raytraverse Documentation

Release 1.1.2

Stephen Wasilewski

Feb 19, 2021

CONTENTS

raytraverse is a complete workflow for climate based daylight modelling, simulation, and evaluation of architectural spaces. Built around a wavelet guided 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/en/latest/.

CHAPTER

ONE

INSTALLATION

The easiest way to install raytraverse is with pip:

```
pip install --upgrade pip setuptools wheel
pip install raytraverse
```

or if you have cloned this repository:

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

note that on first run the skycalc module may download some auxilary data which could take a minute, after that first run start-up is much faster.

CHAPTER

TWO

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 docs/requirements.txt
make docs
```

to generate local documentation.

CHAPTER

THREE

API

3.1 raytraverse.scene

3.1.1 BaseScene

class raytraverse.scene.**BaseScene**(*outdir*, *scene=None*, *frozen=True*, *formatter=None*, *reload=True*, *overwrite=False*, *log=True*)

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
- **frozen** (*bool, optional*) – create a frozen octree
- **formatter** (raytraverse.formatter.Formatter, *optional*) – intended renderer format
- **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
- **log** (*bool, optional*) – log progress events to outdir/log.txt

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

log(*instance*, *message*, *err=False*)

3.1.2 Scene

class raytraverse.scene.**Scene**(*outdir*, *scene=None*, *frozen=True*, *formatter=<class 'raytraverse.formatter.radianceformatter.RadianceFormatter'>*, ***kwargs*)

Bases: raytraverse.scene.basescene.BaseScene

container for radiance scene description

Parameters

- **outdir** (*str*) – path to store scene info and output files
- **formatter** (raytraverse.formatter.RadianceFormatter, *optional*) – intended renderer format

3.1.3 ImageScene

class raytraverse.scene.**ImageScene**(*outdir*, *scene=None*, *formatter=<class 'raytraverse.formatter.formatter.Formatter'>*, *reload=True*, *log=False*)

Bases: raytraverse.scene.basescene.BaseScene

scene for image sampling

Parameters

- **outdir** (*str*) – path to store scene info and output files
- **scene** (*str, optional*) – image file (hdr format -vta projection)

3.2 raytraverse.mapper

3.2.1 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*)

pixels(*res*)

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

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

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*)

init_img(*res=512*, *pt=0, 0, 0*)
Initialize an Image array with vectors and mask

Parameters

- **res** (*int, optional*) – image array resolution
- **pt** (*tuple, optional*) – view point for image header

Returns

- **img** (*np.array*) – zero array of shape (res*self.aspect, res)
- **vecs** (*np.array*) – direction vectors corresponding to each pixel (img.size, 3)
- **mask** (*np.array*) – indices of flattened img that are in view
- **mask2** (*np.array None*) –

  if ViewMapper is 360 degree, include mask for opposite view to use:

```
add_to_img(img, vecs[mask], mask)
add_to_img(img[res:], vecs[res:][mask2], mask2
```

- **header** (*str*)

3.2.2 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*)

3.2.3 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

3.3 raytraverse.formatter

3.3.1 Formatter

class raytraverse.formatter.**Formatter**
Bases: object

scene formatter readies scene files for simulation, must be compatible with desired renderer.

comment = '#'
line comment character

direct_args = ''
arguments for direct trace

scene_ext = ''
extension for renderer scene file

static make_scene(*scene_files*, *out*, *frozen=True*)
compile scene

static add_source(*scene*, *src*)
add source files to compiled scene

static get_skydef(*color*, *ground=True*, *name='skyglow'*)
assemble sky definition

static get_sundef(*vec*, *color*, *size=0.5333*, *mat_name='solar'*, *mat_id='sun'*, *glow=False*)
assemble sun definition

static get_contribution_args(*render_args*, *side*, *name*)
prepare arguments for contribution based simulation

static get_standard_args(*render_args*, *ambfile=None*)
prepare arguments for standard simulations

static extract_sources(*srcdef*, *accuracy*)
scan scene file for sun source definitions

3.3.2 RadianceFormatter

class raytraverse.formatter.**RadianceFormatter**
Bases: raytraverse.formatter.formatter.Formatter

scene formatter readies scene files for simulation, must be compatible with desired renderer.

comment = '#'
line comment character

scene_ext = '.oct'
extension for renderer scene file

static make_scene(*scene_files*, *out*, *frozen=True*)
compile scene

static add_source(*scene*, *src*)
add source files to compiled scene

static get_skydef(*color*, *ground=True*, *name='skyglow'*)
assemble sky definition

static get_sundef(*vec*, *color*, *size=0.5333*, *mat_name='solar'*, *mat_id='sun'*)
assemble sun definition

static extract_sources(*srcdef*, *accuracy*)
scan scene file for sun source definitions

3.4 raytraverse.renderer

3.4.1 Renderer

class raytraverse.renderer.**Renderer**
Bases: object

virtual singleton renderer class. the Renderer is implemented as a singleton as specific subclasses (rtrace, rcontrib) have many global variables set at import time. This ensures the python object is connected to the current state of the engine c++-class.

All renderer classes are callable with with a numpy array of shape (N,6) representing the origin and direction of ray samples to calculate.

args = None

_pyinstance = None

instance = <raytraverse.renderer.renderer._VirtEngine object>

scene = None

3.4.2 RadianceRenderer

class raytraverse.renderer.**RadianceRenderer**(*rayargs=None*, *scene=None*, *nproc=None*, *default_args=True*)
Bases: raytraverse.renderer.renderer.Renderer

Virtual class for wrapping c++ Radiance renderer executable classes

Do not use directly, either subclass or use existing: Rtrace, Rcontrib

name = 'radiance_virtual'

engine = <MagicMock id='140334625906512'>
raytraverse.crenderer.cRtrace

srcn = 1

defaultargs = ''

args = ''

classmethod get_default_args()

classmethod reset()
reset engine instance and unset associated attributees

classmethod set_args(*args*, *nproc=None*)
prepare arguments to call engine instance initialization

Parameters

- **args** (*str*) – rendering options
- **nproc** (*int, optional*) – cpu limit

classmethod load_scene(*scene*)
load octree file to engine instance

Parameters **scene** (*str*) – path to octree file

Raises **ValueError:** – can only be called after set_args, otherwise engine instance will abort.

3.4.3 Rtrace

class raytraverse.renderer.**Rtrace**(*rayargs=None*, *scene=None*, *nproc=None*, *default_args=True*, *direct=False*)

Bases: raytraverse.renderer.radiancerenderer.RadianceRenderer

singleton wrapper for c++ raytrraverse.crenderer.cRtrace class

this class sets default arguments, helps with initialization and setting cpu limits of the cRtrace instance. see raytrraverse.crenderer.cRtrace for more details.

Parameters

- **rayargs** (*str, optional*) – argument string (options and flags only) raises ValueError if arguments are not recognized by cRtrace.
- **scene** (*str, optional*) – path to octree
- **nproc** (*int, optional*) – if None, sets nproc to cpu count, or the RAYTRAVERSE_PROC_CAP environment variable
- **default_args** (*bool, optional*) – if True, prepend default args to rayargs parameter
- **direct** (*bool, optional*) – if True use Rtrace.directargs in place of default (also if True, sets default_args to True.

Examples

Basic Initialization and call:

```
r = renderer.Rtrace(args, scene)
ans = r(vecs)
# ans.shape -> (vecs.shape[0], 1)
```

name = 'rtrace'

engine = <MagicMock id='140334625887760'>
raytraverse.crenderer.cRtrace

defaultargs = '-av 0 0 0 -aa 0 -ab 7 -ad 128 -as 0 -c 10 -as 0 -lw 1e-5'

directargs = '-av 0 0 0 -ab 0 -lr 0'

usedirect = False

classmethod get_default_args()
return default arguments of the class

classmethod update_ospec(*vs*)
set output of cRtrace instance

Parameters **vs** (*str*) –

output specifiers for rtrace:: o origin (input) d direction (normalized) v value (radiance) V contribution (radiance) w weight W color coefficient l effective length of ray L first intersection distance c local (u,v) coordinates p point of intersection n normal at intersection (perturbed) N normal at intersection (unperturbed) r mirrored value contribution x unmirrored value contribution R mirrored ray length X unmirrored ray length

Returns **outcnt** – the number of output columns to expect when calling rtrace instance

Return type int

Raises **ValueError:** – when an output specifier is not recognized

classmethod load_source(*srcname*, *freesrc=- 1*)
add a source description to the loaded scene

Parameters

- **srcname** (*str*) – path to radiance scene file containing sources, these should not change the bounding box of the octree and has only been tested with the "source" type.
- **freesrc** (*int, optional*) – the number of objects to unload from the end of the rtrace object list, if -1 unloads all objects loaded by previous calls to load_source

3.4.4 Rcontrib

class raytraverse.renderer.**Rcontrib**(*rayargs=None*, *scene=None*, *nproc=None*, *skyres=10.0*, *modname='skyglow'*, *ground=True*, *default_args=True*)
Bases: raytraverse.renderer.radiancerenderer.RadianceRenderer

singleton wrapper for c++ raytrraverse.crenderer.cRcontrib class

this class sets default arguments, helps with initialization and setting cpu limits of the cRcontrib instance. see raytrraverse.crenderer.cRcontrib for more details.

Parameters

- **rayargs** (*str, optional*) – argument string (options and flags only) raises ValueError if arguments are not recognized by cRtrace.
- **scene** (*str, optional*) – path to octree
- **nproc** (*int, optional*) – if None, sets nproc to cpu count, or the RAYTRAVERSE_PROC_CAP environment variable
- **skyres** (*float, optional*) – approximate resolution for skypatch subdivision (in degrees). Patches will have (rounded) size skyres x skyres. So if skyres=10, each patch will be 100 sq. degrees (0.03046174197 steradians) and there will be 18 * 18 = 324 sky patches.
- **modname** (*str, optional*) – passed the -m option of cRcontrib initialization
- **ground** (*bool, optional*) – if True include a ground source (included as a final bin)
- **default_args** (*bool, optional*) – if True, prepend default args to rayargs parameter

Examples

Basic Initialization and call:

```
r = renderer.Rcontrib(args, scene)
ans = r(vecs)
# ans.shape -> (vecs.shape[0], 325)
```

name = 'rcontrib'

engine = <MagicMock id='140334625472208'>
raytraverse.crenderer.cRcontrib

ground = True

side = 18

srcn = 325

modname = 'skyglow'

classmethod setup(*scene=None*, *ground=True*, *modname='skyglow'*, *skyres=10.0*)
set class attributes for proper argument initialization

Parameters

- **scene** (*str, optional*) – path to octree
- **ground** (*bool, optional*) – if True include a ground source (included as a final bin)
- **modname** (*str, optional*) – passed the -m option of cRcontrib initialization
- **skyres** (*float, optional*) – approximate resolution for skypatch subdivision (in degrees). Patches will have (rounded) size skyres x skyres. So if skyres=10, each patch will be 100 sq. degrees (0.03046174197 steradians) and there will be 18 * 18 = 324 sky patches.

Returns **scene** – path to scene with added sky definition

Return type str

classmethod get_default_args()
construct default arguments

classmethod set_args(*args*, *nproc=None*)
prepare arguments to call engine instance initialization

Parameters

- **args** (*str*) – rendering options
- **nproc** (*int, optional*) – cpu limit

3.4.5 ImageRenderer

class raytraverse.renderer.**ImageRenderer**(*scene*, *viewmapper=None*, *method='linear'*)
Bases: raytraverse.renderer.renderer.Renderer

interface to treat image data as the source for ray tracing results

not implemented as a singleton, so multiple instances can exist in parallel.

Parameters

- **scene** (*str*) – path to hdr image file with projecting matching ViewMapper
- **viewmapper** (raytraverse.mapper.ViewMapper, *optional*) – if None, assumes 180 degree angular fisheye (vta)
- **method** (*str, optional*) – passed to scipy.interpolate.RegularGridInterpolator

3.5 raytraverse.sky

3.5.1 skycalc

functions for loading sky data and computing sun position

raytraverse.sky.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.sky.skycalc.**read_epw_full**(*epw*, *columns=None*)

Parameters

- **epw** –
- **columns** (*list, optional*) – integer indices or keys of columns to return

Returns

Return type requested columns from epw as np.array shape (8760, N)

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

raytraverse.sky.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.sky.skycalc.**row_2_datetime64**(*ts*, *year=2020*)

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

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

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

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

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

raytraverse.sky.skycalc.**scale_efficacy**(*dirdif*, *sunz*, *csunz*, *skybright*, *catn*, *td=10.9735311509*)

raytraverse.sky.skycalc.**perez**(*sxyz*, *dirdif*, *md=None*, *ground_fac=0.2*, *td=10.9735311509*)
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^2
- **md** (*np.array, optional*) – (N, 2) month day of sky calcs (for more precise eccentricity calc)
- **ground_fac** (*float*) – scaling factor (reflectance) for ground brightness
- **td** (*np.array, float*) – (N,) dew point temperature in C

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

Return type np.array

raytraverse.sky.skycalc.**sky_mtx**(*sxyz*, *dirdif*, *side*, *jn=4*, *ground_fac=0.2*)

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^2) (N, 2)
- **side** (*int*) – sky subdivision
- **jn** (*int*) – sky patch subdivision n = jn^2
- **ground_fac** (*float*) – scaling factor (reflecctance) for ground brightness

Returns

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

3.5.2 SkyData

class raytraverse.sky.**SkyData**(*wea*, *suns=None*, *loc=None*, *skyro=0.0*, *ground_fac=0.15*, *skyres=10.0*)

Bases: object

class to generate sky conditions

This class provides an interface to generate sky data using the perez sky model

Parameters

- **wea** (*str np.array*) – path to epw, wea, or .npy 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 convinience can be. However, self.skydata must be initialized (directly or through self.sky) before calling integrate.
- **suns** (raytraverse.sky.Suns, *optional*) –
- **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)

- **ground_fac** (*float, optional*) – ground reflectance
- **skyres** (*float, optional*) – approximate square patch size in degrees

_loc = None
location and sky rotation information

property skyres

property skyro
sky rotation (in degrees, ccw)

property loc
lot, lon, mer (in degrees, west is positive)

property smtx
shape (np.sum(daysteps), skyres**2 + 1) coefficients for each sky patch each row is a timestep, coefficients exclude sun

property sun
shape (np.sum(daysteps), 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).

property daysteps
shape (len(skydata),) boolean array masking timesteps when sun is below horizon

property proxysort
sorting indices to arange daystep axis by solar proxy this is useful when combining sky/sun kdtrees without writing to disk to only do the interpolation once for a set of sky conditions.

property invsort
reverse sorting indices to restore input daystep order

property serr
the error (in degrees) between the actual sun position and the applied sunproxy

property skydata
sun position and dirnorm diffhoriz

