

---

# **raytraverse Documentation**

***Release 0.1.3***

**Stephen Wasilewski**

**Sep 24, 2020**



CONTENTS

<b>1</b>	<b>raytraverse</b>	<b>1</b>
1.1	Installation . . . . .	1
1.2	Usage . . . . .	1
1.3	Git Stuff . . . . .	2
1.4	Licence . . . . .	2
1.5	Credits . . . . .	2
	<b>Python Module Index</b>	<b>39</b>
	<b>Index</b>	<b>41</b>



## RAYTRAVERSE

raytraverse is a complete workflow for climate based daylight modelling, simulation, and evaluation of architectural spaces. Built around a variance based adaptive sampling strategy, raytraverse can fully explore the daylight conditions throughout a space with efficient use of processing power and storage space.

- Free software: Mozilla Public License 2.0 (MPL 2.0)
- Documentation: <https://raytraverse.readthedocs.io>.

### 1.1 Installation

The easiest way to install raytraverse is with pip:

```
pip install raytraverse
```

or if you have cloned this repository:

```
cd path/to/this/file  
pip install .
```

note that on first run one of the required modules will download some auxiliary data which could take a minute, after that first run start-up is much faster.

### 1.2 Usage

raytraverse includes a complete command line interface with all commands nested under the *raytraverse* parent command enter:

```
raytraverse --help
```

raytraverse also exposes an object oriented API written primarily in python. calls to Radiance are made through Renderer objects that wrap the radiance c source code in c++ classes, which are made available in python with pybind11. see the src/ directory for more.

For complete documentation of the API and the command line interface either use the Documentation link included above or:

```
pip install -r requirements_dev.txt  
make docs
```

to generate local documentation.

## 1.3 Git Stuff

this project is hosted in too places, a private repo (master branch) at:

<https://gitlab.enterpriselab.ch/lightfields/raytraverse>

and a public repo (release branch) at:

<https://github.com/stephanwaz/raytraverse>

the repo also depends on two submodules, to initialize run the following:

```
git clone https://github.com/stephanwaz/raytraverse
cd raytraverse
git submodule init
git submodule update --remote
git -C src/Radiance config core.sparseCheckout true
cp src/sparse-checkout .git/modules/src/Radiance/info/
git submodule update --remote --force src/Radiance
```

after a “git pull” make sure you also run:

```
git submodule update
```

to track with the latest commit used by raytraverse.

## 1.4 Licence

Copyright (c) 2020 Stephen Wasilewski

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>.

## 1.5 Credits

This package was created with [Cookiecutter](#) and the [audreyr/cookiecutter-pypackage](#) project template.

### 1.5.1 raytraverse.scene

#### Scene

```
class raytraverse.scene.Scene(outdir, scene=None, area=None, reload=True, over-
                             write=False, ptres=1.0, ptro=0.0, pttol=1.0, viewdir=0, 1, 0,
                             viewangle=360, skyres=10.0, maxspec=0.3, **kwargs)
```

Bases: object

container for scene description

#### Parameters

- **outdir** (*str*) – path to store scene info and output files
- **scene** (*str*, optional (required if not reload)) – space separated list of radiance scene files (no sky) or octree
- **area** (*str*, optional (required if not reload)) – radiance scene file containing planar geometry of analysis area or a list of points (line per point, space seperated, first 3 columns x, y, z

- **reload** (*bool, optional*) – if True attempts to load existing scene files in new instance overrides ‘overwrite’
- **overwrite** (*bool, optional*) – if True and outdir exists, will overwrite, else raises a FileExistsError
- **ptres** (*float, optional*) – final spatial resolution in scene geometry units
- **ptro** (*float, optional*) – angle in degrees counter-clockwise to point grid
- **pttol** (*float, optional*) – tolerance for point search when using point list for area
- **viewdir** (*(float, float, float), optional*) – vector (x,y,z) view direction (orients UV space)
- **viewangle** (*float, optional*) – should be 1-180 or 360
- **skyres** (*float, optional*) – approximate square patch size (sets sun resolution too)
- **maxspec** (*float, optional*) – maximum specular transmission in scene (used to clip pdf for sun sampling)

**outdir = None**

path to store scene info and output files

**Type** str

**maxspec = None**

maximum specular transmission in scene

**Type** float

**reload = None**

try to reload scene files

**Type** bool

**view = None**

view translation class

**Type** raytraverse.viewmapper.ViewMapper

**property skyres**

**property scene**

render scene files (octree)

**Getter** Returns this samplers’s scene file path

**Setter** Sets this samplers’s scene file path and creates run files

**Type** str

**pts** ()

## SunSetter

**class** raytraverse.scene.SunSetter (*scene, srct=0.01, skyro=0.0, reload=True, sunres=10.0, \*\*kwargs*)

Bases: object

select suns to sample based on sky pdf and scene.

### Parameters

- **scene** (*raytraverse.scene.Scene*) – scene class containing geometry, location and analysis plane

- **srct** (*float*, *optional*) – threshold of sky contribution for determining appropriate srcn
- **skyro** (*float*, *optional*) – sky rotation (in degrees, ccw)
- **reload** (*bool*) – if True reloads existing sun positions, else always generates new

**srct** = None

threshold of sky contribution for determining appropriate srcn

**Type** float

**skyro** = None

ccw rotation (in degrees) for sky

**Type** float

**scene** = None

raytraverse.scene.Scene

**property sunres**

**property sun\_kd**

sun kdtree for directional queries

**property suns**

holds sun positions

**Getter** Returns the sun source array

**Setter** Set the sun source array and write to files

**Type** np.array

**choose\_suns** ()

**load\_sky\_facs** ()

**direct\_view** ()

**write\_sun** (*i*)

**proxy\_src** (*tsuns*, *tol=10.0*)

check if sun directions have matching source in SunSetter

**Parameters**

- **tsuns** (*np.array*) – (N, 3) array containing sun source vectors to check
- **tol** (*float*) – tolerance (in degrees)

**Returns**

- *np.array* – (N,) boolean array if sun has a match
- *np.array* – (N,) index to proxy src

**\_write\_suns** (*sunfile*)

write suns to file

**Parameters** **sunfile** –



## SunSetterLoc

**class** raytraverse.scene.SunSetterLoc (*scene, loc, skyro=0.0, \*\*kwargs*)

Bases: raytraverse.scene.sunsetter.SunSetter

select suns to sample based on sky pdf and scene.

### Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **loc** (tuple) – lat, lon, tz (in degrees, west is positive)
- **srct** (float, optional) – threshold of sky contribution for determining appropriate srcn
- **skyro** (float, optional) – sky rotation (in degrees, ccw)
- **reload** (bool) – if True reloads existing sun positions, else always generates new

**sky** = None

raytraverse.scene.SkyInfo

**choose\_suns** ()

## SunSetterPositions

**class** raytraverse.scene.SunSetterPositions (*scene, wea, skyro=0.0, \*\*kwargs*)

Bases: raytraverse.scene.sunsetter.SunSetter

select suns to sample based on sky pdf and scene.

### Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **wea** (str, np.array, optional) – path to sun position file or wea file, or array of sun positions
- **srct** (float, optional) – threshold of sky contribution for determining appropriate srcn
- **skyro** (float, optional) – sky rotation (in degrees, ccw)
- **reload** (bool) – if True reloads existing sun positions, else always generates new

**scene** = None

raytraverse.scene.Scene

**skyro** = None

ccw rotation (in degrees) for sky

**Type** float

**property candidates**

raytraverse.scene.SkyInfo

**choose\_suns** ()

## SkyInfo

**class** raytraverse.scene.SkyInfo (*loc, skyro=0.0*)

Bases: object

sky location data object

### Parameters

- **loc** (*tuple*) – lat, lon, tz (in degrees, west is positive)
- **skyro** (*float*) – sky rotation (in degrees, ccw)

**skyro = None**

ccw rotation (in degrees) for sky

**Type** float

**property solarbounds**

read only extent of solar bounds for given location set via loc

**Getter** Returns solar bounds

**Type** (np.array, np.array)

**property loc**

scene location

**Getter** Returns location

**Setter** Sets location and self.solarbounds

**Type** (float, float, int)

**in\_solarbounds** (*uv, size=0.0*)

for checking if src direction is in solar transit

### Parameters

- **uv** (*np.array*) – source directions
- **size** (*float*) – offset around UV to test

**Returns result** – Truth of ray.src within solar transit

**Return type** np.array

## 1.5.2 raytraverse.mapper

### SpaceMapper

**class** raytraverse.mapper.SpaceMapper (*dfile, ptres=1.0, rotation=0.0, tolerance=1.0*)

Bases: object

translate between world coordinates and normalized UV space

**rotation = None**

ccw rotation (in degrees) for point grid on plane

**Type** float

**tolerance = None**

tolerance for point search when using point list for area

**Type** float

**ptres = None**

point resolution for area

**Type** float

**property pt\_kd**  
point kdtree for spatial queries built at first use

**property sf**  
bbox scale factor

**property ptshape**  
shape of point grid

**property npts**  
number of points

**property bbox**  
boundary frame for translating between coordinates [[xmin ymin zmin] [xmax ymax zmax]]

**Type** np.array

**\_ro\_pts** (*points*, *rdir=-1*)  
rotate points

**Parameters**

- **points** (*np.ndarray*) – world coordinate points of shape (N, 3)
- **rdir** (*-1 or 1*) –  
**rotation direction:** -1 to rotate from uv space 1 to rotate to uvspace

**uv2pt** (*uv*)  
convert UV → world

**Parameters** **uv** (*np.array*) – normalized UV coordinates of shape (N, 2)

**Returns** **pt** – world xyz coordinates of shape (N, 3)

**Return type** np.array

**pt2uv** (*xyz*)  
convert world → UV

**Parameters** **xyz** (*np.array*) – world xyz coordinates, shape (N, 3)

**Returns** **uv** – normalized UV coordinates of shape (N, 2)

**Return type** np.array

**idx2pt** (*idx*)

**pts** ()

**in\_area** (*xyz*)  
check if point is in boundary path

**Parameters** **xyz** (*np.array*) – uv coordinates, shape (N, 3)

**Returns** **mask** – boolean array, shape (N,)

**Return type** np.array

**\_rad\_scene\_to\_bbox** (*plane*)

## SpaceMapperPt

**class** raytraverse.mapper.SpaceMapperPt (*dfile, ptres=1.0, rotation=0.0, tolerance=1.0*)

Bases: raytraverse.mapper.spacemapper.SpaceMapper

translate between world coordinates and normalized UV space

**property sf**

bbox scale factor

**property ptshape**

shape of point grid

**property bbox**

bounding box

**Type** np.array of shape (3,2)

**uv2pt** (*uv*)

convert UV → world

**Parameters** **uv** (*np.array*) – normalized UV coordinates of shape (N, 2)

**Returns** **pt** – world xyz coordinates of shape (N, 3)

**Return type** np.array

**pt2uv** (*xyz*)

convert world → UV

**Parameters** **xyz** (*np.array*) – world xyz coordinates, shape (N, 3)

**Returns** **uv** – normalized UV coordinates of shape (N, 2)

**Return type** np.array

**idx2pt** (*idx*)

**pts** ()

**in\_area** (*xyz*)

check if point is in boundary path

**Parameters** **xyz** (*np.array*) – uv coordinates, shape (N, 3)

**Returns** **mask** – boolean array, shape (N,)

**Return type** np.array

## ViewMapper

**class** raytraverse.mapper.ViewMapper (*dxyz=0.0, 1.0, 0.0, viewangle=360.0, name='view',  
mtxs=None, imtxs=None*)

Bases: object

translate between world and normalized UV space based on direction and view angle

**Parameters**

- **dxyz** (*tuple, optional*) – central view direction
- **viewangle** (*float, optional*) – if < 180, the horizontal and vertical view angle, if greater, view becomes 360, 180

**property viewangle**

view angle

**property ymtx**

yaw rotation matrix (to standard z-direction y-up)

