Fresnel Documentation

Release 0.6.0

The Regents of the University of Michigan

Jul 09, 2018

Getting started

1 Samples **3**

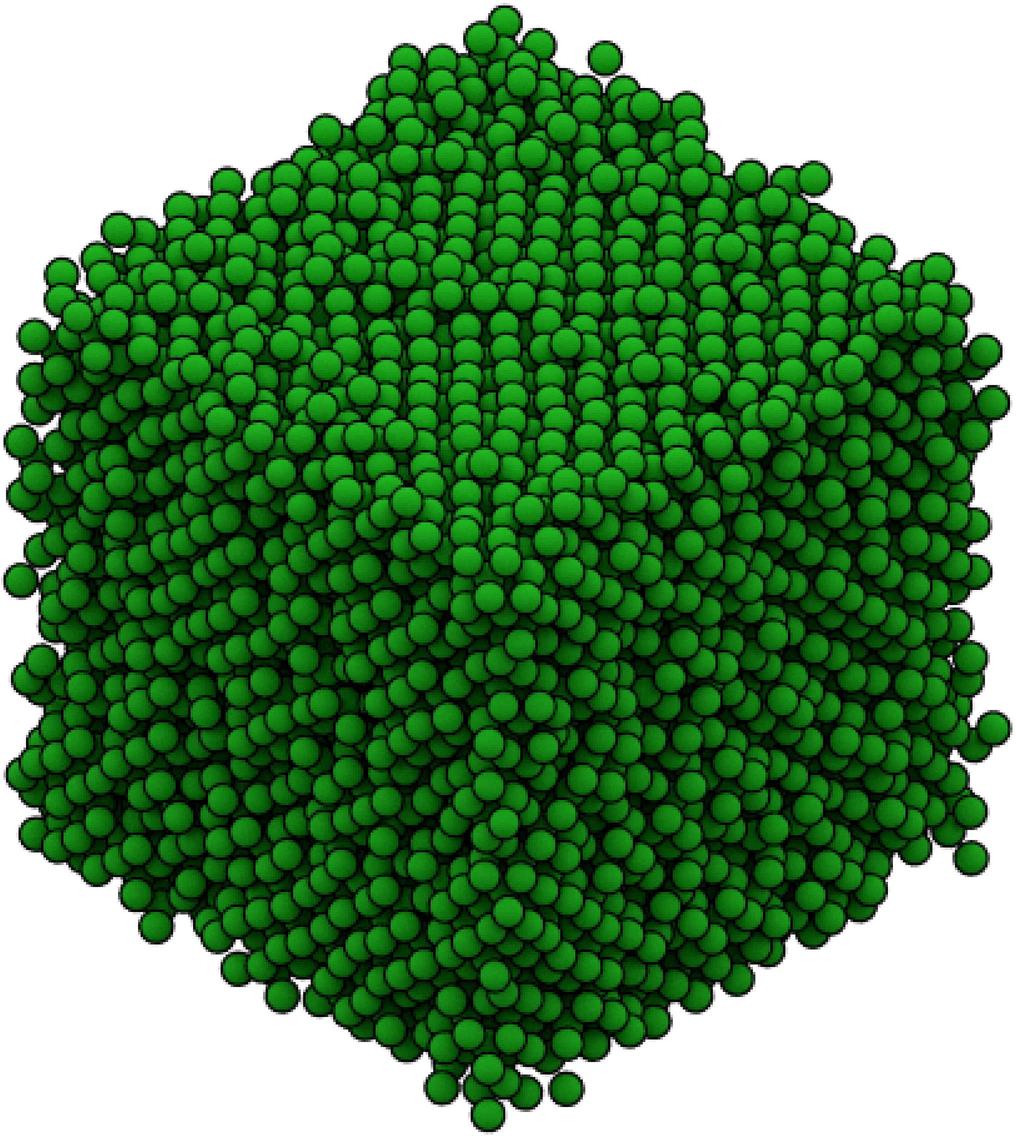
Python Module Index **109**

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 controlled by intuitive parameters (like *roughness*, *specular*, and *metal*).

CHAPTER 1

Samples

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

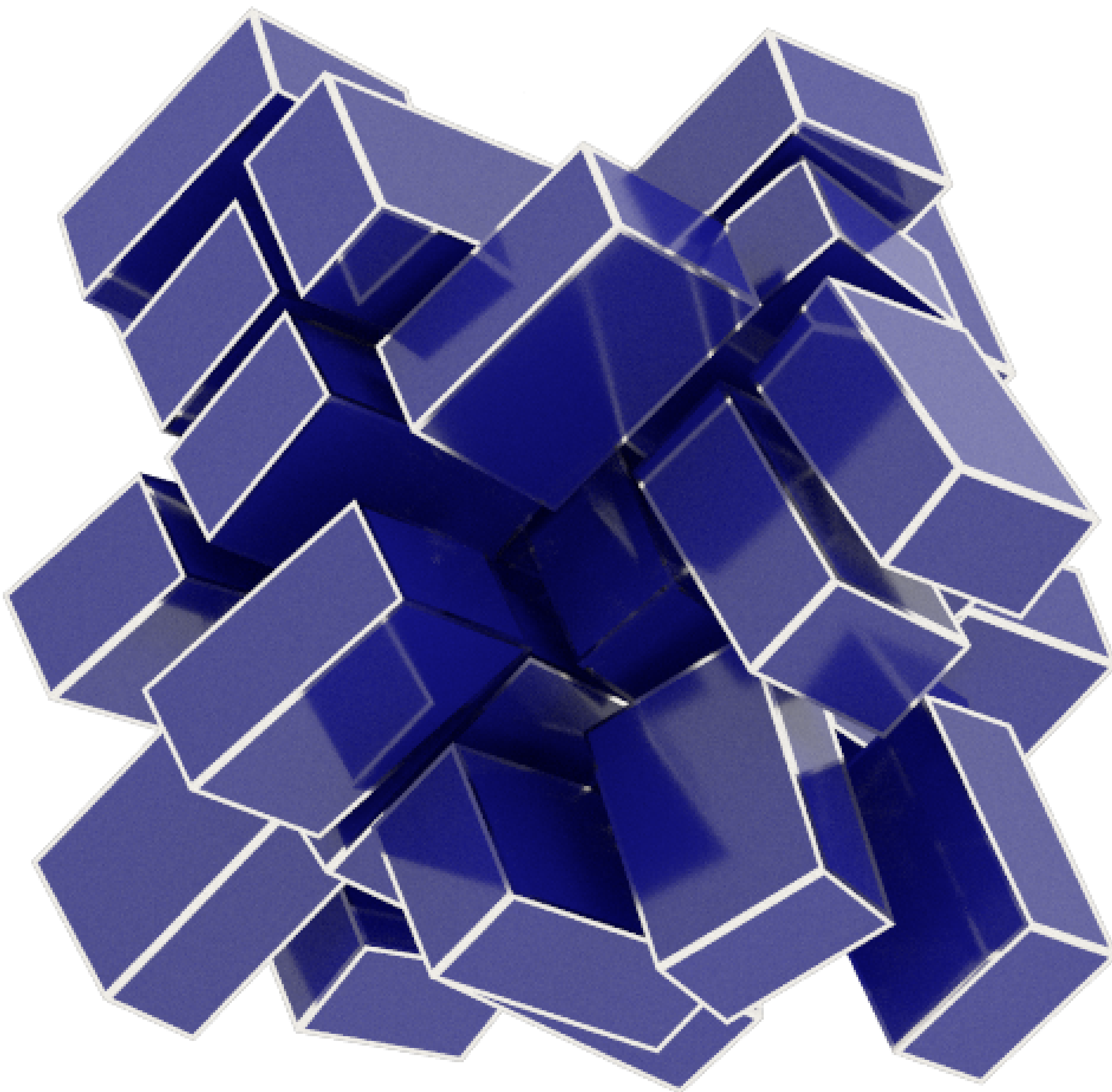


```
import fresnel, numpy, PIL

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(
```

(continues on next page)

(continued from previous page)

```
    color = fresnel.color.linear([0.1, 0.8, 0.1]),
    roughness = 0.8,
    specular = 0.2)

out = fresnel.pathtrace(scene, samples=64, light_samples=32, w=500, h=500)
```



```
import fresnel, numpy, math, PIL

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

scene = fresnel.Scene()
scene.lights = fresnel.light.lightbox()
```

(continues on next page)

(continued from previous page)

```
geometry = fresnel.geometry.ConvexPolyhedron(
    scene,
    origins = [[-data['width'][0],0,0], [data['width'][0],0,0], [0, -data['width'][1],
↪ 0],
               [0, data['width'][1], 0], [0, 0, -data['width'][2]], [0, 0, data['width
↪'][2]]],
    normals = [[-1,0,0], [1,0,0], [0, -1, 0],
               [0, 1, 0],[0, 0, -1], [0, 0, 1]],
    r = math.sqrt(data['width'][0]**2 + data['width'][1]**2 + data['width'][2]**2),
    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.fit(scene, view='front')

out = fresnel.pathtrace(scene, samples=64, light_samples=32, w=500, h=500)
```

1.1 Installation

Fresnel binary images are available on Docker Hub and packages on conda-forge. You can also compile **fresnel** from source.

1.1.1 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, install

```
$ conda install jupyter matplotlib
```

You can update **fresnel** with:

```
$ conda update fresnel
```

1.1.2 Docker images

Pull the docker image glotzerlab/software to get **fresnel** along with many other tools commonly used in simulation and analysis workflows. See full usage information on the glotzerlab/software docker hub page.

Singularity:

```
$ umask 002
$ singularity pull docker://glotzerlab/software
```

Docker:

```
$ docker pull glotzerlab/software
```

1.1.3 Compile from 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-v0.6.0.tar.gz
```

Or, clone using git:

```
$ git clone --recursive  https://bitbucket.org/glotzer/fresnel.git
```

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

Prerequisites

- C++11 capable compiler
- CMake >= 2.8
- Python >= 2.7
- 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 >= 4.0
  - CUDA >= 7.5
- To execute tests:
  - pytest
  - pillow

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

Optional dependencies

- pytest
  - Required to execute unit tests.
- pillow
  - Required to display rendered output in Jupyter notebooks automatically.
- sphinx

  - Required to build the user documentation.

- doxygen
  - Requited to build developer documentation.

Compile

Configure with **cmake** and compile with **make**. Replace `${PREFIX}` your desired installation location.

```
$ mkdir build
$ cd build
$ cmake ../ -DCMAKE_INSTALL_PREFIX=${PREFIX}/lib/python
$ make install -j10
```

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

Add `${PREFIX}/lib/python` to your `PYTHONPATH` to use **fresnel**.

```
$ export PYTHONPATH=$PYTHONPATH:${PREFIX}/lib/python
```

Run tests

Fresnel has extensive unit tests to verify correct execution.

```
$ export PYTHONPATH=/path/to/build
$ cd /path/to/fresnel
$ cd test
$ pytest
```

Build user documentation

Build the user documentation with **sphinx**:

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

Specify search paths

OptiX, **TBB**, **Embree**, and **Python** may be installed in a variety of locations. To specify locations for libraries, use these methods the *first* time you invoke `cmake` in a clean build directory.

Library	Default search path	CMake Custom search path
OptiX	`/opt/optix`	`-DOptiX_INSTALL_DIR=/path/to/optix`
TBB	*system*	`TBB_LINK=/path/to/tbb/lib` (env var)
Embree	*system*	`-Dembree_DIR=/path/to/embree-3.x.y` (the directory containing embree-config.cmake)
Python	$PATH	`-DPYTHON_EXECUTABLE=/path/to/bin/python`

Build C++ Documentation

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

1.2 Change log

fresnel releases follow semantic versioning.

1.2.1 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

1.2.2 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.

1.2.3 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

1.2.4 v0.3.0 (2017-03-09)

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

1.2.5 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

1.2.6 v0.1.0 (2017-02-02)

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

1.3 User community

1.3.1 fresnel-users mailing list

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

1.3.2 Issue tracker

File bug reports on frenel's issue tracker.

1.3.3 Contribute

fresnel is an open source project. Contributions are accepted via pull request to fresnel's bitbucket 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.

1.4 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.

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

1.4.1 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.

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

1.4.2 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.

```
In [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:

```
In [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]]
```

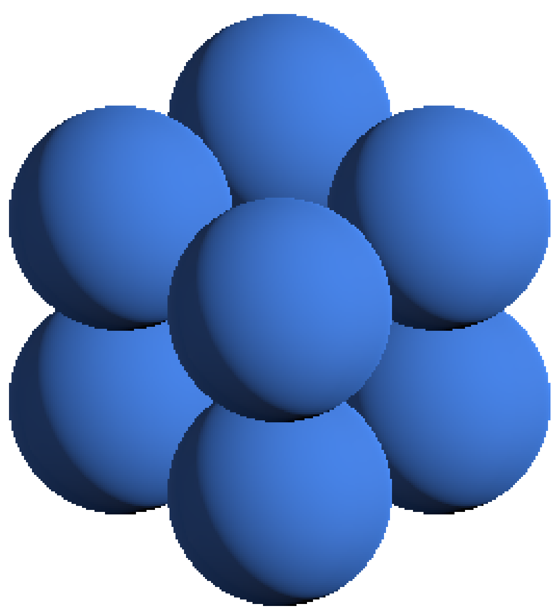
Set the **material** of the geometry object to a rough blue surface:

```
In [5]: geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.9]),
                                                      roughness=0.8)
```

1.4.3 Render the scene

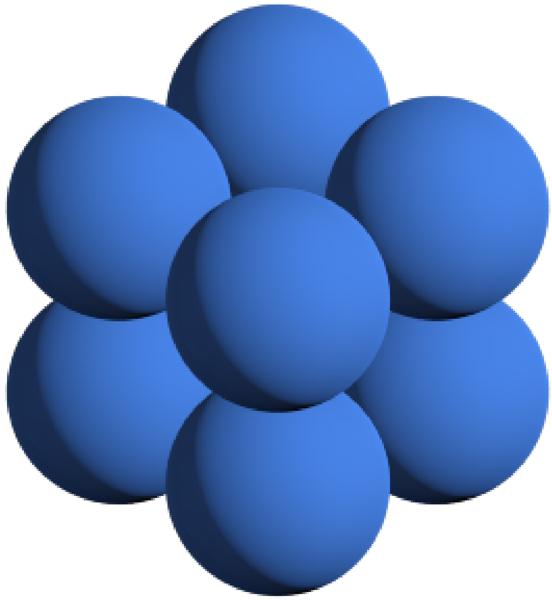
preview quickly renders the scene from the view point of the camera.

```
In [6]: fresnel.preview(scene)
```



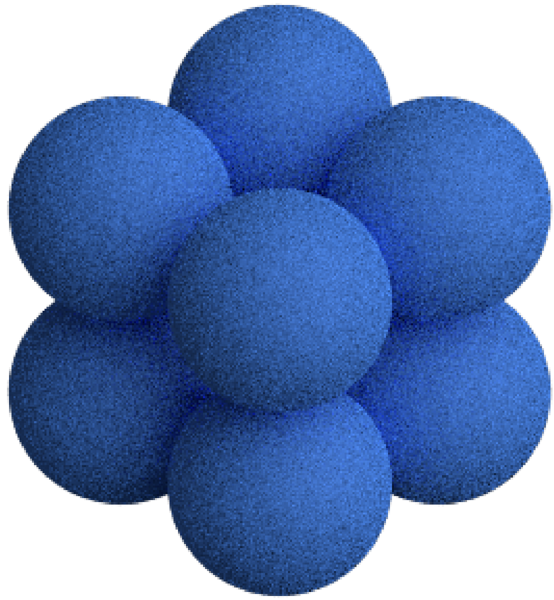
Enable *anti-aliasing* if you plan to use the output of **preview** for production use. The scene will take longer to render, but will have crisp edges.

```
In [7]: fresnel.preview(scene, aa_level=3)
```



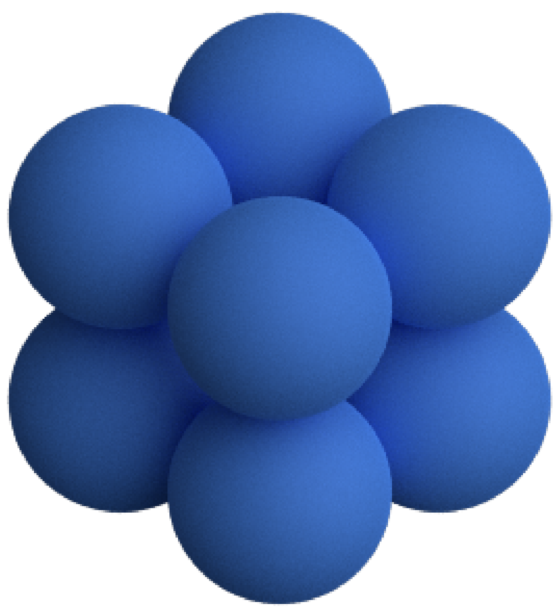
preview only applies direct lighting. Use **pathtrace** to account for indirect lighting. (anti-aliasing is always enabled when path tracing).

```
In [8]: fresnel.pathtrace(scene)
```



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

```
In [9]: fresnel.pathtrace(scene, light_samples=40)
```



1.4.4 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).

```
In [10]: out = fresnel.preview(scene, aa_level=3)
         print(out[:].shape)
         print(out[:].dtype)
```

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

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

```
In [11]: import PIL
```

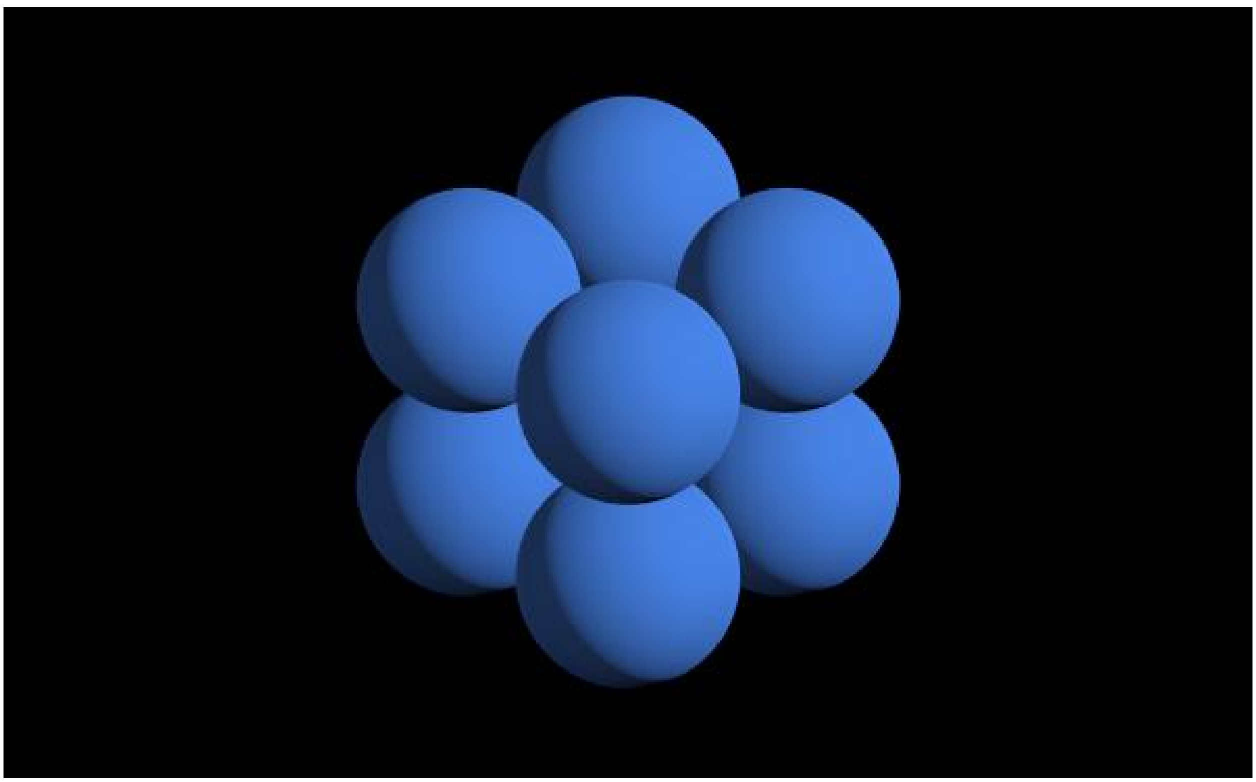
```
In [12]: image = PIL.Image.fromarray(out[:], mode='RGBA')
         image.save('output.png')
```

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

```
In [13]: 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):

```
In [16]: import IPython.display
         IPython.display.Image('output.jpeg')
```



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

1.5 Primitive properties

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

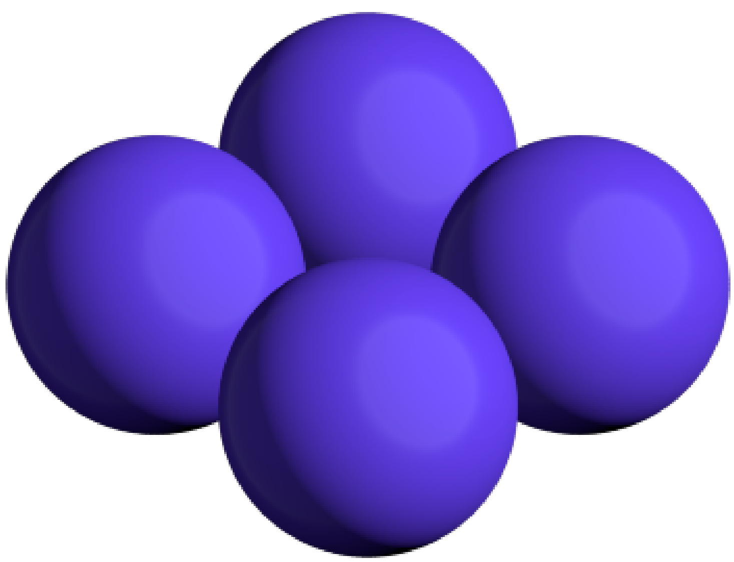
```
In [1]: import fresnel
        scene = fresnel.Scene()
```

1.5.1 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.

```
In [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.l
                                           # per-primitive color left default
                                           )
```

```
In [3]: fresnel.preview(scene, aa_level=3)
```

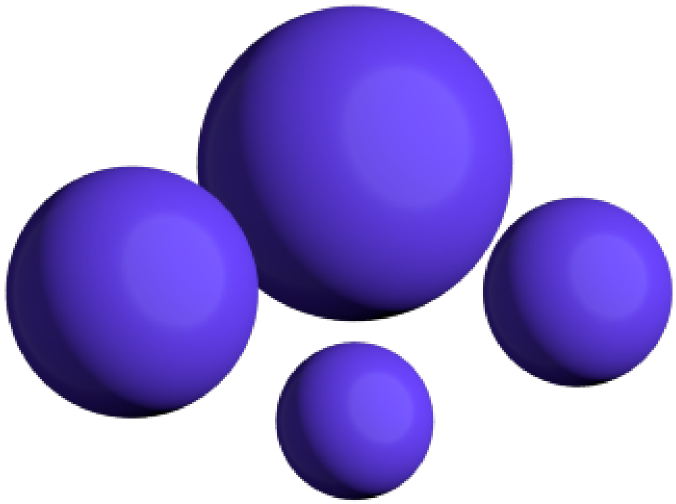


1.5.2 Changing properties after creation

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

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

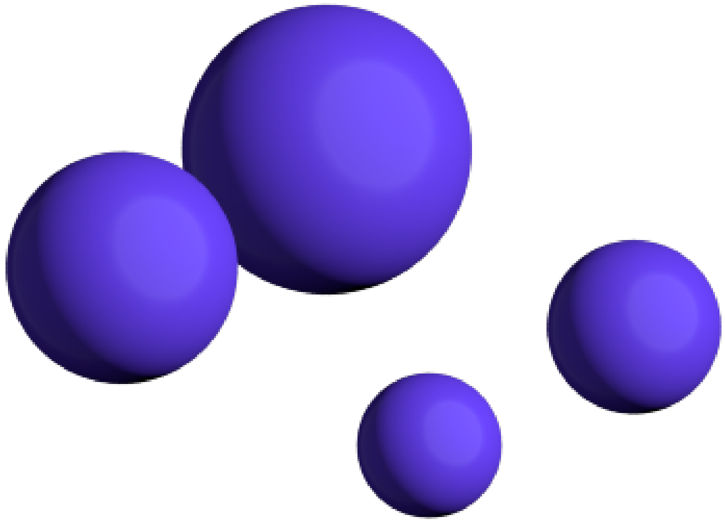
```
In [5]: fresnel.preview(scene, aa_level=3)
```



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

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

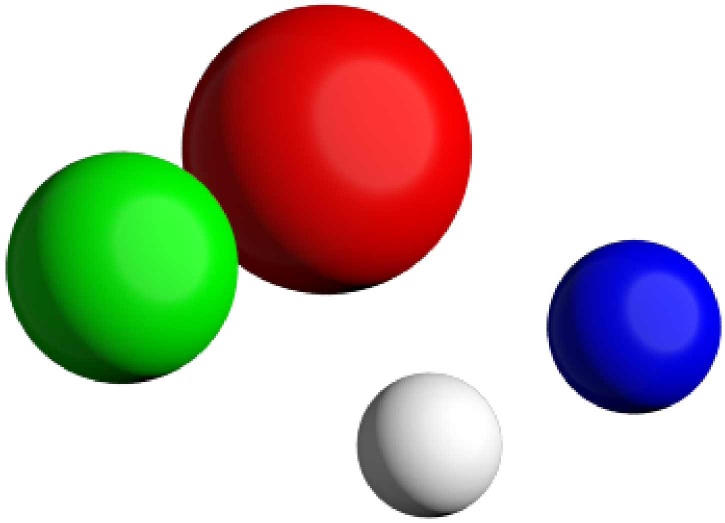
```
In [7]: fresnel.preview(scene, aa_level=3)
```



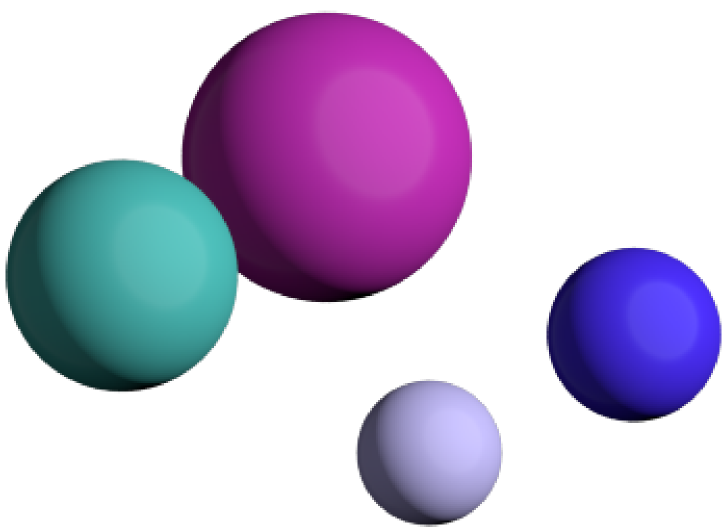
The *color* property sets a per primitive color. The geometry **material color** and the **primitive color** are mixed with fraction **primitive_color_mix**. A value of 1.0 selects the primitive color, 0.0 selects the *material* color and values in between mix the colors.

