raytraverse Documentation

Release 1.3.4

Stephen Wasilewski

Jun 21, 2022

COMMAND LINE INTERFACE

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 .
```

1.1 Windows

Currently raytraverse is only compatible with macOS and linux operating systems. One way to use raytraverse on a Windows machine is with Docker. In adddition to the Docker installation, this process will require about 1.5 GB of disk space.

1. Install Docker from: https://www.docker.com/products/docker-desktop/ (click on "Windows") and then follow the installation instructions.
2. Open the newly installed Docker Desktop application (you do not need to sign in or create an account)
3. Create a new local folder and with a file named Dockerfile:

```
# syntax=docker/dockerfile:1
FROM python:3.9
WORKDIR /working
SHELL ["/bin/bash", "-c"]
RUN pip3 install raytraverse

CMD raytraverse --help
```

4. in a command prompt navigate to this folder and:

```
docker build . --tag raytraverse
```

5. To use raytraverse, navigate to a local folder that contains all necessary files (radiance scene files, sky data, etc.).
6. Now, in this folder:

```
docker run -it --name rayt --mount type=bind,source="$(pwd)",target=/working␣
↪raytraverse /bin/bash
```

7. You now have a linux/bash command prompt in an environment with raytraverse installed. The currrent directory will be named "working" within the linux environment and is a shared resource with the host

(changes on the host side are immediately seen in the container and vice versa). When you are finished, exit the linux shell ("exit"), then in the (now) windows command prompt:

```
docker rm rayt
```

8. for ease of use, you can put these to lines in a .bat file somewhere in your execution PATH, just make sure that docker desktop is running before calling:

```
docker run -it --name rayt --mount type=bind,source="$(pwd)",target=/working raytraverse /bin/bash
docker rm rayt
```

9. to update raytraverse, just repeat step 4 in a directory with the Dockerfile in step 3.
10. see the Docker settings for information about resource allocation to the docker container

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 craytraverse (https://pypi.org/project/craytraverse/).

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

The raytraverse command provides command line access to executing common tasks. The best way to manage all of the options is with a .cfg file. First, generate a template:

```
raytraverse --template > options.cfg
```

and then edit the options for each file. for example, here is a simplified configuration for a low accuracy sample simulation, assuming a model scaled in meters where plane.rad is betwee 4m and 10m on each side:

```
[shared]
weather_file = weather.epw

[raytraverse_scene]
out = outdir
scene = room.rad

[raytraverse_area]
ptres = 2.0
zone = plane.rad

[raytraverse_suns]
epwloc = True
loc = ${shared:weather_file}

[raytraverse_skydata]
wea = ${shared:weather_file}
skyres = 10

[raytraverse_skyengine]
accuracy = 2.0
skyres = 10

[raytraverse_sunengine]
accuracy = 2.0
nlev = 5

[raytraverse_skyrun]
accuracy = 2.0
edgemode = reflect
nlev = 2

[raytraverse_sunrun]
accuracy = 3.0
edgemode = reflect
nlev = 2
srcaccuracy = 2.0
```

(continues on next page)

(continued from previous page)

```
srcnlev = 2

[raytraverse_images]
basename = results
blursun = True
interpolate = highc
res = 800
resampleview = True
sdirs = None
sensors = None
skymask = 0:24

[raytraverse_evaluate]
basename = results
sdirs = None
sensors = None
skymask = None

[raytraverse_pull]
col = metric point
gridhdr = True
lr = results.npz
ofiles = results
skyfill = ${shared:weather_file}
viewfilter = 0
```

and then from the command line run:

```
raytraverse -c options.cfg skyrun directskyrun sunrun evaluate pull
```

3.1 raytraverse

```
raytraverse [OPTIONS] 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 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 skyrun sunrun
```

as all required precursor commands will be invoked automatically as needed.

Options

VALUE OPTIONS:

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

-n <INTEGER>
: sets the environment variable RAYTRAVERSE_PROC_CAP set to 0 to clear (parallel processes will use cpu_limit)

-out <DIRECTORY>

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
: define scene files for renderer and output…

area
: define sampling area

suns
: define solar sampling space

skydata
: define sky conditions for evaluation

skyengine
: initialize engine for skyrun

sunengine
initialize engine for sunrun

sourceengine
initialize engine for sunrun

skyrun
run scene under sky for a set of points…

directskyrun

sunrun
run scene for a set of suns (defined by…

sourcerun
run scene for a single source (or multiple…

images
render images

evaluate
evaluate metrics

pull

3.1.1 scene

```
raytraverse scene [OPTIONS]
```

define scene files for renderer and output directory

Effects

- creates outdir and outdir/scene.oct

Options

VALUE OPTIONS:

-out <DIRECTORY>

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

FLAGS (DEFAULT TRUE):

--log, --no-log
log progress to stderr

Default
True

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

Default
True

FLAGS (DEFAULT FALSE):

--overwrite, --no-overwrite
Warning! if set to True 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

3.1.2 area

```
raytraverse area [OPTIONS]
```

define sampling area

Effects

- None

Options

VALUE OPTIONS:

-jitterrate <FLOAT>
fraction of each axis to jitter over

Default
0.5

-name <TEXT>
name for zone/point group (impacts file naming)

Default
plan

-printlevel <INTEGER>
print a set of sun positions at sampling level (overrides printdata)

-ptres <FLOAT>

initial sampling resolution for points (in model units

Default
1.0

-rotation <FLOAT>

positive Z rotation for point grid alignment

Default
0.0

-static_points <TEXT>

points to simulate, 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. points should either all be 3 componnent (x,y,z) or 6 component (x,y,z,dx,dy,dz) but the dx,dy,dz is ignored

-zheight <FLOAT>

replaces z in points or zone

-zone <TEXT>

zone boundary to dynamically sample. can either be a radiance scene file defining a plane to sample or an array of points (same input options as -static_points). Pointsare used to define a convex hull with an offset of 1/2*ptres in which to sample. Note that if static_pointsand zone are both give, static_points is silently ignored

FLAGS (DEFAULT FALSE):

--printdata, --no-printdata

if True, print areamapper positions (either boundary or static points)

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]
```

define solar sampling space

Effects

- None

Options

VALUE OPTIONS:

-jitterrate <FLOAT>
fraction of each axis to jitter over

Default
0.5

-loc <TEXT>
can be a number of formats:

1. a string of 3 space seperated values (lat lon mer) where lat is +west and mer is tz*15 (matching gendaylit).
2. a string of comma seperated sun positions with multiple items seperated by spaces: "0,-.7,.7 .7,0,.7" following the shape requirements of 3.
3. a file loadable with np.loadtxt) of shape (N, 2), (N,3), (N,4), or (N,5):
   a. 2 elements: alt, azm (angles in degrees)
   b. 3 elements: dx,dy,dz of sun positions
   c. 4 elements: alt, azm, dirnorm, diffhoriz (angles in degrees)
   d. 5 elements: dx, dy, dz, dirnorm, diffhoriz.
4. path to an epw or wea formatted file: solar positions are generated and used as candidates unless –epwloc is True.
5. None (default) all possible sun positions are considered

in the case of a location, sun positions are considered valid when in the solar transit for that location. for candidate options (2., 3., 4.), sun positions are drawn from this set (with one randomly chosen from all candidates within adaptive grid.

-name <TEXT>
name for solar source group (impacts file naming)

Default
suns

-printlevel <INTEGER>
print a set of sun positions at sampling level (overrides printdata)

-skyro <FLOAT>
counterclockwise sky-rotation in degrees (equivalent to clockwise project north rotation)

Default
0.0

-sunres <INTEGER>

initial sampling resolution for suns (as sqrt of samples per hemisphere)

Default
9

FLAGS (DEFAULT FALSE):

--epwloc, --no-epwloc

if True, use location from epw/wea argument to -loc as a transit mask (like -loc option 1.) instead of as a list of candidate sun positions.

Default
False

--printdata, --no-printdata

if True, print skymapper sun positions (either boundary or candidates in xyz coordinates)

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 skydata

```
raytraverse skydata [OPTIONS]
```

define sky conditions for evaluation

Effects

- Invokes scene
- write outdir/name.npz (SkyData initialization object)

Options

VALUE OPTIONS:

-ground_fac <FLOAT>
ground reflectance

Default
0.2

-loc <FLOATS>
location data given as 'lat lon mer' with + west of prime meridian overrides location data in wea

-minalt <FLOAT>
minimum solar altitude for daylight masking

Default
2.0

-mindiff <FLOAT>
minumum diffuse horizontal irradiance for daylight masking

Default
5.0

-mindir <FLOAT>
minumum direct normal irradiance for daylight masking

Default
0.0

-name <TEXT>
output file name for skydata

Default
skydata

-skyres <INTEGER>
resolution of sky patches (sqrt(patches / hemisphere)).

Default
15

-skyro <FLOAT>
angle in degrees counter-clockwise to rotate sky (to correct model north, equivalent to clockwise rotation of scene)

Default
0.0

-wea <TEXT>
path to epw, wea, .npy file or np.array, or .npz file,if loc not set attempts to extract location data (if needed).

FLAGS (DEFAULT TRUE):

`--reload, --no-reload`

reload saved skydata if it exists in scene directory

Default
True

FLAGS (DEFAULT FALSE):

`--printdata, --no-printdata`

if True, print solar position and dirnorm/diff of loaded data

Default
False

`--printfull, --no-printfull`

with printdata, if True, print full unmasked skydata

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 skyengine

```
raytraverse skyengine [OPTIONS]
```

initialize engine for skyrun

Effects

- Invokes scene
- creates outdir/scene_sky.oct

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

`-dcompargs` <TEXT>

additional arguments for running direct component. when using, set -ab in sunengine.rayargs to this ab minus one.

Default
-ab 1

`-idres` <INTEGER>

the initial directional sampling resolution (as sqrt of samples per hemisphere)

Default
32

`-nlev` <INTEGER>

number of directional sampling levels, yielding a finalresolution of idres^2 * 2^(nlev) samples per hemisphere

Default
5

`-rayargs` <TEXT>

additional arguments to pass to the rendering engine

`-skyres` <INTEGER>

resolution of sky patches (sqrt(patches / hemisphere)).Must match argument givein to skydata

Default
15

`-vlt` <FLOAT>

primary transmitting vlt, used to scale the accuracy parameter to the expected scene variance. Optional, but helpful with, for example, electrochromic glazing or shades

Default
0.64

FLAGS (DEFAULT TRUE):

--default-args, --no-default-args
use raytraverse defaults before -rayargs, if False, uses radiance defaults

Default
True

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 sunengine

```
raytraverse sunengine [OPTIONS]
```

initialize engine for sunrun

Effects

- Invokes scene

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

-idres <INTEGER>
the initial directional sampling resolution (as sqrt of samples per hemisphere)

Default
32

-nlev <INTEGER>

number of directional sampling levels, yielding a finalresolution of idres^2 * 2^(nlev) samples per hemisphere

Default
6

-rayargs <TEXT>

additional arguments to pass to the rendering engine

-vlt <FLOAT>

primary transmitting vlt, used to scale the accuracy parameter to the expected scene variance. Optional, but helpful with, for example, electrochromic glazing or shades

Default
0.64

FLAGS (DEFAULT TRUE):

--default-args, --no-default-args

use raytraverse defaults before -rayargs, if False, uses radiance defaults

Default
True

HELP:

-opts, --opts

check parsed options

Default
False

--debug

show traceback on exceptions

Default
False

--version

Show the version and exit.

Default
False

3.1.7 sourceengine

```
raytraverse sourceengine [OPTIONS]
```

initialize engine for sunrun

Effects

- Invokes scene

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

-idres <INTEGER>

the initial directional sampling resolution (as sqrt of samples per hemisphere)

Default
32

-nlev <INTEGER>

number of directional sampling levels, yielding a finalresolution of idres^2 * 2^(nlev) samples per hemisphere

Default
6

-rayargs <TEXT>

additional arguments to pass to the rendering engine

-source <TEXT>

name for this source

Default
source

-srcfile <FILE>

scene source description (required)

-vlt <FLOAT>

Leave at 1.0 for interior light sources. primary transmitting vlt, used to scale the accuracy parameter to the expected scene variance. Optional, but helpful with, for example, electrochromic glazing or shades

Default
1.0

FLAGS (DEFAULT TRUE):

--color, --no-color

Default
True

--default-args, --no-default-args

use raytraverse defaults before -rayargs, if False, uses radiance defaults

Default
True

HELP:

-opts, --opts
check parsed options

Default
False

--debug
show traceback on exceptions

Default
False

--version
Show the version and exit.

Default
False

3.1.8 skyrun

```
raytraverse skyrun [OPTIONS]
```

run scene under sky for a set of points (defined by area)

Effects

- Invokes scene
- Invokes area (no effects)
- Invokes skyengine
- **creates outdir/area.name/sky_points.tsv**
  - contents: 5cols x N rows: [sample_level idx x y z]
- **creates outdir/area.name/sky/######.rytpt**
  - each file is a LightPointKD initialization object

Options

VALUE OPTIONS:

-accuracy <FLOAT>
parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling)

Default
1.0

-edgemode <CHOICE>
if 'constant' value is set to -self.t1, so edge is always seen as detail. Internal edges (resulting from PlanMapper borders) will behave like 'nearest' for all options except 'constant'

Default
constant

Options
constant | reflect | nearest | mirror | wrap

-nlev <INTEGER>

number of levels to sample (final resolution will be ptres/2^(nlev-1))

Default
3

FLAGS (DEFAULT TRUE):

--jitter, --no-jitter

jitter samples on plane within adaptive sampling grid

Default
True

FLAGS (DEFAULT FALSE):

--overwrite, --no-overwrite

If True, reruns sampler when invoked, otherwise will first attempt to load results

Default
False

--plotp, --no-plotp

plot pdfs and sample vecs for each level

Default
False

HELP:

-opts, --opts

check parsed options

Default
False

--debug

show traceback on exceptions

Default
False

--version

Show the version and exit.

Default
False

3.1.9 directskyrun

```
raytraverse directskyrun [OPTIONS]
```

Options

FLAGS (DEFAULT FALSE):

--overwrite, --no-overwrite
If True, reruns sampler when invoked, otherwise will first attempt to load results

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

3.1.10 sunrun

```
raytraverse sunrun [OPTIONS]
```

run scene for a set of suns (defined by suns) for a set of points (defined by area)

Effects

- Invokes scene
- Invokes area (no effects)
- Invokes sunengine (no effects)
- **creates outdir/area.name/sun_####_points.tsv**
  - contents: 5cols x N rows: [sample_level idx x y z]
- **creates outdir/area.name/sky/sun_####/######.rytpt**
  - each file is a LightPointKD initialization object

Options

VALUE OPTIONS:

-accuracy <FLOAT>

parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling)

Default
1.0

-edgemode <CHOICE>

if 'constant' value is set to -self.t1, so edge is always seen as detail. Internal edges (resulting from PlanMapper borders) will behave like 'nearest' for all options except 'constant'

Default
constant

Options
constant | reflect | nearest | mirror | wrap

-nlev <INTEGER>

number of levels to sample (final resolution will be ptres/2^(nlev-1))

Default
3

-srcaccuracy <FLOAT>

parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling)

Default
1.0

-srcnlev <INTEGER>

number of levels to sample (final resolution will be sunres*2^(nlev-1))

Default
3

FLAGS (DEFAULT TRUE):

--jitter, --no-jitter

jitter samples on plane within adaptive sampling grid

Default
True

--recover, --no-recover

If True, recovers existing sampling

Default
True

--srcjitter, --no-srcjitter

jitter solar source within adaptive sampling grid for candidate SkyMappers, only affects weighting of selecting candidates in the same grid true positions are still used

Default
True

FLAGS (DEFAULT FALSE):

`--overwrite, --no-overwrite`
If True, reruns sampler when invoked, otherwise will first attempt to load results

Default
False

`--plotp, --no-plotp`
plot pdfs and sample vecs for each level

Default
False

HELP:

`-opts, --opts`
check parsed options

Default
False

`--debug`
show traceback on exceptions

Default
False

`--version`
Show the version and exit.

Default
False

3.1.11 sourcerun

```
raytraverse sourcerun [OPTIONS]
```

run scene for a single source (or multiple defined in a single scene file)

- Do not run as part of the same call as sunrun
- make sure rayargs are properly set in sunengine (not -ab 0)

Effects

- Invokes scene
- Invokes area (no effects)
- Invokes sunengine (no effects)
- **creates outdir/area.name/SOURCE_points.tsv**
  - contents: 5cols x N rows: [sample_level idx x y z]
- **creates outdir/area.name/sky/SOURCE/######.rytpt**
  - each file is a LightPointKD initialization object

Options

VALUE OPTIONS:

-accuracy <FLOAT>

parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling)

Default
1.0

-distance <FLOAT>

when using scene detail, the difference in ray length equivalent to final sampling luminance threshold

Default
0.5

-edgemode <CHOICE>

if 'constant' value is set to -self.t1, so edge is always seen as detail. Internal edges (resulting from PlanMapper borders) will behave like 'nearest' for all options except 'constant'

Default
constant

Options
constant | reflect | nearest | mirror | wrap

-nlev <INTEGER>

number of levels to sample (final resolution will be ptres/2^(nlev-1))

Default
3

-normal <FLOAT>

when using scene detail, the difference in surface normal (degrees) equivalent to final sampling luminance threshold

Default
5.0

FLAGS (DEFAULT TRUE):

--jitter, --no-jitter

jitter samples on plane within adaptive sampling grid

Default
True

FLAGS (DEFAULT FALSE):

--overwrite, --no-overwrite

If True, reruns sampler when invoked, otherwise will first attempt to load results

Default
False

--plotp, --no-plotp

plot pdfs and sample vecs for each level

Default
False

--scenedetail, --no-scenedetail

If True, includes scene details (distance, surface normal,and modifier as features). Increases sampling rate to improve image reconstruction

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

3.1.12 images

```
raytraverse images [OPTIONS]
```

render images

Prequisites:

- skyrun and sunrun must be manually invoked prior to this

Effects:

- Invokes scene
- Invokes skydata
- invokes area (no effects)
- invokes suns (no effects)
- writes: output images according to –namebyindex

Options

VALUE OPTIONS:

-basename <TEXT>

prefix of namebyindex.

