script:

- Geometry: *geometry.Sphere*: *radius* = 0.5, *outline_width* = 0.1
  - *material*: *roughness* = 0.8, *specular* = 0.2, *metal* = 0, *spec_trans* = 0
  - *outline_material*: *solid* = 1, *color* = (0,0,0)
  - positions: output of a HOOMD simulation
- Lighting: *light.cloudy*
- Rendered with: *tracer.Path*: *samples* = 256, *light_samples* = 16

Source code

```
"""Sphere example scene."""

import fresnel
import numpy
import PIL
import sys

data = numpy.load('spheres.npz')

scene = fresnel.Scene()
scene.lights = fresnel.light.cloudy()

geometry = fresnel.geometry.Sphere(scene,
                                   position=data['position'],
                                   radius=0.5,
                                   outline_width=0.1)

geometry.material = fresnel.material.Material(color=fresnel.color.linear(
    [0.1, 0.8, 0.1]),
                                              roughness=0.8,
                                              specular=0.2)

scene.camera = fresnel.camera.Orthographic.fit(scene)
out = fresnel.pathtrace(scene, samples=64, light_samples=32, w=580, h=580)
PIL.Image.fromarray(out[:], mode='RGBA').save('sphere.png')

if len(sys.argv) > 1 and sys.argv[1] == 'hires':
    out = fresnel.pathtrace(scene,
                            samples=256,
                            light_samples=16,
                            w=1380,
                            h=1380)
    PIL.Image.fromarray(out[:], mode='RGBA').save('sphere-hires.png')
```

Author

Joshua A. Anderson

3.3 Gumballs

Spheres rendered as gumballs made from a Monte Carlo simulation with HOOMD.

- Geometry: `geometry.Sphere`: *radius* = 0.5
  - `material`: *primitive_color_mix* = 1.0, *roughness* = 0.2, *specular* = 0.8
  - positions: output of a HOOMD simulation

  - colors: randomly assigned from a set of gumball colors
- Lighting: *light.lightbox* with an additional light
- Rendered with: *tracer.Path*: *samples* = 256, *light_samples* = 64 on the GPU

Source code

```
"""Gumballs example scene."""

import fresnel
import numpy as np
from matplotlib.colors import LinearSegmentedColormap
import PIL
import sys

# First, we create a color map for gumballs.
colors = [
    '#e56d60',
    '#ee9944',
    '#716e80',
    '#eadecd',
    '#cec746',
    '#c0443f',
    '#734d56',
    '#5d5f7b',
    '#ecb642',
    '#8a9441',
]
cmap = LinearSegmentedColormap.from_list(name='gumball',
                                         colors=colors,
                                         N=len(colors))

# Next, we gather information needed for the geometry.
position = np.load('gumballs.npz')['position']
np.random.seed(123)
color = fresnel.color.linear(cmap(np.random.rand(len(position))))
material = fresnel.material.Material(
    primitive_color_mix=1.0,
    roughness=0.2,
    specular=0.8,
)

# We create a fresnel scene and its geometry.
scene = fresnel.Scene()

geometry = fresnel.geometry.Sphere(
    scene,
    position=position,
    radius=0.5,
    color=color,
    material=material,
)

# Configure camera and lighting.
scene.camera = fresnel.camera.Perspective(position=(0, 0, 25),
                                          look_at=(0, 0, 0),
```

(continues on next page)

(continued from previous page)

```
                                        up=(0, 1, 0),
                                        focal_length=0.5,
                                        f_stop=0.25)
scene.camera.focus_on = (0, 0, 5.6)
scene.lights = fresnel.light.lightbox()
scene.lights.append(
    fresnel.light.Light(direction=(0.3, -0.3, 1),
                        color=(0.5, 0.5, 0.5),
                        theta=np.pi))

# Execute rendering.
out = fresnel.pathtrace(scene, w=600, h=600, samples=128, light_samples=64)
PIL.Image.fromarray(out[:], mode='RGBA').save('gumballs.png')

if len(sys.argv) > 1 and sys.argv[1] == 'hires':
    out = fresnel.pathtrace(scene,
                            w=1500,
                            h=1500,
                            samples=256,
                            light_samples=64)
    PIL.Image.fromarray(out[:], mode='RGBA').save('gumballs-hires.png')
```

Author

Bradley Dice

CHAPTER

FOUR

INSTALLATION

Fresnel binaries are available in the glotzerlab-software Docker/Singularity images and in packages on conda-forge. You can also compile **fresnel** from source.

4.1 Binaries

4.1.1 Anaconda package

Fresnel is available on conda-forge. To install, first download and install miniconda. Then add the `conda-forge` channel and install **fresnel**:

```
$ conda config --add channels conda-forge
$ conda install fresnel
```

jupyter and **matplotlib** are required to execute the fresnel example notebooks:

```
$ conda install jupyter matplotlib
```

Note: The **fresnel** package on `conda-forge` does not support GPUs

4.1.2 Singularity / Docker images

See the glotzerlab-software documentation for container usage information and cluster specific instructions.

4.2 Compile from source

4.2.1 Obtain the source

Download source releases directly from the web: https://glotzerlab.engin.umich.edu/downloads/fresnel:

```
$ curl -O https://glotzerlab.engin.umich.edu/downloads/fresnel/fresnel-v0.13.1.tar.gz
```

Or, clone using git:

```
$ git clone --recursive  https://github.com/glotzerlab/fresnel
```

Fresnel uses git submodules. Either clone with the `--recursive` option, or execute `git submodule update --init` to fetch the submodules.

4.2.2 Configure a virtual environment

When using a shared Python installation, create a virtual environment where you can install **fresnel**:

```
$ python3 -m venv /path/to/virtual/environment --system-site-packages
```

Activate the environment before configuring and before executing **fresnel** scripts:

```
$ source /path/to/virtual/environment/bin/activate
```

Tell CMake to search in the virtual environment first:

```
$ export CMAKE_PREFIX_PATH=/path/to/virtual/environment
```

Note: Other types of virtual environments (such as *conda*) may work, but are not thoroughly tested.

4.2.3 Install Prerequisites

fresnel requires:

- C++14 capable compiler
- CMake >= 3.8
- pybind11 >= 2.2
- Python >= 3.6
- numpy
- Qhull >= 2015.2
- For CPU execution (required when `ENABLE_EMBREE=ON`):
  - Intel TBB >= 4.3.20150611
  - Intel Embree >= 3.0.0
- For GPU execution (required when `ENABLE_OPTIX=ON`):
  - OptiX == 6.0
  - CUDA >= 10

`ENABLE_EMBREE` (*defaults ON*) and `ENABLE_OPTIX` (*defaults OFF*) are orthogonal settings, either or both may be enabled.

Additional packages may be needed:

- pyside2
  - Required t.o enable interactive widgets. (runtime)
- pillow
  - Required to display rendered output in Jupyter notebooks automatically. (runtime)
  - Required to execute unit tests.

- pytest
  - Required to execute unit tests.
- sphinx, sphinx_rtd_theme, and nbspinx
  - Required to build the user documentation.
- doxygen
  - Requited to build developer documentation.

Install these tools with your system or virtual environment package manager. **fresnel** developers have had success with `pacman` (arch linux), `apt-get` (ubuntu), Homebrew (macOS), and MacPorts (macOS):

```
$ your-package-manager install cmake doxygen embree pybind11 python python-pillow␣
↪python-pytest python-sphinx python-sphinx_rtd_theme python-nbsphinx intell-tbb qhull
```

Typical HPC cluster environments provide python, numpy, and cmake via a module system:

```
$ module load gcc python cmake
```

Note: Packages may be named differently, check your system's package list. Install any `-dev` packages as needed.

Tip: You can install numpy and other python packages into your virtual environment:

```
python3 -m pip install numpy
```

4.2.4 Compile

Configure with **cmake** and compile with **make**:

```
$ cd /path/to/fresnel
$ mkdir build
$ cd build
$ cmake ../
$ make install -j10
```

By default, **fresnel** builds the Embree (CPU) backend. Pass `-DENABLE_OPTIX=ON` to **cmake** to enable the GPU accelerated OptiX backend.

4.2.5 Run tests

To run tests, execute `pytest` in the build directory or in an environment where fresnel is installed to run all tests.

```
$ pytest --pyargs fresnel
```

4.2.6 Build user documentation

Build the user documentation with **sphinx**:

```
$ cd /path/to/fresnel
$ cd doc
$ make html
$ open build/html/index.html
```

4.2.7 Build C++ Documentation

To build the developer documentation, execute `doxygen` in the repository root. It will write HTML output in `devdoc/html/index.html`.

CHAPTER

FIVE

CHANGE LOG

fresnel releases follow semantic versioning.

5.1 v0.x

5.1.1 v0.13.1 (2021-03-11)

Fixed

- Add missing `version` module

5.1.2 v0.13.0 (2021-03-11)

Added

- Perspective camera.
- Depth of field effect.

Changed

- Reduce latency in `interact.SceneView` while rotating the view.
- Improve user experience with mouse rotations in `interact.SceneView`.
- [breaking] - Moved `camera.orthographic` to `camera.Orthographic`.
- [breaking] - Moved `camera.fit` to `camera.Orthographic.fit`.

Removed

- [breaking] - Removed "auto" camera in `Scene`. Use `camera.Orthographic.fit`

5.1.3 v0.12.0 (2020-02-27)

Added

- `preview` and `tracer.Preview` accept a boolean flag `anti_alias` to enable or disable anti-aliasing.

Changed

- `preview` and `tracer.Preview` enable anti-alisasing by default.
- Python, Cython, and C code must follow strict style guidelines.
- Renamed `util.array` to `util.Array`

- Renamed `util.image_array` to `util.ImageArray`
- Converted `interact.SceneView.setScene` to a property: `scene`

Removed

- `preview` and `tracer.Preview` no longer accept the `aa_level` argument - use `anti_alias`.

5.1.4 v0.11.0 (2019-10-30)

Added

- Added box geometry convenience class `Box`.

Removed

- Support for **Python** 3.5.

Fixed

- Compile on systems where `libqhullcpp.a` is missing or broken.
- Find **Embree** headers when they are not in the same path as **TBB**.

5.1.5 v0.10.1 (2019-09-05)

Fixed

- Restore missing examples on readthedocs.

5.1.6 v0.10.0 (2019-08-19)

Changed

- **CMake** >= 3.8 is required at build time.
- **pybind11** >= 2.2 is required at build time.
- **qhull** >= 2015 is required.
- install to the **Python** `site-packages` directory by default.
- **CI** tests execute on Microsoft Azure Pipelines.

Fixed

- Improved installation documentation.

5.1.7 v0.9.0 (2019-04-30)

- Added support for linearizing colors of shape (4,).
- Improve examples.

5.1.8 v0.8.0 (2019-03-05)

- Documentation improvements.
- Add `geometry.Polygon`: Simple and/or rounded polygons in the *z=0* plane.
- API breaking changes:
  - Remove: `geometry.Prism`

5.1.9 v0.7.1 (2019-02-05)

- Fix **conda-forge** build on mac

5.1.10 v0.7.0 (2019-02-05)

- Add `util.convex_polyhedron_from_vertices`: compute convex polyhedron plane origins and normals given a set of vertices
- Improve documentation
- Add `interact.SceneView`: **pyside2** widget for interactively rendering scenes with path tracing
- Add `geometry.Mesh`: Arbitrary triangular mesh geometry, instanced with N positions and orientations
- **fresnel** development is now hosted on github: https://github.com/glotzerlab/fresnel/
- Improve `light.lightbox` lighting setup
- API breaking changes:
  - `geometry.ConvexPolyhedron` arguments changed. It now accepts polyhedron information as a dictionary.

5.1.11 v0.6.0 (2018-07-06)

- Implement `tracer.Path` on the GPU.
- Implement `ConvexPolyhedron` geometry on the GPU.
- Improve path tracer performance with Russian roulette termination.
- Compile warning-free.
- Fix sphere intersection test bugs on the GPU.
- `tracer.Path` now correctly starts sampling over when resized.
- Wrap C++ code with **pybind** 2.2
- Make documentation available on readthedocs: http://fresnel.readthedocs.io
- Fresnel is now available on **conda-forge**: https://anaconda.org/conda-forge/fresnel
- embree >= 3.0 is now required for CPU support
- Improve documentation

5.1.12 v0.5.0 (2017-07-27)

- Add new lighting setups
  - `lightbox`
  - `cloudy`
  - `ring`
- Adjust brightness of lights in existing setups
- Remove `clearcoat` material parameter
- Add `spec_trans` material parameter
- Add `Path` tracer to render scenes with indirect lighting, reflections, and transparency (*CPU-only*)
- Add `ConvexPolyhedron` geometry (*CPU-only*, *beta API, subject to change*)
- Add `fresnel.preview` function to easily generate `Preview` traced renders with one line
- Add `fresnel.pathtrace` function to easily generate `Path` traced renders with one line
- Add anti-aliasing (always on for the `Path` tracer, `set aa_level` > 0 to enable for `Preview`)
- API breaking changes:
  - `render` no longer exists. Use `preview` or `pathtrace`.
  - `tracer.Direct` is now `tracer.Preview`.

CPU-only features will be implemented on the GPU in a future release.

5.1.13 v0.4.0 (2017-04-03)

- Enforce requirement: Embree >= 2.10.0
- Enforce requirement Pybind =1.8.1
- Enforce requirement TBB >= 4.3
- Rewrite camera API, add camera.fit to fit the scene
- scenes default to an automatic fit camera
- Implement area lights, add default lighting setups
- `Scene` now supports up to 4 lights, specified in camera space
- Implement Disney's principled BRDF
- `Tracer.histogram` computes a histogram of the rendered image
- `Tracer.enable_highlight_warning` highlights overexposed pixels with a given warning color
- `Device.available_modes` lists the available execution modes
- `Device.available_gpus` lists the available GPUs
- `Device` can now be limited to *n* GPUs
- API breaking changes:
  - `camera.Orthographic` is now `camera.orthographic`
  - `Device` now takes the argument *n* instead of *limit*

  - `Scene` no longer has a `light_direction` member

5.1.14 v0.3.0 (2017-03-09)

- Suppress "cannot import name" messages
- Support Nx3 and Nx4 inputs to `color.linear`

5.1.15 v0.2.0 (2017-03-03)

- Parallel rendering on the CPU
- Fix PTX file installation
- Fix python 2.7 support
- Unit tests
- Fix bug in sphere rendering on GPU

5.1.16 v0.1.0 (2017-02-02)

- Prototype API
- Sphere geometry
- Prism geometry
- outline materials
- diffuse materials
- Direct tracer

CHAPTER

SIX

USER COMMUNITY

6.1 fresnel-users mailing list

Subscribe to the fresnel-users mailing list to receive release announcements, post questions for advice on using the software, and discuss potential new features.

6.2 Issue tracker

File bug reports on fresnel's issue tracker.

6.3 Contribute

fresnel is an open source project. Contributions are accepted via pull request to fresnel's github repository. Please review `CONTRIBUTING.MD` in the repository before starting development. You are encouraged to discuss your proposed contribution with the **fresnel** user and developer community who can help you design your contribution to fit smoothly into the existing ecosystem.

CHAPTER

SEVEN

INTRODUCTION

Fresnel is a python library that can ray trace publication quality images in real time. It provides a simple python API to define a **scene** consisting of any number of **geometry** primitives and **render** it to an output image.

To start, import the `fresnel` python module.

[1]:
```
import fresnel
```

7.1 Define a scene

A **Scene** defines a coordinate system, the **camera** view, the **light sources**, and contains a number of **geometry** primitives. Create a new **Scene** class instance. Scenes come with a default automatic camera that fits the geometry and a default set of lights.

[2]:
```
scene = fresnel.Scene()
```

7.2 Add geometry to the scene

A **Scene** may consist of any number of **geometry** objects. Each **geometry** object consists of N primitives of the same type, and a **material** that describes how the primitives interact with light sources. Create 8 spheres with radius 1.0.

[3]:
```
geometry = fresnel.geometry.Sphere(scene, N=8, radius=1.0)
```

Geometry objects have a number of per-primitive attributes. These are exposed with an interface compatible with **numpy** arrays, and can copy data from **numpy** arrays efficiently. Set the positions of the spheres:

[4]:
```
geometry.position[:] = [[1,1,1],
                        [1,1,-1],
                        [1,-1,1],
                        [1,-1,-1],
                        [-1,1,1],
                        [-1,1,-1],
                        [-1,-1,1],
                        [-1,-1,-1]]
```

Set the **material** of the geometry object to a rough blue surface:

[5]:
```
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.
↪9]),
                                              roughness=0.8)
```

7.3 Set the camera

The **camera** defines the view of the scene. **fresnel** can auto-fit a camera to the **scene's** geometry:

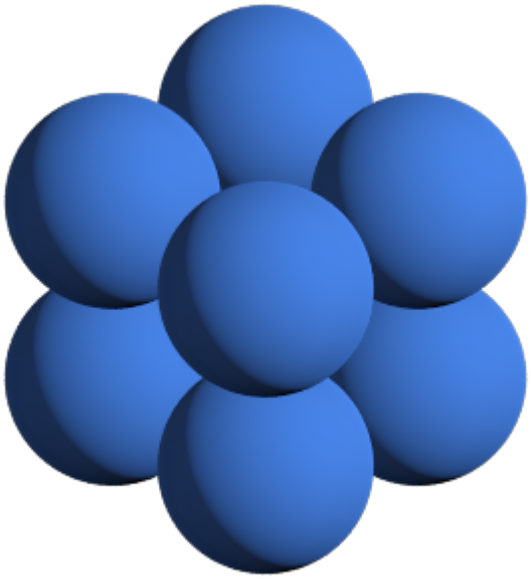
[6]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

7.4 Render the scene

preview quickly renders the scene from the view point of the camera. Anti-aliasing is on by default to smooth edges in the image.

[7]:
```
fresnel.preview(scene)
```

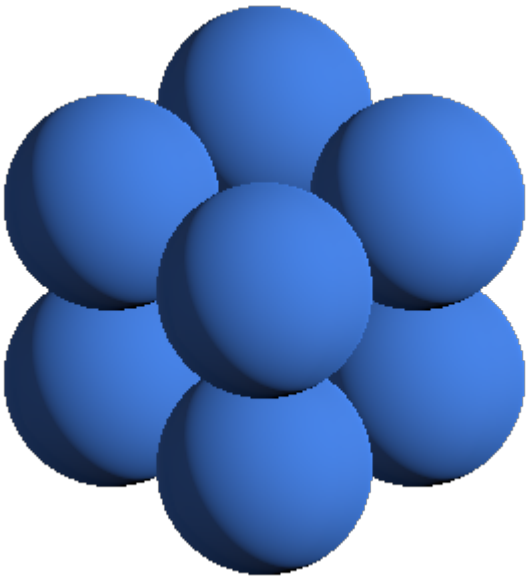
[7]:



Disable *anti-aliasing* if you desire a quicker render.

[8]:
```
fresnel.preview(scene, anti_alias=False)
```

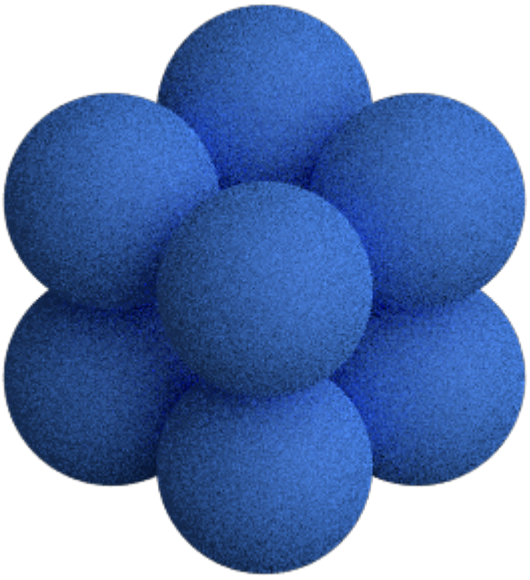
[8]:



preview only applies direct lighting. Use **pathtrace** to account for indirect lighting. (anti-aliasing is always enabled when path tracing).

[9]:
```
fresnel.pathtrace(scene)
```

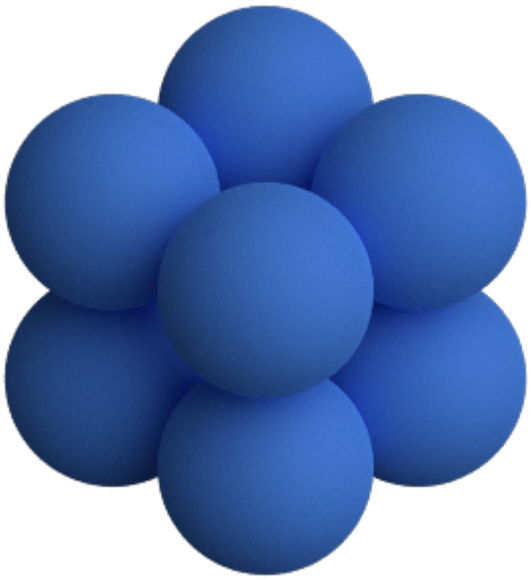
[9]:



The resulting image is noisy. Increase the number of *light samples* to obtain a clean image.

[10]:
```
fresnel.pathtrace(scene, light_samples=40)
```

[10]:



7.5 Save output

preview and *pathtrace* return output buffers that can be used like `HxWx4` RGBA **numpy** arrays. You can pass this standard format on to other python libraries that work images (e.g. matplotlib).

[11]:
```
out = fresnel.preview(scene)
print(out[:].shape)
print(out[:].dtype)
```

```
(370, 600, 4)
uint8
```

Use Pillow to save the rendered output to a png file with transparency.

[12]:
```
import PIL
```

[13]:
```
image = PIL.Image.fromarray(out[:], mode='RGBA')
image.save('output.png')
```

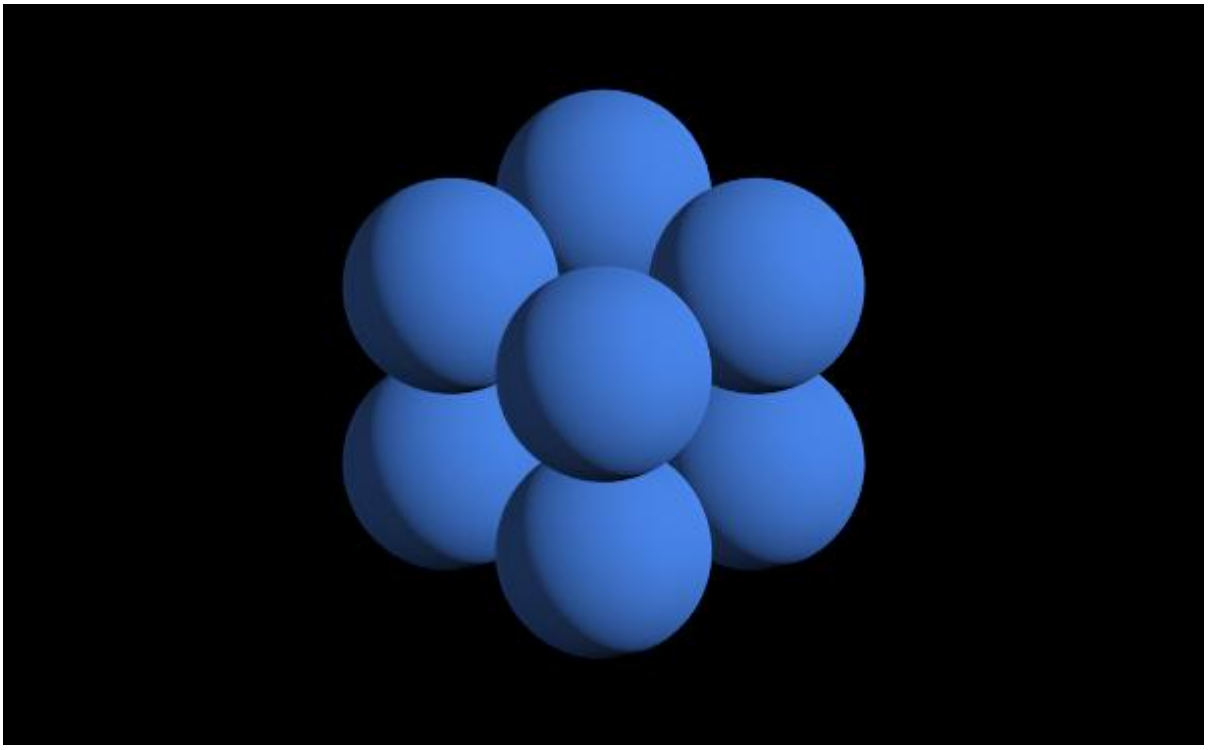
To save a JPEG, create an RGB image. This ignores the alpha channel, so the scene background color will show.

[14]:
```
image = PIL.Image.fromarray(out[:,:,0:3], mode='RGB')
image.save('output.jpeg')
```

This is what `output.jpeg` looks like (the default background color is black):

[15]:
```
import IPython.display
IPython.display.Image('output.jpeg')
```

[15]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

EIGHT

PRIMITIVE PROPERTIES

Each **geometry** type specifies a number of per-primitive properties. For example, the **Sphere** geometry has per-primitive *position*, *radius*, and *color*.

[1]:
```
import fresnel
scene = fresnel.Scene()
```

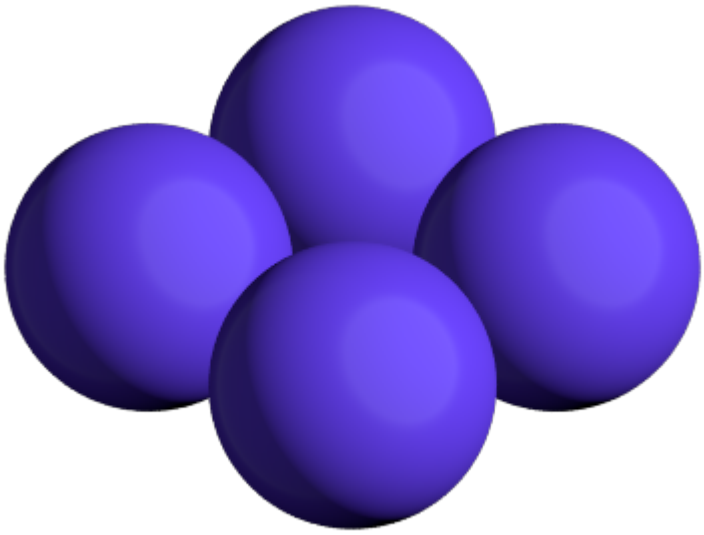
8.1 Setting properties when creating the geometry

Any of the properties may be set when the **geometry** is created, or they may be left as default values.

