raytraverse Documentation

Release 0.2.4

Stephen Wasilewski

Oct 09, 2020

CONTENTS

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

- Free software: Mozilla Public License 2.0 (MPL 2.0)
- Documentation: https://raytraverse.readthedocs.io/en/v0.2.4/.

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 .
```

for a development install (pip install -e may not work correctly):

```
python setup.py develop
```

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

CHAPTER

TWO

USAGE

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

```
raytraverse --help
```

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

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

```
pip install -r docs/requirements.txt
make docs
```

to generate local documentation.

CHAPTER

THREE

GIT STUFF

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

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

and a public repo (release branch) at:

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

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

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

after a "git pull" make sure you also run:

```
git submodule update
```

to track with the latest commit used by raytraverse.

CHAPTER

FOUR

LICENCE

CHAPTER

FIVE

CREDITS

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

5.1 raytraverse.scene

5.1.1 Scene

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

Bases: object

container for scene description

Parameters

- **outdir** (*str*) – path to store scene info and output files
- **scene** (*str, optional (required if not reload)*) – space separated list of radiance scene files (no sky) or octree
- **area** (*str, optional (required if not reload)*) – radiance scene file containing planar geometry of analysis area or a list of points (line per point, space seperated, first 3 columns x, y, z
- **reload** (*bool, optional*) – if True attempts to load existing scene files in new instance overrides 'overwrite'
- **overwrite** (*bool, optional*) – if True and outdir exists, will overwrite, else raises a FileExistsError
- **ptres** (*float, optional*) – final spatial resolution in scene geometry units
- **ptro** (*float, optional*) – angle in degrees counter-clockwise to point grid
- **pttol** (*float, optional*) – tolerance for point search when using point list for area
- **viewdir** (*(float, float, float), optional*) – vector (x,y,z) view direction (orients UV space)
- **viewangle** (*float, optional*) – should be 1-180 or 360
- **skyres** (*float, optional*) – approximate square patch size (sets sun resolution too)
- **maxspec** (*float, optional*) – maximum specular transmission in scene (used to clip pdf for sun sampling)
- **frozen** (*bool, optional*) – create a frozen octree

outdir = None
path to store scene info and output files

Type str

maxspec = None
maximum specular transmission in scene

Type float

reload = None
try to reload scene files

Type bool

view = None
view translation class

Type raytraverse.viewmapper.ViewMapper

property skyres

property scene
render scene files (octree)

Getter Returns this samplers's scene file path

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

Type str

pts()

5.1.2 SunSetter

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

Bases: object

select suns to sample based on sky pdf and scene.

Parameters

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

srct = None
threshold of sky contribution for determining appropriate srcn

Type float

skyro = None
ccw rotation (in degrees) for sky

Type float

scene = None
raytraverse.scene.Scene

property sunres

property sun_kd
sun kdtree for directional queries

property suns
holds sun positions

Getter Returns the sun source array

Setter Set the sun source array and write to files

Type np.array

choose_suns()

load_sky_facs()

direct_view()

write_sun(*i*)

proxy_src(*tsuns*, *tol=10.0*)
check if sun directions have matching source in SunSetter

Parameters

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

Returns

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

_write_suns(*sunfile*)
write suns to file

Parameters **sunfile** –

5.1.3 SunSetterLoc

class raytraverse.scene.**SunSetterLoc**(*scene*, *loc*, *skyro=0.0*, ***kwargs*)
Bases: raytraverse.scene.sunsetter.SunSetter

select suns to sample based on sky pdf and scene.

Parameters

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

sky = None
raytraverse.scene.SkyInfo

choose_suns()

5.1.4 SunSetterPositions

class raytraverse.scene.**SunSetterPositions**(*scene*, *wea*, *skyro=0.0*, ***kwargs*)
Bases: raytraverse.scene.sunsetter.SunSetter

select suns to sample based on sky pdf and scene.

Parameters

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

scene = None
raytraverse.scene.Scene

skyro = None
ccw rotation (in degrees) for sky

Type float

property candidates
raytraverse.scene.SkyInfo

choose_suns()

5.1.5 SkyInfo

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

sky location data object

Parameters

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

skyro = None
ccw rotation (in degrees) for sky

Type float

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

Getter Returns solar bounds

Type (np.array, np.array)

property loc
scene location

Getter Returns location

Setter Sets location and self.solarbounds

Type (float, float, int)

in_solarbounds (*uv*, *size=0.0*)
for checking if src direction is in solar transit

Parameters

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

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

Return type np.array

5.2 raytraverse.mapper

5.2.1 SpaceMapper

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

translate between world coordinates and normalized UV space

rotation = None
ccw rotation (in degrees) for point grid on plane

Type float

tolerance = None
tolerance for point search when using point list for area

Type float

ptres = None
point resolution for area

Type float

property pt_kd
point kdtree for spatial queries built at first use

property sf
bbox scale factor

property ptshape
shape of point grid

property npts
number of points