Default
results

-interpolate <CHOICE>

Options
linear | fast | high | fastc | highc | | None | False

-res <INTEGER>

image resolution

Default
800

-resamprad <FLOAT>

radius for resampling sun vecs

Default
0.0

-resuntol <FLOAT>

tolerance for resampling sun views

Default
1.0

-sdirs <TEXT>

sensor directions, 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. vectors should all be 3 componnent (dx,dy,dz). used with3-component -sensors argument, all points are run for allviews, creating len(sensors)*len(sdirs) results. thisis the preferred option for multiple view directions, asthe calculations are grouped more efficiently

-sensors <TEXT>

sensor points, 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. points should either all be 3 componnent (x,y,z) or 6 component (x,y,z,dx,dy,dz). If 3 component, -sdirs is required, if 6-component, -sdirs is ignored

-simtype <TEXT>

simulation process/integration type:

- 1comp: standard DC method, sky patch only, full contribution depending on skyengine settings
- 2comp: sky patch for sky contribution, sun run for sun contribution, depth of contributions depends on skyengine and sunengine settings, no approximation for sun from sky patch
- 3comp: 2-phase DDS, sky handles sky+indirect sun, sun handles direct sun requires directskyrun -ab 1 and sunrun -ab 0
- 1compdv: standad DC method, but with direct view replacement of sun and specular reflections
- directview: only evaluate srcviewpts (direct views to sun and specular reflections
- directpatch: only evaluate results from dskyrun
- sunonly: only evaluate results from sunrun
- sunpatch: use skyrun results to evaluate sun contribution
- skyonly: use skyrun to evaluate sky contribution only

or source name (overrides –rresampleview, –directview, etc.

Default
3comp

-skymask <INTS>

mask to reduce output from full SkyData, enter as index rows in wea/epw file using space seperated list or python range notation:

- 370 371 372 (10AM-12PM on jan. 16th)

- 12:8760:24 (everyday at Noon)

-viewangle <FLOAT>

Default
180.0

FLAGS (DEFAULT TRUE):

--maskfull, --maskday
if false, skymask assumes daystep indices

Default
True

FLAGS (DEFAULT FALSE):

--blursun, --no-blursun
for simulating point spread function for direct sun view

Default
False

--directview, --no-directview
if True, ignore sky data and use daylight factors directly

Default
False

--namebyindex, --no-namebyindex
if False (default), names images by: <prefix>_sky-<row>_pt-<x>_<y>_<z>_vd-<dx>_<dy>_<dz>.hdr if True, names images by: <prefix>_sky-<row>_pt-<pidx>_vd-<vidx>.hdr, where pidx, vidx refer to the order of points, and vm.

Default
False

--resampleview, --no-resampleview
resample direct sun view directions

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

3.1.13 evaluate

```
raytraverse evaluate [OPTIONS]
```

evaluate metrics

Prequisites:

- skyrun and sunrun must be manually invoked prior to this

Effects:

- Invokes scene
- Invokes skydata
- invokes area (no effects)
- invokes suns (no effects)
- writes: <basename>.npz light result file (use "raytraverse pull" to extract data views)

Options

VALUE OPTIONS:

-basename <TEXT>

LightResult object is written to basename.npz.

Default
results

-metrics <TEXTS>

metrics to compute, choices: ["illum", "avglum", "gcr", "ugp", "dgp", "tasklum", "backlum", "dgp_t1", "log_gc", "dgp_t2", "ugr", "threshold", "pwsl2", "view_area", "backlum_true", "srcillum", "srcarea", "maxlum"]

Default
illum dgp ugp

-resamprad <FLOAT>

radius for resampling sun vecs

Default
0.0

-resuntol <FLOAT>

tolerance for resampling sun views

Default
1.0

-sdirs <TEXT>

sensor directions, 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. vectors should all be 3 componnent (dx,dy,dz). used with3-component -sensors argument, all points are run for allviews, creating len(sensors)*len(sdirs) results. thisis the preferred option for multiple view directions, asthe calculations are grouped more efficiently

-sensors <TEXT>

sensor points, 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. points should either all be 3 componnent (x,y,z) or 6 component (x,y,z,dx,dy,dz). If 3 component, -sdirs is required, if 6-component, -sdirs is ignored

-simtype <TEXT>

simulation process/integration type:

- 1comp: standard DC method, sky patch only, full contribution depending on skyengine settings
- 2comp: sky patch for sky contribution, sun run for sun contribution, depth of contributions depends on skyengine and sunengine settings, no approximation for sun from sky patch
- 3comp: 2-phase DDS, sky handles sky+indirect sun, sun handles direct sun requires directskyrun -ab 1 and sunrun -ab 0
- 1compdv: standad DC method, but with direct view replacement of sun and specular reflections
- directview: only evaluate srcviewpts (direct views to sun and specular reflections
- directpatch: only evaluate results from dskyrun
- sunonly: only evaluate results from sunrun
- sunpatch: use skyrun results to evaluate sun contribution
- skyonly: use skyrun to evaluate sky contribution only

or source name (overrides –rresampleview, –directview, etc.

Default
3comp

-skymask <INTS>

mask to reduce output from full SkyData, enter as index rows in wea/epw file using space seperated list or python range notation:

- 370 371 372 (10AM-12PM on jan. 16th)
- 12:8760:24 (everyday at Noon)

-threshold <FLOAT>

same as the evalglare -b option. if factor is larger than 100, it is used as constant threshold in cd/m2, else this factor is multiplied by the average task luminance. task position is center of image with a 30 degree field of view

Default
2000.0

-viewangle <FLOAT>

Default
180.0

FLAGS (DEFAULT TRUE):

--maskfull, --maskday

if false, skymask assumes daystep indices

Default
True

--npz, --no-npz

write LightResult object to .npz, use 'raytraverse pull'or LightResult('basename.npz') to access results

Default
True

FLAGS (DEFAULT FALSE):

`--blursun, --no-blursun`
: for simulating point spread function for direct sun view

    Default
    False

`--coercesumsafe, --no-coercesumsafe`
: to speed up evaluation, treat sources seperately,only compatible with illum, avglum, ugp (but note this is often WRONG!!!), dgp

    Default
    False

`--lowlight, --no-lowlight`
: use lowlight correction for dgp

    Default
    False

`--resampleview, --no-resampleview`
: resample direct sun view directions

    Default
    False

`--serr, --no-serr`
: include columns of sampling info/errors columns are: sun_pt_err, sun_pt_bin, sky_pt_err, sky_pt_bin, sun_err, sun_bin. 'err' is distance from queried vector to actual. 'bin' is the unraveled idx of source vector at a 500^2 resolution of the mapper.

    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

3.1.14 pull

```
raytraverse pull [OPTIONS]
```

Options

VALUE OPTIONS:

-col <TEXTS>

axis to preserve and order for flattening, if not all axes are specified default order is (sky, point, view, metric) the first value is the column preserved, the second (with -ofiles) is the file to write, and the rest determine the order for ravelling into rows.

Default
metric

-imgfilter <INTS>

image indices to return (ignored for lightfield result)

-imgzone <TEXT>

for making images from ZonalLightResult, path to areato sample over.

-lr <FILE>

.npz LightResult, overrides lightresult from chained commands (evaluate/imgmetric). required if not chained with evaluate or imgmetric.

-metricfilter <TEXTS>

metrics to return (non-existant are ignored)

-ofiles <TEXT>

if given output serialized files along first axis (given by order) with naming [ofiles]_XXXX.txt

-ptfilter <INTS>

point indices to return (ignored for imgmetric result)

-skyfill <FILE>

path to skydata file. assumes rows are timesteps. skyfilter should be None and other beside col should reduce to 1 or ofiles is given and sky is not first in order and all but first reduces to 1. LightResult should be a full evaluation (not masked)

-skyfilter <INTS>

sky indices to return (ignored for imgmetric result)

-spd <INTEGER>

steps per day. for use with –gridhdr col != sky matches data underlying -skyfill

Default
24

-viewfilter <INTS>

view direction indices to return (ignored for imgmetric result)

FLAGS (DEFAULT TRUE):

--header, --no-header
print col labels

Default
True

--rowlabel, --no-rowlabel
label row

Default
True

FLAGS (DEFAULT FALSE):

--gridhdr, --no-gridhdr
use with 'ofiles', order 'X point/sky Y' and make sure Y only has one value (with appropriate filter)

Default
False

--info, --no-info
skip execution and return shape and axis info about LightResult

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

3.2 raytu

```
raytu [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

the raytu executable is a command line interface to utility commands as part of the raytraverse python package.

the easiest way to manage options is to use a configuration file, to make a template:

```
raytu --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, for example:

```
raytraverse -c run.cfg imgmetric pull
```

will calculate metrics on a set of images and then print to the stdout.

Options

VALUE OPTIONS:

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

-n <INTEGER>
: sets the environment variable RAYTRAVERSE_PROC_CAP set to 0 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

transform
: coordinate transformations

imgmetric
: calculate metrics for hdr images, similar…

project
: project images between angular and…

lp2img
: make hdr directview of lightpoint

pull

padsky

pad filtered result data according to sky…

3.2.1 transform

```
raytu transform [OPTIONS]
```

coordinate transformations

Options

VALUE OPTIONS:

-cols <INTS>

coordinate columns (if none uses first N as required)

-d <TEXT>

a .npy file, a whitespace seperated text file (can be - for stdin) or entered as a string with commas between components of a point and spaces between rows.

-op <CHOICE>

transformation: 'xyz2xy': cartesian direction vector to equiangular. 'xyz2aa': cartesian direction vector to alt/azimuth. 'xyz2tp': cartesian to spherical (normalized). 'xyz2uv': cartesian to shirley-chiu square. 'uv2xyz': shirley-chiu square to certesian.

Default
xyz2xy

Options
xyz2xy | xyz2aa | xyz2tp | xyz2uv | uv2xyz

-outf <TEXT>

if none, return to stdout, else save as text file

-reshape <INTS>

reshape before transform (before flip)

FLAGS (DEFAULT FALSE):

--flip, --no-flip

transpose matrix before transform (after reshape)

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

3.2.2 imgmetric

```
raytu imgmetric [OPTIONS]
```

calculate metrics for hdr images, similar to evalglare but without glare source grouping, equivalent to -r 0 in evalglare. This ensures that all glare source positions are weighted by the metrics to which they are applied. Additional peak normalization reduces the deviation between images processed in different ways, for example pfilt with -r, rpict drawsource(), or an undersampled vwrays | rtrace run where the pixels give a coarse estimate of the actual sun area.

Options

VALUE OPTIONS:

-basename <TEXT>

LightResult object is written to basename.npz.

Default
img_metrics

-imgs <FILES>

hdr image files, must be angular fisheye projection,if no view in header, assumes 180 degree

-metrics <TEXTS>

metrics to compute, choices: [“illum”, “avglum”, “gcr”, “ugp”, “dgp”, “tasklum”, “backlum”, “dgp_t1”, “log_gc”, “dgp_t2”, “ugr”, “threshold”, “pwsl2”, “view_area”, “backlum_true”, “srcillum”, “srcarea”, “maxlum”]

Default
illum dgp ugp

-peaka <FLOAT>

expected peak area over which peak energy is distributed

Default
6.7967e-05

-peakr <FLOAT>

for peaks that do not meet expected area (such as partial suns, to determines the ratio of what counts as part of the source (max/peakr)

Default
4.0

-peakt <FLOAT>

include down to this threshold in possible peak, note thatonce expected peak energy is satisfied remaining pixels aremaintained, so it is safe-ish to keep this value low

Default
100000.0

-scale <FLOAT>
scale factor applied to pixel values to convert to cd/m^2

Default
179.0

-threshold <FLOAT>
same as the evalglare -b option. if factor is larger than 100, it is used as constant threshold in cd/m2, else this factor is multiplied by the average task luminance. task position is center of image with a 30 degree field of view

Default
2000.0

FLAGS (DEFAULT TRUE):

--npz, --no-npz
write LightResult object to .npz, use 'raytraverse pull'or LightResult('basename.npz') to access results

Default
True

--parallel, --no-parallel
use available cores

Default
True

--peakn, --no-peakn
corrrect aliasing and/or filtering artifacts for direct sun by assigning up to expected energy to peakarea

Default
True

FLAGS (DEFAULT FALSE):

--blursun, --no-blursun
applies human PSF to peak glare source (only if peekn=True

Default
False

--lowlight, --no-lowlight
use lowlight correction for dgp

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

3.2.3 project

```
raytu project [OPTIONS]
```

project images between angular and shirley-chiu square coordinates

Options

VALUE OPTIONS:

-img <FILES>
hdr image files, uv anr angular

FLAGS (DEFAULT FALSE):

--uv2ang, --ang2uv
direction of transform

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

3.2.4 lp2img

```
raytu lp2img [OPTIONS]
```

make hdr directview of lightpoint

Options

REQUIRED:

-lp <FILES>

Required path to lightpoint(s)

HELP:

-opts, --opts

check parsed options

Default
False

--debug

show traceback on exceptions

Default
False

--version

Show the version and exit.

Default
False

3.2.5 pull

```
raytu pull [OPTIONS]
```

Options

VALUE OPTIONS:

-col <TEXTS>

axis to preserve and order for flattening, if not all axes are specified default order is (sky, point, view, metric) the first value is the column preserved, the second (with -ofiles) is the file to write, and the rest determine the order for ravelling into rows.

Default
metric

-imgfilter <INTS>
: image indices to return (ignored for lightfield result)

-imgzone <TEXT>
: for making images from ZonalLightResult, path to areato sample over.

-lr <FILE>
: .npz LightResult, overrides lightresult from chained commands (evaluate/imgmetric). required if not chained with evaluate or imgmetric.

-metricfilter <TEXTS>
: metrics to return (non-existant are ignored)

-ofiles <TEXT>
: if given output serialized files along first axis (given by order) with naming [ofiles]_XXXX.txt

-ptfilter <INTS>
: point indices to return (ignored for imgmetric result)

-skyfill <FILE>
: path to skydata file. assumes rows are timesteps. skyfilter should be None and other beside col should reduce to 1 or ofiles is given and sky is not first in order and all but first reduces to 1. LightResult should be a full evaluation (not masked)

-skyfilter <INTS>
: sky indices to return (ignored for imgmetric result)

-spd <INTEGER>
: steps per day. for use with –gridhdr col != sky matches data underlying -skyfill

  Default
  24

-viewfilter <INTS>
: view direction indices to return (ignored for imgmetric result)

FLAGS (DEFAULT TRUE):

--header, --no-header
: print col labels

  Default
  True

--rowlabel, --no-rowlabel
: label row

  Default
  True

FLAGS (DEFAULT FALSE):

--gridhdr, --no-gridhdr
: use with 'ofiles', order 'X point/sky Y' and make sure Y only has one value (with appropriate filter)

  Default
  False

--info, --no-info
: skip execution and return shape and axis info about LightResult

    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

3.2.6 padsky

```
raytu padsky [OPTIONS]
```

pad filtered result data according to sky filtering

Options

VALUE OPTIONS:

-cols <INTS>
: cols of data to return (default all)

-data <TEXT>
: data to pad

-loc <FLOATS>
: location data given as 'lat lon mer' with + west of prime meridian overrides location data in wea

-minalt <FLOAT>
: minimum solar altitude for daylight masking

    Default
    2.0

-mindiff <FLOAT>
: minumum diffuse horizontal irradiance for daylight masking

    Default
    5.0

-mindir <FLOAT>
minumum direct normal irradiance for daylight masking

Default
0.0

-wea <TEXT>
path to epw, wea, .npy file or np.array, or .npz file,if loc not set attempts to extract location data (if needed).

HELP:

-opts, --opts
check parsed options

Default
False

--debug
show traceback on exceptions

Default
False

--version
Show the version and exit.

Default
False

3.3 raytraverse.scene

3.3.1 BaseScene

`class raytraverse.scene.BaseScene`(*outdir*, *scene=None*, *frozen=True*, *formatter=<class 'raytraverse.formatter.formatter.Formatter'>*, *reload=True*, *overwrite=False*, *log=True*, *loglevel=10*, *utc=False*)

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
- **loglevel** (*int, optional*) – maximum sampler level to log

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

reflection_search_scene()

log(*instance*, *message*, *err=False*, *level=0*)

print a message to the log file or stderr

Parameters

- **instance** (*Any*) – the parent class for the progress bar
- **message** (*str, optional*) – the message contents
- **err** (*bool, optional*) – print to stderr instead of self._logf
- **level** (*int, optional*) – the nested level of the message

progress_bar(*instance*, *iterable=None*, *message=None*, *total=None*, *level=0*, *workers=False*)

generate a tqdm progress bar and concurrent.futures Executor class

Parameters

- **instance** (*Any*) – the parent class for the progress bar
- **iterable** (*Sequence, optional*) – passed to tqdm, the sequence to loop over
- **message** (*str, optional*) – the prefix message for the progress bar
- **total** (*int, optional*) – the number of expected iterations (when interable is none)
- **level** (*int, optional*) – the nested level of the progress bar
- **workers** (*Union[bool, str], optional*) – if "thread/threads/t" returns a ThreadPoolExecutor, else if True returns a ProcessPoolExecutor.

Returns
a subclass of tqdm that decorates messages and has a pool property for multiprocessing.

Return type
TStqdm

Examples

with an iterable:

```
for i in self.scene.progress_bar(self, np.arange(10)):
    do stuff...
```

with workers=True:

with self.scene.progress_bar(self, total=len(jobs) workers=True) as pbar:
exc = pbar.pool do stuff… pbar.update(1)

3.3.2 Scene

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

Bases: *BaseScene*

container for radiance scene description

WARNING!! if scene parameter contains and instance primitive, sunsampler will throw a segmentation fault when it tries to change the source. As scene instanciation will make a frozen octree, it is better to feed complete scene description files, or an octree.