[2]:
```
geometry = fresnel.geometry.Sphere(scene,
                                   position = [[1,0,1],
                                               [1,0,-1],
                                               [-1,0,1],
                                               [-1,0,-1]],
                                   radius=1.0,
                                   material = fresnel.material.Material(color=fresnel.
↪color.linear([0.42,0.267,1]))
                                   # per-primitive color left default
                                   )
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

[3]:
```
fresnel.preview(scene)
```

[3]:



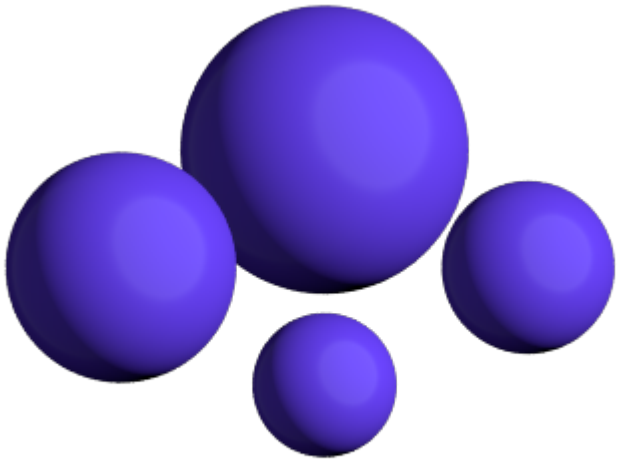
8.2 Changing properties after creation

Access the per-primitive properties as if they were **numpy** arrays. The *radius* property for the **Sphere geometry** sets the radius of each primitive.

[4]:
```
geometry.radius[:] = [0.5, 0.6, 0.8, 1.0]
```

[5]:
```
fresnel.preview(scene)
```

[5]:

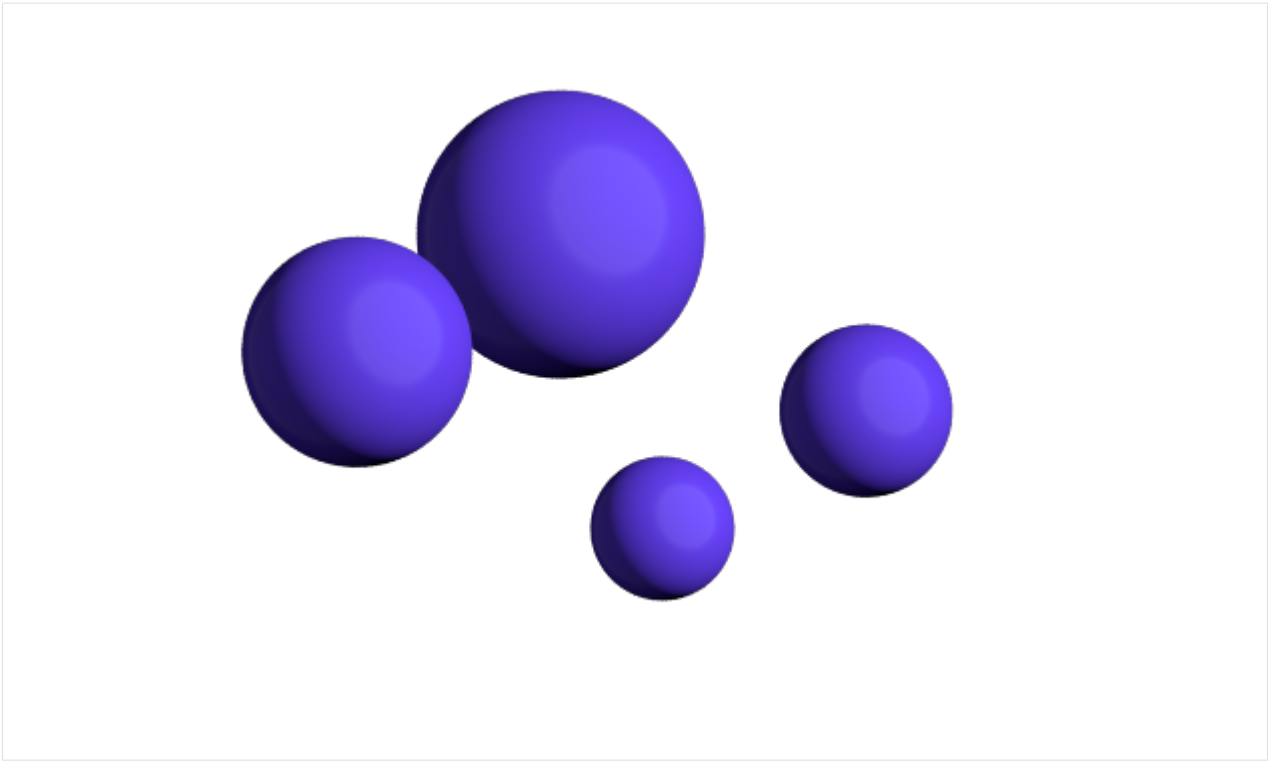


The *position* property sets the position of each sphere in the scene's coordinate system.

[6]:
```
geometry.position[:] = [[1.5,0,1],
                        [1.5,0,-1],
                        [-1.5,0,1],
                        [-1.5,0,-1]]
```

[7]:
```
fresnel.preview(scene)
```

[7]:

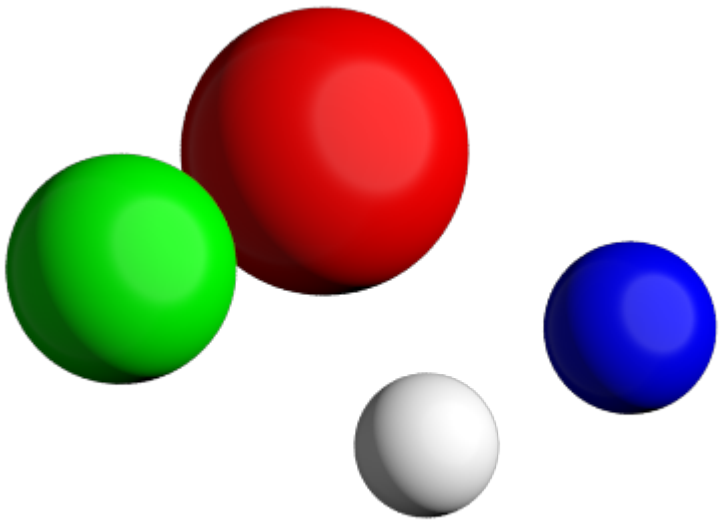


The *color* property sets a per primitive color. The geometry **material color** and the **primitive color** are mixed with fraction **primitive_color_mix**. A value of 1.0 selects the primitive color, 0.0 selects the *material* color and values in between mix the colors.

[8]:
```
geometry.material.primitive_color_mix = 1.0
geometry.color[:] = fresnel.color.linear([[1,1,1], [0,0,1], [0,1,0], [1,0,0]])
```

[9]:
```
fresnel.preview(scene)
```

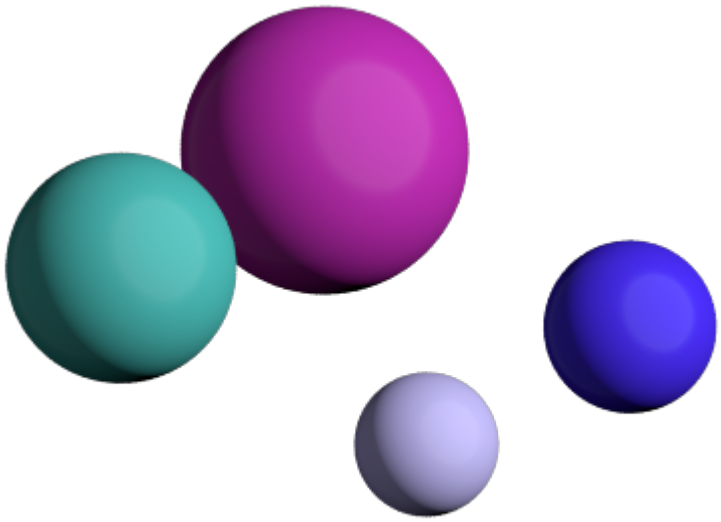
[9]:



[10]:
```
geometry.material.primitive_color_mix = 0.5
```

[11]:
```
fresnel.preview(scene)
```

[11]:



8.3 Reading primitive properties

Primitive properties may be read as well as written.

[12]:
```
geometry.radius[:]
```

[12]:
```
array([0.5, 0.6, 0.8, 1. ], dtype=float32)
```

[13]:
```
geometry.position[:]
```

[13]:
```
array([[ 1.5,  0. ,  1. ],
       [ 1.5,  0. , -1. ],
       [-1.5,  0. ,  1. ],
       [-1.5,  0. , -1. ]], dtype=float32)
```

[14]:
```
geometry.color[:]
```

[14]:
```
array([[1., 1., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.]], dtype=float32)
```

8.4 Common errors

Primitive properties may be accessed like **numpy** arrays, but they may not be assigned directly.

[15]:
```
geometry.radius = 1.0
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-15-020bd663bace> in <module>
----> 1 geometry.radius = 1.0

AttributeError: can't set attribute
```

This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

NINE

MATERIAL PROPERTIES

Each **geometry** has an associated **material**. The **material** is a set of parameters that defines how light interacts with the **geometry**. Here is a test scene to demonstrate these properties.

[1]:
```
import fresnel
import math
device = fresnel.Device()
scene = fresnel.Scene(device)
position = []
for k in range(5):
    for i in range(5):
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

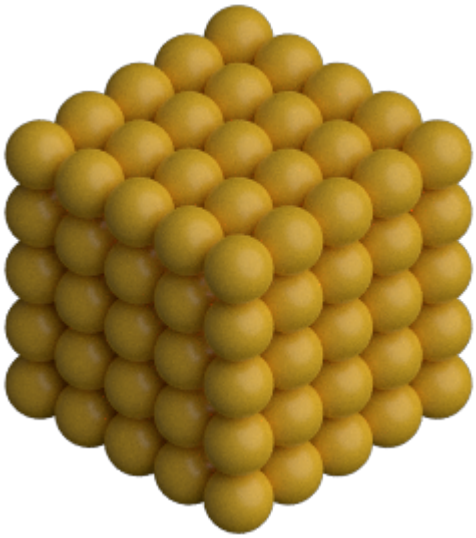
9.1 Material color

The **color** of a material sets its base color. Default material parameters set a primarily diffuse material with light specular highlights.

[2]:
```
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.9,0.714,0.
↪169]))
```

[3]:
```
fresnel.pathtrace(scene, w=300, h=300, light_samples=40)
```

[3]:



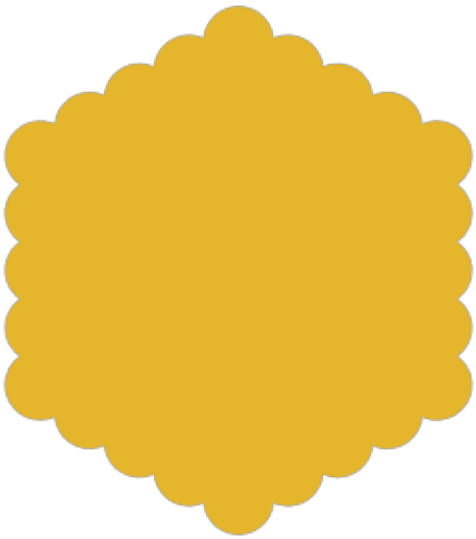
9.2 Solid color materials

Set the **solid** parameter to 1.0 to disable material interaction with light. A **solid** material has no shading applied and always displays as **color**.

[4]:
```
geometry.material.solid = 1.0
```

[5]:
```
fresnel.preview(scene, w=300, h=300)
```

[5]:



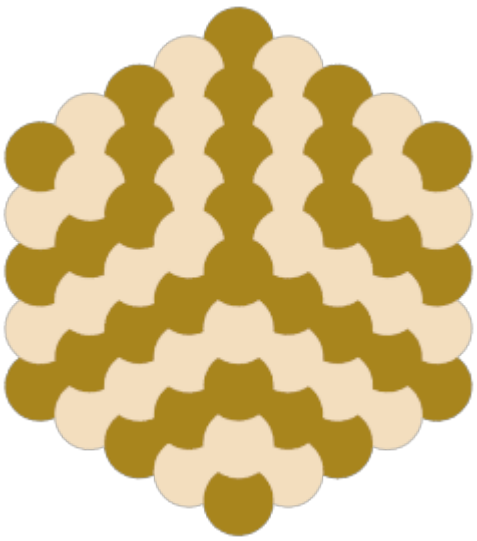
9.3 Geometry / primitive color mixing

Set **primitive_color_mix** to any value in the range 0.0 to 1.0 to control the amount that the per-primitive colors mix with the geometry color.

[6]:
```
geometry.material.primitive_color_mix = 0.5
geometry.color[::2] = fresnel.color.linear([0,0,0])
geometry.color[1::2] = fresnel.color.linear([1,1,1])
```

[7]:
```
fresnel.preview(scene, w=300, h=300)
```

[7]:

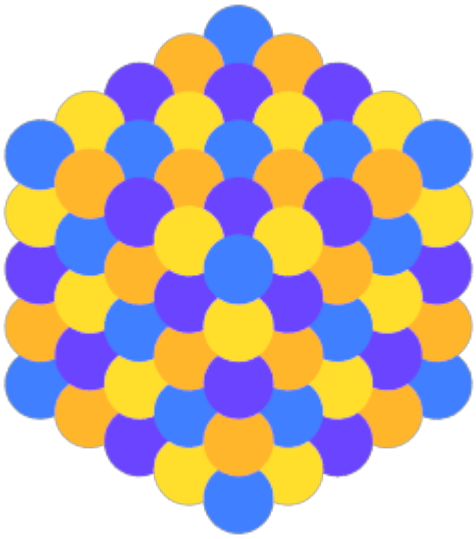


Typical use cases utilize values of either 0.0 (force a single color defined by the material) or 1.0 (force the per primitive color.)

[8]:
```
geometry.material.primitive_color_mix = 1.0

geometry.color[::4] = fresnel.color.linear([0.25,0.5,1])
geometry.color[1::4] = fresnel.color.linear([1,0.714,0.169])
geometry.color[2::4] = fresnel.color.linear([0.42,0.267,1])
geometry.color[3::4] = fresnel.color.linear([1,0.874,0.169])
```

[9]:
```
fresnel.preview(scene, w=300, h=300)
```

[9]:

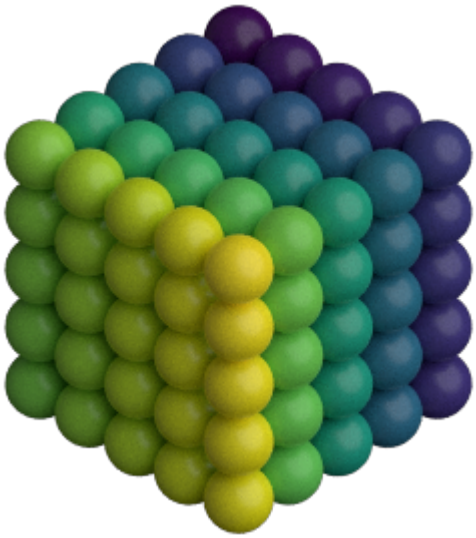


To use a **matplotlib** color map, pass the output of the color map to `fresnel.color.linear` so the output colors appear as intended.

[10]:
```
import matplotlib, matplotlib.cm
import numpy
geometry.material.solid = 0.0
mapper = matplotlib.cm.ScalarMappable(norm = matplotlib.colors.Normalize(vmin=0,␣
↪vmax=1, clip=True),
                                      cmap = matplotlib.cm.get_cmap(name='viridis'))

v = numpy.linspace(0,1,len(position))
geometry.color[:] = fresnel.color.linear(mapper.to_rgba(v))
```

[11]:
```
fresnel.pathtrace(scene, w=300, h=300, light_samples=40)
```

[11]:



9.4 All properties

Materials have a number of intuitive properties. All are defined in a nominal range from 0 to 1, though some values can be pushed past 1 for extremely strong effects.

- roughness - Set the roughness of the material. Recommend values >= 0.1.
- specular - Control the strength of the specular highlights
- metal - 0: dielectric materials (plastic, glass, etc...). 1: pure metals.
- spec_trans - Set the fraction of light that passes through the material.

Here are some examples of different material parameters.

[12]:
```
scene2 = fresnel.Scene(device)
spheres = []
for i in range(11):
    spheres.append(fresnel.geometry.Sphere(scene2, position = (i, 0, 0), radius=0.4))
    spheres[i].material = fresnel.material.Material(color=(.1,.7,.1))

tracer = fresnel.tracer.Path(device=device, w=1000, h=75)

scene2.lights = [fresnel.light.Light(direction=(1,1,-1), color=(0.5, 0.5, 0.5)),
                 fresnel.light.Light(direction=(-1,-1,1), color=(0.5, 0.5, 0.5))]

scene2.camera = fresnel.camera.Orthographic.fit(scene2)
```

9.4.1 Examples

These examples are front lit from the lower left and back lit from the upper right.

Vary roughness in a specular material from 0.1 to 1.1

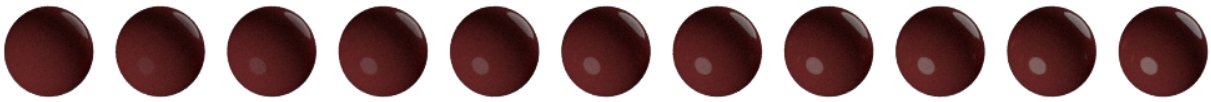
[13]:
```
for i in range(11):
    spheres[i].material.specular = 1.0
    spheres[i].material.roughness = i/10+0.1

tracer.sample(scene2, samples=64, light_samples=40)
```

[13]:

Vary specular from 0 to 1 with constant roughness.

[14]:
```
for i in range(11):
    spheres[i].material.specular = i/10
    spheres[i].material.roughness = 0.1
    spheres[i].material.color=(.7,.1,.1)

tracer.sample(scene2, samples=64, light_samples=40)
```

[14]:



The following examples use cloudy lighting which places a bright hemisphere of light above the scene and a dim hemisphere of light below the scene.

Vary metal from 0 to 1 with a rough material. (metal materials look best when there is other geometry to reflect from the surface)

[15]:
```
for i in range(11):
    spheres[i].material.specular = 1.0
    spheres[i].material.color=(.7,.7,.7)
    spheres[i].material.metal = i/10

scene2.lights = fresnel.light.cloudy()
tracer.sample(scene2, samples=64, light_samples=40)
```

[15]:


Vary spec_trans from 0 to 1 with all other quantities constant.

[16]:
```
for i in range(11):
    spheres[i].material.metal = 0.0
    spheres[i].material.spec_trans = i/10
    spheres[i].material.color=(.1,.1,.7)

tracer.sample(scene2, samples=64, light_samples=40)
```

[16]:

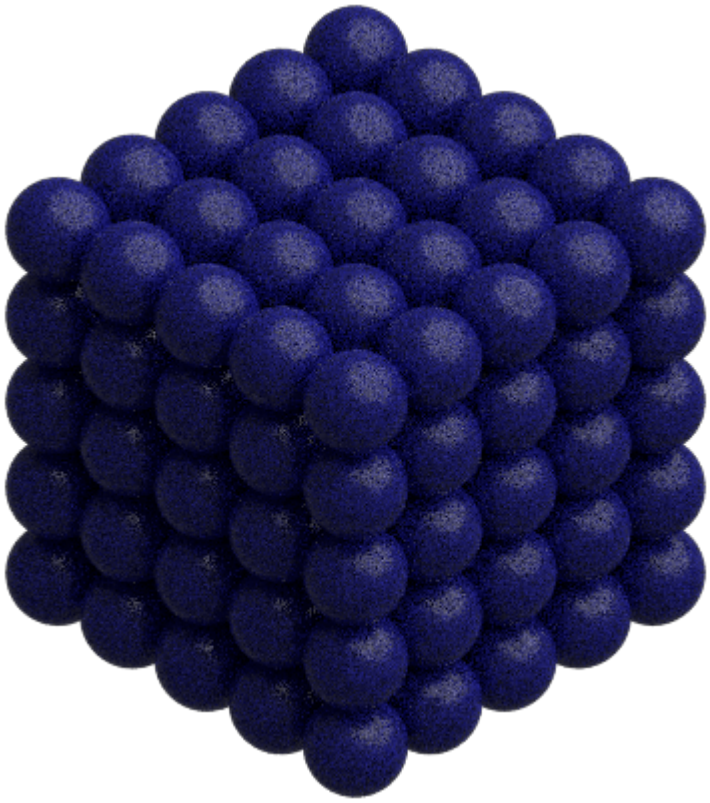

Execute this notebook with **ipywidgets** installed and use the panel below to explore the material parameters and how they react to different lighting angles.

[17]:
```
import ipywidgets

tracer.resize(450,450)

@ipywidgets.interact(color=ipywidgets.ColorPicker(value='#1c1c7f'),
                     primitive_color_mix=ipywidgets.FloatSlider(value=0.0, min=0.0,␣
↪max=1.0, step=0.1, continuous_update=False),
                     roughness=ipywidgets.FloatSlider(value=0.3, min=0.1, max=1.0,␣
↪step=0.1, continuous_update=False),
                     specular=ipywidgets.FloatSlider(value=0.5, min=0.0, max=1.0,␣
↪step=0.1, continuous_update=False),
                     spec_trans=ipywidgets.FloatSlider(value=0.0, min=0.0, max=1.0,␣
↪step=0.1, continuous_update=False),
                     metal=ipywidgets.FloatSlider(value=0, min=0.0, max=1.0, step=1.0,
↪ continuous_update=False),
                     light_theta=ipywidgets.FloatSlider(value=5.5, min=0.0,␣
↪max=2*math.pi, step=0.1, continuous_update=False),
                     light_phi=ipywidgets.FloatSlider(value=0.8, min=0.0, max=math.pi,
↪ step=0.1, continuous_update=False))
def test(color, primitive_color_mix, roughness, specular, spec_trans, metal, light_
↪theta, light_phi):
    r = int(color[1:3], 16)/255;
    g = int(color[3:5], 16)/255;
    b = int(color[5:7], 16)/255;
    scene.lights[0].direction = (math.sin(light_phi)*math.cos(-light_theta),
                                 math.cos(light_phi),
                                 math.sin(light_phi)*math.sin(-light_theta))
```

(continues on next page)

(continued from previous page)

```
    scene.lights[1].theta = math.pi
    geometry.material = fresnel.material.Material(color=fresnel.color.linear([r,g,b]),
                                                  primitive_color_mix=primitive_color_
↪mix,
                                                  roughness=roughness,
                                                  metal=metal,
                                                  specular=specular,
                                                  spec_trans=spec_trans
                                                 )
    return tracer.sample(scene, samples=64, light_samples=1)
```



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

TEN

OUTLINE MATERIALS

Each **geometry** has an associated **outline material** and an **outline width**. The **outline material** has all the same attributes as a normal material, but it is only applied in a thin line around each geometry primitive. The width of that line is the **outline width**.

[1]:
```
import fresnel
import math
scene = fresnel.Scene()
position = []
for k in range(5):
    for i in range(5):
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.color[::4] = fresnel.color.linear([0.25,0.5,1])
geometry.color[1::4] = fresnel.color.linear([1,0.714,0.169])
geometry.color[2::4] = fresnel.color.linear([0.42,0.267,1])
geometry.color[3::4] = fresnel.color.linear([1,0.874,0.169])
geometry.material = fresnel.material.Material(solid=0.0, primitive_color_mix=1.0,␣
↪color=fresnel.color.linear([0,0,0]))
fresnel.light.cloudy();
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

10.1 Enabling outlines

The default **outline width** is 0. Set a non-zero outline width to enable the outlines.

[2]:
```
geometry.outline_width
```

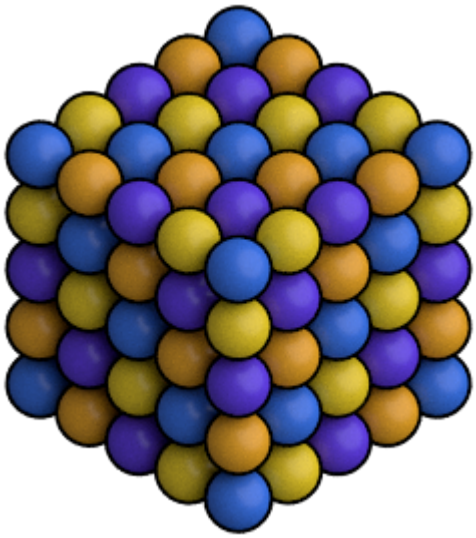
[2]:
```
0.0
```

The **outline width** is in distance units in the same coordinate system as scene. The is *width* units wide perpendicular to the view direction. Outlines enhance the separation between primitives visually. They work well with diffuse and solid colored primitives.

[3]:
```
geometry.outline_width = 0.12
```

[4]:
```
fresnel.pathtrace(scene, w=300, h=300,light_samples=40)
```

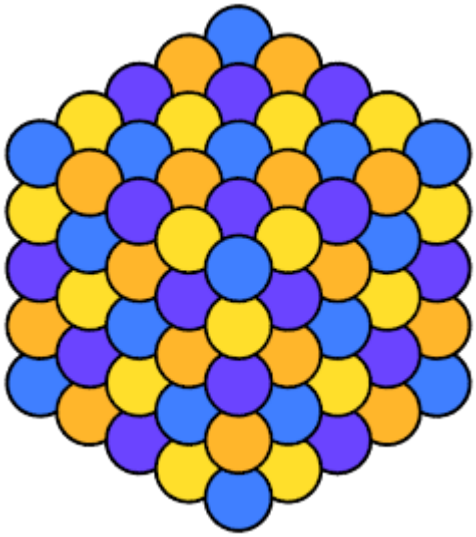
[4]:



[5]:
```
geometry.material.solid = 1.0
```

[6]:
```
fresnel.preview(scene, w=300, h=300)
```

[6]:



10.2 Outline material properties

The default **outline material** is a solid black.

[7]:
```
geometry.outline_material.color
```

[7]:
```
(0.0, 0.0, 0.0)
```

[8]:
```
geometry.outline_material.solid
```

[8]:
```
1.0
```

[9]:
```
geometry.outline_material.primitive_color_mix
```

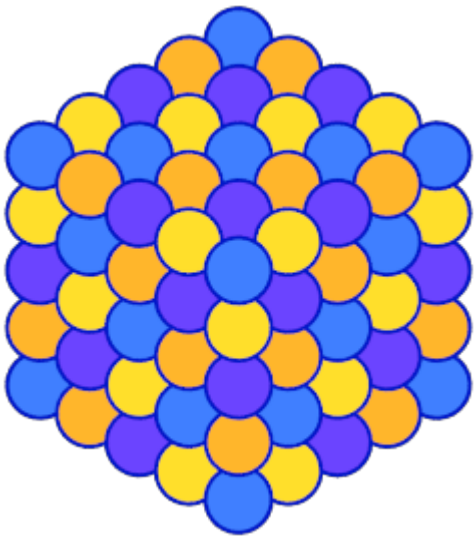
[9]:
```
0.0
```

The **outline material** has all the same properties as a normal material.

[10]:
```
geometry.outline_material.color = fresnel.color.linear(fresnel.color.linear([0.08,0.
↪341,0.9]))
```

[11]:
```
fresnel.preview(scene, w=300, h=300)
```

[11]:

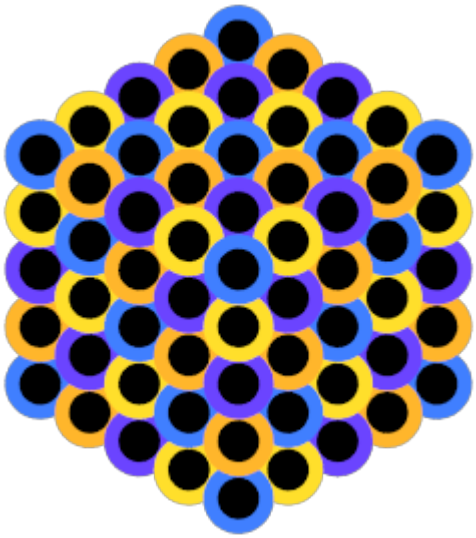


Outlines may be colored by the primitives:

[12]:
```
geometry.material.primitive_color_mix = 0.0
geometry.outline_material.primitive_color_mix = 1.0
geometry.outline_width = 0.4
```

[13]:
```
fresnel.preview(scene, w=300, h=300)
```

[13]:

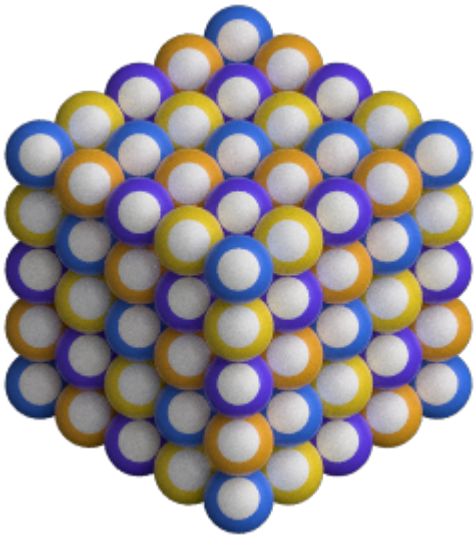


Outlines may have diffuse shading:

[14]:
```
geometry.material.color = fresnel.color.linear([1,1,1])
geometry.material.solid = 0
geometry.outline_material.solid = 0
```

[15]:
```
fresnel.pathtrace(scene, w=300, h=300,light_samples=40)
```

[15]:



Or be metallic:

[16]:
```
geometry.material.color = fresnel.color.linear([0.08,0.341,0.9])

geometry.outline_material.solid = 0
geometry.outline_material.color = [0.95,0.95,0.95]
```

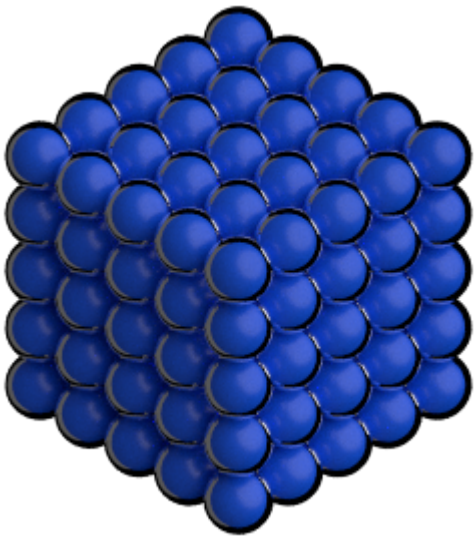
(continues on next page)

(continued from previous page)

```
geometry.outline_material.roughness = 0.1
geometry.outline_material.metal = 1
geometry.outline_material.primitive_color_mix = 0.0
geometry.outline_width = 0.2
```

[17]:
```
fresnel.pathtrace(scene, w=300, h=300,light_samples=40)
```

[17]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

ELEVEN

SCENE PROPERTIES

Each **Scene** has a **background color** and **alpha**, **lights**, and a **camera**.

[1]:
```
import fresnel
import math
scene = fresnel.Scene()
position = []
for i in range(6):
    position.append([2*math.cos(i*2*math.pi / 6), 2*math.sin(i*2*math.pi / 6), 0])

geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.material = fresnel.material.Material(solid=0.0, color=fresnel.color.
↪linear([1,0.874,0.169]))
geometry.outline_width = 0.12
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

11.1 Background color and alpha

The default **background color** is black (0,0,0) and the **background alpha** is 0 (transparent).