```
In [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]])
```

```
In [9]: fresnel.preview(scene, aa_level=3)
```



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

In [11]: fresnel.preview(scene, aa_level=3)
```



1.5.3 Reading primitive properties

Primitive properties may be read as well as written.

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

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

In [13]: geometry.position[:]

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

In [14]: geometry.color[:]

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

1.5.4 Common errors

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

```
In [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.

1.6 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.

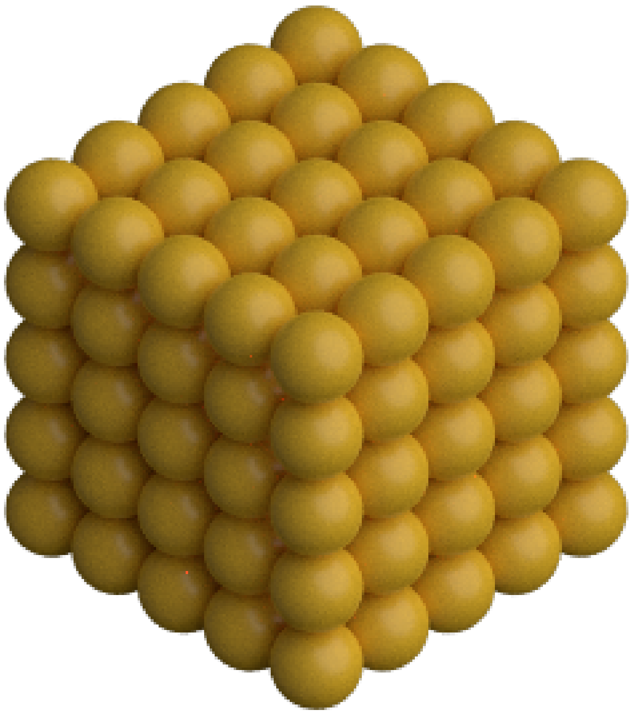
```
In [1]: import fresnel
        import math
        device = fresnel.Device()
        scene = fresnel.Scene(device)
        position = []
        for k in range(5):
            for i in range(5):
                for j in range(5):
                    position.append([2*i, 2*j, 2*k])
        geometry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

1.6.1 Material color

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

```
In [2]: geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.9,0.714,0.169]))
```

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

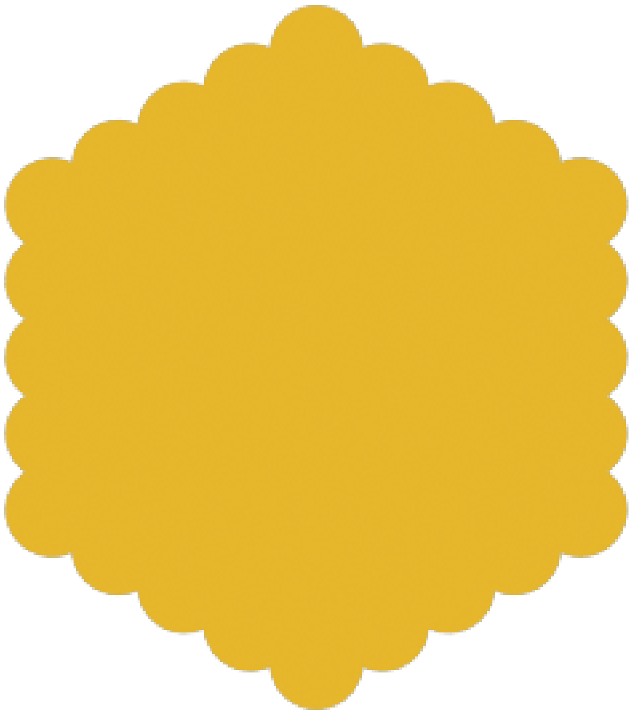


1.6.2 Solid color materials

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**.

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

```
In [5]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```

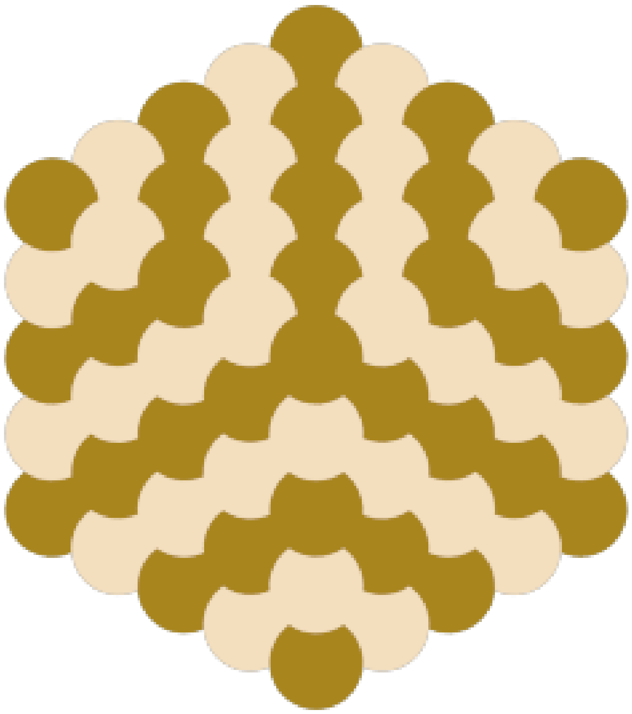


1.6.3 Geometry / primitive color mixing

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.

```
In [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])
```

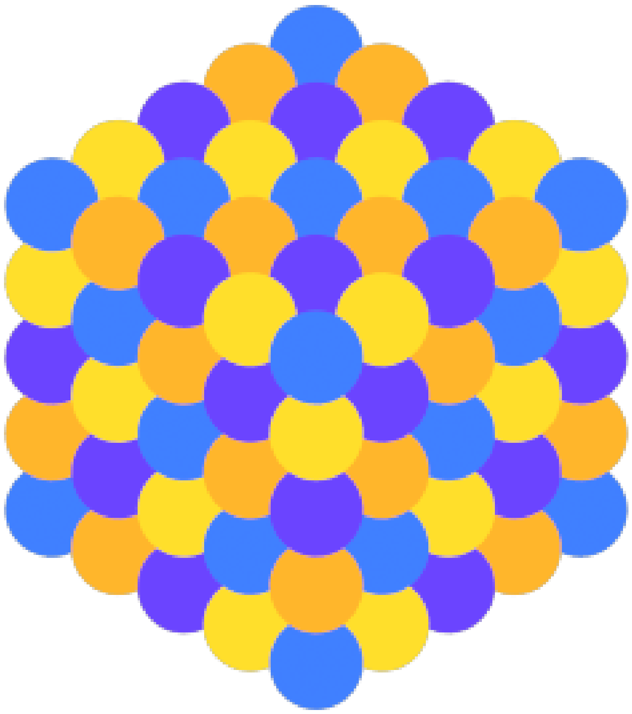
```
In [7]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```



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.)

```
In [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])
```

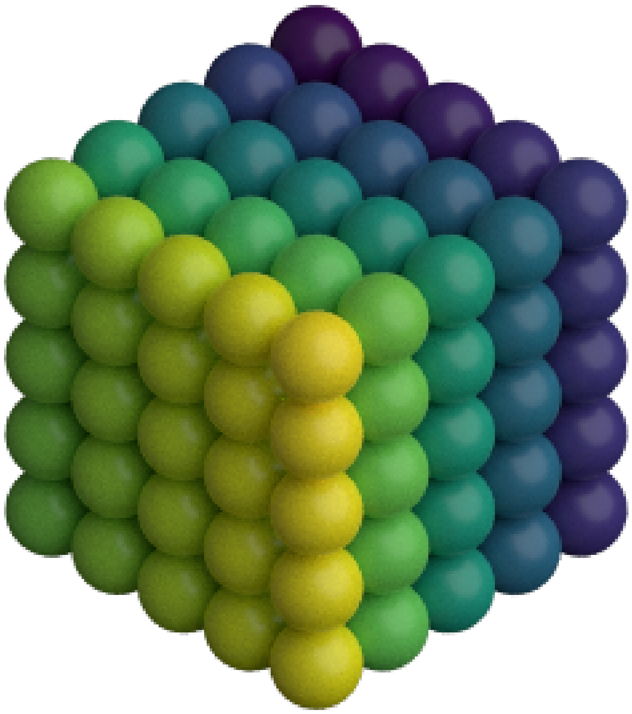
```
In [9]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```



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

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

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

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



1.6.4 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.

```
In [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))]
```

Examples

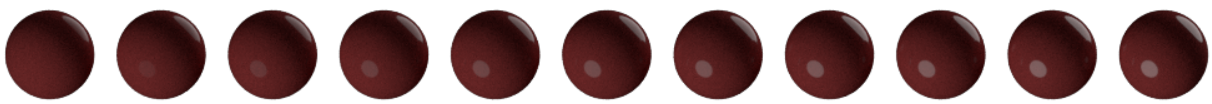
These examples are front lit from the lower left and back lit from the upper right.

Vary roughness in a specular material from 0.1 to 1.1

```
In [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)
```

Vary specular from 0 to 1 with constant roughness.

```
In [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)
```



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)

```
In [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)
```

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

```
In [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)
```

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

```
In [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,
                     roughness=ipywidgets.FloatSlider(value=0.3, min=0.1, max=1.0, step=0.1,
                     specular=ipywidgets.FloatSlider(value=0.5, min=0.0, max=1.0, step=0.1,
                     spec_trans=ipywidgets.FloatSlider(value=0.0, min=0.0, max=1.0, step=0.
                     metal=ipywidgets.FloatSlider(value=0, min=0.0, max=1.0, step=1.0, cont
                     light_theta=ipywidgets.FloatSlider(value=5.5, min=0.0, max=2*math.pi, s
                     light_phi=ipywidgets.FloatSlider(value=0.8, min=0.0, max=math.pi, step=
def test(color, primitive_color_mix, roughness, specular, spec_trans, metal, light_theta, l
    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))

    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.

1.7 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**.

In [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=fresn
fresnel.light.cloudy();
```

Out[1]:
```
[<fresnel.light.Light at 0x7f6a380e19e8>,
 <fresnel.light.Light at 0x7f6a380a9358>]
```

1.7.1 Enabling outlines

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

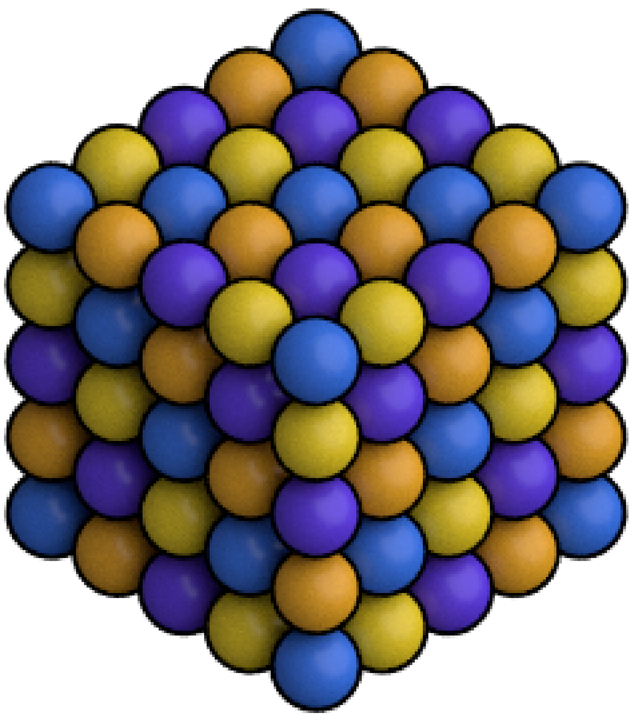
```
In [2]: geometry.outline_width
```

```
Out[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.

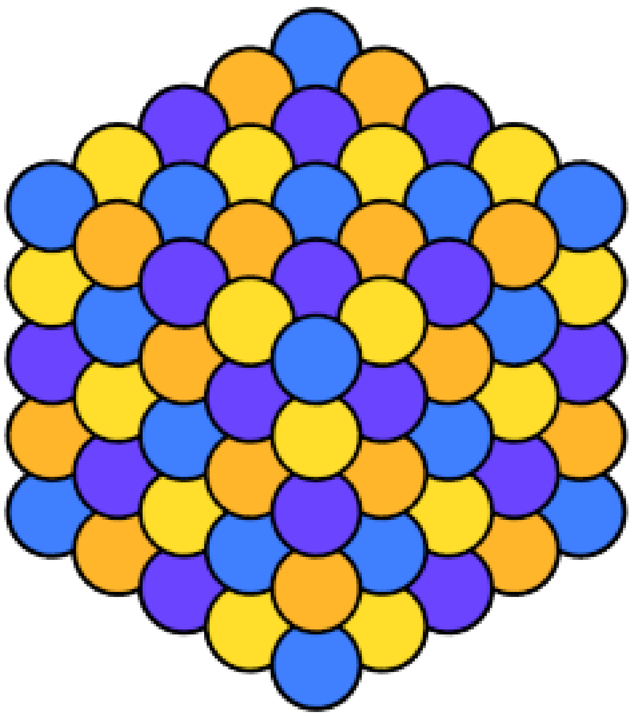
```
In [3]: geometry.outline_width = 0.12
```

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



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

```
In [6]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```



1.7.2 Outline material properties

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

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

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

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

```
Out[8]: 1.0
```

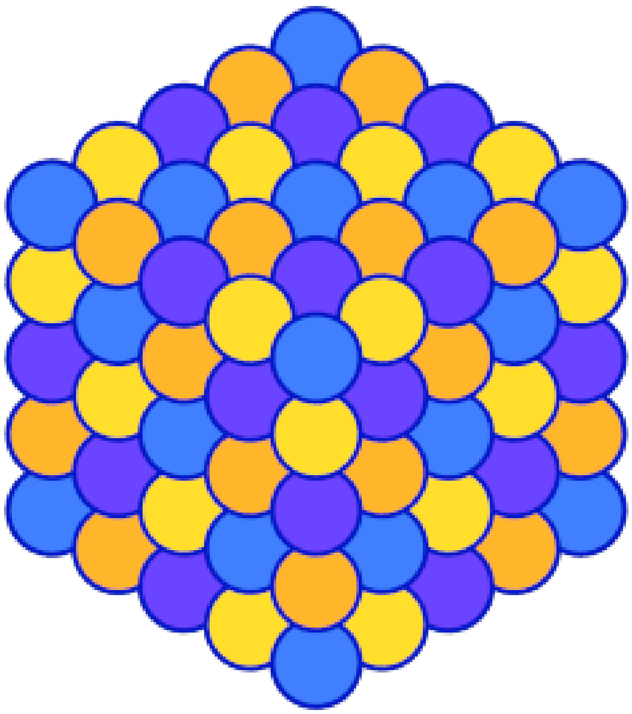
```
In [9]: geometry.outline_material.primitive_color_mix
```

```
Out[9]: 0.0
```

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

```
In [10]: geometry.outline_material.color = fresnel.color.linear(fresnel.color.linear([0.08,0.341,0.9
```

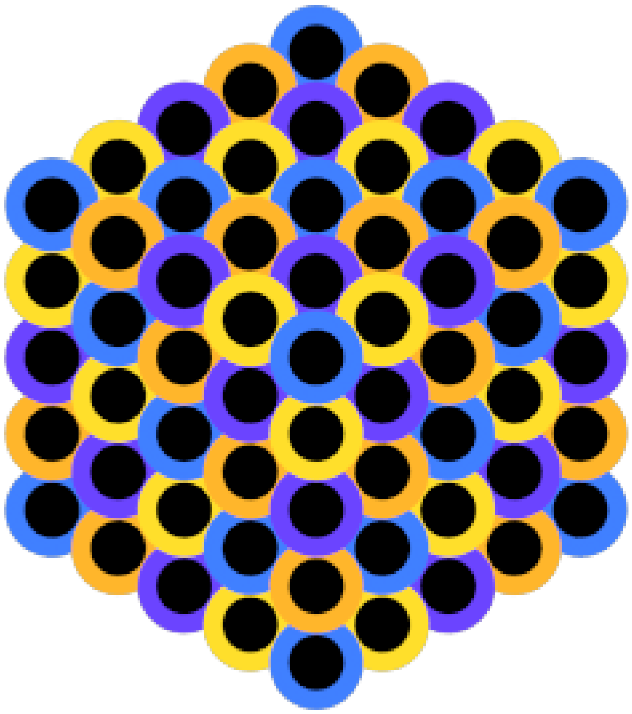
```
In [11]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```



Outlines may be colored by the primitives:

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

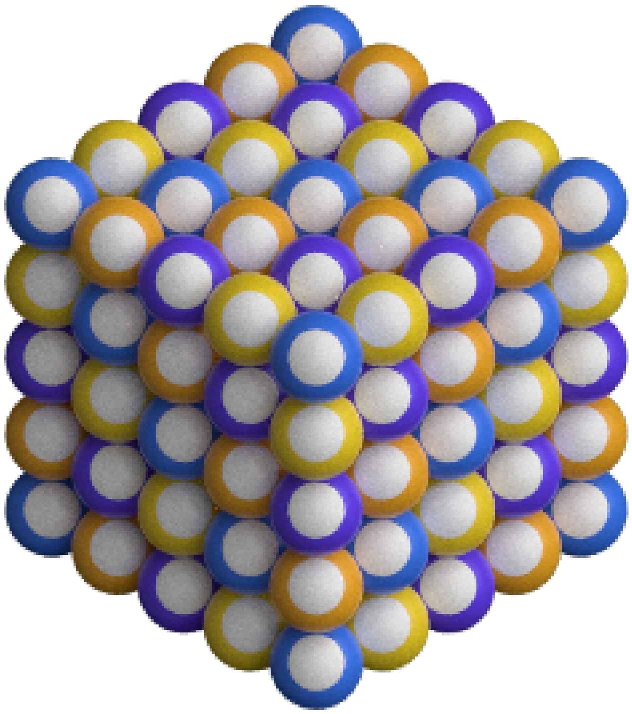
```
In [13]: fresnel.preview(scene, w=300, h=300, aa_level=3)
```



Outlines may have diffuse shading:

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

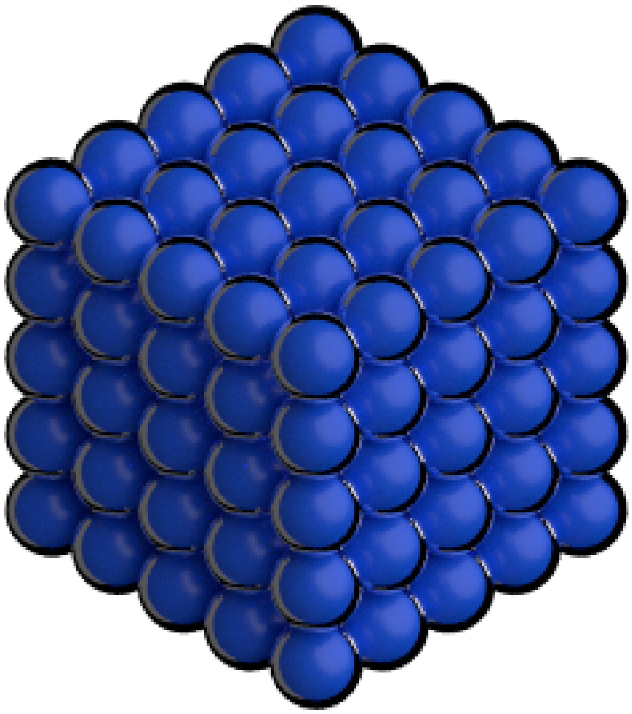
```
In [15]: fresnel.pathtrace(scene, w=300, h=300,light_samples=40)
```



Or be metallic:

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

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



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

1.8 Scene properties

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

```
In [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,
        geometry.outline_width = 0.12
```

1.8.1 Background color and alpha

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

```
In [4]: scene.background_color
```

```
Out[4]: (0.0, 0.0, 0.0)
```

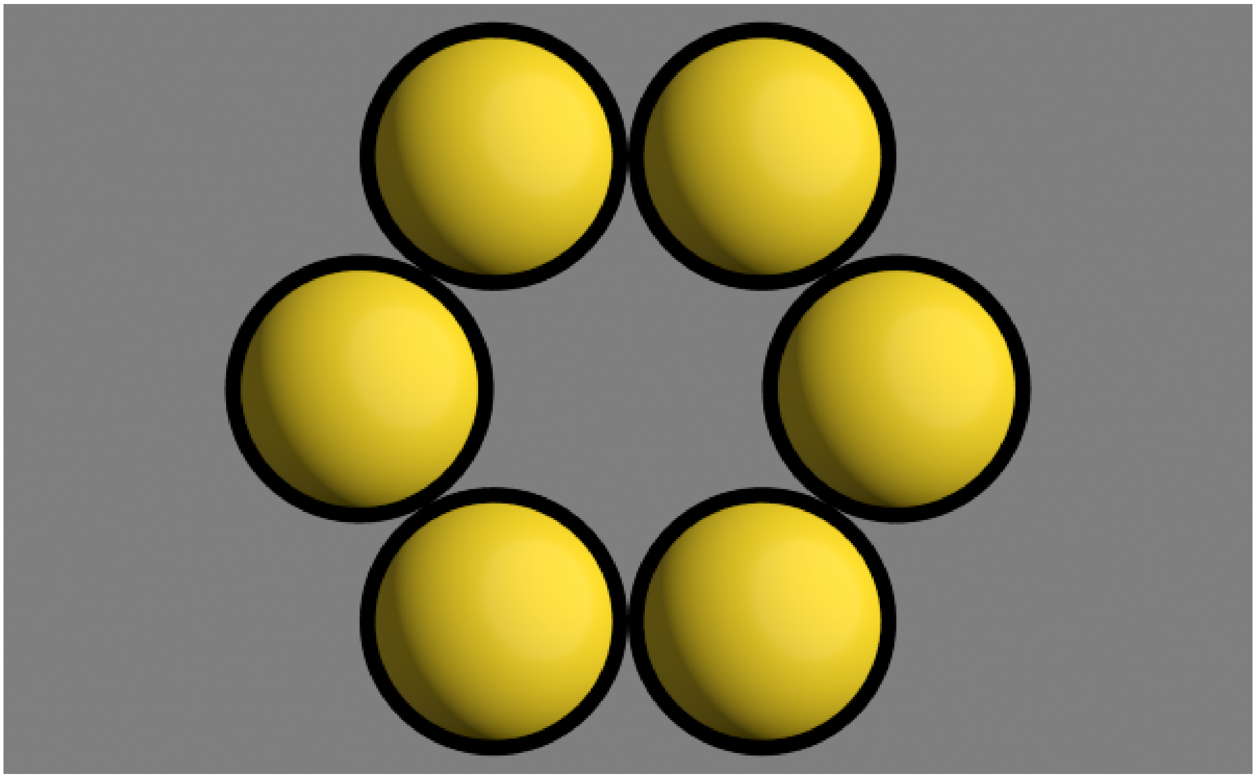
```
In [5]: scene.background_alpha
```

```
Out[5]: 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:

```
In [6]: scene.background_alpha = 0.5
```

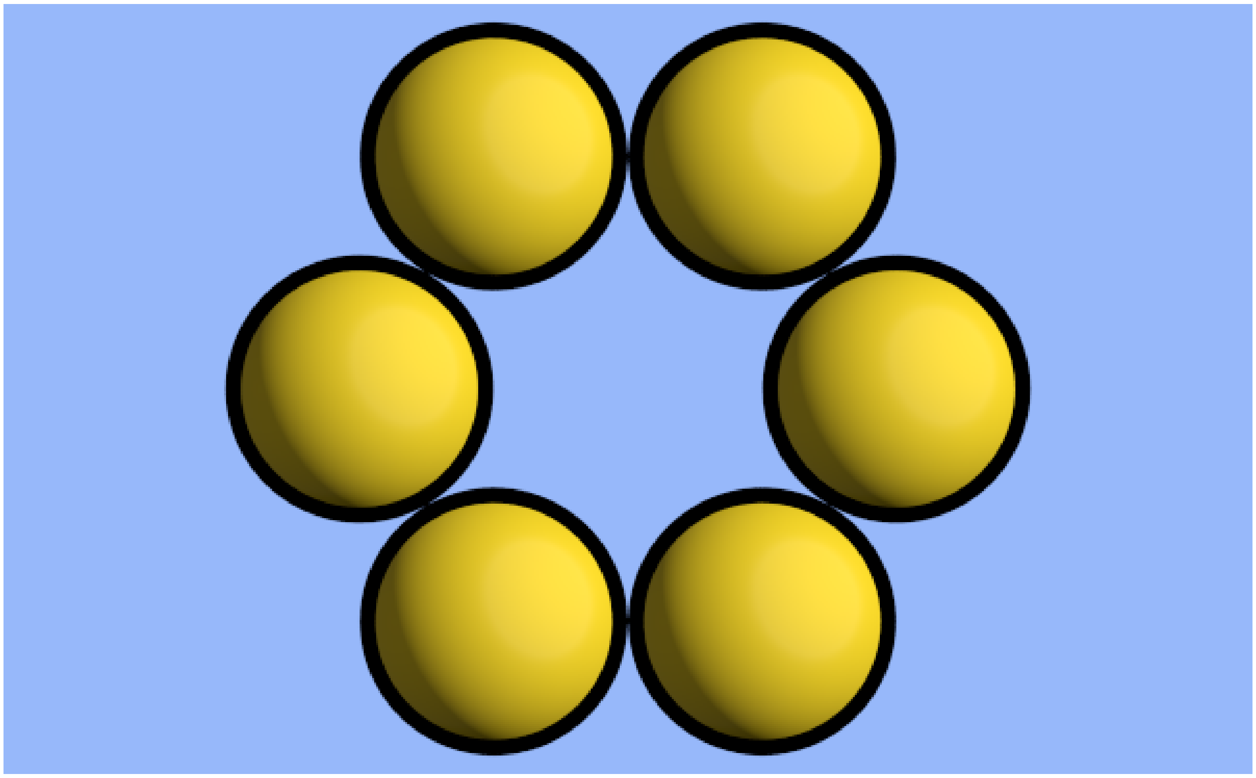
```
In [7]: fresnel.preview(scene, aa_level=3)
```



Set a solid background color:

```
In [8]: scene.background_alpha = 1.0
        scene.background_color = fresnel.color.linear([0.592, 0.722, 0.98])
```

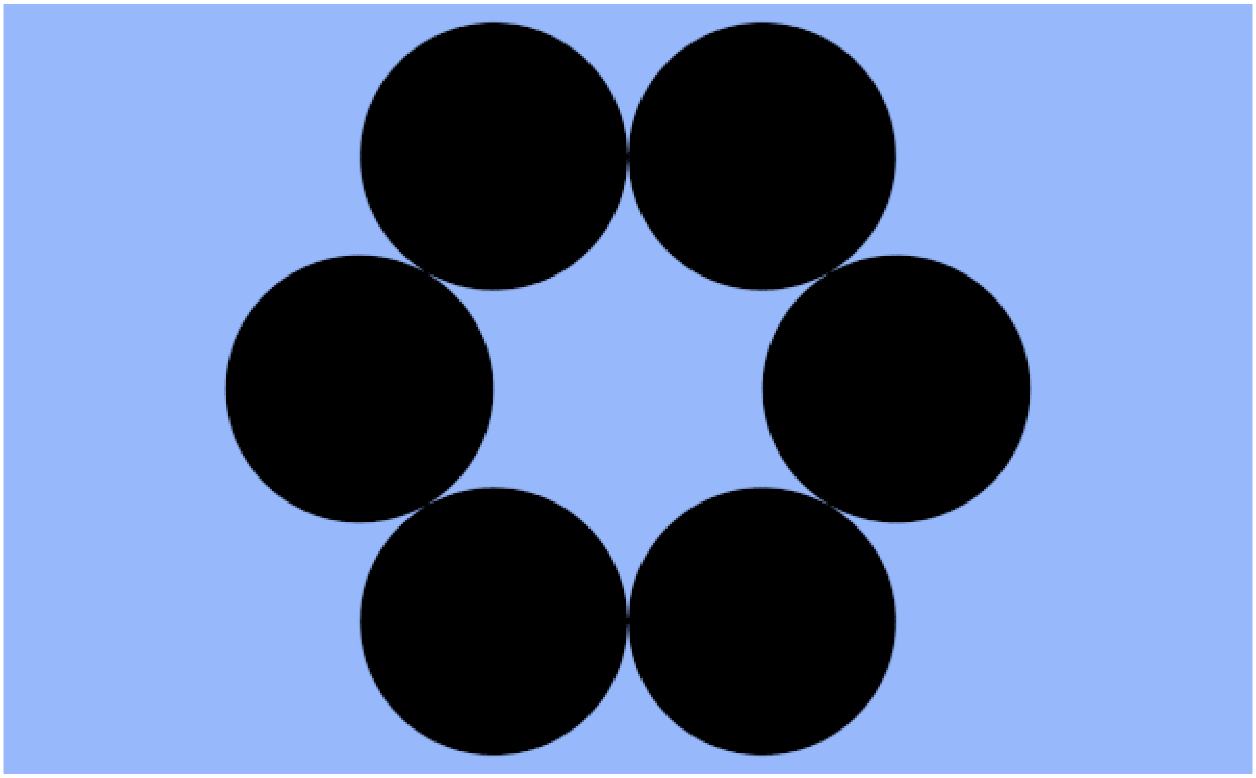
```
In [9]: fresnel.preview(scene, aa_level=3)
```



1.8.2 Light sources

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

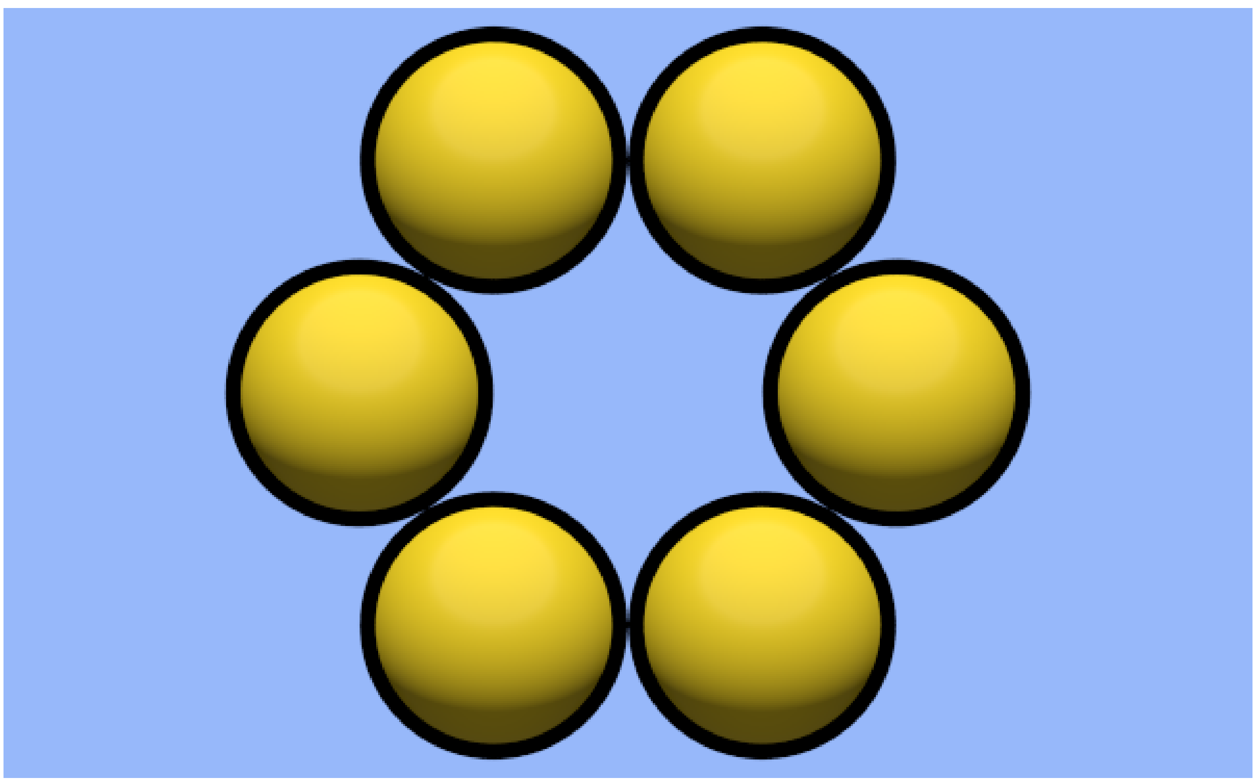
```
In [10]: scene.lights.clear()

In [11]: fresnel.preview(scene, aa_level=3)
```



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

```
In [12]: scene.lights = fresnel.light.butterfly()
```

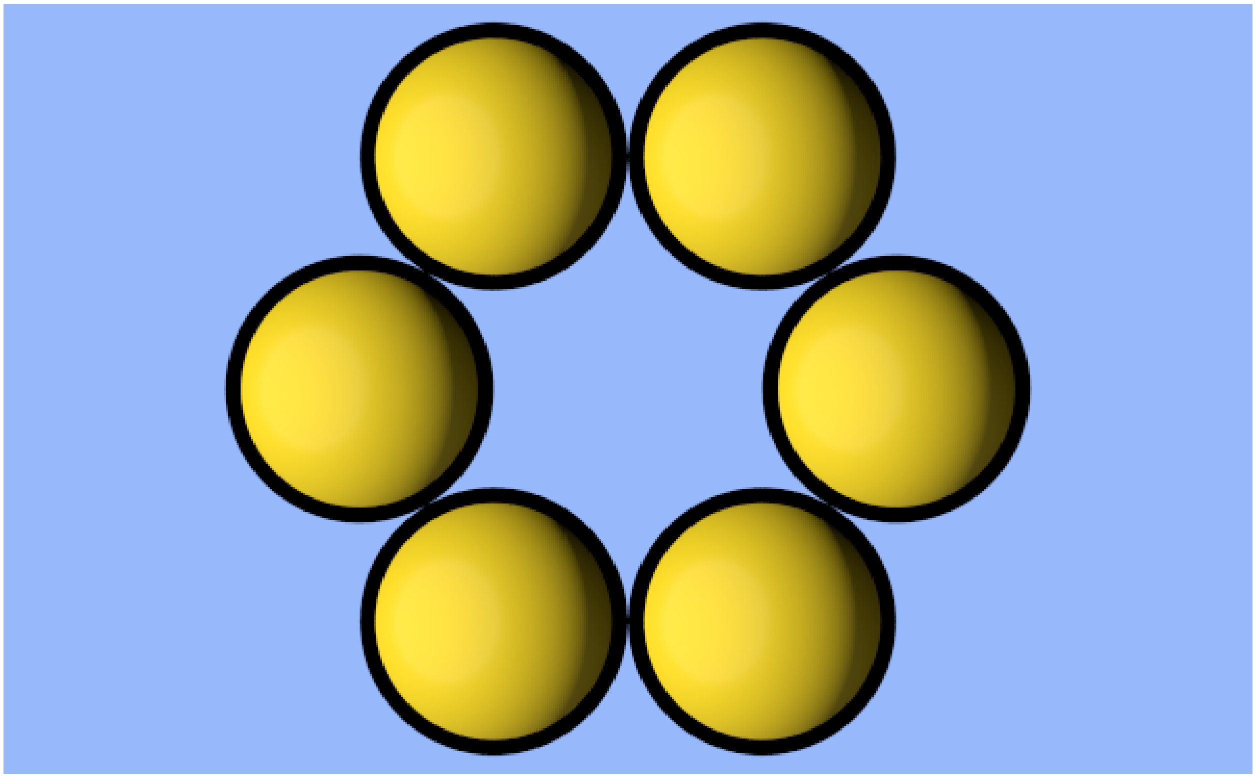
```
In [13]: fresnel.preview(scene, aa_level=3)
```



You can modify individual lights.

```
In [15]: scene.lights[0].direction = (-1, 0, 1)
```

```
In [16]: fresnel.preview(scene, aa_level=3)
```



1.8.3 Camera

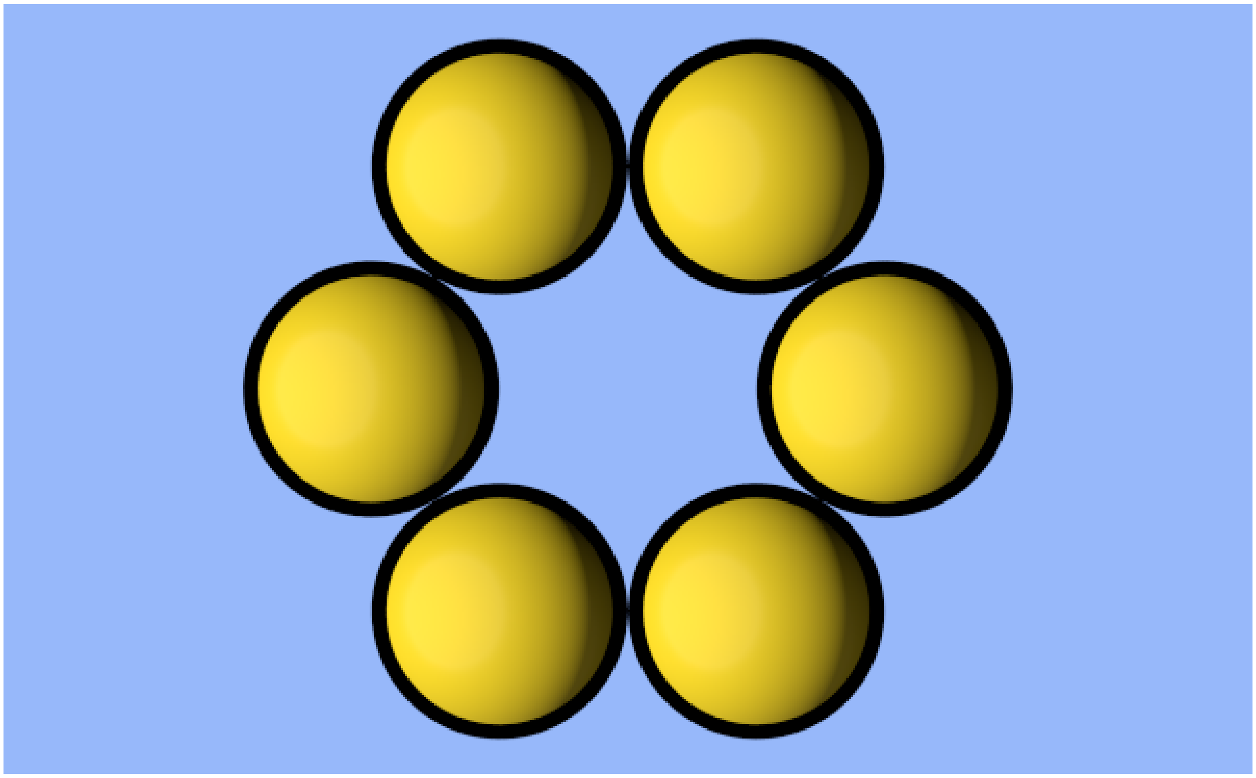
The camera defines the view to render into the scene. By default, the camera is `auto` and the camera is automatically selected to fit the scene every time it is rendered.

```
In [17]: print(scene.camera)
```

```
auto
```

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.

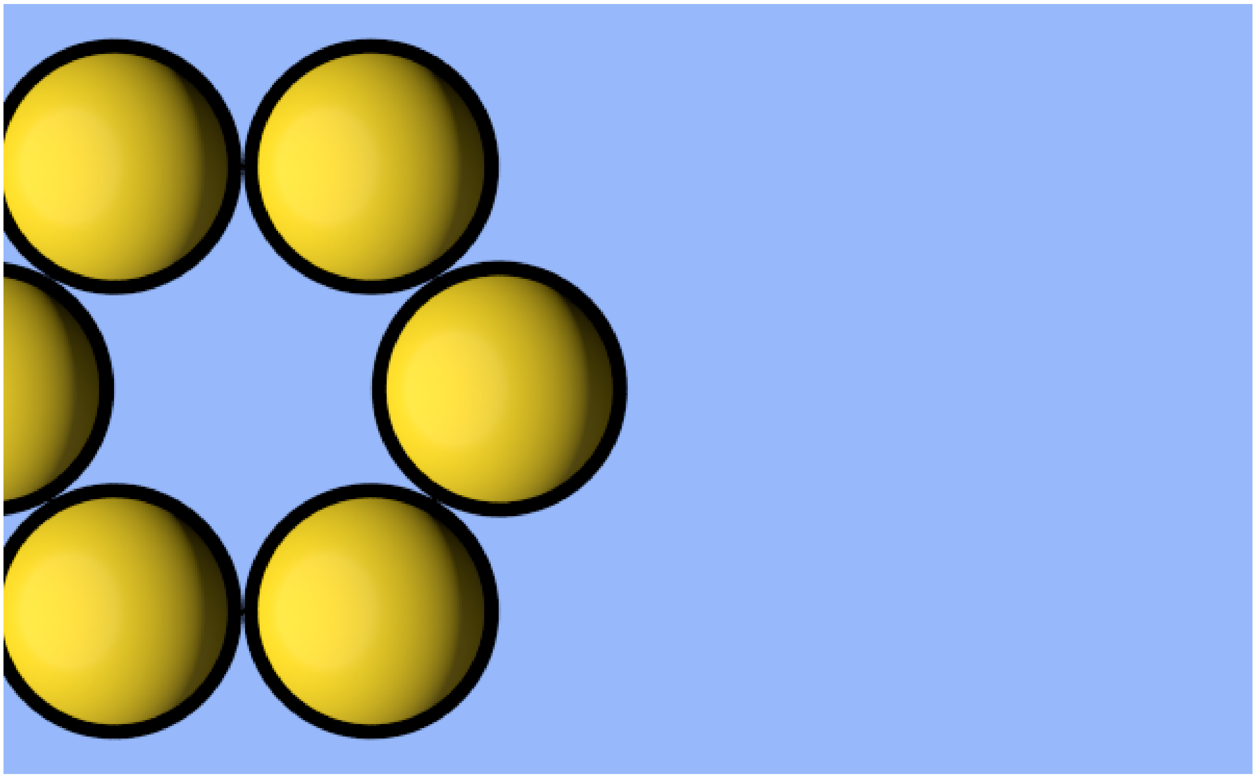
```
In [18]: scene.camera = fresnel.camera.orthographic(position=(0,0,2), look_at=(0,0,0), up=(0,1,0), he
         fresnel.preview(scene, aa_level=3)
```



You can modify these parameters individually.

```
In [19]: scene.camera.position = (3, 0, 10)
         scene.camera.look_at=(3,0,0)
```

```
In [20]: fresnel.preview(scene, aa_level=3)
```



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

1.9 Lighting setups

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