**property pmtx**  
pitch rotation matrix (to standard z-direction y-up)

**property bbox**  
bounding box of view  
**Type** np.array of shape (2,2)

**property sf**  
bbox scale factor

**property ivm**  
viewmapper for opposite view direction (in case of 360 degree view)

**property dxyz**  
(float, float, float) central view direction

**view2world** (xyz, i=0)

**world2view** (xyz, i=0)

**xyz2uv** (xyz, i=0)

**uv2xyz** (uv, i=0)

**xyz2xy** (xyz, i=0)

**pixelrays** (res, i=0)

**ray2pixel** (xyz, res, i=0)

**pixel2ray** (pxy, res, i=0)

**pixel2omega** (pxy, res)

**ctheta** (vec, i=0)

**radians** (vec, i=0)

**degrees** (vec, i=0)

**in\_view** (vec, i=0, indices=True)

## SunMapper

**class** raytraverse.mapper.**SunMapper** (suns)  
Bases: raytraverse.mapper.viewmapper.ViewMapper  
translate between view and normalized UV space  
**Parameters** **suns** (np.array) – dx,dy,dz sun positions

## 1.5.3 raytraverse.renderer

### Renderer

**class** raytraverse.renderer.**Renderer** (rayargs=None, scene=None, nproc=None, \*\*kwargs)  
Bases: object  
virtual renderer class  
**initialized** = False  
**instance** = None  
**\_pyinstance** = None  
**Engine** = None

```
name = None
header = ''
arg_prefix = ''
classmethod _set_args (args, iot, nproc)
classmethod initialize (args, scene, nproc=None, **kwargs)
classmethod call (rayfile, store=True, outf=None)
classmethod reset ()
classmethod reset_instance ()
```

## RadianceRenderer

```
class raytraverse.renderer.RadianceRenderer (rayargs=None, scene=None,
                                             nproc=None, **kwargs)
    Bases: raytraverse.renderer.renderer.Renderer
    Virtual class for wrapping c++ Radiance renderer executable classes
    returnbytes = False
    classmethod update_param (args, nproc=None, iot='ff')
    classmethod initialize (args, scene, nproc=None, iot='ff')
    classmethod call (rayfile, store=True, outf=None)
    classmethod reset ()
    classmethod reset_instance ()
```

## Rtrace

```
class raytraverse.renderer.Rtrace (rayargs=None, scene=None, nproc=None, **kwargs)
    Bases: raytraverse.renderer.radiancerenderer.RadianceRenderer
    singleton wrapper for c++ crenderer.cRtrace singleton class
    Engine
        alias of raytraverse.renderer.sprenderer.SPRtrace
    name = 'rtrace'
    classmethod update_ospec (vs, of='a')
    classmethod new_amb (af)
    classmethod load_source (srcname, freesrc=- 1)
```

## Rcontrib

```
class raytraverse.renderer.Rcontrib (rayargs=None, scene=None, nproc=None,
                                     **kwargs)
    Bases: raytraverse.renderer.radiancerenderer.RadianceRenderer
    singleton wrapper for c++ crenderer.cRcontrib singleton class
    Engine
        alias of raytraverse.renderer.sprenderer.SPRcontrib
    name = 'rcontrib'
    arg_prefix = '-o !cat'
```

## SPRenderer

```

class raytraverse.renderer.SPRenderer (rayargs=None,    scene=None,    nproc=None,
                                     **kwargs)
    Bases: raytraverse.renderer.renderer.Renderer
    Subprocess renderer class

    cleanup = 'rcalc -if3 -of -e $1=.265074126*$1+.670114631*$2+.064811243*$3'
    scene = ''

    classmethod get_instance ()
    classmethod initialize (args, scene=None, nproc=None, iot='ff')
    classmethod load_scene (scene)
    classmethod call (rayfile, store=True, outf=None, vecs2stdin=True)

```

## SPRtrace

```

class raytraverse.renderer.SPRtrace (rayargs=None,    scene=None,    nproc=None,
                                     **kwargs)
    Bases: raytraverse.renderer.sprenderer.SPRenderer
    Engine = 'rtrace'
    name = 'rtrace'

```

## SPRcontrib

```

class raytraverse.renderer.SPRcontrib (rayargs=None,    scene=None,    nproc=None,
                                       **kwargs)
    Bases: raytraverse.renderer.sprenderer.SPRenderer
    Engine = 'rcontrib'
    name = 'rcontrib'

```

## 1.5.4 raytraverse.sampler

### Sampler

```

class raytraverse.sampler.Sampler (scene, fdres=9, srcn=1, accuracy=1.0, idres=4,
                                   stype='generic', srcdef=None, plotp=False, bands=1,
                                   engine_args="", nproc=None, **kwargs)

```

Bases: object

base sampling class

To implement a proper-subclass an engine attribute must be set to a renderer instance prior to calling `Sampler.__init__`. Also, the method `sample` must be overridden to properly set up arguments for the `renderer.call`

#### Parameters

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry, location and analysis plane
- **fdres** (`int`, *optional*) – final directional resolution given as  $\log_2(\text{res})$
- **srcn** (`int`, *optional*) – number of sources return per vector by run
- **accuracy** (`float`, *optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)

- **idres** (*int*, *optional*) – initial direction resolution (as  $\log_2(\text{res})$ )
- **stype** (*str*, *optional*) – sampler type (prefixes output files)
- **srcdef** (*str*, *optional*) – path or string with source definition to add to scene
- **plotp** (*bool*, *optional*) – show probability distribution plots at each level (first point only)
- **bands** (*int*, *optional*) – number of spectral bands returned by the engine
- **engine\_args** (*str*, *optional*) – command line arguments used to initialize engine
- **nproc** (*int*, *optional*) – number of processors to give to the engine, if None, uses `os.cpu_count()`

**engine** = None

**bands** = None

number of spectral bands / channels returned by renderer based on given renderopts (user ensures these agree).

**Type** int

**samplemap** = None

mapper to use for sampling

**Type** func

**srcn** = None

number of sources return per vector by run

**Type** int

**idres** = None

initial direction resolution (as  $\log_2(\text{res})$ )

**Type** int

**weights** = None

holds weights for self.draw

**Type** np.array

**stype** = None

sampler type

**Type** str

**property compiledscene**

**property idx**

index of next level to sample

**Type** int

**property levels**

sampling scheme

**Getter** Returns the sampling scheme

**Setter** Set the sampling scheme from (ptres, fdres, skres)

**Type** np.array

**property scene**

scene information

**Getter** Returns this sampler's scene

**Setter** Set this sampler's scene and create octree with source desc



Type `raytraverse.scene.Scene`

**sample** (*vecf*)

generic sample function

**\_uv2xyz** (*uv, si*)

including to allow overriding mapping behavior of daughter classes

**sample\_idx** (*pdraws*)

generate samples vectors from flat draw indices

**Parameters** **pdraws** (*np.array*) – flat index positions of samples to generate

**Returns**

- **si** (*np.array*) – index array of draws matching `samps.shape`
- **vecs** (*np.array*) – sample vectors

**dump\_vecs** (*vecs, si=None*)

save vectors to file

**Parameters**

- **vecs** (*np.array*) – ray directions to write
- **si** (*np.array, optional*) – sample indices

**draw** ()

draw samples based on detail calculated from weights detail is calculated across direction only as it is the most precise dimension

**Returns** **pdraws** – index array of flattened samples chosen to sample at next level

**Return type** `np.array`

**update\_pdf** (*si, lum*)

update self.weights (which holds values used to calculate pdf)

**Parameters**

- **si** (*np.array*) – multidimensional indices to update
- **lum** – values to update with

**run\_callback** ()

**get\_scheme** ()

**run** ()

execute sampler

## SCBinSampler

**class** `raytraverse.sampler.SCBinSampler` (*scene, accuracy=1, rcopts='-ab 7 -ad 60000 -as 30000 -lw 1e-7', \*\*kwargs*)

Bases: `raytraverse.sampler.sampler.Sampler`

sample contributions from the sky hemisphere according to a square grid transformed by shirley-chiu mapping using `rcontrib`.

**Parameters**

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry, location and analysis plane
- **srcn** (*int, optional*) – side of square sky resolution

**sample** (*vecf*)

call rendering engine to sample sky contribution

**Parameters** **vecf** (*str*) – path of file name with sample vectors shape (N, 6) vectors in binary float format

**Returns** **lum** – array of shape (N,) to update weights

**Return type** np.array

## SunSampler

**class** raytraverse.sampler.SunSampler (*scene, suns, plotp=False, \*\*kwargs*)

Bases: object

factory class for SingleSunSamplers.

### Parameters

- **scene** (*raytraverse.scene.Scene*) – scene class containing geometry, location and analysis plane
- **suns** (*raytraverse.sunsetter.SunSetter*) – sun class containing sun locations.

**viewsampler** = None

raytraverse.sampler.SunViewSampler

**sampleargs** = None

sampling arguments for SingleSunSampler

**Type** dict

**reflsampler** = None

raytraverse.sampler.SingleSunSampler

**run** (*view=True, reflection=True*)

## SingleSunSampler

**class** raytraverse.sampler.SingleSunSampler (*scene, suns, idx, speclevel=9, fdres=10, accuracy=1, rcopts='-ab 6 -ad 3000 -as 1500 -st 0 -ss 16 -aa .1', keepamb=False, ambcache=True, \*\*kwargs*)

Bases: raytraverse.sampler.sampler.Sampler

sample contributions from direct suns.

### Parameters

- **scene** (*raytraverse.scene.Scene*) – scene class containing geometry, location and analysis plane
- **suns** (*raytraverse.sunsetter.SunSetter*) – sun class containing sun locations.
- **idx** (*int*) – sun index to sample
- **speclevel** (*int, optional*) – at this sampling level, pdf is made from brightness of sky sampling rather than progressive variance to look for fine scale specular highlights, this should be atleast 1 level from the end and the resolution of this level should be smaller than the size of the source
- **keepamb** (*bool, optional*) – whether to keep ambient files after run, if kept, a successive call will load these ambient files, so care must be taken to not change any parameters
- **ambcache** (*bool, optional*) – whether the rcopts indicate that the calculation will use ambient caching (and thus should write an -af file argument to the engine)

**EngineClass**alias of `raytraverse.renderer.rtrace.Rtrace`**slimit = None**

controls sampling limit in case of limited contribution

**Type** float**specidx = None**

index of level at which brightness sampling occurs

**Type** int**sunpos = None**

sun position x,y,z

**Type** np.array**pdf\_from\_sky** (*skyfield*, *interp=12*, *rebuild=False*, *zero=True*, *filterpts=True*)**sample** (*vecf*)

call rendering engine to sample sky contribution

**Parameters** **vecf** (*str*) – path of file name with sample vectors shape (N, 6) vectors in binary float format**Returns** **lum** – array of shape (N,) to update weights**Return type** np.array**draw** ()

draw samples based on detail calculated from weights detail is calculated across direction only as it is the most precise dimension

**Returns** **pdraws** – index array of flattened samples chosen to sample at next level**Return type** np.array**SunViewSampler****class** `raytraverse.sampler.SunViewSampler` (*scene*, *suns*, *\*\*kwargs*)Bases: `raytraverse.sampler.sampler.Sampler`

sample view rays to direct suns.

here idres and fdres are sampled on a per sun basis for a view centered on each sun direction with a view angle of .533 degrees (hardcoded in sunmapper class).

**Parameters**

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry, location and analysis plane
- **suns** (`raytraverse.sunsetter.SunSetter`) – sun class containing sun locations.
- **loadsrc** (*bool*) – include suns.rad in base scene initialization. if False, `self.engine.load_source` must be invoked before call.