[2]:
```
scene.background_color
```

[2]:
```
array([0., 0., 0.], dtype=float32)
```

[3]:
```
scene.background_alpha
```

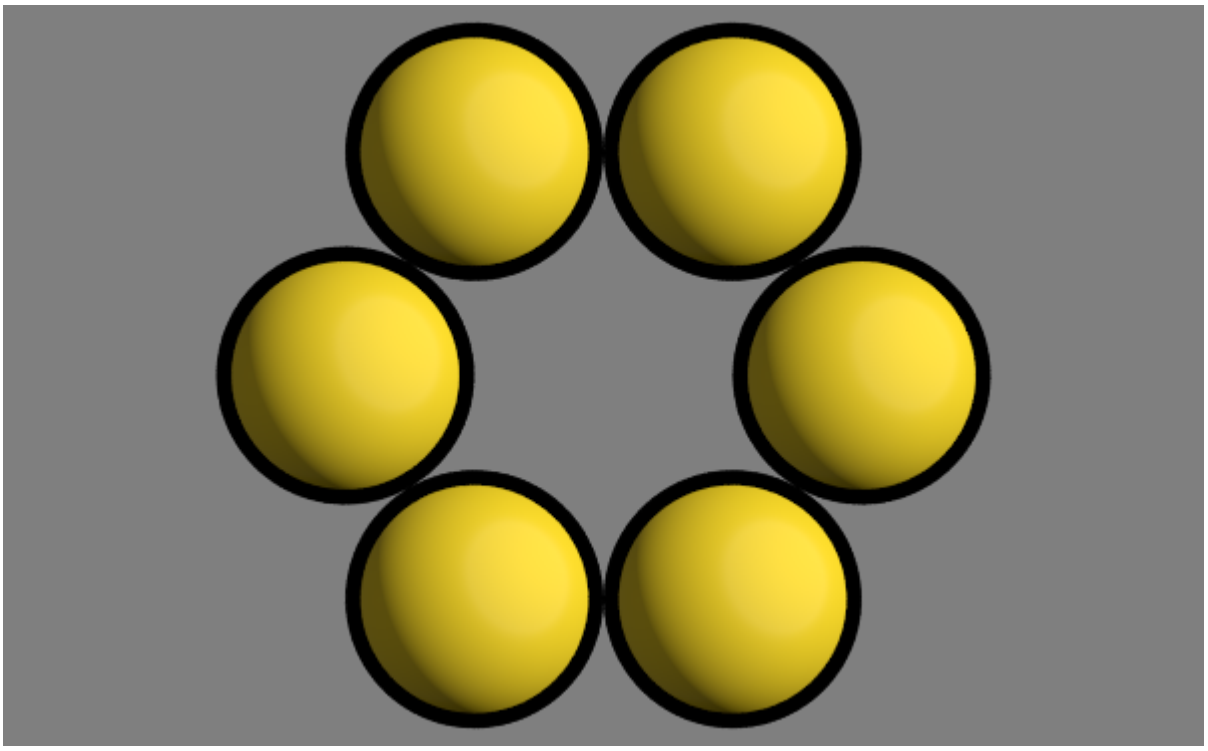
[3]:
```
0.0
```

The background color is applied to any pixel in the output image where no object is present. Change the **background alpha** to only partially transparent:

[4]:
```
scene.background_alpha = 0.5
```

[5]:
```
fresnel.preview(scene)
```

[5]:

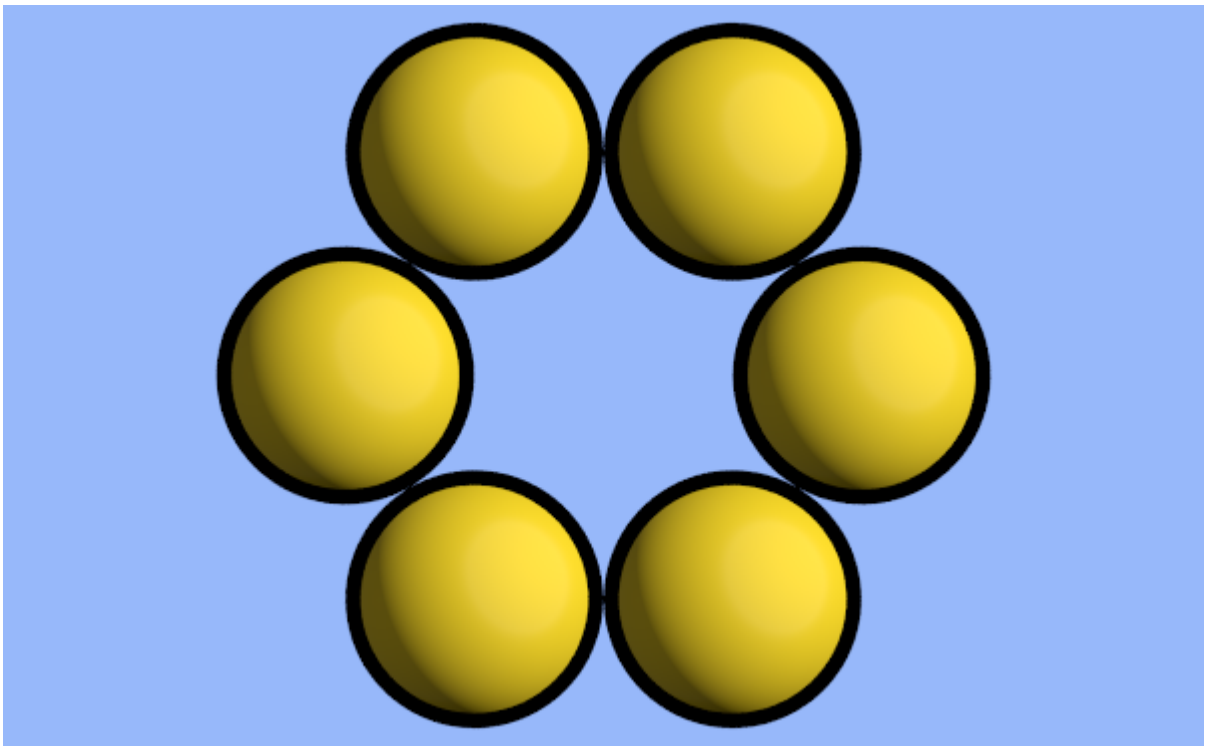


Set a solid background color:

[6]:
```
scene.background_alpha = 1.0
scene.background_color = fresnel.color.linear([0.592, 0.722, 0.98])
```

[7]:
```
fresnel.preview(scene)
```

[7]:



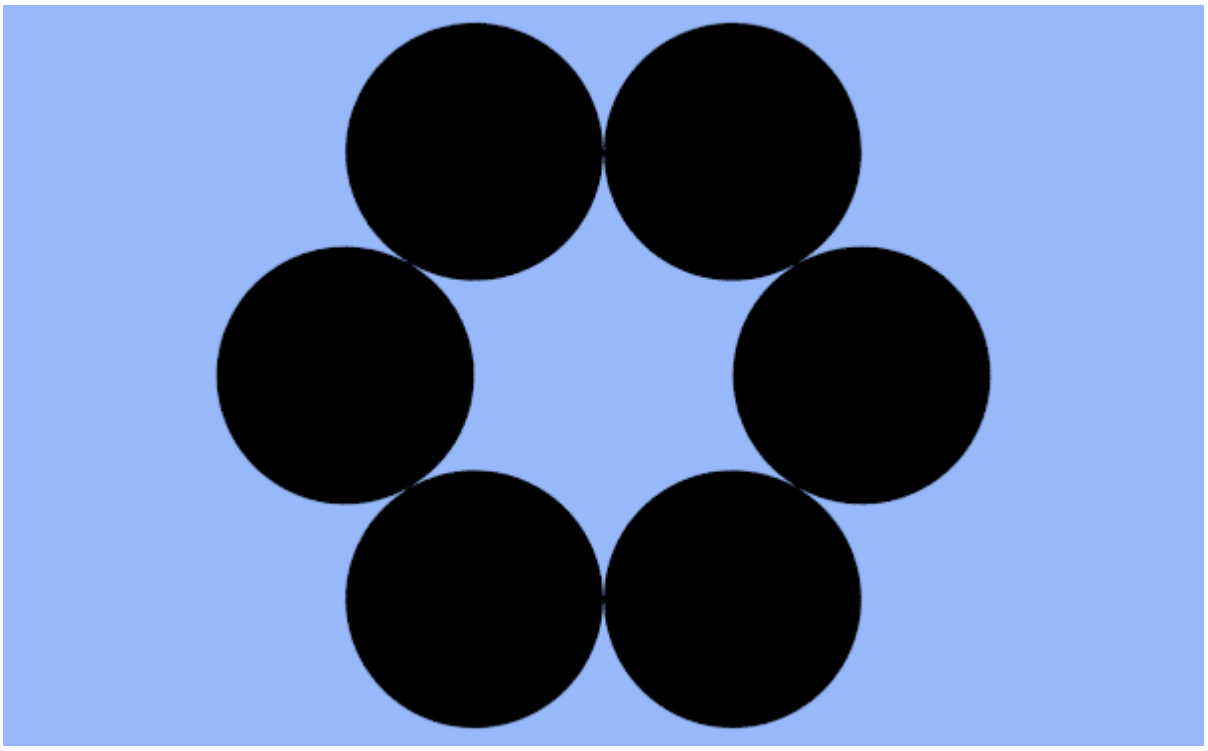
11.2 Light sources

Light sources light the objects in the scene. Without lights, all objects are black.

[8]:
```
scene.lights.clear()
```

[9]:
```
fresnel.preview(scene)
```

[9]:

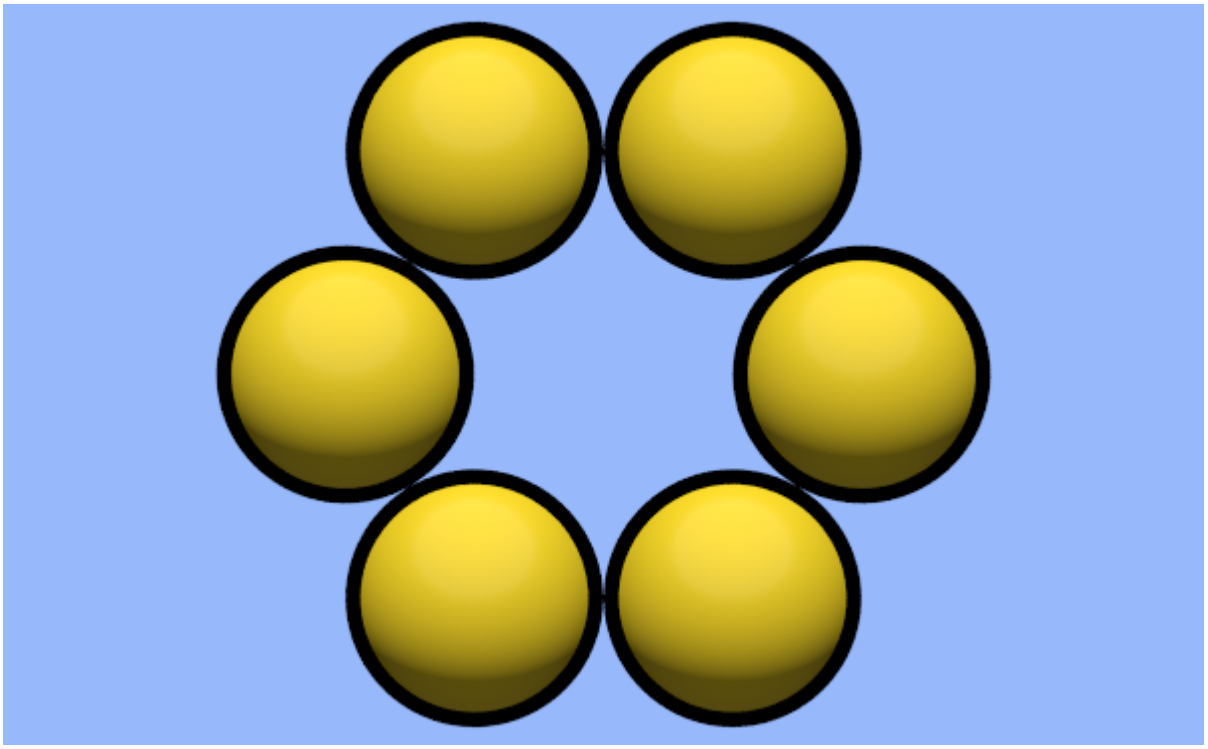


Fresnel defines several standard lighting setups that may be easily applied.

[10]:
```
scene.lights = fresnel.light.butterfly()
```

[11]:
```
fresnel.preview(scene)
```

[11]:

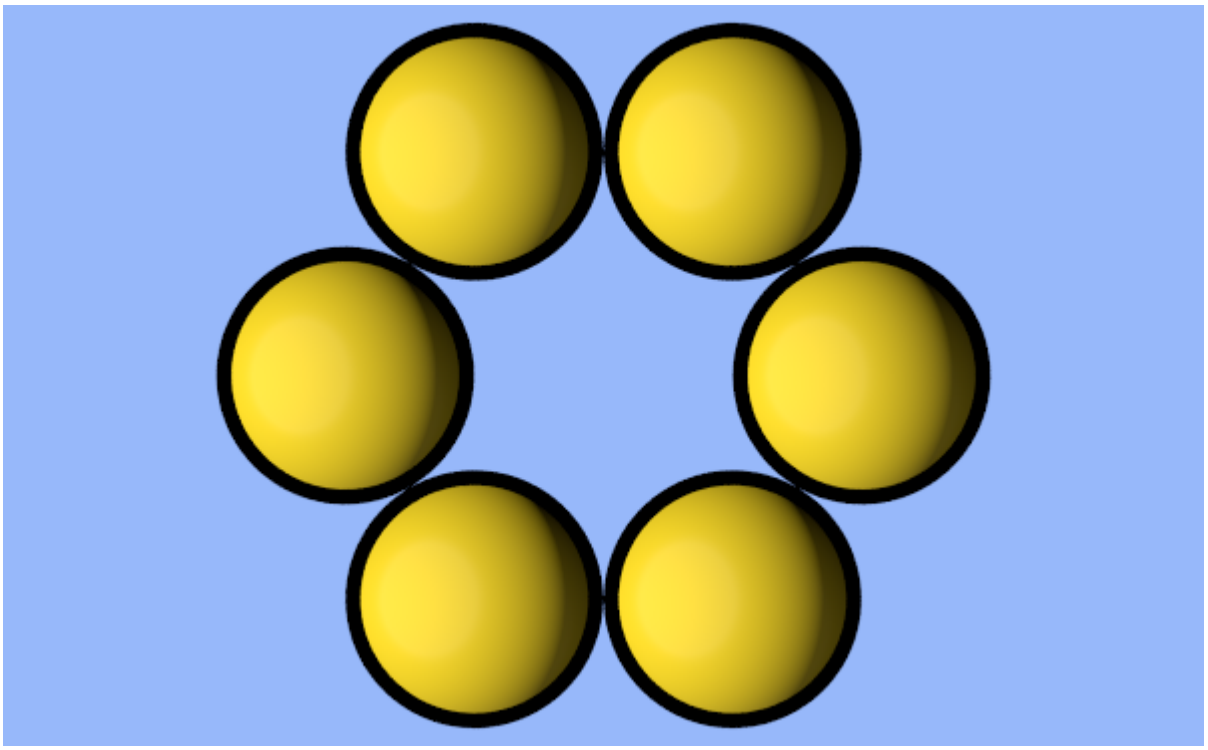


You can modify individual lights.

[12]:
```
scene.lights[0].direction = (-1, 0, 1)
```

[13]:
```
fresnel.preview(scene)
```

[13]:



11.3 Camera

The camera defines the view to render into the scene. The default camera is an orthographic camera at (0,0,100), look at (0,0,0), and has a height of 100:

[14]:
```
scene2 = fresnel.Scene()
print(scene2.camera)
```

```
fresnel.camera.Orthographic(position=(0.0, 0.0, 100.0), look_at=(0.0, 0.0, 0.0),␣
↪up=(0.0, 1.0, 0.0), height=100.0)
```

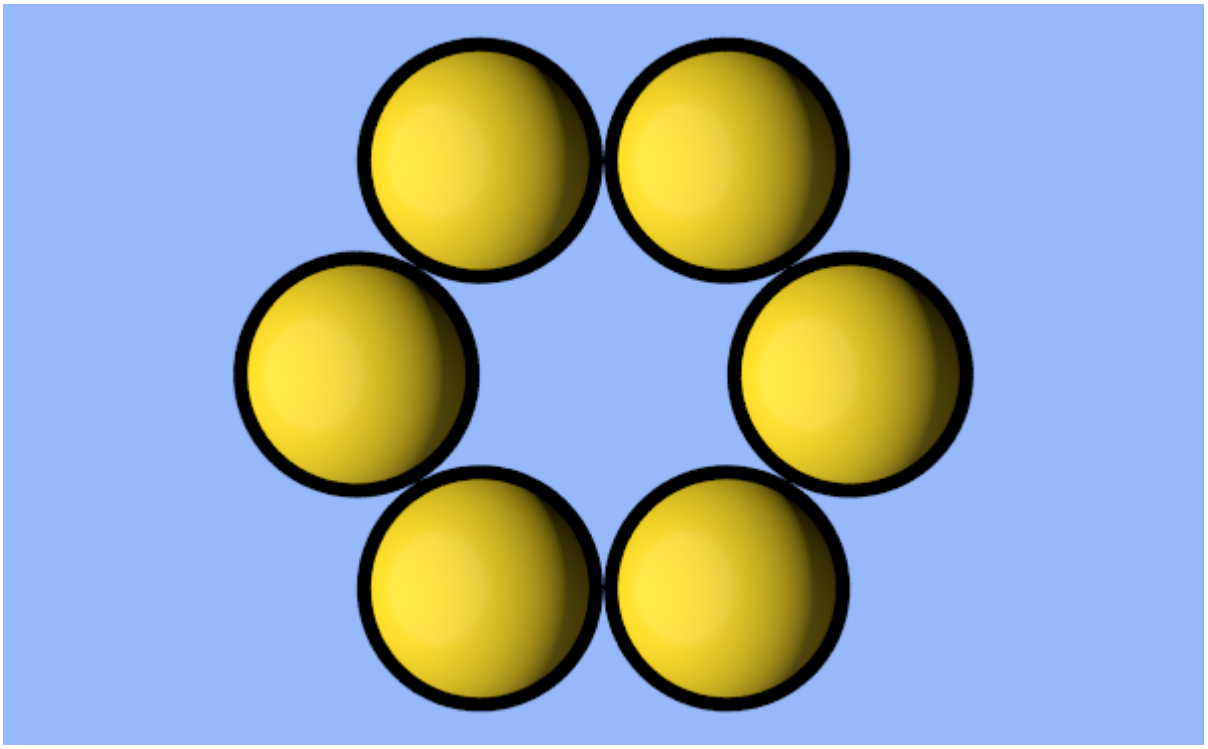
You can automatically fit an orthographic camera to the scene with `camera.Orthographic.fit`. Call it **after** defining all of the geometry in your scene.

[15]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

A **camera** is defined by its *position*, *look-at point*, *up vector* and *height* of the view into the scene. All of these quantities are in scene coordinates.

[16]:
```
scene.camera = fresnel.camera.Orthographic(position=(0,0,2), look_at=(0,0,0), up=(0,1,
↪0), height=6)
fresnel.preview(scene)
```

[16]:

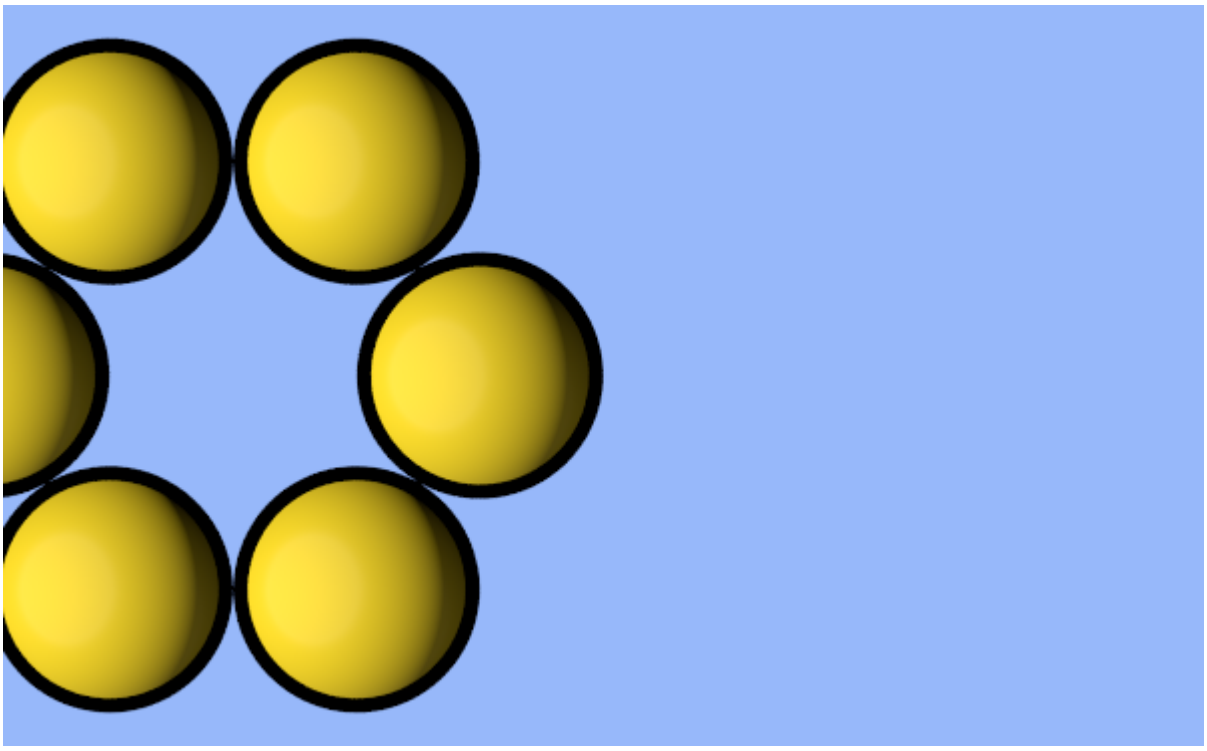


You can modify these parameters individually.

[17]:
```
scene.camera.position = (3, 0, 10)
scene.camera.look_at=(3,0,0)
```

[18]:
```
fresnel.preview(scene)
```

[18]:



Print the full representation of the camera.

[19]:
```
print(repr(scene.camera))
```

```
fresnel.camera.Orthographic(position=(3.0, 0.0, 10.0), look_at=(3.0, 0.0, 0.0), up=(0.
↪0, 1.0, 0.0), height=6.0)
```

You can copy and paste this text to reproduce the same camera elsewhere.

[20]:
```
scene.camera = fresnel.camera.Orthographic(position=(3.0, 0.0, 10.0),
                                           look_at=(3.0, 0.0, 0.0),
                                           up=(0.0, 1.0, 0.0),
                                           height=6.0)
```

This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

TWELVE

LIGHTING SETUPS

Each **Scene** has associated **lights**. The lights control how the objects in a scene is lit.

[1]:
```
import fresnel
import math
import matplotlib, matplotlib.cm
from matplotlib import pyplot
%matplotlib inline
import numpy

device = fresnel.Device()
scene = fresnel.Scene(device)
position = []
for k in range(5):
    for i in range(5):
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.material = fresnel.material.Material(primitive_color_mix=1.0, color=(1,1,1))
mapper = matplotlib.cm.ScalarMappable(norm = matplotlib.colors.Normalize(vmin=0,␣
↪vmax=1, clip=True),
                                      cmap = matplotlib.cm.get_cmap(name='viridis'))

v = numpy.linspace(0,1,len(position))
geometry.color[:] = fresnel.color.linear(mapper.to_rgba(v))
scene.camera = fresnel.camera.Orthographic.fit(scene, view='isometric')
tracer = fresnel.tracer.Path(device, w=450, h=450)
```

12.1 Lighting presets

Fresnel defines many lighting presets that use classic photography techniques to light the scene. Create a setup and assign it to the Scene's lights.

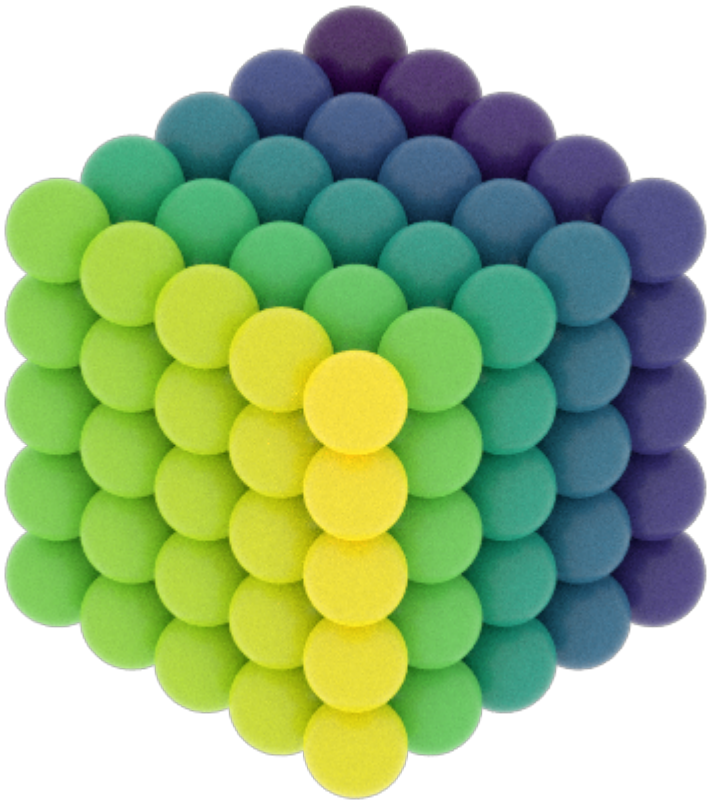
The images in these examples are noisy because of the small number of samples. Increase the number of samples to obtain less noisy images.

12.1.1 Light box

A light box lights the scene equally from all sides. This type of lighting is commonly used product photography.

[2]:
```
scene.lights = fresnel.light.lightbox()
tracer.sample(scene, samples=64, light_samples=10)
```

[2]:

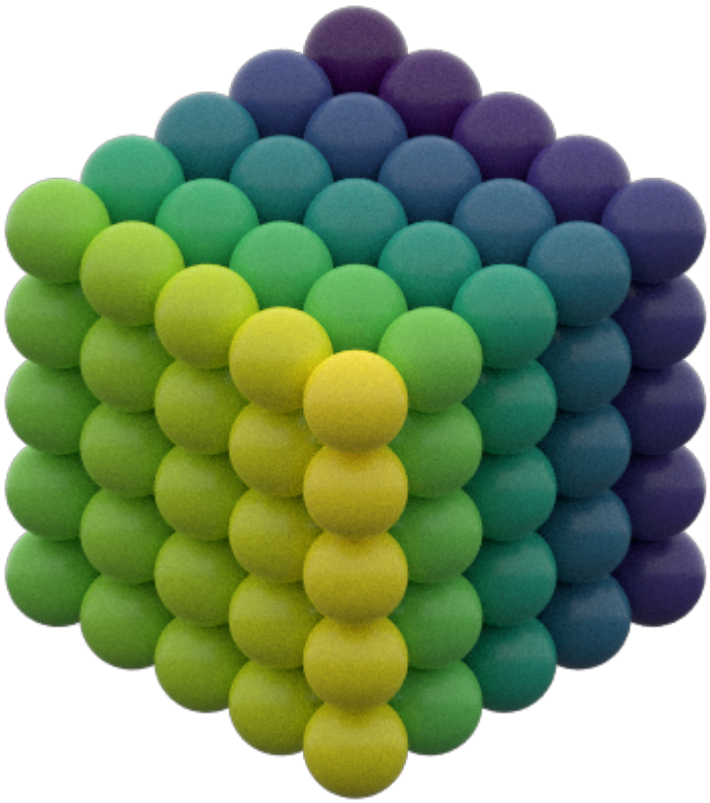


12.1.2 Cloudy

Cloudy lighting mimics a cloudy day. Strong light comes from all directions above, while weak light comes from below.

[3]:
```
scene.lights = fresnel.light.cloudy()
tracer.sample(scene, samples=64, light_samples=10)
```

[3]:

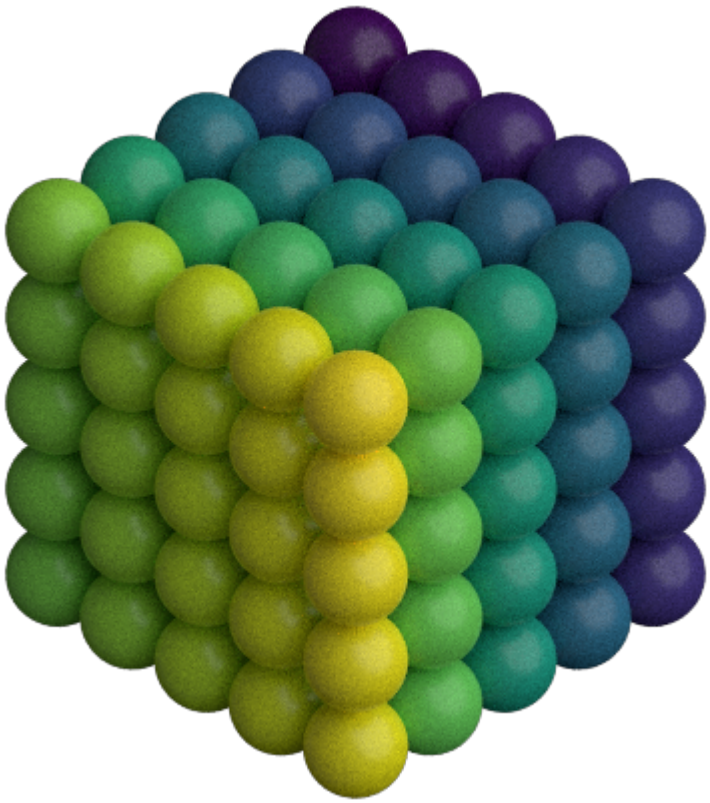


12.1.3 Rembrandt

Rembrandt lighting places the key light 45 degrees to one side and slightly up.

[4]:
```
scene.lights = fresnel.light.rembrandt()
tracer.sample(scene, samples=64, light_samples=10)
```

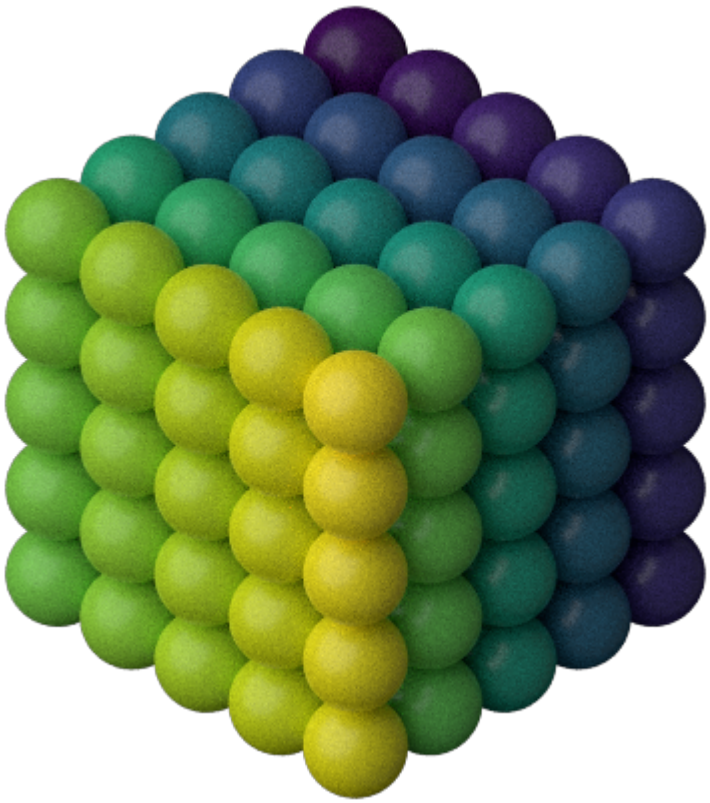
[4]:



Use the *side* argument specify which side to place the key light on.

[5]:
```
scene.lights = fresnel.light.rembrandt(side='left')
tracer.sample(scene, samples=64, light_samples=10)
```

[5]:

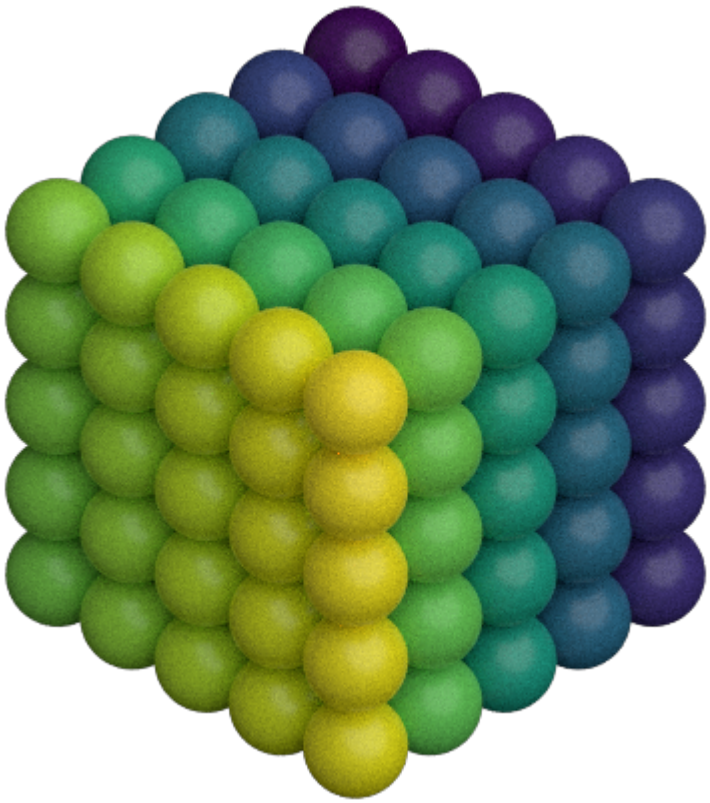


12.1.4 Loop lighting

Loop lighting places the key light slightly to one side and slightly up.