```
In [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
                                              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.fit(scene, view='isometric')
        tracer = fresnel.tracer.Path(device, w=450, h=450)
```

1.9.1 Lighting presets

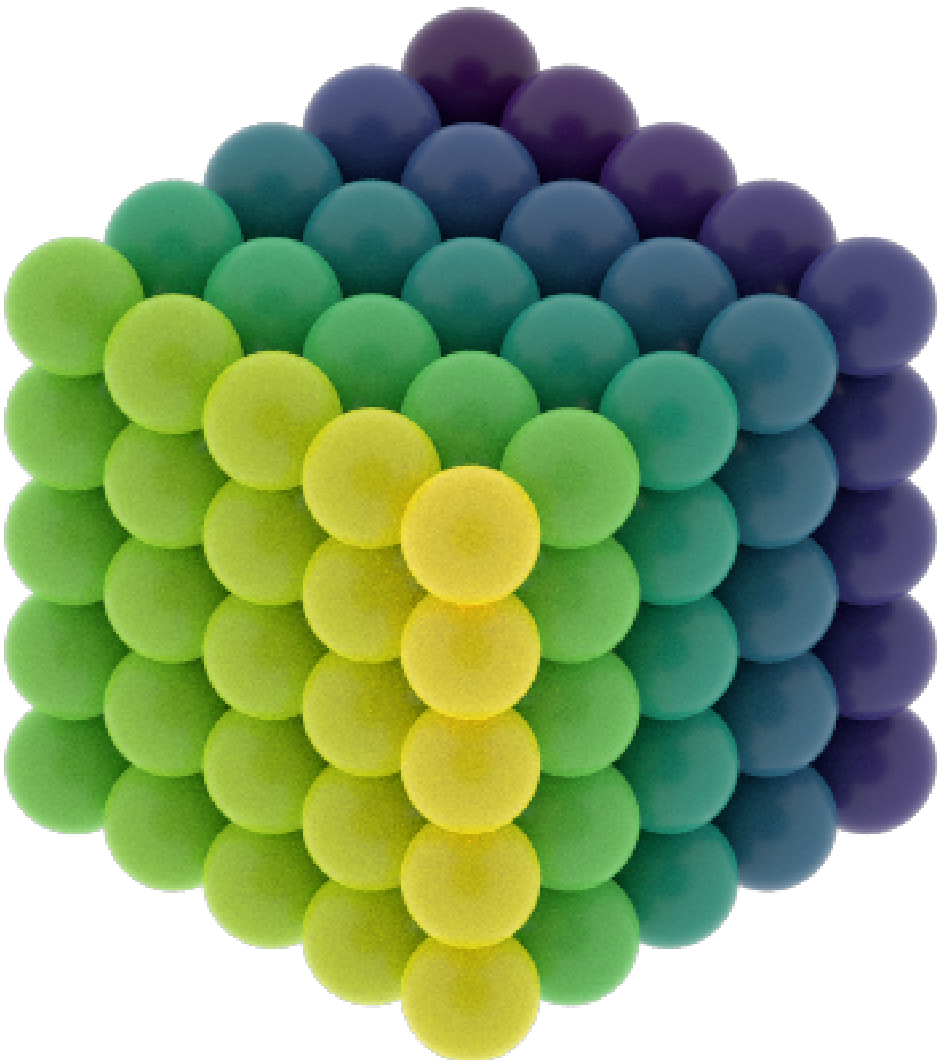
Fresnel defines many lighting presets that use classic photography techniques to light the scene. Create a setup and assign it to the Scene's lights.

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

Light box

Light box lighting lights from the top, bottom, left, and right. This type of lighting is commonly used product photography.

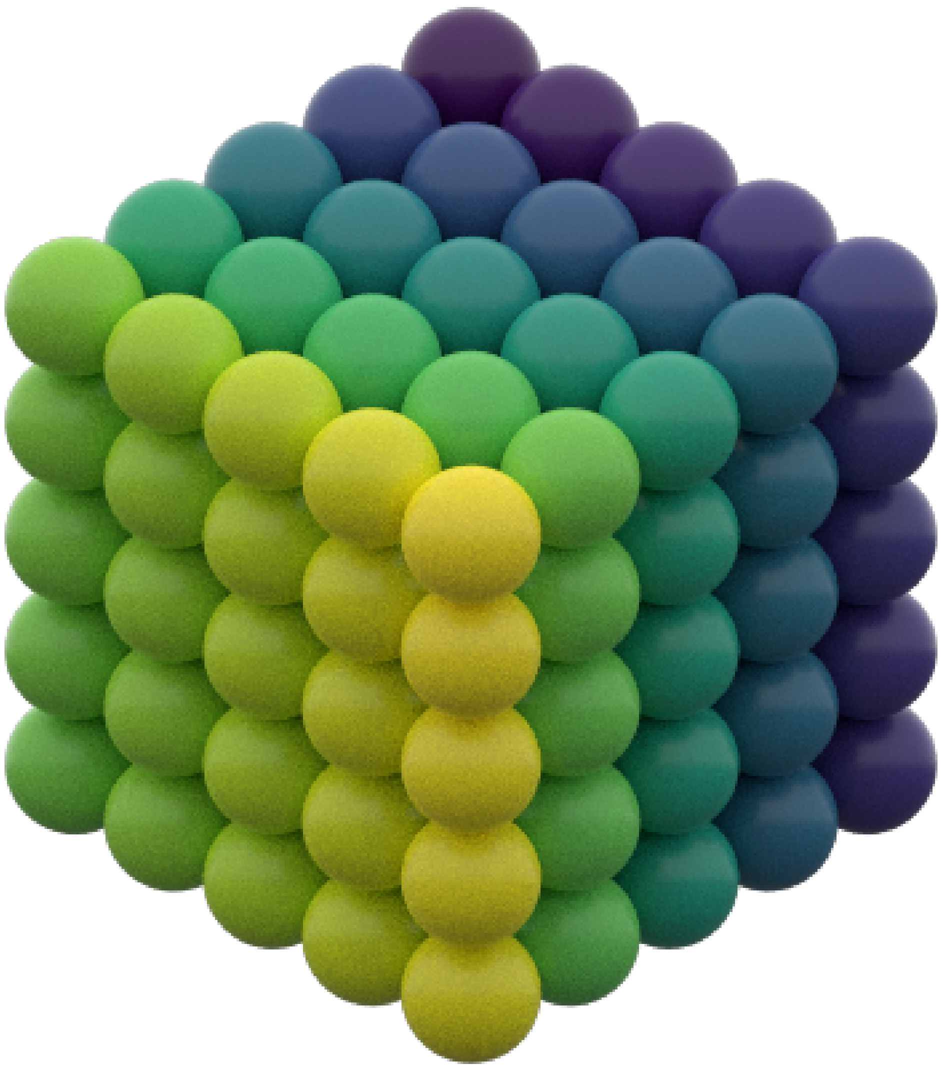
```
In [2]: scene.lights = fresnel.light.lightbox()
        tracer.sample(scene, samples=64, light_samples=10)
```



Cloudy

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

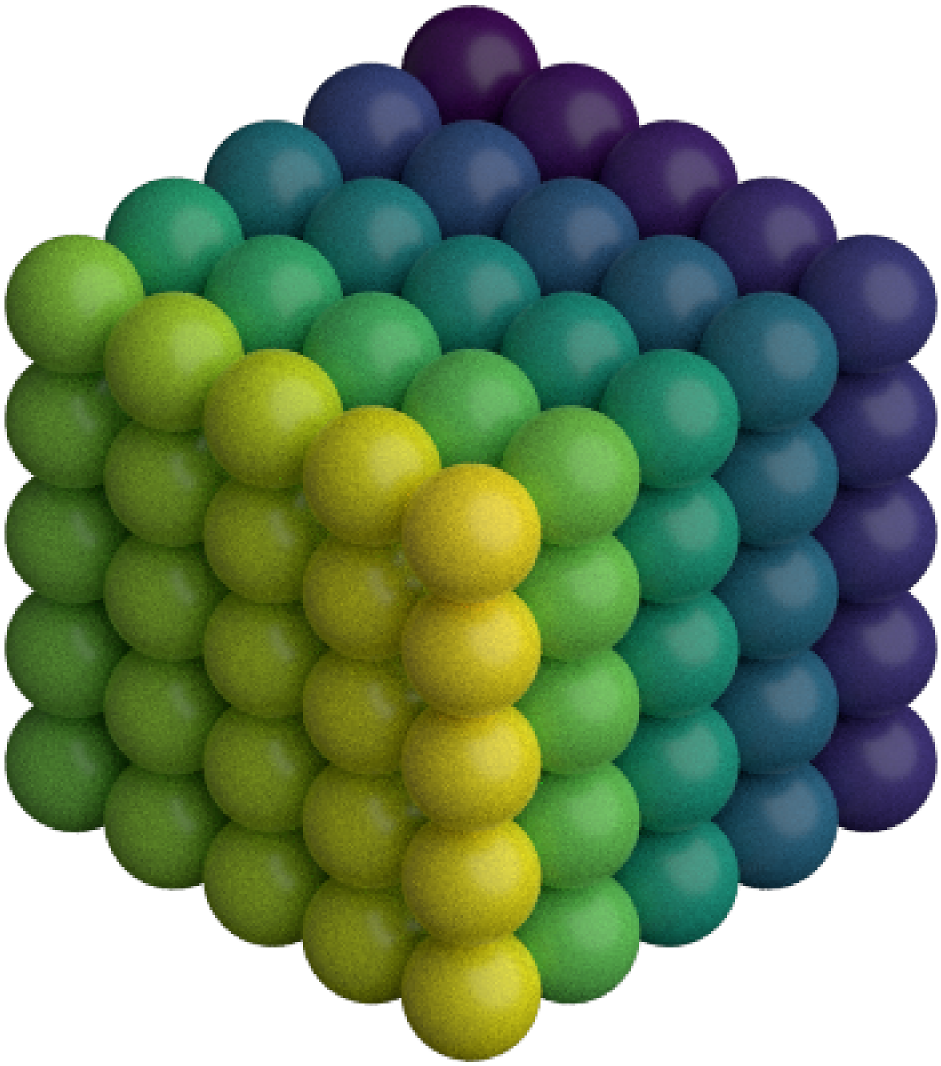
```
In [3]: scene.lights = fresnel.light.cloudy()
        tracer.sample(scene, samples=64, light_samples=10)
```



Rembrandt

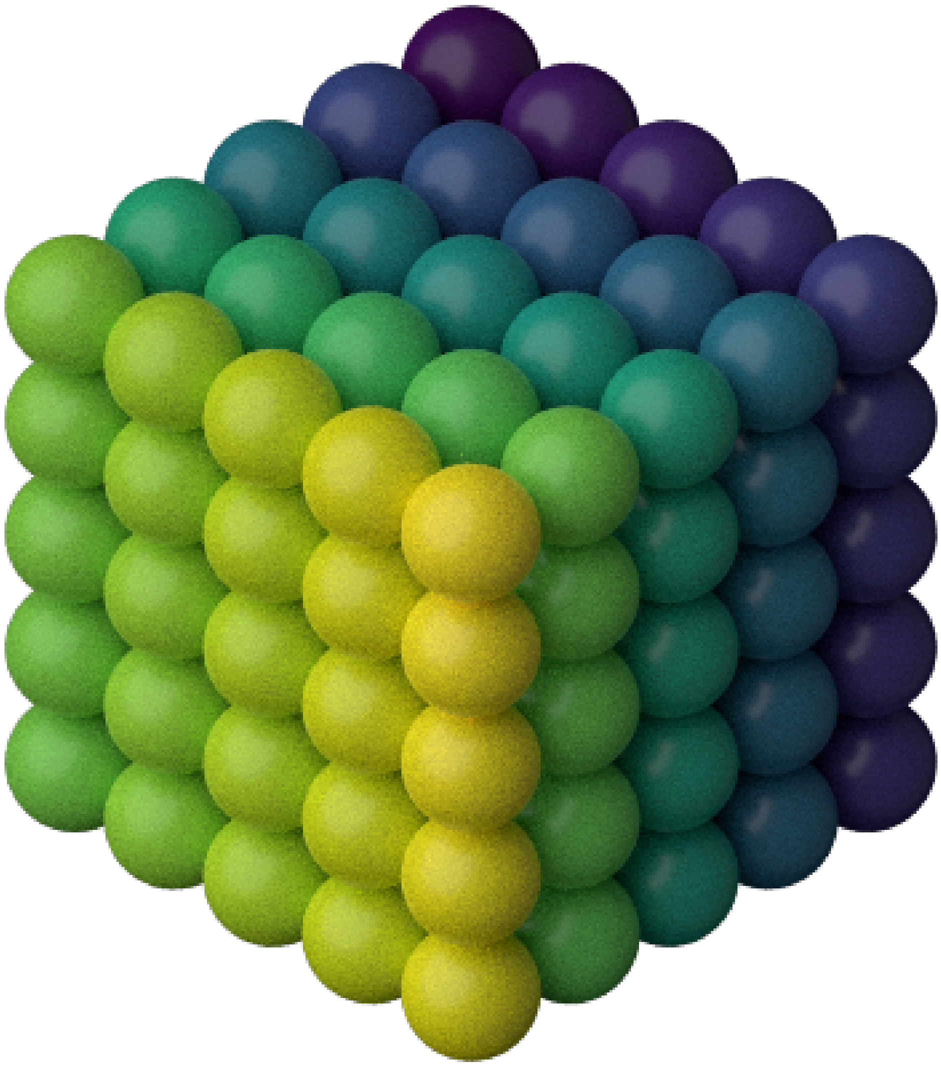
Rembrandt lighting places the key light 45 degrees to one side and slightly up.

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



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

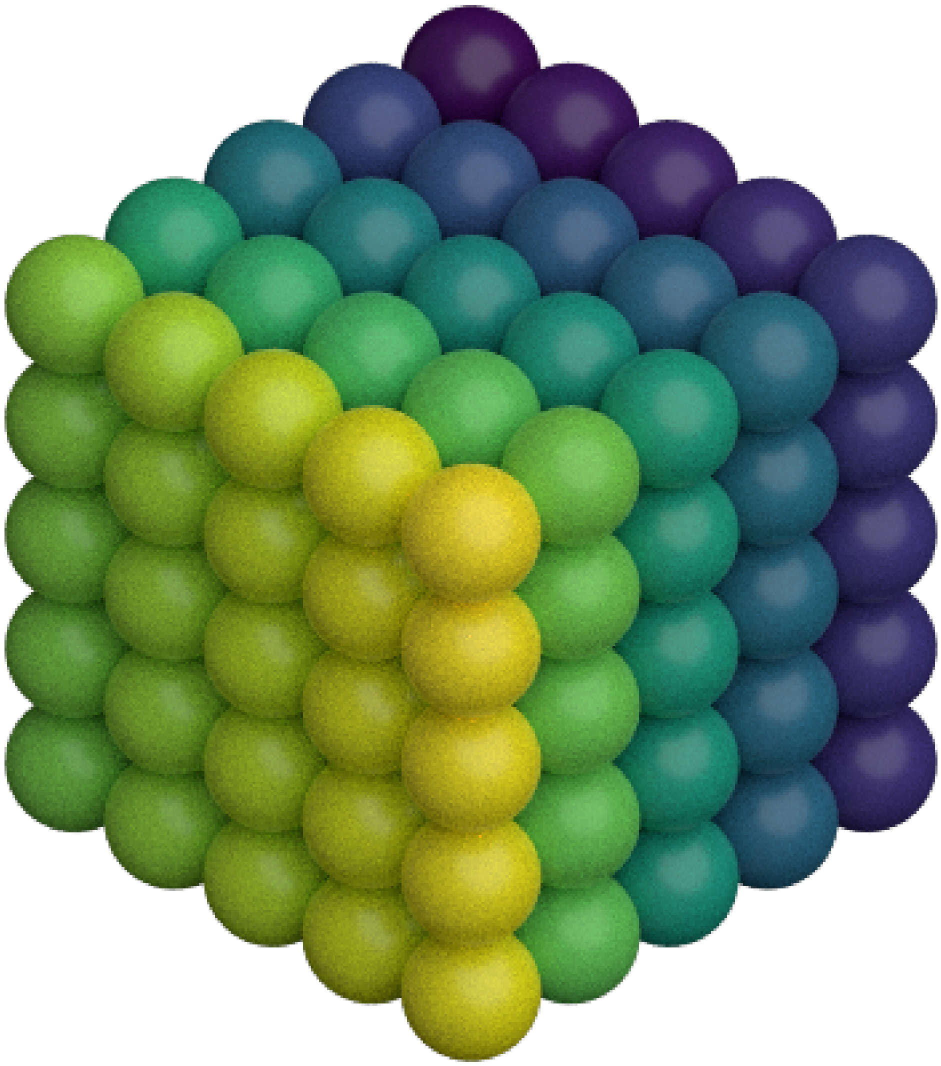
```
In [5]: scene.lights = fresnel.light.rembrandt(side='left')
        tracer.sample(scene, samples=64, light_samples=10)
```



Loop lighting

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

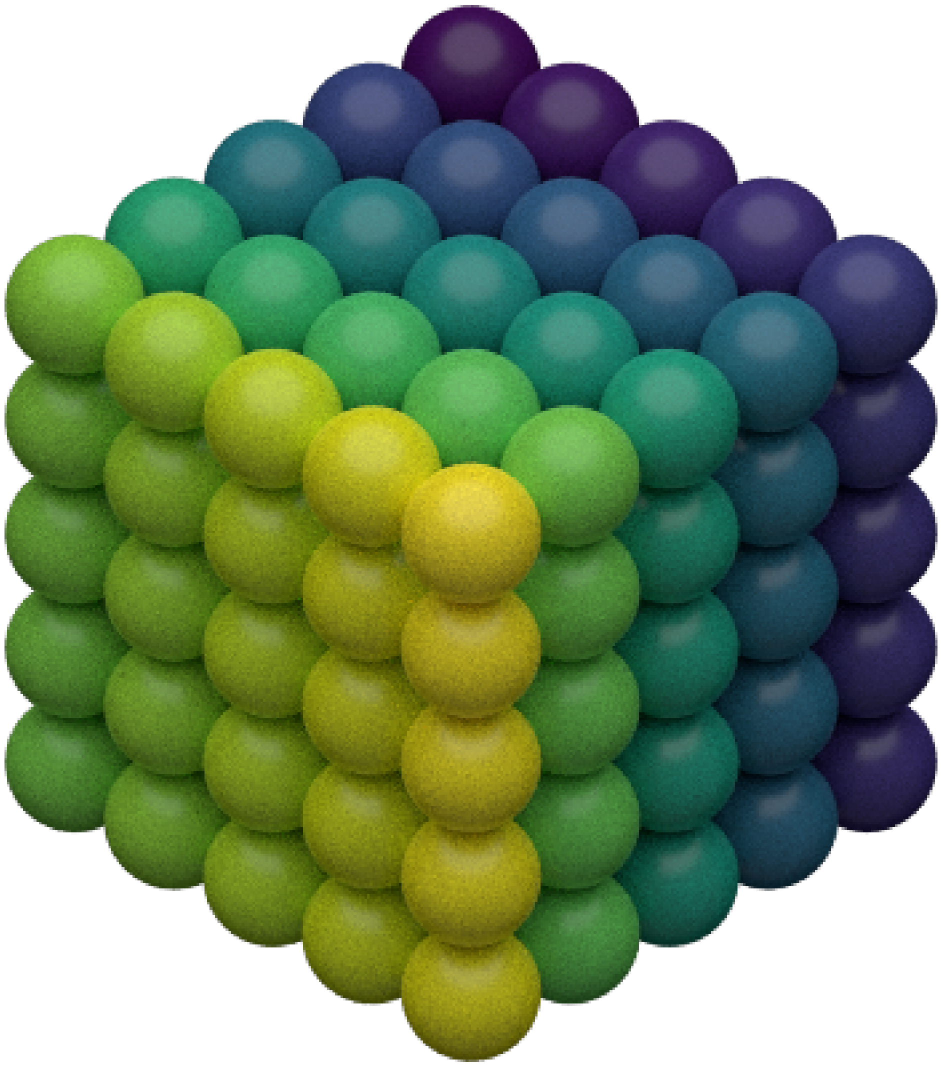
```
In [6]: scene.lights = fresnel.light.loop()
        tracer.sample(scene, samples=64, light_samples=10)
```



Butterfly lighting

Butterfly lighting places the key light high above the camera.

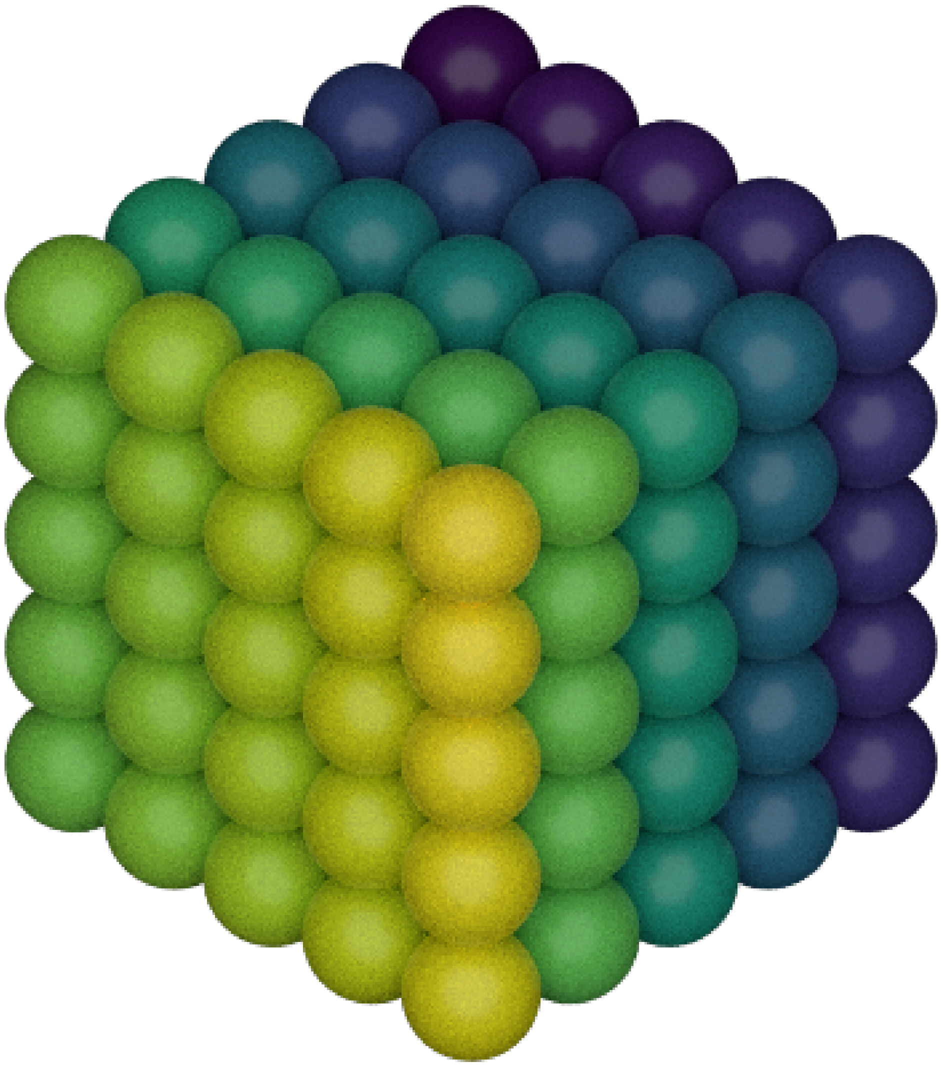
```
In [7]: scene.lights = fresnel.light.butterfly()
        tracer.sample(scene, samples=64, light_samples=10)
```



Ring lighting

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

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

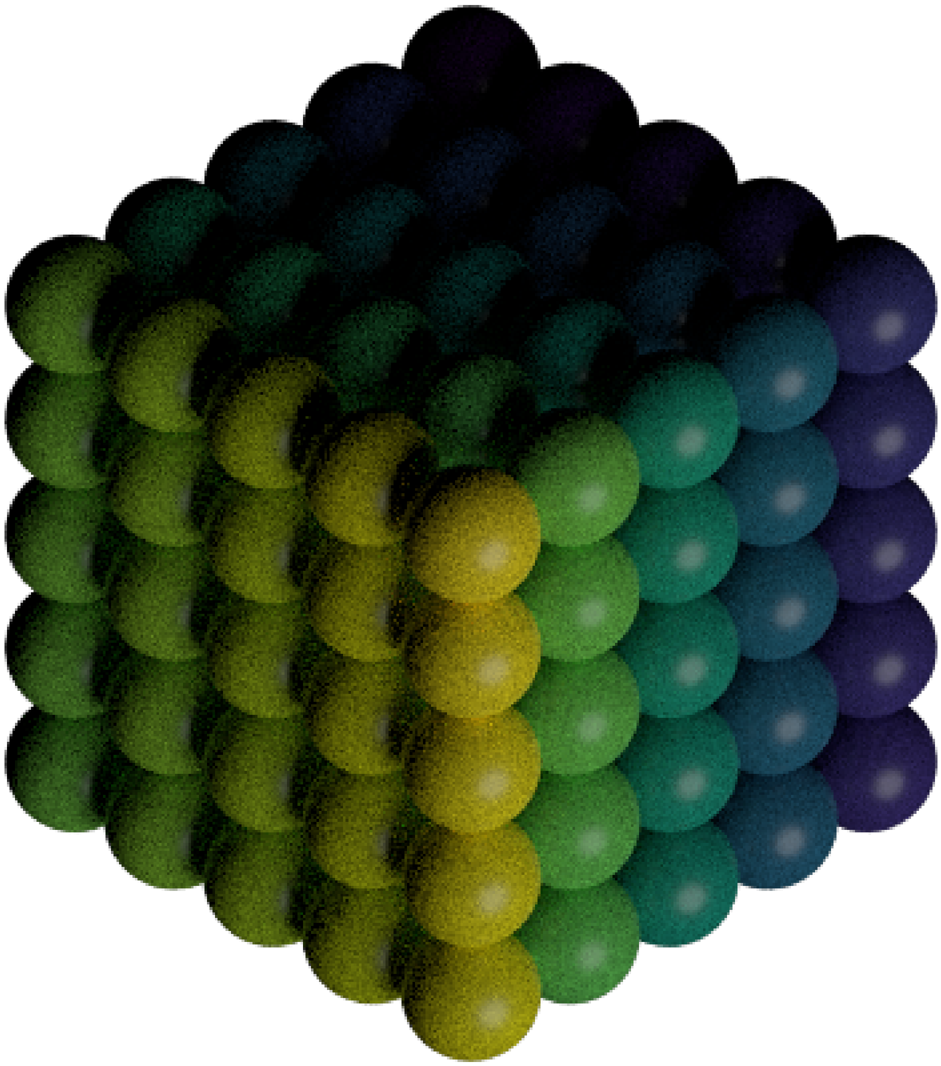


1.9.2 Custom lights

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.

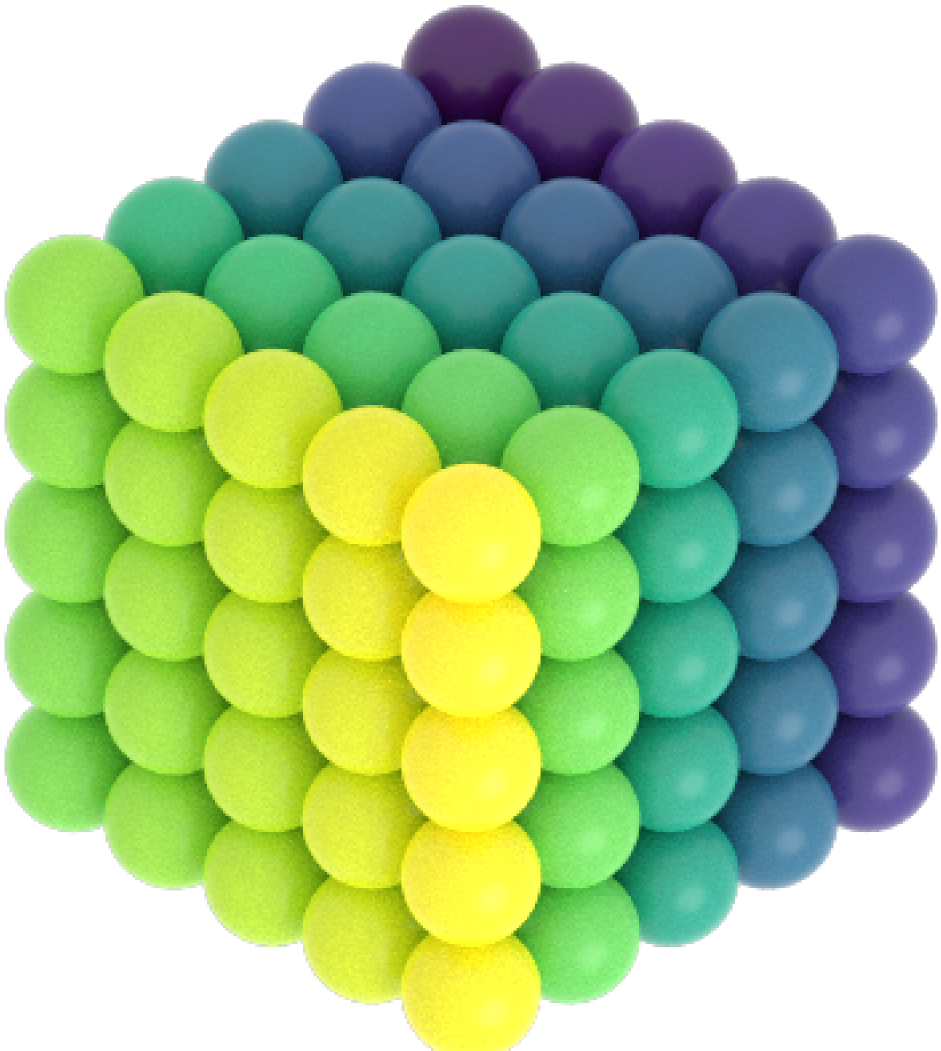
```
In [9]: my_lights = [fresnel.light.Light(direction=(1,-1,1), color=(1,1,1))]
        scene.lights = my_lights
```

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

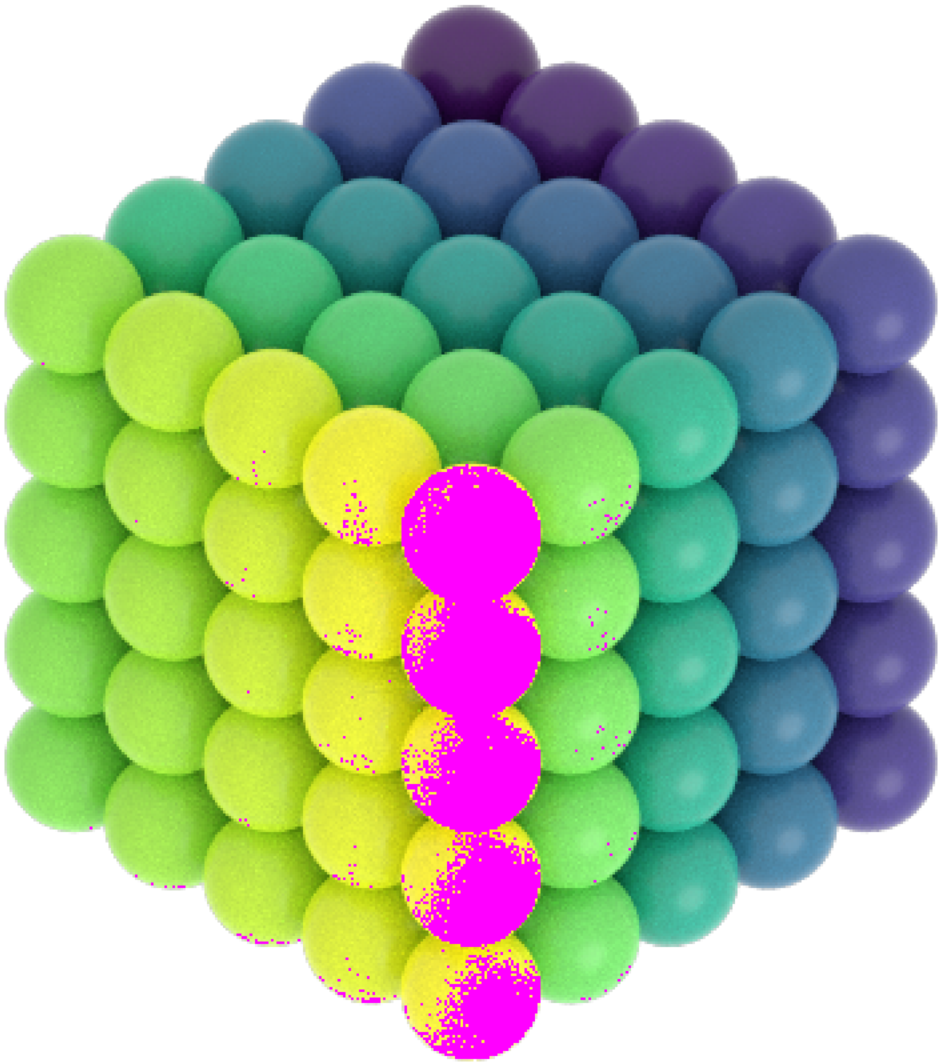
```
In [11]: scene.lights.append(fresnel.light.Light(direction=(0,0,1), color=(1,1,1), theta=3.14))
         tracer.sample(scene, samples=64, light_samples=10)
```



This image is overexposed.

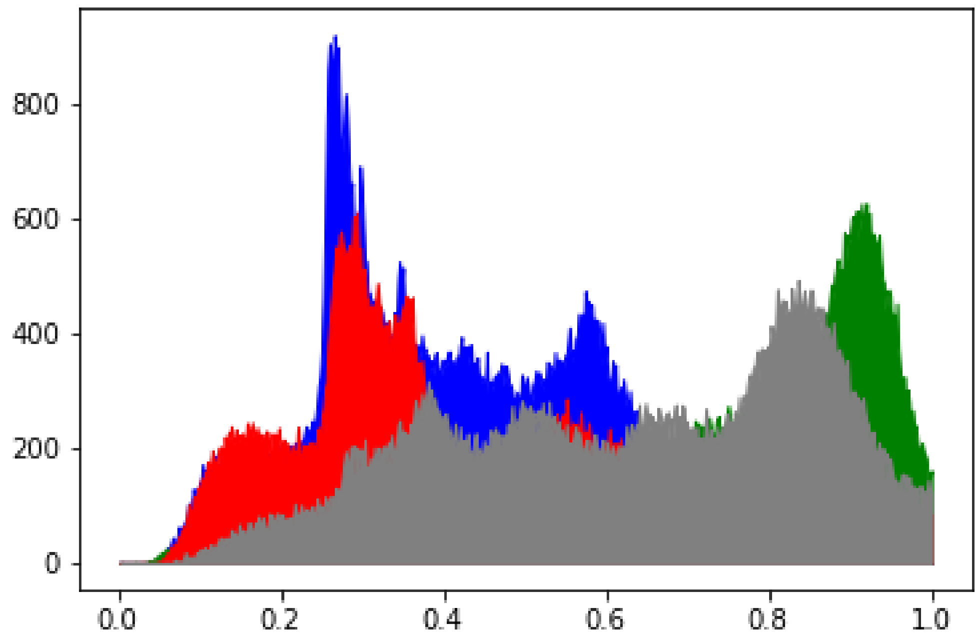
Highlight warnings show overexposed areas of the image as a special color (default: magenta).

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



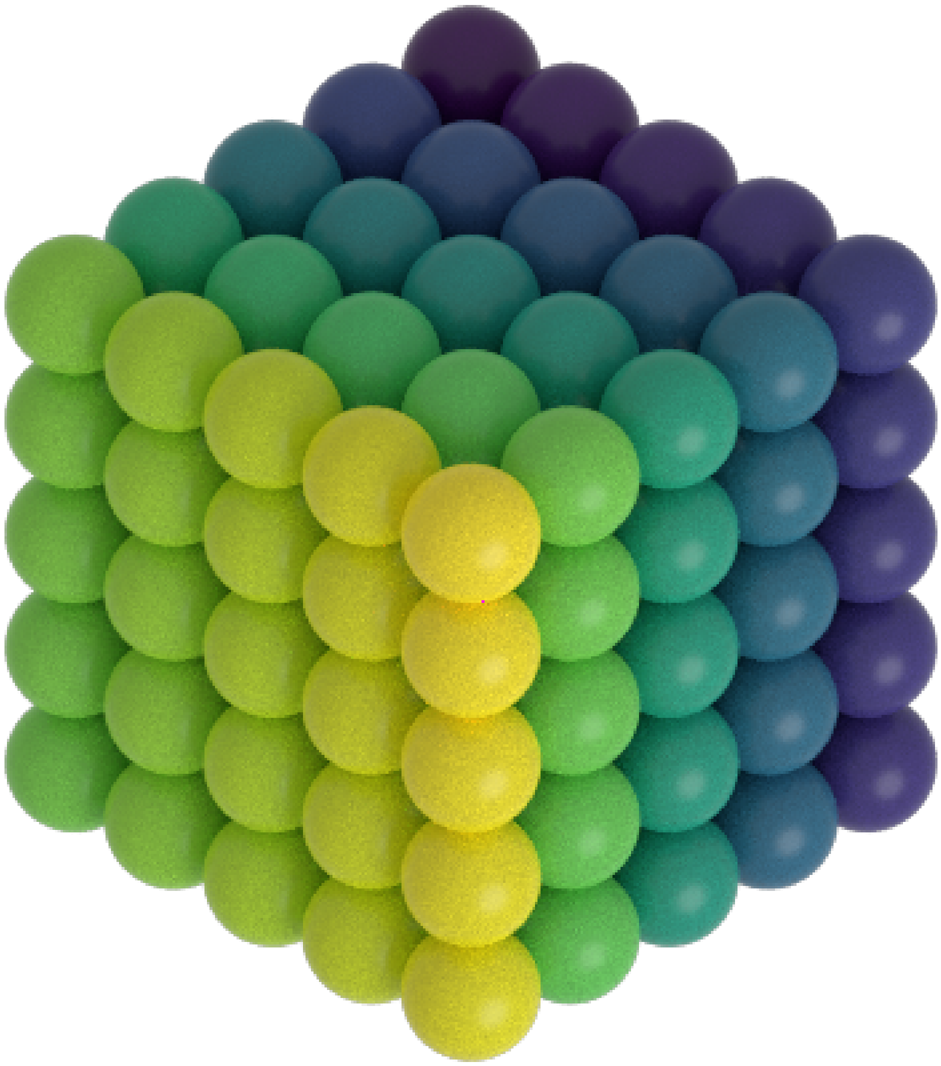
If the histogram is blocking up at 1.0, there are overexposed highlights.