**EngineClass**alias of `raytraverse.renderer.rtrace.Rtrace`**property levels**

sampling scheme

**Getter** Returns the sampling scheme**Setter** Set the sampling scheme from (ptres, fdres, skres)**Type** np.array

**check\_viz** ()

**sample** (*vecf*)

call rendering engine to sample direct view rays

**Parameters** **vecf** (*str*) – path of file name with sample vectors shape (N, 6) vectors in binary float format

**Returns** **lum** – array of shape (N,) to update weights

**Return type** np.array

**\_uv2xyz** (*uv, si*)

including to allow overriding mapping behavior of daughter classes

**draw** ()

draw first level based on sky visibility

**run\_callback** ()

post sampling, right full resolution (including interpolated values) non zero rays to result file.

## 1.5.5 raytraverse.lightfield

### LightField

**class** raytraverse.lightfield.**LightField** (*scene, rebuild=False, prefix='sky', srcn=1, rmraw=False*)

Bases: object

container for accessing sampled data

#### Parameters

- **scene** (*raytraverse.scene.Scene*) – scene class containing geometry, location and analysis plane
- **rebuild** (*bool, optional*) – build kd-tree even if one exists
- **prefix** (*str, optional*) – prefix of data files to map

**rebuild** = None

force rebuild kd-tree

**Type** bool

**prefix** = None

prefix of data files from sampler (stype)

**Type** str

**raw\_files** ()

**property** **vec**

direction vector (3,)

**property** **lum**

luminance (srcn,)

**property** **omega**

solid angle (1,)

**outfile** (*idx*)

**items** ()

## LightFieldKD

```
class raytraverse.lightfield.LightFieldKD (scene, rebuild=False, prefix='sky', srcn=1,
                                           rmraw=False)
    Bases: raytraverse.lightfield.lightfield.LightField
    light field with KDtree structures for spatial query

    static mk_vector_ball (v)

    property d_kd
        list of direction kdtrees

        Getter Returns kd tree structure
        Type list of scipy.spatial.cKDTree

    property scene
        scene information

        Getter Returns this integrator's scene
        Setter Set this integrator's scene
        Type raytraverse.scene.Scene

    raw_files ()
        get list of files used to build field

    _to_mem_map (ar, offset=0)

    _get_vl (npts, pref="", ltype=<class 'raytraverse.lightfield.memarraydict.MemArrayDict'>, os0=0,
            fvrays=False)

    _mk_tree (pref="", ltype=<class 'raytraverse.lightfield.memarraydict.MemArrayDict'>, os0=0)

    apply_coef (pi, coefs)

    add_to_img (img, mask, pi, i, d, coefs=1, vm=None, radius=3)

    query_ray (pi, vecs, interp=1)

    query_all_pts (vecs, interp=1)

    query_ball (pi, vecs, viewangle=180)

    _dview (idx, pdirs, mask, res=800, showsample=True)

    direct_view (res=800, showsample=True, items=None)
        create a summary image of lightfield for each vpt
```

## SCBinField

```
class raytraverse.lightfield.SCBinField (scene, rebuild=False, prefix='sky', **kwargs)
    Bases: raytraverse.lightfield.lightfieldkd.LightFieldKD
    container for accessing sampled data where every ray has a value for each source

    apply_coef (pi, coefs)
```

## SunField

**class** raytraverse.lightfield.SunField(*scene, suns, rebuild=False, rmraw=False*)

Bases: raytraverse.lightfield.lightfieldkd.LightFieldKD

container for sun view data

### Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **suns** (raytraverse.sunsetter.SunSetter) – sun class containing sun vectors and SunMapper (passed to SunViewField)
- **rebuild** (*bool, optional*) – build kd-tree even if one exists

**suns** = None

raytraverse.sunsetter.SunSetter

**view** = None

raytraverse.lightfield.SunViewField

**raw\_files** ()

get list of files used to build field

**\_mk\_tree** (*pref="", ltype=<class 'list'>, os0=0*)

**items** ()

**keymap** ()

**add\_to\_img** (*img, mask, pi, i, d, coefs=1, vm=None, radius=3*)

**direct\_view** (*res=200, showsample=False, items=None*)

create a summary image of lightfield for each vpt

## SunViewField

**class** raytraverse.lightfield.SunViewField(*scene, suns, rebuild=False, rmraw=False*)

Bases: raytraverse.lightfield.lightfield.LightField

container for sun view data

### Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **suns** (raytraverse.sunsetter.SunSetter) – prefix of data files to integrate
- **rebuild** (*bool, optional*) – build kd-tree even if one exists

**suns** = None

raytraverse.sunsetter.SunSetter

**sunmap** = None

raytraverse.sunmapper.SunMapper

**raw\_files** ()

get list of files used to build field

**property raster**

individual pixels forming shape of the sun, stored as uv coordinates with basis viewmapper about sun direction and diameter. indexed like vec, lum, and omega

**property scene**

scene information

**Getter** Returns this integrator's scene

**Setter** Set this integrator's scene

**Type** *raytraverse.scene.Scene*

**items** ()

**\_grp\_by\_sun** (*vecs, lums, shape, pti, suni*)

**\_build\_suns** (*vecs, lums, shape*)  
loop through points/suns and group adjacent rays

**\_to\_pix** (*rxyz, atv, vm, res*)

**\_smudge** (*cnt, omegap, omegasp*)  
hack to ensure equal energy and max luminance)

**add\_to\_img** (*img, pi, sun, vm*)

#### Parameters

- **img** –
- **pi** –
- **sun** –
- **vm** (*raytraverse.mapper.ViewMapper*) –

**metric** (*psi, v, s, metricfuncs, \*\*kwargs*)

**direct\_view** (*res=2*)  
create a summary image of all sun discs from each of vpts

## MemArrayDict

**class** *raytraverse.lightfield.memarraydict.MemArrayDict*

Bases: dict

a dictionary like object that holds arguments for numpy.memmap, the getter returns a view to the array

**static** **\_map** (*i*)

**values** () → an object providing a view on D's values

**constructors** ()

**full\_array** ()

**full\_constructor** ()

**index\_strides** ()

## 1.5.6 raytraverse.integrator

### Integrator

**class** *raytraverse.integrator.Integrator* (*skyfield, sunfield=None, wea=None, loc=None, skyro=0.0, \*\*kwargs*)

Bases: object

class to generate outputs from skyfield sunfield and sky conditions

This class provides an interface to:

1. generate sky data using the perez
2. combine lightfield data for sky and sun (handling sparsely populated sun data)

3. apply sky data to the lightfield queries
4. output luminance maps and photometric quantities and visual comfort metrics

### Parameters

- **skyfield** (`raytraverse.lightfield.SCBinField`) – class containing sky data
- **sunfield** (`raytraverse.lightfield.SunField`) – class containing sun data (should share its scene parameter with the given skyfield)
- **wea** (`str, np.array, optional`) – path to epw, wea, or .npz file or np.array, if loc not set attempts to extract location data (if needed). The Integrator does not need to be initialized with weather data but for convenience can be. If skydata is not set then the optional parameter of `get_sky_matrix()` is required or an Exception will be raised.
- **loc** (`((float, float, int), optional)`) – location data given as lat, lon, mer with + west of prime meridian overrides location data in wea (but not in sunfield)
- **skyro** (`(float, optional)`) – angle in degrees counter-clockwise to rotate sky (to correct model north, equivalent to clockwise rotation of scene) does not override rotation in SunField)

**scene = None**

`raytraverse.scene.Scene`

**suns = None**

`raytraverse.sunsetter.SunSetter`

**sky = None**

`raytraverse.scene.SkyInfo`

**sunfield = None**

`raytraverse.lightfield.SunField`

**skyfield = None**

`raytraverse.lightfield.SCBinField`

**property skydata**

sky data formatted as dx, dy, dz, dirnorm, diffhoriz

**Getter** Returns this scene's skydata

**Setter** Sets this scene's skydata from file path or

**Type** np.array

**format\_skydata** (*dat*)

process dat argument as skydata

**Parameters** **dat** (`str, np.array`) – This method takes either a file path or np.array. File path can point to a wea, epw, or .npz file. Loaded array must be one of the following:  
- 4 col: alt, az, dir, diff - 5 col: dx, dy, dz, dir, diff - 5 col: m, d, h, dir, diff'

**Returns** dx, dy, dz, dir, diff

**Return type** np.array

**get\_sky\_mtx** (*skydata=None*)

generate sky, grnd and sun coefficients from sky data using perez

**Parameters** **skydata** (`str, np.array, optional`) – if None, uses object skydata (will raise error if unassigned) see `format_skydata()` for argument specifics.

**Returns**



- **smtx** (*np.array*) – shape (len(skydata), skyres\*\*2) coefficients for each sky patch each row is a timestep, timesteps where a sun exists exclude the sun coefficient, otherwise the patch enclosing the sun position contains the energy of the sun
- **grnd** (*np.array*) – shape (len(skydata),) coefficients for ground at each timestep
- **sun** (*np.array*) – shape (len(skydata), 5) sun position (index 0,1,2) and coefficients for sun at each timestep assuming the true solid angle of the sun (index 3) and the weighted value for the sky patch (index 4).
- **si** (*np.array*) – shape (len(skydata), 2) index for self.suns.suns pointing to correct proxy source (col 0) and sunbin for patch mapping (col 1)
- **daysteps** (*np.array*) – shape (len(skydata),) boolean array masking timesteps when sun is below horizon

**header** ()

generate image header string

**\_hdr** (*res, vm, keymap, pi, suni, sun, pdirs, si, vstr, outf, interp, mask, skyv*)

**hdr** (*pts, smtx, grnd=None, suns=None, suni=None, daysteps=None, vname='view', viewangle=180.0, res=400, interp=1*)

#### Parameters

- **pts** (*np.array*) – points and view dirs shape (N, 6)
- **smtx** (*np.array*) – shape (len(skydata), skyres\*\*2) coefficients for each sky patch each row is a timestep, timesteps where a sun exists exclude the sun coefficient, otherwise the patch enclosing the sun position contains the energy of the sun
- **grnd** (*np.array*) – shape (len(skydata),) coefficients for ground at each timestep
- **suns** (*np.array*) – shape (len(skydata), 5) sun position (index 0,1,2) and coefficients for sun at each timestep assuming the true solid angle of the sun (index 3) and the weighted value for the sky patch (index 4).
- **suni** (*np.array*) – shape (len(skydata), 2) index for self.suns.suns pointing to correct proxy source (col 0) and sunbin for patch mapping (col 1)
- **daysteps** (*np.array*) – shape (len(skydata),) boolean array masking timesteps when sun is below horizon
- **vname** (*str*) – view name for output file
- **viewangle** (*float, optional*) – degree opening of view cone
- **res** (*int, optional*) – image resolution
- **interp** (*int, optional*) – number of nearest points to interpolate between. 1 will resemble voronoi patches

**metric** (*pts, smtx, grnd=None, suns=None, suni=None, daysteps=None, metricfuncs=(<function illum>, ), \*\*kwargs*)

calculate luminance based metrics for given sensors and skyvecs

#### Parameters

- **pts** (*np.array*) – points and view dirs shape (N, 6)
- **smtx** (*np.array*) – shape (len(skydata), skyres^2) coefficients for each sky patch each row is a timestep, timesteps where a sun exists exclude the sun coefficient, otherwise the patch enclosing the sun position contains the energy of the sun
- **grnd** (*np.array*) – shape (len(skydata),) coefficients for ground at each timestep
- **suns** (*np.array*) – shape (len(skydata), 5) sun position (index 0,1,2) and coefficients for sun at each timestep assuming the true solid angle of the sun (index 3) and the weighted value for the sky patch (index 4).

- **sun\_i** (*np.array*) – shape (len(skydata), 2) index for self.suns.suns pointing to correct proxy source (col 0) and sunbin for patch mapping (col 1)
- **daysteps** (*np.array*) – shape (len(skydata),) boolean array masking timesteps when sun is below horizon
- **metricfuncs** (*tuple*) – list of metric functions to apply, with the call signature: `f(view, rays, omegas, lum, kwargs)`

**Returns** **metrics** – metrics at each point/direction and sky weighting axes ordered as: (view, points, skys, metrics) an additional column on the final axis is 0 when only the skyfield is used, 1 when sun reflections exists and 2 when a view to the sun also exists.