property sunproxy
array of sun proxy data shape (len(daysteps), 2). column 0 is the corresponding sky bin (column of smtx), column 1 is the row of self.suns

smtx_patch_sun()
generate smtx with solar energy applied to proxy patch for directly applying to skysampler data (without direct sun components can also be used in a partial mode (with sun view / without sun reflection.

header()
generate image header string

_format_skydata(*dat*)
process dat argument as skydata

see sky.setter for details on argument

Returns dx, dy, dz, dir, diff

Return type np.array

3.5.3 SolarBoundary

class raytraverse.sky.**SolarBoundary**(*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

3.5.4 Suns

class raytraverse.sky.**Suns**(*scene*, *skyro=0.0*, *reload=True*, *sunres=10.0*, *prefix='suns'*, *suns=None*, ***kwargs*)
Bases: object

select suns to sample based on sky pdf and scene.

Parameters

- **scene** (*str,*) – path of scene
- **skyro** (*float, optional*) – sky rotation (in degrees, ccw)
- **reload** (*bool*) – if True reloads existing sun positions, else always generates new
- **sunres** (*float*) –
- **prefix** (*str*) –
- **suns** (*np.array*) – shape (N, 3) to directly set suns.

skyro = None
ccw rotation (in degrees) for sky

Type float

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

property sbins
holds sun bin numbers

_jitter_suns(*si*)

choose_suns()

direct_view()

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,) index to proxy src
- *list* – (N,) error in degrees to proxy sun

3.5.5 SunsLoc

class raytraverse.sky.**SunsLoc**(*scene*, *loc*, *skyro=0.0*, ***kwargs*)
Bases: raytraverse.sky.suns.Suns

select suns to sample based on sky pdf, scene, and location.

Parameters

- **scene** (*str,*) – path of scene
- **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.sky.SolarBoundary

choose_suns()

3.5.6 SunsPos

class raytraverse.sky.**SunsPos**(*scene*, *wea*, *skyro=0.0*, ***kwargs*)

Bases: raytraverse.sky.suns.Suns

select suns to sample based on sun positions. the wea argument provides a list of sun positions to draw from rather than randomly generating the sun position like Suns and SunsLoc.

Parameters

- **sscene** (*str,*) – path of scene
- **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

skyro = None

ccw rotation (in degrees) for sky

Type float

property candidates

candidate sun positions

Getter Returns the sun source array

Setter Set the sun source array and write to files

Type np.array

choose_suns()

3.6 raytraverse.sampler

3.6.1 draw

wavelet and associated probability functions.

raytraverse.sampler.draw.**get_detail**(*samps*, *f1=None*, *f2=None*, *f3=None*)

raytraverse.sampler.draw.**from_pdf**(*pdf*, *threshold*, *lb=0.5*, *ub=4*)

3.6.2 Sampler

class raytraverse.sampler.**Sampler**(*scene*, *engine*, *idres=5*, *fdres=9*, *accuracy=1.0*, *srcn=1*, *stype='generic'*, *bands=1*, ***kwargs*)

Bases: object

wavelet based sampling class

Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry and formatter compatible with engine
- **engine** (raytraverse.renderer.Renderer) – should inherit from raytraverse.renderer.Renderer
- **idres** (*int, optional*) – initial direction resolution (as log2(res))

- **fdres** (*int, optional*) – final directional resolution given as log2(res)
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)
- **srcn** (*int, optional*) – number of sources return per vector by run
- **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()

t0 = 0.00390625
coefficients used to set the sampling thresholds

t1 = 0.0625

lb = 0.25
lower and upper bounds for drawing from pdf

ub = 8

engine = None
raytraverse.renderer.Renderer

bands = None
number of spectral bands / channels returned by renderer based on given renderopts (user ensures these agree).

Type int

scene = None
scene information

Type *raytraverse.scene.Scene*

srcn = None
number of sources return per vector by run

Type int

accuracy = None
accuracy parameter

Type float

idres = None
initial direction resolution (as log2(res))

Type int

weights = None
holds weights for self.draw

Type np.array

stype = None
sampler type

Type str

property levels
sampling scheme

Getter Returns the sampling scheme

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

Type np.array

sample(*vecs*)
call rendering engine to sample rays

Parameters **vecs** (*np.array*) – sample vectors (subclasses can choose which to use)

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

Return type np.array

_offset(*shape*, *dim*)
for modifying jitter behavior of UV direction samples

Parameters

- **shape** (*tuple*) – shape of samples to jitter/offset
- **dim** (*int*) – number of divisions in square side

sample_to_uv(*pdraws*, *shape*)
generate samples vectors from flat draw indices

Parameters

- **pdraws** (*np.array*) – flat index positions of samples to generate
- **shape** (*tuple*) – shape of level samples

Returns

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

_plot_p(*p*, *level*, *vm*, *name*, *suffix='.hdr'*, *fisheye=True*)

_plot_vecs(*vecs*, *level*, *vm*, *name*, *suffix='.hdr'*)

_linear(*x*, *x1*, *x2*)

threshold(*idx*)
threshold for determining sample count

detailfunc = 'wav3'

filters = {'cross': (<MagicMock name='mock().__truediv__()' id='140334628065936'>,

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