Parameters

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

reflection_search_scene()

reflection_search(*vecs*, *res=5*)

source_scene(*srcfile*, *srcname*)

3.3.3 ImageScene

class raytraverse.scene.**ImageScene**(*outdir*, *scene=None*, *reload=True*, *log=False*)

Bases: *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)

3.4 raytraverse.mapper

3.4.1 Mapper

class raytraverse.mapper.**Mapper**(*dxyz=(0.0, 0.0, 1.0)*, *sf=(1, 1)*, *bbox=((0, 0), (1, 1))*, *aspect=None*, *name='mapper'*, *origin=(0, 0, 0)*, *jitterrate=1.0*)

Bases: object

translate between world and normalized UV space. do not use directly, instead use an inheriting class.

Parameters

- **sf** (*tuple np.array, optional*) – scale factor for each axis (array of length(2)
- **bbox** (*tuple np.array, optional*) – bounding box for mapper shape (2, 2)
- **name** (*str, optional*) – used for output file naming

property aspect

property dxyz

(float, float, float) central view direction

property bbox
bounding box of view

Type
np.array of shape (2,2)

view2world(*xyz*)
rotate vectors from view direction to world Z

world2view(*xyz*)
rotate vectors from world Z to view direction

xyz2uv(*xyz*)
transform from world xyz space to mapper UV space

uv2xyz(*uv*, *stackorigin=False*)
transform from mapper UV space to world xyz

idx2uv(*idx*, *shape*, *jitter=True*)

Parameters

- **idx** (*flattened index*) –
- **shape** – the shape to unravel into
- **jitter** (*bool, optional*) – randomly offset coordinates within grid

Returns
uv – uv coordinates

Return type
np.array

static uv2idx(*uv*, *shape*)

xyz2vxy(*xyz*)
transform from world xyz to view image space (2d)

vxy2xyz(*xy*, *stackorigin=False*)
transform from view image space (2d) to world xyz

framesize(*res*)

pixels(*res*, *jitter=0.0*)
generate pixel coordinates for image space

pixelrays(*res*, *jitter=0.0*)
world xyz coordinates for pixels in view image space

ray2pixel(*xyz*, *res*, *integer=True*)
world xyz to pixel coordinate

pixel2ray(*pxy*, *res*)
pixel coordinate to world xyz vector

pixel2omega(*pxy*, *res*)
pixel area

in_view(*vec*, *indices=True*)
generate mask for vec that are in the field of view

header(***kwargs*)

init_img(*res=512*, *jitter=0.0*, ***kwargs*)
Initialize an image array with vectors and mask

Parameters

- **res** (*int, optional*) – image array resolution
- **jitter** (*float, optional*) – pixel jitter rate
- **kwargs** – passed to self.header

Returns

- **img** (*np.array*) – zero array of shape (res, 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 has inverse, mask for opposite view, usage:

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

- **header** (*str*)

add_vecs_to_img(*img*, *v*, *channels=(1, 0, 0)*, *grow=0*, ***kwargs*)

plot(*xyz*, *outf*, *res=1000*, *grow=1*, ***kwargs*)

3.4.2 AngularMixin

class raytraverse.mapper.angularmixin.**AngularMixin**
Bases: object

includes overrides of transformation functions for angular type mapper classes. Inherit before raytraverse.mapper.Mapper eg:

```
NewMapper(AngularMixin, Mapper)
```

initialization of NewMapper must include declarations of:

```
self._viewangle = viewangle
self._chordfactor = chordfactor
self._ivm = ivm
```

xyz2uv(*xyz*)
transform from world xyz space to mapper UV space

uv2xyz(*uv*, *stackorigin=False*)
transform from mapper UV space to world xyz

xyz2vxy(*xyz*)
transform from world xyz to view image space (2d)

vxy2xyz(*xy*, *stackorigin=False*)
transform from view image space (2d) to world xyz

static **framesize**(*res*)

pixelrays(*res*)
world xyz coordinates for pixels in view image space

pixel2omega(*pxy*, *res*)

pixel solid angle

in_view(*vec*, *indices=True*, *tol=0.0*)

generate mask for vec that are in the field of view (up to 180 degrees) if view aspect is 2, only tests against primary view direction

header(*pt=(0, 0, 0)*, ***kwargs*)

init_img(*res=512*, *pt=(0, 0, 0)*, *features=1*, ***kwargs*)

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 features > 1, use mask 2 fro color images
- **header** (*str*)

add_vecs_to_img(*img*, *v*, *channels=(1, 0, 0)*, *grow=0*, *fisheye=True*)

property viewangle

view angle

property ivm

viewmapper for opposite view direction (in case of 360 degree view

ctheta(*vec*)

cos(theta) (dot product) between view direction and vec

radians(*vec*)

angle in radians betweeen vieew direction and vec

degrees(*vec*)

angle in degrees betweeen vieew direction and vec

3.4.3 ViewMapper

class raytraverse.mapper.ViewMapper(*dxyz=(0.0, 1.0, 0.0)*, *viewangle=360.0*, *name='view'*, *origin=(0, 0, 0)*, *jitterrate=0.9*)

Bases: *AngularMixin*, *Mapper*

translate between world direction vectors and normalized UV space for a given view angle. pixel projection yields equiangular projection

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 aspect

property dxyz

(float, float, float) central view direction

idx2uv(*idx*, *shape*, *jitter=True*)

Parameters

- **idx** (*flattened index*) –
- **shape** – the shape to unravel into
- **jitter** (*bool, optional*) – randomly offset coordinates within grid

Returns
uv – uv coordinates

Return type
np.array

3.4.4 SkyMapper

class raytraverse.mapper.**SkyMapper**(*loc=None*, *skyro=0.0*, *sunres=9*, *name='suns'*, *jitterrate=0.5*)

Bases: *AngularMixin*, *Mapper*

translate between world direction vectors and normalized UV space for a given view angle. pixel projection yields equiangular projection

Parameters

- **loc** (*any, optional*) – can be a number of formats:

  1. either a numeric iterable of length 3 (lat, lon, mer) where lat is +west and mer is tz*15 (matching gendaylit).
  2. an array (or tsv file loadable with np.loadtxt) of shape (N,3), (N,4), or (N,5):
     a. 2 elements: alt, azm (angles in degrees)
     b. 3 elements: dx,dy,dz of sun positions
     c. 4 elements: alt, azm, dirnorm, diffhoriz (angles in degrees)
     d. 5 elements: dx, dy, dz, dirnorm, diffhoriz.
  3. path to an epw or wea formatted file
  4. None (default) all possible sun positions are considered self.in_solarbounds always returns True

  in the case of a geo location, sun positions are considered valid when in the solar transit for that location. for candidate options, sun positions are drawn from this set (with one randomly chosen from all candidates within bin.

- **skyro** (*float, optional*) – counterclockwise sky-rotation in degrees (equivalent to clockwise project north rotation)
- **sunres** (*float, optional*) – initial sampling resolution for suns
- **name** (*str, optional*) –

property skyro

property loc

property solarbounds

property candidates

in_solarbounds(*xyz*, *level=0*, *include='any'*)

for checking if src direction is in solar transit

Parameters

- **xyz** (*np.array*) – source directions
- **level** (*int*) – for determining patch size, 2**level resolution from sunres
- **include** (*{'center', 'all', 'any'}, optional*) – boundary test condition. 'center' tests uv only, 'all' requires for corners of box centered at uv to be in, 'any' requires atleast one corner. 'any' is the least restrictive and 'all' is the most, but with increasing levels 'any' will exclude more positions while 'all' will exclude less (both approaching 'center' as level -> N)

Returns

result – Truth of ray.src within solar transit

Return type

np.array

shape(*level=0*)

solar_grid(*jitter=True*, *level=0*, *masked=True*)

generate a grid of solar positions

Parameters

- **jitter** (*bool, optional*) – if None, use the instance default, if True jitters point samples within stratified grid
- **level** (*int, optional*) – sets the resolution of the grid as a power of 2 from sunress
- **masked** (*bool, optional*) – apply in_solarbounds to suns before returning

Returns

shape (N, 3)

Return type

np.array

3.4.5 PlanMapper

class raytraverse.mapper.**PlanMapper**(*area*, *ptres=1.0*, *rotation=0.0*, *zheight=None*, *name='plan'*, *jitterrate=0.5*, *autorotate=False*, *autogrid=None*)

Bases: *Mapper*

translate between world positions on a horizontal plane and normalized UV space for a given view angle. pixel projection yields a parallel plan projection

Parameters

- **area** (*str np.array, optional*) – radiance scene geometry defining a plane to sample, tsv file of points to generate bounding box, or np.array of points.
- **ptres** (*float, optional*) – resolution for considering points duplicates, border generation (1/2) and add_grid(). updateable
- **rotation** (*float, optional*) – positive Z rotation for point grid alignment
- **zheight** (*float, optional*) – override calculated zheight
- **name** (*str, optional*) – plan mapper name used for output file naming
- **jitterrate** (*float, optional*) – proportion of cell to jitter within
- **autorotate** (*bool, optional*) – if true set rotation based on long axis of area geometry

- **autogrid** (*int, optional*) – if given, autoset ptres based on this minimum number of points at level 0 along the minimum dimemsion (width or height)

ptres

point resolution for area look ups and grid

Type
float

property dxyz

(float, float, float) central view point

property rotation

ccw rotation (in degrees) for point grid on plane

Type
float

property bbox

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

Type
np.array

update_bbox(*plane*, *level=0*, *updatez=True*)

handle bounding box generation from plane or points

uv2xyz(*uv*, *stackorigin=False*)

transform from mapper UV space to world xyz

in_view_uv(*uv*, *indices=True*, ***kwargs*)

in_view(*vec*, *indices=True*)

check if point is in boundary path

Parameters

- **vec** (*np.array*) – xyz coordinates, shape (N, 3)
- **indices** (*bool, optional*) – return indices of True items rather than boolean array

Returns
mask – boolean array, shape (N,)

Return type
np.array

header(***kwargs*)

borders()

world coordinate vertices of planmapper boundaries

property boundary

bbox_vertices(*offset=0*, *close=False*)

shape(*level=0*)

point_grid(*jitter=True*, *level=0*, *masked=True*, *snap=None*)

generate a grid of points

Parameters

- **jitter** (*bool, optional*) – if None, use the instance default, if True jitters point samples within stratified grid
- **level** (*int, optional*) – sets the resolution of the grid as a power of 2 from ptres

- **masked** (*bool, optional*) – apply in_view to points before returning
- **snap** (*int, optional*) – level to snap samples to when jitter=False should be > level

Returns
shape (N, 3)

Return type
np.array

point_grid_uv(*jitter=True*, *level=0*, *masked=True*, *snap=None*)
add a grid of UV coordinates

Parameters

- **jitter** (*bool, optional*) – if None, use the instance default, if True jitters point samples within stratified grid
- **level** (*int, optional*) – sets the resolution of the grid as a power of 2 from ptres
- **masked** (*bool, optional*) – apply in_view to points before returning
- **snap** (*int, optional*) – level to snap samples to when jitter=False should be > level

Returns
shape (N, 2)

Return type
np.array

3.4.6 MaskedPlanMapper

class raytraverse.mapper.**MaskedPlanMapper**(*pm*, *valid*, *level*)
Bases: *PlanMapper*

translate between world positions on a horizontal plane and normalized UV space for a given view angle. pixel projection yields a parallel plan projection

Parameters

- **pm** (raytraverse.mapper.PlanMapper) – the source mapper to copy
- **valid** (*np.array*) – a list of valid points used to make a mask, grid cells not represented by one of valid will be masked
- **level** (*int, optional*) – the level at which to grid the valid candidates

update_mask(*valid*, *level*)

in_view_uv(*uv*, *indices=True*, *usemask=True*)

3.5 raytraverse.formatter

3.5.1 Formatter

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

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

comment = '#'
line comment character

scene_ext = ''
extension for renderer scene file

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

static get_scene(*scene*)

static get_skydef(*color=None*, *ground=True*, *name='skyglow'*)
assemble sky definition

static get_sundef(*vec*, *color*, ***kwargs*)
assemble sun definition

3.5.2 RadianceFormatter

class raytraverse.formatter.**RadianceFormatter**
Bases: *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 get_scene(*scene*)
recover scene file paths from compiled octree

Parameters
scene (*octree file*) –

Returns

- **files** (*string to use in new octree generation. -i prepended before*)
- *each actree*
- **frozen** (*if result will be a frozen octree*)

static get_skydef(*color=(0.96, 1.004, 1.118)*, *ground=True*, *name='skyglow'*, *mod='void'*, *groundname=None*, *groundcolor=(1, 1, 1)*)
assemble sky definition

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

3.6 raytraverse.renderer

3.6.1 RadianceRenderer

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

Bases: `object`

Virtual class for wrapping c++ Radiance renderer executable classes

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

name = 'radiance_virtual'

instance = None

srcn = 1

features = 1

defaultargs = ''

args = None

nproc = None

run(**args*, ***kwargs*)

alias for call, for consistency with SamplerPt classes for nested dimensions of evaluation

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.

3.6.2 Rtrace

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

Bases: `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 raytraverse.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)
```

If rayargs include cache files (ambient cache or photon map) be careful with updating sources. If you are going to swap sources, update the arguments as well with the new paths:

```
r = renderer.Rtrace(args, scene)
r.set_args(args.replace("temp.amb", "temp2.amb"))
r.load_source(srcdef)
```

Note that if you are using ambient caching, you must give an ambient file, because without a file ambient values are not shared across processes or successive calls to the instance.

name = 'rtrace'

instance = <MagicMock name='mock.get_instance()' id='140186802307920'>

craytraverse.crenderer.cRtrace

defaultargs = '-u+ -ab 16 -av 0 0 0 -aa 0 -as 0 -dc 1 -dt 0 -lr -14 -ad 1000 -lw 0.00004 -st 0 -ss 16 -w-'

directargs = '-w- -av 0 0 0 -ab 0 -lr 1 -n 1'

usedirect = False

ospec = 'Z'

classmethod get_default_args()

return default arguments of the class

classmethod set_args(*args*, *nproc=None*)

prepare arguments to call engine instance initialization

Parameters

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

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 check_amb(args)`

`classmethod load_source(srcfile, freesrc=- 1, ambfile=None)`

add a source description to the loaded scene

Parameters

- **srcfile** (*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
- **ambfile** (*str, optional*) – path to ambient file. if given, and arguments

`classmethod load_solar_source(scene, sun, ambfile=None, intens=1)`

`classmethod get_sources()`

returns source information

Returns

- **sources** (*np.array*) – x,y,z,v,a distant: direction, view angle, solid angle not distant: location, max radius, area
- **distant** (*np.arrary*) – booleans, true if source type is distant

3.6.3 Rcontrib

`class raytraverse.renderer.Rcontrib(rayargs=None, scene=None, nproc=None, skyres=15, modname='skyglow', ground=True, default_args=True)`

Bases: `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** (*int, optional*) – resolution of sky patches (sqrt(patches / hemisphere)). So if skyres=18, 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'

instance = <MagicMock name='mock.get_instance()' id='140186669859984'>

ground = True

skyres = 15

srcn = 226

modname = 'skyglow'

classmethod setup(*scene=None*, *ground=True*, *modname='skyglow'*, *skyres=18*)

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`) – resolution of sky patches (sqrt(patches / hemisphere)). 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

3.6.4 ImageRenderer

class raytraverse.renderer.**ImageRenderer**(*scene*, *viewmapper=None*, *method='linear'*, *color=False*)

Bases: `object`

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

run(**args*, ***kwargs*)

alias for call, for consistency with SamplerPt classes for nested dimensions of evaluation

3.6.5 SpRenderer

class raytraverse.renderer.**SpRenderer**(*rayargs=None*, *scene=None*, *nproc=None*, *default_args=True*)

Bases: `object`

sub-process renderer for calling external executables

args = None

scene = None

name = 'rtrace'

defaultargs = ''

nproc = None

run(**args*, ***kwargs*)

alias for call, for consistency with SamplerPt classes for nested dimensions of evaluation

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.

3.7 raytraverse.sky

3.7.1 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

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

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*, *intersky=True*)

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, optional*) – (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*, *intersky=True*, ***kwargs*)

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, optional*) – sky patch subdivision n = jn^2
- **intersky** (*bool, optional*) – include interreflection between ground and sky (mimics perezlum.cal, not present in gendaymtx
- **kwargs** (*dict, optional*) – passed to perez()

Returns

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

raytraverse.sky.skycalc.**radiance_skydef**(*sunpos*, *dirdif*, *loc=None*, *md=None*, *ground_fac=0.2*, *td=10.9735311509*, *ro=0.0*)

similar to gendaylit, returns strings

Parameters

- **sunpos** (*Sequence*) – dx, dy, dz sun position or m,d,h (if loc is not None)
- **dirdif** (*Sequence*) – direct normal, diffuse horizontal W/m^2
- **loc** (*tuple, optional*) – location data given as lat, lon, mer with + west of prime meridian triggers sunpos treated as timestep
- **md** (*tuple, optional*) – month day of sky calcs (for more precise eccentricity calc with xyz sunpos)

- **ground_fac** (*float*) – scaling factor (reflectance) for ground brightness
- **td** (*np.array float, optional*) – (N,) dew point temperature in C
- **ro** (*float, optional*) – ignored if sunpos is xyz, else angle in degrees counter-clockwise to rotate sky (to correct model north, equivalent to clockwise rotation of scene)

Returns

- **desc** (*str*) – comments with sky info
- **sund** (*str*) – solar material and sun object ("" if no sun)
- **skyd** (*str*) – perezlum brightfunc definition and sky/ground objects

3.7.2 SkyData

class raytraverse.sky.**SkyData**(*wea*, *loc=None*, *skyro=0.0*, *ground_fac=0.2*, *intersky=True*, *skyres=15*, *minalt=2.0*, *mindiff=5.0*, *mindir=0.0*, *ground=True*, *srcname='sky'*)

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, .npy file or np.array, or .npz file, if loc not set attempts to extract location data (if needed).
- **loc** (*tuple, 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)
- **ground_fac** (*float, optional*) – ground reflectance
- **intersky** (*bool, optional*) – include interreflection between ground and sky (mimics perezlum.cal, not present in gendaymtx)
- **skyres** (*int, optional*) – resolution of sky patches (sqrt(patches / hemisphere))
- **minalt** (*float, optional*) – minimum solar altitude for daylight masking
- **mindiff** (*float, optional*) – minumum diffuse horizontal irradiance for daylight masking

skyres

sky patach resolution

property skyro

sky rotation (in degrees, ccw)

property loc

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

property rowlabel

m,d,h (if known)

property skydata

sun position and dirnorm diffhoriz

write(*name='skydata'*, *scene=None*, *compressed=True*)

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

property daysteps

property daymask

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

property fullmask

property maskindices

property mask

an additional mask for smtx data

property smtx

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

property sun

shape (np.sum(daymask), 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 sunproxy

corresponding sky bin for each sun position in daymask

smtx_patch_sun(*includesky=True*)

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

fill_data(*x*, *fill_value=0.0*, *rowlabels=False*)

Parameters

- **x** (*np.array*) – first axis size = len(self.daymask[self.mask])
- **fill_value** (*Union[int, float], optional*) – value in padded array
- **rowlabels** (*bool, optional*) – include rowlabels

Returns
data in x padded with fill value to original shape of skydata

Return type
np.array

label(*x*)

masked_idx(*i*)

radiance_sky_matrix(*outf*, *fmt='float'*, *sun=True*, *sky=True*, *ncomps=3*)

sky_description(*i*, *prefix='skydata'*, *grid=False*, *sun=True*, *ground=True*, *sunpatch=False*)

generate radiance scene files to directly render sky data at index i

Parameters

- **i** (*int*) – index of sky vector to generate (indexed from skydata, not daymask)
- **prefix** (*str, optional*) – name/path for output files
- **grid** (*bool, optional*) – render sky patches with grid lines
- **sun** (*bool, optional*) – include sun source in rad file
- **ground** (*bool, optional*) – include ground source
- **sunpatch** (*bool, optional*) – include sun energy in sun_patch (sun should be false)

Returns

basename of 3 files written: prefix_i (.rad, .cal, and .dat) .cal and .dat must be located in RAYPATH (which can include .) or else edit the .rad file to explicitly point to their locations. note that if grid is True, the sky will not be accurate, so only use this for illustrative purposes.

Return type

str

Raises

IndexError – if i is not in masked indices

3.7.3 SkyDataMask

class raytraverse.sky.**SkyDataMask**(*hours*)

Bases: *SkyData*

spoofed skydata class for use with light results

Parameters

hours (*np.array*) – hours of year given as (m, d, h) where hour is H.5 (assumes 8760) to use as daymask.

property skydata

sun position and dirnorm diffhoriz

3.8 raytraverse.sampler

3.8.1 draw

wavelet and associated probability functions.

raytraverse.sampler.draw.**get_detail**(*data*, **args*, *mode='reflect'*, *cval=0.0*)

convolve a set of kernels with data. computes the sum of the absolute values of each convolution.

Parameters

- **data** (*np.array*) – source data (atleast 2D), detail calculated over last 2D
- **args** (*np.array*) – filters
- **mode** (*str*) – signal extension mode (passed to scipy.ndimage.convolve)
- **cval** (*float*) – constant value (passed to scipy.ndimage.convolve, used when mode='constant')

Returns
detail_array – 1d array of detail coefficients (row major order) matching size of data

Return type
np.array

`raytraverse.sampler.draw.from_pdf`(*pdf*, *threshold*, *lb=0.5*, *ub=4*, *minsamp=0*)

generate choices from a numeric probability distribution

Parameters

- **pdf** (*np.array*) – 1-d array of weights
- **threshold** (*float*) – the threshold used to determine the number of choices to draw given by pdf > threshold
- **lb** (*float, optional*) – values below threshold * lb will be excluded from candidates (lb must be in (0,1)
- **ub** (*float, optional*) – the maximum weight is set to ub*threshold, meaning all values in pdf >= to ub*threshold have an equal chance of being selected. in cases where extreme values are much higher than moderate values, but 100% sampling of extreme areas should be avoided, this value should be lower, such as when a region is sampled at a very high resolution (as is the case with directional sampling). On the other hand, set this value higher for sampling schemes with a low final resolution (like area sampling). If ub <= 1, then a deterministic choice is made, returning the idx of all values in pdf > threshold.

Returns
idx – an index array of choices, size varies.

Return type
np.array

3.8.2 BaseSampler

class `raytraverse.sampler.BaseSampler`(*scene*, *engine*, *accuracy=1.0*, *stype='generic'*, *samplerlevel=0*, *featurefunc=<function amax>*, *features=1*, *weightfunc=<function amax>*)

Bases: `object`

wavelet based sampling class this is a virutal class that holds the shared sampling methods across directional, area, and sunposition samplers. subclasses are named as: {Source}Sampler{SamplingRange}, for instance:

- **SamplerPt: virtual base class for sampling directions from a point**
  - SkySamplerPt: sampling directions from a point with a sky patch source.
  - SunSamplerPt: sampling directions from a point with a single sun source
  - SunSamplerPtView: sampling the view from a point of the sun
  - ImageSampler: (re)sampling a fisheye image, useful for testing
- SamplerArea: sampling points on a horizontal planar area with any source type
- SamplerSuns: sampling sun positions (with nested area sampler)

Parameters

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry and formatter compatible with engine
- **engine** – has a run() method
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)

- **stype** (*str, optional*) – sampler type (prefixes output files)
- **featurefunc** (*func, optional*) – takes detail array as an argument, shape: (features,N, M) and an axis=0 keyword argument, returns shape (N, M). could be np.max, np.sum np.average or us custom function following the same pattern.
- **features** (*int, optional*) – number of values evaluated for detail

t0 = 0.00390625

initial sampling threshold coefficient this value times the accuracy parameter is passed to raytraverse.sampler.draw.from_pdf() at level 0 (usually not used)

t1 = 0.0625

final sampling threshold coefficient this value times the accuracy parameter is passed to raytraverse.sampler.draw.from_pdf() at final level, intermediate sampling levels are thresholded by a linearly interpolated between t0 and t1

lb = 0.25

lower bound for drawing from pdf passed to raytraverse.sampler.draw.from_pdf()

ub = 8

upper bound for drawing from pdf passed to raytraverse.sampler.draw.from_pdf()

scene

scene information

Type
raytraverse.scene.Scene

accuracy

accuracy parameter some subclassed samplers may apply a scale factor to normalize threshold values depending on source brightness (see for instance ImageSampler and SunSamplerPt)

Type
float

stype

sampler type

Type
str

weights

holds weights for self.draw

Type
np.array

featurefunc

func takes weights and axis=0 argument to reduce detail

weightfunc

func takes weights and axis=1 argument to reduce output from engine when engine produces more features than sampler needs

property levels

sampling scheme

Getter
Returns the sampling scheme

Setter
Set the sampling scheme

Type
np.array

`sampling_scheme`(**args*)
calculate sampling scheme

`run`(*mapper*, *name*, *levels*, *plotp=False*, *log='err'*, *pfish=True*, ***kwargs*)
trigger a sampling run. subclasses should return a LightPoint/LightField from the executed object state (first call this method with super().run(…)

Parameters

- **mapper** (`raytraverse.mapper.Mapper`) – mapper to sample
- **name** (*str*) – output name
- **levels** (*np.array*) – the sampling scheme
- **plotp** (*bool, optional*) – plot weights, detail and vectors for each level
- **log** (*str, optional*) – 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
- **pfish** (*bool, optional*) – if True and plotp, use fisheye projection for detail/weight/vector images.
- **kwargs** – unused

`draw`(*level*)
draw samples based on detail calculated from weights

Returns

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

`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

`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

`detailfunc = 'wav'`
filter banks for calculating detail choices:

'haar': [[1 -1]]/2, [[1] [-1]]/2, [[1, 0] [0, -1]]/2

'wav': [[-1 2 -1]] / 2, [[-1] [2] [-1]] / 2, [[-1 0 0] [0 2 0] [0 0 -1]] / 2

3.8.3 ISamplerArea

class raytraverse.sampler.**ISamplerArea**(*scene*, *engine*, *accuracy=1.0*, *nlev=3*, *jitter=True*, *edgemode='constant'*, ***kwargs*)

Bases: *SamplerArea*

wavelet based area sampling class using Sensor as engine

Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry and formatter compatible with engine
- **engine** (*raytraverse.sampler.Sensor*) – renderer
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)
- **nlev** (*int, optional*) – number of levels to sample
- **jitter** (*bool, optional*) – jitter samples
- **edgemode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – default: 'constant', if 'constant' value is set to -self.t1, so edge is always seen as detail. Internal edges (resulting from PlanMapper borders) will behave like 'nearest' for all options except 'constant'

srcn

number of sources return per vector by run

Type
int

run(*mapper*, *plotp=False*, ***kwargs*)

adapively sample an area defined by mapper

Parameters

- **mapper** (raytraverse.mapper.PlanMapper) – the pointset to build/run
- **plotp** (*bool, optional*) – plot weights, detail and vectors for each level
- **kwargs** – passed to self.run()

Return type
raytraverse.lightplane.SensorPlaneKD

repeat(*guide*, *stype*)

repeat the sampling of a guide SensorPlane (to match all points)

Parameters

- **guide** (LightPlaneKD) –
- **stype** (*str*) – alternate stype name. raises a ValueError if it matches the guide.

Return type
raytraverse.lightfield.SensorPlaneKD

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

3.8.4 ISamplerSuns

class raytraverse.sampler.**ISamplerSuns**(*scene*, *engine*, *accuracy=1.0*, *nlev=3*, *jitter=True*, *areakwargs=None*)

Bases: *SamplerSuns*

wavelet based sun position sampling class

Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry and formatter compatible with engine
- **engine** (*raytraverse.sampler.Sensor*) – with initialized renderer instance (with scene loaded, no sources)
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)
- **nlev** (*int, optional*) – number of levels to sample
- **jitter** (*bool, optional*) – jitter samples
- **ptkwargs** (*dict, optional*) – kwargs for raytraveerse.sampler.SunSamplerPt initialization
- **areakwargs** (*dict, optional*) – kwargs for raytravrse.sampler.SamplerArea initialization
- **metricset** (*iterable, optional*) – subset of samplerarea.metric set to use for sun detail calculation.

get_existing_run(*skymapper*, *areamapper*)

check for file conflicts before running/overwriting parameters match call to run

Parameters

- **skymapper** (raytraverse.mapper.SkyMapper) – the mapping for drawing suns
- **areamapper** (raytraverse.mapper.PlanMapper) – the mapping for drawing points

Returns

conflicts –

a tuple of found conflicts (None for each if no conflicts:

- suns: np.array of sun positions in vfile
- ptfiles: existing point files

Return type
tuple

run(*skymapper*, *areamapper*, ***kwargs*)

adaptively sample sun positions for an area (also adaptively sampled)

Parameters

- **skymapper** (raytraverse.mapper.SkyMapper) – the mapping for drawing suns
- **areamapper** (raytraverse.mapper.PlanMapper) – the mapping for drawing points
- **kwargs** – passed to self.run()

Return type
raytraverse.lightlplane.LightPlaneKD

3.8.5 SamplerSuns

`class raytraverse.sampler.SamplerSuns`(*scene*, *engine*, *accuracy=1.0*, *nlev=3*, *jitter=True*, *ptkwargs=None*, *areakwargs=None*, *metricset=('avglum', 'loggcr')*)

Bases: `BaseSampler`

wavelet based sun position sampling class

Parameters

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry and formatter compatible with engine
- **engine** (`raytraverse.renderer.Rtrace`) – initialized renderer instance (with scene loaded, no sources)
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)
- **nlev** (*int, optional*) – number of levels to sample
- **jitter** (*bool, optional*) – jitter samples
- **ptkwargs** (*dict, optional*) – kwargs for raytraveerse.sampler.SunSamplerPt initialization
- **areakwargs** (*dict, optional*) – kwargs for raytravrse.sampler.SamplerArea initialization
- **metricset** (*iterable, optional*) – subset of samplerarea.metric set to use for sun detail calculation.

t0 = 0.05
initial sampling threshold coefficient

t1 = 0.125
final sampling threshold coefficient

ub = 8
upper bound for drawing from pdf

sampling_scheme(*mapper*)
calculate sampling scheme

get_existing_run(*skymapper*, *areamapper*)
check for file conflicts before running/overwriting parameters match call to run

Parameters

- **skymapper** (`raytraverse.mapper.SkyMapper`) – the mapping for drawing suns
- **areamapper** (`raytraverse.mapper.PlanMapper`) – the mapping for drawing points

Returns

conflicts –

a tuple of found conflicts (None for each if no conflicts:

- suns: np.array of sun positions in vfile
- ptfiles: existing point files

Return type
tuple

run(*skymapper*, *areamapper*, *specguide=None*, *recover=True*, ***kwargs*)
adaptively sample sun positions for an area (also adaptively sampled)

Parameters

- **skymapper** (raytraverse.mapper.SkyMapper) – the mapping for drawing suns
- **areamapper** (raytraverse.mapper.PlanMapper) – the mapping for drawing points
- **specguide** (*Union[*raytraverse.lightfield.LightPlaneKD*, Bool]*) – sky source lightfield to use as specular guide for sampling
- **recover** (*continue run on top of existing files, if false, overwrites*) – previous run.
- **kwargs** – passed to self.run()

Return type
raytraverse.lightlplane.LightPlaneKD

draw(*level*)
draw on condition of in_solarbounds from skymapper. In this way all solar positions are presented to the area sampler, but the area sampler is initialized with a weighting to sample only where there is variance between sun position. this keeps the subsampling of area and solar position independent.

Returns

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

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

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

Parameters
vecs (*np.array*) – sample vectors

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

Return type
np.array

idxvecs()

3.8.6 SamplerArea

class raytraverse.sampler.**SamplerArea**(*scene*, *engine*, *accuracy=1.0*, *nlev=3*, *jitter=True*, *edgemode='constant'*, *metricclass=<class 'raytraverse.evaluate.samplingmetrics.SamplingMetrics'>*, *metricset=('avglum', 'loggcr', 'xpeak', 'ypeak')*, ***kwargs*)

Bases: `BaseSampler`

wavelet based area sampling class

Parameters

- **scene** (`raytraverse.scene.Scene`) – scene class containing geometry and formatter compatible with engine
- **engine** (`raytraverse.sampler.SamplerPt`) – point sampler
- **accuracy** (*float, optional*) – parameter to set threshold at sampling level relative to final level threshold (smaller number will increase sampling, default is 1.0)
- **nlev** (*int, optional*) – number of levels to sample
- **jitter** (*bool, optional*) – jitter samples
- **edgemode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – default: 'constant', if 'constant' value is set to -self.t1, so edge is always seen as detail. Internal edges (resulting from PlanMapper borders) will behave like 'nearest' for all options except 'constant'
- **metricclass** (`raytraverse.evaluate.BaseMetricSet`, *optional*) – the metric calculator used to compute weights
- **metricset** (*iterable, optional*) – list of metrics (must be recognized by metric-class. metrics containing "lum" will be normalized to 0-1)

t0 = 0.1

initial sampling threshold coefficient

t1 = 0.9

final sampling threshold coefficient

ub = 100

upper bound for drawing from pdf

metricclass

raytraverse.evaluate.BaseMetricSet

metricset

iterable

features

int:

property edgemode

sampling_scheme(*mapper*)

calculate sampling scheme

run(*mapper*, *specguide=None*, *plotp=False*, ***kwargs*)

adapively sample an area defined by mapper

Parameters

- **mapper** (`raytraverse.mapper.PlanMapper`) – the pointset to build/run
- **specguide** (*Union[None, bool, str]*) –

- **plotp** (*bool, optional*) – plot weights, detail and vectors for each level
- **kwargs** – passed to self.run()

Return type
raytraverse.lightlplane.LightPlaneKD

repeat(*guide*, *stype*)
repeat the sampling of a guide LightPlane (to match all rays)

Parameters

- **guide** (LightPlaneKD) –
- **stype** (*str*) – alternate stype name for samplerpt. raises a ValueError if it matches the guide.

draw(*level*)
draw samples based on detail calculated from weights

Returns

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

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

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

idxvecs()

3.8.7 SamplerPt

class raytraverse.sampler.**SamplerPt**(*scene*, *engine*, *idres=32*, *nlev=5*, *accuracy=1.0*, *srcn=1*, *stype='generic'*, *features=1*, *samplerlevel=0*, ***kwargs*)

Bases: *BaseSampler*

wavelet based sampling class for direction rays from a point

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 sqrt of samples per hemisphere)
- **nlev** (*int, optional*) – number of levels to sample (eeach lvl doubles idres)
- **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)
- **features** (*int, optional*) – number of values evaluated for detail
- **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()

srcn

number of sources return per vector by run

Type
int

idres

initial direction resolution (as sqrt of samples per hemisphere (or view angle)

Type
int

sampling_scheme(*a*)

calculate sampling scheme

run(*point*, *posidx*, *mapper=None*, *lpargs=None*, ***kwargs*)

sample a single point, position index handles file naming

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **mapper** (raytraverse.mapper.ViewMapper) – view direction to sample
- **lpargs** (*dict, optional*) – keyword arguments forwarded to LightPointKD construction
- **kwargs** – passed to BaseSampler.run()

Return type
LightPointKD

repeat(*guide*, *stype*)

3.8.8 SkySamplerPt

class raytraverse.sampler.**SkySamplerPt**(*scene*, *engine*, ***kwargs*)

Bases: *SamplerPt*

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

3.8.9 SunSamplerPt

class raytraverse.sampler.**SunSamplerPt**(*scene*, *engine*, *sun*, *sunbin*, *nlev=6*, *stype='sun'*, ***kwargs*)

Bases: *SamplerPt*

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

sunpos

sun position x,y,z

Type

np.array

run(*point*, *posidx*, *specguide=None*, ***kwargs*)

sample a single point, position index handles file naming

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **mapper** (raytraverse.mapper.ViewMapper) – view direction to sample
- **lpargs** (*dict, optional*) – keyword arguments forwarded to LightPointKD construction
- **kwargs** – passed to BaseSampler.run()

Return type

LightPointKD

3.8.10 SunSamplerPtView

class raytraverse.sampler.**SunSamplerPtView**(*scene*, *engine*, *sun*, *sunbin*, ***kwargs*)

Bases: *SamplerPt*

sample view rays to a source.

Parameters

- **scene** (raytraverse.scene.Scene) – scene class containing geometry, location and analysis plane
- **sun** (*np.array*) – the direction to the source
- **sunbin** (*int*) – index for naming

ub = 1

deterministic sample draws

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

sample a single point, position index handles file naming

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **mapper** (raytraverse.mapper.ViewMapper) – view direction to sample
- **lpargs** (*dict, optional*) – keyword arguments forwarded to LightPointKD construction
- **kwargs** – passed to BaseSampler.run()

Return type

LightPointKD

3.8.11 ImageSampler

class raytraverse.sampler.**ImageSampler**(*scene*, *vm=None*, *scalefac=None*, *method='linear'*, *color=False*, ***kwargs*)

Bases: *SamplerPt*

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

3.8.12 DeterministicImageSampler

class raytraverse.sampler.**DeterministicImageSampler**(*scene*, *vm=None*, *scalefac=None*, *method='linear'*, *color=False*, ***kwargs*)

Bases: *ImageSampler*

ub = 1

upper bound for drawing from pdf passed to raytraverse.sampler.draw.from_pdf()

run(*point*, *posidx*, *mapper=None*, *lpargs=None*, ***kwargs*)

sample a single point, position index handles file naming

Parameters

- **point** (*np.array*) – point to sample
- **posidx** (*int*) – position index
- **mapper** (raytraverse.mapper.ViewMapper) – view direction to sample
- **lpargs** (*dict, optional*) – keyword arguments forwarded to LightPointKD construction
- **kwargs** – passed to BaseSampler.run()

Return type

LightPointKD

3.9 raytraverse.lightpoint

3.9.1 LightPointKD

class raytraverse.lightpoint.**LightPointKD**(*scene*, *vec=None*, *lum=None*, *vm=None*, *pt=(0, 0, 0)*, *posidx=0*, *src='sky'*, *srcn=1*, *srcdir=(0, 0, 1)*, *calcomega=True*, *write=True*, *omega=None*, *filterviews=True*, *srcviews=None*, *parent=None*, *srcviewidxs=None*, *features=1*)

Bases: object

light distribution from a point with KDtree structure for directional query

Parameters

- **scene** (raytraverse.scene.BaseScene) –
- **vec** (*np.array, optional*) – shape (N, >=3) where last three columns are normalized direction vectors of samples. If not given, tries to load from scene.outdir
- **lum** (*np.array, optional*) – reshapeable to (N, srcn). sample values for each source corresponding to vec. If not given, tries to load from scene.outdir
- **vm** (raytraverse.mapper.ViewMapper*, optional*) – a default viewmapper for image and metric calculations, should match viewmapper of sampler.run() if possible.
- **pt** (*tuple list np.array*) – 3 item point location of light distribution
- **posidx** (*int, optional*) – index position of point, will govern file naming so must be set to avoid clobbering writes. also used by spacemapper for planar sampling
- **src** (*str, optional*) – name of source group. will govern file naming so must be set to avoid clobbering writes.
- **srcn** (*int, optional*) – must match lum, does not need to be set if reloading from scene.outdir

- **calcomega** (*bool, optional*) – if True (default) calculate solid angle of rays. This is unnecessary if point will be combined before calculating any metrics. setting to False will save some computation time.
- **write** (*bool, optional*) – whether to save ray data to disk.
- **omega** (*np.array, optional*) – provide precomputed omega values, if given, overrides calcomega

`vm`
raytraverse.mapper.ViewMapper

`scene`
raytraverse.scene.Scene

`posidx`
index for point

Type
int

`pt`
point location

Type
np.array

`src`
source key

Type
str

`file`
relative path to disk storage

Type
str

`srcdir`
direction to source(s)

`load()`

`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

`set_srcviews`(*srcviews*, *idxs=None*)

`calc_omega`(*write=True*)
calculate solid angle

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

`apply_coef`(*coefs*)
apply coefficient vector to self.lum

Parameters
coefs (*np.array int float list*) – shape (N, self.srcn) or broadcastable

Returns
alum – shape (N, self.vec.shape[0])

Return type
np.array

`add_to_img`(*img*, *vecs*, *mask=None*, *skyvec=1*, *interp=False*, *idx=None*, *interpweights=None*, *omega=False*, *vm=None*, *rnd=False*, *engine=None*, ***kwargs*)
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, optional*) – indices to img that correspond to vec (in case where whole image is not being updated, such as corners of fisheye)
- **skyvec** (*int float np.array, optional*) – source coefficients, shape is (1,) or (srcn,)
- **interp** (*Union[bool, str], optional*) –
  - if "precomp", use index and interpweights
  - if True and engine is None, linearinterpolation
  - if "fastc" and engine: uses content_interp (best after sampling w/o detail)
  - if "highc" and engine: uses content_interp_wedge (best after sampling w/o detail)
  - if "fast": use interp_fast (pair with sampling w/ detail)
  - if "high": use interp_wedge (pair with sampling w/ detail)
- **idx** (*np.array, optional*) – precomputed query/interpolation result
- **interpweights** (*np.array, optional*) – precomputted interpolation weights
- **omega** (*bool*) – if true, add value of ray solid angle instead of luminance
- **vm** (*raytraverse.mapper.ViewMapper, optional*) –
- **rnd** (*bool, optional*) – use random values as contribution (for visualizing data shape)
- **engine** (*raytraverse.renderer.Rtrace, optional*) – engine for content aware interpolation
- **kwargs** (*dict, optional*) – passed to interpolationn functions

`evaluate`(*skyvec*, *vm=None*, *idx=None*, *srconly=False*, *blursun=False*, *includeviews=True*)

return rays within view with skyvec applied. this is the analog to add_to_img for metric calculations

Parameters

- **skyvec** (*int float np.array, optional*) – source coefficients, shape is (1,) or (srcn,)
- **vm** (*raytraverse.mapper.ViewMapper, optional*) –
- **idx** (*np.array, optional*) – precomputed query_ball result
- **srconly** (*bool, optional*) – only evaluate direct sources (stored in self.srcviews)
- **includeviews** (*bool, optional*) – include src views in returned results

Returns

- **rays** (*np.array*) – shape (N, 3) rays falling within view
- **omega** (*np.array*) – shape (N,) associated solid angles
- **lum** (*np.array*) – shape (N,) associated luminances

`query_ray`(*vecs*)

return the index and distance of the nearest ray to each of vecs

Parameters

vecs (*np.array*) – shape (N, 3) normalized vectors to query, could represent image pixels for example.

Returns

- **i** (*np.array*) – integer indices of closest ray to each query
- **d** (*np.array*) – distance (corresponds to chord length on unit sphere) from query to ray in lightpoint. use translate.chord2theta to convert to angle.

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

return set of rays within a view cone

Parameters

- **vecs** (*np.array*) – shape (N, 3) vectors to query.
- **viewangle** (*int float*) – opening angle of view cone

Returns

i – if vecs is a single point, a list of vector indices of rays within view cone. if vecs is a set of point an array of lists, one for each vec is returned.

Return type

list np.array

`make_image`(*outf*, *skyvec*, *vm=None*, *res=1024*, *interp=False*, *showsample=False*)

`direct_view`(*res=512*, *showsample=False*, *showweight=True*, *rnd=False*, *srcidx=None*, *interp=False*, *omega=False*, *scalefactor=1*, *vm=None*, *fisheye=True*)

create an unweighted summary image of lightpoint

`add`(*lf2*, *src=None*, *calcomega=True*, *write=False*, *sumsrc=False*)

add light points of distinct sources together results in a new lightpoint with srcn=self.srcn+srcn2 and vector size=self.vecsize+vecsize2

