raytraverse Documentation

Release 1.4.0

Stephen Wasilewski

Jan 12, 2023

COMMAND LINE INTERFACE

raytraverse is a workflow for climate based daylight simulation for the 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. The code base has been split into three seperate packages: this one as well as two others that provide much of the backend functionality. craytraverse (https://pypi.org/project/craytraverse) is doccumented here and contains code to run radiance simulations from within python. raytools is documented here: https://raytools.readthedocs.io/en/latest/ and contains a lot of the library functions with working with hdr and ray based data. It is its' own repository because it includes stand alone tools for evaluating hdr images for glare metrics.

- 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 .
```

while raytraverse installs with the necessary essentials of radiance, it is recommended to also install radiance (see: https://github.com/LBNL-ETA/Radiance/releases and make sure you or the installer also sets the $RAYPATH variable) this is especially important if your material or light source definitions rely on .cal files distributed with radiance, such as perezlum.cal, window.cal, etc. Missing .cal files or other scene errors can cause raytraverse traverse to abort with cryptic eror messages (the number value is not meaningful and will be different every time):

```
python: : Unknown error -1624667552
```

If you encounter such an error, make sure your scene is valid in your current environment, using rvu, rpict, or rtrace.

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 2.5 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. In an empty directory make a file called Dockerfile_first with the following contents:

```
# syntax=docker/dockerfile:1
# docker build -f Dockerfile_first . --tag raytraverse:latest
FROM python:3.9

WORKDIR /build
RUN apt-get update
RUN apt-get -y install man

SHELL ["/bin/bash", "-c"]
RUN pip3 install raytraverse
RUN curl -s https://api.github.com/repos/LBNL-ETA/Radiance/releases\?per_page\=1␣
```

(continues on next page)

(continued from previous page)

```
↪\
| grep "browser_download_url.*Linux.zip" | cut -d: -f2,3 | tr -d \" | wget -i -
RUN unzip Radiance_*_Linux.zip
RUN tar -xzf radiance-*-Linux.tar.gz
WORKDIR /radiance
RUN rm -rf bin lib man
RUN mv /build/radiance-*-Linux/usr/local/radiance/* ./
RUN rm -rf /build

ENV RAYPATH=.:/radiance/lib
ENV MANPATH=/radiance/man
ENV PATH=/radiance/bin:$PATH
RUN raytraverse --help
WORKDIR /working
```

4. in a command prompt navigate to this folder and run the following to create a docker image with raytraverse and radiance installed:

```
docker build - --tag raytraverse:latest < Dockerfile_first
```

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 (note that you may need to change the syntax of "$(pwd)" to be compatible with your shell, this works with the basic windows command prompt):

```
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, radiance, and python 3.9 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, the process is similar to step 4, but with a slightly different dockerfile:

```
# syntax=docker/dockerfile:1
# docker build -f Dockerfile_update . --tag raytraverse:latest
FROM raytraverse:latest

WORKDIR /build

SHELL ["/bin/bash", "-c"]
RUN pip3 install --upgrade --no-deps craytraverse
RUN pip3 install --upgrade --no-deps clasp
RUN pip3 install --upgrade --no-deps raytraverse
RUN curl -s https://api.github.com/repos/LBNL-ETA/Radiance/releases\?per_page\=1␣
↪\
| grep "browser_download_url.*Linux.zip" | cut -d: -f2,3 | tr -d \" | wget -i -
```

(continues on next page)

(continued from previous page)

```
RUN unzip Radiance_*_Linux.zip
RUN tar -xzf radiance-*-Linux.tar.gz
WORKDIR /radiance
RUN rm -rf bin lib man
RUN mv /build/radiance-*-Linux/usr/local/radiance/* ./
RUN rm -rf /build

ENV RAYPATH=.:/radiance/lib
ENV MANPATH=/radiance/man
ENV PATH=/radiance/bin:$PATH
RUN raytraverse --help
WORKDIR /working
```

and this command:

```
docker build - --tag raytraverse:latest < Dockerfile_update
```

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
rayargs = -ab 0
nlev = 5

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

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

(continues on next page)