Returns

- **pdraws** (*np.array*) – index array of flattened samples chosen to sample at next level
- **p** (*np.array*) – computed probabilities

update_weights(*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(*point*, *posidx*, *vm*)
handle class specific cleanup and lightpointKD construction

_dump_vecs(*vecs*)

run(*point*, *posidx*, *vm=None*, *plotp=False*, *log=False*, ***kwargs*)

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **vm** (raytraverse.mapper.ViewMapper) – view direction to sample
- **plotp** – plot weights, detail and vectors for each level
- **log** – whether to log level sampling rates can be 'scene', 'err' or None 'scene' - logs to Scene log file 'err' - logs to stderr anything else - does not log incremental progress

3.6.3 SkySampler

class raytraverse.sampler.**SkySampler**(*scene*, *engine*, ***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 scene: str, optional (required if not reload) space separated list of radiance scene files (no sky) or octree
- **engine** (raytraverse.renderer.Rcontrib) – initialized rendering instance

sample(*vecs*)
call rendering engine to sample rays

Parameters **vecs** (*np.array*) – sample vectors (subclasses can choose which to use)

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

Return type np.array

3.6.4 SunSampler

class raytraverse.sampler.**SunSampler**(*scene*, *engine*, *sun*, *sunbin*, *speclevel=9*, *fdres=10*, *engine_args='-ab 7 -ad 128 -as 0 -c 10 -as 0 -lw 1e-5'*, *slimit=0.01*, *maxspec=0.3*, ***kwargs*)
Bases: raytraverse.sampler.sampler.Sampler

sample contributions from direct suns.

Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **engine** (raytraverse.renderer.Rtrace) – initialized renderer instance (with scene loaded, no sources)
- **sun** (*np.array*) – shape 3, sun position
- **sunbin** (*int*) – sun bin
- **ropts** (*str, optional*) – arguments for engine

- **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)

specidx = None
index of level at which brightness sampling occurs

Type int

sunpos = None
sun position x,y,z

Type np.array

draw(*level*)
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

run_callback(*point*, *posidx*, *vm*)
handle class specific cleanup and lightpointKD construction

_load_specguide(*point*, *posidx*, *vm*)

run(*point*, *posidx*, *vm=None*, *plotp=False*, ***kwargs*)

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **vm** (raytraverse.mapper.ViewMapper) – view direction to sample
- **plotp** – plot weights, detail and vectors for each level
- **log** – whether to log level sampling rates can be 'scene', 'err' or None 'scene' - logs to Scene log file 'err' - logs to stderr anything else - does not log incremental progress

3.6.5 SunViewSampler

class raytraverse.sampler.**SunViewSampler**(*scene*, *engine*, *sun*, *sunbin*, ***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.

ub = 1
deterministic sample draws

_offset(*shape*, *dim*)
no jitter on sun view because of very fine resolution and potentially large number of samples bog down random number generator

run_callback(*point*, *posidx*, *vm*)
post sampling, write full resolution (including interpolated values) non zero rays to result file.

run(*point*, *posidx*, *vm=None*, *plotp=False*, ***kwargs*)

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **vm** (raytraverse.mapper.ViewMapper) – view direction to sample
- **plotp** – plot weights, detail and vectors for each level
- **log** – whether to log level sampling rates can be 'scene', 'err' or None 'scene' - logs to Scene log file 'err' - logs to stderr anything else - does not log incremental progress

3.6.6 ImageSampler

class raytraverse.sampler.**ImageSampler**(*scene*, *vm=None*, *scalefac=None*, *method='linear'*, ***kwargs*)
Bases: raytraverse.sampler.sampler.Sampler

sample image (for testing algorithms).

Parameters

- **scene** (raytraverse.scene.ImageScene) – scene class containing image file information
- **scalefac** (*float, optional*) – by default set to the average of non-zero pixels in the image used to establish sampling thresholds similar to contribution based samplers

run_callback(*point*, *posidx*, *vm*)
handle class specific cleanup and lightpointKD construction

3.6.7 DeterministicImageSampler

class raytraverse.sampler.**DeterministicImageSampler**(*scene*, *vm=None*, *scalefac=None*, *method='linear'*, ***kwargs*)
Bases: raytraverse.sampler.imagesampler.ImageSampler

ub = 1

_offset(*shape*, *dim*)
for modifying jitter behavior of UV direction samples

3.7 raytraverse.lightpoint

3.7.1 LightPointKD

class raytraverse.lightpoint.**LightPointKD**(*scene*, *vec=None*, *lum=None*, *vm=None*, *pt=0, 0, 0*, *posidx=0*, *src='sky'*, *srcn=1*, *calcomega=True*, *write=True*)

Bases: object

light field with KDtree structures for spatial query

vm = None
raytraverse.mapper.ViewMapper

scene = None
raytraverse.scene.Scene

posidx = None
index for point

Type int

pt = None
point location

Type np.array

src = None
source key

Type str

file = None
relative path to disk storage

Type str

dump()

property vec
direction vector (N,3)

property lum
luminance (N,srcn)

property d_kd
kd tree for spatial query

Getter Returns kd tree structure

Type scipy.spatial.cKDTree

property omega
solid angle (N)

Getter Returns array of solid angles