Parameters

- **lf2** (*raytraverse.lightpoint.LightPointKD*) –
- **src** (*str, optional*) – if None (default), src is "{lf1.src}_{lf2.src}"
- **calcomega** (*bool, optional*) – passed to LightPointKD constructor

- **write** (*bool, optional*) – passed to LightPointKD constructor
- **sumsrc** (*bool, optional*) – if True adds matching source indices together (must be same shape) this assumes that the two lightpoints represent the same source but different components (such as direct/indirect)

Returns
will be subtyped according to self, unless lf2 is needed to preserve data

Return type
raytraverse.lightpoint.LightPointKD

update(*vec*, *lum*, *omega=None*, *calcomega=True*, *write=True*, *filterviews=False*)
add additional rays to lightpoint in place

Parameters

- **vec** (*np.array, optional*) – shape (N, >=3) where last three columns are normalized direction vectors of samples.
- **lum** (*np.array, optional*) – reshapeable to (N, srcn). sample values for each source corresponding to vec.
- **omega** (*np.array, optional*) – provide precomputed omega values, if given, overrides calcomega
- **calcomega** (*bool, optional*) – if True (default) calculate solid angle of rays. This is unnecessary if point will be combined before calculating any metrics. setting to False will save some computation time. If False, resets omega to None!
- **write** (*bool, optional*) – whether to save updated ray data to disk.
- **filterviews** (*bool, optional*) – delete rays near sourceviews

linear_interp(*vm*, *srcvals*, *destvecs*)

static apply_interp(*i*, *srcvals*, *weights=None*)

content_interp_wedge(*rt*, *destvecs*, *bandwidth=10*, *srfnormtol=5.0*, *disttol=0.5*, *oversample=2*, ***kwargs*)

interp_wedge(*destvecs*, *bandwidth=5*, ***kwargs*)

content_interp(*rt*, *destvecs*, *bandwidth=10*, *srfnormtol=5.0*, *disttol=0.5*, ***kwargs*)

interp_fast(*destvecs*, *bandwidth=10*, ***kwargs*)

3.9.2 SrcViewPoint

class raytraverse.lightpoint.**SrcViewPoint**(*scene*, *vecs*, *lum*, *pt=(0, 0, 0)*, *posidx=0*, *src='sunview'*, *res=64*, *srcomega=6.796702357283834e-05*, *isdistant=True*)

Bases: `object`

interface for sun view data

static offset(*points*, *target*)

scene
raytraverse.scene.Scene

posidx
index for point

Type
int

pt

point location

Type
np.array

src

source key

Type
str

raster

individual vectors that hit the source (pixels)

Type
np.array

lum

source luminance (average)

Type
float

radius

source radius

Type
float

isdistant = True

property vm

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

evaluate(*sunval*, *vm=None*, *blursun=False*)

direct_view(*res=80*)

3.9.3 CompressedPointKD

class raytraverse.lightpoint.**CompressedPointKD**(*scene*, *vec=None*, *lum=None*, *write=True*, *src=None*, *dist=0.0981*, *lerr=0.01*, *plotc=False*, ***kwargs*)

Bases: *LightPointKD*

compressed data needs special methods for making images.

can be initialized either like LightPointKD (but with required omega argument), or if 'scene' is a LightPointKD then a compressed output is calculated from the input

Parameters

- **scene** (*BaseScene LightpointKD*) –
- **src** (*str, optional*) – new name for src passed to LightPointKD constructor
- **dist** (*float, optional*) – translate.theta2chord(np.pi/32), primary clustering distance using the birch algorithm, for lossy compression of lf. this is the maximum radius of a cluster, preserving important directional information. clustering acts on ray direction and luminance, with weight of luminance dimension controlled by the lweight parameter.
- **lerr** (*float, optional*) – min-max normalized error in luminance grouping.

- **plotc** (*bool, optional*) – make directview plot of compressed output showing source vectors

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

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, optional*) – indices to img that correspond to vec (in case where whole image is not being updated, such as corners of fisheye)
- **skyvec** (*int float np.array, optional*) – source coefficients, shape is (1,) or (srcn,)
- **vm** (*raytraverse.mapper.ViewMapper, optional*) –

compress(*lp*, *src=None*, *dist=0.0981*, *lerr=0.01*)

A lossy compression based on clustering. Rays are clustered using the birch algoritm on a 4D vector (x,y,z,lum) where lum is the sum of contributions from all sources in the LightPoint. In the optional second stage (activated with secondary=True) sources are further grouped through agglomerative cluster using an average linkage. this is to help with source indentification/matching between LightPoints, but can introduce significant errors to computing non energy conserving metrics in cases where the applied sky vectors have large relative differences between adjacent patches (> 1.5:1) or if the variance in peak luminance above the lthreshold parameter is significant. These include cases where nearby transmitting materials is varied (example: a trans upper above a clear lower), or lthreshold is set too low. For this reason, it is better to use single stage compression for metric computation and only do glare source grouping for interpolation between LightPoints.

Parameters

- **lp** (*LightPointKD*) –
- **src** (*str, optional*) – new name for src passed to LightPointKD constructor
- **dist** (*float, optional*) – translate.theta2chord(np.pi/32), primary clustering distance using the birch algorithm, for lossy compression of lf. this is the maximum radius of a cluster, preserving important directional information. clustering acts on ray direction and luminance, with weight of luminance dimension controlled by the lweight parameter.
- **lerr** (*float, optional*) – min-max normalized error in luminance grouping.
- **plotc** (*bool, optional*) – make directview plot of compressed output showing source vectors

Return type

arguments for initializing a CompressedPointKD

3.10 raytraverse.lightfield

3.10.1 LightField

class raytraverse.lightfield.**LightField**(*scene*, *vecs*, *pm*, *src*)

Bases: object

collection of light data with KDtree structure for spatial query

Parameters

- **scene** (*raytraverse.scene.BaseScene*) –

- **vecs** (*np.array str*) – the vectors used to organizing the child data as array or file shape (N,3) or (N,4) if 3, indexed from 0
- **pm** (raytraverse.mapper.PlanMapper) –
- **src** (*str*) – name of source group.

property samplelevel

the level at which the vec was sampled (all zero if not provided upon initialization

property vecs

indexing vectors (such as position, sun positions, etc.)

property data

light data

property kd

kdtree for spatial queries built on demand

property omega

solid angle or area

query(*vecs*)

return the index and distance of the nearest point to each of points

Parameters
vecs (*np.array*) – shape (N, 3) vectors to query.

Returns

- **i** (*np.array*) – integer indices of closest ray to each query
- **d** (*np.array*) – distance from query to point in spacemapper.

evaluate(**args*, ***kwargs*)

3.10.2 LightPlaneKD

class raytraverse.lightfield.**LightPlaneKD**(*scene*, *vecs*, *pm*, *src*)

Bases: *LightField*

collection of lightpoints with KDtree structure for positional query

property data

LightPointSet

property omega

representative area of each point

Getter
Returns array of areas

Setter
sets areas

Type
np.array

evaluate(*skyvec*, *points=None*, *vm=None*, *metricclass=<class 'raytraverse.evaluate.metricset.MetricSet'>*, *metrics=None*, *mask=True*, ***kwargs*)

make_image(*outf*, *vals*, *res=1024*, *interp=False*, *showsample=False*)

make an image from precomputed values for every point in LightPlane

Parameters

- **outf** (*str*) – the file to write
- **vals** (*np.array*) – shape (len(self.points),) the values computed for each point
- **res** (*int, optional*) – image resolution (the largest dimension
- **interp** (*bool, optional*) – apply linear interpolation, points outside convex hull of results fall back to nearest
- **showsample** (*bool, optionaal*) – color pixel at sample location red

direct_view(*res=512*, *showsample=True*, *vm=None*, *area=False*, *metricclass=<class 'raytraverse.evaluate.metricset.MetricSet'>*, *metrics=('avglum',)*, *interp=False*)

create a summary image of lightplane showing samples and areas

3.10.3 SunsPlaneKD

class raytraverse.lightfield.**SunsPlaneKD**(*scene*, *vecs*, *pm*, *src*)

Bases: *LightField*

collection of lightplanes with KDtree structure for sun position query

property vecs

indexing vectors (sx, sy, sz, px, py, pz)

property suns

property data

LightPlaneSet

property kd

kdtree for spatial queries built on demand

property sunkd

kdtree for sun position queries built on demand

query(*vecs*)

return the index and distance of the nearest vec to each of vecs

Parameters

vecs (*np.array*) – shape (N, 6) vectors to query.

Returns

- **i** (*np.array*) – integer indices of closest ray to each query
- **d** (*np.array*) – distance from query to point, positional distance is normalized by the average chord-length between level 0 sun samples divided by the average distance between level 0 pt samples.

query_by_sun(*sunvec*, *fixed_points=None*, *stol=10*, *minsun=1*)

for finding vectors across zone, sun vector based query

Parameters

- **sunvec** (*Sequence*) – sun direction vector (normalized, xyz)
- **fixed_points** (*Sequence, optional*) – 2d array like, shape (N, 3) of additional fixed points to return use for example with a matching sky query. Note that if point filter is to large not all of these points are necessarily returned.

- **stol** (*Union[float, int], optional*) – maximum angle (in degrees) for matching sun vectors
- **minsun** (*int, optional*) – if atleast these many suns are not returned based on stol, directly query for this number of results (regardless of sun error)

Returns

- **vecs** (*np.array*) – shape (N, 6) final vectors, because of fixed_points, this may not match exactly with self.vecs[i] so this array mus be used in further processing
- **i** (*np.array*) – integer indices of the closest rays to each query
- **d** (*np.array*) – angle (in degrees) between queried sunvec and returned index

query_by_suns(*sunvecs*, *fixed_points=None*, *stol=10*, *minsun=1*)

parallel processing call to query_by_sun for 2d array of sunvecs

Parameters

- **sunvecs** (*np.array*) – shape (N, 3) sun direction vectors (normalized, xyz)
- **fixed_points** (*Sequence, optional*) – 2d array like, shape (N, 3) of additional fixed points to return use for example with a matching sky query. Note that if point filter is to large not all of these points are necessarily returned.
- **stol** (*Union[float, int], optional*) – maximum angle (in degrees) for matching sun vectors
- **minsun** (*int, optional*) – if atleast these many suns are not returned based on stol, directly query for this number of results (regardless of sun error)

Returns

- **vecs** (*list*) – list of np.array, one for each sunvec (see query_by_sun)
- **idx** (*list*) – list of np.array, one for each sunvec (see query_by_sun)
- **d** (*list*) – list of np.array, one for each sunvec (see query_by_sun)

3.10.4 SensorPlaneKD

class raytraverse.lightfield.**SensorPlaneKD**(*scene*, *vecs*, *pm*, *src*)

Bases: *LightPlaneKD*

collection of sensor results with KDtree structure for positional query

data has shape (pts, sensors, sources, bands)

property sensors

property vecs

indexing vectors (such as position, sun positions, etc.)

property data

light data

static apply_coef(*data*, *coefs*)

apply coefficient vector to data

Parameters

- **data** (*np.array*) – ndims should match self.data (N, sensors, nsrcs, nfeatures)
- **coefs** (*np.array int float list*) – shape (L, self.srcn) or broadcastable

Returns

alum – shape (L, N, sensors, nfeatures)

Return type
np.array

evaluate(*skyvec*, *points=None*, *sensoridx=None*, *mask=True*, ***kwargs*)

direct_view(*res=512*, *showsample=True*, *area=False*, *interp=False*, *sensoridx=None*, ***kwargs*)
create a summary image of lightplane showing samples and areas

3.10.5 SunSensorPlaneKD

class raytraverse.lightfield.**SunSensorPlaneKD**(*scene*, *vecs*, *pm*, *src*)
Bases: *SunsPlaneKD*

collection of sensorplanes with KDtree structure for sun position query

data has shape (pts * suns, sensors, sources, bands)

property sensors

property suns

property data
LightPlaneSet

static apply_coef(*data*, *coefs*)
apply coefficient vector to data

Parameters

- **data** (*np.array*) – ndims should match self.data (N, M, nsrcs, nfeatures)
- **coefs** (*np.array int float list*) – shape (L, self.srcn) or broadcastable

Returns
alum – shape (L, N, M, nfeatures)

Return type
np.array

3.10.6 LightResult

class raytraverse.lightfield.**LightResult**(*data*, **axes*)
Bases: object

a dense representation of lightfield data analyzed for a set of metrics

this class handles writing and loading results to disk as binary data and intuitive result extraction and reshaping for downstream visualisation and analysis using one of the "pull" methods. axes are indexed both numerically and names for increased transparency and ease of use.

Parameters

- **data** (*np.array str*) – multidimensional array of result data or file path to saved LightResule
- **axes** (*Sequence[raytraverse.lightfield.ResultAxis]*) – axis information

property data

property axes

property names

property file

`axis`(*name*)

`load`(*file*)

`write`(*file*, *compressed=True*)

`pull`(**axes*, *preserve=1*, ***kwargs*)

arrange and extract data slices from result.

Integrators construct a light result with these axes:

0. sky
1. point
2. view
3. metric

Parameters

- **axes** (`Union[int, str]`) – the axes (by name or integer index) to reorder output, list will fill with default object order.
- **preserve** (`int, optional`) – number of dimensions to preserve (result will be N+1).
- **kwargs** (`dict, optional`) – keys with axis names will be used to filter output.

Returns

- **result** (*np.array*) – the result array, will have 1+len(axes) dims, with the shaped determined by axis size and any indices argument.
- **labels** (*Sequence*) – list of labels for each axis, for flattened axes will be a tuple of broadcast axis labels.
- **names** (*Sequence*) – list of strings of returned axis names

static `row_labels`(*labels*)

static `fmt_names`(*name*, *labels*)

`pull_header`(*names*, *labels*, *rowlabel=True*)

`print`(*col*, *header=True*, *rowlabel=True*, *file=None*, *skyfill=None*, ***kwargs*)

first calls pull and then prints 2d result to file

`sky_percentile`(*metric*, *per=(50,)*, ***kwargs*)

`print_serial`(*col*, *basename*, *header=True*, *rowlabel=True*, *skyfill=None*, ***kwargs*)

print 3d result to series of 2d files

`pull2hdr`(*col*, *basename*, *skyfill=None*, *spd=24*, *pm=None*, ***kwargs*)

`info`()

3.10.7 ZonalLightResult

class raytraverse.lightfield.**ZonalLightResult**(*data*, **axes*, *pointmetrics=None*)

Bases: *LightResult*

a semi-dense representation of lightfield data analyzed for a set of metrics

this class handles writing and loading results to disk as binary data and intuitive result extraction and reshaping for downstream visualisation and analysis using one of the "pull" methods. axes are indexed both numerically and names for increased transparency and ease of use.

property data

load(*file*)

write(*file*, *compressed=True*)

pull2hdr(*imgzone*, *basename*, *showsample=False*, ***kwargs*)

3.10.8 sets

LightSet

class raytraverse.lightfield.sets.**LightSet**(*dataclass*, *scene*, *points*, *idx*, ***kwargs*)

Bases: object

LightPointSet

class raytraverse.lightfield.sets.**LightPointSet**(*scene*, *points*, *idx*, *src*, *parent*)

Bases: *LightSet*

a collection of LightPoints, initialized by getitem

MultiLightPointSet

class raytraverse.lightfield.sets.**MultiLightPointSet**(*scene*, *points*, *idx*, *src*, *parent*)

Bases: *LightSet*

SensorPointSet

class raytraverse.lightfield.sets.**SensorPointSet**(*data*, *idx*, ***kwargs*)

Bases: *LightSet*

3.10.9 RaggedResult

class raytraverse.lightfield.**RaggedResult**(*a*)

Bases: tuple

has a shape parameter and indexing similar to a np.array, but with varying shape along the second axis. composed of a list of np.arrays whose shape match after the first dimension.

3.10.10 ResultAxis

class raytraverse.lightfield.**ResultAxis**(*values*, *name*, *cols=None*)

Bases: object

value_array()

property cols

3.11 raytraverse.integrator

3.11.1 Integrator

class raytraverse.integrator.**Integrator**(**lightplanes*, *includesky=True*, *includesun=True*, *sunviewengine=None*, *ds=False*, *dv=False*)

Bases: object

collection of lightplanes with KDtree structure for sun position query

Parameters
lightplanes (*Sequence[raytraverse.lightfield.LightPlaneKD]*) –

make_images(*skydata*, *points*, *vm*, *viewangle=180.0*, *res=512*, *interp=False*, *prefix='img'*, *namebyindex=False*, *suntol=10.0*, *blursun=False*, *resamprad=0.0*)

see namebyindex for file naming conventions

Parameters

- **skydata** (*raytraverse.sky.Skydata*) –
- **points** (*np.array*) – shape (N, 3)
- **vm** (*Union[raytraverse.mapper.ViewMapper, np.array]*) – either a predefined ViewMapper (used for all points) or an array of view directions (will use a 180 degree view angle when initializing ViewMapper)
- **viewangle** (*float, optional*) – view opening for sensor (0-180,360) when vm is given as an array of view directions.
- **res** (*int, optional*) – image resolution
- **interp** (*bool, optional*) – interpolate image
- **prefix** (*str, optional*) – prefix for output file naming
- **namebyindex** (*bool, optional*) – if False (default), names images by: <prefix>_sky-<row>_pt-<x>_<y>_<z>_vd-<dx>_<dy>_<dz>.hdr if True, names images by: <prefix>_sky-<row>_pt-<pidx>_vd-<vidx>.hdr, where pidx, vidx are refer to the order of points, and vm.

Return type
np.array of out_files shape (skies, points, views)

evaluate(*skydata*, *points*, *vm*, *viewangle=180.0*, *metricclass=<class 'raytraverse.evaluate.metricset.MetricSet'>*, *metrics=None*, *datainfo=False*, *srconly=False*, *suntol=10.0*, *blursun=False*, *coercesumsafe=False*, *stol=10*, *minsun=1*, ***kwargs*)

apply sky data and view queries to daylightplane to return metrics parallelizes and optimizes run order.

Parameters

- **skydata** (*raytraverse.sky.Skydata*) –
- **points** (*np.array*) – shape (N, 3)

- **vm** (*Union[raytraverse.mapper.ViewMapper, np.array]*) – either a predefined ViewMapper (used for all points) or an array of view directions (will use 'viewangle' when initializing ViewMapper)
- **viewangle** (*float, optional*) – view opening for sensor (0-180,360) when vm is given as an array of view directions, note that for illuminance based metrics, a value of 360 may not make sense as values behind will be negative.
- **metricclass** (*raytraverse.evaluate.BaseMetricSet, optional*) –
- **metrics** (*Sized, optional*) –
- **srconly** (*bool, optional*) – sun only calculations
- **suntol** (*float, optional*) – if Integrator has an engine, resample sun views when actual sun position error is greater than this many degrees.
- **blursun** (*bool, optional*) – apply human PSF to small bright sources
- **coercesumsafe** (*bool, optional*) – attempt to calculate sumsafe metrics
- **datainfo** (*Union[Sized[str], bool], optional*) – include information about source data as additional metrics. Valid values include: ["pt_err", "pt_idx", "src_err", "src_idx"]. If True, includes all.
- **stol** (*Union[float, int], optional*) – maximum angle (in degrees) for matching sun vectors (zonal)
- **minsun** (*int, optional*) – if atleast these many suns are not returned based on stol, directly query for this number of results (regardless of sun error) (zonal)

Return type

raytraverse.lightfield.LightResult

zonal_evaluate(*skydata*, *pm*, *vm*, *viewangle=180.0*, *metricclass=<class 'raytraverse.evaluate.metricset.MetricSet'>*, *metrics=None*, *srconly=False*, *suntol=10.0*, *blursun=False*, *coercesumsafe=False*, *stol=10*, *minsun=1*, *datainfo=False*, ***kwargs*)

apply sky data and view queries to daylightplane to return metrics parallelizes and optimizes run order.

Parameters

evaluate (*see*) –

Return type

raytraverse.lightfield.ZonalLightResult

3.11.2 helpers

parallelization functions for integration

raytraverse.integrator.helpers.**evaluate_pt**(*lpts*, *skyvecs*, *suns*, *vm=None*, *vms=None*, *metricclass=None*, *metrics=None*, *srconly=False*, *sumsafe=False*, *suntol=1.0*, *svengine=None*, *blursun=False*, *refl=None*, *resamprad=0.0*, ***kwargs*)

point by point evaluation suitable for submitting to ProcessPool

raytraverse.integrator.helpers.**img_pt**(*lpts*, *skyvecs*, *suns*, *vms=None*, *combos=None*, *qpts=None*, *skinfo=None*, *res=512*, *interp=False*, *prefix='img'*, *suntol=1.0*, *svengine=None*, *refl=None*, *resamprad=0.0*, ***kwargs*)

point by point evaluation suitable for submitting to ProcessPool

raytraverse.integrator.helpers.**prep_ds**(*lpts*, *skyvecs*)

raytraverse.integrator.helpers.**evaluate_pt_ds**(*lpts*, *skyvecs*, *suns*, ***kwargs*) raytraverse.integrator.helpers.**img_pt_ds**(*lpts*, *skyvecs*, *suns*, ***kwargs*)

raytraverse.integrator.helpers.**evaluate_pt_dv**(*lpts*, *skyvecs*, *suns*, ***kwargs*)

raytraverse.integrator.helpers.**img_pt_dv**(*lpts*, *skyvecs*, *suns*, ***kwargs*)

raytraverse.integrator.helpers.**prep_resamp**(*lpts*, *refl=None*, *resamprad=0.0*)

raytraverse.integrator.helpers.**update_src_view**(*engine*, *lpt*, *sun*, *vm=None*, *tol=1.0*, *refl=None*, *resampvecs=None*, *reflarea=None*, *resamprad=0.0*)

raytraverse.integrator.helpers.**apply_dsky_patch**(*skp*, *skd*, *skyvecs*, *skdir*, *dirlum=None*)

3.12 raytraverse.evaluate

3.12.1 BaseMetricSet

class raytraverse.evaluate.**BaseMetricSet**(*vec*, *omega*, *lum*, *vm*, *metricset=None*, *scale=179.0*, *omega_as_view_area=True*, *guth=True*, *warnings=False*, ***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
- **omega_as_view_area** (*bool, optional*) – take sum(omega) as view area. if false corrects omega to vm.area
- **warnings** (*bool, optional*) – if False, suppresses numpy warnings (zero div, etc…) when accessed via __call__
- **kwargs** – additional arguments that may be required by additional properties

allmetrics = ['illum', 'avglum', 'loggcr', 'gcr', 'pwgcr', 'logpwgcr', 'density', 'avgraylum', 'pwavglum', 'maxlum']

safe2sum = {'avglum', 'density', 'illum'}

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

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

classmethod check_safe2sum(*metrics*)

checks if list if metrics is safe to compute for seperate sources before adding

property vec

property lum

property omega

property ctheta

cos angle between ray and view

property radians

angle between ray and view

property pos_idx

property pweight

property pweighted_area

property illum

illuminance

property avglum

average luminance

property maxlum

average luminance

property pwavglum

position weighted 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 pwgcr

a unitless measure of relative contrast defined as the average of the squared luminances divided by the average luminance squared weighted by a position index

property logpwgcr

a unitless measure of relative contrast defined as the log of gcr

property loggcr

a unitless measure of relative contrast defined as the log of gcr

property density

3.12.2 MultiLumMetricSet

class raytraverse.evaluate.**MultiLumMetricSet**(*vec*, *omega*, *lum*, *vm*, *metricset=None*, *scale=179.0*, *omega_as_view_area=True*, ***kwargs*)

Bases: *BaseMetricSet*

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, M) 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
- **kwargs** – additional arguments that may be required by additional properties

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

3.12.3 MetricSet

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

Bases: *BaseMetricSet*

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

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

allmetrics = ['illum', 'avglum', 'loggcr', 'gcr', 'pwgcr', 'logpwgcr', 'density', 'avgraylum', 'pwavglum', 'maxlum', 'ugp', 'dgp', 'tasklum', 'backlum', 'dgp_t1', 'log_gc', 'dgp_t2', 'ugr', 'threshold', 'pwsl2', 'view_area', 'backlum_true', 'srcillum', 'srcarea', 'maxlum']

safe2sum = {'avglum', 'density', 'illum', 'pwsl2', 'srcillum'}

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
source illuminance

property srcarea
total source area

property maxlum
peak 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 dgp

property dgp_t1

property log_gc

property dgp_t2

property ugr

property ugp
//dx.doi.org/10.1016/j.buildenv.2016.08.005

Type
http

3.12.4 FieldMetric

class raytraverse.evaluate.**FieldMetric**(*vec*, *omega*, *lum*, *vm=None*, *scale=1.0*, *npts=360*, *close=True*, *sigma=0.05*, *omega_as_view_area=True*, ***kwargs*)

Bases: *BaseMetricSet*

calculate metrics on full spherical point clouds rather than view based metrics.

Parameters

- **vec** (*np.array*) – (N, 3) directions of all rays
- **omega** (*np.array*) – (N,) solid angle of all rays
- **lum** (*np.array*) – (N,) luminance of all rays (multiplied by "scale")
- **metricset** (*list, optional*) – keys of metrics to return, same as property names
- **scale** (*float, optional*) – scalefactor for luminance
- **npts** (*int, optional*) – for equatorial metrics, the number of points to interpolate
- **close** (*bool, optional*) – include npts+1 duplicate to draw closed curve
- **sigma** (*float, optional*) – scale parameter of gaussian for kernel estimated metrics
- **omega_as_view_area** (*bool, optional*) – set to true when vectors either represent a whole sphere or a subset that does not match the viewmapper. if False, corrects boundary omega to properly trim to correct size.
- **kwargs** – additional arguments that may be required by additional properties

property tp

vectors in spherical coordinates

property phi

interpolated output phi values

property eq_xyz

interpolated output xyz vectors

property avg

overall vector (with magnitude)

property peak

overall vector (with magnitude)

property eq_lum

luminance along an interpolated equator with a bandwidth=sigma

property eq_density

ray density along an interpolated equator

property eq_illum

illuminiance along an interpolated equator

property eq_gcr

cosine weighted gcr along an interpolated equator

property eq_loggc

property eq_dgp

3.12.5 SamplingMetrics

class raytraverse.evaluate.**SamplingMetrics**(*vec*, *omega*, *lum*, *vm*, *scale=1.0*, *peakthreshold=0.0001*, *lmin=0*, *gcrnorm=8*, ***kwargs*)

Bases: *BaseMetricSet*

default metricset for areasampler

defaultmetrics = ['avglum', 'loggcr', 'xpeak', 'ypeak']

available metrics (and the default return set)

allmetrics = ['avglum', 'loggcr', 'xpeak', 'ypeak']

property peakvec

average vector (with magnitude) for peak rays

property xpeak

x-component of avgvec as positive number (in range 0-1)

property ypeak

y-component of avgvec as positive number (in range 0-1)

property loggcr

log of global contrast ratio

3.12.6 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.array

positions_vec(*viewvec*, *srcvec*, *up=(0, 0, 1)*)

3.12.7 retina

raytraverse.evaluate.retina.**hpsf**(*x*, *fwhm=0.183333*)

estimate of human eye point-spread function

from: Yang, Yr., Wanek, J. & Shahidi, M. Representing the retinal line spread shape with mathematical functions. J. Zhejiang Univ. Sci. B 9, 996–1002 (2008). https://doi.org/10.1631/jzus.B0820184

raytraverse.evaluate.retina.**inv_hpsf**(*y*, *fwhm=0.183333*)

inverse of hpsf

raytraverse.evaluate.retina.**blur_sun**(*omega*, *lmax*, *lmin=279.33*, *fwhm=0.183333*)

calculate source correction to small bright source

returned value should be multiplied by omega and divides luminance

Parameters

- **omega** (*Union[float, np.array]*) – solid angle in steradians of source
- **lmax** (*Union[float, np.array]*) – maximum radiance in source (cd/m^2)/179
- **lmin** (*Union[float, np.array], optional*) – minimum radiance value to gather after spread (mimic peak extraction of evalglare, but note the different units (cd/m^2)/179
- **fwhm** (*Union[float, np.array], optional*) – full width half max of Lorentzian curve (radius in degrees) default is 11 arcmin.

Returns

correction factor – value should be multiplied by omega and divides luminance

Return type

Union[float, np.array]

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

3.12.8 hvsgsm

raytraverse.evaluate.hvsgsm.**gss_compute**(*imgs*, *illums=None*, *save=False*, *suffix='_rg.hdr'*, *outdir=None*, ***kwargs*)

initialize a GSS instance and compute multiple images in parallel

Parameters

- **imgs** (*Sequence*) – list of image file paths to compute. images should by 180 degree HDR angular fisheyes scaled at 1/179 cd/m^2 (standard radiance HDR)
- **illums** (*Sequence, optional*) – If images onnly contain glare sources but not an accurate background provide illuminance calculated seperately (like eDGPs process)
- **save** (*bool, optional*) – If true saves an image of the glare response
- **suffix** (*str, optional*) – suffix to append to image when save is True
- **outdir** (*str, optional*) – save response images to a different directory
- **kwargs** – passed to GSS initialization

Returns
GSS – glare sensation scores for all images (in order given)

Return type
list

raytraverse.evaluate.hvsgsm.**process_gss**(*img*, *illum*, *ins*, *outf=False*, *outdir=None*, *suffix='_rg.hdr'*)

called by gss_compute in parallel

raytraverse.evaluate.hvsgsm.**f_b**(*b*, *c*, *phi*)

component of point spread function

J.K. Ijspeert, T.J.T.P. Van Den Berg, H. Spekreijse, An improved
mathematical description of the foveal visual point spread function with parameters for age, pupil size and pigmentation, Vision Research, Volume 33, Issue 1, 1993,Pages 15-20, ISSN 0042-6989, https://doi.org/10.1016/0042-6989(93)90053-Y.

raytraverse.evaluate.hvsgsm.**l_b**(*b*, *c*, *phi*)

component of line spread function

J.K. Ijspeert, T.J.T.P. Van Den Berg, H. Spekreijse, An improved
mathematical description of the foveal visual point spread function with parameters for age, pupil size and pigmentation, Vision Research, Volume 33, Issue 1, 1993,Pages 15-20, ISSN 0042-6989, https://doi.org/10.1016/0042-6989(93)90053-Y.

3.12.9 GSS

class raytraverse.evaluate.**GSS**(*view=None*, *gst=0*, *age=40*, *f=16.67*, *scale=179*, *pigmentation=0.106*, *fwidth=10*, *psf=True*, *adaptmove=True*, *directmove=True*, *raw=False*)

Bases: `object`

calculate GSS for images with angular fisheye projection

application of model described in:

A GENERIC GLARE SENSATION MODEL BASED ON THE HUMAN VISUAL SYSTEM Vissenberg, M.C.J.M., Perz, M., Donners, M.A.H., Sekulovski, D. Signify Research, Eindhoven, THE NETHERLANDS gilles.vissenberg@signify.com DOI 10.25039/x48.2021.0P23

see methods for citations associated with each step in model.

the model requires the following steps:

Done when setting an image with a new resolution: 1. calculate solid angle of pixels 2. calculate eccentricity from guth position idx

Steps for applying model to an image: 1. calculate eye illuminance from image 2. mask non-glare source pixels (not described in model, fixed thresh) 3. calculate pupil area and diameter 4. calculate global retinal irradiance 5. calculate incident retinal irradiance of glare sources 6. apply PSF to (5) 7. apply movement affecting adaptation to (6) 8. apply movement affecting direct response to (6) 9. calculate local adaptation using (7) 10. calculate V/V_m photoreceptor response (8) 11. calculate receptor field response to (10) as DoG 12. normalize field response with logistic 13. apply position weighting 14. sum GSS

Parameters

- **view** – can be None, a view file, a ViewMapperr, or an hdrimage with a valid view specification (must be -vta)
- **gst** – glare source threshold (cd/m^2)
- **age** – age of observer
- **f** – eye focal length
- **scale** – factor to apply to raw pixel values to convert to cd/m^2
- **pigmentation** –

  from Ijspeert et al. 1993:
  mean for blue eyes: 0.16 brown eyes: 0.106 dark brown eyes: 0.056
- **fwidth** (*Union[int, float], optional*) – the width of the frame for psf
- **psf** (*bool, optional*) – apply pointspread function for light arriving at retina
- **adaptmove** (*bool, optional*) – apply involuntary eye movement effect on local adaptation
- **directmove** (*book, optional*) – apply involuntary eye movement effect on direct cone response

Notes

set self.lum, either by initializing with an image, or with the parameter setter, then compute:

```
gss = GSS("img.hdr")
gss.lum = "img.hdr"
score = gss.compute()
```

additional images can be loaded and computed with the parameter setter by calling images with the same resolution and view size on an initialized object, subsantial re-computation can be avoided.

Alternatively, to get access to process arrays or to override pupil adaptation and or isolating glare sources:

```
e_g, pupa, pupd = self.adapt(ev_eye)
img_gs = self.get_glare_sources()
r_g, parrays = self.glare_response(img_gs, e_g, pupa, pupd,
return_arrays=True)
```

For processing multiple images with the same GSS initialization in parallel, see hvsgsm.gss_compute()

emax = 0.12

emin = 0.009

fr_a = 22

fr_b = 0.25

fr_k = 0.67

norm = 4

contrast = 0.8

adapt(*ev_eye=None*)
step 1 in compute, adapt eye to image

get_glare_sources()
step 2 in compute, isolate glare sources

glare_response(*img_gs*, *e_g*, *pupa*, *pupd*, *return_arrays=False*)
step 3 in compute, apply steps of Vissenberg et al. model

Parameters

- **img_gs** (*np.array*) – representing all glare sources
- **e_g** (*float*) – global retinal irradiance
- **pupa** (*float*) – pupil area (mm^2)
- **pupd** (*float*) – pupil diameter (mm)
- **return_arrays** (*bool, optional*) – if True returns second value with dict of process arrays else return r_w only

Returns

- **r_w** (*np.array*) – weighted glare response for entire retina as represented by image
- **parrays** (*dict, optional*) – with returned_arrays=True keys: retinal_irrad, psf, adapt_eye_movement, direct_eye_movement, local_adaptation, response_ratio, response_lin, response_log

compute(*save=None*, *ev_eye=None*)
apply glare sensation model to loaded image

Parameters

- **save** (*str*) – if given save response image to file specified (.hdr)
- **ev_eye** (*float, opttional*) – externally calculated Ev

Return type
float

property lum

property res
resolution, set via lum

property vecs
directions, set via lum

property omega
solid angle, set via lum

property mask
view mask, set via lum

property ctheta
cos between vectors and view direction, set via lum

property sigma_c

position index scaled to eccentricity .009-.12 (used in field_response)

Note that this differs from the implementation dscribed by Vissenberg et al., and uses ganglion cell field density from:

Andrew B. Watson; A formula for human retinal ganglion cell receptive field density as a function of visual field location. Journal of Vision 2014;14(7):15. doi: https://doi.org/10.1167/14.7.15.

property vm

pupil(*ev*)

calculate pupil area

Based on: Donners, Maurice & Vissenberg, Michel & Geerdinck, L.M. & Broek-Cools, J. (2015). A PSYCHOPHYSICAL MODEL OF DISCOMFORT GLARE IN BOTH OUTDOOR AND INDOOR APPLICATIONS.

Parameters

ev – illumiance at eye (lux)

retinal_irradiance(*lum*, *pupa*)

adjust incident light on retina based on pupil size and focal-length

from Vissenberg et al. 2021 equation (1): (1) E_r = A_p * L / f^2 E_r: local retinal irrradiance L: field luminance

prep_kernel()

construct an array to hold a kernel scaled to image resolution

psf_coef(*pupd*)

age, pupil size and pigmentation adjusted PSF coefficients

PSF:

PSF(phi) = sum(c * f_b(phi)) f_b(phi) = b/(2 * (sin^2(phi) + b^2*cos^2(phi))^1.5) 1/steradian

LSF:

LSF(phi) = sum(c * l_b(phi)) l_b(phi) = b/(* (sin^2(phi) + b^2*cos^2(phi))) 1/rad

based on: J.K. Ijspeert, T.J.T.P. Van Den Berg, H. Spekreijse, An improved mathematical description of the foveal visual point spread function with parameters for age, pupil size and pigmentation, Vision Research, Volume 33, Issue 1, 1993,Pages 15-20, ISSN 0042-6989, https://doi.org/10.1016/0042-6989(93)90053-Y.

apply_psf(*e_r*, *pupd*)

apply human foveal point spread function

based on: J.K. Ijspeert, T.J.T.P. Van Den Berg, H. Spekreijse, An improved mathematical description of the foveal visual point spread function with parameters for age, pupil size and pigmentation, Vision Research, Volume 33, Issue 1, 1993,Pages 15-20, ISSN 0042-6989, https://doi.org/10.1016/0042-6989(93)90053-Y.

apply_eye_movement_1(*e_r*)

eye movement gaussian adaptation model to blur image at the time- scale of adaptation response.

based on: R. A. Normann, B. S. Baxter, H. Ravindra and P. J. Anderton, "Photoreceptor contributions to contrast sensitivity: Applications in radiological diagnosis," in IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, pp. 944-953, Sept.-Oct. 1983, doi: 10.1109/TSMC.1983.6313090.

Parameters

e_r (*np.array*) – retinal irradiance (optical correction)

Returns

retinal irradiance (with adaptation scale movement and optical correction)

Return type
adapt_eye_movement

apply_eye_movement_2(*e_r*, *e_g*)

blur image due to eye movement during direct response

from Vissenberg et al. 2021 equations (5) and (6): (5) = 100/(E_g * f^2)^0.12 ms tau (): cone integration time

(6) w = 2 * sqrt(D *) D = 30.0 arcmin^2 * s^-1 (occular drift) D = 250.0 (micro saccades)

Parameters

- **e_r** (*np.array*) – retinal irradiance (optical correction)
- **e_g** (*float*) – global retinal irrradiance

Returns
retinal irradiance (with movement and optical correction)

Return type
direct_eye_movement

local_eye_adaptation(*e_r*, *e_g*)

calculate locallized eye adaptation

from Vissenberg et al. 2021 equation (4): log_10(E_a) = p * log_10(E_r) + (1-p) * log_10(E_g) E_a: adaptation illuminance p: 0.8 (indoor / moderate) - 0.9 (outdoor / strong) contrast

Parameters

- **e_r** (*np.array*) – retinal irradiance (optical correction)
- **e_g** (*float*) – global retinal irrradiance

Return type
local_adaptation

static cone_response(*e_r*, *e_a*)

calculate local response as a fraction of maximum at current adaptation

from Vissenberg et al. 2021 equations (2) and (3): (2) V/V_m = E_r^n / (E_r^n + ^n) V: photoreceptor response V_m: maximum response E_r: local retinal illuminance (apply w to this E_r) n: 0.74

(3) = (5.701055^(1/2.55) + E_a^(1/2.55))^2.55 sigma (): half-saturation retinal illuminance value

Parameters

- **e_r** (*np.array*) – retinal irradiance (with movement and optical correction)
- **e_a** (*np.array*) – local adaptation

Return type
response_ratio

field_response(*vvm*)

receptive field response

from Vissenberg et al. 2021 equation (7):