[6]:
```
scene.lights = fresnel.light.loop()
tracer.sample(scene, samples=64, light_samples=10)
```

[6]:

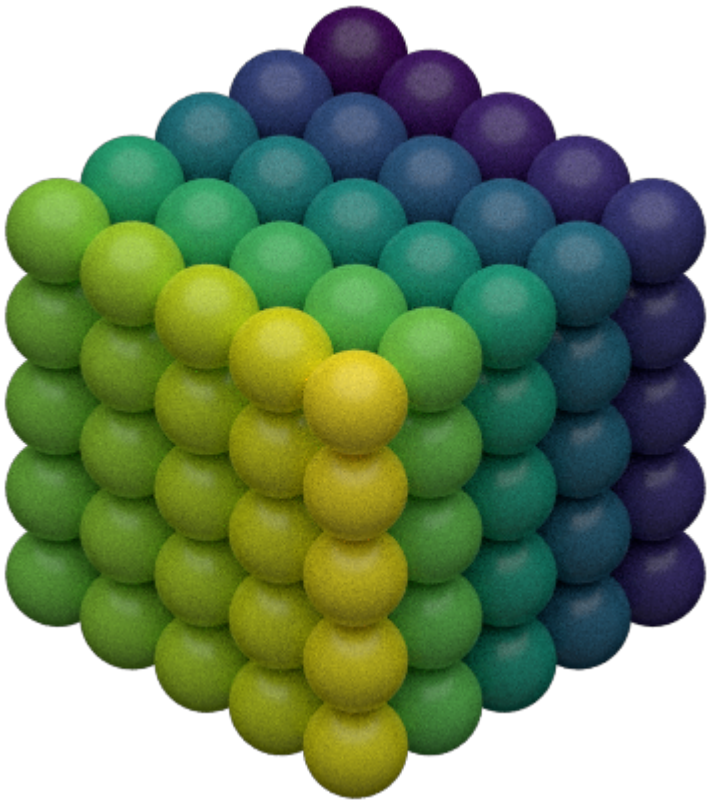


12.1.5 Butterfly lighting

Butterfly lighting places the key light high above the camera.

[7]:
```
scene.lights = fresnel.light.butterfly()
tracer.sample(scene, samples=64, light_samples=10)
```

[7]:

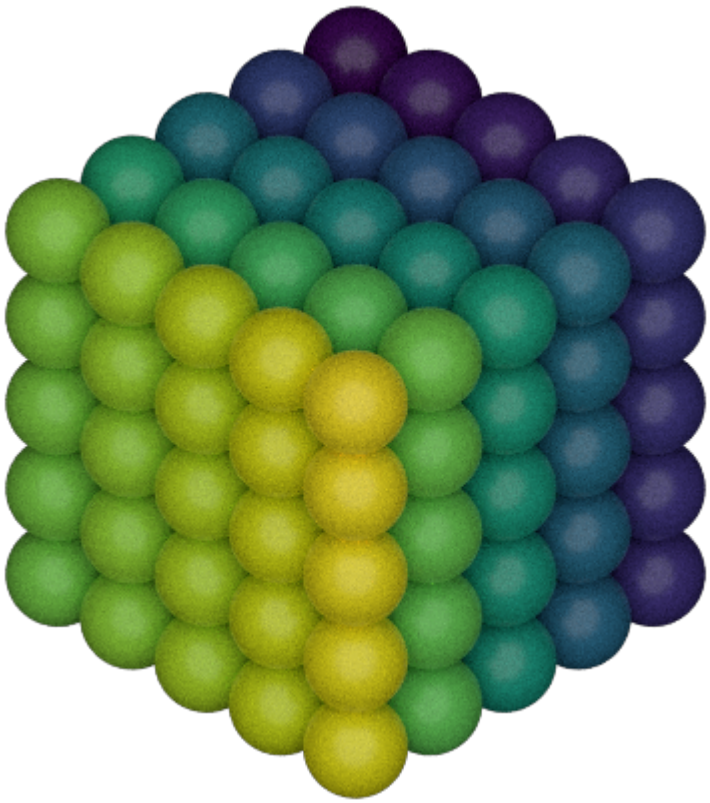


12.1.6 Ring lighting

The ring lighting setup provides a strong front area light. This type of lighting is common in fashion photography.

[8]:
```
scene.lights = fresnel.light.ring()
tracer.sample(scene, samples=64, light_samples=10)
```

[8]:



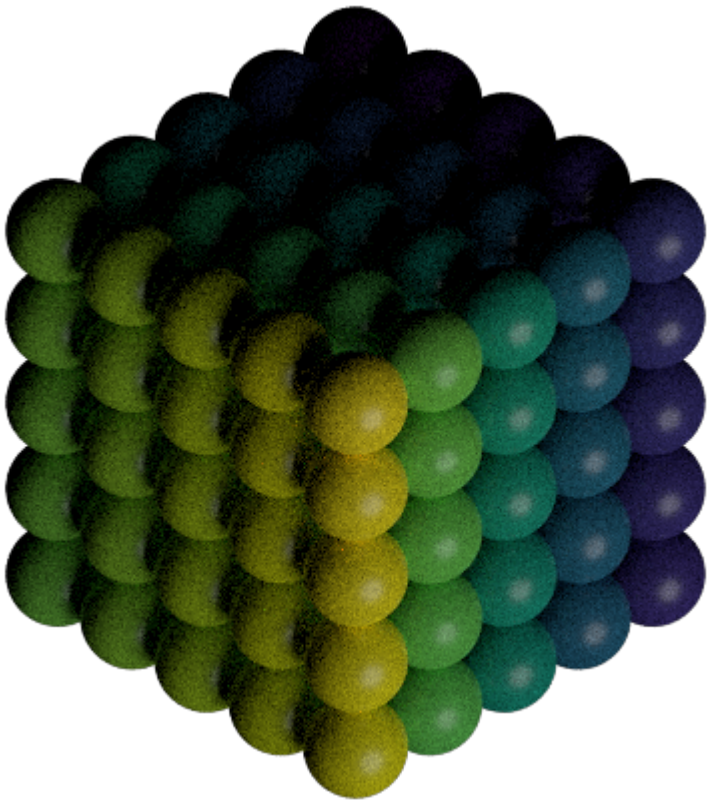
12.2 Custom lights

You can define your own custom lights. Provide a *direction* vector pointing to the light in the coordinate system of the camera (+x points to the right, +y points up, and +z points out of the screen). The light color defines both the color (RGB) and the intensity of the light in a linear sRGB color space.

[9]:
```
my_lights = [fresnel.light.Light(direction=(1,-1,1), color=(1,1,1))]
scene.lights = my_lights
```

[10]:
```
tracer.sample(scene, samples=64, light_samples=10)
```

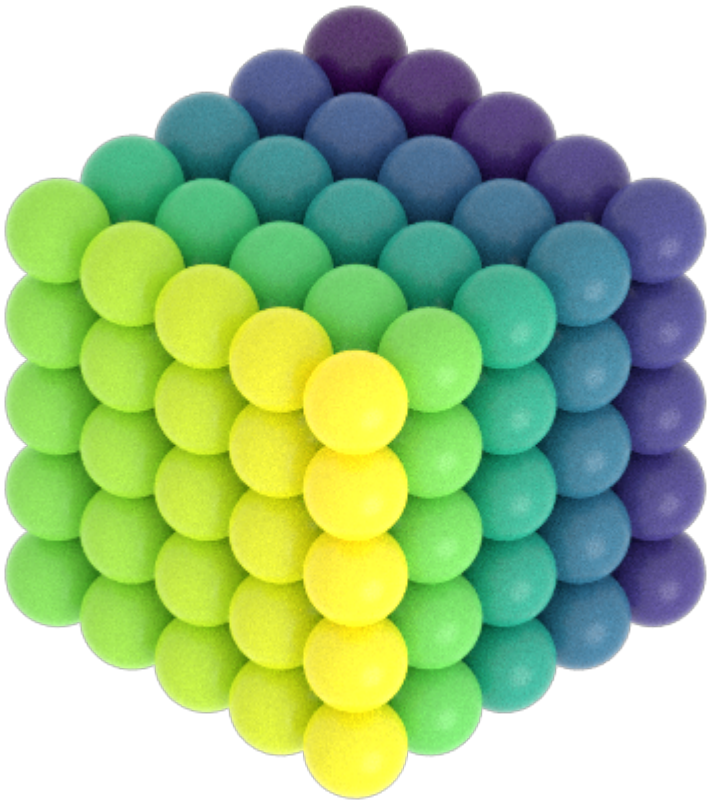
[10]:



The shadows are very dark. Add another light to fill them in. You can access the scene's lights directly. The value *theta* defines the half angle width of the light source. Large lights provide soft shadows.

[11]:
```
scene.lights.append(fresnel.light.Light(direction=(0,0,1), color=(1,1,1), theta=3.14))
tracer.sample(scene, samples=64, light_samples=10)
```

[11]:

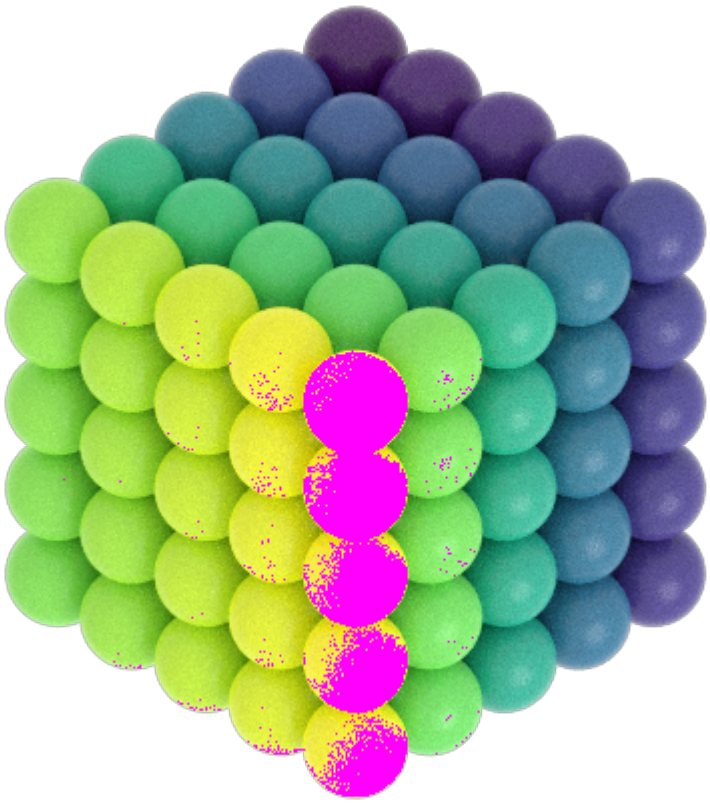


This image is overexposed.

Highlight warnings show overexposed areas of the image as a special color (default: magenta).

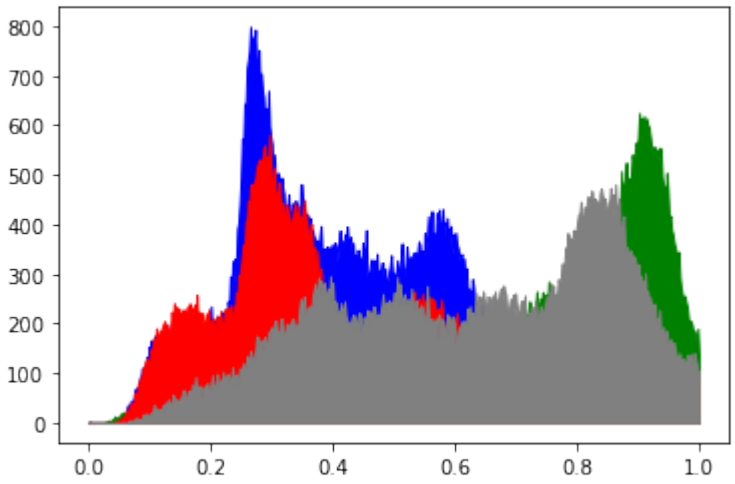
[12]:
```
tracer.enable_highlight_warning()
tracer.render(scene)
```

[12]:



If the histogram is blocking up at 1.0, there are overexposed highlights.

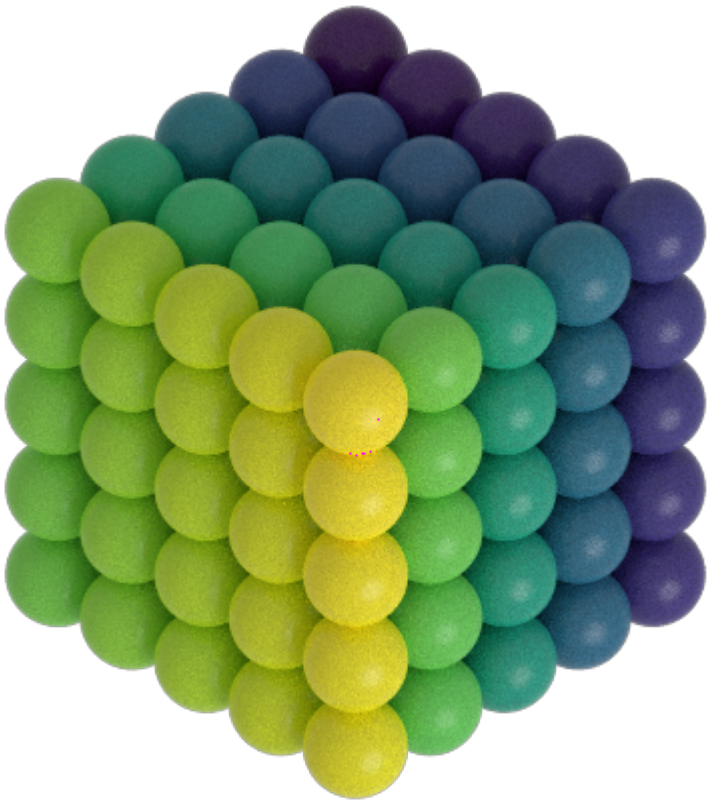
[13]:
```
L, bins = tracer.histogram()
pyplot.fill_between(bins, L[:,3], color='blue');
pyplot.fill_between(bins, L[:,2], color='green');
pyplot.fill_between(bins, L[:,1], color='red');
pyplot.fill_between(bins, L[:,0], color='gray');
```



Reduce the intensity of the light to correctly expose the image.

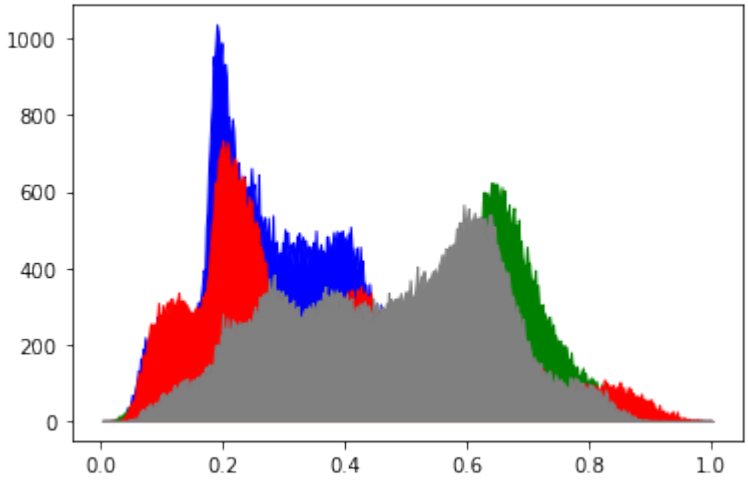
[14]:
```
scene.lights[1].color=(0.45,0.45,0.45)
tracer.sample(scene, samples=64, light_samples=10)
```

[14]:



Now there are no clipping warnings and the histogram shows a perfectly exposed image.

[15]:
```
L, bins = tracer.histogram()
pyplot.fill_between(bins, L[:,3], color='blue');
pyplot.fill_between(bins, L[:,2], color='green');
pyplot.fill_between(bins, L[:,1], color='red');
pyplot.fill_between(bins, L[:,0], color='gray');
```

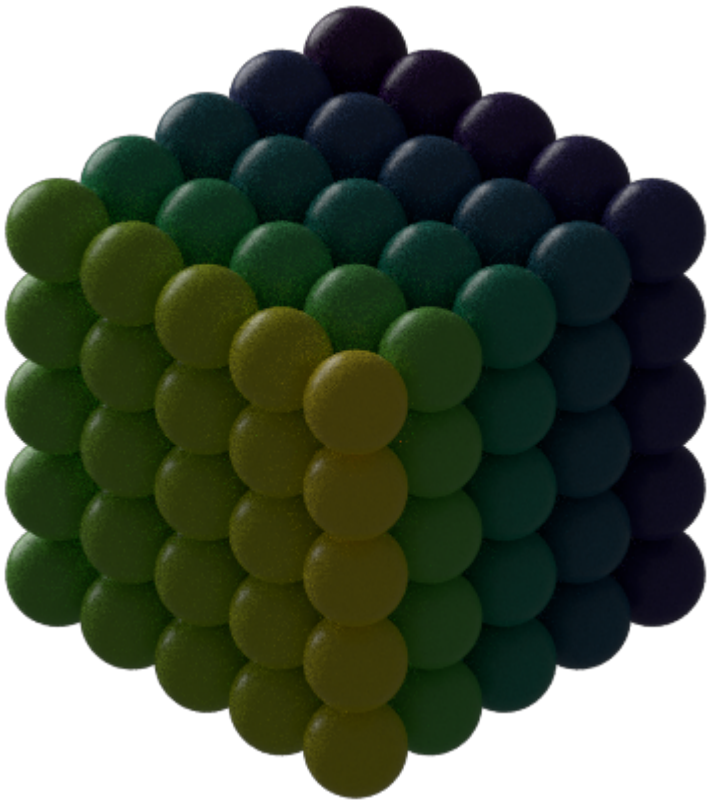


`scene.lights` has typical sequence like behavior. You can assign a sequence of Light objects to it, append lights to it, and loop over the lights in it. For example, reverse the direction of every light:

[16]:
```
for l in scene.lights:
    d = l.direction;
    l.direction = (-d[0], -d[1], -d[2])
```

[17]:
```
scene.lights[1].color=(0.05,0.05,0.05)
tracer.disable_highlight_warning()
tracer.sample(scene, samples=64, light_samples=10)
```

[17]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

THIRTEEN

SPHERE

```
[ ]: import fresnel
     scene = fresnel.Scene()
```

The **sphere geometry** defines a set of N spheres. Each sphere has its own *position*, *radius*, and *color*.

```
[ ]: geometry = fresnel.geometry.Sphere(scene, N=3)
     geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.
     ↪9]),
                                                   roughness=0.8)
```

13.1 Geometric properties

position defines the position of each sphere.

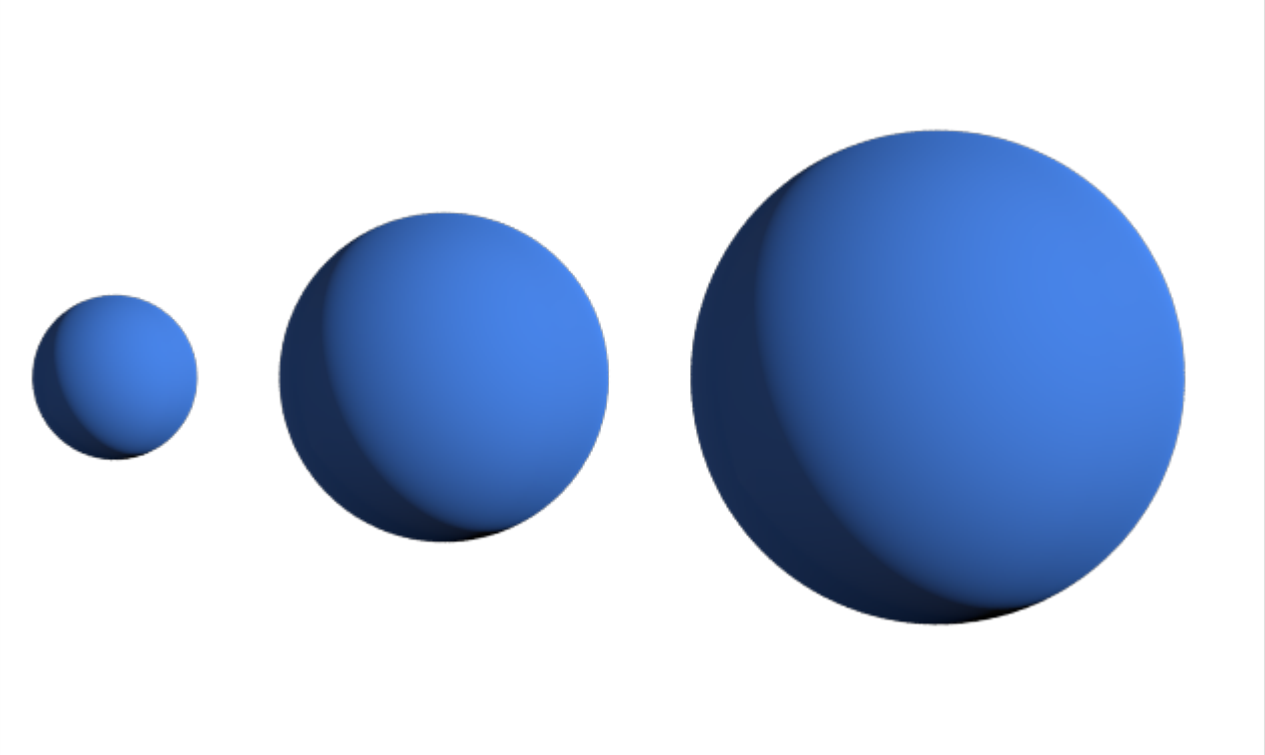
```
[ ]: geometry.position[:] = [[-2,0,0], [0, 0, 0], [3, 0, 0]]
```

radius sets the radius of each sphere.

```
[ ]: geometry.radius[:] = [0.5, 1.0, 1.5]
```

```
[5]: scene.camera = fresnel.camera.Orthographic.fit(scene, view='front', margin=0.5)
     fresnel.preview(scene)
```

[5]:



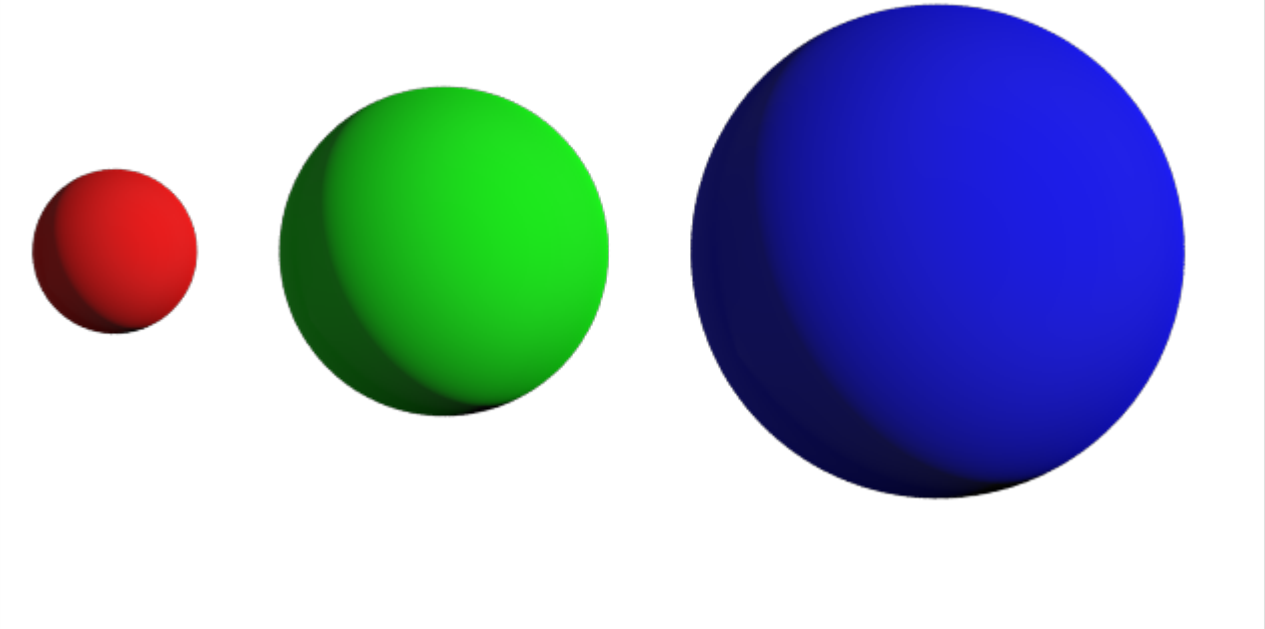
13.2 Color

color sets the color of each sphere (when when *primitive_color_mix* > *0*)

[6]:
```
geometry.color[:] = fresnel.color.linear([[0.9,0,0], [0, 0.9, 0], [0, 0, 0.9]])
geometry.material.primitive_color_mix = 1.0
```

[7]:
```
fresnel.preview(scene)
```

[7]:



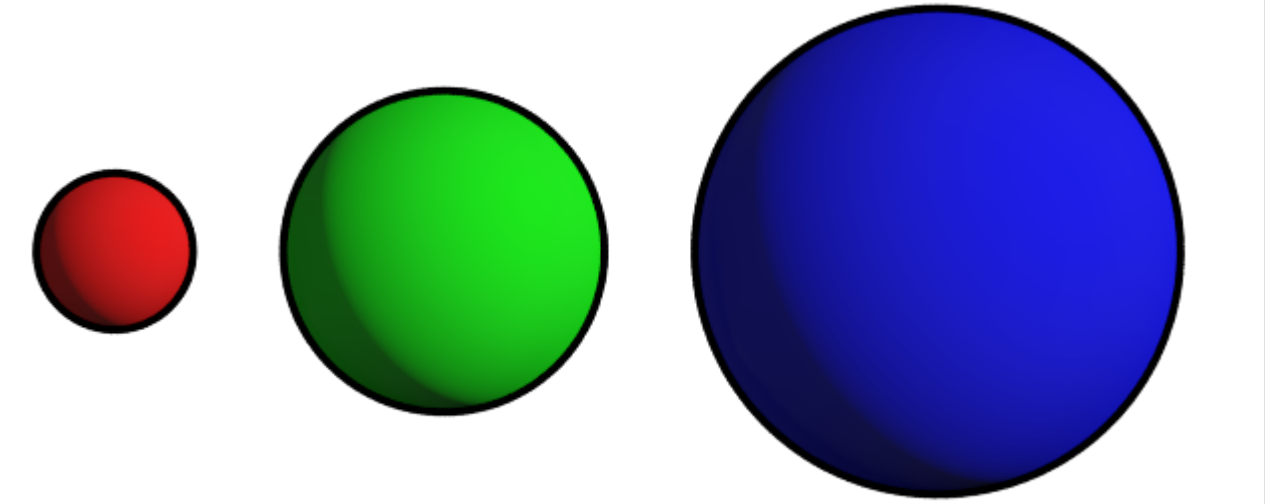
13.3 Outlines

Outlines are applied on the outer edge of the sphere in the view plane.

[8]:
```
geometry.outline_width = 0.05
```

[9]:
```
fresnel.preview(scene)
```

[9]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

FOURTEEN

CYLINDER

[1]:
```
import fresnel
scene = fresnel.Scene()
```

The **cylinder geometry** defines a set of N spherocylinders. Each spherocylinder is defined by two end points and has its own *radius*, and end point *colors*.

[2]:
```
geometry = fresnel.geometry.Cylinder(scene, N=3)
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.
↪9]),
                                              roughness=0.8)
```

14.1 Geometric properties

points defines the end points of each cylinder.

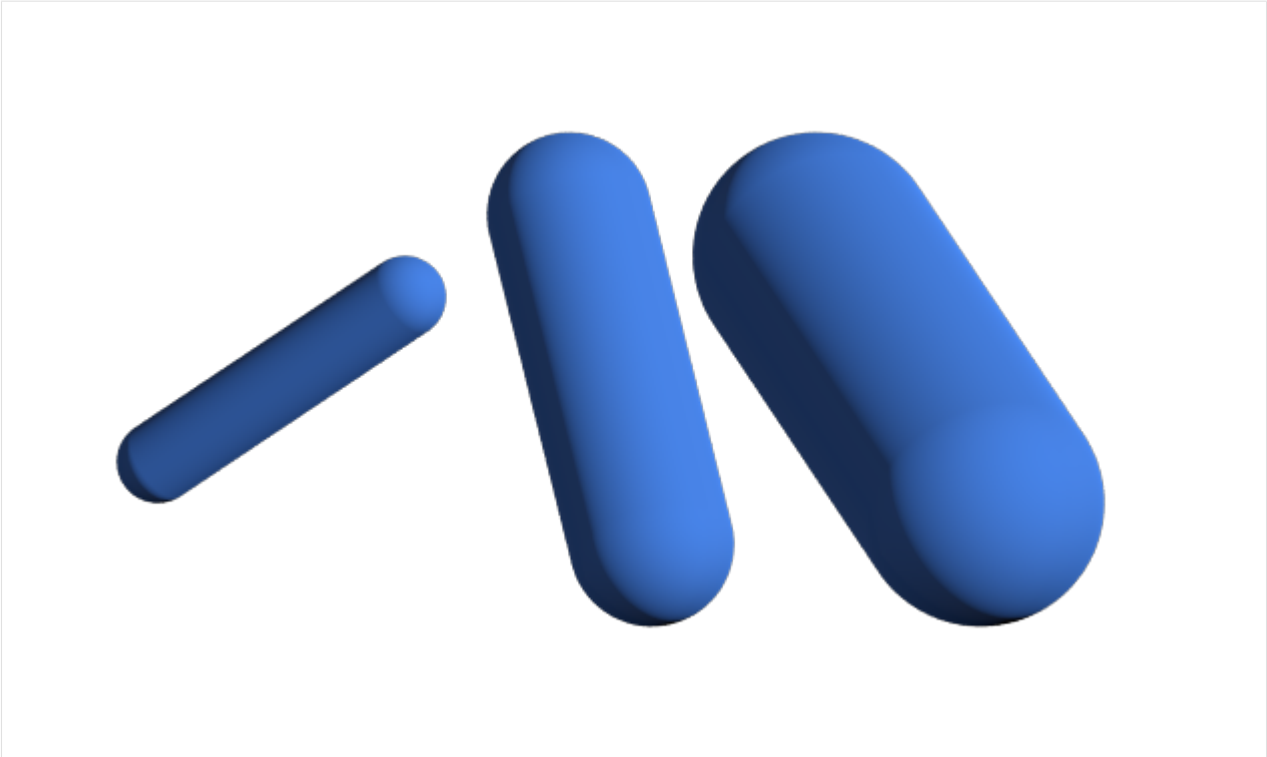
[3]:
```
geometry.points[:] = [[[-5,-1,-1], [-2, 1, 1]],
                      [[1, -2, 1],[0, 2, -1]],
                      [[5, -1.5, 2], [3, 1.5, -2]]]
```

radius sets the radius of each spherocylinder.