Setter sets soolid angles with viewmapper

Type np.array

calc_omega(*write=True*)
calculate solid angle

Parameters **write** (*bool, optional*) – update/write kdtree data to file

apply_coef(*coefs*)

add_to_img(*img*, *vecs*, *mask=None*, *coefs=1*, *interp=False*, *omega=False*, *vm=None*)
add luminance contributions to image array (updates in place)

Parameters

- **img** (*np.array*) – 2D image array to add to (either zeros or with other source)
- **vecs** (*np.array*) – vectors corresponding to img pixels shape (N, 3)
- **mask** (*np.array*) – indices to img that correspond to vec (in case where whole image is not being updated, such as corners of fisheye)
- **coefs** (*int, float, np.array*) – source coefficients, shape is (1,) or (srcn,)
- **interp** (*bool, optional*) – for linear interpolation (falls back to nearest outside of convexhull
- **omega** (*bool*) – if true, add value of ray solid angle instead of luminance
- **vm** (*raytraverse.mapper.ViewMapper, optional*) –

Returns

Return type None

get_applied_rays(*skyvec*, *vm=None*)
the analog to add_to_img for metric calculations

query_ray(*vecs*)

query_ball(*vecs*, *viewangle=180*)

direct_view(*res=512*, *showsample=False*, *showweight=True*, *srcidx=None*, *interp=False*, *omega=False*, *scalefactor=1*)
create a summary image of lightfield for each vpt

static _build(*vec*, *lum*, *srcn*)
load samples and build data structure

3.7.2 SunPointKD

class raytraverse.lightpoint.**SunPointKD**(*scene*, *vec=None*, *lum=None*, *sun=0, 0, 0*, ***kwargs*)

Bases: raytraverse.lightpoint.lightpointkd.LightPointKD

removes stray rays from accidental direct sun hits during build

_build(*vec*, *lum*, *srcn*)
load samples and build data structure remove lucky hits of direct sun (since these are accounted for by the SunViewSampler)

3.7.3 SunViewPoint

class raytraverse.lightpoint.**SunViewPoint**(*scene*, *vecs*, *lum*, *pt=0, 0, 0*, *posidx=0*, *src='sunview'*, *res=64*, *blursun=1.0*)

Bases: object

interface for sun view data

solar_omega = 6.796702357283834e-05

static offset(*points*, *target*)

scene = None
raytraverse.scene.Scene

posidx = None
index for point

Type int

pt = None
point location

Type np.array

src = None
source key

Type str

property vm

_to_pix(*atv*, *vm*, *res*)

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

add_to_img(*img*, *vecs*, *mask=None*, *coefs=1*, *vm=None*)

get_applied_rays(*sunval*, *vm=None*)

direct_view(*res=80*)

3.8 raytraverse.evaluate

3.8.1 MetricSet

class raytraverse.evaluate.**MetricSet**(*vm*, *vec*, *omega*, *lum*, *metricset=None*, *scale=179.0*, *threshold=2000.0*, *guth=True*, *tradius=30.0*, ***kwargs*)

Bases: object

object for calculating metrics based on a view direction, and rays consisting on direction, solid angle and luminance information

by encapsulating these calculations within a class, metrics with redundant calculations can take advantage of cached results, for example dgp does not need to recalculate illuminance when it has been directly requested. all metrics can be accessed as properties (and are calculated just in time) or the object can be called (no arguments) to return a np.array of all metrics defined in "metricset"

Parameters

- **vm** (raytraverse.mapper.ViewMapper) – the view direction
- **vec** (*np.array*) – (N, 3) directions of all rays in view
- **omega** (*np.array*) – (N,) solid angle of all rays in view
- **lum** (*np.array*) – (N,) luminance of all rays in view (multiplied by "scale")
- **metricset** (*list, optional*) – keys of metrics to return, same as property names
- **scale** (*float, optional*) – scalefactor for luminance
- **threshold** (*float, optional*) – threshold for glaresource/background similar behavior to evalglare '-b' paramenter. if greater than 100 used as a fixed luminance threshold. otherwise used as a factor times the task luminance (defined by 'tradius')
- **guth** (*bool, optional*) – if True, use Guth for the upper field of view and iwata for the lower if False, use Kim
- **tradius** (*float, optional*) – radius in degrees for task luminance calculation
- **kwargs** – additional arguments that may be required by additional properties

allmetrics = ['illum', 'avglum', 'gcr', 'ugp', 'dgp', 'tasklum', 'backlum', 'dgp_t1

defaultmetrics = ['illum', 'avglum', 'gcr', 'ugp', 'dgp']
available metrics (and the default return set)

static check_metrics(*metrics*, *raise_error=False*)
returns list of valid metric names from argument if raise_error is True, raises an Atrribute Error

property vec

property lum

property omega

property ctheta
cos angle between ray and view

property radians
cos angle between ray and view

property src_mask
boolean mask for filtering source/background rays

property task_mask

property sources
vec, omega, lum of rays above threshold

property background
vec, omega, lum of rays below threshold

property source_pos_idx

property threshold
threshold for glaresource/background similar behavior to evalglare '-b' paramenter

property pwsl2
position weighted source luminance squared, used by dgp, ugr, etc sum(Ls^2*omega/Ps^2)

property srcillum
average background luminance

property backlum
average background luminance CIE estimate (official for some metrics)

property backlum_true
average background luminance mathematical

property tasklum