```
In [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');
```



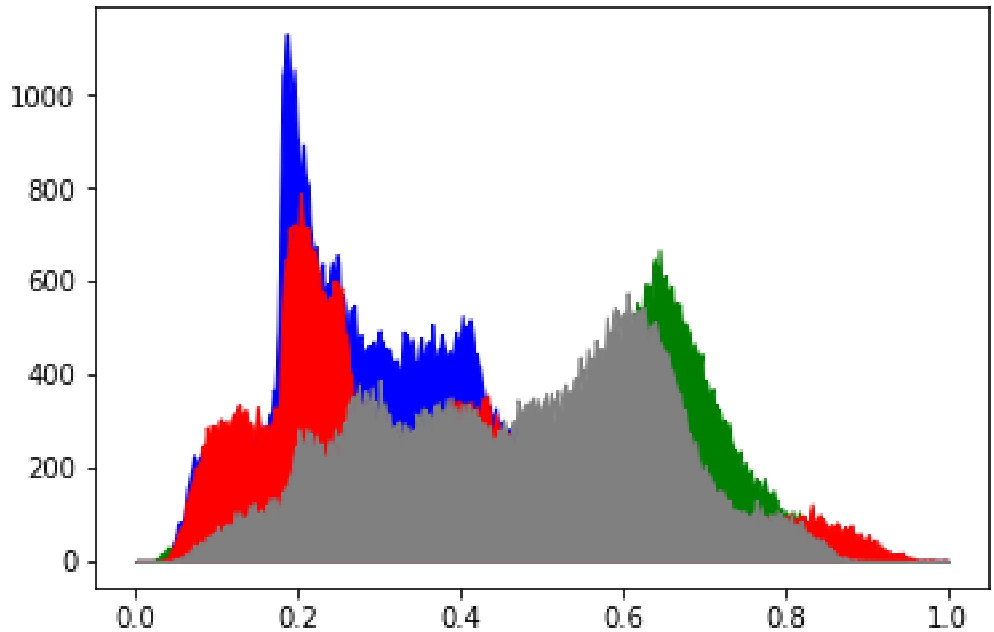
Reduce the intensity of the light to correctly expose the image.

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



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

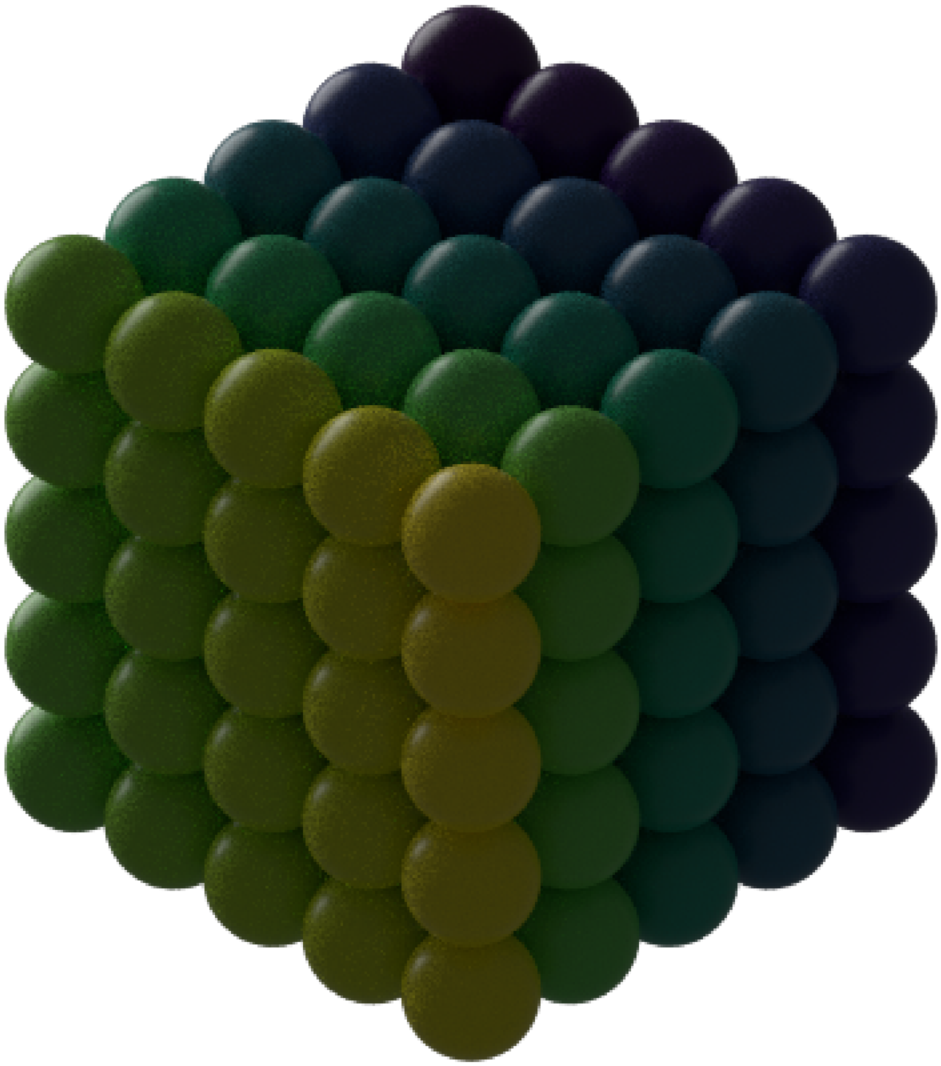
```
In [15]: L, bins = tracer.histogram()
         pyplot.fill_between(bins, L[:,3], color='blue');
         pyplot.fill_between(bins, L[:,2], color='green');
         pyplot.fill_between(bins, L[:,1], color='red');
         pyplot.fill_between(bins, L[:,0], color='gray');
```



`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:

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

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



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

1.10 Sphere

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

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

```
In [2]: 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)
```

1.10.1 Geometric properties

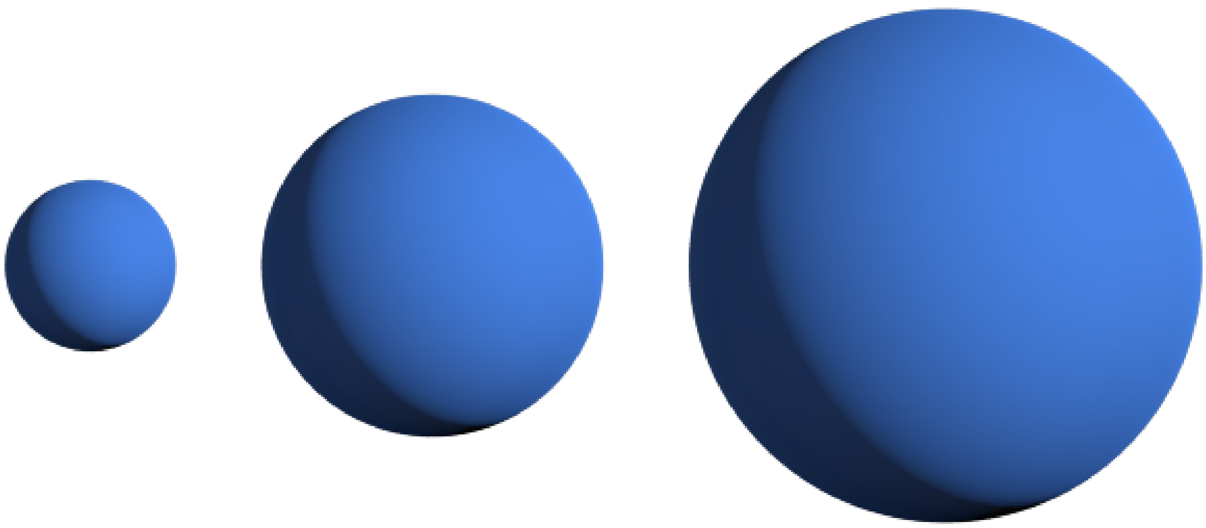
position defines the position of each sphere.

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

radius sets the radius of each sphere.

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

```
In [5]: scene.camera = fresnel.camera.fit(scene, view='front', margin=0.5)
        fresnel.preview(scene, aa_level=3)
```

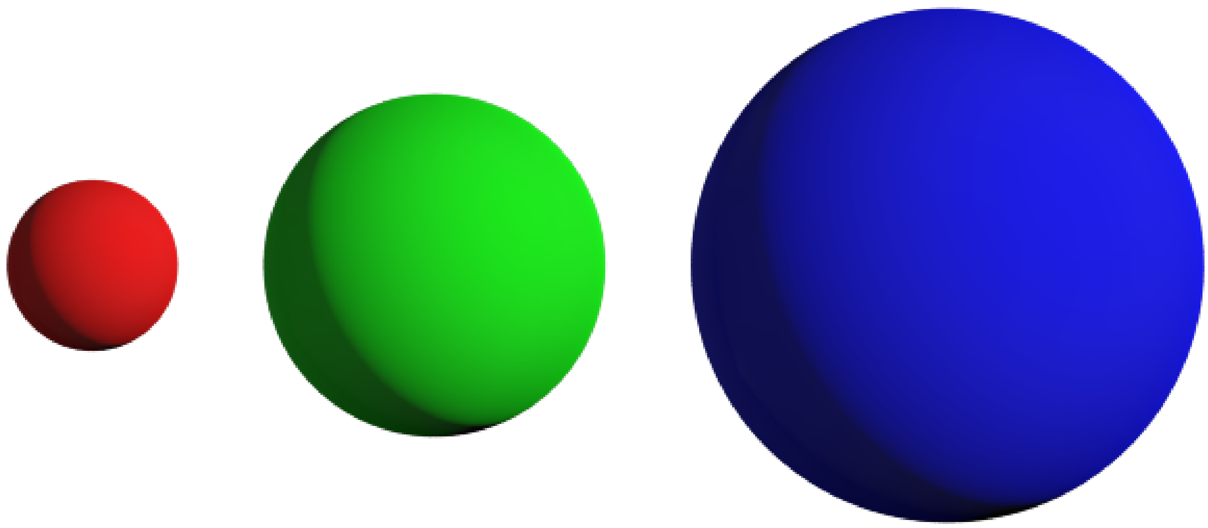


1.10.2 Color

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

```
In [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
```

```
In [7]: fresnel.preview(scene, aa_level=3)
```

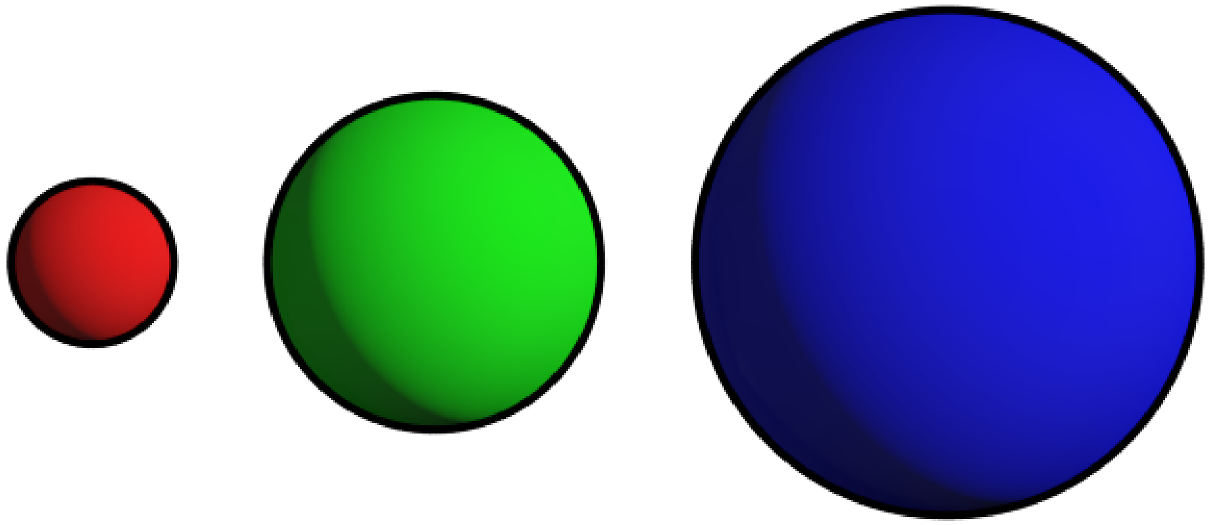


1.10.3 Outlines

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

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

```
In [9]: fresnel.preview(scene, aa_level=3)
```



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

1.11 Cylinder

```
In [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*.

```
In [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)
```

1.11.1 Geometric properties

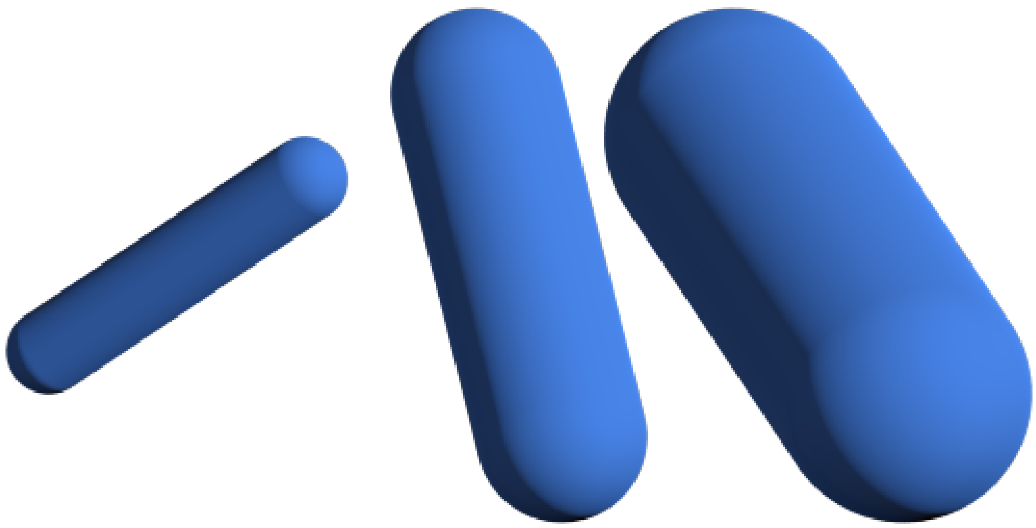
points defines the end points of each cylinder.

```
In [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.

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

```
In [5]: scene.camera = fresnel.camera.fit(scene, view='front', margin=0.5)
        fresnel.preview(scene, aa_level=3)
```

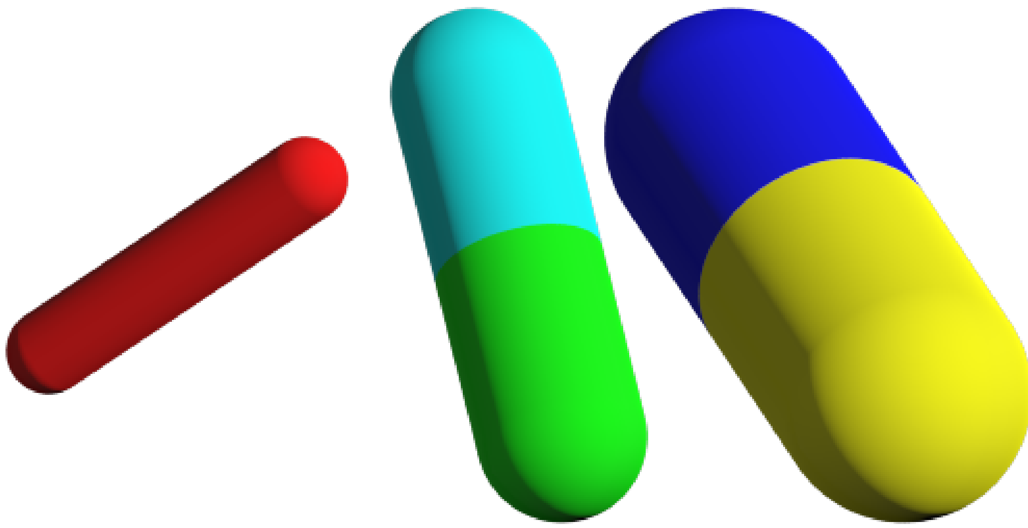


1.11.2 Color

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

```
In [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
```

```
In [7]: fresnel.preview(scene, aa_level=3)
```

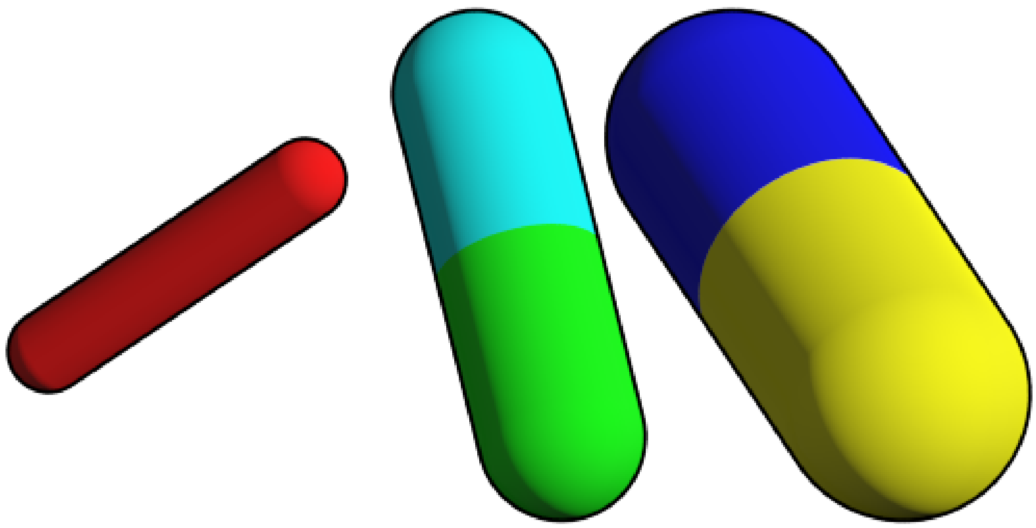


1.11.3 Outlines

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

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

```
In [9]: fresnel.preview(scene, aa_level=3)
```



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

1.12 Convex polyhedron

```
In [1]: import fresnel
        import math
        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 (**fresnel** currently doesn't attempt to calculate this).