**Return type** `np.array`

## 1.5.7 raytraverse.craytraverse

## 1.5.8 raytraverse.draw

wavelet and associated probability functions.

`raytraverse.draw.get_detail(samps, axes)`  
run high pass filter over given axes

`raytraverse.draw.from_pdf(pdf, threshold)`

## 1.5.9 raytraverse.io

functions for reading and writing

**class** `raytraverse.io.CaptureStdOut` (*b=False, store=True, outf=None*)  
Bases: `object`

redirect output streams at system level (including `c printf`)

### Parameters

- **b** (*bool, optional*) – read data as bytes
- **store** (*bool, optional*) – record stdout in a `IOStream`, value accesible through `self.stdout`
- **outf** (*IOBase, optional*) – if not `None`, must be writable, closed on exit

### Notes

```
with CaptureStdOut() as capture:
    do stuff
capout = capture.stdout
```

when using with `pytest` include the `-s` flag or this class has no effect

**property** `stdout`

**drain\_bytes** ()  
read stdout as bytes

**drain\_str** ()  
read stdout as unicode

`raytraverse.io.call_sampler(outf, command, vecs, shape)`  
make subprocess call to sampler given as `command`, expects `rgb` value as return for each `vec`

### Parameters

- **outf** (*str*) – path to write out to
- **command** (*str*) – command line with executable and options
- **vecs** (*np.array*) – vectors to pass as stdin to command
- **shape** (*tuple*) – shape of expected output

**Returns** **lums** – of length `vectors.shape[0]`

**Return type** `np.array`

`raytraverse.io.bytefile2rad` (*f, shape, slc=Ellipsis, subs='ijk,k->ij', offset=0*)

`raytraverse.io.np2bytes` (*ar, dtype='<f'*)  
format *ar* as bytestring

#### Parameters

- **ar** (*np.array*) –
- **dtype** (*str*) – argument to pass to `np.dtype()`

#### Returns

**Return type** `bytes`

`raytraverse.io.bytes2np` (*buf, shape, dtype='<f'*)  
read *ar* from bytestring

#### Parameters

- **buf** (*bytes, str*) –
- **shape** (*tuple*) – array shape
- **dtype** (*str*) – argument to pass to `np.dtype()`

#### Returns

**Return type** `np.array`

`raytraverse.io.bytefile2np` (*f, shape, dtype='<f'*)  
read binary data from *f*

#### Parameters

- **f** (*IOBase*) – file object to read array from
- **shape** (*tuple*) – array shape
- **dtype** (*str*) – argument to pass to `np.dtype()`

**Returns** necessary for reconstruction

**Return type** `ar.shape`

`raytraverse.io.array2hdr` (*ar, imgf, header=None*)  
write 2d `np.array` (*x,y*) to `hdr` image format

#### Parameters

- **ar** (*np.array*) –
- **imgf** (*file path to right*) –
- **header** (*list of header lines to append to image header*) –

`raytraverse.io.carray2hdr` (*ar, imgf, header=None*)  
write color channel `np.array` (*3, x, y*) to `hdr` image format

#### Parameters

- **ar** (*np.array*) –
- **imgf** (*file path to right*) –

- **header** (*list of header lines to append to image header*) –

`raytraverse.io.hdr2array (imgf)`  
read np.array from hdr image

**Parameters** `imgf` (*file path of image*) –

**Returns** `ar`

**Return type** np.array

`raytraverse.io.rgb2lum (rgbe)`  
convert from Radiance hdr rgbe 4-byte data format to floating point luminance.

**Parameters** `rgbe` (*np.array*) – r,g,b,e unsigned integers according to: <http://radsite.lbl.gov/radiance/refer/filefmts.pdf>

**Returns** `lum`

**Return type** luminance in cd/m<sup>2</sup>

## 1.5.10 raytraverse.metric

standardized metric functions

`raytraverse.metric.illum (vm, vec, omega, lum, scale=179, **kwargs)`

`raytraverse.metric.avglum (vm, vec, omega, lum, scale=179, area=None, **kwargs)`

`raytraverse.metric.sqlum (vm, vec, omega, lum, scale=179, area=None, **kwargs)`

## 1.5.11 raytraverse.plot

functions for plotting data

`raytraverse.plot.save_img (fig, ax, outf, title=None)`

`raytraverse.plot.imshow (im, figsize=10, 10, outf=None, **kwargs)`

`raytraverse.plot.mk_img_setup (lums, bounds=None, figsize=10, 10, ext=1)`

`raytraverse.plot.set_ang_ticks (ax, ext)`

`raytraverse.plot.colormap (colors, norm)`

`raytraverse.plot.plot_patches (ax, patches, patchargs=None)`

## 1.5.12 raytraverse.quickplot

functions for plotting data

`raytraverse.quickplot.imshow (im, figsize=10, 10, outf=None, **kwargs)`

`raytraverse.quickplot.hist (lums, bins='auto', outf=None, **kwargs)`

### 1.5.13 raytraverse.skycalc

functions for loading sky data and computing sun position

`raytraverse.skycalc.read_epw(epw)`  
read daylight sky data from epw or wea file

**Returns out** – (month, day, hour, dirnorn, difhoriz)

**Return type** np.array

`raytraverse.skycalc.get_loc_epw(epw, name=False)`  
get location from epw or wea header

`raytraverse.skycalc.sunpos_utc(timesteps, lat, lon, builtin=True)`  
Calculate sun position with local time

Calculate sun position (altitude, azimuth) for a particular location (longitude, latitude) for a specific date and time (time is in UTC)

#### Parameters

- **timesteps** (`np.array(datetime.datetime)`) –
- **lon** (`float`) – longitude in decimals. West is +ve
- **lat** (`float`) – latitude in decimals. North is +ve
- **builtin** (`bool`) – use skyfield builtin timescale

#### Returns

- (`skyfield.units.Angle, skyfield.units.Angle`)
- *altitude and azimuth in degrees*

`raytraverse.skycalc.row_2_datetime64(ts, year=2020)`

`raytraverse.skycalc.datetime64_2_datetime(timesteps, mer=0.0)`  
convert datetime representation and offset for timezone

#### Parameters

- **timesteps** (`np.array(np.datetime64)`) –
- **mer** (`float`) – Meridian of the time zone. West is +ve

#### Returns

**Return type** np.array(datetime.datetime)

`raytraverse.skycalc.sunpos_degrees(timesteps, lat, lon, mer, builtin=True, ro=0.0)`  
Calculate sun position with local time

Calculate sun position (altitude, azimuth) for a particular location (longitude, latitude) for a specific date and time (time is in local time)

#### Parameters

- **timesteps** (`np.array(np.datetime64)`) –
- **lon** (`float`) – longitude in decimals. West is +ve
- **lat** (`float`) – latitude in decimals. North is +ve
- **mer** (`float`) – Meridian of the time zone. West is +ve
- **builtin** (`bool, optional`) – use skyfield builtin timescale
- **ro** (`float, optional`) – ccw rotation (project to true north) in degrees

**Returns** Sun position as (altitude, azimuth) in degrees

**Return type** np.array([float, float])

`raytraverse.skycalc.sunpos_radians` (*timesteps, lat, lon, mer, builtin=True, ro=0.0*)

Calculate sun position with local time

Calculate sun position (altitude, azimuth) for a particular location (longitude, latitude) for a specific date and time (time is in local time)

**Parameters**

- **timesteps** (*np.array(np.datetime64)*) –
- **lon** (*float*) – longitude in decimals. West is +ve
- **lat** (*float*) – latitude in decimals. North is +ve
- **mer** (*float*) – Meridian of the time zone. West is +ve
- **builtin** (*bool*) – use skyfield builtin timescale
- **ro** (*float, optional*) – ccw rotation (project to true north) in radians

**Returns** Sun position as (altitude, azimuth) in radians

**Return type** `np.array([float, float])`

`raytraverse.skycalc.sunpos_xyz` (*timesteps, lat, lon, mer, builtin=True, ro=0.0*)

Calculate sun position with local time

Calculate sun position (altitude, azimuth) for a particular location (longitude, latitude) for a specific date and time (time is in local time)

**Parameters**

- **timesteps** (*np.array(np.datetime64)*) –
- **lon** (*float*) – longitude in decimals. West is +ve
- **lat** (*float*) – latitude in decimals. North is +ve
- **mer** (*float*) – Meridian of the time zone. West is +ve
- **builtin** (*bool*) – use skyfield builtin timescale
- **ro** (*float, optional*) – ccw rotation (project to true north) in degrees

**Returns** Sun position as (x, y, z)

**Return type** `np.array`

`raytraverse.skycalc.generate_wea` (*ts, wea, interp='linear'*)

`raytraverse.skycalc.coeff_lum_perez` (*sunz, epsilon, delta, catn*)  
matches `coeff_lum_perez` in `gendaylit.c`

`raytraverse.skycalc.perez_apply_coef` (*coefs, cgamma, dz*)

`raytraverse.skycalc.perez_lum_raw` (*tp, dz, sunz, coefs*)  
matches `calc_rel_lum_perez` in `gendaylit.c`

`raytraverse.skycalc.perez_lum` (*xyz, coefs*)  
matches `perezlum.cal`

`raytraverse.skycalc.perez` (*sxyz, dirdif, md=None*)  
compute perez coefficients

## Notes

to match the results of gendaylit, for a given sun angle without associated date, the assumed eccentricity is 1.035020

### Parameters

- **sxyz** (*np.array*) – (N, 3) dx, dy, dz sun position
- **dirdif** (*np.array*) – (N, 2) direct normal, diffuse horizontal W/m<sup>2</sup>
- **md** (*np.array, optional*) – (N, 2) month day of sky calcs (for more precise eccentricity calc)

**Returns** **perez** – (N, 10) diffuse normalization, ground brightness, perez coefs, x, y, z

**Return type** *np.array*

`raytraverse.skycalc.sky_mtx(sxyz, dirdif, side, jn=4)`  
generate sky, ground and sun values from sun position and sky values

### Parameters

- **sxyz** (*np.array*) – sun directions (N, 3)
- **dirdif** (*np.array*) – direct normal and diffuse horizontal radiation (W/m<sup>2</sup>) (N, 2)
- **side** (*int*) – sky subdivision
- **jn** (*int*) – sky patch subdivision  $n = jn^2$

### Returns

- **skymtx** (*np.array*) – (N, side\*side)
- **grndval** (*np.array*) – (N,)
- **sunval** (*np.array*) – (N, 4) - sun direction and radiance

## 1.5.14 raytraverse.translate

functions for translating between coordinate spaces and resolutions

`raytraverse.translate.norm(v)`  
normalize 2D array of vectors along last dimension

`raytraverse.translate.norm1(v)`  
normalize flat vector

`raytraverse.translate.tpnorm(thetaphi)`  
normalize angular vector to 0-pi, 0-2pi

`raytraverse.translate.uv2xy(uv)`  
translate from unit square (0,1),(0,1) to disk (x,y) <http://psgraphics.blogspot.com/2011/01/improved-code-for-concentric-map.html>.

`raytraverse.translate.uv2xyz(uv, axes=0, 1, 2, xsign=-1)`  
translate from 2 x unit square (0,2),(0,1) to unit sphere (x,y,z) <http://psgraphics.blogspot.com/2011/01/improved-code-for-concentric-map.html>.

`raytraverse.translate.xyz2uv(xyz, normalize=False, axes=0, 1, 2, flipu=True)`  
translate from vector x,y,z (normalized) to u,v (0,2),(0,1) Shirley, Peter, and Kenneth Chiu. A Low Distortion Map Between Disk and Square. Journal of Graphics Tools, vol. 2, no. 3, Jan. 1997, pp. 45-52. Taylor and Francis+NEJM, doi:10.1080/10867651.1997.10487479.