average task luminance

property illum
illuminance

property avglum
average luminance

property avgraylum
average luminance (not weighted by omega

property gcr
a unitless measure of relative contrast defined as the average of the squared luminances divided by the average luminance squared

property dgp

property dgp_t1

property log_gc

property dgp_t2

property ugr

property ugp

property density

property reldensity

property lumcenter

3.8.2 PositionIndex

class raytraverse.evaluate.**PositionIndex**(*guth=True*)
Bases: object

calculate position index according to guth/iwata or kim

Parameters **guth** (*bool*) – if True, use Guth for the upper field of view and iwata for the lower if False, use Kim

positions(*vm*, *vec*)
calculate position indices for a set of vectors

Parameters

- **vm** (raytraverse.mapper.ViewMapper) – the view/analysis point, should have 180 degree field of view
- **vec** (*np.array*) – shape (N,3) the view vectors to calculate

Returns **posidx** – shape (N,) the position indices

Return type np.arrray

static _to_plane(*n*, *vec*)

static _angle_vv(*a*, *b*)

static _get_pidx_guth(*sigma*, *tau*)

static _get_pidx_iwata(*phi*, *theta*)

static _get_pidx_kim(*sigma*, *tau*)

3.8.3 retina

raytraverse.evaluate.retina.**rgcf_density_on_meridian**(*deg*, *mi*)
retinal ganlgion cell field density along a meridian as a functional best fit.

the field density accounts for the input region of the ganglion cell to account for displaced ganglion cells. This value is estimate from cone density and the inferred density of midget ganglion cells. see Watson (2014) for important caveats.

Parameters

- **deg** (*np.array*) – eccentricity in degrees along merdian
- **mi** (*int*) – meridian index. [0, 1, 2, 3] for Temporal, Superior, Nasal, Inferior.

Returns 1d array of retinal ganglion cell density along a merdian

Return type np.array

raytraverse.evaluate.retina.**rgc_density_on_meridian**(*deg*, *mi*)
retinal ganglion cell density along a merdian as a linear interpolation between non-zero measurements

As opposed to the field density this estimate the actual location of ganglion cells, which could be important to consider for intrinsically photosensitive cells. These are (partially?) responsible for pupillary response. However, even iprgc (may?) receive signals from rods/cones

Parameters

- **deg** (*np.array*) – eccentricity in degrees along merdian
- **mi** (*int*) – meridian index. [0, 1, 2, 3] for Temporal, Superior, Nasal, Inferior.

Returns 1d array of retinal ganglion cell density along a merdian

Return type np.array

raytraverse.evaluate.retina.**rgcf_density_xy**(*xy*, *func=<function rgcf_density_on_meridian>*)

interpolate density between meridia, selected by quadrant

Parameters

- **xy** (*np.array*) – xy visual field coordinates on a disk in degrees (eccentricity 0-90 from fovea)
- **func** (*callable*) – density function along a meridian, takes r in degrees and an axes index: [0, 1, 2, 3] for Temporal, Superior, Nasal, Inferior.

Returns 1d array of single eye densities

Return type np.array

raytraverse.evaluate.retina.**binocular_density**(*xy*, *func=<function rgcf_density_on_meridian>*)

average denisty between both eyes.

Parameters

- **xy** (*np.array*) – xy visual field coordinates on a disk (eccentricity 0-1 from fovea)
- **func** (*callable*) – density function along a meridian, takes r in degrees and an axes index: [0, 1, 2, 3] for Temporal, Superior, Nasal, Inferior. coordinates are for the visual field.

Returns 1d array of average binocular densities

Return type np.array

raytraverse.evaluate.retina.**rgcf_density**(*xy*)

retinal ganglion cell field density

Parameters **xy** (*np.array*) – xy visual field coordinates on a disk (eccentricity 0-1 from fovea)

Returns 1d array retinal ganglion cell field density according to model by Watson

Return type np.array

raytraverse.evaluate.retina.**rgc_density**(*xy*)

retinal ganglion cell density (includes displaced ganglion cells)

Parameters **xy** (*np.array*) – xy visual field coordinates on a disk (eccentricity 0-1 from fovea)

Returns 1d array retinal ganglion cell density according to measurements by Curcio

Return type np.array

3.9 raytraverse.craytraverse

3.10 raytraverse.io

functions for reading and writing

raytraverse.io.**get_nproc**(*nproc=None*)

raytraverse.io.**set_nproc**(*nproc*)

raytraverse.io.**unset_nproc**()

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.**np2bytefile**(*ar*, *outf*, *dtype='<f'*, *mode='wb'*)
save vectors to file

Parameters

- **ar** (*np.array*) – array to write
- **outf** (*str*) – file to write to
- **dtype** (*str*) – argument to pass to np.dtype()

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.**version_header**()
generate image header string

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

Parameters

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

Returns

Return type imgf

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

Parameters

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

Returns

Return type imgf

raytraverse.io.**uvarray2hdr**(*uvarray*, *imgf*, *header=None*)

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

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

Returns **ar**

Return type np.array

raytraverse.io.**rgb2rad**(*rgb*)