The planes of the polyhedron are given as a set of P origins and outward pointing normals. Construct a truncated cube:

```
In [2]: origins=[];
        normals=[];
        colors=[];

        for v in [-1, 1]:
            origins.append([v, 0, 0])
            normals.append([v, 0, 0])
            colors.append(fresnel.color.linear([0.70,0.87,0.54])*0.8)

            origins.append([0, v, 0])
            normals.append([0, v, 0])
            colors.append(fresnel.color.linear([0.70,0.87,0.54])*0.8)

            origins.append([0, 0, v])
```

```
            normals.append([0, 0, v])
            colors.append(fresnel.color.linear([0.70,0.87,0.54])*0.8)

        for x in [-1,1]:
            for y in [-1,1]:
                for z in [-1,1]:
                    normals.append([x,y,z])
                    origins.append([x*0.75, y*0.75, z*0.75])
                    colors.append(fresnel.color.linear([0.65,0.81,0.89])*0.8)
```

```
In [3]: geometry = fresnel.geometry.ConvexPolyhedron(scene,
                                                     N=3,
                                                     origins=origins,
                                                     normals=normals,
                                                     face_colors=fresnel.color.linear(colors),
                                                     r=math.sqrt(3)
                                                    )
        geometry.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.9]),
                                                      roughness=0.8)
```

1.12.1 Geometric properties

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

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

orientation sets the orientation of each convex polyhedron as a quaternion

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

```
In [6]: scene.camera = fresnel.camera.fit(scene, view='front', margin=0.5)
        fresnel.preview(scene, aa_level=3)
```

1.12.2 Color

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

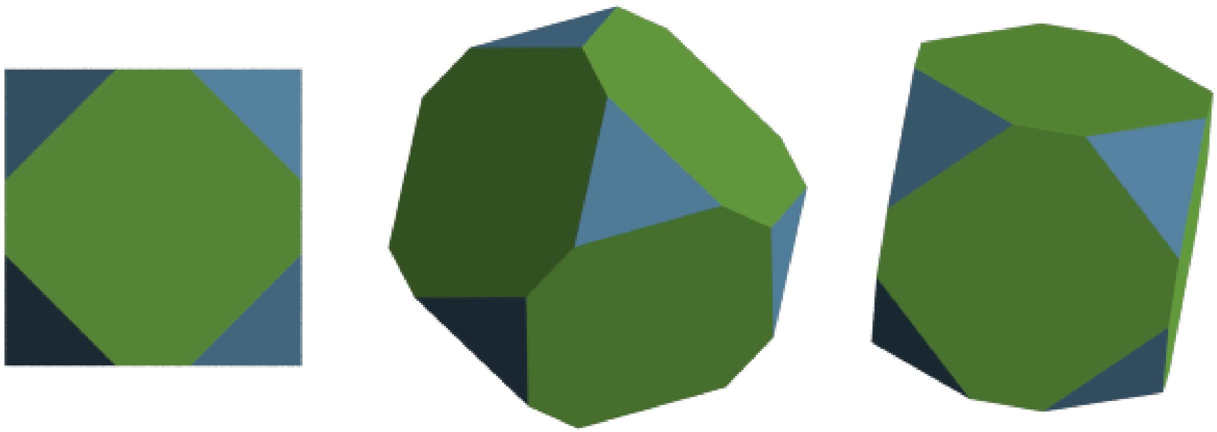
```
In [7]: geometry.color[:] = fresnel.color.linear([[0.9,0,0], [0, 0.9, 0], [0, 0, 0.9]])
        geometry.material.primitive_color_mix = 1.0
```

```
In [8]: fresnel.preview(scene, aa_level=3)
```

set **color_by_face** > 0 to color the faces of the polyhedra independently. **face_colors** (set above) sets the color of each face.

```
In [9]: geometry.color_by_face = 1.0
```

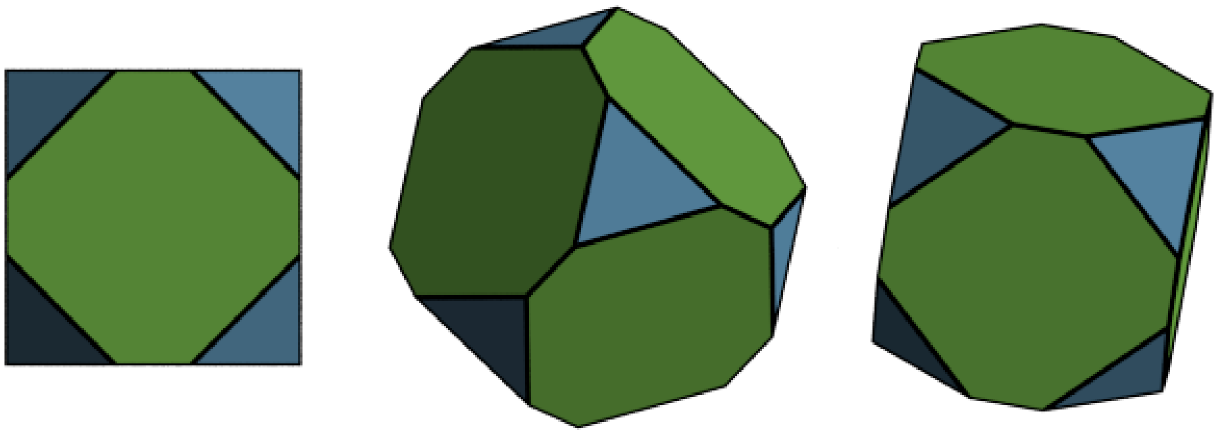
```
In [10]: fresnel.preview(scene, aa_level=3)
```



1.12.3 Outlines

Outlines are applied at the outer edge of each face.

```
In [11]: geometry.outline_width = 0.02

In [12]: fresnel.preview(scene, aa_level=3)
```



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

1.13 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.

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

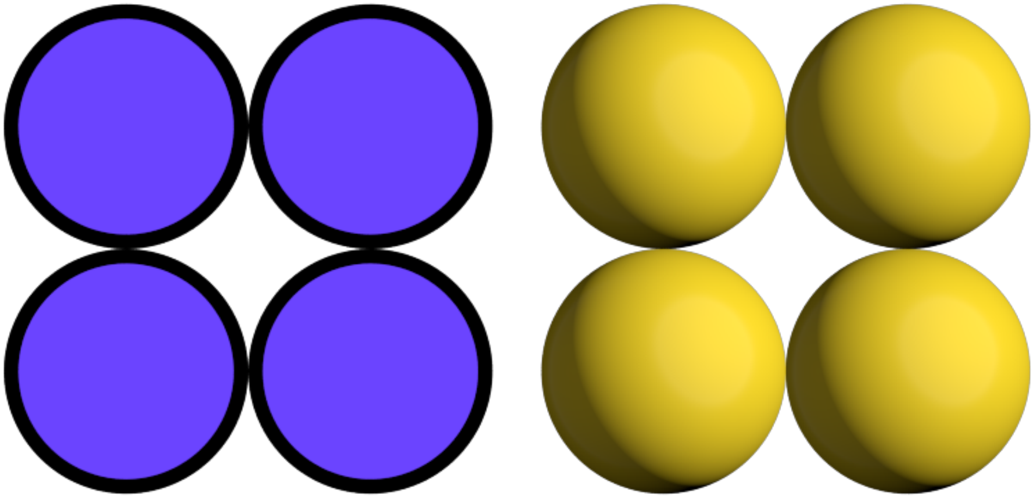
1.13.1 Create multiple geometries

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

```
In [2]: geom1 = fresnel.geometry.Sphere(scene, position = [[-3.2, 1, 0], [-3.2, -1, 0], [-1.2, 1, 0],
        geom1.material = fresnel.material.Material(solid=1.0, color=fresnel.color.linear([0.42,0.267,
        geom1.outline_width = 0.12
```

```
In [3]: geom2 = fresnel.geometry.Sphere(scene, position = [[3.2, 1, 0], [3.2, -1, 0], [1.2, 1, 0], [1
        geom2.material = fresnel.material.Material(solid=0.0, color=fresnel.color.linear([1,0.874,0.1
```

```
In [4]: fresnel.preview(scene, w=900, h=370, aa_level=3)
```

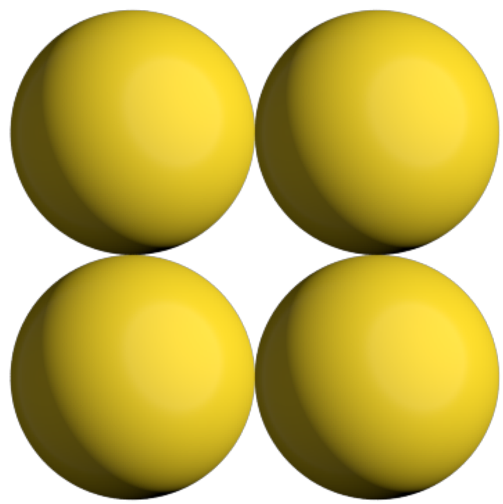


1.13.2 Disable geometries

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

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

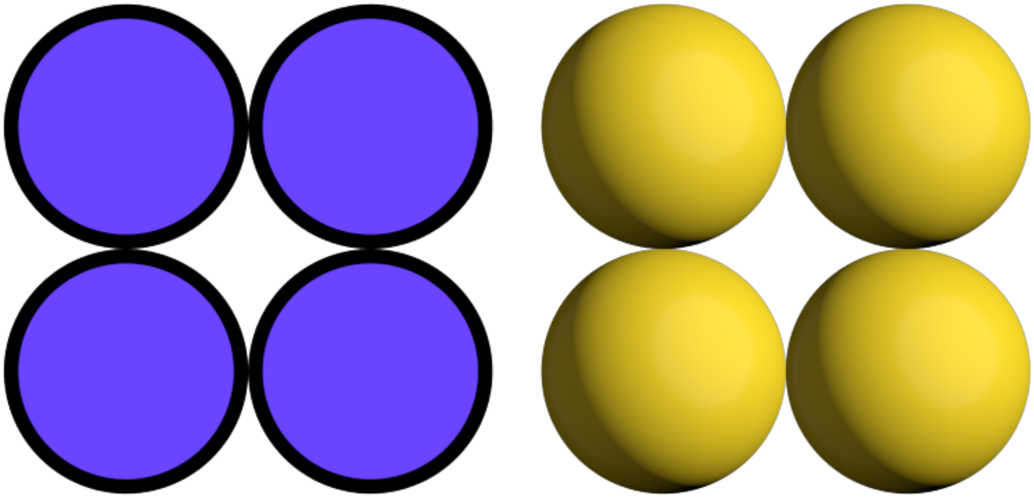
```
In [6]: fresnel.preview(scene, w=900, h=370, aa_level=3)
```



enable the geometry to make it appear again.

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

```
In [8]: fresnel.preview(scene, w=900, h=370, aa_level=3)
```

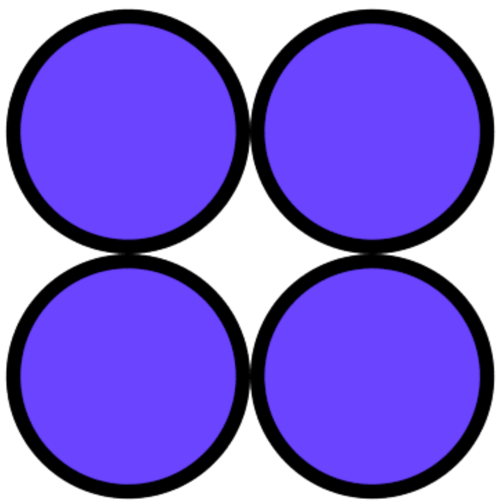


1.13.3 Remove geometry

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

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

```
In [10]: fresnel.preview(scene, w=900, h=370, aa_level=3)
```



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

1.14 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.

1.14.1 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**.

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

1.14.2 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.

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

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

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

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

```
 [0]:           Quadro GP100   56 SM_6.0 @ 1.44 GHz,  8276 MiB DRAM
 [1]:           Quadro GP100   56 SM_6.0 @ 1.44 GHz,  8276 MiB DRAM
```

1.14.3 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.

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

```
<fresnel.Device: Enabled OptiX devices:
 [0]:           Quadro GP100   56 SM_6.0 @ 1.44 GHz,  8276 MiB DRAM
 [1]:           Quadro GP100   56 SM_6.0 @ 1.44 GHz,  8276 MiB DRAM
>
```

```
In [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.

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

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

1.14.4 Attach a scene to a device

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

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

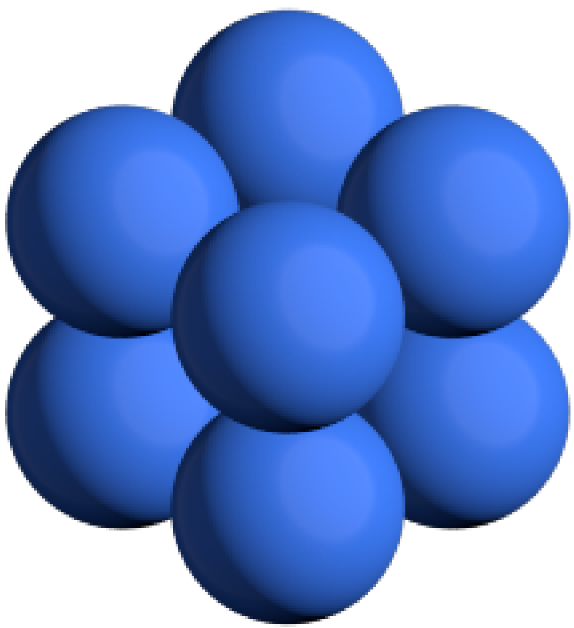
```
In [8]: scene_cpu = fresnel.Scene(device=cpu)
```

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

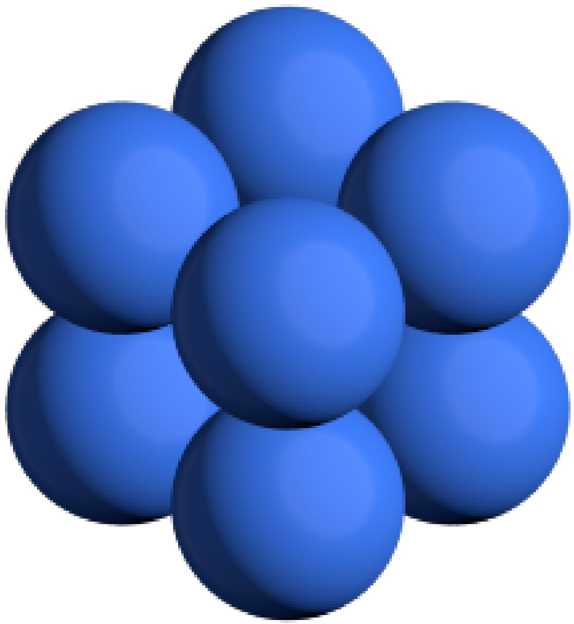
```
In [9]: for scene in [scene_cpu, scene_gpu]:
            geometry = fresnel.geometry.Sphere(scene, N=8, radius=1.0)
            geometry.position[:] = [[1,1,1],
                                    [1,1,-1],
                                    [1,-1,1],
                                    [1,-1,-1],
                                    [-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]))
```

Rendered output is essentially identical from the two devices.

```
In [10]: fresnel.preview(scene_gpu, w=300, h=300, aa_level=3)
```



```
In [11]: fresnel.preview(scene_cpu, w=300, h=300, aa_level=3)
```

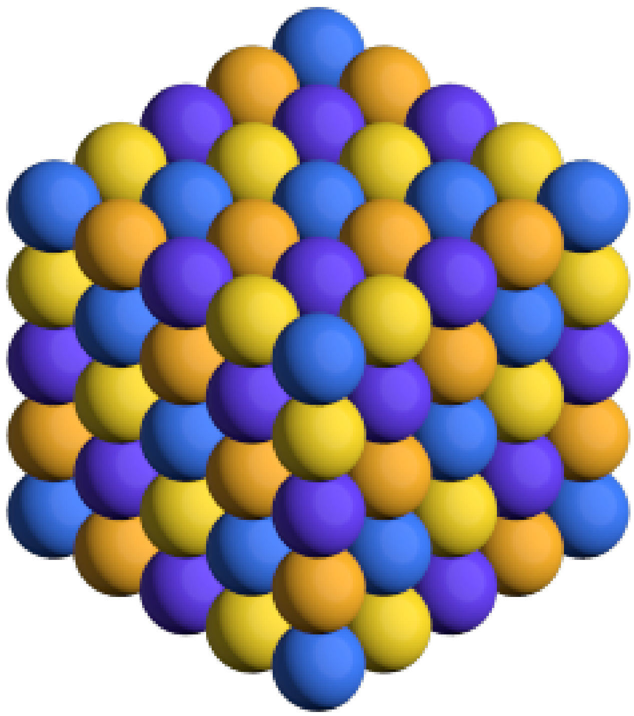


1.14.5 Memory consumption

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

```
In [12]: import math
         scene2_gpu = fresnel.Scene(device=gpu)
         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(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)
```

```
In [13]: fresnel.preview(scene2_gpu, w=300, h=300, aa_level=3)
```



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

1.15 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.

In [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,
geometry.outline_width = 0.12
scene.camera = fresnel.camera.fit(scene, view='front', margin=0.2)
```

1.15.1 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.

```
In [2]: tracer = fresnel.tracer.Preview(device=device, w=300, h=300, aa_level=3)
```

Rendering and accessing output images

The **render** method renders the output.

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

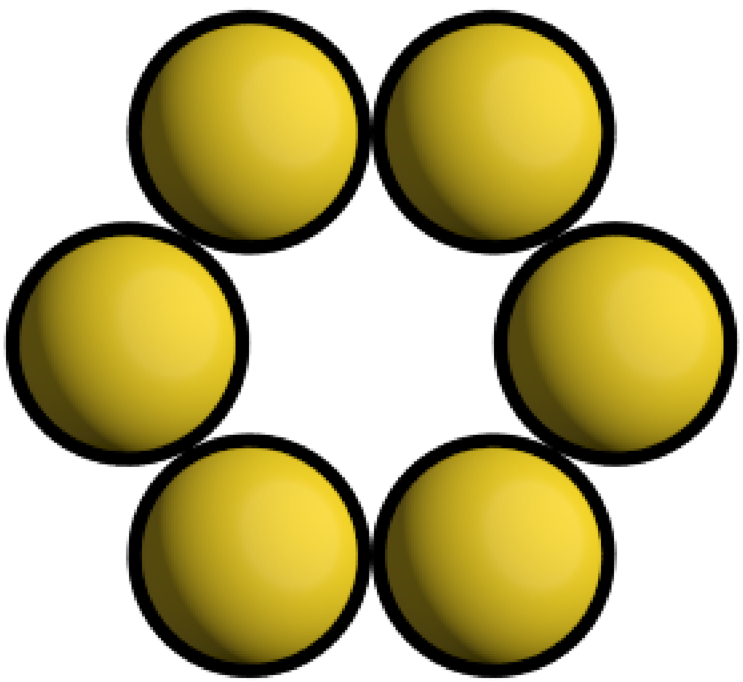
The return value of **render** is a proxy reference to the internal image buffer of the **Tracer**. You can access with a **numpy** array like interface.

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

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

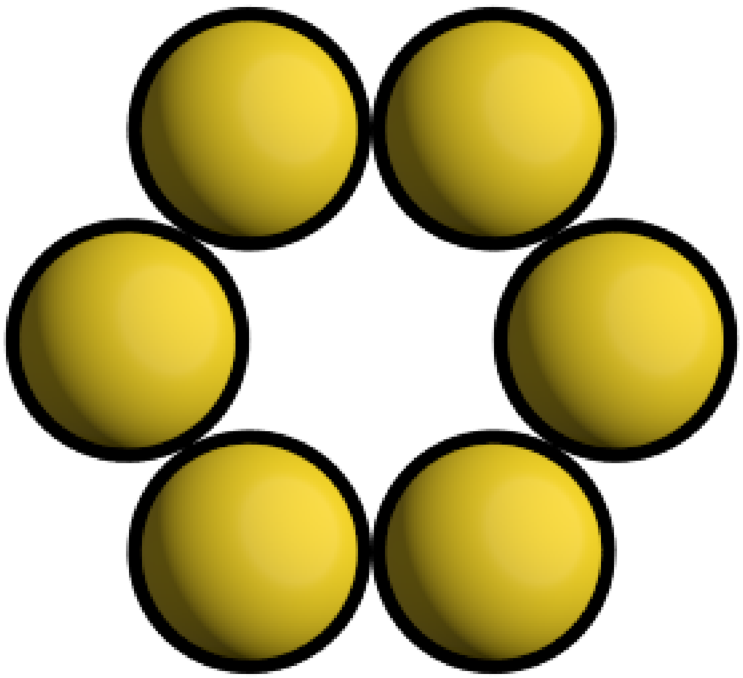
The output object also provides an interface for **jupyter** to display the image.

```
In [5]: out
```



tracer.output also accesses the output buffer.