(continued from previous page)

```
srcaccuracy = 2.0
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
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.scene

3.1.1 BaseScene

3.1.2 Scene

3.1.3 ImageScene

3.2 raytraverse.mapper

3.2.1 SkyMapper

3.2.2 PlanMapper

3.2.3 MaskedPlanMapper

3.3 raytraverse.formatter

3.3.1 Formatter

3.3.2 RadianceFormatter

3.4 raytraverse.renderer

3.4.1 ImageRenderer

3.4.2 SpRenderer

3.5 raytraverse.sky

3.5.1 skycalc

3.5.2 SkyData

3.5.3 SkyDataMask

3.6 raytraverse.sampler

3.6.1 draw

3.6.2 BaseSampler

3.6.3 Sensor

3.6.4 ISamplerArea

3.6.5 ISamplerSuns

3.6.6 SamplerSuns

3.6.7 SamplerArea

3.6.8 SamplerPt

LightSet

LightPointSet

MultiLightPointSet

SensorPointSet

3.8.9 RaggedResult

3.8.10 ResultAxis

3.9 raytraverse.integrator

3.9.1 Integrator

3.9.2 SensorIntegrator

3.9.3 helpers

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

3.10.8 hvsgsm

3.10.9 GSS

3.11 raytraverse.craytraverse

3.12 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 1.3.11

- updated image generation for comppressedpoint
- added option to directly sample imagerenderer without transformation

4.2.2 1.3.10 (2022-12-20)

- paralellize sensorintegrator matrix multiplication
- include interpolated/represented area in zonallightresult.rebase() output
- fix scikit-learn in requiirements.txt/setup.py
- add random patch color view to lightpointkd.direct_view() for diagrams

4.2.3 1.3.9 (2022-12-08)

- improvements to more easily load lightpoints directly from files without context
- api.load_config for better compatibility between scripting/CLI
- change eccentricity model in hvsgsm
- fix bug in autorotate in planmapper when result should be 0 or 90

4.2.4 1.3.8 (2022-11-04)

- 0 variance bug in image interpolation
- added boundaries to LightResults
- accept dew_point in ttsv format to skydata
- outtput skydata with dewpoint using skydata_dew
- pull gridhdr no longer needs zone if it is embedded in LightResult

4.2.5 1.3.7 (2022-10-26)

- updated resonse fit in hsvgm
- resolution option for pull 2 hdr cli
- modules directly available from import raytraverse
- ensure parameters set correctly so sun is always resampled in 1compdv
- bug in integrator log showing too many chunks
- rebase method added to basic LightResult

- rewrote lightpoint image interpolation
- SrcSamplerPt no longer uses accuracy, instead, set t0, at t1 with physical units
- t0 and t1 now instance properties (settable from init)
- added direct view options to raytu lp2img (warning, non-fisheye color throws error)
- preserve -ss in direct view sampling
- clean up srcview sampling (always distant) and fix double counting image when rough specular gets re-samples
- new factoring with craytraverse / renderer inheritance fixes rcontrib reset

4.2.6 1.3.6 (2022-07-28)

- add scale to sensor integrator forr proper unit conversion (lux by default)
- parallel processing in zonallightresult.pull2hdr
- add lightresult.pull2planhdr to match signature of zonallightresult
- add zonallightresult.rebase to make standard lightresult from zonal
- fixed bug in sunsplanekd.query_by_sun that returned all points, not just best matches
- added index() function to resultaxis
- bug fixes in sensorintegrator, needed additional function overrides and index broadcasting
- avoid IndexError at the end of skydata.maskindices
- add lightresult.merge (and cli interface with raytu merge) for combining LightResults
- change chunking of large calls to evaluate for better performance and the save intermediate results
- pass jitterrate to MaskedPlanMapper constructor
- rewrote RadianceFormatter.get_scene() parser, not based on file extensions
- bug in SamplerArea when operating with MaskedPlanMapper, possible to have no samples, leading to IndexError, fixed at self._mask initialization so atleast one cell is True.
- added gss to raytu imgmetric (no options yet, uses standard observer)

4.2.7 1.3.5 (2022-07-05)

- better memory management in zonal sensorintegrator
- plot each weight in srcsamplerpt when using detail/color
- slight reorganization in Integrator to accommodate sensorintegrator changes
- fixed bug in pull with -skyfilter but no -skyfill
- allow skydata write without scene
- change default sunrun parameter to -ab 0
- updated installation instructions and Dockerfiles to include radiance installation
- added adpatch for better control over default args in Rcontrib
- 2x speedup in translate.calc_omega by checking for containment before intersection left commented code for pygeos method, but it is slower without better way to read in voronoi (creation with pygeos only uses small fraction of points).
- formatting change in CLI docstrings to avoid error with latest docutils

4.2.8 1.3.4 (2022-06-21)

- 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

4.2.9 1.3.3 (2022-06-07)

- 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

4.2.10 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

4.2.11 1.3.1 (2022-04-19)

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

4.2.12 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

4.2.13 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

4.2.14 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

4.2.15 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)

4.2.16 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

4.2.17 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)

4.2.18 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

4.2.19 1.2.0/2 (2021-05-24)

- command line interface development

4.2.20 1.1.2 (2021-02-19)

- improved documentation

4.2.21 1.1.0/1 (2021-02-10)

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

4.2.22 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

4.2.23 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

4.2.24 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.

4.2.25 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.