`raytraverse.translate.xyz2xy(xyz, axes=0, 1, 2, flip=True)`

`raytraverse.translate.pxy2xyz(pxy, viewangle=180.0)`

`raytraverse.translate.tp2xyz (thetaphi, normalize=True)`  
calculate x,y,z vector from theta (0-pi) and phi (0-2pi) RHS Z-up

`raytraverse.translate.xyz2tp (xyz)`  
calculate theta (0-pi), phi from x,y,z RHS Z-up

`raytraverse.translate.tp2uv (thetaphi)`  
calculate UV from theta (0-pi), phi

`raytraverse.translate.uv2tp (uv)`  
calculate theta (0-pi), phi from UV

`raytraverse.translate.uv2ij (uv, side)`

`raytraverse.translate.uv2bin (uv, side)`

`raytraverse.translate.bin2uv (bn, side)`

`raytraverse.translate.bin_borders (sb, side)`

`raytraverse.translate.resample (samps, ts=None, gauss=True, radius=None)`  
simple array resampling. requires whole number multiple scaling.

**Parameters**

- **samps** (*np.array*) – array to resample along each axis
- **ts** (*tuple, optional*) – shape of output array, should be multiple of samps.shape
- **gauss** (*bool, optional*) – apply gaussian filter to upsampling
- **radius** (*float, optional*) – when gauss is True, filter radius, default is the scale ratio - 1

**Returns** to resampled array

**Return type** `np.array`

`raytraverse.translate.interpolate2d (a, s)`

`raytraverse.translate.rmtx_elem (theta, axis=2, degrees=True)`

`raytraverse.translate.rotate_elem (v, theta, axis=2, degrees=True)`

`raytraverse.translate.rmtx_yp (v)`  
generate a pair of rotation matrices to transform from vector v to z, enforcing a z-up in the source space and a y-up in the destination. If v is z, returns pair of identity matrices, if v is -z returns pair of 180 degree rotation matrices.

**Parameters** **v** (*array-like of size (3,)*) – the vector direction representing the starting coordinate space

**Returns** **ymtx**, **pmtx** – two rotation matrices to be premultiplied in order to reverse transform, swap order and transpose. Forward: `pmtx@(ymtx@xyz.T)).T` Backward: `ymtx.T@(pmtx.T@xyz.T)).T`

**Return type** (`np.array`, `np.array`)

`raytraverse.translate.chord2theta (c)`  
compute angle from chord on unit circle

**Parameters** **c** (*float*) – chord or euclidean distance between normalized direction vectors

**Returns** **theta** – angle captured by chord

**Return type** `float`

`raytraverse.translate.theta2chord (theta)`  
compute chord length on unit sphere from angle

**Parameters** **theta** (*float*) – angle

**Returns** **c** – chord or euclidean distance between normalized direction vectors



**Return type** float

```
raytraverse.translate.aa2xyz (aa)
```

```
raytraverse.translate.xyz2aa (xyz)
```

## 1.5.15 raytraverse

```
raytraverse [OPTIONS] OUT COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

the raytraverse executable is a command line interface to the raytraverse python package for running and evaluating climate based daylight models. sub commands of raytraverse can be chained but should be invoked in the order given.

the easiest way to manage options and sure that Scene and SunSetter classes are properly reloaded is to use a configuration file, to make a template:

```
raytraverse --template > run.cfg
```

after adjusting the settings, than each command can be invoked in turn and any dependencies will be loaded with the correct options, a complete run and evaluation can then be called by:

```
raytraverse -c run.cfg OUT sky sunrun integrate
```

as both scene and sun will be invoked automatically as needed.

### Arguments:

- out: path to new or existing directory for raytraverse run

## Arguments

### OUT

Required argument

## Options

### VALUE OPTIONS:

**-c, --config** <PATH>  
path of config file to load

### FLAGS (DEFAULT FALSE):

**--template, --no-template**  
write default options to std out as config

**Default** False

**HELP:****-opts, --opts**

check parsed options

**Default** False**--debug**

show traceback on exceptions

**Default** False**--version**

Show the version and exit.

**Default** False**Commands****scene**

The scene commands creates a Scene object. . .

**sky**

the sky command initializes and runs a sky. . .

**suns**

the suns command provides a number of options. . .

**sunrun**

the sunrun command initializes and runs a . . .

**integrate**

the integrate command combines sky and sun. . .

**scene**

```
raytraverse scene [OPTIONS]
```

The scene commands creates a Scene object which holds geometric information about the model including object geometry (and defined materials), the analysis plane and the desired resolutions for sky and analysis plane subdivision

**Options****VALUE OPTIONS:****-area** <TEXT>

radiance scene file containing planar geometry of analysis area

**-maxspec** <FLOAT>

an important parameter for guiding reflected sun rays. contribution values above this threshold are assumed to be direct view rays. If possible, (1) this value should be less than the tvis of the darkest glass in the scene, and (2) greater than the highest expected contribution from a specular reflection or scattering interaction. If it is not possible to meet both conditions, then ensure that condition (2) is met and consider using a substantially higher skyres to avoid massive over sampling of direct view rays

**Default** 0.3**-ptres** <FLOAT>

resolution of point subdivision on analysis plane. units match radiance scene file

**Default** 2.0

**-scene** <TEXT>

space separated list of radiance scene files (no sky) or precompiled octree

**-skyres** <FLOAT>

sky is subdivided according to a shirley-chiu disk to square mapping, total number of sky bins will equal skyres<sup>2</sup>. solid angle of each patch will be  $2\pi/(\text{skyres}^2)$

**Default** 10.0

### FLAGS (DEFAULT TRUE):

**--reload, --no-reload**

if a scene already exists at OUT reload it, note that if this is False and overwrite is False, the program will abort

**Default** True

### FLAGS (DEFAULT FALSE):

**--info, --no-info**

print info on scene to stderr

**Default** False

**--overwrite, --no-overwrite**

Warning! if set to True and reload is False all files in OUT will be deleted

**Default** False

**--points, --no-points**

print point locations to stdout

**Default** False

### HELP:

**-opts, --opts**

check parsed options

**Default** False

**--debug**

show traceback on exceptions

**Default** False

**--version**

Show the version and exit.

**Default** False

### sky

```
raytraverse sky [OPTIONS]
```

the sky command initializes and runs a sky sampler and then reads the results for integration by building a SCBinField. sky should be invoked before calling suns, as the sky contributions are used to select the necessary sun positions to run

## Options

### VALUE OPTIONS:

**-accuracy** <FLOAT>

a generic accuracy parameter that sets the threshold variance to sample. A value of 1 will have a sample count at the final sampling level equal to the number of directions with a contribution variance greater than .25

**Default** 1.0

**-fdres** <INTEGER>

the final directional sampling resolution, yielding a grid of potential samples at  $2^{\text{fdres}} \times 2^{\text{fdres}}$  per hemisphere

**Default** 9

**-idres** <INTEGER>

the initial directional sampling resolution. each side of the sampling square (representing a hemisphere) will be subdivided  $2^{\text{idres}}$ , yielding  $2^{(2*\text{idres})}$  samples and a resolution of  $2^{(2*\text{idres})}/(2\pi)$  samples/steradian. this value should be smaller than 1/2 the size of the smallest view to an aperture that should be captured with 100% certainty

**Default** 4

**-rcopts** <TEXT>

rtrace options to pass to the rcontrib call see the man pages for rtrace, rcontrib, and rcontrib -defaults for more information

**Default** -ab 7 -ad 60000 -as 30000 -lw 1e-7 -st 0 -ss 16

### FLAGS (DEFAULT TRUE):

**--rmraw, --no-rmraw**

if True removes output of sampler.run(), after SCBinField is constructed. Note that SCBinField cannot be rebuilt once raw files are removed

**Default** True

**--run, --no-run**

if True calls sampler.run()

**Default** True

### FLAGS (DEFAULT FALSE):

**--plotdview, --no-plotdview**

plot a direct view of the sky field (as a .hdr file), this is equivalent to integrating with a value of 1 for all sky patches with no interpolation, plots pixels of actualsample vectors in red

**Default** False

**--plotp, --no-plotp**

for diagnostics only, plots the pdf at each level for point[0,0] in an interactive display (note that program will hang until the user closes the plot window at each level)

**Default** False

**HELP:****-opts, --opts**

check parsed options

**Default** False**--debug**

show traceback on exceptions

**Default** False**--version**

Show the version and exit.

**Default** False**suns**

```
raytraverse suns [OPTIONS]
```

the suns command provides a number of options for creating sun positions used by sunrun see wea and usepositions options for details

Note:

the wea and skyro parameters are used to reduce the number of suns in cases where a specific site is known. Only suns within the solar transit (or positions if usepositions is True will be selected. It is important to note that when integrating, if a sun position outside this range is queried than results will not include the more detailed simulations involved in sunrun and will instead place the suns energy within the nearest sky patch. if skyres is small and or the patch is directly visible this will introduce significant bias in most metrics.

**Options****VALUE OPTIONS:****-loc <FLOATS>**

specify the scene location (if not specified in -wea or to override. give as “lat lon mer” where lat is + North, lon is + West and mer is the timezone meridian (full hours are 15 degree increments)

**-skyro <FLOAT>**

counter clockwise rotation (in degrees) of the sky to rotate true North to project North, so if project North is 10 degrees East of North, skyro=10

**Default** 0.0**-srct <FLOAT>**

if the contribution of a sky patch (for any view ray) is above this threshold, a sun will be created in this patch

**Default** 0.01**-sunres <FLOAT>**

resolution in degrees of the sky patch grid in which to stratify sun samples. Suns are randomly located within the grid, so this corresponds to the average distance between sources. The average error to a randomly selected sun position will be on average ~0.4 times this value

**Default** 10.0**-wea <TEXT>**

path to weather/sun position file. possible formats are:

1. .wea file

2. .wea file without header (require -loc and --no-usepositions)
3. .epw file
4. .epw file without header (require -loc and --no-usepositions)
5. 3 column tsv file, each row is dx, dy, dz of candidate sun position (requires --usepositions)
6. 4 column tsv file, each row is altitude, azimuth, direct normal, diff. horizontal of candidate suns (requires --usepositions)
7. 5 column tsv file, each row is dx, dy, dz, direct normal, diff. horizontal of candidate suns (requires --usepositions)

tsv files are loaded with loadtxt

### FLAGS (DEFAULT TRUE):

#### **--reload, --no-reload**

if False, regenerates sun positions, because positions may be randomly selected this will make any sunrun results obsolete

**Default** True

### FLAGS (DEFAULT FALSE):

#### **--plotdview, --no-plotdview**

creates a png showing sun positions on an angular fisheye projection of the sky. sky patches are colored by the maximum contributing ray to the scene

**Default** False

#### **--usepositions, --no-usepositions**

if True, sun positions will be chosen from the positions listed in wea. if more than one position is a candidate for that particular sky patch (as determined by sunres) than a random choice will be made. by using one of the tsv format options for wea, and preselecting sun positions such that there is 1 per patch a deterministic result can be achieved.

**Default** False

### HELP:

#### **-opts, --opts**

check parsed options

**Default** False

#### **--debug**

show traceback on exceptions

**Default** False

#### **--version**

Show the version and exit.

**Default** False

## sunrun