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

Type np.array

_ro_pts (*points*, *rdir=- 1*)
rotate points

Parameters

- **points** (*np.ndarray*) – world coordinate points of shape (N, 3)
- **rdir** (*-1 or 1*) –

  rotation direction: -1 to rotate from uv space 1 to rotate to uvspace

uv2pt (*uv*)
convert UV –> world

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

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

Return type np.array

pt2uv(*xyz*)
convert world –> UV

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

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

Return type np.array

idx2pt(*idx*)

pts()

in_area(*xyz*)
check if point is in boundary path

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

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

Return type np.array

_rad_scene_to_bbox(*plane*)

5.2.2 SpaceMapperPt

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

translate between world coordinates and normalized UV space

property sf
bbox scale factor

property ptshape
shape of point grid

property bbox
bounding box

Type np.array of shape (3,2)

uv2pt(*uv*)
convert UV –> world

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

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

Return type np.array

pt2uv(*xyz*)
convert world –> UV

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

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

Return type np.array

idx2pt(*idx*)

pts()

in_area(*xyz*)
check if point is in boundary path

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

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

Return type np.array

5.2.3 ViewMapper

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

Bases: object

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

Parameters

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

property viewangle
view angle

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

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

property bbox
bounding box of view

Type np.array of shape (2,2)

property sf
bbox scale factor

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

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

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

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

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

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

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

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

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

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

pixel2omega(*pxy*, *res*)

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

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

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

in_view(*vec*, *i=0*, *indices=True*)

5.2.4 SunMapper

class raytraverse.mapper.**SunMapper**(*suns*)

Bases: raytraverse.mapper.viewmapper.ViewMapper

translate between view and normalized UV space

Parameters **suns** (*np.array*) – dx,dy,dz sun positions

5.3 raytraverse.renderer

5.3.1 Renderer

5.3.2 RadianceRenderer

5.3.3 Rtrace

5.3.4 Rcontrib

5.3.5 SPRenderer

5.3.6 SPRtrace

5.3.7 SPRcontrib

5.4 raytraverse.sampler

5.4.1 Sampler

5.4.2 SCBinSampler

5.4.3 SunSampler

5.4.4 SingleSunSampler

5.4.5 SunViewSampler

5.5 raytraverse.lightfield

5.5.1 LightField

5.5.2 LightFieldKD

5.5.3 SCBinField

5.5.4 SunField

5.5.5 SunSkyPt

5.5.6 SunViewField

5.5.7 MemArrayDict

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

Bases: dict

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

static _map(*i*)

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

constructors()

full_array()

full_constructor()

index_strides()

5.6 raytraverse.integrator

5.6.1 Integrator

5.7 raytraverse.craytraverse

5.8 raytraverse.draw

5.9 raytraverse.io

functions for reading and writing

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

Bases: object

redirect output streams at system level (including c printf)

Parameters

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

Notes

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

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

property stdout

drain_bytes()

read stdout as bytes

drain_str()

read stdout as unicode

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

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

raytraverse.io.**unset_nproc**()

raytraverse.io.**call_sampler**(*outf*, *command*, *vecs*, *shape*)

make subprocess call to sampler given as command, expects rgb value as return for each vec

Parameters

- **outf** (*str*) – path to write out to

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

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

Return type np.array

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

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

Parameters

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

Returns

Return type bytes

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

Parameters

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

Returns

Return type np.array

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

Parameters

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

Returns necessary for reconstruction

Return type ar.shape

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

Parameters

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

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

Parameters

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

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

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

Returns **ar**

Return type np.array

raytraverse.io.**rgbe2lum**(*rgbe*)
convert from Radiance hdr rgbe 4-byte data format to floating point luminance.

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

Returns **lum**

Return type luminance in cd/m^2

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

5.10 raytraverse.metricfuncs

standardized metric functions

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

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