[4]:
```
geometry.radius[:] = [0.5, 1.0, 1.5]
```

[5]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene, view='front', margin=0.5)
fresnel.preview(scene)
```

[5]:



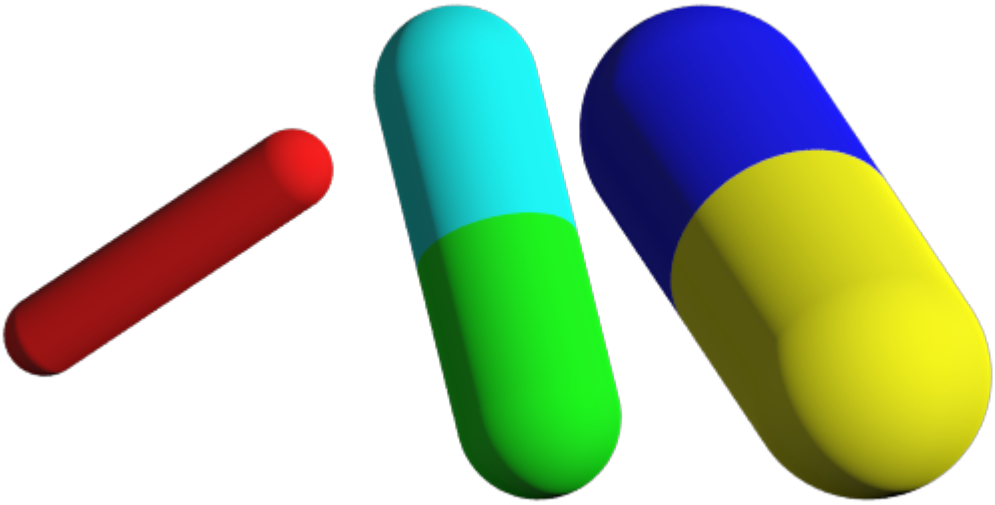
14.2 Color

color sets the color of the end points of each cylinder (when *primitive_color_mix* > *0*). The color transitions at the midpoint.

[6]:
```
geometry.color[:] = [[[0.9,0,0], [0.9, 0, 0]],
                     [[0, 0.9, 0], [0, 0.9, 0.9]],
                    [[0.9, 0.9, 0], [0, 0, 0.9]]]
geometry.material.primitive_color_mix = 1.0
```

[7]:
```
fresnel.preview(scene)
```

[7]:



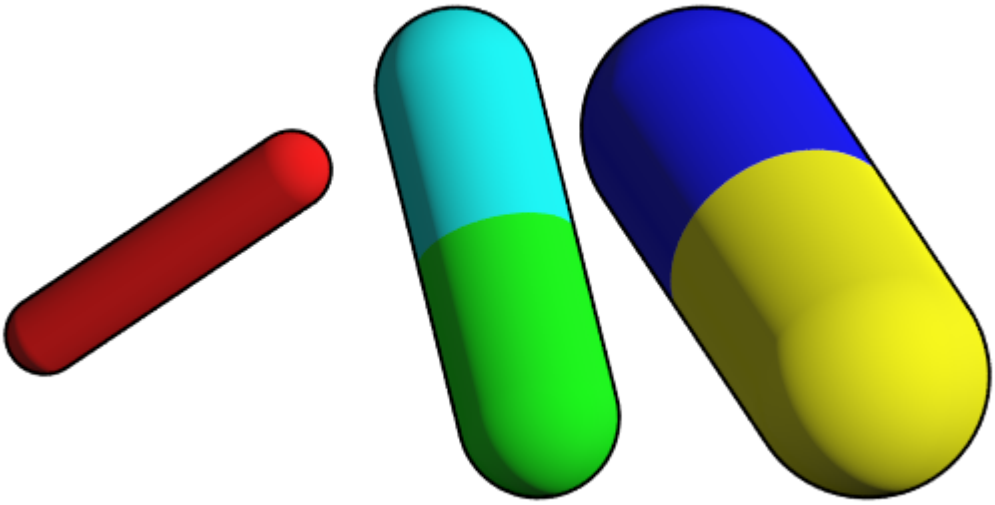
14.3 Outlines

Outlines are applied on the outer edge of the cylinder in the view plane.

[8]:
```
geometry.outline_width = 0.05
```

[9]:
```
fresnel.preview(scene)
```

[9]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

FIFTEEN

CONVEX POLYHEDRON

[1]:
```
import fresnel
import itertools
import math
import numpy as np
device = fresnel.Device()
scene = fresnel.Scene(device)
```

The **convex polyhedron geometry** defines a set of N convex polyhedra. The shape of all N polyhedra is identical and defined by P planes. Each polyhedron has its own *position*, *orientation*, and *color*. You must also specify the circumsphere radius r. Note that the information used to draw a convex polyhedron is easily obtained from it's vertices via the `util.convex_polyhedron_from_vertices()` utility function.

To construct a truncated cube:

[2]:
```
# first get cube verts
pm = [-1, 1]
cube_verts = list(itertools.product(pm, repeat=3))
trunc_cube_verts = []
# truncate by removing corners and adding vertices to edges
for p1, p2 in itertools.combinations(cube_verts, 2):
    # don't add points along any diagonals
    match = (p1[0]==p2[0], p1[1]==p2[1], p1[2]==p2[2])
    if match.count(False) == 1: # only 1 coordinate changes, not a diagonal
        p1, p2 = np.array(p1), np.array(p2)
        vec = p2 - p1
        trunc_cube_verts.append(p1 + vec/3)
        trunc_cube_verts.append(p1 + 2*vec/3)
```

[3]:
```
c1 = fresnel.color.linear([0.70, 0.87, 0.54])*0.8
c2 = fresnel.color.linear([0.65,0.81,0.89])*0.8
colors = {8: c1, 3: c2}
poly_info = fresnel.util.convex_polyhedron_from_vertices(trunc_cube_verts)
for idx, fs in enumerate(poly_info['face_sides']):
    poly_info['face_color'][idx] = colors[fs]
geometry = fresnel.geometry.ConvexPolyhedron(scene,
                                             poly_info,
                                             N=3
                                            )
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.
↪9]),
                                              roughness=0.8)
```

15.1 Geometric properties

position defines the position of the center of each convex polyhedron.

[4]:
```
geometry.position[:] = [[-3,0,0], [0, 0, 0], [3, 0, 0]]
```

orientation sets the orientation of each convex polyhedron as a quaternion

[5]:
```
geometry.orientation[:] = [[1, 0, 0, 0],
                           [0.80777943,  0.41672122,  0.00255412,  0.41692838],
                           [0.0347298, 0.0801457, 0.98045, 0.176321]]
```

[6]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene, view='front', margin=0.8)
fresnel.preview(scene)
```

[6]:

15.2 Color

color sets the color of each individual convex polyhedron (when *primitive_color_mix* > *0* and *color_by_face* < *1*)

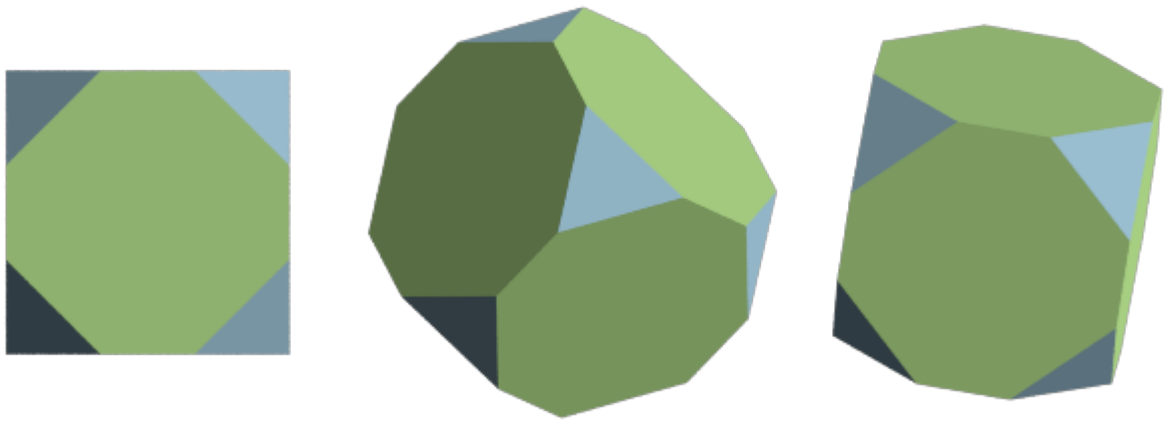
[7]:
```
geometry.color[:] = fresnel.color.linear([[0.9,0,0], [0, 0.9, 0], [0, 0, 0.9]])
geometry.material.primitive_color_mix = 1.0
fresnel.preview(scene)
```

[7]:

Set **color_by_face** > 0 to color the faces of the polyhedra independently. `poly_info['face_colors']` (i.e., the output of convex_polyhedron_from_vertices, which we modified above) sets the color of each face. Above, we set the color of the each face based on number of sides it has.

[8]:
```
geometry.color_by_face = 1.0
fresnel.preview(scene)
```

[8]:

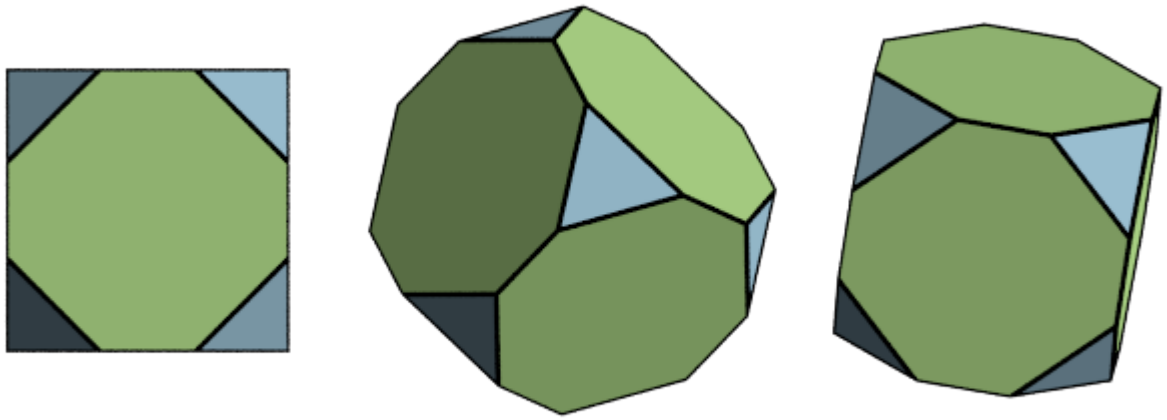


15.3 Outlines

Outlines are applied at the outer edge of each face.

[9]:
```
geometry.outline_width = 0.02
fresnel.preview(scene)
```

[9]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

[1]:
```
import fresnel
import numpy
import matplotlib, matplotlib.cm
```

CHAPTER

SIXTEEN

MESH

The **mesh geometry** defines a generic triangle mesh. Define a mesh with an 3Tx3 array where T is the number of triangles. Triangles must be specified with a counter clockwise winding. Here is the Standford bunny as an example:

[2]:
```
# https://graphics.stanford.edu/~mdfisher/Data/Meshes/bunny.obj
verts = numpy.load('bunny.npy')
```

16.1 Geometric properties

Pass the vertices to the mesh geometry.

[3]:
```
scene1 = fresnel.Scene()
bunny = fresnel.geometry.Mesh(scene1,vertices=verts,N=1)
```

[4]:
```
bunny.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.9]),
↪ roughness=0.6)
scene1.camera = fresnel.camera.Orthographic.fit(scene1,margin=0)
scene1.lights = fresnel.light.cloudy()
fresnel.pathtrace(scene1, samples=200)
```

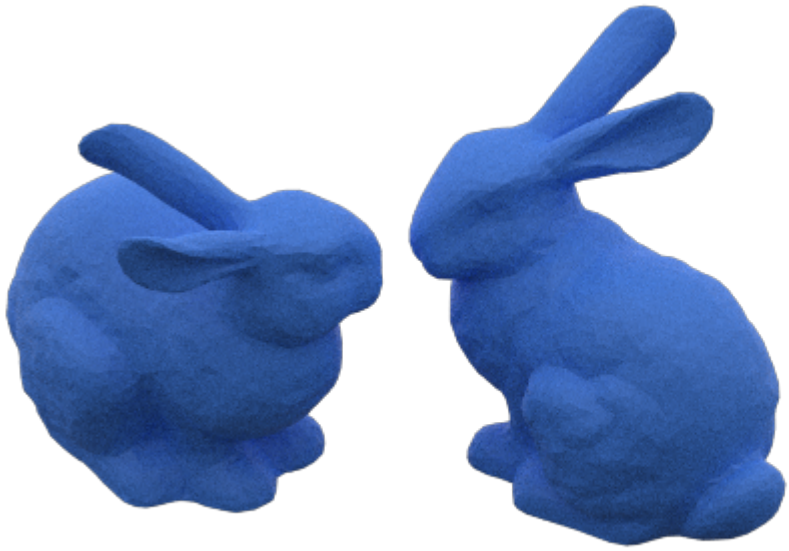
[4]:

Specify position and orientation to instantiate the mesh many times.

[5]:
```
scene2 = fresnel.Scene()
bunnies = fresnel.geometry.Mesh(scene2,vertices=verts,N=2)
bunnies.position[:] = [[0,0,0], [-0.11,0,0.1]]
bunnies.orientation[:] = [[1,0,0,0], [0,0,1,0]]
```

[6]:
```
bunnies.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.
↪9]), roughness=0.6)
scene2.camera = fresnel.camera.Orthographic.fit(scene1,margin=0)
scene2.lights = fresnel.light.cloudy()
fresnel.pathtrace(scene2, samples=200)
```

[6]:



16.2 Color

Specify per vertex colors. These colors are smoothly interpolated across the triangles. Set `primitive_color_mix=1` to choose the per-vertex colors.

Color the bunny based on the y-coordinate of the mesh:

[7]:
```
mapper = matplotlib.cm.ScalarMappable(norm = matplotlib.colors.Normalize(vmin=-0.08,␣
↪vmax=0.05, clip=True),
                                      cmap = matplotlib.cm.get_cmap(name='viridis'))

bunny.color[:] = fresnel.color.linear(mapper.to_rgba(verts[:,1]))
bunny.material.primitive_color_mix = 1.0
```

[8]:
```
fresnel.pathtrace(scene1, samples=200)
```

[8]:

Here is a single triangle demo to demonstrate the interpolation:

[9]:
```
scene3 = fresnel.Scene()
triangle = fresnel.geometry.Mesh(scene3,vertices=[[0,0,0],[1,0,0],[0,1,0]],N=1)
triangle.material.solid = 1
triangle.material.primitive_color_mix = 1.0
triangle.color[:] = [[1,0,0], [0,1,0], [0,0,1]]
```

[10]:
```
scene3.camera = fresnel.camera.Orthographic.fit(scene3, view='front')
fresnel.preview(scene3)
```

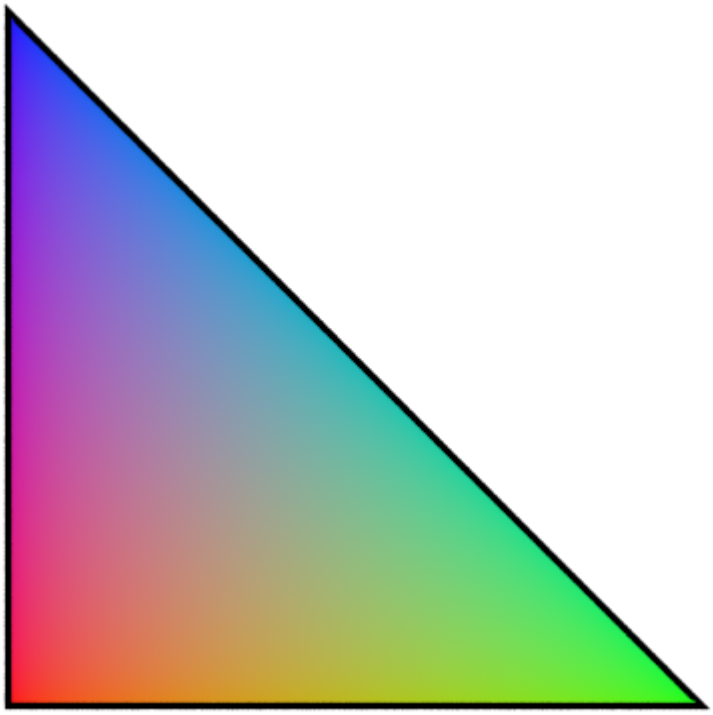
[10]:

16.3 Outlines

Outlines are placed on the outer edge of each triangle in the mesh.

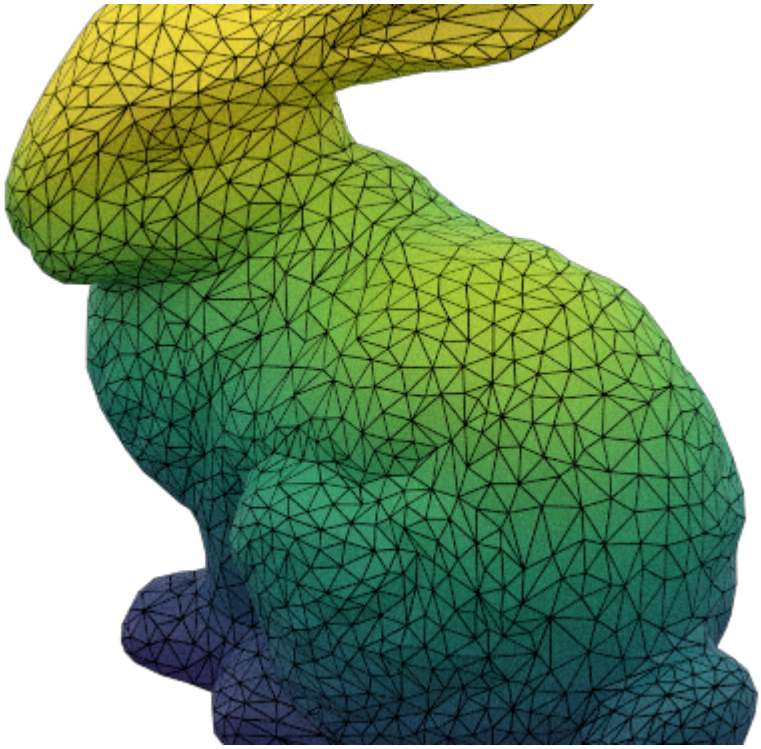
[11]:
```
triangle.outline_width=0.01
fresnel.preview(scene3)
```

[11]:



[12]:
```
bunny.outline_width=0.0002
scene1.camera.height *= 0.5
fresnel.pathtrace(scene1, samples=200)
```

[12]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-

examples repository.

CHAPTER

SEVENTEEN

POLYGON

[1]:
```
import fresnel
scene = fresnel.Scene()
```

The **polygon geometry** defines a set of N simple polygons in two dimensions. All polygons in the geometry have the same vertices. Each polygon has a separate *position*, orientation *angle*, and *color*.

[2]:
```
geometry = fresnel.geometry.Polygon(scene,
                                    N=2,
                                    vertices = [[0, -1], [1, 1],
                                                [0, 0.5], [-1, 1]])
geometry.material.color = fresnel.color.linear([0.20,0.64,0.58])
geometry.material.solid=1
```

17.1 Geometric properties

position defines the position of each polygon in the *z=0* plane.

[3]:
```
geometry.position[:] = [[-1,0],
                        [1, 0]]
```

angle defines the rotation angle of each polygon

[4]:
```
geometry.angle[:] = [0.1, -1.0]
```

[5]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

[5]:

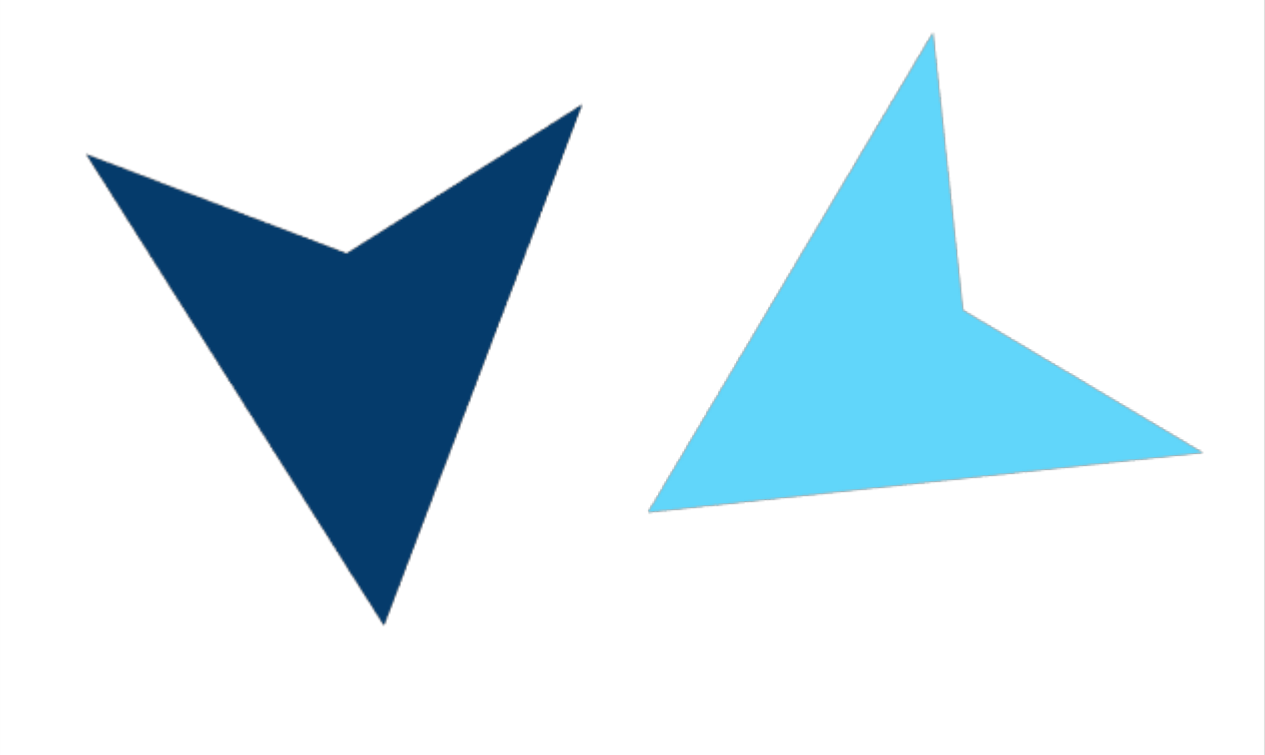
17.2 Color

color sets the color of each polygon (when *primitive_color_mix* > *0*).

[6]:
```
geometry.color[:] = [fresnel.color.linear([0.02,0.23,0.42]),
                     fresnel.color.linear([0.38,0.84,0.98])];
geometry.material.primitive_color_mix = 1.0
```

[7]:
```
fresnel.preview(scene)
```

[7]:



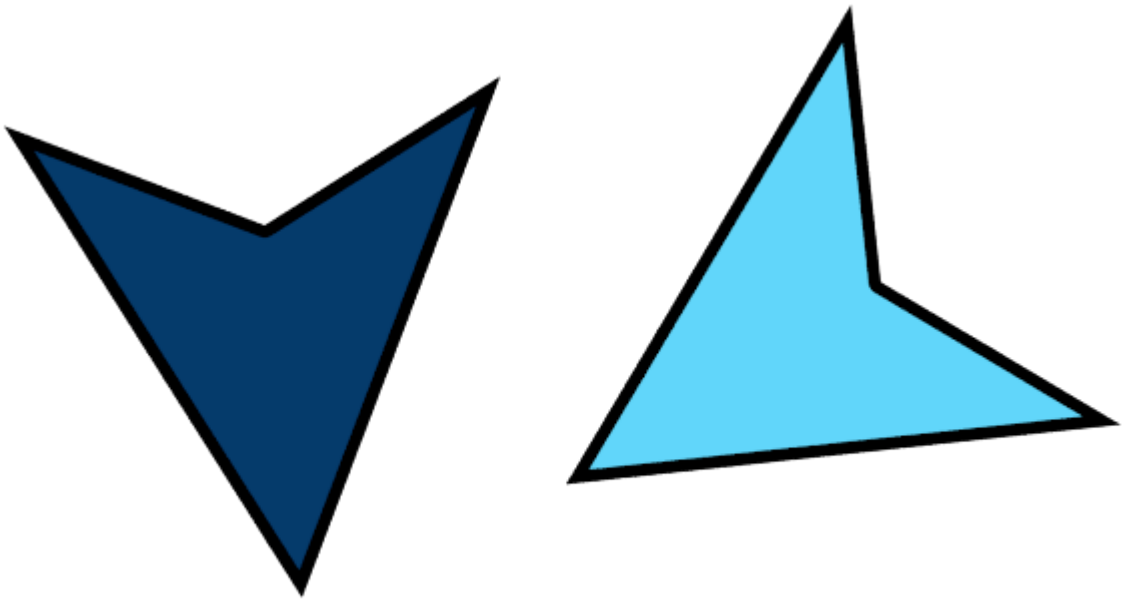
17.3 Outlines

Outlines are applied inside the outer edge of the polygon in the z=0 plane.

[8]:
```
geometry.outline_width = 0.05
```

[9]:
```
fresnel.preview(scene)
```

[9]:



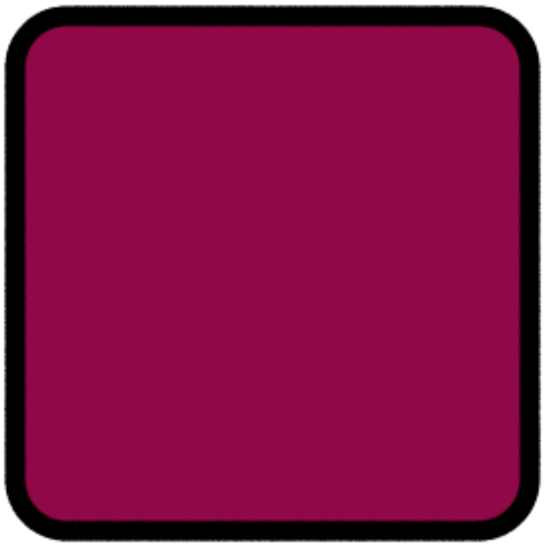
17.4 Rounded polygons

Specify *rounding_radius* to round the edges of the polygon.

[10]:
```
scene2 = fresnel.Scene()
geometry2 = fresnel.geometry.Polygon(scene2,
                                     rounding_radius=0.3,
                                     N=1,
                                     vertices = [[-1, -1], [1, -1],
                                                 [1, 1], [-1, 1]],
                                     outline_width=0.1)
geometry2.material.color=fresnel.color.linear([0.56,0.03,0.28])
geometry2.material.solid=1
scene2.camera = fresnel.camera.Orthographic.fit(scene2)
```

[11]:
```
fresnel.preview(scene2)
```

[11]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

EIGHTEEN

BOX

[1]:
```
import sys

import fresnel

scene = fresnel.Scene()
```

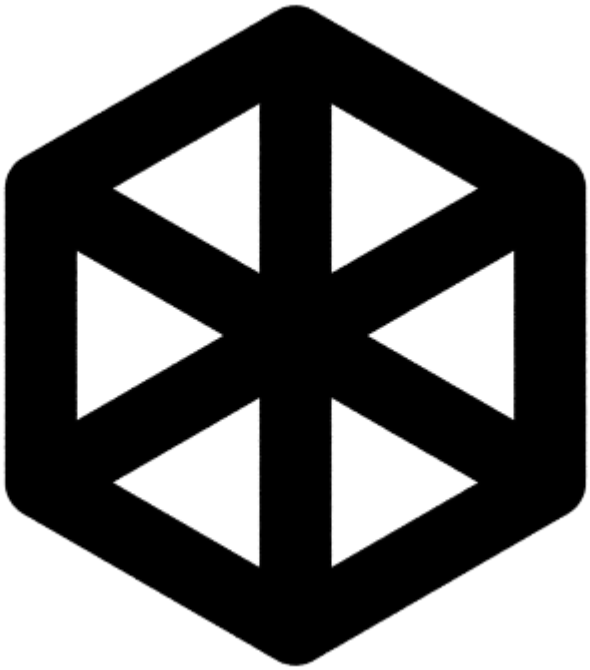
The Box geometry is a convenience class that uses the Cylinder geometry to draw a box. The Box geometry supports triclinic boxes and follows hoomd-blue box conventions. For cubic boxes, only one length is needed.

[2]:
```
my_box = [5]
```

[3]:
```
geometry = fresnel.geometry.Box(scene, my_box)
```

[4]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

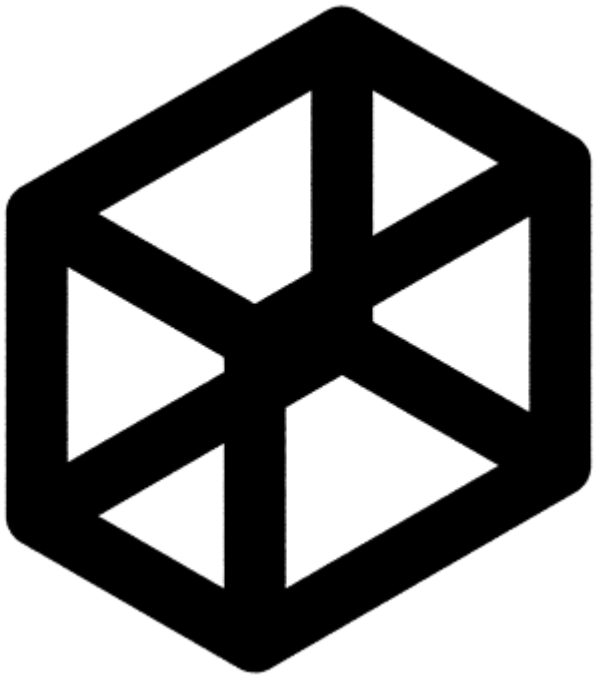
[4]:



Orthorhombic boxes require an Lx, Ly, and Lz

[5]:
```
my_box = [5,6,7]
scene = fresnel.Scene()
geometry = fresnel.geometry.Box(scene, my_box)
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

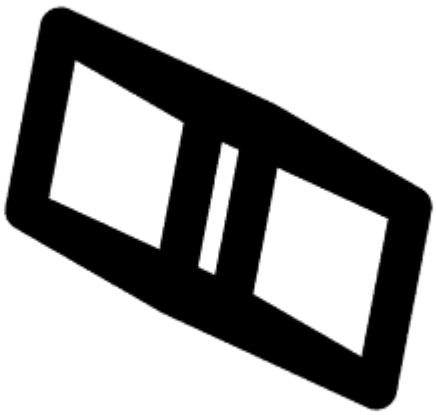
[5]:



Triclinic boxes are supported by using 6 terms, Lx, Ly, Lz, xy, xz, yz.

[6]:
```
my_box = [5,6,7, .2, 0, .9]
scene = fresnel.Scene()
geometry = fresnel.geometry.Box(scene, my_box)
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

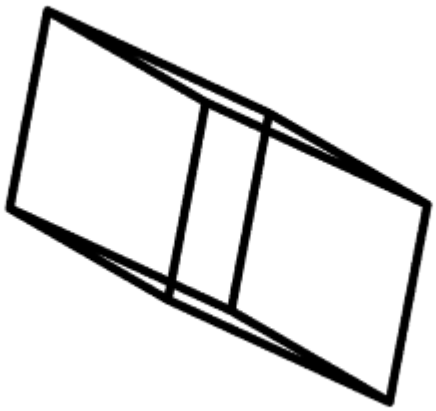
[6]:



The radius defaults to `0.5` but can be specifed when the box is created

[7]:
```
my_box = [5,6,7, .2, 0, .9]
scene = fresnel.Scene()
geometry = fresnel.geometry.Box(scene, my_box, box_radius=.1)
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

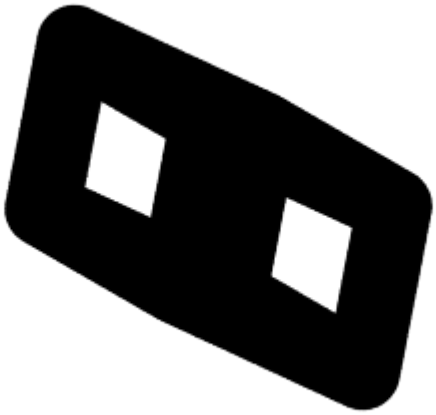
[7]:



Or changed later

[8]:
```
geometry.box_radius = 1
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

[8]:



The box color can also be set on initialization.