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

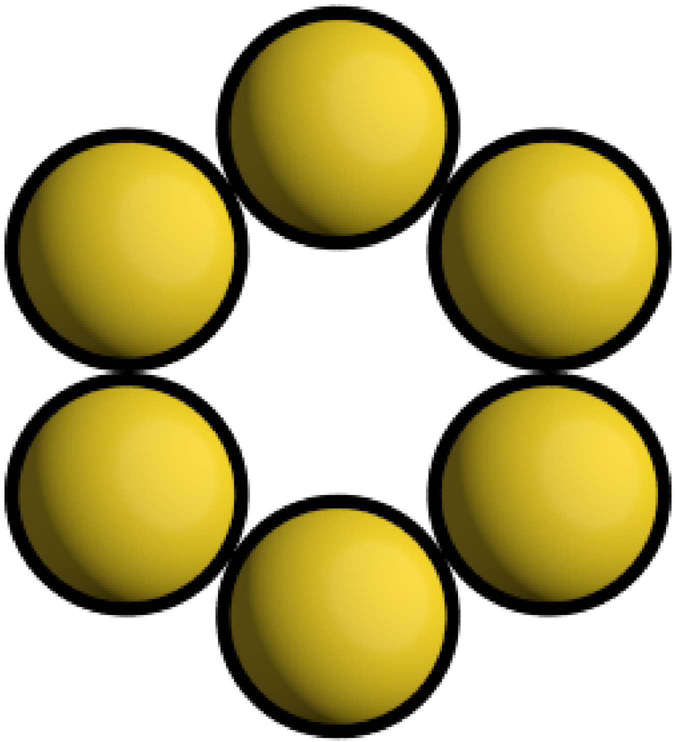


The tracer can render a modified scene without the initialization overhead.

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

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

```
In [8]: out
```



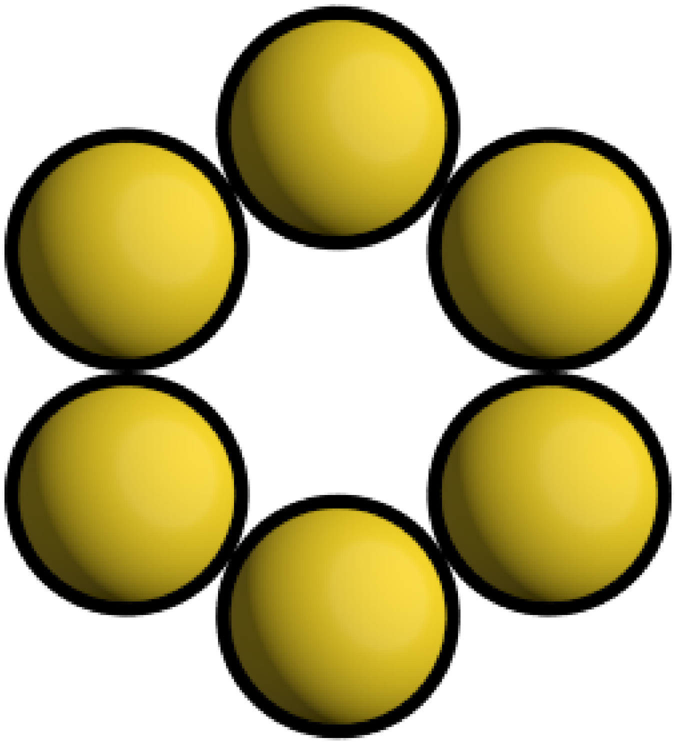
Evaluate image exposure

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

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

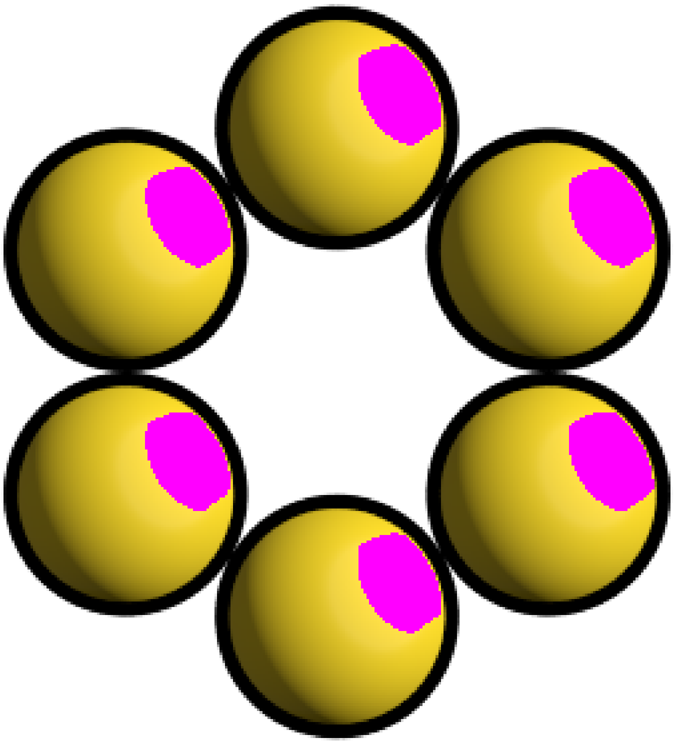
The test image is exposed correctly, there are no warning pixels.

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



Make the main light brighter to show the highlight warnings.

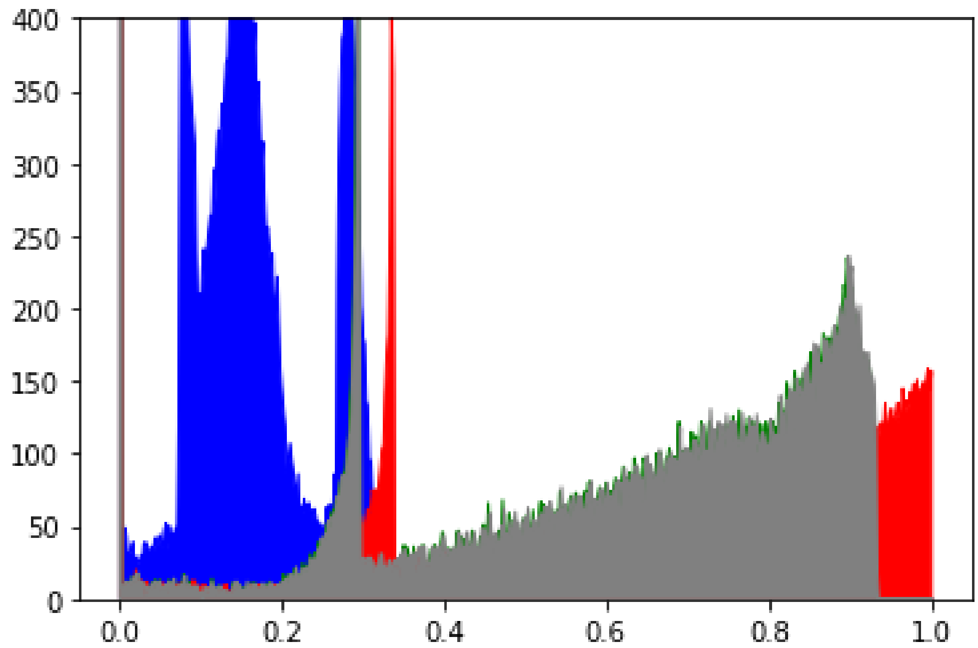
```
In [11]: scene.lights[0].color = (1.2, 1.2, 1.2)
         tracer.render(scene)
```



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

```
In [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)
```

```
Out[12]: (-0.04794921875, 1.04990234375, 0, 400)
```




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

Resizing the output buffer

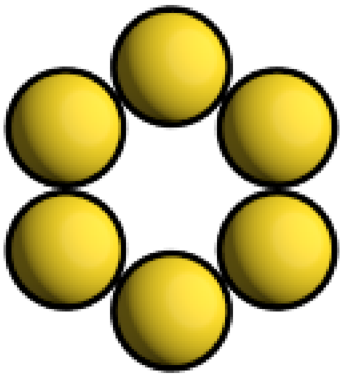
Call **resize** to set a new size for the output. When the image is resized, any existing rendered output is lost.

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

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

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

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



1.15.2 The Preview tracer

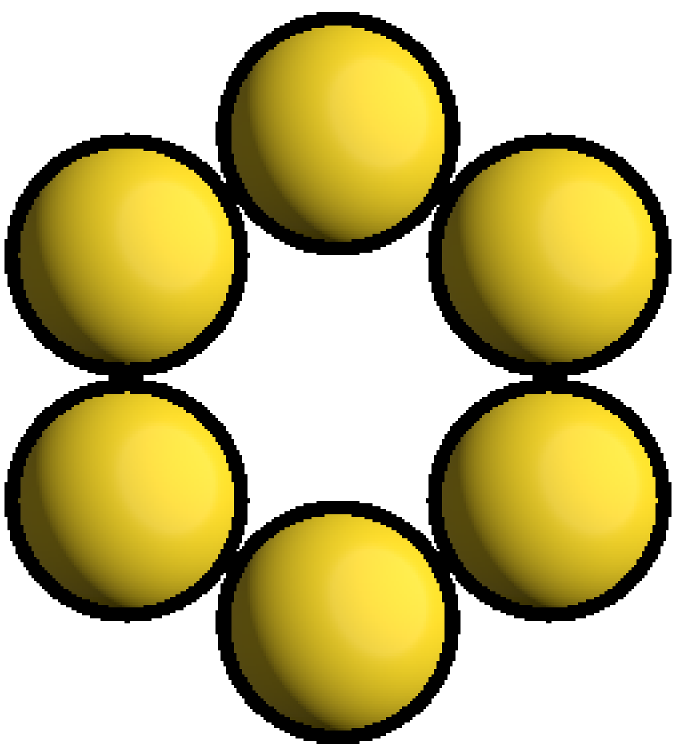
The **Preview** tracer renders output images quickly with approximate lighting effects.

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

The *aa_level* attribute controls the strength of the anti-aliasing.

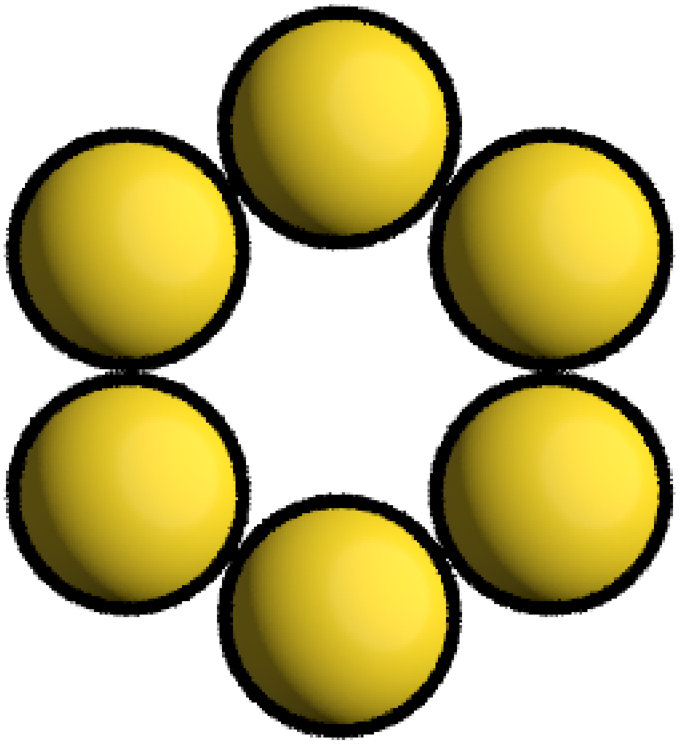
```
In [18]: tracer.aa_level
Out[18]: 0
```

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

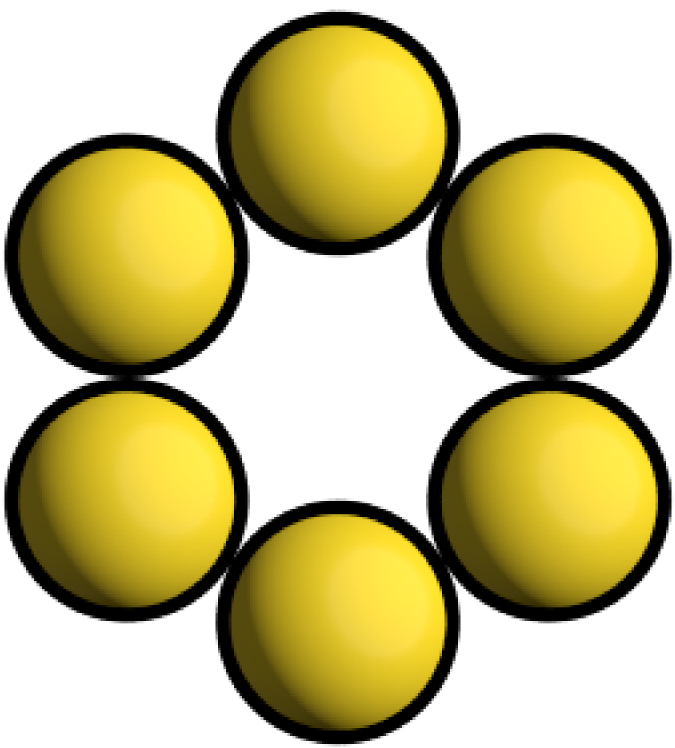


```
In [20]: tracer.aa_level = 1
```

```
In [21]: tracer.render(scene)
```



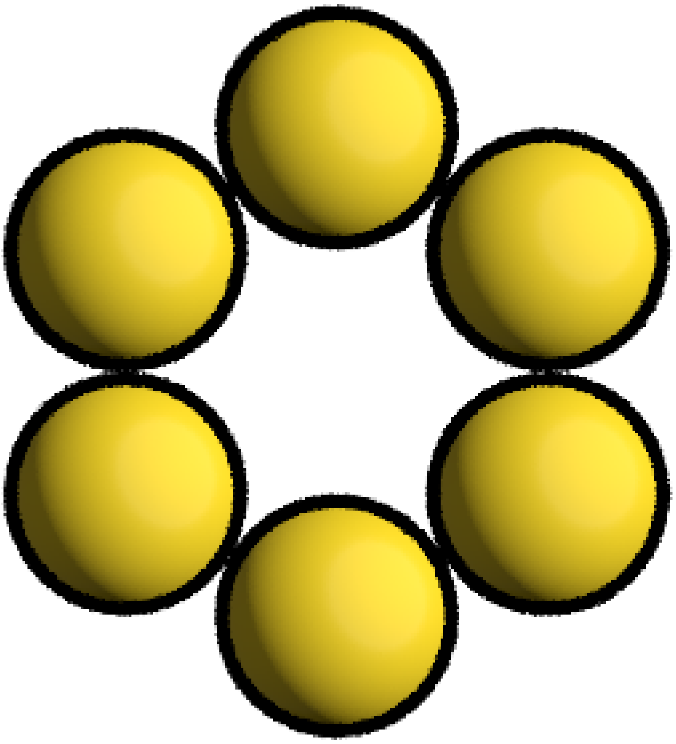
```
In [22]: tracer.aa_level = 3

In [23]: tracer.render(scene)
```



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

```
In [24]: tracer.seed = 12
         tracer.aa_level = 1
```

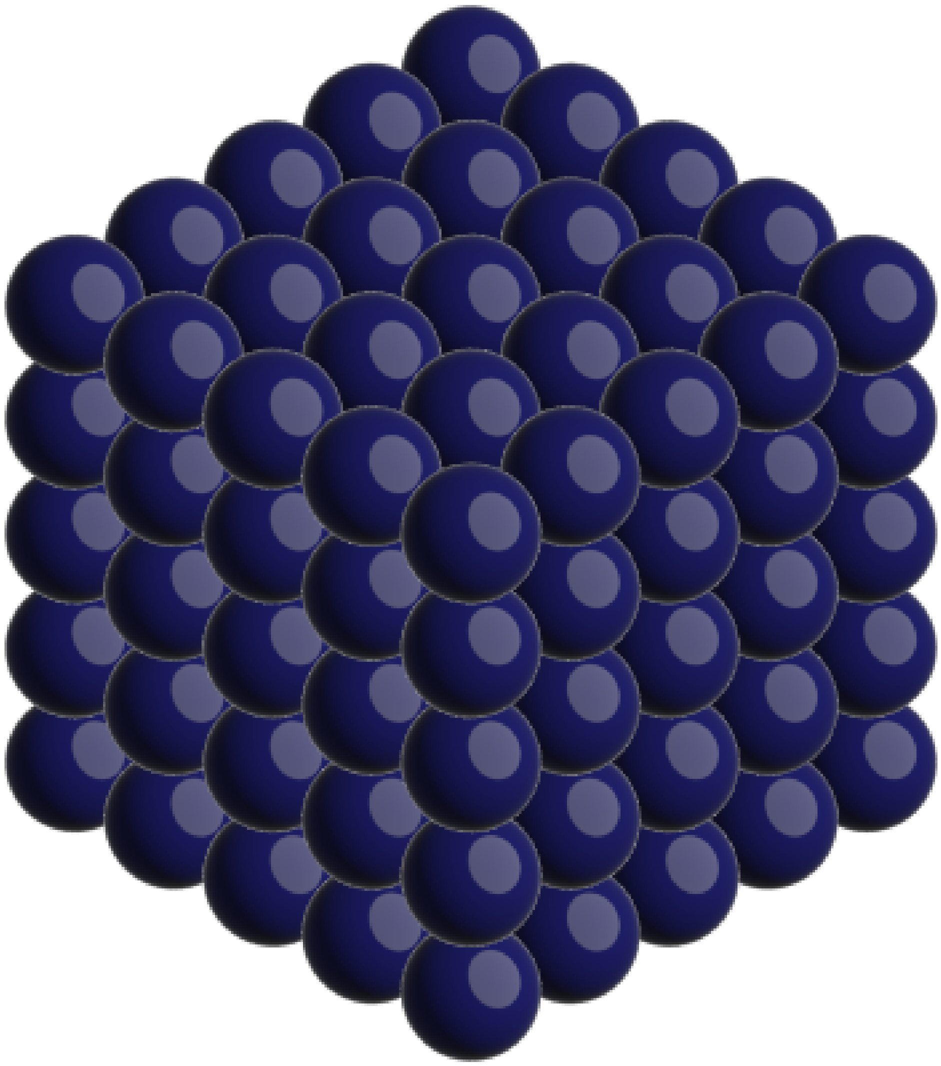
```
In [25]: tracer.render(scene)
```



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

```
In [28]: 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)
```

```
In [29]: tracer.resize(w=450, h=450)
         tracer.aa_level = 3
         tracer.render(scene)
```



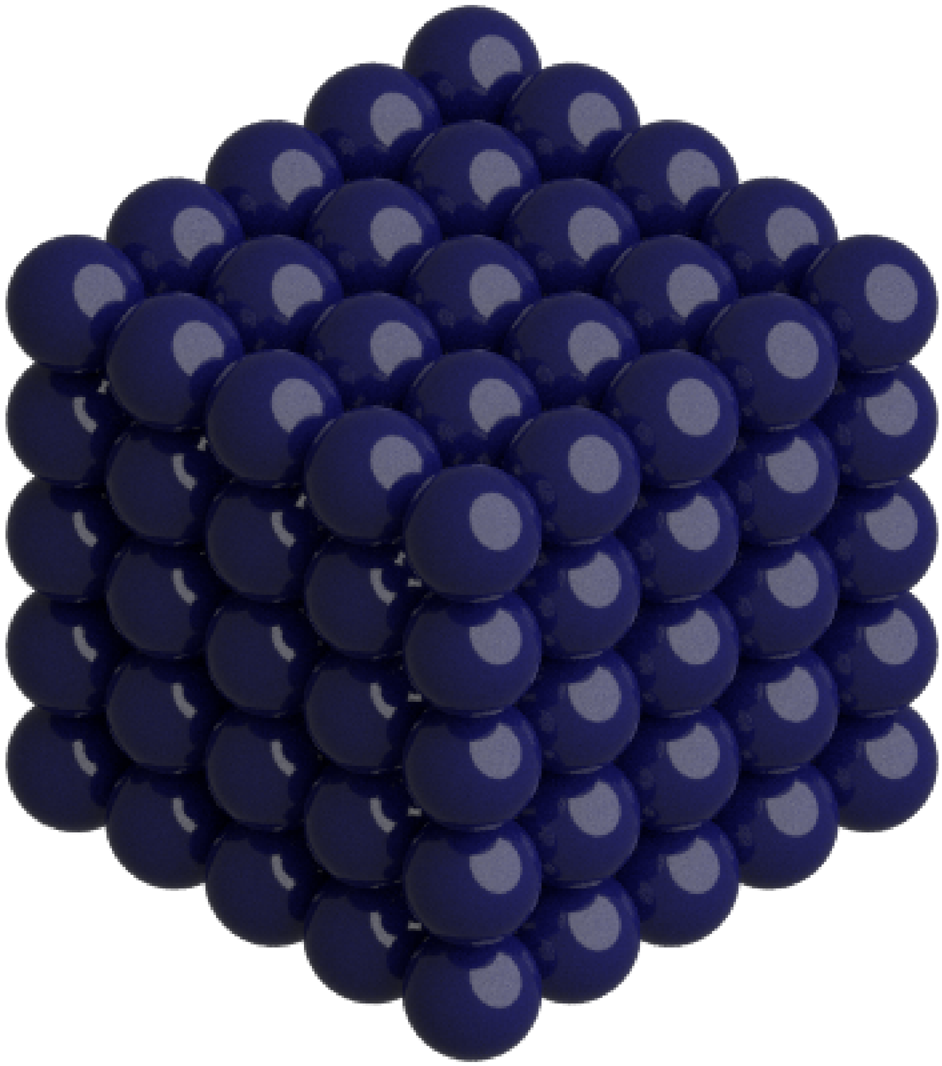
1.15.3 The Path tracer

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

Here is the same scene with the path tracer:

```
In [30]: path_tracer = fresnel.tracer.Path(device=device, w=450, h=450)
```

```
In [31]: path_tracer.sample(scene, samples=64, light_samples=40)
```

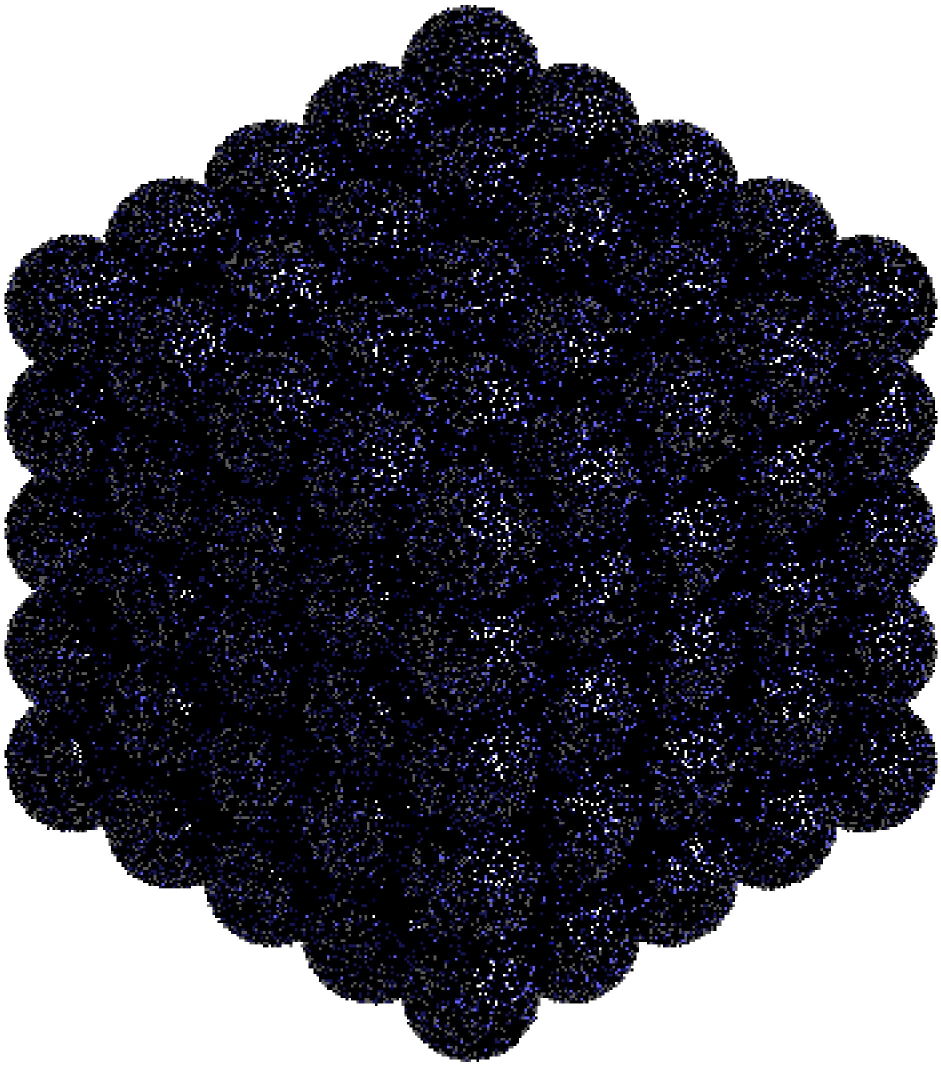


The **Path** tracer performs many independent samples and averages them together. **reset()** starts averaging a new image.

```
In [32]: path_tracer.reset()
```

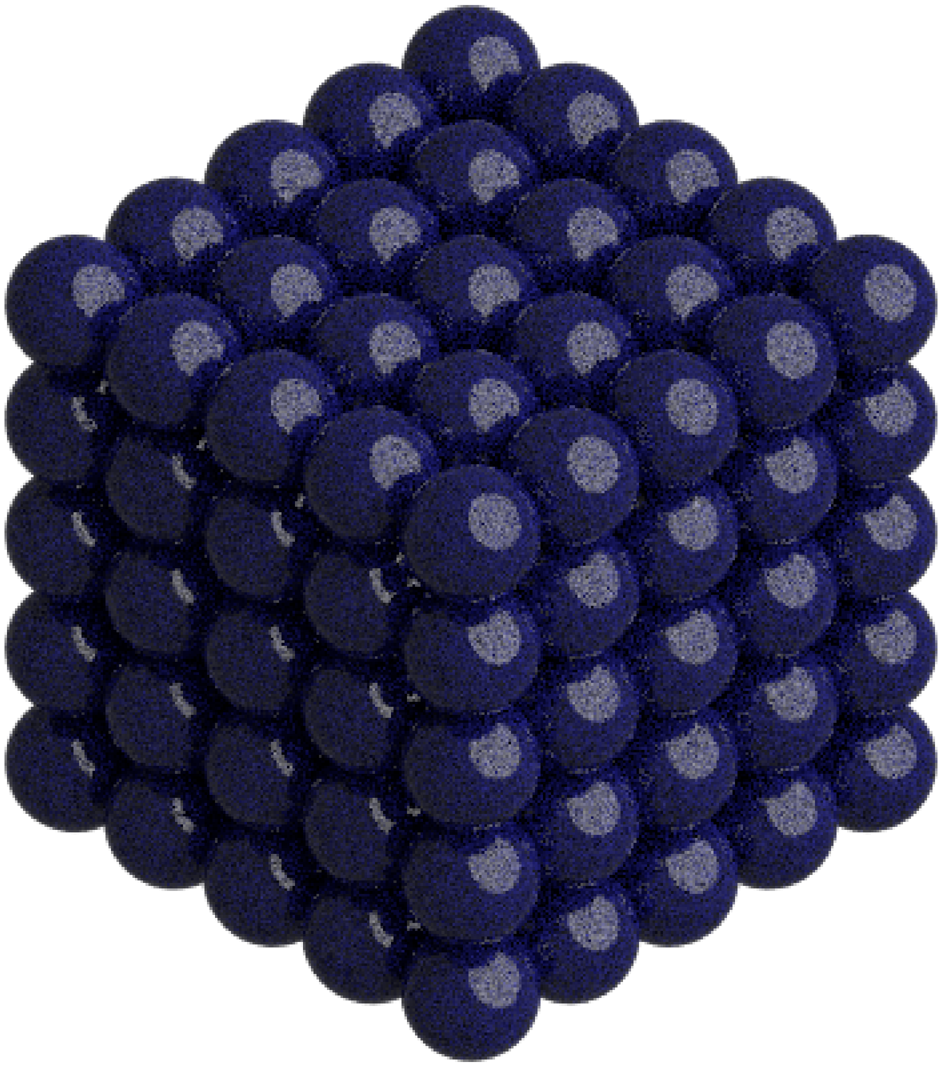
render() accumulates a single sample into the resulting image.

```
In [33]: path_tracer.render(scene)
```



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

```
In [34]: for i in range(64):
             path_tracer.render(scene)

         path_tracer.output
```



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

1.16 fresnel

Overview

fresnel.Device	Hardware device to use for ray tracing.
fresnel.Scene	Content of the scene to ray trace.