```
raytraverse sunrun [OPTIONS]
```

the sunrun command initializes and runs a sun sampler and then readies the results for integration by building a SunField.

## Options

### VALUE OPTIONS:

#### **-accuracy** <FLOAT>

a generic accuracy parameter that sets the threshold variance to sample. A value of 1 will have a sample count at the final sampling level equal to the number of directions with a contribution variance greater than .25

**Default** 1.0

#### **-fdres** <INTEGER>

the final directional sampling resolution, yielding a grid of potential samples at  $2^{\text{fdres}} \times 2^{\text{fdres}}$  per hemisphere

**Default** 10

#### **-idres** <INTEGER>

the initial directional sampling resolution. each side of the sampling square (representing a hemisphere) will be subdivided  $2^{\text{idres}}$ , yielding  $2^{(2*\text{idres})}$  samples and a resolution of  $2^{(2*\text{idres})}/(2\pi)$  samples/steradian. this value should be smaller than 1/2 the size of the smallest view to an aperture that should be captured with 100% certainty

**Default** 4

#### **-rcopts** <TEXT>

rtrace options for sun reflection runs see the man pages for rtrace, and rtrace -defaults for more information

**Default** -ab 6 -ad 3000 -as 1500 -st 0 -ss 16 -aa .1

#### **-speclevel** <INTEGER>

at this sampling level, pdf is made from brightness of sky sampling rather than progressive variance to look for fine scale specular highlights, this should be atleast 1 level from the end and the resolution of this level should be smaller than the size of the source

**Default** 9

### FLAGS (DEFAULT TRUE):

#### **--ambcache, --no-ambcache**

whether the rcopts indicate that the calculation will use ambient caching (and thus should write an -af file argument to the engine)

**Default** True

#### **--reflection, --no-reflection**

run/build/plot reflected sun components

**Default** True

#### **--rmraw, --no-rmraw**

if True removes output of sampler.run(), after SCBinField is constructed. Note that SCBinField cannot be rebuilt once raw files are removed

**Default** True

**--run, --no-run**  
if True calls `sampler.run()`

**Default** True

**--view, --no-view**  
run/build/plot direct sun views

**Default** True

## FLAGS (DEFAULT FALSE):

**--keepamb, --no-keepamb**  
whether to keep ambient files after run, if kept, a successive call will load these ambient files, so care must be taken to not change any parameters

**Default** False

**--plotdview, --no-plotdview**  
plot a direct view of the sky field (as a .hdr file), this is equivalent to integrating with a value of 1 for all sky patches with no interpolation, plots pixels of actualsample vectors in red

**Default** False

**--plotp, --no-plotp**  
for diagnostics only, plots the pdf at each level for point[0,0] in an interactive display (note that program will hang until the user closes the plot window at each level)

**Default** False

## HELP:

**-opts, --opts**  
check parsed options

**Default** False

**--debug**  
show traceback on exceptions

**Default** False

**--version**  
Show the version and exit.

**Default** False

## integrate

```
raytraverse integrate [OPTIONS]
```

the integrate command combines sky and sun results and evaluates the given set of positions and sky conditions



## Options

### VALUE OPTIONS:

**-interp** <INTEGER>

number of nearby rays to use for interpolation of hdr output (weighted by a gaussian filter). this does not apply to metric calculations

**Default** 12

**-loc** <FLOATS>

specify the scene location (if not specified in -wea or to override. give as "lat lon mer" where lat is + North, lon is + West and mer is the timezone meridian (full hours are 15 degree increments)

**-pts** <TEXT>

points to evaluate, this can be a .npy file, a whitespace separated text file or entered as a string with commas between components of a point and spaces between points. in all cases each point requires 6 numbers x,y,z,dx,dy,dz so the shape of the array will be (N, 6)

**Default** 0,0,0,0,-1,0

**-res** <INTEGER>

the resolution of hdr output in pixels

**Default** 800

**-skyro** <FLOAT>

counter clockwise rotation (in degrees) of the sky to rotate true North to project North, so if project North is 10 degrees East of North, skyro=10

**Default** 0.0

**-vname** <TEXT>

name to include with hdr outputs

**Default** view

**-wea** <TEXT>

path to weather/sun position file. possible formats are:

1. .wea file
2. .wea file without header (requires -loc)
3. .epw file
4. .epw file without header (requires -loc)
5. 4 column tsv file, each row is altitude, azimuth, direct normal, diff. horizontal of candidate suns (requires -usepositions)
6. 5 column tsv file, each row is dx, dy, dz, direct normal, diff. horizontal of candidate suns (requires -usepositions)

tsv files are loaded with loadtxt

### FLAGS (DEFAULT TRUE):

**--hdr, --no-hdr**

produce an hdr output for each point and line in wea

**Default** True

**--metric, --no-metric**

calculate metrics for each point and wea file output is ordered by point than sky

**Default** True

### FLAGS (DEFAULT FALSE):

**--header, --no-header**

print column headings on metric output

**Default** False

**--skyonly, --no-skyonly**

if True, only integrate on Sky Field, useful for diagnostics

**Default** False

### HELP:

**-opts, --opts**

check parsed options

**Default** False

**--debug**

show traceback on exceptions

**Default** False

**--version**

Show the version and exit.

**Default** False

## 1.5.16 History

### 0.1.0 (2020-05-19)

- First release on PyPI.

## 1.5.17 Index

## 1.5.18 Search

## 1.5.19 Todo

## PYTHON MODULE INDEX

### r

- `raytraverse.draw`, [22](#)
- `raytraverse.io`, [22](#)
- `raytraverse.metric`, [24](#)
- `raytraverse.plot`, [24](#)
- `raytraverse.quickplot`, [24](#)
- `raytraverse.skycalc`, [25](#)
- `raytraverse.translate`, [27](#)



## Symbols

`_build_suns()` (*raytraverse.lightfield.SunViewField method*), 19  
`_dview()` (*raytraverse.lightfield.LightFieldKD method*), 17  
`_get_vl()` (*raytraverse.lightfield.LightFieldKD method*), 17  
`_grp_by_sun()` (*raytraverse.lightfield.SunViewField method*), 19  
`_hdr()` (*raytraverse.integrator.Integrator method*), 21  
`_map()` (*raytraverse.lightfield.memarraydict.MemArrayDict static method*), 19  
`_mk_tree()` (*raytraverse.lightfield.LightFieldKD method*), 17  
`_mk_tree()` (*raytraverse.lightfield.SunField method*), 18  
`_pyinstance` (*raytraverse.renderer.Renderer attribute*), 9  
`_rad_scene_to_bbox()` (*raytraverse.mapper.SpaceMapper method*), 7  
`_ro_pts()` (*raytraverse.mapper.SpaceMapper method*), 7  
`_set_args()` (*raytraverse.renderer.Renderer class method*), 10  
`_smudge()` (*raytraverse.lightfield.SunViewField method*), 19  
`_to_mem_map()` (*raytraverse.lightfield.LightFieldKD method*), 17  
`_to_pix()` (*raytraverse.lightfield.SunViewField method*), 19  
`_uv2xyz()` (*raytraverse.sampler.Sampler method*), 13  
`_uv2xyz()` (*raytraverse.sampler.SunViewSampler method*), 16  
`_write_suns()` (*raytraverse.scene.SunSetter method*), 4  
`--ambcache`  
     raytraverse-sunrun command line option, 35  
`--config <PATH>`  
     raytraverse command line option, 29  
`--debug`  
     raytraverse command line option, 30  
     raytraverse-integrate command line option, 38  
     raytraverse-scene command line option, 31  
     raytraverse-sky command line option, 33  
     raytraverse-sunrun command line option, 36  
     raytraverse-suns command line option, 34  
`--hdr`  
     raytraverse-integrate command line option, 38  
`--header`  
     raytraverse-integrate command line option, 38  
`--info`  
     raytraverse-scene command line option, 31  
`--keepamb`  
     raytraverse-sunrun command line option, 36  
`--metric`  
     raytraverse-integrate command line option, 38  
`--no-ambcache`  
     raytraverse-sunrun command line option, 35  
`--no-hdr`  
     raytraverse-integrate command line option, 38  
`--no-header`  
     raytraverse-integrate command line option, 38  
`--no-info`  
     raytraverse-scene command line option, 31  
`--no-keepamb`  
     raytraverse-sunrun command line option, 36  
`--no-metric`  
     raytraverse-integrate command line option, 38  
`--no-overwrite`  
     raytraverse-scene command line option, 31

```
--no-plotdview
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,36
    raytraverse-suns command line
        option,34
--no-plotp
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,36
--no-points
    raytraverse-scene command line
        option,31
--no-reflection
    raytraverse-sunrun command line
        option,35
--no-reload
    raytraverse-scene command line
        option,31
    raytraverse-suns command line
        option,34
--no-rmraw
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,35
--no-run
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,35
--no-skyonly
    raytraverse-integrate command
        line option,38
--no-template
    raytraverse command line option,29
--no-usepositions
    raytraverse-suns command line
        option,34
--no-view
    raytraverse-sunrun command line
        option,36
--opts
    raytraverse command line option,30
    raytraverse-integrate command
        line option,38
    raytraverse-scene command line
        option,31
    raytraverse-sky command line
        option,33
    raytraverse-sunrun command line
        option,36
    raytraverse-suns command line
        option,34
--overwrite
    raytraverse-scene command line
        option,31
--plotdview
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,36
    raytraverse-suns command line
        option,34
--plotp
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,36
--points
    raytraverse-scene command line
        option,31
--reflection
    raytraverse-sunrun command line
        option,35
--reload
    raytraverse-scene command line
        option,31
    raytraverse-suns command line
        option,34
--rmraw
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,35
--run
    raytraverse-sky command line
        option,32
    raytraverse-sunrun command line
        option,35
--skyonly
    raytraverse-integrate command
        line option,38
--template
    raytraverse command line option,29
--usepositions
    raytraverse-suns command line
        option,34
--version
    raytraverse command line option,30
    raytraverse-integrate command
        line option,38
    raytraverse-scene command line
        option,31
    raytraverse-sky command line
        option,33
    raytraverse-sunrun command line
        option,36
    raytraverse-suns command line
        option,34
--view
    raytraverse-sunrun command line
        option,36
-accuracy <FLOAT>
```

