raytraverse Documentation

Release 1.3.1

Stephen Wasilewski

Apr 19, 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 .
```

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.

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:

```
[raytraverse_scene]
out = outdir
scene = room.rad

[raytraverse_area]
ptres = 2.0
zone = plane.rad

[raytraverse_suns]
loc = weather.epw
epwloc = True

[raytraverse_skydata]
wea = weather.epw

[raytraverse_skyengine]
accuracy = 2.0
rayargs = -ab 2 -ad 4 -c 1000

[raytraverse_sunengine]
accuracy = 2.0
rayargs = -ab 2 -c 1

[raytraverse_skyrun]
accuracy = 2.0
jitter = True
nlev = 2
overwrite = False

[raytraverse_sunrun]
accuracy = 2.0
nlev = 2
srcaccuracy = 2.0
```

and then from the command line run:

```
raytraverse -c options.cfg skyrun sunrun
```

3.1 raytraverse.scene

3.1.1 BaseScene

3.1.2 Scene

3.1.3 ImageScene

3.2 raytraverse.mapper

3.2.1 Mapper

3.2.2 AngularMixin

3.2.3 ViewMapper

3.2.4 SkyMapper

3.2.5 PlanMapper

3.2.6 MaskedPlanMapper

3.3 raytraverse.formatter

3.3.1 Formatter

3.3.2 RadianceFormatter

3.4 raytraverse.renderer

3.4.1 Renderer

3.4.2 RadianceRenderer

3.4.3 Rtrace

3.4.4 Rcontrib

3.4.5 ImageRenderer

3.5 raytraverse.sky

3.5.1 skycalc

3.5.2 SkyData

3.6 raytraverse.sampler

3.6.1 draw

3.6.2 BaseSampler

3.6.3 SamplerSuns

LightSet

LightPointSet

MultiLightPointSet

3.8.7 RaggedResult

3.8.8 ResultAxis

3.9 raytraverse.integrator

3.9.1 Integrator

3.9.2 IntegratorDS

3.9.3 IntegratorDV

3.9.4 ZonalIntegrator

3.9.5 ZonalIntegratorDS

3.10 raytraverse.evaluate

3.10.1 BaseMetricSet

3.10.2 MultiLumMetricSet

3.10.3 MetricSet

3.10.4 FieldMetric

3.10.5 SamplingMetrics

3.10.6 PositionIndex

3.10.7 retina

3.11 raytraverse.craytraverse

3.12 raytraverse.io

3.13 raytraverse.translate

functions for translating between coordinate spaces and resolutions

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

raytraverse.translate.**norm1**(*v*)
: normalize flat vector

raytraverse.translate.**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*)

3.14 raytraverse.utility

3.14.1 imagetools

3.14.2 cli

3.14.3 TStqdm

3.15 raytraverse.api

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 1.3.1 (2022-04-19)

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

4.2.2 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

4.2.3 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

4.2.4 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

4.2.5 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)

4.2.6 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

4.2.7 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)

4.2.8 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

4.2.9 1.2.0/2 (2021-05-24)

- command line interface development

4.2.10 1.1.2 (2021-02-19)

- improved documentation

4.2.11 1.1.0/1 (2021-02-10)

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

4.2.12 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

4.2.13 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

4.2.14 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.

4.2.15 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

Additionally, please cite this conference paper for a description of the directional sampling and integration method:

> 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.

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