raytraverse.io.**rgb2lum**(*rgb*)

raytraverse.io.**rgbe2lum**(*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^2

raytraverse.io.**add_vecs_to_img**(*vm*, *img*, *v*, *channels=1, 0, 0*, *grow=0*)

3.11 raytraverse.plot

functions for plotting data

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

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

3.12 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.**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.**xyz2skybin**(*xyz*, *side*, *tol=0*, *normalize=False*)

raytraverse.translate.**skybin2xyz**(*bn*, *side*)

raytraverse.translate.**xyz2xy**(*xyz*, *axes=0, 1, 2*, *flip=True*)
xyz coordinates to xy mapping of angular fisheye proejection

raytraverse.translate.**pxy2xyz**(*pxy*, *viewangle=180.0*)
pixel coordinates of angular fisheye to xyz

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

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.**aa2xyz**(*aa*)
calculate altitude (0-90), azimuth (-180,180) from xyz

raytraverse.translate.**xyz2aa**(*xyz*)
calculate xyz from altitude (0-90), azimuth (-180,180)

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.**uv2ij**(*uv*, *side*, *aspect=2*)

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

raytraverse.translate.**bin2uv**(*bn*, *side*, *offset=0.5*)

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.**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 transorm, swap order and transpose. Forward: pmtx@(ymtx@xyz.T)).T Backward: ymtx.T@(pmtx.T@xyz.T)).T

Return type (np.array, np.array)

CHAPTER

FOUR

LICENCE

CHAPTER

FIVE

ACKNOWLEDGEMENTS

Thanks to additional project collaborators and advisors Marilyne Andersen, Lars Grobe, Roland Schregle, Jan Wienold, and Stephen Wittkopf

This software development was financially supported by the Swiss National Science Foundation as part of the ongoing research project "Light fields in climate-based daylight modeling for spatio-temporal glare assessment" (SNSF #179067).

CHAPTER

SIX

SOFTWARE CREDITS

- Raytraverse uses Radiance
- As well as all packages listed in the requirements.txt file, raytraverse relies heavily on the Python packages numpy, scipy, and for key parts of the implementation.
- C++ bindings, including exposing core radiance functions as methods to the renderer classes are made with pybind11
- Installation and building from source uses cmake and scikit-build
- This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

6.1 History

6.1.1 1.0.4 (2020-11-18)

- create and manage log file (attribute of Scene) for run directories
- possible fix for bug in interpolate_kd resulting in index range errors
- protect imports in cli.py so documentation can be built without installing

6.1.2 1.0.3 (2020-11-10)

- new module for calculating position based on retinal features
- view specifications for directview plotting
- options for samples/weight visibility on directview plotting

6.1.3 0.2.0 (2020-09-25)

- Build now includes all radiance dependencies to setup multi-platform testing
- In the absence of craytraverse, sampler falls back to SPRenderer
- install process streamlined for developer mode
- travis ci deploys linux and mac wheels directly to pypi
- **release.sh should be run after updating this file, tests past locally and** docs build.

6.1.4 0.1.0 (2020-05-19)

- First release on PyPI.

6.2 Index

6.3 Search

6.4 Todo

6.5 Git Info

this project is hosted in two 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.

CHAPTER

SEVEN

COMMAND LINE INTERFACE

7.1 raytraverse-cli

7.1.1 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:

- ctx: click.Context
- out: path to new or existing directory for raytraverse run
- config: path to config file
- n: max number of processes to spawn

Arguments

OUT

Required argument

Options

VALUE OPTIONS:

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

-n <INTEGER>
sets the environment variable RAYTRAVERSE_PROC_CAP set to0 to clear (parallel processes will use cpu_limit)

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

suns
the suns command provides a number of options...

scene
The scene commands creates a Scene object...

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

-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 canditate suns (requires –usepositions)
7. 5 column tsv file, each row is dx, dy, dz, direct normal, diff. horizontal of canditate 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

--printsuns, --no-printsuns
print sun positions to stdout

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 canbe 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

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:

-scene <TEXT>
space separated list of radiance scene files (no sky) or precompiled octree

FLAGS (DEFAULT TRUE):

--frozen, --no-frozen
create frozen octree from scene files

Default True

--log, --no-log
log progress to <out>/log.txt

Default True

--reload, --no-reload
if a scene already exists at OUT reload it, note that ifthis is False and overwrite is False, the program willabort

Default True

FLAGS (DEFAULT FALSE):

--overwrite, --no-overwrite
Warning! if set to True and reload is False all files inOUT will be deleted

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

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y