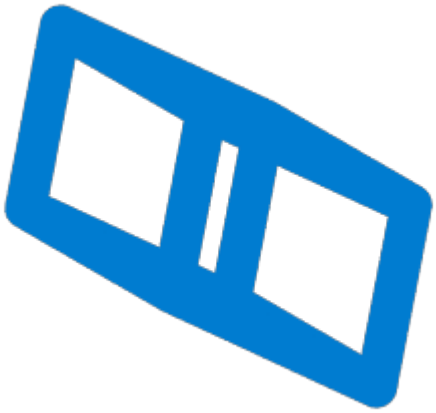
[9]:
```
my_box = [5, 6, 7, 0.2, 0, 0.9]
scene = fresnel.Scene()
geometry = fresnel.geometry.Box(scene, my_box, box_color=[214 / 255, 67 / 255, 9 /␣
↪255])
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene)
```

[9]:

Or changed later.

[10]:
```
geometry.box_color = [0, 51 / 255, 160 / 255]
fresnel.preview(scene)
```

[10]:



The box size and shape can also be updated.

[11]:
```
geometry.box = [4, 4, 5, 0.6, 0,0]
fresnel.preview(scene)
```

[11]:

CHAPTER

NINETEEN

MULTIPLE GEOMETRIES

A **Scene** may consist of more than one geometry object. For fast performance, try to condense the scene down to as few geometries with as many primitives as possible. Multiple geometries allow for different materials to be applied to the same type of geometry and for different types of geometry in the same scene.

[1]:
```
import fresnel
scene = fresnel.Scene()
```

19.1 Create multiple geometries

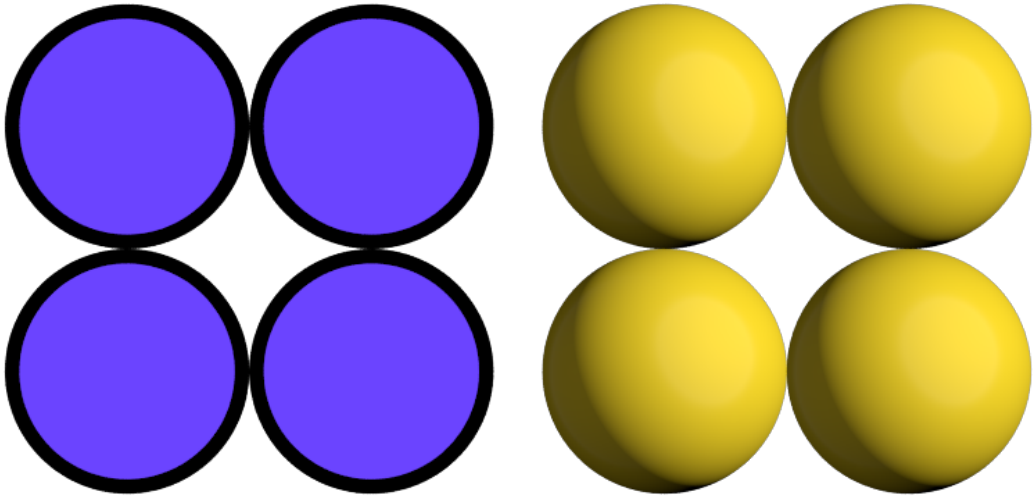
To create multiple geometries, instantiate several instances of the geometry class.

[2]:
```
geom1 = fresnel.geometry.Sphere(scene, position = [[-3.2, 1, 0], [-3.2, -1, 0], [-1.2,
↪ 1, 0], [-1.2, -1, 0]], radius=1.0)
geom1.material = fresnel.material.Material(solid=1.0, color=fresnel.color.linear([0.
↪42,0.267,1]))
geom1.outline_width = 0.12
```

[3]:
```
geom2 = fresnel.geometry.Sphere(scene, position = [[3.2, 1, 0], [3.2, -1, 0], [1.2, 1,
↪ 0], [1.2, -1, 0]], radius=1.0)
geom2.material = fresnel.material.Material(solid=0.0, color=fresnel.color.linear([1,0.
↪874,0.169]))
```

[4]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
fresnel.preview(scene, w=900, h=370)
```

[4]:



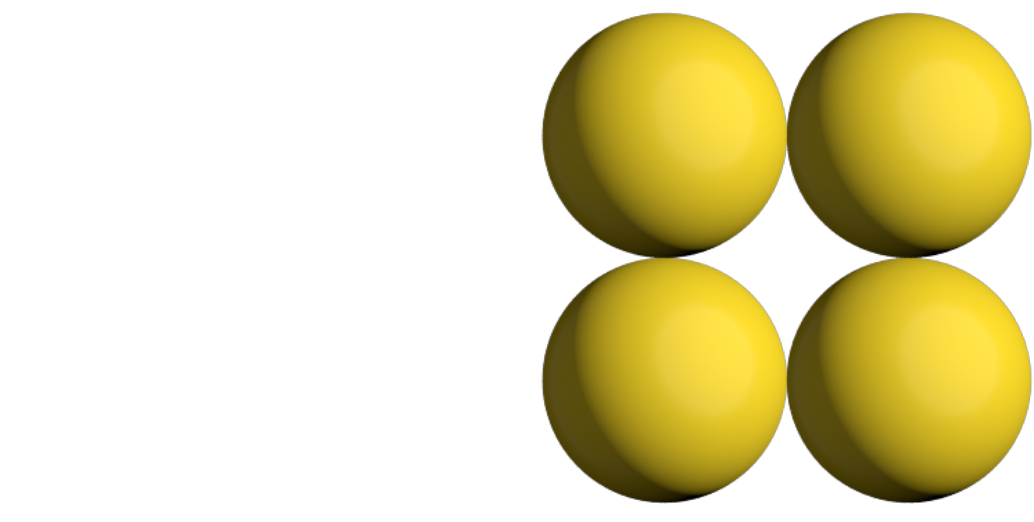
19.2 Disable geometries

disable a geometry to prevent it from appearing in the scene.

[5]:
```
geom1.disable()
```

[6]:
```
fresnel.preview(scene, w=900, h=370)
```

[6]:

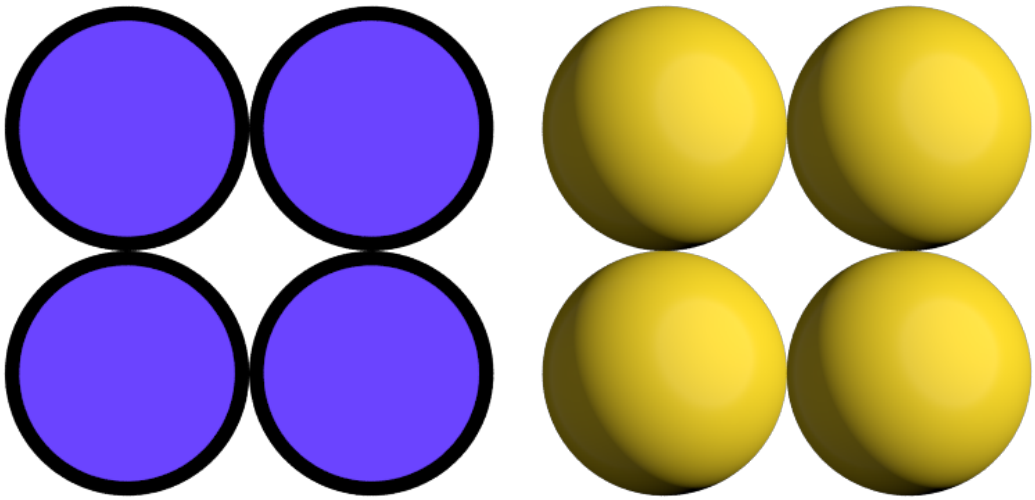


enable the geometry to make it appear again.

[7]:
```
geom1.enable()
```

[8]:
```
fresnel.preview(scene, w=900, h=370)
```

[8]:



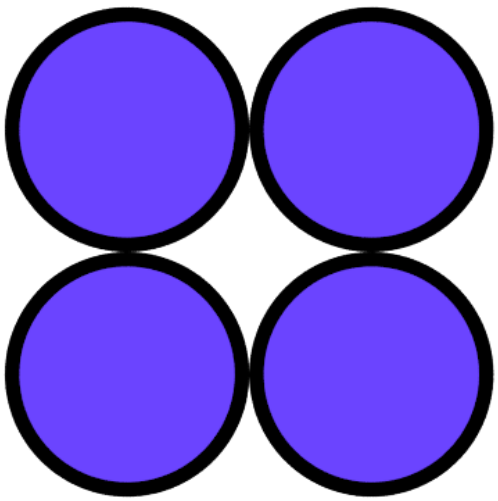
19.3 Remove geometry

Call **remove** to completely remove a geometry instance from the scene. It cannot be added back.

[9]:
```
geom2.remove()
```

[10]:
```
fresnel.preview(scene, w=900, h=370)
```

[10]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

TWENTY

DEVICES

Each **Scene** is attached to a specific **Device**. The **Device** controls what hardware the ray tracing executes on. **Scene** implicitly creates a default **Device** when you do not specify one.

20.1 The default device

The default device automatically selects **GPU ray tracing** if the *gpu module is compiled* and *there is at least one gpu present in the system* - otherwise it selects **CPU ray tracing**.

[1]:
```
import fresnel
device = fresnel.Device()
```

20.2 Query available execution modes

The `available_modes` static variable lists which execution modes are available. This will vary based on compile time options and whether there is a GPU present in the system.

[2]:
```
print(fresnel.Device.available_modes)
```

```
['gpu', 'cpu', 'auto']
```

`available_gpus` lists the GPUs available for rendering in the system.

[3]:
```
for g in fresnel.Device.available_gpus:
    print(g)
```

```
 [0]:        Quadro RTX 5000   48 SM_7.5 @ 1.82 GHz,  8198 MiB DRAM
```

20.3 Choose execution hardware

Explicitly manage a **Device** to control what hardware the ray tracing executes on. Converting the device to a string provides a short summary of the device configuration.

[4]:
```
gpu = fresnel.Device(mode='gpu')
print(gpu)
```

```
<fresnel.Device: Enabled OptiX devices:
 [0]:        Quadro RTX 5000   48 SM_7.5 @ 1.82 GHz,  8198 MiB DRAM
>
```

[5]:
```
cpu = fresnel.Device(mode='cpu')
print(cpu)
```

```
<fresnel.Device: All available CPU threads>
```

Set `n` to specify how many CPU threads or GPUs to use in parallel. By default, a device will use all available CPU cores or GPUs in the system.

[6]:
```
cpu_limit = fresnel.Device(mode='cpu', n=6)
print(cpu_limit)
```

```
<fresnel.Device: 6 CPU threads>
```

20.4 Attach a scene to a device

Each **Scene** must be attached to a device when created.

[7]:
```
scene_gpu = fresnel.Scene(device=gpu)
```

[8]:
```
scene_cpu = fresnel.Scene(device=cpu)
```

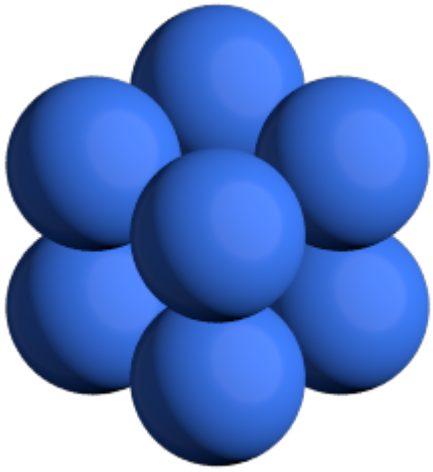
These two scenes have the same API, but different implementations.

[9]:
```
for scene in [scene_cpu, scene_gpu]:
    geometry = fresnel.geometry.Sphere(scene, N=8, radius=1.0)
    geometry.position[:] = [[1,1,1],
                            [1,1,-1],
                            [1,-1,1],
                            [1,-1,-1],
                            [-1,1,1],
                            [-1,1,-1],
                            [-1,-1,1],
                            [-1,-1,-1]]
    geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.
↪5,1]))
    scene.camera = fresnel.camera.Orthographic.fit(scene)
```

Rendered output is essentially identical from the two devices.

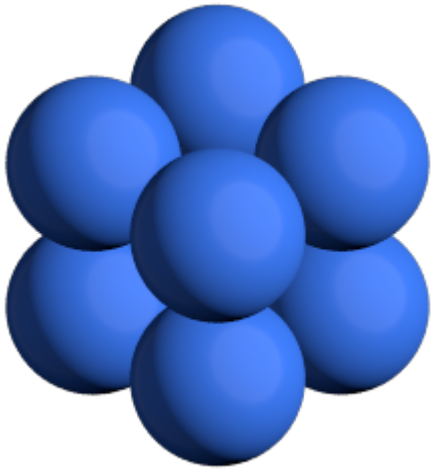
[10]:
```
fresnel.preview(scene_gpu, w=300, h=300)
```

[10]:



[11]:
```
fresnel.preview(scene_cpu, w=300, h=300)
```

[11]:



20.5 Memory consumption

Each **Device** consumes memory by itself. When maintaining multiple scenes, attach them all to the same device to reduce memory consumption.

[12]:
```
import math
scene2_gpu = fresnel.Scene(device=gpu)
position = []
for k in range(5):
    for i in range(5):
```

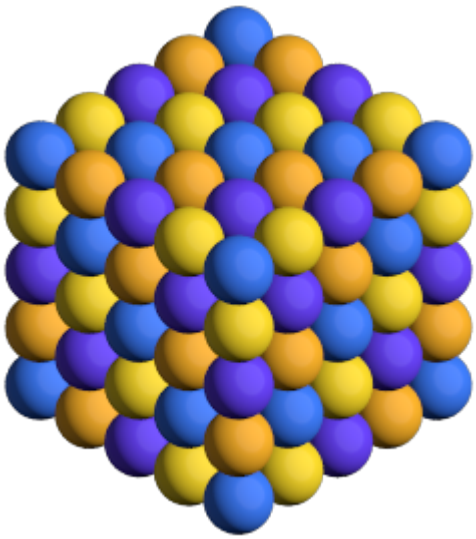
(continues on next page)

(continued from previous page)

```
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
geometry = fresnel.geometry.Sphere(scene2_gpu, position = position, radius=1.0)
geometry.color[::4] = fresnel.color.linear([0.25,0.5,1])
geometry.color[1::4] = fresnel.color.linear([1,0.714,0.169])
geometry.color[2::4] = fresnel.color.linear([0.42,0.267,1])
geometry.color[3::4] = fresnel.color.linear([1,0.874,0.169])
geometry.material = fresnel.material.Material(solid=0.0, primitive_color_mix=1.0)
scene2_gpu.camera = fresnel.camera.Orthographic.fit(scene2_gpu)
```

[13]:
```
fresnel.preview(scene2_gpu, w=300, h=300)
```

[13]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

TWENTYONE

TRACER METHODS

Most of the tutorials use **fresnel.preview()** and **fresnel.pathtrace()** to render output images. This is a convenience API, and there are cases where it is not appropriate. To render many frames, such as in a movie or interactive visualization, use a **Tracer** directly to avoid overhead.

[1]:
```
import fresnel
import math
from matplotlib import pyplot
%matplotlib inline
device = fresnel.Device()
scene = fresnel.Scene(device=device)
position = []
for i in range(6):
    position.append([2*math.cos(i*2*math.pi / 6), 2*math.sin(i*2*math.pi / 6), 0])

geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.material = fresnel.material.Material(solid=0.0, color=fresnel.color.
↪linear([1,0.874,0.169])*0.9)
geometry.outline_width = 0.12
scene.camera = fresnel.camera.Orthographic.fit(scene, view='front', margin=0.2)
```

21.1 Common Tracer operations

The **Tracer** must use the same device as the **Scenes** it renders. Each **Tracer** maintains an output image, and the width **w** and height **h** must be defined when the tracer is created.

[2]:
```
tracer = fresnel.tracer.Preview(device=device, w=300, h=300)
```

21.1.1 Rendering and accessing output images

The **render** method renders the output.

[3]:
```
out = tracer.render(scene)
```

The return value of **render** is a proxy reference to the internal image buffer of the **Tracer**. You can access with a **numpy** array like interface.

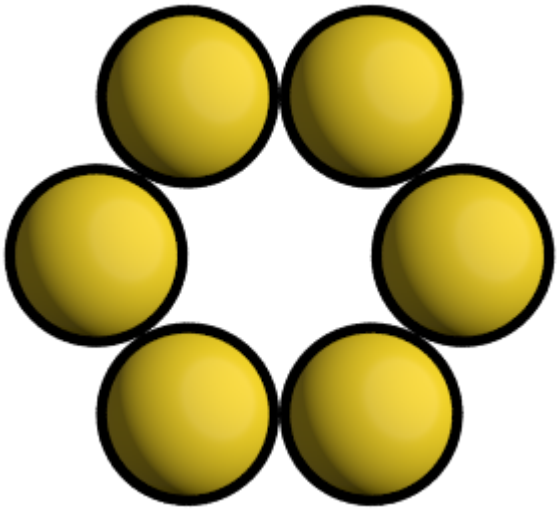
[4]:
```
out[100,100]
```

[4]:
```
array([139, 121,  21, 255], dtype=uint8)
```

The output object also provides an interface for **jupyter** to display the image.

[5]:
```
out
```

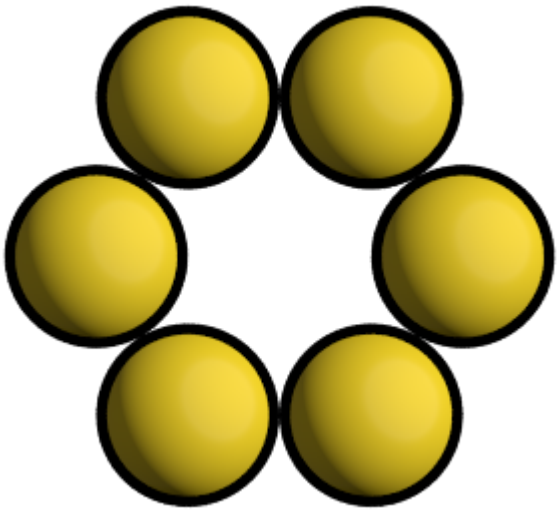
[5]:



tracer.output also accesses the output buffer.

[6]:
```
tracer.output
```

[6]:



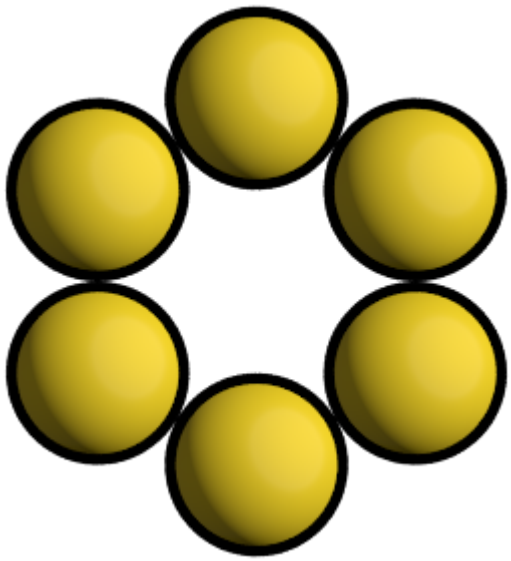
The tracer can render a modified scene without the initialization overhead.

[7]:
```
scene.camera.up = (1,0,0)
tracer.render(scene);
```

After rendering, existing references to the output buffer will access the newly rendered image.

[8]:
```
out
```

[8]:



21.1.2 Evaluate image exposure

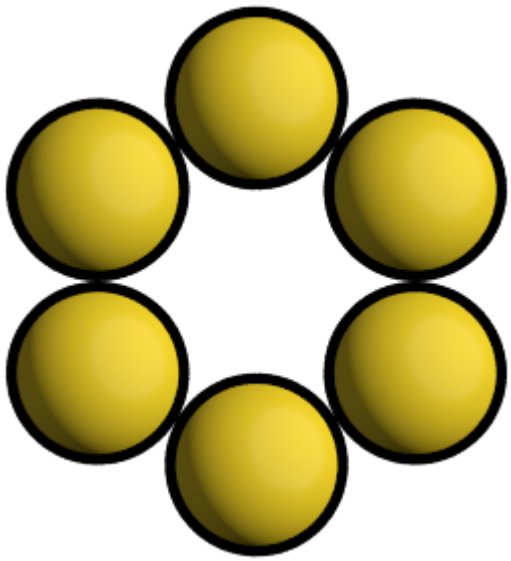
Tracer provides several methods to evaluate image exposure. Enable highlight warnings to flag overexposed pixels in the output image.

[9]:
```
tracer.enable_highlight_warning()
```

The test image is exposed correctly, there are no warning pixels.

[10]:
```
tracer.render(scene)
```

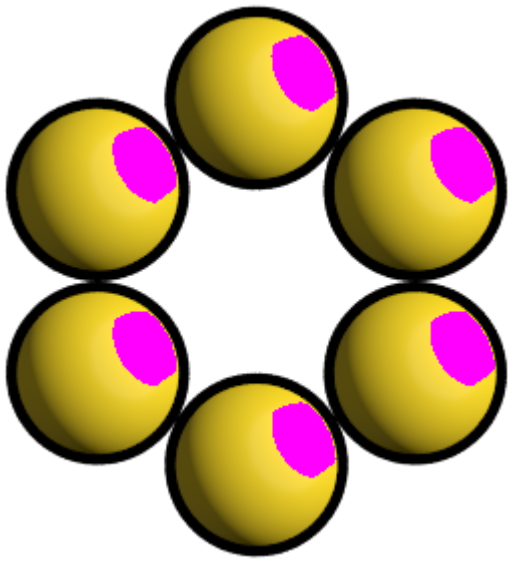
[10]:



Make the main light brighter to show the highlight warnings.

[11]:
```
scene.lights[0].color = (1.2, 1.2, 1.2)
tracer.render(scene)
```

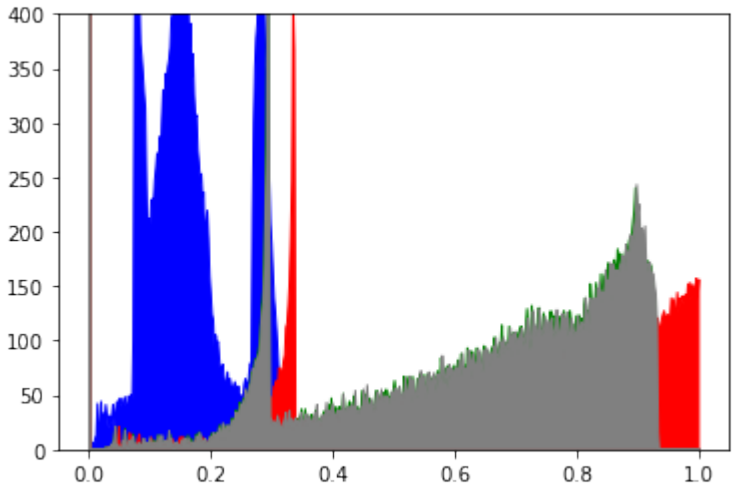
[11]:



Tracer can also compute the image histogram to evaluate image exposure.

[12]:
```
L, bins = tracer.histogram()
pyplot.fill_between(bins, L[:,3], color='blue');
pyplot.fill_between(bins, L[:,2], color='green');
pyplot.fill_between(bins, L[:,1], color='red');
pyplot.fill_between(bins, L[:,0], color='gray');
pyplot.axis(ymax=400, ymin=0)
```

[12]:
```
(-0.04794921875, 1.04990234375, 0.0, 400.0)
```



[13]:
```
tracer.disable_highlight_warning()
```

21.1.3 Resizing the output buffer

Call **resize** to set a new size for the output. When the image is resized, any existing rendered output is lost.

[14]:
```
tracer.resize(w=150, h=150)
```

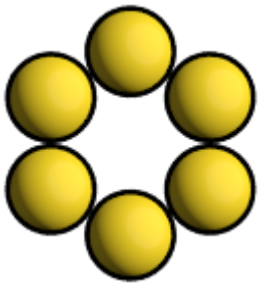
[15]:
```
tracer.output
```

[15]:

The next call to render will render into the new output size.

[16]:
```
tracer.render(scene)
```

[16]:



21.2 The Preview tracer

The **Preview** tracer renders output images quickly with approximate lighting effects.

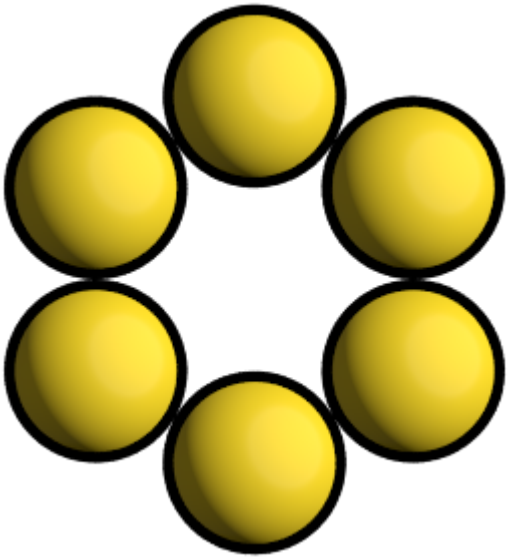
[17]:
```
tracer = fresnel.tracer.Preview(device=device, w=300, h=300)
```

A different random number seed will result in different jittered anti-aliasing samples.

[18]:
```
tracer.seed = 12
```

[19]:
```
tracer.render(scene)
```

[19]:

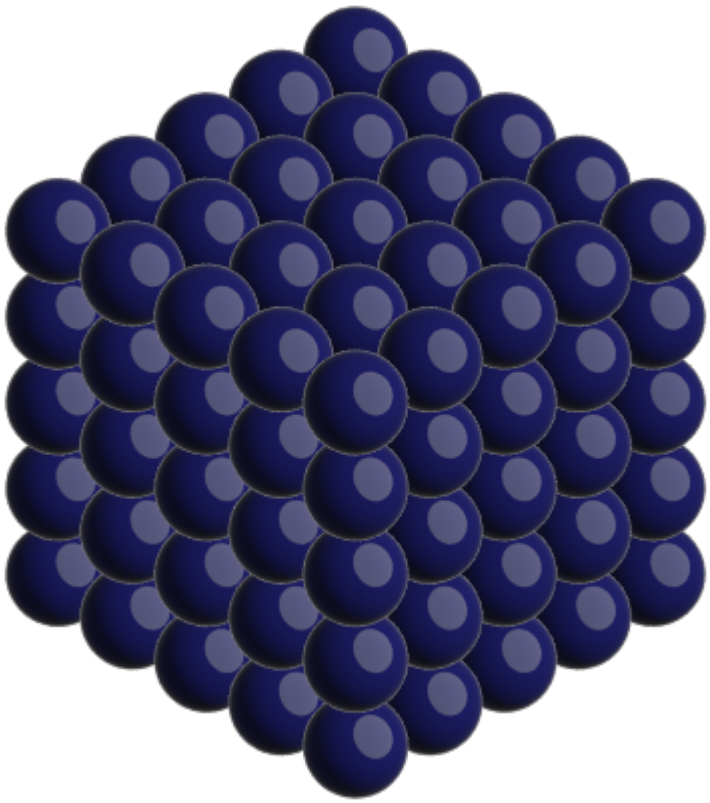


Here is a different scene rendered with the **Preview** tracer:

[20]:
```
position = []
for k in range(5):
    for i in range(5):
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
scene = fresnel.Scene(device)
scene.lights[1].theta = math.pi

geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.1,0.1,0.
↪4]),
                                              roughness=0.1,
                                              specular=1.0)
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

[21]:
```
tracer.resize(w=450, h=450)
tracer.aa_level = 3
tracer.render(scene)
```

[21]:



21.3 The Path tracer

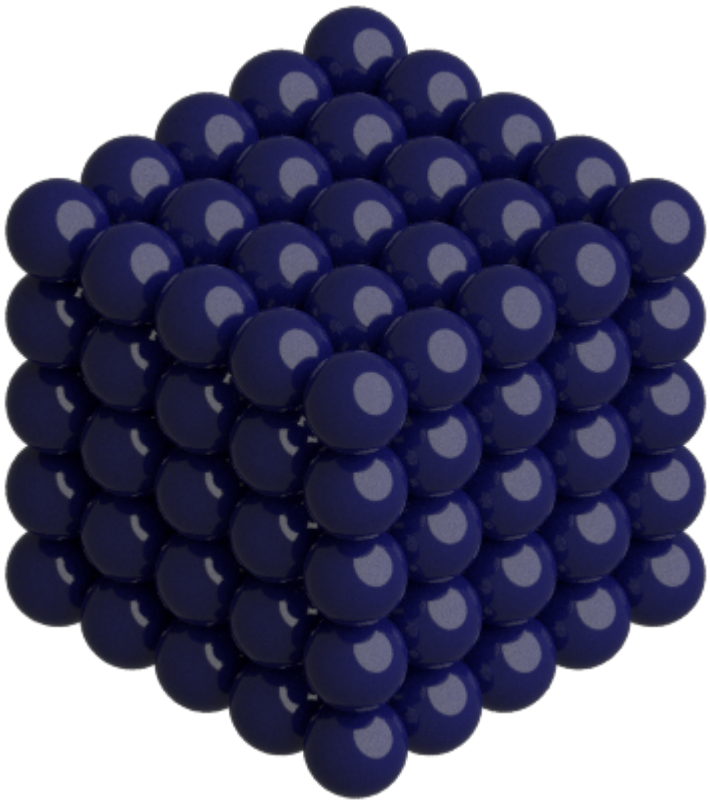
The **Path** tracer supports soft lighting, reflections, and other lighting effects.

