raytraverse Documentation

Release 1.0.4

Stephen Wasilewski

Nov 18, 2020

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/stable/.

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

COMMAND LINE INTERFACE

3.1 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

scene
The scene commands creates a Scene object. . .

sky
the sky command intitializes and runs a sky. . .

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

sunrun
the sunrun command intitializes and runs a. . .

onesky
the onesky command is for running one off sky. . .

integrate
the integrate command combines sky and sun. . .

3.1.1 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 accoring to a shirley-chiu disk to square mapping, approximate square patch size in degrees. set by: int(np.floor(90/s)*2) to ensure an even number

Default 10.0

FLAGS (DEFAULT TRUE):

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

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):

--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 inOUT 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

3.1.2 sky

```
raytraverse sky [OPTIONS]
```

the sky command intitializes and runs a sky sampler and then readies 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

-dpts <TEXT>
if given points to evaluate with –plotdview, this can be a .npy file, a whitespace seperated 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)

-dviewpatch <FLOATS>
to plot direct view for a single patch,give an x,y,z direction

-fdres <INTEGER>
the final directional sampling resolution, yielding a grid of potential samples at 2^fdres x 2^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^idres, yielding 2^(2*idres) samples and a resolution of 2^(2*idres)/(2pi) 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

`--showsample, --no-showsample`
show samples on dviews

Default True

`--showweight, --no-showweight`
show weights on dviews

Default True

FLAGS (DEFAULT FALSE):

`--overwrite, --no-overwrite`
execute run even if simulations exist

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

`--rebuild, --no-rebuild`
force rebuild kdtree

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

3.1.3 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 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

--skyfilter, --no-skyfilter
use sky simulation to threshold possible solar positions (with –usepositions)

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

3.1.4 sunrun

```
raytraverse sunrun [OPTIONS]
```

the sunrun command intitializes 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

-dpts <TEXT>
if given points to evaluate with –plotdview, this can be a .npy file, a whitespace seperated 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)

-fdres <INTEGER>
the final directional sampling resolution, yielding a grid of potential samples at 2^fdres x 2^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^idres, yielding 2^(2*idres) samples and a resolution of 2^(2*idres)/(2pi) 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

--showsample, --no-showsample
show samples on dviews

Default True

--showweight, --no-showweight
show weights on dviews

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

--overwrite, --no-overwrite
execute run even if simulations exist

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

--rebuild, --no-rebuild
force rebuild kdtree

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

3.1.5 onesky

```
raytraverse onesky [OPTIONS]
```

the onesky command is for running one off sky definitions.

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

-dpts <TEXT>
if given points to evaluate with –plotdview, this can be a .npy file, a whitespace seperated 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)

-fdres <INTEGER>
the final directional sampling resolution, yielding a grid of potential samples at 2^fdres x 2^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^idres, yielding 2^(2*idres) samples and a resolution of 2^(2*idres)/(2pi) 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

-skydef <FILE>
sky scene file (.rad)

-skyname <TEXT>
basename for result files

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

--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

--showsample, --no-showsample
show samples on dviews

Default True

--showweight, --no-showweight
show weights on dviews

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

--overwrite, --no-overwrite
execute run even if simulations exist

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

--rebuild, --no-rebuild
force rebuild kdtree

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

3.1.6 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:

-blursun <INTEGER>
spread sun to mimic camera or human eye, 1 is no blur a value of 4 will double the apparent radius

Default 1

-ground_fac <FLOAT>
ground reflectance

Default 0.15

-interp <INTEGER>
number of nearby rays to use for interpolation of hdroutput (weighted by a gaussian filter). this doesnot 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)

-metricset <TEXTS>
which metrics to return items must be in: illum, avglum, lum2, ugr, dgp, tasklum, backlum, dgp_t1, dgp_t2, threshold, pwsl2, view_area, density, reldensity, lumcenter

Default illum avglum lum2 dgp ugr

-pts <TEXT>
points to evaluate, this can be a .npy file, a whitespace seperated 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

-static <TEXT>
name of static field to integrate (overrides other opts)

-threshold <FLOAT>
Threshold factor; if factor is larger than 100, it is used as constant threshold in cd/m2. If factor is less or equal than 100, this factor multiplied by the average task luminance will be used as threshold for detecting the glare sources. task luminance is taken from the centerof the view and encompasses tradius (see parameter -tradius)