```

    raytraverse-sky command line
        option, 32
    raytraverse-sunrun command line
        option, 35
-area <TEXT>
    raytraverse-scene command line
        option, 30
-c
    raytraverse command line option, 29
-fdres <INTEGER>
    raytraverse-sky command line
        option, 32
    raytraverse-sunrun command line
        option, 35
-idres <INTEGER>
    raytraverse-sky command line
        option, 32
    raytraverse-sunrun command line
        option, 35
-interp <INTEGER>
    raytraverse-integrate command
        line option, 37
-loc <FLOATS>
    raytraverse-integrate command
        line option, 37
    raytraverse-suns command line
        option, 33
-maxspec <FLOAT>
    raytraverse-scene command line
        option, 30
-opts
    raytraverse command line option, 30
    raytraverse-integrate command
        line option, 38
    raytraverse-scene command line
        option, 31
    raytraverse-sky command line
        option, 33
    raytraverse-sunrun command line
        option, 36
    raytraverse-suns command line
        option, 34
-ptres <FLOAT>
    raytraverse-scene command line
        option, 30
-pts <TEXT>
    raytraverse-integrate command
        line option, 37
-rcopts <TEXT>
    raytraverse-sky command line
        option, 32
    raytraverse-sunrun command line
        option, 35
-res <INTEGER>
    raytraverse-integrate command
        line option, 37
-scene <TEXT>
    raytraverse-scene command line
        option, 30
-skyres <FLOAT>
    raytraverse-scene command line
        option, 31
-skyro <FLOAT>
    raytraverse-integrate command
        line option, 37
    raytraverse-suns command line
        option, 33
-speclevel <INTEGER>
    raytraverse-sunrun command line
        option, 35
-srct <FLOAT>
    raytraverse-suns command line
        option, 33
-sunres <FLOAT>
    raytraverse-suns command line
        option, 33
-vname <TEXT>
    raytraverse-integrate command
        line option, 37
-wea <TEXT>
    raytraverse-integrate command
        line option, 37
    raytraverse-suns command line
        option, 33

```

## A

aa2xyz() (in module *raytraverse.translate*), 29

add\_to\_img() (*raytraverse.lightfield.LightFieldKD* method), 17

add\_to\_img() (*raytraverse.lightfield.SunField* method), 18

add\_to\_img() (*raytraverse.lightfield.SunViewField* method), 19

apply\_coef() (*raytraverse.lightfield.LightFieldKD* method), 17

apply\_coef() (*raytraverse.lightfield.SCBinField* method), 17

arg\_prefix (*raytraverse.renderer.Rcontrib* attribute), 10

arg\_prefix (*raytraverse.renderer.Renderer* attribute), 10

array2hdr() (in module *raytraverse.io*), 23

avglum() (in module *raytraverse.metric*), 24

## B

bands (*raytraverse.sampler.Sampler* attribute), 12

bbox() (*raytraverse.mapper.SpaceMapper* property), 7

bbox() (*raytraverse.mapper.SpaceMapperPt* property), 8

bbox() (*raytraverse.mapper.ViewMapper* property), 9

bin2uv() (in module *raytraverse.translate*), 28

bin\_borders() (in module *raytraverse.translate*), 28

bytefile2np() (in module *raytraverse.io*), 23

bytefile2rad() (in module *raytraverse.io*), 23

`bytes2np()` (in module `raytraverse.io`), 23

## C

`call()` (`raytraverse.renderer.RadianceRenderer` class method), 10

`call()` (`raytraverse.renderer.Renderer` class method), 10

`call()` (`raytraverse.renderer.SPRenderer` class method), 11

`call_sampler()` (in module `raytraverse.io`), 22

`candidates()` (`raytraverse.scene.SunSetterPositions` property), 5

`CaptureStdOut` (class in `raytraverse.io`), 22

`carray2hdr()` (in module `raytraverse.io`), 23

`check_viz()` (`raytraverse.sampler.SunViewSampler` method), 15

`choose_suns()` (`raytraverse.scene.SunSetter` method), 4

`choose_suns()` (`raytraverse.scene.SunSetterLoc` method), 5

`choose_suns()` (`raytraverse.scene.SunSetterPositions` method), 5

`chord2theta()` (in module `raytraverse.translate`), 28

`cleanup` (`raytraverse.renderer.SPRenderer` attribute), 11

`coeff_lum_perez()` (in module `raytraverse.skycalc`), 26

`colormap()` (in module `raytraverse.plot`), 24

`compiledscene()` (`raytraverse.sampler.Sampler` property), 12

`constructors()` (`raytraverse.lightfield.memarraydict.MemArrayDict` method), 19

`ctheta()` (`raytraverse.mapper.ViewMapper` method), 9

## D

`d_kd()` (`raytraverse.lightfield.LightFieldKD` property), 17

`datetime64_2_datetime()` (in module `raytraverse.skycalc`), 25

`degrees()` (`raytraverse.mapper.ViewMapper` method), 9

`direct_view()` (`raytraverse.lightfield.LightFieldKD` method), 17

`direct_view()` (`raytraverse.lightfield.SunField` method), 18

`direct_view()` (`raytraverse.lightfield.SunViewField` method), 19

`direct_view()` (`raytraverse.scene.SunSetter` method), 4

`drain_bytes()` (`raytraverse.io.CaptureStdOut` method), 22

`drain_str()` (`raytraverse.io.CaptureStdOut` method), 22

`draw()` (`raytraverse.sampler.Sampler` method), 13

`draw()` (`raytraverse.sampler.SingleSunSampler` method), 15

`draw()` (`raytraverse.sampler.SunViewSampler` method), 16

`dump_vecs()` (`raytraverse.sampler.Sampler` method), 13

`dxyz()` (`raytraverse.mapper.ViewMapper` property), 9

## E

`Engine` (`raytraverse.renderer.Rcontrib` attribute), 10

`Engine` (`raytraverse.renderer.Renderer` attribute), 9

`Engine` (`raytraverse.renderer.Rtrace` attribute), 10

`Engine` (`raytraverse.renderer.SPRcontrib` attribute), 11

`Engine` (`raytraverse.renderer.SPRtrace` attribute), 11

`engine` (`raytraverse.sampler.Sampler` attribute), 12

`EngineClass` (`raytraverse.sampler.SingleSunSampler` attribute), 14

`EngineClass` (`raytraverse.sampler.SunViewSampler` attribute), 15

## F

`format_skydata()` (`raytraverse.integrator.Integrator` method), 20

`from_pdf()` (in module `raytraverse.draw`), 22

`full_array()` (`raytraverse.lightfield.memarraydict.MemArrayDict` method), 19

`full_constructor()` (`raytraverse.lightfield.memarraydict.MemArrayDict` method), 19

## G

`generate_wea()` (in module `raytraverse.skycalc`), 26

`get_detail()` (in module `raytraverse.draw`), 22

`get_instance()` (`raytraverse.renderer.SPRenderer` class method), 11

`get_loc_epw()` (in module `raytraverse.skycalc`), 25

`get_scheme()` (`raytraverse.sampler.Sampler` method), 13

`get_sky_mtx()` (`raytraverse.integrator.Integrator` method), 20

## H

`hdr()` (`raytraverse.integrator.Integrator` method), 21

`hdr2array()` (in module `raytraverse.io`), 24

`header` (`raytraverse.renderer.Renderer` attribute), 10

`header()` (`raytraverse.integrator.Integrator` method), 21

`hist()` (in module `raytraverse.quickplot`), 24

## I

`idres` (`raytraverse.sampler.Sampler` attribute), 12



`idx()` (*raytraverse.sampler.Sampler* property), 12  
`idx2pt()` (*raytraverse.mapper.SpaceMapper* method), 7  
`idx2pt()` (*raytraverse.mapper.SpaceMapperPt* method), 8  
`illum()` (in module *raytraverse.metric*), 24  
`imshow()` (in module *raytraverse.plot*), 24  
`imshow()` (in module *raytraverse.quickplot*), 24  
`in_area()` (*raytraverse.mapper.SpaceMapper* method), 7  
`in_area()` (*raytraverse.mapper.SpaceMapperPt* method), 8  
`in_solarbounds()` (*raytraverse.scene.SkyInfo* method), 6  
`in_view()` (*raytraverse.mapper.ViewMapper* method), 9  
`index_strides()` (*raytraverse.lightfield.memarraydict.MemArrayDict* method), 19  
`initialize()` (*raytraverse.renderer.RadianceRenderer* class method), 10  
`initialize()` (*raytraverse.renderer.Renderer* class method), 10  
`initialize()` (*raytraverse.renderer.SPRenderer* class method), 11  
`initialized` (*raytraverse.renderer.Renderer* attribute), 9  
`instance` (*raytraverse.renderer.Renderer* attribute), 9  
`Integrator` (class in *raytraverse.integrator*), 19  
`interpolate2d()` (in module *raytraverse.translate*), 28  
`items()` (*raytraverse.lightfield.LightField* method), 16  
`items()` (*raytraverse.lightfield.SunField* method), 18  
`items()` (*raytraverse.lightfield.SunViewField* method), 19  
`ivm()` (*raytraverse.mapper.ViewMapper* property), 9

## K

`keymap()` (*raytraverse.lightfield.SunField* method), 18

## L

`levels()` (*raytraverse.sampler.Sampler* property), 12  
`levels()` (*raytraverse.sampler.SunViewSampler* property), 15  
`LightField` (class in *raytraverse.lightfield*), 16  
`LightFieldKD` (class in *raytraverse.lightfield*), 17  
`load_scene()` (*raytraverse.renderer.SPRenderer* class method), 11  
`load_sky_facs()` (*raytraverse.scene.SunSetter* method), 4  
`load_source()` (*raytraverse.renderer.Rtrace* class method), 10  
`loc()` (*raytraverse.scene.SkyInfo* property), 6

`lum()` (*raytraverse.lightfield.LightField* property), 16

## M

`maxspec` (*raytraverse.scene.Scene* attribute), 3  
`MemArrayDict` (class in *raytraverse.lightfield.memarraydict*), 19  
`metric()` (*raytraverse.integrator.Integrator* method), 21  
`metric()` (*raytraverse.lightfield.SunViewField* method), 19  
`mk_img_setup()` (in module *raytraverse.plot*), 24  
`mk_vector_ball()` (*raytraverse.lightfield.LightFieldKD* static method), 17  
`module`  
     *raytraverse.draw*, 22  
     *raytraverse.io*, 22  
     *raytraverse.metric*, 24  
     *raytraverse.plot*, 24  
     *raytraverse.quickplot*, 24  
     *raytraverse.skycalc*, 25  
     *raytraverse.translate*, 27

## N

`name` (*raytraverse.renderer.Rcontrib* attribute), 10  
`name` (*raytraverse.renderer.Renderer* attribute), 9  
`name` (*raytraverse.renderer.Rtrace* attribute), 10  
`name` (*raytraverse.renderer.SPRenderer* attribute), 11  
`name` (*raytraverse.renderer.SPRenderer* attribute), 11  
`new_amb()` (*raytraverse.renderer.Rtrace* class method), 10  
`norm()` (in module *raytraverse.translate*), 27  
`norm1()` (in module *raytraverse.translate*), 27  
`np2bytes()` (in module *raytraverse.io*), 23  
`npts()` (*raytraverse.mapper.SpaceMapper* property), 7

## O

`omega()` (*raytraverse.lightfield.LightField* property), 16  
`OUT`  
     *raytraverse* command line option, 29  
`outdir` (*raytraverse.scene.Scene* attribute), 3  
`outfile()` (*raytraverse.lightfield.LightField* method), 16

## P

`pdf_from_sky()` (*raytraverse.sampler.SingleSunSampler* method), 15  
`perez()` (in module *raytraverse.skycalc*), 26  
`perez_apply_coef()` (in module *raytraverse.skycalc*), 26  
`perez_lum()` (in module *raytraverse.skycalc*), 26  
`perez_lum_raw()` (in module *raytraverse.skycalc*), 26  
`pixel2omega()` (*raytraverse.mapper.ViewMapper* method), 9

`pixel2ray()` (*raytraverse.mapper.ViewMapper method*), 9

`pixelrays()` (*raytraverse.mapper.ViewMapper method*), 9

`plot_patches()` (*in module raytraverse.plot*), 24

`pmtx()` (*raytraverse.mapper.ViewMapper property*), 8

`prefix` (*raytraverse.lightfield.LightField attribute*), 16

`proxy_src()` (*raytraverse.scene.SunSetter method*), 4

`pt2uv()` (*raytraverse.mapper.SpaceMapper method*), 7