Details

The fresnel ray tracing package.

fresnel.**__version__**
str – Fresnel version

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* sets no limit.

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*.

When mode is *auto*, the default, *Device* GPU rendering and fall back on CPU rendering if there is no GPU available or GPU support was not compiled in. 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_modes
list – List of the available execution modes (static member).

available_gpus
list – List of the available gpus (static member).

mode
string – The active mode

class fresnel.**Scene**(*device=None*, *camera='auto'*, *lights=[<fresnel.light.Light object>, <fresnel.light.Light object>]*)
Content of the scene to ray trace.

Parameters **device** (*Device*) – Device to create this Scene on.

Scene defines the contents of the scene to be ray traced, including any number of *geometry* objects, the *camera*, *background color*, *background alpha*, and the *lights*.

Every *Scene* attaches to a *Device*. For convenience, *Scene* creates a default *Device* when **device** is *None*. If you want a non-default device, you must create it explicitly.

Lights

lights is a sequence of up to 4 directional lights that apply to the scene globally. Each light has a direction and color. You can assign lights using one of the predefined setups:

```
scene.lights = fresnel.light.butterfly()
```

You can assign a sequence of *Light* objects:

```
scene.lights = [fresnel.light.Light(direction=(1,2,3))]
```

You can modify the lights in place:

```
>>> print(len(scene.lights))
2
>>> l.append(fresnel.light.Light(direction=(1,0,0), color=(1,1,1)))
>>> print(len(3))
1
>>> print(l[2]).direction
(1,0,0)
>>> l[0].direction = (-1,0,0)
>>> print(l[0]).direction
(-1,0,0)
```

device
Device – Device this Scene is attached to.

camera
camera.Camera – Camera view parameters, or 'auto' to automatically choose a camera.

background_color
tuple[float] – Background color (r,g,b) as a tuple or other 3-length python object, in the linearized color space. Use *fresnel.color.linear()* to convert standard sRGB colors

background_alpha
float – Background alpha (opacity).

lights
list of *light.Light* – Globals lights in the scene.

get_extents()
Get the extents of the scene

Returns

[[minimum x, minimum y, minimum z], [maximum x, maximum y, maximum z]]

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.
- **h** (*int*) – Output image height.
- **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 to rendering output with the *Path* tracer. See the *Path* tracer for a complete description.

`fresnel.preview(scene, w=600, h=370, aa_level=0)`
Preview a scene.

Parameters

- **scene** (*Scene*) – Scene to render.
- **w** (*int*) – Output image width.
- **h** (*int*) – Output image height.
- **aa_level** (*int*) – Amount of anti-aliasing to perform

preview() is a shortcut to rendering output with the *Preview* tracer. See the *Preview* tracer for a complete description.

Modules

1.16.1 fresnel.camera

Overview

fresnel.camera.Camera
fresnel.camera.fit
fresnel.camera.orthographic

Details

Cameras.

class `fresnel.camera.Camera(_camera=None)`
Defines the view into the *Scene*.

Use one of the creation functions to create a *Camera*:

- *orthographic()*

The camera is a property of the *Scene*. You may read and modify any of these camera attributes.

position
tuple – the position of the camera (the center of projection).

look_at
tuple – the point the camera looks at (the center of the focal plane).

up
tuple – a vector pointing up.

height
the height of the image plane.

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. *Camera* space shares units with *Scene* space.

TODO: Move description of spaces to an overview page and create figures.

`fresnel.camera.`**`fit`**(*scene*, *view='auto'*, *margin=0.05*)

Fit a camera to a `Scene`

Create a camera that fits the entire hight of the scene in the image plane.

Parameters

- **`scene`** (`Scene`) – The scene to fit the camera to.
- **`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.

`fresnel.camera.`**`orthographic`**(*position*, *look_at*, *up*, *height*)

Orthographic camera

Parameters

- **`position`** (`tuple`) – the position of the camera.
- **`look_at`** (`tuple`) – the point the camera looks at (the center of the focal plane).
- **`up`** (`tuple`) – a vector pointing up.
- **`height`** – 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*. Objects in front of the plane will appear in the rendered image, objects behind the plane will not.

up is a vector in `Scene` space that defines which direction points up (+y direction in the image). *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 in `Scene` units. The image width is determined by the aspect ratio of the image. The area *width* by *height* about the *look_at* point will be included in the rendered image.

TODO: show a figure

1.16.2 fresnel.color

Overview

`fresnel.color.linear`	Convert a sRGB color (or array of such colors) from the gamma corrected color space into the linear space.

Details

Color utilities.

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

Convert a sRGB color (or array of such colors) from the gamma corrected color space into the linear space.

Standard tools for working with RGB colors provide gamma corrected values. fresnel needs to perform calculations in a linear color space. This method converts a sRGB into that linear space. Use `linear()` when specifying material or particle colors with sRGB inputs.

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

Parameters **color** (*tuple*) – 3-length (or Nx3, or Nx4) list, or other object convertible to a numpy array (in the range 0-1).

Returns A numpy array with the linearized color.

1.16.3 fresnel.geometry

Overview

fresnel.geometry.Geometry	Base class for all geometry.
fresnel.geometry.Cylinder	Cylinder geometry.
fresnel.geometry.ConvexPolyhedron	Convex polyhedron geometry.
fresnel.geometry.Sphere	Sphere geometry.

Details

Geometric primitives.

class fresnel.geometry.**ConvexPolyhedron**(*scene*, *origins*, *normals*, *r*, *face_colors=None*, *position=None*, *orientation=None*, *color=None*, *N=None*, *material=<fresnel.material.Material object>*, *outline_material=<fresnel.material.Material object>*, *outline_width=0.0*)

Convex polyhedron geometry.

Define a set of convex polyhedron primitives. A convex polyhedron is defined by P outward facing planes (origin and normal vector) and a radius that encompass the shape.

Note: Future versions may (or may not) provide a more user friendly interface.

Parameters

- **scene** (*fresnel.Scene*) – Add the geometry to this scene
- **origins** – Origins of the planes in particle local coordinates. **Type:** anything convertible by numpy to a Px3 array of floats.
- **normals** – Origins of the planes in particle local coordinates. **Type:** anything convertible by numpy to a Px3 array of floats.
- **r** (*float*) – Radius of the circumscribing sphere (centered at the origin) that encompasses the polyhedron.
- **face_colors** – Colors of the polyhedron faces **Type:** anything convertible by numpy to a Px3 array of floats.
- **position** – Positions of the polyhedra, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.

- **orientation** – Rotation quaternion of each polyhedron, *optional*. **Type:** anything convertible by numpy to a Nx4 array of floats.
- **color** – (r,g,b) color of each particle, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.
- **N** (*int*) – Number of spheres in the geometry. If `None`, determine `N` from `position`.

Note: The constructor arguments `position`, `orientation`, and `color` are optional, and just short-hand for assigning the attribute after construction.

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

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

---

position
fresnel.util.array – Read or modify the positions of the prisms.

orientation
fresnel.util.array – Read or modify the orientations of the prisms.

color
fresnel.util.array – Read or modify the color of the prisms.

color_by_face
float – Set to 0 to color particles by the per-particle color. Set to 1 to color faces by the per-face color.

get_extents()
Get the extents of the geometry

Returns

[[minimum x, minimum y, minimum z], [maximum x, maximum y, maximum z]]

class fresnel.geometry.**Cylinder**(*scene*, *points=None*, *radius=None*, *color=None*, *N=None*, *material=<fresnel.material.Material object>*, *outline_material=<fresnel.material.Material object>*, *outline_width=0.0*)

Cylinder geometry.

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

Parameters

- **scene** (*fresnel.Scene*) – Add the geometry to this scene
- **points** – cylinder start and end points, *optional*. **Type:** anything convertible by numpy to a Nx2x3 array of floats.
- **radius** – Radius of each cylinder, *optional*. **Type:** anything convertible by numpy to a N length array of floats.
- **color** – (r,g,b) color of each particle, *optional*. **Type:** anything convertible by numpy to a Nx2x3 array of floats.
- **N** (*int*) – Number of cylinders in the geometry. If `None`, determine `N` from `position`.

Note: The constructor arguments `points`, `radius`, and `color` are optional, and just short-hand for assigning the properties after construction.

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

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

points
fresnel.util.array – Read or modify the start and end points of the cylinders.

radius
fresnel.util.array – Read or modify the radii of the cylinders.

color
fresnel.util.array – Read or modify the colors of the start and end points of the cylinders.

get_extents()
Get the extents of the geometry

Returns

[[minimum x, minimum y, minimum z], [maximum x, maximum y, maximum z]]

class fresnel.geometry.**Geometry**
Base class for all geometry.

Geometry provides operations common to all geometry classes.

material
fresnel.material.Material – Read, set, or modify the geometry's material.

outline_material
fresnel.material.Material – Read, set, or modify the geometry's outline material.

outline_width
`float` – The geometry's outline width, in distance units in the scene's coordinate system.

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

disable()
Disable the geometry.

When disabled, the geometry will not be present in the scene. No rays will intersect it.

enable()
Enable the geometry.

When enabled, the geometry will be present when rendering the scene.

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()* if you want a reversible operation.

class fresnel.geometry.**Prism**(*scene*, *vertices*, *position=None*, *angle=None*, *height=None*, *color=None*, *N=None*, *material=<fresnel.material.Material object>*, *outline_material=<fresnel.material.Material object>*, *outline_width=0.0*)

Prism geometry.

Define a set of right convex prism primitives. The bottom polygon face is always in the xy plane. Each prism may have a different height and rotation angle.

Parameters

- **scene** (*fresnel.Scene*) – Add the geometry to this scene
- **vertices** – The vertices of the polygon in a counter clockwise winding direction. **Type:** anything convertible by numpy to a Nx2 array of floats.
- **position** – Positions of the prisms, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.
- **height** – Height of each prism in the z direction, *optional*. **Type:** anything convertible by numpy to a N length array of floats.
- **angle** – Rotation angle of each prism (in radians), *optional*. **Type:** anything convertible by numpy to a N length array of floats.
- **color** – (r,g,b) color of each particle, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.
- **N** (*int*) – Number of spheres in the geometry. If `None`, determine `N` from `position`.

Note: The constructor arguments `position`, `height`, `angle`, and `color` are optional, and just short-hand for assigning the attribute after construction.

Colors are in the linearized sRGB color space. Use `fresnel.color.linear()` to convert standard sRGB colors into this space.

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

position
fresnel.util.array – Read or modify the positions of the prisms.

height
fresnel.util.array – Read or modify the heights of the prisms.

angle
fresnel.util.array – Read or modify the angles of the prisms.

color
fresnel.util.array – Read or modify the color of the prisms.

class fresnel.geometry.**Sphere**(*scene*, *position=None*, *radius=None*, *color=None*, *N=None*, *material=<fresnel.material.Material object>*, *outline_material=<fresnel.material.Material object>*, *outline_width=0.0*)

Sphere geometry.

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

Parameters

- **scene** (*fresnel.Scene*) – Add the geometry to this scene
- **position** – Positions of the spheres, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.
- **radius** – Radius of each sphere, *optional*. **Type:** anything convertible by numpy to a N length array of floats.
- **color** – (r,g,b) color of each particle, *optional*. **Type:** anything convertible by numpy to a Nx3 array of floats.
- **N** (*int*) – Number of spheres in the geometry. If `None`, determine `N` from `position`.

Note: The constructor arguments `position`, `radius`, and `color` are optional, and just short-hand for assigning the properties after construction.

Colors are in the linearized sRGB color space. Use `fresnel.color.linear()` to convert standard sRGB colors into this space.

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

position
fresnel.util.array – Read or modify the positions of the spheres.

radius
fresnel.util.array – Read or modify the radii of the spheres.

color
fresnel.util.array – Read or modify the color of the spheres.

get_extents()
Get the extents of the geometry

Returns

[[minimum x, minimum y, minimum z], [maximum x, maximum y, maximum z]]

1.16.4 fresnel.light

Overview

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

Details

Lights.

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

Define a single light

Parameters

- **`direction`** – A 3-tuple that defines the direction the light points in camera space.
- **`color`** – A 3-tuple that defines the color and intensity of the light as a linear sRGB value (see *`fresnel.color.linear()`*)
- **`theta`** (*`float`*) – Half angle of the cone that defines the area of the light (in radians)

The direction vector may have any non-zero length, but only the direction the vector points matters.

The color also sets the light intensity. A (0.5, 0.5, 0.5) light is twice as bright as (0.25, 0.25, 0.25).

`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 A list of lights.

`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 A list of lights.

`fresnel.light.`**`lightbox`**()

Create a light box lighting setup.

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

Returns A list of lights.

`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 A list of lights.

`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 A list of lights.

`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 A list of lights.

1.16.5 fresnel.material

Overview

fresnel.material.Material	Define material properties.

Details

Materials describe the way light interacts with surfaces.

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

Define material properties.

Parameters

- **solid** (*float*) – Set to 1 to pass through a solid color, regardless of the light and view angle.
- **color** (*tuple*) – The linear RGB color of the material as a 3-tuple, list or other iterable.
- **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], though 0.1 is a realistic minimum.
- **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.

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

1.16.6 fresnel.tracer

Overview

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

Details

Ray tracers.

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

Path tracer.

Parameters

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

The path tracer applies advanced lighting effects, including soft shadows, reflections, etc.... 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 and 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*)

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

Returns A reference to the current output buffer as a `fresnel.util.image_array`.

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*, *aa_level=0*)

Preview ray tracer.

Parameters

- **device** (*Device*) – Device to use for rendering.
- **w** (*int*) – Output image width.
- **h** (*int*) – Output image height.
- **aa_level** (*int*) – Amount of anti-aliasing to perform

aa_level

int – Amount of anti-aliasing to perform

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, and other lighting effects.

TODO: show examples

Anti-aliasing

Set `aa_level` to control the amount of anti-aliasing performed. The default value of 0 performs no anti-aliasing to enable the fastest possible preview renders. A value of 1 samples 2x2 subpixels, a value of 2 samples 4x4 subpixels, a value of 3 samples 8x8 subpixels, etc … Samples are jittered with random numbers. Different `seed` values will result in different output images.

TODO: show examples

Tip: Use `aa_level` = 3 when using the `Preview` tracer to render production quality output.

---

class `fresnel.tracer.`**`Tracer`**

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 current data stored in the buffer is overwritten with the new image.

Note: You cannot instantiate a Tracer directly. Use one of the sub classes.

`output`

`fresnel.util.image_array` – Reference to the current output buffer (modified by `render()`)

`linear_output`

`fresnel.util.array` – Reference to the current output buffer in linear color space (modified by `render()`)

`seed`

int – Random number seed.

`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).

`render`(*scene*)

Render a scene.

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

Returns A reference to the current output buffer as a `fresnel.util.image_array`.

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 any existing image in the output buffer.

1.16.7 fresnel.util

Overview

fresnel.util.array	Map internal fresnel buffers as numpy arrays.
fresnel.util.image_array	Map internal fresnel image buffers as numpy arrays.

Details

Utility classes and methods.

class fresnel.util.**array**(*buf*, *geom*)
Map internal fresnel buffers as numpy arrays.

fresnel.util.array provides a python interface to access internal data of memory buffers stored and managed by fresnel. These buffers may exist on the CPU or GPU depending on the device configuration, so *fresnel.util.array* only allows certain operations: read/write of array data, and read-only querying of array properties.

You can access a *fresnel.util.array* as if it were a numpy array (with limited operations).

Write to an array with `array[slice] = v` where *v* is a numpy array or anything that numpy can convert to an array. When *v* is a contiguous numpy array of the appropriate data type, the data is copied directly from *v* into the internal buffer.

Read from an array with `v = array[slice]`. This returns a **copy** of the data as a numpy array because the array references internal data structures in fresnel that may exist on the GPU.

shape
tuple – Dimensions of the array.

dtype
Numpy data type

class fresnel.util.**image_array**(*buf*, *geom*)
Map internal fresnel image buffers as numpy arrays.

Inherits from *array* and provides all of its functionality, plus some additional convenience methods specific to working with images. Images are represented as WxHx4 numpy arrays of unsigned chars in RGBA format.

Specifically, when a *image_array* is the result of an image in a Jupyter notebook cell, Jupyter will display the image.

1.17 License

```
Fresnel Open Source Software License Copyright (c) 2016-2018 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.
```

1.18 Credits

1.18.1 Fresnel Developers

Joshua A. Anderson, University of Michigan - **Lead developer**

Vyas Ramasubramani, University of Michigan

- Sphere geometry (GPU)
- Review and discussions on API and design.

Bryan Vansaders, University of Michigan

- Sphere geometry (CPU)
- Review and discussions on API and design.

1.18.2 Libraries

Fresnel links to the following libraries:

Python

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

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.

   "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
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

```
      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
   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.
```

(continues on next page)

(continued from previous page)

```
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.
```

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.
```

(continues on next page)

(continued from previous page)

OptiX SDK

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

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
  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.
```

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,
```

(continues on next page)

(continued from previous page)

```
   "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,
   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,
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

1.19 Index

- genindex
- modindex

Python Module Index

f

fresnel, 84
fresnel.camera, 86
fresnel.color, 87
fresnel.geometry, 88
fresnel.light, 92
fresnel.material, 94
fresnel.tracer, 94
fresnel.util, 97

Index

Symbols

__version__ (in module fresnel), 84

A

aa_level (fresnel.tracer.Preview attribute), 95
angle (fresnel.geometry.Prism attribute), 91
array (class in fresnel.util), 97
available_gpus (fresnel.Device attribute), 84
available_modes (fresnel.Device attribute), 84

B

background_alpha (fresnel.Scene attribute), 85
background_color (fresnel.Scene attribute), 85
butterfly() (in module fresnel.light), 93

C

Camera (class in fresnel.camera), 86
camera (fresnel.Scene attribute), 85
cloudy() (in module fresnel.light), 93
color (fresnel.geometry.ConvexPolyhedron attribute), 89
color (fresnel.geometry.Cylinder attribute), 90
color (fresnel.geometry.Prism attribute), 91
color (fresnel.geometry.Sphere attribute), 92
color_by_face (fresnel.geometry.ConvexPolyhedron attribute), 89
ConvexPolyhedron (class in fresnel.geometry), 88
Cylinder (class in fresnel.geometry), 89

D

Device (class in fresnel), 84
device (fresnel.Scene attribute), 85
disable() (fresnel.geometry.Geometry method), 90
disable_highlight_warning() (fresnel.tracer.Tracer method), 96
dtype (fresnel.util.array attribute), 97

E

enable() (fresnel.geometry.Geometry method), 90
enable_highlight_warning() (fresnel.tracer.Tracer method), 96

F

fit() (in module fresnel.camera), 86
fresnel (module), 84
fresnel.camera (module), 86
fresnel.color (module), 87
fresnel.geometry (module), 88
fresnel.light (module), 92
fresnel.material (module), 94
fresnel.tracer (module), 94
fresnel.util (module), 97

G

Geometry (class in fresnel.geometry), 90
get_extents() (fresnel.geometry.ConvexPolyhedron method), 89
get_extents() (fresnel.geometry.Cylinder method), 90
get_extents() (fresnel.geometry.Sphere method), 92
get_extents() (fresnel.Scene method), 85

H

height (fresnel.camera.Camera attribute), 86
height (fresnel.geometry.Prism attribute), 91
histogram() (fresnel.tracer.Tracer method), 96

I

image_array (class in fresnel.util), 97

L

Light (class in fresnel.light), 92
lightbox() (in module fresnel.light), 93
lights (fresnel.Scene attribute), 85
linear() (in module fresnel.color), 87
linear_output (fresnel.tracer.Tracer attribute), 96
look_at (fresnel.camera.Camera attribute), 86
loop() (in module fresnel.light), 93

M

Material (class in fresnel.material), 94
material (fresnel.geometry.Geometry attribute), 90
mode (fresnel.Device attribute), 84

O

orientation (fresnel.geometry.ConvexPolyhedron attribute), 89
orthographic() (in module fresnel.camera), 87
outline_material (fresnel.geometry.Geometry attribute), 90
outline_width (fresnel.geometry.Geometry attribute), 90
output (fresnel.tracer.Tracer attribute), 96

P

Path (class in fresnel.tracer), 94
pathtrace() (in module fresnel), 85
points (fresnel.geometry.Cylinder attribute), 90
position (fresnel.camera.Camera attribute), 86
position (fresnel.geometry.ConvexPolyhedron attribute), 89
position (fresnel.geometry.Prism attribute), 91
position (fresnel.geometry.Sphere attribute), 92
Preview (class in fresnel.tracer), 95
preview() (in module fresnel), 86
Prism (class in fresnel.geometry), 90

R

radius (fresnel.geometry.Cylinder attribute), 90
radius (fresnel.geometry.Sphere attribute), 92
rembrandt() (in module fresnel.light), 93
remove() (fresnel.geometry.Geometry method), 90
render() (fresnel.tracer.Tracer method), 96
reset() (fresnel.tracer.Path method), 95
resize() (fresnel.tracer.Tracer method), 97
ring() (in module fresnel.light), 93

S

sample() (fresnel.tracer.Path method), 95
Scene (class in fresnel), 84
seed (fresnel.tracer.Tracer attribute), 96
shape (fresnel.util.array attribute), 97
Sphere (class in fresnel.geometry), 91

T

Tracer (class in fresnel.tracer), 96

U

up (fresnel.camera.Camera attribute), 86