```
R_RF(r) = e^(-r^2/(2_c^2)) / (2_c^2)
          - K * e^(-r^2/(2_s^2)) / (2_s^2)
```

R_RF: receptive field response r: distance to receptive field center (degrees) _c: gaussian width of center (0.009 (center) - 0.12 (edge FOV) degrees) _s: gaussian width of surround 3.5 * _c K: DoG balance factor 0.67

Parameters
vvm (*np.array*) – response_ratio (saturation)

Returns
linear, difference of gussians

Return type
response_lin

`normalized_field_response`(*r*)
normalized non-linear ganglion response

from Vissenberg et al. 2021 equation (8): R_G = 1 / (1 + e^(-a * (R_lin - b))) R_G: normalized non-linear ganglion response a: slope of logistic = 22 b: 0.25

Parameters
r (*np.array*) – response_lin

Returns
logistic

Return type
response_log

`weight_response`(*r*)
weight rectified response by position index

Parameters
r (*np.array*) – response_log

Return type
position weighted glare response

Notes

fit on guth data using BCD = 2843.58 * e^(x + 1.5 * x^2) / 179 with a 2.12 degree source and 34.26 cd/m^2 background

numpy.polynomial.Polynomial.fit(x, y, 6) where x = eccentricity (.009 -.12 from 0 to 55 degree vertical angle and y = 1/unweighted GSS

results:

```
1.0598742512189994 - 0.9135529200712416·x¹ + 0.8471705621553406·x² -
0.5535443101789258·x³ - 0.38772352579868125·x⁴ + 0.9083844574646001·x⁵ -
0.07637393810523314·x⁶ - 0.3026419768162507·x⁷
```

`gss`(*r_g*)
calculate minkowski sum on normalized response

from Vissenberg et al. 2021 equation (9): (9) GSS = sum_i(R_G,i^m _i)^m GSS: glare sensation score m: minkowski norm (4) delta (): solid angle of pixel (steradians)

3.13 raytraverse.craytraverse

3.14 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()

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()

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

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

Return type
imgf

`raytraverse.io.hdr2array`(*imgf*, *stdin=None*)

read np.array from hdr image

Parameters

- **imgf** (*file path of image*) –
- **stdin** – passed to Popen (imgf should be "")

Returns
ar

Return type
np.array

`raytraverse.io.hdr2carray`(*imgf*, *stdin=None*)

read np.array from color hdr image

Parameters

- **imgf** (*file path of image*) –
- **stdin** – passed to Popen (imgf should be "")

Returns
ar

Return type
np.array

`raytraverse.io.rgb2rad`(*rgb*, *vlambda=(0.265, 0.67, 0.065)*)

`raytraverse.io.rgb2lum`(*rgb*, *vlambda=(0.265, 0.67, 0.065)*)

`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.load_txt`(*farray*, ***kwargs*)

consistent error handing of np.loadtxt

Parameters

- **farray** (*any*) – candidate to load
- **kwargs** – passed to np.loadtxt

Return type
np.array

Raises

- **ValueError:** – file exists, but is not loadable
- **FileNotFoundError:** – farray is str, but file does not exist
- **TypeError:** – farray is not str or bytes.

3.15 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=False)`

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)`

generate source vectors from sky bins

Parameters

- **bn** (*np.array*) – bin numbers
- **side** (*int*) – square side of discretization

Returns
xyz – direction to center of sky patches

Return type
np.array

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

xyz coordinates to xy mapping of angular fisheye proejection

`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.ctheta(a, b)`
: cos(theta) (dot product) between a and b

`raytraverse.translate.radians(a, b)`
: angle in radians betweeen a and b

`raytraverse.translate.degrees(a, b)`
: angle in degrees betweeen a and b

`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 (N, 3)*) – the vector direction representing the starting coordinate space

Returns
ymtx, pmtx – two rotation matrices to be premultiplied in order to reverse transform, swap order and transpose.

Return type
(np.array, np.array)

Notes

if N is one: Forward: (pmtx@(ymtx@xyz.T)).T or np.einsum("ij,kj,li->kl", ymtx, xyz, pmtx) Backward: (ymtx.T@(pmtx.T@xyz.T)).T or np.einsum("ji,kj,il-kl", pmtx, nv, ymtx) else: Forward: np.einsum("vij,vkj,vli->vkl", ymtx, xyz, pmtx) Backward: np.einsum("vji,vkj,vil-vkl", pmtx, nv, ymtx)

`raytraverse.translate.cull_vectors`(*vecs*, *tol*)

return mask to cull duplicate vectors within tolerance

Parameters

- **vecs** (*Union[cKDTree, np.array]*) – prebuilt KDTree or np.array to build a new one. culling keeps first vector in array used to build tree.
- **tol** (*float*) – tolerance for culling

Returns
boolean mask of vecs (or vecs.data) to cull vectors

Return type
np.array

`raytraverse.translate.reflect`(*ray*, *normal*, *returnmasked=False*)

`raytraverse.translate.simple_take`(*ar*, **slices*, *axes=None*)

consistent array indexing with arrays, lists, tuples and slices

Parameters

- **ar** (*np.array*) – the multidimensional arary to index
- **slices** (*tuple*) – if sequence, takes those indices along axis, if None, take whole dimension, if slice, applies to index array before take
- **axes** (*Union[Sequence, int], optional*) – when None, slices are automatically taken starting on axes 0. Use this argument to only operate on a subset of dimensions.

Returns
matches ndims of ar

Return type
np.array

raytraverse.translate.**calc_omega**(*vecs*, *pm*)
calculate area

3.16 raytraverse.utility

raytraverse.utility.utility.**pool_call**(*func*, *args*, **fixed_args*, *cap=None*, *expandarg=True*, *desc='processing'*, *workers=True*, *pbar=True*, ***kwargs*)
calls func for a sequence of arguments using a ProcessPool executor and a progress bar. result is equivalent to:

```
result = []
for arg in args:
    result.append(func(*args, *fixed_args, **kwargs))
return result
```

Parameters

- **func** (*callable*) – the function to execute in parallel
- **args** (*Sequence[Sequence]*) – list of arguments (each item is expanded with '*' unless expandarg is false). first N args of func
- **fixed_args** (*Sequence*) – arguments passed to func that are the same for all calls (next N arguments after args)
- **cap** (*int, optional*) – execution cap for ProcessPool
- **expandarg** (*bool, optional*) – expand args with '*' when calling func
- **desc** (*str, optional*) – label for progress bar
- **kwargs** – additional keyword arguments passed to func

Return type
sequence of results from func (order preserved)

3.16.1 imagetools

functions for translating from mappers to hdr

raytraverse.utility.imagetools.**hdr_uv2ang**(*imgf*)

raytraverse.utility.imagetools.**hdr_ang2uv**(*imgf*)

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

raytraverse.utility.imagetools.**hdr2uvarray**(*imgf*, *vm=None*, *res=None*)

raytraverse.utility.imagetools.**hdr2vol**(*imgf*, *vm=None*)

raytraverse.utility.imagetools.**vf_to_vm**(*view*)
view file to ViewMapper

raytraverse.utility.imagetools.**hdr2vm**(*imgf*, *vpt=False*)
hdr to ViewMapper

raytraverse.utility.imagetools.**normalize_peak**(*v*, *o*, *l*, *scale=179*, *peaka=6.7967e-05*, *peakt=100000.0*, *peakr=4*, *blursun=False*)

raytraverse.utility.imagetools.**imgmetric**(*imgf*, *metrics*, *peakn=False*, *scale=179*, *threshold=2000.0*, *lowlight=False*, ***peakwargs*)

3.16.2 cli

raytraverse.utility.cli.**np_load**(*ctx*, *param*, *s*)

read np array from command line

trys np.load (numpy binary), then np.loadtxt (space seperated txt file) then split row by spaces and columns by commas.

raytraverse.utility.cli.**np_load_safe**(*ctx*, *param*, *s*)

raytraverse.utility.cli.**shared_pull**(*ctx*, *lr=None*, *col=('metric',)*, *ofiles=None*, *ptfilter=None*, *viewfilter=None*, *skyfilter=None*, *imgfilter=None*, *metricfilter=None*, *skyfill=None*, *header=True*, *spd=24*, *rowlabel=True*, *info=False*, *gridhdr=False*, *imgzone=None*, ***kwargs*)

used by both raytraverse.cli and raytu, add pull_decs and clk.command_decs as clk.shared_decs in main script so click can properly load options

3.16.3 TStqdm

class raytraverse.utility.**TStqdm**(**_*, ***__*)

Bases: `tqdm`

ts_message(*s*)

write(*s*, *file=None*, *end='\n'*, *nolock=False*)

Print a message via tqdm (without overlap with bars).

set_description(*desc=None*, *refresh=True*)

Set/modify description of the progress bar.

Parameters

- **desc** (*str, optional*) –
- **refresh** (*bool, optional*) – Forces refresh [default: True].

3.17 raytraverse.api

factory functions for easy api access raytraverse.

raytraverse.api.**auto_reload**(*scndir*, *area*, *areaname='plan'*, *skydata='skydata'*, *ptres=1.0*, *rotation=0.0*, *zheight=None*)

reload associated class instances from file paths

Parameters

- **scndir** (*str*) – matches outdir argument of Scene()
- **area** (*str np.array*) – radiance scene geometry defining a plane to sample, tsv file of points to generate bounding box, or np.array of points.
- **areaname** (*str, optional*) – matches name argument of PlanMapper()
- **skydata** (*str, optional*) – matches name argument of SkyData.write()
- **ptres** (*float, optional*) – resolution for considering points duplicates, border generation (1/2) and add_grid(). updateable
- **rotation** (*float, optional*) – positive Z rotation for point grid alignment
- **zheight** (*float, optional*) – override calculated zheight

Returns

- *Scene*
- *PlanMapper*
- *SkyData*

`raytraverse.api.`**`load_lp`**(*path*, *hasparent=True*)

`raytraverse.api.`**`get_integrator`**(*scn*, *pm*, *srcname='suns'*, *simtype='2comp'*, *sunviewengine=None*)

CHAPTER

FOUR

TUTORIALS

4.1 Directional Sampling Overview

(starting at 4:56:25)

4.1.1 Transcript

1. Title Slide

Hello, my name is Stephen Wasilewski and I am presenting some work I have prepared along with my co-authors. Raytraverse is a new method that guides the sampling process of a daylight simulation.

2. The Daylight Simulation Process

To understand how this method can enhance the daylight simulation process, it is useful to view the process by parts.

2.b

The model describes how geometry, materials, and light sources are represented.

2.c

Sampling determines how the analysis dimensions are subdivided into discrete points to simulate.

2.d

These views rays are solved for by a renderer, yielding a radiance or an irradiance value for each view ray.

2.e

This output is evaluated according to some metric or otherwise preparing the data for interpretation.

3. Assumptions

To make a viable workflow, each of these parts require (whether explicitly or implicitly) a number of assumptions that define the limitations and opportunities of the method. To explain this in practical terms, here are three examples of well known climate based modeling methods for visual comfort.

4. CBDM Methods for Visual Comfort: Ev based

Illuminance based methods, including DGPs (simplified Daylight Glare Probability), limit the directional sampling resolution to a single sample per view direction in order to efficiently sample a larger number of positions and sky conditions throughout a space.

Unfortunately: Even if the employed rendering method perfectly captures the true Illuminance, as a model for discomfort glare it fails to account for scenes where the dominant driver of discomfort is contrast based or due to small bright sources in an otherwise dim scene.

5. CBDM Methods for Visual Comfort: 3/5 Phase

The 3-phase and 5-phase methods focus on the model and render steps. These methods fix the implementations of the material and sky models by discretizing the transmitting materials and sky dome in order to replace some steps of the rendering process with a matrix multiplication.

6. CBDM Methods for Visual Comfort: eDGPs

Like the 5-phase method, The enhanced-simplified daylight glare probability method, developed to overcome the limitations of illuminance only metrics, uses separate sampling and rendering assumptions for the indirect contribution and direct view rays. The adaptation level is captured by an illuminance value, but glare sources are identified with an image calculated for direct view ray contributions only.

7. Existing Options For Sampling a Point

In all of these methods, the sampling is treated as a fixed assumption.

7.b

Either directional sampling is directly integrated into an illuminance by the renderer,

7.c

or a high resolution image is generated.

7.d

This is because at intermediate image resolutions the accuracy of the results can be worse than an illuminance sample, and are unreliable for capturing contrast effects due to small sources.

7.e

So unlike sampling positions or timesteps which can be set at arbitrary spacing and easily tuned to the needs of the analysis, directional sampling is much more of an all or nothing choice; where the additional insights offered by an image can require 1 million times more data than a point sample. But is this really necessary?

7.f

Whether through direct image interpretation or any of the commonly used glare metrics, the critical information embedded in an HDR image is usually simplified to a small set of sources and background, each with a size, direction and intensity. We cannot directly sample this small set of rays because we do not know these important directions ahead of time, but how close can we get?

7.g

The raytraverse method provides a means to bridge the gap between point samples and high resolution images, allowing for a tunable tradeoff between simulation time and accuracy.

Our approach is structured by a wavelet space representation of the directional sampling. It works by applying a set of filters to an image to locate these important details.

8. Wavelet Decomposition

To match our sampling space, we apply these filters to a square image space based on the Shirley-Chiu disk to square transform, which preserves adjacency and area, both necessary for locating true details.

8.b

For each level of the decomposition, The high pass filters, applied across each axis (vertical, horizontal, and in combination) isolate the detail in the image, and the low pass filter performs an averaging yielding an image of half the size. This process is repeated, applying the high pass filters to the approximation, down to some base resolution. Each level of the decomposition stores the relative change in intensity at a particular resolution (or frequency).

8.c

The total size of the output arrays is the same as the original, and can be used to perfectly recover the original signal through the inverse transform.

The benefit to compression comes from the fact that the magnitude of the detail coefficients effectively rank the data in terms of their contribution to the reconstruction. By thresholding the coefficients, less important data can be discarded.

8.d

Even after discarding over 99% of the wavelet coefficients, the main image details are recoverable and only some minor artifacts have been introduced.

This property, that the wavelet coefficients rank the importance of samples at given resolutions, makes detail coefficients useful for guiding the sampling of view rays from a point.

9. Reconstruction Through Sampling

This process works as follows:

Beginning with a low resolution initial sampling the large scale features of the scene are captured.

Mimicking the wavelet transform, We apply a set of filters to this estimate and then use the resulting detail coefficients both to find an appropriate number of samples, and as probability distribution for the direction of these samples.

The new sample results returned by the renderer are used to update the estimate, which is lifted to a higher resolution.

This process is repeated up to a maximum resolution, equivalent to (or higher than) what a full resolution image might be rendered at.

10. Component Sampling

There are some cases where the wavelet based sampling will not find important details, such as specular views and reflections of the direct sun. Fortunately, because our method uses sky-patch coefficients to efficiently capture arbitrary sky conditions (similar to 3 phase and others), we can structure the simulation process in such a way to compensate for these misses. I refer you to our paper for details on how this works.

11. Results

Instead, I'll spend my remaining time sharing a few examples of scenes captured with: our approach, a high resolution reference and a matching uniform resolution image to demonstrate the benefits of variable sampling.

In addition to image reconstructions, the relative deviation from the reference is shown for vertical illuminance (characterizing energy conservation) and UGR (Unified Glare Rating, characterizing contrast), relative errors greater than 10% are highlighted in red.

This very glary scene highlights the different paths that light takes from the sun to the eye, including direct views, rough specular and diffuse reflections of the sun and sky. While the deviation in the low resolution image is unlikely to change a prediction in this case, the large errors show a failure case for uniform low-res sampling.

11.b

A more complex, but also more likely scenario is that roller shades will be closed. While there are open questions on how to evaluate the specular transmission of such materials, raytraverse does not introduce any substantial new errors to this process.

11.c

Raytraverse performs similarly well for partially open venetian blinds.

11.d Including deeper in a space where the floor reflection dominates.

11.e

Raytraverse, without virtual sources or other rendering tricks, handles the case of specular reflections of the direct sun, a difficult problem for low resolution sampling.

11.f

One case that we would expect raytraverse to struggle with would be a high frequency pattern like the dot frit shown here. And while the sampling does miss parts of the pattern, especially the lower contrast areas, enough of the detail is caught to meaningfully understand the image and, because of the direct sun view sampling, maintains high accuracy.

11.g

In cases where more image fidelity is desired, raytraverse can be tuned to increase the sampling rate with a proportional increase in simulation time, but in our paper we show that the low sampling rates previously shown achieve a high level of accuracy for field of view metrics.

12. Thank you

Thank you for watching my presentation.

4.2 History

4.2.1 1.3.4

- do not use srcview for local light sources, include atleast 1 level of clean-up
- make sure kd tree is rebuilt when lucky squirrel
- ambient file handling in rtrace
- better memory management in reflection_search (still a problem?)
- new example config with proper settings
- with minsamp > 0 make sure from_pdf returns something so sampling can complete

4.2.2 1.3.3

- static light source sampler, directly samples electric lights at appropriate level, will use lots of extra samples with very long thin fixtures
- color support in lightPointKD and samplers, but for now only works with imagesampler and sourcesampler because need to update skydata to work with color (and handle mixed data)
- use scene detail in sampler (in this case image reconstruction works better WITHOUT scene detail, new interpolation keywords fastc and highc for context interpolation)
- consolidated integrator/zonalintegrator and special methods dv/ds into one class
- changed zonal sunplane query algorithm: filter suns, penalize, query instead of filter suns, sort, filter points
- removed ptfilter keyword for zonal evaluation (new process does not use)
- sunplane normalization based on level 0 distance of sampled suns and level 0 distance of areas for level 0 sampled suns
- SensorIntegrator to process sensorplane results
- manage stranded open OS file descriptors
- wait to calculate omega on demand in lightplaneKD
- removed img2lf in imagetools, creates circular reference, need to add to different module
- allow None vector argument for lightplane initialization (cconstructs filename)
- zero pad hour labels in lightresult for proper file name sorting
- calc_omega method now passes "QJ" to qhull which seems to reliable return regions for all points in case of failure, distributed area among points sharing region (moved from integrator.helpers to translate) so LightPointKD can share
- fixed mistakes in GSS implementation and recalibrated

4.2.3 1.3.2 (2022-04-28)

- force 'fork' for multiprocessing to ensure radiance state is copied to processes
- restructure radiancerrenderers - not singleton, just a stateful class, pickleable with get/set state
- dummy skydatamask class useful for intializing with lightresult axes to handle fill
- value_array method added to ResultAxis for easier syntax
- settable sigma_c method in hvsgsm
- make integrator.helpers public for overrides
- supress warnings from radiance during reflection search
- implement ZonalIntegratorDV

4.2.4 1.3.1 (2022-04-19)

- moved craytraverse to separate repository, now a requirement
- implemented glare sensation model, not yet available from CLI

4.2.5 1.3.0 (2022-04-01)

- first version compatible on linux systems
- changed skyres specification to int (defining side) for consistency with other resolution parameters

4.2.6 1.2.8 (2022-03-15)

- include radius around sun and reflections when resampling view. for 3comp, -ss should be 0 for skyengine
- handle stray hits when resampling radius around sun
- new simtype: 1compdv / integratordv

4.2.7 1.2.7 (2022-03-01)

- parametric search radius for specguide in sunsamplerpt
- integratorDS checks whether it is more memory efficient to apply skyvectors before adding points
- fixed double printing of 360 direct_views
- exposd lowlight and threshold parameter access to cli (both imgmetric and evaluate)
- changed to general precision formatting for lightresult printing
- fixed -skyfilter in pull, needs a skydata file to correctly index, otherwise based on array size
- new sampling metric normalizations, can now control logging and pbars with scene parameter

4.2.8 1.2.6 (2022-02-19)

- add hours when available to skydata
- proper masking of 360 images
- integratorDS handles stray roughness from direct patch
- planmapper, z set to median instead of max, added autorotation/alignment
- bugs/features/consistency in LightResult, need better usage documentation
- directviews from cli (only works with sky)

4.2.9 1.2.5 (2022-02-15)

- integrated zonal calcs in cli
- fall back to regular light result when possible (but keep area)
- fixed bugs in LightResult, ZonalLightResult
- added physically based point spread calculation that ~matches gregs gblur script, but using acutal lorentzian from reference
- added blur psf to sources in image evaluation

4.2.10 1.2.4 (2021-12-03) (not posted until 2022-02-10)

- organized command line code
- use process pool for sun sampler when raytracing is fast (such as -ab 0 runs with dcomp)
- propogate plotp to child sampler if sampling one level
- separated utility command line to own entry point. fixed ambiguity in coordinate handedness of some functions (changed kwarg defaults)

4.2.11 1.2.3 (2021-09-03)

- fixed rcontrib to work with Radiance/HEAD, radiance version string includes commit
- daylightplane - add indirect to -ab 0 sun run (daysim/5-phase style)
- lightpointkd - handle adding points with same sample rays
- sampler - add repeat function to follow an existing sampling scheme
- lightresult - added print function
- scene - remove logging from scene class
- **cli.py**
  - new command imgmetric, extract rays from image and use same metricfuncs
  - mew command pull, filter and output 2d data frames from lightresult
  - add printdata option to suns, to see candidates or border
- make TStqdm progress bar class public
- **include PositionIndex calculation in BaseMetricSet**
  - new metrics: loggcr and position weighted luminance/gcr
- skymapper: filter candidates by positive dirnorm when initialized with epw/wea

- **imagetools: parallel process image metrics, also normalize peak with some**
  assumptions
- lightresult: accept slices for findices argument
- **sunsamplerpt: at second and thrid sampling levels supplement sampling with**
  spec_guide at 1/100 the threshold. helps with imterior spaces to find smaller patches of sun
- positionindex: fix bug transcribed from evalglare with the positionindex below horizon

4.2.12 1.2.0/2 (2021-05-24)

- command line interface development

4.2.13 1.1.2 (2021-02-19)

- improved documentation

4.2.14 1.1.0/1 (2021-02-10)

- refactor code to operate on a single point at a time

4.2.15 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

4.2.16 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

4.2.17 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.

4.2.18 0.1.0 (2020-05-19)

- First release on PyPI.

4.3 Index

4.4 Search

CHAPTER FIVE

CITATION

Either the latest or specific releases of this software are archived with a DOI at zenodo. See: https://doi.org/10.5281/zenodo.4091318

Please cite this journal article for a description and validation of the method:

> Stephen Wasilewski, Lars O. Grobe, Jan Wienold, Marilyne Andersen, *Efficient Simulation for Visual Comfort Evaluations*, Energy and Buildings, Volume 267, 2022, 112141, ISSN 0378-7788, https://doi.org/10.1016/j.enbuild.2022.112141.

Additional peer-reviewed references related to this software:

> Stephen Wasilewski, Lars O. Grobe, Roland Schregle, Jan Wienold, and Marilyne Andersen. 2021. *Raytraverse: Navigating the Lightfield to Enhance Climate-Based Daylight Modeling*. In 2021 Proceedings of the Symposium on Simulation in Architecture and Urban Design. https://infoscience.epfl.ch/record/290685?ln=en

> Quek, G., Wasilewski, S., Wienold, J., Andersen, M., 2021a. *Spatial evaluation of potential saturation and contrast effects of discomfort glare in an open-plan office*, in: BS2021. Presented at the Building Simulation 2021 Conference, Bruges, Belgium. https://infoscience.epfl.ch/record/288945

CHAPTER

SIX

LICENCE

CHAPTER

SEVEN

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

EIGHT

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.

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z