Here is the same scene with the path tracer:

[22]:
```
path_tracer = fresnel.tracer.Path(device=device, w=450, h=450)
```

[23]:
```
path_tracer.sample(scene, samples=64, light_samples=40)
```

[23]:



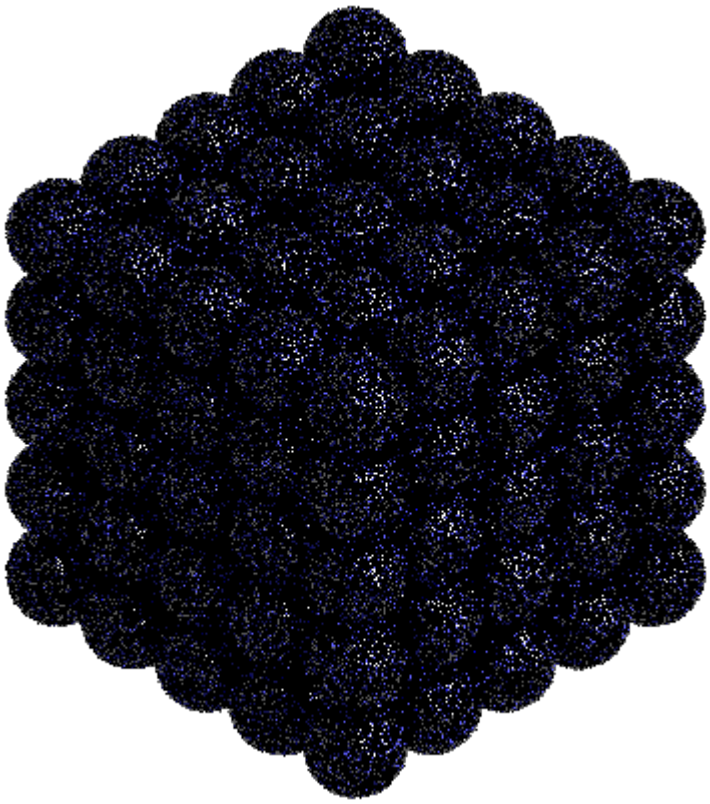
The **Path** tracer performs many independent samples and averages them together. **reset()** starts averaging a new image.

[24]:
```
path_tracer.reset()
```

render() accumulates a single sample into the resulting image.

[25]:
```
path_tracer.render(scene)
```

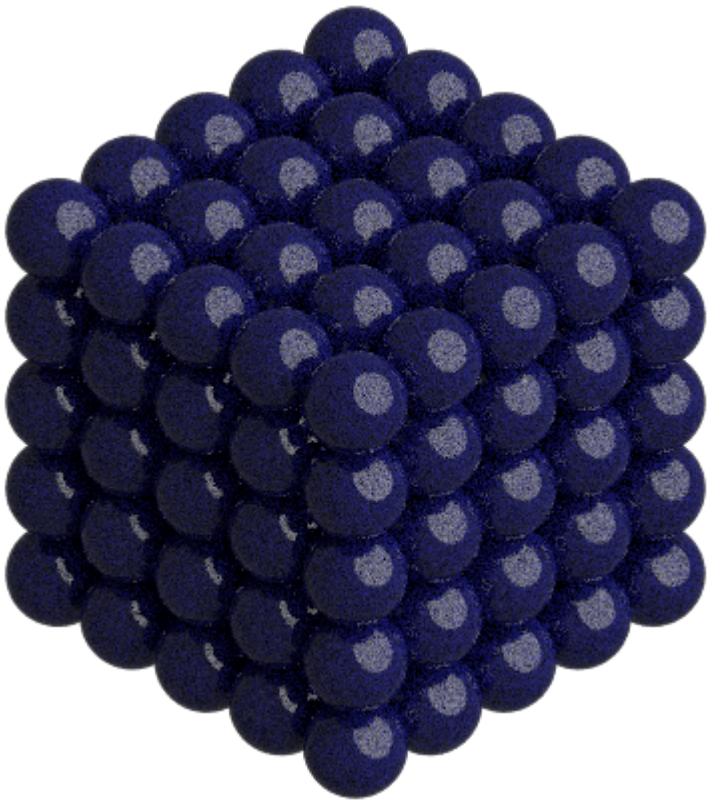
[25]:



The resulting image is noisy, average many samples together to obtain a clean image.

[26]:
```
for i in range(64):
    path_tracer.render(scene)

path_tracer.output
```

[26]:



This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

[1]:
```
import fresnel
import math
```

CHAPTER

TWENTYTWO

INTERACTIVE SCENE VIEW

fresnel provides a Qt widget to interactively display scenes rendered with the **path tracer**. This is implemented with the PySide2 library. Using jupyter support for this library, you can open an interactive window outside if the browser and interact with it from the jupyter notebook.

First, initialize jupyter's pyside2 integration.

[2]:
```
from PySide2 import QtCore
%gui qt
```

Then, import `fresnel.interact`. This **must** be done after `%gui qt`.

[ ]:
```
import fresnel.interact
```

Build a scene

[ ]:
```
position = []
for k in range(5):
    for i in range(5):
        for j in range(5):
            position.append([2*i, 2*j, 2*k])
scene = fresnel.Scene()
scene.lights[1].theta = math.pi

geometry = fresnel.geometry.Sphere(scene, position = position, radius=1.0)
geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.1,0.1,0.
↪8]),
                                              roughness=0.1,
                                              specular=1.0)
scene.camera = fresnel.camera.Orthographic.fit(scene)
```

22.1 SceneView widget

Create a `interact.SceneView` widget to visualize the scene.

[ ]:
```
view = fresnel.interact.SceneView(scene)
```

When the SceneView is the result of a cell, the windows shows and gets focus. In JupyterLab environments, you may need to use `view.show()`

[ ]:
```
view

# use view.show() if in JupyterLab
```

In the new window, you can click and drag to rotate the camera. Jupyter is still running so you can query changes to the window here. For example, after rotating the camera, inspect the new camera configuration:

[ ]:
```
repr(scene.camera)
```

After you change scene properties, call `setScene` to re-render the scene with the changes. For example: change the material color.

[ ]:
```
geometry.material.color = fresnel.color.linear([0.8,0.1,0.1])
view.setScene(scene)
```

This page was generated from a jupyter notebook. You can download and run the notebook locally from the fresnel-examples repository.

CHAPTER

TWENTYTHREE

RENDERING IMAGES IN MATPLOTLIB

Images rendered by **fresnel** can be converted to RGBA arrays for display with the **imshow** command in **matplotlib**. This example shows how to build subplots that display the geometries of the Platonic Solids.

[1]:
```
import numpy as np
import fresnel
import matplotlib
import matplotlib.pyplot as plt
```

[2]:
```
platonic_solid_vertices = {
    'Tetrahedron': [
        [0.0, 0.0, 0.612372],
        [-0.288675, -0.5, -0.204124],
        [-0.288675, 0.5, -0.204124],
        [0.57735, 0.0, -0.204124]],
    'Cube': [
        [-0.5, -0.5, -0.5],
        [-0.5, -0.5, 0.5],
        [-0.5, 0.5, -0.5],
        [-0.5, 0.5, 0.5],
        [0.5, -0.5, -0.5],
        [0.5, -0.5, 0.5],
        [0.5, 0.5, -0.5],
        [0.5, 0.5, 0.5]],
    'Octahedron': [
        [-0.707107, 0.0, 0.0],
        [0.0, 0.707107, 0.0],
        [0.0, 0.0, -0.707107],
        [0.0, 0.0, 0.707107],
        [0.0, -0.707107, 0.0],
        [0.707107, 0.0, 0.0]],
    'Dodecahedron': [
        [-1.37638, 0.0, 0.262866],
        [1.37638, 0.0, -0.262866],
        [-0.425325, -1.30902, 0.262866],
        [-0.425325, 1.30902, 0.262866],
        [1.11352, -0.809017, 0.262866],
        [1.11352, 0.809017, 0.262866],
        [-0.262866, -0.809017, 1.11352],
        [-0.262866, 0.809017, 1.11352],
        [-0.688191, -0.5, -1.11352],
        [-0.688191, 0.5, -1.11352],
        [0.688191, -0.5, 1.11352],
        [0.688191, 0.5, 1.11352],
```

(continues on next page)

(continued from previous page)

```
        [0.850651, 0.0, -1.11352],
        [-1.11352, -0.809017, -0.262866],
        [-1.11352, 0.809017, -0.262866],
        [-0.850651, 0.0, 1.11352],
        [0.262866, -0.809017, -1.11352],
        [0.262866, 0.809017, -1.11352],
        [0.425325, -1.30902, -0.262866],
        [0.425325, 1.30902, -0.262866]],
    'Icosahedron': [
        [0.0, 0.0, -0.951057],
        [0.0, 0.0, 0.951057],
        [-0.850651, 0.0, -0.425325],
        [0.850651, 0.0, 0.425325],
        [0.688191, -0.5, -0.425325],
        [0.688191, 0.5, -0.425325],
        [-0.688191, -0.5, 0.425325],
        [-0.688191, 0.5, 0.425325],
        [-0.262866, -0.809017, -0.425325],
        [-0.262866, 0.809017, -0.425325],
        [0.262866, -0.809017, 0.425325],
        [0.262866, 0.809017, 0.425325]],
    }
```

The `render` function returns a NumPy array of the output buffer, which can be passed directly to **imshow**.

[3]:
```
def render(shape, color_id=0):
    verts = platonic_solid_vertices[shape]
    scene = fresnel.Scene(fresnel.Device(mode='cpu'))
    scene.lights = fresnel.light.lightbox()
    poly_info = fresnel.util.convex_polyhedron_from_vertices(verts)
    cmap = matplotlib.cm.get_cmap('tab10')
    geometry = fresnel.geometry.ConvexPolyhedron(
        scene, poly_info,
        position = [0, 0, 0],
        orientation = [0.975528, 0.154508, -0.154508, -0.024472],
        outline_width = 0.015)
    geometry.material = fresnel.material.Material(
        color = fresnel.color.linear(cmap(color_id)[:3]),
        roughness = 0.1,
        specular = 1)
    geometry.outline_material = fresnel.material.Material(
        color = (0., 0., 0.),
        roughness = 0.1,
        metal = 1.0)

    scene.camera = fresnel.camera.Orthographic.fit(scene, view='front')
    out = fresnel.pathtrace(scene, samples=64,
                            light_samples=32,
                            w=200, h=200)
    return out[:]
```

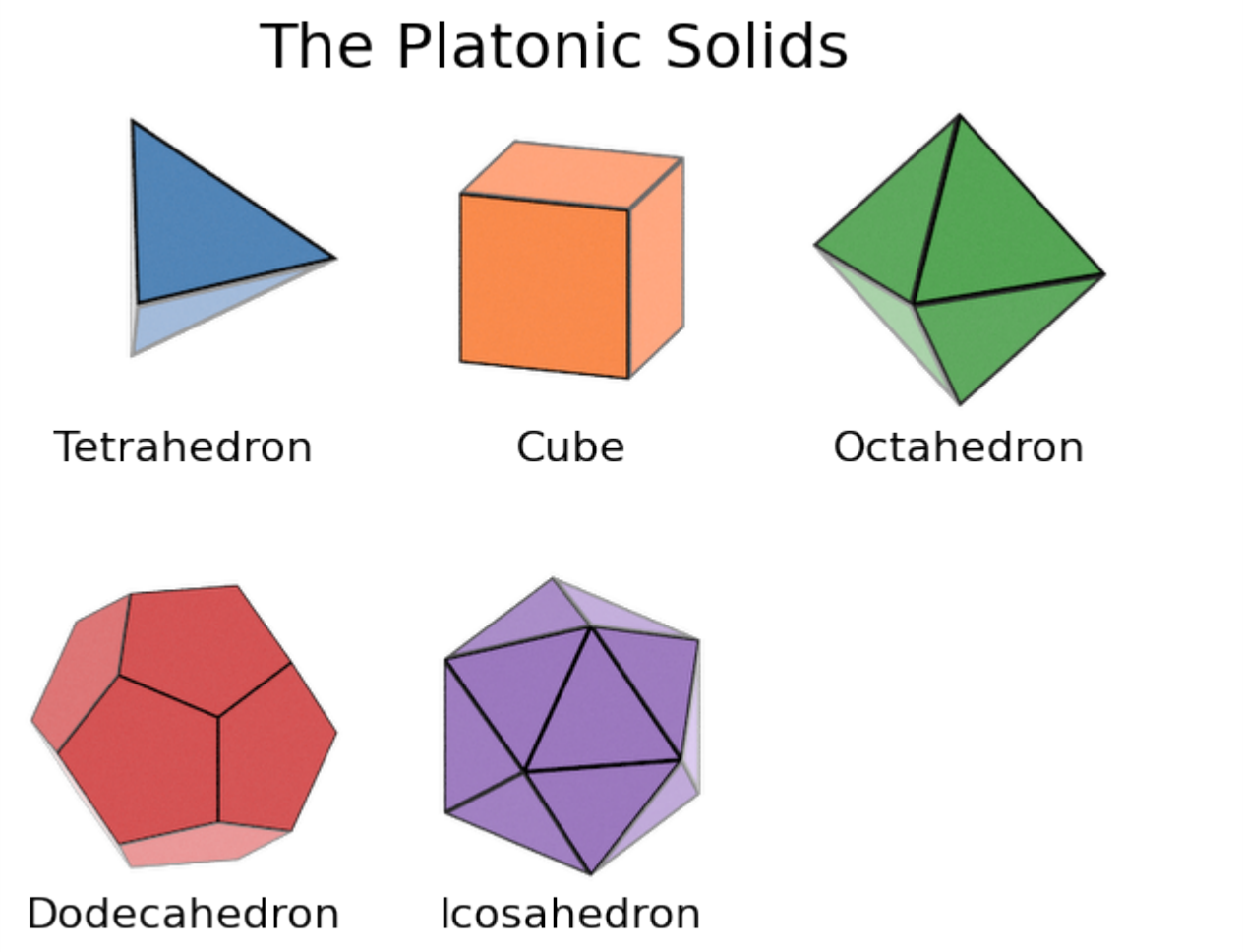
Below, **imshow** is used to render one scene in each subplot. Specifying an interpolation with **imshow** improves image quality.

[4]:
```
def show_shape(shape, location, color_id):
    ax = axs[location]
```

(continues on next page)

(continued from previous page)

```
    ax.imshow(render(shape, color_id), interpolation='lanczos')
    ax.set_xlabel(shape, fontsize=22)

fig, axs = plt.subplots(ncols=3, nrows=2, figsize=(10, 8))

show_shape('Tetrahedron', (0, 0), 0)
show_shape('Cube', (0, 1), 1)
show_shape('Octahedron', (0, 2), 2)
show_shape('Dodecahedron', (1, 0), 3)
show_shape('Icosahedron', (1, 1), 4)

for ax in axs.flatten():
    ax.set_xticks([])
    ax.set_yticks([])
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['left'].set_visible(False)

fig.suptitle('The Platonic Solids', y=0.92, fontsize=32)
plt.show()
```



CHAPTER

TWENTYFOUR

VISUALIZING GSD FILE

In this example, we will use `fresnel` to visualize a gsd file. We will color the particles & bonds by types, as well as visualize the simulation box.

We will need the gsd package to run this example.

[ ]:
```
import fresnel
import gsd.hoomd
import numpy as np
```

First we read in the `.gsd` file.

[ ]:
```
with gsd.hoomd.open(name="molecules.gsd", mode="rb") as gsd_file:
    snap = gsd_file[0]

box = snap.configuration.box
```

We want to color by particle type. We will color A types red, B types blue, and C types green.

[3]:
```
N = snap.particles.N
particle_types = snap.particles.typeid
colors = np.empty((N, 3))

# Color by typeid
colors[particle_types == 0] = fresnel.color.linear([.95, 0, 0]) # A type
colors[particle_types == 1] = fresnel.color.linear([0, .95, 0]) # B type
colors[particle_types == 2] = fresnel.color.linear([0, 0, .95]) # C type
```

[4]:
```
scene = fresnel.Scene()

# Spheres for every particle in the system
geometry = fresnel.geometry.Sphere(scene, N=N, radius=0.2)
geometry.position[:] = snap.particles.position
geometry.material = fresnel.material.Material(roughness=0.9)
geometry.outline_width = 0.05



# use color instead of material.color
geometry.material.primitive_color_mix = 1.0
geometry.color[:] = fresnel.color.linear(colors)
```

[5]:
```
# create box in fresnel
fresnel.geometry.Box(scene, box, box_radius=.07)
```

[5]:
```
<fresnel.geometry.Box at 0x7f5fbcb92670>
```

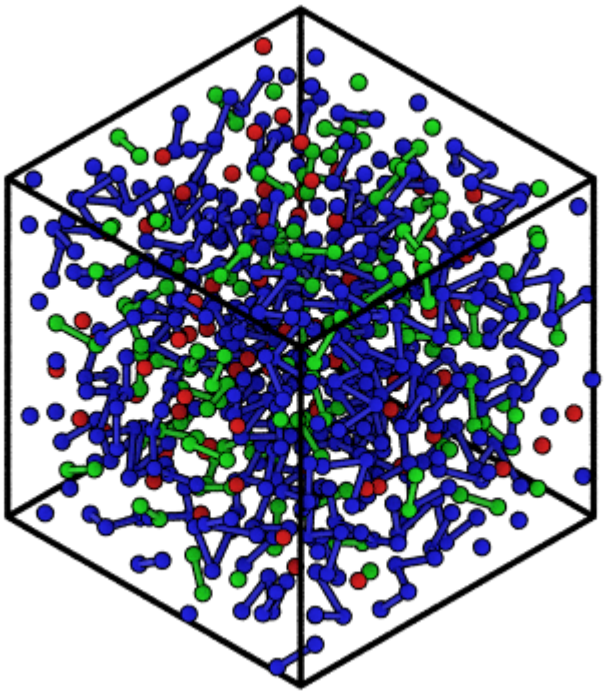
We will visualize bonds using cylinders, and color the bonds to match the particle types. To aid visualization, we will first remove any bonds that span the periodic boundary.

[6]:
```
all_bonds = np.stack(
    [
        snap.particles.position[snap.bonds.group[:, 0]],
        snap.particles.position[snap.bonds.group[:, 1]],
    ],
    axis=1,
)

# Use a distance cutoff (L/2) to filter bonds that span the periodic boundary
bond_distances = np.linalg.norm(all_bonds[:,0,:]-all_bonds[:,1,:], axis=1)

# This simple method will work for cubic cells
L = box[0]
bond_indices = np.where(bond_distances < L/2)[0]
filtered_bonds = all_bonds[bond_indices, :, :]

N_bonds = filtered_bonds.shape[0]
bonds = fresnel.geometry.Cylinder(scene, N=N_bonds)
bonds.material = fresnel.material.Material(roughness=0.5)
bonds.outline_width = 0.05



# Color by bond typeid
bond_ids = snap.bonds.typeid[bond_indices]
bond_colors = np.empty((N_bonds, 3))
bond_colors[bond_ids == 0] = fresnel.color.linear([0, .95, 0])  # B-B Bonds
bond_colors[bond_ids == 1] = fresnel.color.linear([0, 0, .95])  # C-C Bonds

bonds.material.primitive_color_mix = 1.0
bonds.points[:] = filtered_bonds

bonds.color[:] = np.stack(
    [fresnel.color.linear(bond_colors), fresnel.color.linear(bond_colors)], axis=1
)
bonds.radius[:] = [0.1] * N_bonds
```

Now that we have everything setup, we will render everything and apply some ring lighting conditions.

[7]:
```
scene.camera = fresnel.camera.Orthographic.fit(scene)
scene.lights = fresnel.light.lightbox()
fresnel.pathtrace(scene, light_samples=5)
```

[7]:



CHAPTER

TWENTYFIVE

FRESNEL

Overview

Device	Hardware device to use for ray tracing.
pathtrace	Path trace a scene.
preview	Preview a scene.
Scene	Content of the scene to ray trace.

Details

The fresnel ray tracing package.

class fresnel.**Device**(*mode='auto'*, *n=None*)

Hardware device to use for ray tracing.

Parameters

- **mode** (*str*) – Specify execution mode: Valid values are `auto`, `gpu`, and `cpu`.
- **n** (*int*) – Specify the number of cpu threads / GPUs this device will use. *None* will use all available threads / devices.

Device defines hardware device to use for ray tracing. *Scene* and *Tracer* instances must be attached to a *Device*. You may attach any number of scenes and tracers to a single *Device*.

See also:

Tutorials:

- *Devices*
- *Tracer methods*

When mode is `auto`, the default, *Device* will select GPU rendering if available and fall back on CPU rendering if not. Set mode to `gpu` or `cpu` to force a specific mode.

Important: By default (`n==None`), this device will use all available GPUs or CPU cores. Set *n* to the number of GPUs or CPU cores this device should use. When selecting *n* GPUs, the device selects the first *n* in the *available_gpus* list.

Tip: Use only a single *Device* to reduce memory consumption.

The static member *available_modes* lists which modes are available. For a mode to be available, the corresponding module must be enabled at compile time. Additionally, there must be at least one GPU present for the gpu mode to be available.

```
>>> fresnel.Device.available_modes
['gpu', 'cpu', 'auto']
```

available_gpus = []
Available GPUS.

Type list[str]

available_modes = []
Available execution modes.

Type list[str]

property mode
The active mode.

Type str

class fresnel.**Scene**(*device=None*, *camera=None*, *lights=None*)
Content of the scene to ray trace.

Parameters

- **device** (Device) – Device to use when rendering the scene.
- **camera** (camera.Camera) – Camera to view the scene. When None, defaults to:

```
camera.Orthographic(position=(0, 0, 100),
                    look_at=(0, 0, 0),
                    up=(0, 1, 0),
                    height=100)
```

- **lights** (*list[Light]*) – Lights to light the scene. When None, defaults to: light.rembrandt()

Scene defines the contents of the scene to be traced, including any number of *Geometry* objects, the *Camera*, the *background_color*, *background_alpha*, and *lights*.

Every *Scene* must be associated with a *Device*. For convenience, *Scene* creates a default *Device* when *device* is None.

See also:

Tutorials:

- *Introduction*
- *Scene properties*
- *Lighting setups*
- *Devices*

property background_alpha
Background alpha (opacity) in the range [0,1].

Type float

property background_color
Background color linear RGB.

Note: Use *fresnel.color.linear* to convert standard sRGB colors into the linear color space used by fresnel.

Type ((3,) numpy.ndarray of numpy.float32)

property camera
Camera view parameters.

Type *camera.Camera*

property device
Device this *Scene* is attached to.

Type *Device*

get_extents()
Get the extents of the scene.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of numpy.float32

property lights
Lights in the scene.

lights is a sequence of up to 4 directional lights that apply to the scene. Each light has a direction, color, and size.

Type list[*Light*]

fresnel.**pathtrace**(*scene*, *w=600*, *h=370*, *samples=64*, *light_samples=1*)
Path trace a scene.

Parameters

- **scene** (*Scene*) – Scene to render.
- **w** (*int*) – Output image width (in pixels).
- **h** (*int*) – Output image height (in pixels).
- **samples** (*int*) – Number of times to sample the pixels of the scene.
- **light_samples** (*int*) – Number of light samples to take for each pixel sample.

pathtrace() is a shortcut that renders output with *tracer.Path*.

fresnel.**preview**(*scene*, *w=600*, *h=370*, *anti_alias=True*)
Preview a scene.

Parameters

- **scene** (*Scene*) – Scene to render.
- **w** (*int*) – Output image width (in pixels).
- **h** (*int*) – Output image height (in pixels).
- **anti_alias** (*bool*) – Whether to perform anti-aliasing.

preview() is a shortcut that renders output with *tracer.Preview*.

Modules

25.1 fresnel.camera

Overview

Camera	Camera base class.
Orthographic	Orthographic camera.
Perspective	Perspective camera.

Details

Cameras.

class fresnel.camera.**Camera**(*_camera*)

Bases: object

Camera base class.

A *Camera* defines the view into the *Scene*.

Camera space is a coordinate system centered on the camera's position. Positive x points to the right in the image, positive y points up, and positive z points out of the screen. The visible area in the image plane is centered on *look_at* with the given *height*. The visible width is `height * aspect` where *aspect* is the aspect ratio determined by the resolution of the image in *Tracer* (`aspect = tracer.w / tracer.h`). *Camera* space shares units with *Scene* space.

Camera provides common methods and properties for all camera implementations. *Camera* cannot be used directly, use one of the subclasses.

See also:

- *Orthographic*
- *Perspective*

property basis

Orthonormal camera basis.

basis is computed from *position*, *look_at*, and *up*. The 3 vectors of the basis define the $+x$, $+y$, and $+z$ camera space directions in scene space.

Type ((3, 3) numpy.ndarray of numpy.float32)

property height

The height of the image plane.

Type float

property look_at

The point the camera looks at.

`position - look_at` defines the $+z$ direction in camera space.

Type ((3,) numpy.ndarray of numpy.float32)

property position
Camera position.

Type ((3,) numpy.ndarray of numpy.float32)

property up
A vector pointing toward the +*y* direction in camera space.

The component of *up* perpendicular to `look_at - position` defines the +*y* direction in camera space.

Type ((3,) numpy.ndarray of numpy.float32)

class fresnel.camera.**Orthographic**(*position*, *look_at*, *up*, *height*)
Bases: *fresnel.camera.Camera*

Orthographic camera.

Parameters

- **position** ((3,) numpy.ndarray of numpy.float32) – Camera position.
- **look_at** ((3,) numpy.ndarray of numpy.float32) – The point the camera looks at (the center of the focal plane).
- **up** ((3,) numpy.ndarray of numpy.float32) – A vector pointing toward the +*y* direction in camera space.
- **height** (*float*) – The height of the image plane.

An orthographic camera traces parallel rays from the image plane into the scene. Lines that are parallel in the *Scene* will remain parallel in the rendered image.

position is the center of the image plane in *Scene* space. *look_at* is the point in *Scene* space that will be in the center of the image. Together, these vectors define the image plane which is perpendicular to the line from *position* to *look_at*.

up is a vector in *Scene* space that defines the (+*y*) direction in the camera space). *up* does not need to be perpendicular to the line from *position* to *look_at*, but it must not be parallel to that line. *height* sets the height of the image sensor in *Scene* units. The width is `height * aspect` where *aspect* is the aspect ratio determined by the resolution of the image in *Tracer* (`aspect = tracer.w / tracer.h`).

Note: Only objects inside the rectangular cuboid defined by corners of the image sensor and the focal plane (extended to infinite height) will appear in the image.

Objects in front of the image plane will appear in the rendered image, objects behind the plane will not.

Tip: Place the camera *position* outside the geometry of the *Scene*. Decrease *height* to zoom in and increase *height* to zoom out.

---

classmethod fit(*scene*, *view='auto'*, *margin=0.05*)
Fit a camera to a *Scene*.

Create an orthographic camera that fits the entire height of the scene in the image plane.

Parameters

- **scene** (*Scene*) – Fit the camera to this scene.

- **view** (*str*) – Select view
- **margin** (*float*) – Fraction of extra space to leave on the top and bottom of the scene.

view may be 'auto', 'isometric', or 'front'.

The isometric view is an orthographic projection from a particular angle so that the x,y, and z directions are equal lengths. The front view is an orthographic projection where +x points to the right, +y points up and +z points out of the screen in the image plane. 'auto' automatically selects 'isometric' for 3D scenes and 'front' for 2D scenes.

class fresnel.camera.**Perspective**(*position*, *look_at*, *up*, *focal_length=0.5*, *focus_distance=10*, *f_stop=inf*, *height=0.24*)

Bases: *fresnel.camera.Camera*

Perspective camera.

Parameters

- **position** ((3,) numpy.ndarray of numpy.float32) – Camera position.
- **look_at** ((3,) numpy.ndarray of numpy.float32) – The point the camera looks at (the center of the focal plane).
- **up** ((3,) numpy.ndarray of numpy.float32) – A vector pointing toward the $+y$ direction in camera space.
- **focal_length** (*float*) – Focal length of the camera lens.
- **focus_distance** (*float*) – Distance to the focal plane.
- **f_stop** (*float*) – F-stop ratio for the lens.
- **height** (*float*) – The height of the image plane.

A perspective camera traces diverging rays from the camera position through the image plane into the scene. Lines that are parallel in the *Scene* will converge rendered image.

position is the center of projection *Scene* space. *look_at* is the point in *Scene* space that will be in the center of the image. Together, these vectors define the image plane which is perpendicular to the line from *position* to *look_at*.

up is a vector in *Scene* space that defines the ($+y$) direction in the camera space). *up* does not need to be perpendicular to the line from *position* to *look_at*, but it must not be parallel to that line.

Note: Only objects inside the rectangular pyramid defined by the position and corners of the image sensor (extended to infinite height) will appear in the image.

Perspective models an ideal camera system with a sensor and a thin lens. The sensor lies in the image plane and is the location where the pixels in the rendered image will be captured. *height* sets the height of the sensor in *Scene* units. The width is `height * aspect` where *aspect* is the aspect ratio determined by the resolution of the image in *Tracer* (`aspect = tracer.w / tracer.h`). *focal_length* sets the distance between *position* and the image plane.

Note: The camera *height* should be small relative to the objects in the *Scene* with those objects in front of the image plane. If the scene units are decimeters, the default *height* of 0.24 is 24 mm, the height of a 35 mm camera sensor.

Tip: There are two ways to zoom a perspective camera. 1) Move the position of the camera while keeping the focal length fixed. Photographers call this "zooming with your feet" and it maintains a fixed field of view. 2) Increase the *focal_length* to zoom in or decrease it to zoom out while keeping position fixed. This is the the equivalent of rotating the focal length setting on a zoom lens. Changing *focal_length* changes the field of view.

Like a digital camera, the *Perspective* camera must be focused. The focal plane is parallel to the image plane at a distance *focus_distance* from the camera *position*. Objects on the focal plane will be in sharp focus. Objects in front of and behind the plane will be out of focus. Out of focus areas in an image are called **bokeh** and can be used to draw the viewer's attention to the subject that is in clear focus. The space in front of and behind the focal plane that appears to be in focus is the **depth of field**. Set *f_stop* to control the amount of depth of field. Small, non-zero values will lead to very little depth of field and a value of *inf* will extend the depth of field to infinity.

Note: There are convenience methods to set the camera parameters:

- *focus_on* takes a point and computes the *focus_distance* to put that point on the focal plane.
- *depth_of_field* computes the *f_stop* needed to achieve a given depth of field.
- *vertical_field_of_view* computes the *focal_length* needed to achieve a given field of view angle.

Tip: The default *height* of 0.24 works well for scene objects that are size ~1 or larger. If the typical objects in your scene are much smaller, adjust *height* by an appropriate fraction.

property depth_of_field

The distance about the focal plane in sharp focus.

The area of sharp focus extends in front and behind the focal plane. The distance between the front and back areas of sharp focus is the depth of field.

The depth of field is a function of *focus_distance*, *focal_length*, *f_stop*, and *height*.

Setting *depth_of_field* computes *f_stop* to obtain the desired depth of field as a function of *focus_distance*, *focal_length*, and *height*.

Note: *depth_of_field* does not remain fixed after setting it.

Type float

property f_stop

F-stop ratio for the lens.

Set the aperture of the opening into the lens in f-stops. This sets the range of the scene that is in sharp focus. Smaller values of *f_stop* result in more background blur.

Tip: Use *depth_of_field* to set the range of sharp focus in *Scene* distance units.

Type float

property focal_length
Focal length of the camera lens.

The focal length relative to the image *height* sets the field of view. Given a fixed *height*, a larger *focal_length* gives a narrower field of view.

Tip: With the default height of 0.24, typical focal lengths range from .18 (wide angle) to 0.5 (normal) to 6.0 (telephoto).