`pt2uv()` (*raytraverse.mapper.SpaceMapperPt method*), 8

`pt_kd()` (*raytraverse.mapper.SpaceMapper property*), 6

`ptres` (*raytraverse.mapper.SpaceMapper attribute*), 6

`pts()` (*raytraverse.mapper.SpaceMapper method*), 7

`pts()` (*raytraverse.mapper.SpaceMapperPt method*), 8

`pts()` (*raytraverse.scene.Scene method*), 3

`ptshape()` (*raytraverse.mapper.SpaceMapper property*), 7

`ptshape()` (*raytraverse.mapper.SpaceMapperPt property*), 8

`pxyz2xyz()` (*in module raytraverse.translate*), 27

## Q

`query_all_pts()` (*raytraverse.lightfield.LightFieldKD method*), 17

`query_ball()` (*raytraverse.lightfield.LightFieldKD method*), 17

`query_ray()` (*raytraverse.lightfield.LightFieldKD method*), 17

## R

`RadianceRenderer` (*class in raytraverse.renderer*), 10

`radians()` (*raytraverse.mapper.ViewMapper method*), 9

`raster()` (*raytraverse.lightfield.SunViewField property*), 18

`raw_files()` (*raytraverse.lightfield.LightField method*), 16

`raw_files()` (*raytraverse.lightfield.LightFieldKD method*), 17

`raw_files()` (*raytraverse.lightfield.SunField method*), 18

`raw_files()` (*raytraverse.lightfield.SunViewField method*), 18

`ray2pixel()` (*raytraverse.mapper.ViewMapper method*), 9

`raytraverse` command line option

- `--config <PATH>`, 29
- `--debug`, 30
- `--no-template`, 29
- `--opts`, 30
- `--template`, 29
- `--version`, 30
- `-c`, 29
- `-opts`, 30
- `OUT`, 29

`raytraverse.draw` module, 22

`raytraverse.io` module, 22

`raytraverse.metric` module, 24

`raytraverse.plot` module, 24

`raytraverse.quickplot` module, 24

`raytraverse.skycalc` module, 25

`raytraverse.translate` module, 27

`raytraverse-integrate` command line option

- `--debug`, 38
- `--hdr`, 38
- `--header`, 38
- `--metric`, 38
- `--no-hdr`, 38
- `--no-header`, 38
- `--no-metric`, 38
- `--no-skyonly`, 38
- `--opts`, 38
- `--skyonly`, 38
- `--version`, 38
- `-interp <INTEGER>`, 37
- `-loc <FLOATS>`, 37
- `-opts`, 38
- `-pts <TEXT>`, 37
- `-res <INTEGER>`, 37
- `-skyro <FLOAT>`, 37
- `-vname <TEXT>`, 37
- `-wea <TEXT>`, 37

`raytraverse-scene` command line option

- `--debug`, 31
- `--info`, 31
- `--no-info`, 31
- `--no-overwrite`, 31
- `--no-points`, 31
- `--no-reload`, 31
- `--opts`, 31
- `--overwrite`, 31
- `--points`, 31
- `--reload`, 31
- `--version`, 31
- `-area <TEXT>`, 30
- `-maxspec <FLOAT>`, 30
- `-opts`, 31
- `-ptres <FLOAT>`, 30
- `-scene <TEXT>`, 30

-skyres <FLOAT>, 31

raytraverse-sky command line option

- debug, 33
- no-plotdview, 32
- no-plotp, 32
- no-rmraw, 32
- no-run, 32
- opts, 33
- plotdview, 32
- plotp, 32
- rmraw, 32
- run, 32
- version, 33
- accuracy <FLOAT>, 32
- fdres <INTEGER>, 32
- idres <INTEGER>, 32
- opts, 33
- rcopts <TEXT>, 32

raytraverse-sunrun command line option

- ambcache, 35
- debug, 36
- keepamb, 36
- no-ambcache, 35
- no-keepamb, 36
- no-plotdview, 36
- no-plotp, 36
- no-reflection, 35
- no-rmraw, 35
- no-run, 35
- no-view, 36
- opts, 36
- plotdview, 36
- plotp, 36
- reflection, 35
- rmraw, 35
- run, 35
- version, 36
- view, 36
- accuracy <FLOAT>, 35
- fdres <INTEGER>, 35
- idres <INTEGER>, 35
- opts, 36
- rcopts <TEXT>, 35
- speclevel <INTEGER>, 35

raytraverse-suns command line option

- debug, 34
- no-plotdview, 34
- no-reload, 34
- no-usepositions, 34
- opts, 34
- plotdview, 34
- reload, 34
- usepositions, 34
- version, 34
- loc <FLOATS>, 33
- opts, 34
- skyro <FLOAT>, 33
- srct <FLOAT>, 33
- sunres <FLOAT>, 33
- wea <TEXT>, 33

Rcontrib (*class in raytraverse.renderer*), 10

read\_epw() (*in module raytraverse.skycalc*), 25

rebuild (*raytraverse.lightfield.LightField attribute*), 16

reflsampler (*raytraverse.sampler.SunSampler attribute*), 14

reload (*raytraverse.scene.Scene attribute*), 3

Renderer (*class in raytraverse.renderer*), 9

resample() (*in module raytraverse.translate*), 28

reset() (*raytraverse.renderer.RadianceRenderer class method*), 10

reset() (*raytraverse.renderer.Renderer class method*), 10

reset\_instance() (*raytraverse.renderer.RadianceRenderer class method*), 10

reset\_instance() (*raytraverse.renderer.Renderer class method*), 10

returnbytes (*raytraverse.renderer.RadianceRenderer attribute*), 10

rgb2lum() (*in module raytraverse.io*), 24

rmtx\_elem() (*in module raytraverse.translate*), 28

rmtx\_yp() (*in module raytraverse.translate*), 28

rotate\_elem() (*in module raytraverse.translate*), 28

rotation (*raytraverse.mapper.SpaceMapper attribute*), 6

row\_2\_datetime64() (*in module raytraverse.skycalc*), 25

Rtrace (*class in raytraverse.renderer*), 10

run() (*raytraverse.sampler.Sampler method*), 13

run() (*raytraverse.sampler.SunSampler method*), 14

run\_callback() (*raytraverse.sampler.Sampler method*), 13

run\_callback() (*raytraverse.sampler.SunViewSampler method*), 16

## S

sample() (*raytraverse.sampler.Sampler method*), 13

sample() (*raytraverse.sampler.SCBinSampler method*), 13

sample() (*raytraverse.sampler.SingleSunSampler method*), 15

sample() (*raytraverse.sampler.SunViewSampler method*), 16

sample\_idx() (*raytraverse.sampler.Sampler method*), 13

sampleargs (*raytraverse.sampler.SunSampler attribute*), 14

samplemap (*raytraverse.sampler.Sampler attribute*), 12

Sampler (*class in raytraverse.sampler*), 11

`save_img()` (in module `raytraverse.plot`), 24  
`SCBinField` (class in `raytraverse.lightfield`), 17  
`SCBinSampler` (class in `raytraverse.sampler`), 13  
`Scene` (class in `raytraverse.scene`), 2  
`scene` (`raytraverse.integrator.Integrator` attribute), 20  
`scene` (`raytraverse.renderer.SPRenderer` attribute), 11  
`scene` (`raytraverse.scene.SunSetter` attribute), 4  
`scene` (`raytraverse.scene.SunSetterPositions` attribute), 5  
`scene()` (`raytraverse.lightfield.LightFieldKD` property), 17  
`scene()` (`raytraverse.lightfield.SunViewField` property), 18  
`scene()` (`raytraverse.sampler.Sampler` property), 12  
`scene()` (`raytraverse.scene.Scene` property), 3  
`set_ang_ticks()` (in module `raytraverse.plot`), 24  
`sf()` (`raytraverse.mapper.SpaceMapper` property), 7  
`sf()` (`raytraverse.mapper.SpaceMapperPt` property), 8  
`sf()` (`raytraverse.mapper.ViewMapper` property), 9  
`SingleSunSampler` (class in `raytraverse.sampler`), 14  
`sky` (`raytraverse.integrator.Integrator` attribute), 20  
`sky` (`raytraverse.scene.SunSetterLoc` attribute), 5  
`sky_mtx()` (in module `raytraverse.skycalc`), 27  
`skydata()` (`raytraverse.integrator.Integrator` property), 20  
`skyfield` (`raytraverse.integrator.Integrator` attribute), 20  
`SkyInfo` (class in `raytraverse.scene`), 6  
`skyres()` (`raytraverse.scene.Scene` property), 3  
`skyro` (`raytraverse.scene.SkyInfo` attribute), 6  
`skyro` (`raytraverse.scene.SunSetter` attribute), 4  
`skyro` (`raytraverse.scene.SunSetterPositions` attribute), 5  
`slimit` (`raytraverse.sampler.SingleSunSampler` attribute), 15  
`solarbounds()` (`raytraverse.scene.SkyInfo` property), 6  
`SpaceMapper` (class in `raytraverse.mapper`), 6  
`SpaceMapperPt` (class in `raytraverse.mapper`), 8  
`specidx` (`raytraverse.sampler.SingleSunSampler` attribute), 15  
`SPRcontrib` (class in `raytraverse.renderer`), 11  
`SPRenderer` (class in `raytraverse.renderer`), 11  
`SPRtrace` (class in `raytraverse.renderer`), 11  
`sqlum()` (in module `raytraverse.metric`), 24  
`srcn` (`raytraverse.sampler.Sampler` attribute), 12  
`srct` (`raytraverse.scene.SunSetter` attribute), 4  
`stdout()` (`raytraverse.io.CaptureStdOut` property), 22  
`stype` (`raytraverse.sampler.Sampler` attribute), 12  
`sun_kd()` (`raytraverse.scene.SunSetter` property), 4  
`SunField` (class in `raytraverse.lightfield`), 18  
`sunfield` (`raytraverse.integrator.Integrator` attribute), 20  
`sunmap` (`raytraverse.lightfield.SunViewField` attribute), 18

`SunMapper` (class in `raytraverse.mapper`), 9  
`sunpos` (`raytraverse.sampler.SingleSunSampler` attribute), 15  
`sunpos_degrees()` (in module `raytraverse.skycalc`), 25  
`sunpos_radians()` (in module `raytraverse.skycalc`), 25  
`sunpos_utc()` (in module `raytraverse.skycalc`), 25  
`sunpos_xyz()` (in module `raytraverse.skycalc`), 26  
`sunres()` (`raytraverse.scene.SunSetter` property), 4  
`suns` (`raytraverse.integrator.Integrator` attribute), 20  
`suns` (`raytraverse.lightfield.SunField` attribute), 18  
`suns` (`raytraverse.lightfield.SunViewField` attribute), 18  
`suns()` (`raytraverse.scene.SunSetter` property), 4  
`SunSampler` (class in `raytraverse.sampler`), 14  
`SunSetter` (class in `raytraverse.scene`), 3  
`SunSetterLoc` (class in `raytraverse.scene`), 5  
`SunSetterPositions` (class in `raytraverse.scene`), 5  
`SunViewField` (class in `raytraverse.lightfield`), 18  
`SunViewSampler` (class in `raytraverse.sampler`), 15

## T

`theta2chord()` (in module `raytraverse.translate`), 28  
`tolerance` (`raytraverse.mapper.SpaceMapper` attribute), 6  
`tp2uv()` (in module `raytraverse.translate`), 28  
`tp2xyz()` (in module `raytraverse.translate`), 27  
`tpnorm()` (in module `raytraverse.translate`), 27

## U

`update_ospec()` (`raytraverse.renderer.Rtrace` class method), 10  
`update_param()` (`raytraverse.renderer.RadianceRenderer` class method), 10  
`update_pdf()` (`raytraverse.sampler.Sampler` method), 13  
`uv2bin()` (in module `raytraverse.translate`), 28  
`uv2ij()` (in module `raytraverse.translate`), 28  
`uv2pt()` (`raytraverse.mapper.SpaceMapper` method), 7  
`uv2pt()` (`raytraverse.mapper.SpaceMapperPt` method), 8  
`uv2tp()` (in module `raytraverse.translate`), 28  
`uv2xy()` (in module `raytraverse.translate`), 27  
`uv2xyz()` (in module `raytraverse.translate`), 27  
`uv2xyz()` (`raytraverse.mapper.ViewMapper` method), 9

## V

`values()` (`raytraverse.lightfield.memarraydict.MemArrayDict` method), 19  
`vec()` (`raytraverse.lightfield.LightField` property), 16  
`view` (`raytraverse.lightfield.SunField` attribute), 18  
`view` (`raytraverse.scene.Scene` attribute), 3

`view2world()` (*raytraverse.mapper.ViewMapper method*), [9](#)  
`viewangle()` (*raytraverse.mapper.ViewMapper property*), [8](#)  
`ViewMapper` (*class in raytraverse.mapper*), [8](#)  
`viewsampler` (*raytraverse.sampler.SunSampler attribute*), [14](#)

## W

`weights` (*raytraverse.sampler.Sampler attribute*), [12](#)  
`world2view()` (*raytraverse.mapper.ViewMapper method*), [9](#)  
`write_sun()` (*raytraverse.scene.SunSetter method*), [4](#)

## X

`xyz2aa()` (*in module raytraverse.translate*), [29](#)  
`xyz2tp()` (*in module raytraverse.translate*), [28](#)  
`xyz2uv()` (*in module raytraverse.translate*), [27](#)  
`xyz2uv()` (*raytraverse.mapper.ViewMapper method*), [9](#)  
`xyz2xy()` (*in module raytraverse.translate*), [27](#)  
`xyz2xy()` (*raytraverse.mapper.ViewMapper method*), [9](#)

## Y

`ymtx()` (*raytraverse.mapper.ViewMapper property*), [8](#)