Default 2000.0

-tradius <FLOAT>
task radius in degrees for calculating task luminance

Default 30.0

-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 canditate suns (requires –usepositions)
6. 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):

--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

--staterr, --no-staterr
print interpolation error info on metric output

Default False

--statidx, --no-statidx
print index columns on metric output

Default False

--statsensor, --no-statsensor
print sensor position and direction columns on metric output

Default False

--statsky, --no-statsky
print sky info (sun x,y,z dirnorm, dirdiff) on metric output

Default False

--sunonly, --no-sunonly
if True, only integrate on Sun Field, useful for diagnostics. Note: only runs if –skyonly is False

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

CHAPTER

FOUR

API

4.1 raytraverse.scene

4.1.1 Scene

class `raytraverse.scene.`**`Scene`**(*outdir*, *scene=None*, *area=None*, *reload=True*, *overwrite=False*, *ptres=1.0*, *ptro=0.0*, *pttol=1.0*, *viewdir=0, 1, 0*, *viewangle=360*, *skyres=10.0*, *maxspec=0.3*, *frozen=True*, ***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 in degrees
- **maxspec** (*float, optional*) – maximum specular transmission in scene (used to clip pdf for sun sampling)
- **frozen** (*bool, optional*) – create a frozen octree

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()

log(*instance*, *message*)

4.1.2 SunSetterBase

class raytraverse.scene.**SunSetterBase**(*scene*, *suns=None*, *prefix='suns'*, *reload=True*)
Bases: `object`

bare bones class for on the fly sunsetter.

Parameters

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry, location and analysis plane
- **suns** (*np.array*) – sun (N, 5) positions, sizes, and intensities

property suns
holds sun positions

Getter Returns the sun source array

Setter Set the sun source array and write to files

Type np.array

write_sun(*i*)

_write_suns(*sunfile*)
write suns to file

Parameters sunfile –

4.1.3 SunSetter

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

Bases: raytraverse.scene.sunsetterbase.SunSetterBase

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

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()

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

4.1.4 SunSetterLoc

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

Bases: `raytraverse.scene.sunsetter.SunSetter`

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

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()

4.1.5 SunSetterPositions

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

Bases: `raytraverse.scene.sunsetter.SunSetter`

select suns to sample based on sky pdf, scene, and sun positions. the wea argument provides a list of sun positions to draw from rather than randomly generating the sun position like SunSetter and SunSetterLoc.

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()

4.1.6 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

4.2 raytraverse.mapper

4.2.1 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*)

4.2.2 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

4.2.3 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*)

4.3 raytraverse.renderer

4.3.1 Renderer

4.3.2 RadianceRenderer

4.3.3 Rtrace

4.3.4 Rcontrib

4.3.5 SPRenderer

4.3.6 SPRtrace

4.3.7 SPRcontrib

4.4 raytraverse.sampler

4.4.1 Sampler

4.4.2 SCBinSampler

4.4.3 SunSampler

4.4.4 SingleSunSampler

4.4.5 SunViewSampler

4.4.6 SkySampler

4.5 raytraverse.lightfield

4.5.1 LightField

4.5.2 LightFieldKD

4.5.3 SCBinField

4.5.4 SunField

4.5.5 SunSkyPt

4.5.6 SunViewField

4.5.7 StaticField

4.5.8 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()

4.6 raytraverse.integrator

4.6.1 BaseIntegrator

4.6.2 Integrator

4.6.3 SunSkyIntegrator

4.6.4 MetricSet

4.6.5 PositionIndex

4.6.6 retina

4.7 raytraverse.craytraverse

4.8 raytraverse.draw

4.9 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.**get_nproc**(*nproc=None*)

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

raytraverse.io.**unset_nproc**()

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

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.**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*)

4.10 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*)

4.11 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*)

4.12 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*, *ground_fac=0.2*)
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 (reflecctance) for ground brightness

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*, *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

4.13 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 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)

`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*`)`

CHAPTER

FIVE

LICENCE

CHAPTER

SIX

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

SEVEN

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 pywavelets 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.

7.1 History

7.1.1 1.0.4

- 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

7.1.2 1.0.3

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

7.1.3 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.

7.1.4 0.1.0 (2020-05-19)

- First release on PyPI.

7.2 Index

7.3 Search

7.4 Todo

7.5 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.

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y