See also:

vertical_field_of_view

Type float

property focus_distance
Distance to the focal plane.

The focus distance is the distance from the camera position to the center of focal plane.

Tip: Use *focus_on* to compute the focus distance to a particular point in the *Scene*.

Type float

property focus_on
A point in the focal plane.

The area of sharp focus extends in front and behind the focal plane.

The focal plane is a function of *focus_distance*, *position*, and *look_at*.

Setting *focus_on* computes *focus_distance* so that the given point is on the focal plane.

Note: *focus_on* does not remain fixed after setting it.

Type (3,) numpy.ndarray of numpy.float32)

property vertical_field_of_view
Vertical field of view.

The vertical field of view is the angle (in radians) that the camera covers in the +y direction. It is a function of *focal_length* and *height*.

Setting *vertical_field_of_view* computes *focal_length* to achieve the given field of view.

Note: *vertical_field_of_view* does not remain fixed after setting it.

Type float

25.2 fresnel.color

Overview

linear	Convert a sRGB color (or colors) into the linear space.

Details

Color utilities.

fresnel.color.**linear**(*color*)

Convert a sRGB color (or colors) into the linear space.

Standard tools for working with sRGB colors provide gamma corrected values. fresnel needs to perform calculations in a linear color space. This method converts from sRGB to the linear space. Use *linear()* when specifying material or particle colors with sRGB inputs (such as you find in a color picker).

linear() accepts RGBA input (such as from matplotlib's colors.to_rgba colormap method), but ignores the alpha channel and outputs an Nx3 array.

Parameters **color** ((3,), (4,), (N, 3), or (N, 4) numpy.ndarray of numpy.float32) – RGB or RGBA colors.

Color components are in the range [0,1].

Returns numpy.ndarray with the linearized color(s), same shape as color.

25.3 fresnel.geometry

Overview

Box	Box geometry.
ConvexPolyhedron	Convex polyhedron geometry.
Cylinder	Cylinder geometry.
Geometry	Geometry base class.
Mesh	Mesh geometry.
Polygon	Polygon geometry.
Sphere	Sphere geometry.

Details

Geometric primitives.

Geometry defines objects that are visible in a *Scene*. The base class *Geometry* provides common operations and properties. Instantiate specific geometry class to add objects to a *Scene*.

See also:

Tutorials:

- *Primitive properties*
- *Material properties*

- *Outline materials*
- *Multiple geometries*

class fresnel.geometry.**Box**(*scene*, *box*, *box_radius=0.5*, *box_color=[0, 0, 0]*)
Bases: *fresnel.geometry.Cylinder*

Box geometry.

Generate a triclinic box outline with *spherocylinders*. The geometry's material defaults to `material.Material(solid=1.0)`.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **box** ((1,), (3,), or (6,) numpy.ndarray of float32) – Box parameters.
- **box_radius** (*float*) – Radius of box edges.
- **box_color** ((3,) numpy.ndarray of float32) – Color of the box edges.

Note: A 1-element *box* array expands to a cube. A 3-element *box* array `[Lx, Ly, Lz]` expands to an orthorhobic cuboid, and a 6-element *box* array represents a fully triclinic box in the same format as GSD and HOOMD: `[Lx, Ly, Lz, xy, xz, yz]`.

See also:

Tutorials:

- *Box*
- *Visualizing GSD File*

Note: The Box class is constructed from *spherocylinders*, which can be modified individually. The convenience attributes `box_radius` and `box_color` can be used to set the thickness and color of the entire box.

---

property box
Box parameters.

Set *box* to update the shape of the box.

Type (1,), (3,), or (6,) numpy.ndarray of float32

property box_color
Color of the box edges.

Note: This property sets the color of the *material*.

Type (3,) numpy.ndarray of float32

property box_radius
Radius of box edges.

Type (float)

class fresnel.geometry.**ConvexPolyhedron**(*scene*, *polyhedron_info*, *position=(0, 0, 0)*, *orientation=(1, 0, 0, 0)*, *color=(0, 0, 0)*, *N=None*, *material=None*, *outline_material=None*, *outline_width=0.0*)

Bases: *fresnel.geometry.Geometry*

Convex polyhedron geometry.

Define a set of convex polyhedron primitives with individual positions, orientations, and colors.

A convex polyhedron is defined by *P* outward facing planes (origin and normal vector) and a radius that encompass the shape. Use *convex_polyhedron_from_vertices* to construct this from the convex hull of a set of vertices.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **polyhedron_info** (*Dict*) – A dictionary containing the face normals (face_normal), origins (face_origin), face colors (face_color), and the radius (radius)).
- **position** ((N, 3) numpy.ndarray of float32) – Position of each polyhedron instance.
- **orientation** ((N, 4) numpy.ndarray of float32) – Orientation of each polyhedron instance (as a quaternion).
- **color** ((N, 3) numpy.ndarray of float32) – Color of each polyhedron.
- **N** (*int*) – Number of spheres in the geometry. If None, determine *N* from *position*.
- **material** (material.Material) – Define how light interacts with the geometry. When None, defaults to material.Material().
- **outline_material** (material.Material) – Define how light interacts with the geometry's outline. When None, defaults to solid black material.Material(solid=1, color=(0,0,0)).
- **outline_width** (*float*) – Width of the outline in scene units.

See also:

Tutorials:

- *Convex polyhedron*

Hint: Avoid costly memory allocations and type conversions by specifying primitive properties in the appropriate array type.

---

property color

The color of each polyhedron.

Type (N, 3) *Array*

property color_by_face

Mix face colors with the per-polyhedron color.

Set to 0 to color particles by the per-particle *color*. Set to 1 to color faces by the per-face color. Set to a value between 0 and 1 to blend between the two colors.

Type float

get_extents()
Get the extents of the geometry.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of float32

property orientation
The orientation of each polyhedron.

Type (N, 4) *Array*

property position
The position of each polyhedron.

Type (N, 3) *Array*

class fresnel.geometry.**Cylinder**(*scene*, *points=((0, 0, 0), (0, 0, 0))*, *radius=0.5*, *color=(0, 0, 0)*, *N=None*, *material=None*, *outline_material=None*, *outline_width=0.0*)
Bases: *fresnel.geometry.Geometry*

Cylinder geometry.

Define a set of spherocylinder primitives with individual start and end positions, radii, and colors.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **points** ((N, 2, 3) numpy.ndarray of float32) – *N* cylinder start and end points.
- **radius** ((N,) numpy.ndarray of float32) – Radius of each cylinder.
- **color** ((N, 2, 3) numpy.ndarray of float32) – Color of each start and end point.
- **N** (*int*) – Number of cylinders in the geometry. When None, determine *N* from *points*.
- **material** (material.Material) – Define how light interacts with the geometry. When None, defaults to material.Material().
- **outline_material** (material.Material) – Define how light interacts with the geometry's outline. When None, defaults to solid black material.Material(solid=1, color=(0,0,0)).
- **outline_width** (*float*) – Width of the outline in scene units.

See also:

Tutorials:

- *Cylinder*

Hint: Avoid costly memory allocations and type conversions by specifying primitive properties in the appropriate array type.

Tip: When all cylinders are the same size or color, pass a single value and NumPy will broadcast it to all elements of the array.

---

property color
Color of each start and end point.

Type (N, 2, 3) *Array*

get_extents()
Get the extents of the geometry.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of float32

property points
The start and end points of the cylinders.

Type (N, 2, 3) *Array*

property radius
The radii of the cylinders.

Type (N,) *Array*

class fresnel.geometry.**Geometry**
Bases: object

Geometry base class.

Geometry provides operations and properties common to all geometry classes.

Note: You cannot instantiate a Geometry directly. Use one of the subclasses.

disable()
Disable the geometry.

When disabled, the geometry will not visible in the *Scene*.

See also:

enable

enable()
Enable the geometry.

When enabled, the geometry will be visible in the *Scene*.

See also:

disable

property material
Define how light interacts with the geometry.

Type *material.Material*

property outline_material
Define how light interacts with the geometry's outline.

Type *material.Material*

property outline_width
Width of the outline in scene units.

Type float

remove()
Remove the geometry from the scene.

After calling *remove*, the geometry is no longer part of the scene. It cannot be added back into the scene. Use *disable* and *enable* hide geometry reversibly.

class fresnel.geometry.**Mesh**(*scene*, *vertices*, *position=(0, 0, 0)*, *orientation=(1, 0, 0, 0)*, *color=(0, 0, 0)*, *N=None*, *material=None*, *outline_material=None*, *outline_width=0.0*)

Bases: *fresnel.geometry.Geometry*

Mesh geometry.

Define a set of triangle mesh primitives with individual positions, orientations, and colors.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **vertices** ((3T, 3) numpy.ndarray of float32) – Vertices of the triangles, listed contiguously. Vertices 0,1,2 define the first triangle, 3,4,5 define the second, and so on.
- **color** ((3T, 3) numpy.ndarray of float32) – Color of each vertex.
- **position** ((N, 3) numpy.ndarray of float32) – Position of each mesh instance.
- **orientation** ((N, 4) numpy.ndarray of float32) – Orientation of each mesh instance (as a quaternion).
- **N** (*int*) – Number of mesh instances in the geometry. If None, determine *N* from *position*.
- **material** (material.Material) – Define how light interacts with the geometry. When None, defaults to material.Material().
- **outline_material** (material.Material) – Define how light interacts with the geometry's outline. When None, defaults to solid black material.Material(solid=1, color=(0,0,0)).
- **outline_width** (*float*) – Width of the outline in scene units.

See also:

Tutorials:

- *Mesh*

Hint: Avoid costly memory allocations and type conversions by specifying primitive properties in the appropriate array type.

---

property color
The color of each sphere.

Type (N, 3) *Array*

get_extents()
Get the extents of the geometry.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of float32

property orientation
The orientation of each mesh.

Type (N, 4) *Array*

property position
The position of each mesh.

Type (N, 3) *Array*

class fresnel.geometry.**Polygon**(*scene*, *vertices*, *position=(0, 0)*, *angle=0*, *color=(0, 0, 0)*, *rounding_radius=0*, *N=None*, *material=None*, *outline_material=None*, *outline_width=0.0*)

Bases: *fresnel.geometry.Geometry*

Polygon geometry.

Define a set of simple polygon primitives in the xy plane with individual positions, rotation angles, and colors.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **vertices** ((N_vert, 2) numpy.ndarray of float32) – Polygon vertices.
- **position** ((N, 2) numpy.ndarray of float32) – Position of each polygon.
- **angle** ((N,) numpy.ndarray of float32) – Orientation angle of each polygon (in radians).
- **color** ((N, 3) numpy.ndarray of float32) – Color of each polygon.
- **rounding_radius** (*float*) – Rounding radius for spheropolygons.
- **N** (*int*) – Number of polygons in the geometry. If None, determine *N* from *position*.
- **material** (material.Material) – Define how light interacts with the geometry. When None, defaults to material.Material().
- **outline_material** (material.Material) – Define how light interacts with the geometry's outline. When None, defaults to solid black material.Material(solid=1, color=(0,0,0)).
- **outline_width** (*float*) – Width of the outline in scene units.

See also:

Tutorials:

- *Polygon*

Hint: Avoid costly memory allocations and type conversions by specifying primitive properties in the appropriate array type.

---

property angle
The rotation angle of each polygon (in radians).

Type (N,) *Array*

property color
The color of each polygon.

Type (N, 2, 3) *Array*

get_extents()
Get the extents of the geometry.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of float32

property position
The position of each polygon.

Type (N, 2) *Array*

class fresnel.geometry.**Sphere**(*scene*, *position=(0, 0, 0)*, *radius=0.5*, *color=(0, 0, 0)*, *N=None*, *material=None*, *outline_material=None*, *outline_width=0.0*)

Bases: *fresnel.geometry.Geometry*

Sphere geometry.

Define a set of sphere primitives with individual positions, radii, and colors.

Parameters

- **scene** (Scene) – Add the geometry to this scene.
- **position** ((N, 3) numpy.ndarray of float32) – Position of each sphere.
- **radius** ((N,) numpy.ndarray of float32) – Radius of each sphere.
- **color** ((N, 3) numpy.ndarray of float32) – Color of each sphere.
- **N** (*int*) – Number of spheres in the geometry. If None, determine *N* from *position*.
- **material** (material.Material) – Define how light interacts with the geometry. When None, defaults to material.Material().
- **outline_material** (material.Material) – Define how light interacts with the geometry's outline. When None, defaults to solid black material.Material(solid=1, color=(0,0,0)).
- **outline_width** (*float*) – Width of the outline in scene units.

See also:

Tutorials:

- *Sphere*

Hint: Avoid costly memory allocations and type conversions by specifying primitive properties in the appropriate array type.

Tip: When all spheres are the same size, pass a single value for *radius* and numpy will broadcast it to all elements of the array.

---

property color

The color of each sphere.

Type (N, 3) *Array*

get_extents()

Get the extents of the geometry.

Returns The lower left and upper right corners of the scene.

Return type (3,2) numpy.ndarray of float32

property position

The position of each sphere.

Type (N, 3) *Array*

property radius

The radius of each sphere.

Type (N,) *Array*

25.4 fresnel.interact

Overview

—

Details

25.5 fresnel.light

Overview

Light	A light.
butterfly	Create a butterfly lighting setup.
cloudy	Create a cloudy day lighting setup.
lightbox	Create a light box lighting setup.
loop	Create a loop lighting setup.
rembrandt	Create a Rembrandt lighting setup.
ring	Create a ring lighting setup.

Details

Lights.

Light objects in a *Scene*.

See also:

Tutorials:

- *Scene properties*
- *Lighting setups*

class fresnel.light.**Light**(*direction*, *color=(1, 1, 1)*, *theta=0.375*)

A light.

Parameters

- **direction** ((3,) numpy.ndarray of float32) – Vector direction the light points (in *Camera* space).
- **color** ((3,) numpy.ndarray of float32) – Linear RGB color and intensity of the light.
- **theta** (*float*) – Half angle of the cone that defines the area of the light (in radians).

In fresnel, lights are area lights at an infinite distance away in the given *direction* and are circular with the size set by *theta*. *color* sets the light intensity. A `(0.5, 0.5, 0.5)` light is twice as bright as `(0.25, 0.25, 0.25)`. Lights are normalized so that `color = (1, 1, 1)` should provide approximately a correct exposure. Color values greater than 1 are allowed.

Note: *direction* is in *Camera* space. A direction of (1, 0, 0) sets a light coming from the right in the image,

regardless of the camera position.

---

fresnel.light.**butterfly**()
Create a butterfly lighting setup.

The butterfly portrait lighting setup is front lighting with the key light (index 0) placed high above the camera and the fill light (index 1) below the camera.

Returns The lights.

Return type list[*Light*]

fresnel.light.**cloudy**()
Create a cloudy day lighting setup.

The cloudy lighting setup mimics a cloudy day. A strong light comes from all directions above. A weaker light comes from all directions below (accounting for light reflected off the ground). Use *Path* tracing for best results with this setup.

Returns The lights.

Return type list[*Light*]

fresnel.light.**lightbox**()
Create a light box lighting setup.

The light box lighting setup places a single area light that covers the top, bottom, left, and right. Use *Path* tracing for best results with this setup.

Returns The lights.

Return type list[*Light*]

fresnel.light.**loop**(*side='right'*)
Create a loop lighting setup.

The loop portrait lighting setup places the key light slightly to one side of the camera and slightly up (index 0). The fill light is on the other side of the camera at the level of the camera (index 1).

Parameters **side** (*str*) – 'right' or 'left' to choose which side of the camera to place the key light.

Returns The lights.

Return type list[*Light*]

fresnel.light.**rembrandt**(*side='right'*)
Create a Rembrandt lighting setup.

The Rembrandt portrait lighting setup places the key light 45 degrees to one side of the camera and slightly up (index 0). The fill light is on the other side of the camera at the level of the camera (index 1).

Parameters **side** (*str*) – 'right' or 'left' to choose which side of the camera to place the key light.

Returns The lights.

Return type list[*Light*]

fresnel.light.**ring**()
Create a ring lighting setup.

The ring lighting setup provides a strong front area light. This type of lighting is common in fashion photography. Use *Path* tracing for best results with this setup.

Returns The lights.

Return type list[*Light*]

25.6 fresnel.material

Overview

Material	Define material properties.

Details

Materials describe the way light interacts with surfaces.

class fresnel.material.**Material**(*solid=0*, *color=(0.9, 0.9, 0.9)*, *primitive_color_mix=0*, *roughness=0.3*, *specular=0.5*, *spec_trans=0*, *metal=0*)

Define material properties.

Materials control how light interacts with the geometry.

Parameters

- **solid** (*float*) – Set to 1 to pass through a solid color, regardless of the light and view angle.
- **color** ((3,) numpy.ndarray of float32)) – Linear material color.
- **primitive_color_mix** (*float*) – Set to 1 to use the color provided in the *Geometry*, 0 to use the color specified in the *Material*, or a value in the range [0, 1] to mix the two colors.
- **roughness** (*float*) – Roughness of the material. Nominally in the range [0.1, 1].
- **specular** (*float*) – Control the strength of the specular highlights. Nominally in the range [0, 1].
- **spec_trans** (*float*) – Control the amount of specular light transmission. In the range [0, 1].
- **metal** (*float*) – Set to 0 for dielectric material, or 1 for metal. Intermediate values interpolate between the two.

See also:

Tutorials:

- *Material properties*

Note: Colors are in the linearized color space. Use *fresnel.color.linear* to convert standard sRGB colors into this space.

---

property color

- Linear material color.

  Type ((3,) numpy.ndarray of float32))

property metal

Set to 0 for dielectric material, or 1 for metal.

Intermediate values interpolate between the two.

Type float

property primitive_color_mix
Mix the material color with the geometry.

Set to 1 to use the color provided in the *Geometry*, 0 to use the color specified in the *Material*, or a value in the range [0, 1] to mix the two colors.

Type float

property roughness
Roughness of the material.

Nominally in the range [0.1, 1].

Type float

property solid
Is this material a solid color?

Set to 1 to pass through a solid color, regardless of the light and view angle.

Type float

property spec_trans
Control the amount of specular light transmission.

In the range [0, 1].

Type float

property specular
Control the strength of the specular highlights.

Nominally in the range [0, 1].

Type float

25.7 fresnel.tracer

Overview

Path	Path tracer.
Preview	Preview ray tracer.
Tracer	Base class for all ray tracers.

Details

Ray tracers process a *Scene* and render output images.

- *Preview* generates a quick approximate render.
- *Path* which provides soft shadows, reflections, and other effects.

See also:

Tutorials:

- *Introduction*
- *Tracer methods*

class fresnel.tracer.**Path**(*device*, *w*, *h*)
Bases: *fresnel.tracer.Tracer*

Path tracer.

Parameters

- **device** (*Device*) – Device to use.
- **w** (*int*) – Output image width.
- **h** (*int*) – Output image height.

The path tracer applies advanced lighting effects, including soft shadows, reflections, and depth of field. It operates by Monte Carlo sampling. Each call to *render* performs one sample per pixel. The *output* image is the mean of all the samples. Many samples are required to produce a smooth image. *sample* provides a convenience API to make many samples with a single call.

reset()
Clear the output buffer.

Start sampling a new image. Increment the random number seed so that the new image is statistically independent from the previous.

sample(*scene*, *samples*, *reset=True*, *light_samples=1*)
Sample the image.

Parameters

- **scene** (*Scene*) – The scene to render.
- **samples** (*int*) – The number of samples to take per pixel.
- **reset** (*bool*) – When True, call *reset* before sampling
- **light_samples** (*int*) – The number of light samples per primary camera ray.

As an unbiased renderer, the sampling noise will scale as $\frac{1}{\sqrt{}}$

Returns
A reference to the current *output* buffer.
Return type
ImageArray

Note: When *reset* is `False`, subsequent calls to *sample* will continue to add samples to the current output image. Use the same number of light samples when sampling an image in this way.

---

class fresnel.tracer.**Preview**(*device*, *w*, *h*, *anti_alias=True*)
Bases: *fresnel.tracer.Tracer*

Preview ray tracer.

Parameters

- **device** (*Device*) – Device to use.
- **w** (*int*) – Output image width.
- **h** (*int*) – Output image height.
- **anti_alias** (*bool*) – Whether to perform anti-aliasing. If True, uses an 64 samples.

Overview

The *Preview* tracer produces a preview of the scene quickly. It approximates the effect of light on materials. The output of the *Preview* tracer will look very similar to that from the *Path* tracer, but will miss soft shadows, reflection, transmittance, depth of field and other effects.

Anti-aliasing

The default value of *anti_alias* is True to smooth sharp edges in the image. The anti-aliasing level corresponds to `aa_level=3` in fresnel versions up to 0.11.0. Different *seed* values will result in different output images.

property anti_alias
Whether to perform anti-aliasing.

Type bool

class fresnel.tracer.**Tracer**
Bases: object

Base class for all ray tracers.

Tracer provides operations common to all ray tracer classes.

Each *Tracer* instance stores a pixel output buffer. When you *render* a *Scene*, the *output* is updated.

Note: You cannot instantiate *Tracer* directly. Use one of the subclasses.

disable_highlight_warning()
Disable the highlight clipping warnings.

enable_highlight_warning(*color=(1, 0, 1)*)
Enable highlight clipping warnings.

When a pixel in the rendered image is too bright to represent, make that pixel the given *color* to flag the problem to the user.

Parameters **color** (*tuple*) – Color to make the highlight warnings.

histogram()
Compute a histogram of the image.

The histogram is computed as a lightness in the sRGB color space. The histogram is computed only over the visible pixels in the image, fully transparent pixels are ignored. The returned histogram is nbins x 4, the first column contains the lightness histogram and the next 3 contain R,B, and G channel histograms respectively.

Returns (histogram, bin_positions).

property linear_output
Reference to the current output buffer in linear color space.

Note: The output buffer is modified by *render* and *resize*.

Type *Array*

property output
Reference to the current output buffer.

Note: The output buffer is modified by *render* and *resize*.

Type *ImageArray*

render(*scene*)
Render a scene.

Parameters **scene** (*Scene*) – The scene to render.

Returns A reference to the current output buffer as a *fresnel.util.ImageArray*.

Render the given scene and write the resulting pixels into the output buffer.

resize(*w*, *h*)
Resize the output buffer.

Parameters

- **w** (*int*) – New output buffer width.
- **h** (*int*) – New output buffer height.

Warning: *resize* clears the output buffer.

property seed
Random number seed.

Type int

25.8 fresnel.util

Overview

Array	Access fresnel memory buffers.
convex_polyhedron_from_vertices	Make a convex polyhedron from vertices.
ImageArray	Access fresnel images.

Details

Utilities.

class fresnel.util.**Array**(*buf*, *geom*)
Bases: object

Access fresnel memory buffers.

Array provides a python interface to access the internal data of memory buffers stored and managed by fresnel. You can access a *Array* as if it were a numpy.ndarray (with limited operations). Below, *slice* is a slice or array indexing mechanic that **numpy** understands.

Writing

Write to an array with `array[slice] = v` where *v* is `numpy.ndarray`, `list`, or scalar value to broadcast. When *v* is a *contiguous* `numpy.ndarray` of the same data type, the data is copied directly from *v* into the internal buffer. Otherwise, it is converted to a `numpy.ndarray` before copying.

Reading

Read from an array with `v = array[slice]`. This returns a **copy** of the data as a `numpy.ndarray` each time it is called.

shape
Dimensions of the array.

Type tuple[int, [int]]

dtype
Numpy data type

class `fresnel.util.`**ImageArray**(*buf*, *geom*)
Bases: *fresnel.util.Array*

Access fresnel images.

Provide *Array* functionality withsome additional convenience methods specific to working with images. Images are represented as `(W, H, 4)` `numpy.ndarray` of `uint8` values in **RGBA** format.

When a *ImageArray* is the result of an image in a Jupyter notebook cell, Jupyter will display the image.

`fresnel.util.`**convex_polyhedron_from_vertices**(*vertices*)
Make a convex polyhedron from vertices.

Parameters **vertices** ((3,) `numpy.ndarray` of `float32`) – Vertices of the polyhedron.

Returns

Convex hull of *vertices* in a format used by *ConvexPolyhedron*.

The dictionary contains the keys `face_origin`, `face_normal`, `face_color`, and `radius`.

Return type dict

The dictionary can be used directly to draw a polyhedron from its vertices:

```
scene = fresnel.Scene()
polyhedron = fresnel.util.convex_polyhedron_from_vertices(vertices)
geometry = fresnel.geometry.ConvexPolyhedron(scene,
                                             polyhedron,
                                             position=[0, 0, 0],
                                             orientation=[1, 0, 0, 0])
```

25.9 fresnel.version

Version and build information.

`fresnel.version.`**`version`**
fresnel package version, following semantic versioning.

Type str

CHAPTER

TWENTYSIX

CODE STYLE

All code in fresnel must follow a consistent style to ensure readability. We provide configuration files for linters (specified below) so that developers can automatically validate and format files.

26.1 Python

Python code in GSD should follow PEP8 with the formatting performed by yapf (configuration in `setup.cfg`). Code should pass all **flake8** tests and formatted by **yapf**.

26.1.1 Tools

- Linter: flake8
  - With these plugins:
    - pep8-naming
    - flake8-docstrings
    - flake8-rst-docstrings
  - Run: `flake8` to see a list of linter violations.
- Autoformatter: yapf
  - Run: `yapf -d -r .` to see needed style changes.
  - Run: `yapf -i file.py` to apply style changes to a whole file, or use your IDE to apply **yapf** to a selection.

26.1.2 Documentation

Python code should be documented with docstrings and added to the Sphinx documentation index in `doc/`. Docstrings should follow Google style formatting for use in Napoleon.

26.2 C++/CUDA

- Style is set by clang-format >= 10
  - Whitesmith's indentation style.
  - 100 character line width.
  - Indent only with spaces.
  - 4 spaces per indent level.
  - See `.clang-format` for the full **clang-format** configuration.
- Naming conventions:
  - Namespaces: All lowercase `somenamespace`
  - Class names: `UpperCamelCase`
  - Methods: `lowerCamelCase`
  - Member variables: `m_` prefix followed by lowercase with words separated by underscores `m_member_variable`
  - Constants: all upper-case with words separated by underscores `SOME_CONSTANT`
  - Functions: `lowerCamelCase`

26.2.1 Tools

- Autoformatter: clang-format.
  - Run: `./run-clang-format.py -r .` to see needed changes.
  - Run: `clang-format -i file.c` to apply the changes.

26.2.2 Documentation

Documentation comments should be in Javadoc format and precede the item they document for compatibility with Doxygen and most source code editors. Multi-line documentation comment blocks start with `/**` and single line ones start with `///`.

26.3 Other file types

Use your best judgment and follow existing patterns when styling CMake and other files types. The following general guidelines apply:

- 100 character line width.
- 4 spaces per indent level.
- 4 space indent.

26.4 Editor configuration

Visual Studio Code users: Open the provided workspace file (`fresnel.code-workspace`) which provides configuration settings for these style guidelines.

CHAPTER

TWENTYSEVEN

LICENSE

```
Fresnel Open Source Software License Copyright (c) 2016-2021 The Regents of
the University of Michigan All rights reserved.

Fresnel may contain modifications ("Contributions") provided, and to which
copyright is held, by various Contributors who have granted The Regents of the
University of Michigan the right to modify and/or distribute such Contributions.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
   may be used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER

TWENTYEIGHT

CREDITS

The following people contributed to **fresnel**.

- Joshua A. Anderson, University of Michigan
- Bradley Dice, University of Michigan
- Jens Glaser, University of Michigan
- Tim Moore, University of Michigan
- Vyas Ramasubramani, University of Michigan
- Bryan VanSaders, University of Michigan
- Mike Henry, Boise State University
- Jenny Fothergill, Boise State University
- Corwin Kerr, University of Michigan

28.1 Libraries

Fresnel links to the following libraries:

28.1.1 Python

Python is used under the Python license (http://www.python.org/psf/license/).

28.1.2 Embree

Embree is used under the Apache License, 2.0:

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.
```

(continues on next page)

(continued from previous page)

```
   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(continues on next page)

(continued from previous page)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
```

(continues on next page)

(continued from previous page)

```
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
```

(continues on next page)

(continued from previous page)

```
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

28.1.3 pybind11

pybind11 is used under the BSD 3-clause license:

```
Copyright (c) 2016 Wenzel Jakob <wenzel.jakob@epfl.ch>, All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or
upgrades to the features, functionality or performance of the source code
("Enhancements") to anyone; however, if you choose to make your Enhancements
```

(continues on next page)

(continued from previous page)

```
available either publicly, or directly to the author of this software, without
imposing a separate written license agreement for such Enhancements, then you
hereby grant the following license: a non-exclusive, royalty-free perpetual
license to install, use, modify, prepare derivative works, incorporate into
other computer software, distribute, and sublicense such enhancements or
derivative works thereof, in binary and source code form.
```

28.1.4 OptiX SDK

Portions of the OptiX SDK are used under the following license:

```
Copyright (c) 2016, NVIDIA CORPORATION. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of NVIDIA CORPORATION nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

28.1.5 Random123

Random123 is used to generate random numbers and is used under the following license:

```
Copyright 2010-2012, D. E. Shaw Research.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions, and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions, and the following disclaimer in the
```

(continues on next page)

(continued from previous page)

```
  documentation and/or other materials provided with the distribution.

* Neither the name of D. E. Shaw Research nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

28.1.6 Intel TBB

Intel's threaded building blocks library provides support for parallel execution on CPUS and is used under the following license:

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
```

(continues on next page)

(continued from previous page)

```
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
```

(continues on next page)

(continued from previous page)

```
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
```

(continues on next page)

(continued from previous page)

```
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

28.1.7 qhull

Qhull is used under the following license:

```
                    Qhull, Copyright (c) 1993-2019

                            C.B. Barber
                           Arlington, MA

                               and

       The National Science and Technology Research Center for
        Computation and Visualization of Geometric Structures
                            (The Geometry Center)
                          University of Minnesota

                          email: qhull@qhull.org

This software includes Qhull from C.B. Barber and The Geometry Center.
Qhull is copyrighted as noted above.  Qhull is free software and may
be obtained via http from www.qhull.org.  It may be freely copied, modified,
and redistributed under the following conditions:

1. All copyright notices must remain intact in all files.

2. A copy of this text file must be distributed along with any copies
   of Qhull that you redistribute; this includes copies that you have
   modified, or copies of programs or other software products that
   include Qhull.

3. If you modify Qhull, you must include a notice giving the
   name of the person performing the modification, the date of
   modification, and the reason for such modification.

4. When distributing modified versions of Qhull, or other software
   products that include Qhull, you must provide notice that the original
   source code may be obtained as noted above.

5. There is no warranty or other guarantee of fitness for Qhull, it is
   provided solely "as is".  Bug reports or fixes may be sent to
   qhull_bug@qhull.org; the authors may or may not act on them as
they desire.
```

CHAPTER

TWENTYNINE

INDEX

- genindex
- modindex

PYTHON MODULE INDEX

f

INDEX

H

I

L

M

O

P

R

S

T

U

V