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

raytraverse.metricfuncs.**to_plane**(*n*, *vec*)

raytraverse.metricfuncs.**angle_vv**(*a*, *b*)

raytraverse.metricfuncs.**get_pidx_guth**(*sigma*, *tau*)

raytraverse.metricfuncs.**get_pidx_iwata**(*phi*, *theta*)

raytraverse.metricfuncs.**get_pidx_kim**(*sigma*, *tau*)

raytraverse.metricfuncs.**get_pos_idx**(*vm*, *vec*, *guth=True*)

5.11 raytraverse.plot

functions for plotting data

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

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

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

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

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

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

5.12 raytraverse.quickplot

functions for plotting data

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

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

5.13 raytraverse.skycalc

functions for loading sky data and computing sun position

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

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

Return type np.array

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

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

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

Return type np.array(datetime.datetime)

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

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

Parameters

- **timesteps** (*np.array(np.datetime64)*) –
- **lon** (*float*) – longitude in decimals. West is +ve

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

Returns Sun position as (altitude, azimuth) in degrees

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

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

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

Parameters

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

Returns Sun position as (altitude, azimuth) in radians

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

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

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

Parameters

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

Returns Sun position as (x, y, z)

Return type np.array

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

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

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

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

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

raytraverse.skycalc.**perez**(*sxyz*, *dirdif*, *md=None*, *ground_fac=0.2*)
compute perez coefficients

Notes

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

Parameters

- **sxyz** (*np.array*) – (N, 3) dx, dy, dz sun position
- **dirdif** (*np.array*) – (N, 2) direct normal, diffuse horizontal W/m^2
- **md** (*np.array, optional*) – (N, 2) month day of sky calcs (for more precise eccentricity calc)
- **ground_fac** (*float*) – scaling factor (reflecctance) for ground brightness

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

Return type np.array

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

generate sky, ground and sun values from sun position and sky values

Parameters

- **sxyz** (*np.array*) – sun directions (N, 3)
- **dirdif** (*np.array*) – direct normal and diffuse horizontal radiation (W/m^2) (N, 2)
- **side** (*int*) – sky subdivision
- **jn** (*int*) – sky patch subdivision n = jn^2
- **ground_fac** (*float*) – scaling factor (reflecctance) for ground brightness

Returns

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

5.14 raytraverse.translate

functions for translating between coordinate spaces and resolutions

raytraverse.translate.**norm**(*v*)

normalize 2D array of vectors along last dimension

raytraverse.translate.**norm1**(*v*)

normalize flat vector

raytraverse.translate.**tpnorm**(*thetaphi*)

normalize angular vector to 0-pi, 0-2pi

raytraverse.translate.**uv2xy**(*uv*)

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

raytraverse.translate.**uv2xyz**(*uv*, *axes=0, 1, 2*, *xsign=- 1*)

translate from 2 x unit square (0,2),(0,1) to unit sphere (x,y,z) http://psgraphics.blogspot.com/2011/01/improved-code-for-concentric -map.html.

raytraverse.translate.**xyz2uv**(*xyz*, *normalize=False*, *axes=0, 1, 2*, *flipu=True*)

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

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

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

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

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

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

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

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

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

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

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

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

Parameters

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

Returns to resampled array

Return type np.array

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

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

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

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

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

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

Return type (np.array, np.array)

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

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

Returns **theta** – angle captured by chord

Return type float

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

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

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

Return type float

raytraverse.translate.**aa2xyz**(*aa*)

raytraverse.translate.**xyz2aa**(*xyz*)

5.15 History

5.15.1 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.

5.15.2 0.1.0 (2020-05-19)

- First release on PyPI.

5.16 Index

5.17 Search

5.18 Todo

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y