Building Technology & Urban Systems Division
Energy Technologies Area
Lawrence Berkeley National Laboratory

A Python Library for Radiance Matrix-based Simulation Control and EnergyPlus Integration

Taoning Wang[1], Greg Ward[2], Eleanor S. Lee[1]

[1]*Lawrence Berkeley National Lab, Berkeley, CA*
[2]*Anyhere Lab Berkeley, CA*

Energy Technologies Area
September 2021

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy,
Building Technologies Office, of the U.S. Department of Energy
under Contract No. DE-AC02-05CH11231.

A Python Library for Radiance Matrix-based Simulation Control and EnergyPlus Integration

Taoning Wang[1], Greg Ward[2], Eleanor S. Lee[1]
[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2]Anyhere Software, Berkeley, CA, USA

Abstract

Radiance matrix-based methods enable efficient parametric simulations, allowing users to vary sky conditions, fenestration systems, and other model parameters at a minimal cost to computation. However, the steep learning curve and complex workflow hinder the widespread adoption of matrix-based methods. The *frads* Python library with a series of command-line tools was developed to automate the entire matrix-based simulation process, lowering entry barriers and reducing human error. Co-simulation between EnergyPlus and Radiance was also enabled using the Python library from EnergyPlus.

Key Innovations

- Command-line based automation of Radiance matrix-based simulation methods
- Python library facilitates broader adoption of Radiance matrix-based simulation methods
- Radiance EnergyPlus run-time integration enabling the modeling of advanced control systems

Practical Implications

The ***frads*** library, with associated command-line tools, provides practitioners with the capability to easily adopt and use Radiance matrix-based simulation methods for various daylighting, solar control, and energy-related evaluations. *Frads'* current form is designed for 1) users familiar with a command-line interface and 2) software developers to integrate the matrix-based methods into existing software packages.

Introduction

In the realm of building simulation, there are many reasons why accurate, efficient ray-tracing-based solar radiation and daylighting simulations are needed:

1) accurate thermal and visual comfort predictions rely on detailed maps of solar radiation and luminance on an occupant's body or field of view (Figure 1);

2) broader adoption of energy efficiency, indoor environmental quality, and healthy building standards require accurate modeling of innovative fenestration solutions; and,

3) increased trends toward integrated, advanced building design and control solutions require efficient models to work seamlessly within co-simulation environments, such as Spawn-of-EnergyPlus (Wetter et al., 2015).

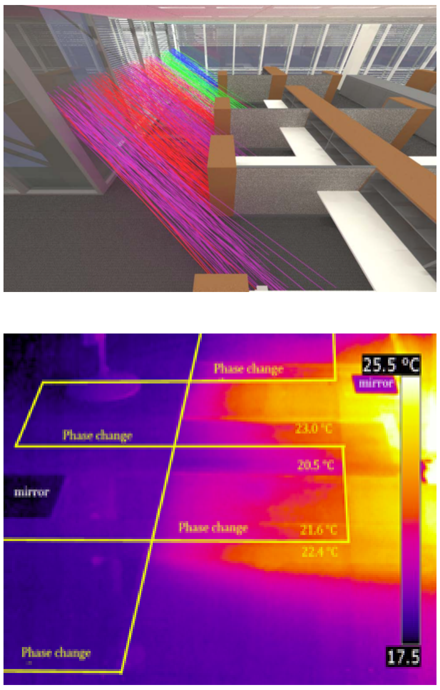

Figure 1. Above: Time-step calculations of solar loads on an underfloor air distribution system using Radiance-EnergyPlus (Lee et al. 2013). Below: LBNL FLEXLAB thermal image of phase-change and conventional flooring irradiated by sunlight; spatially-resolved Radiance data can be used to produce more accurate evaluations of thermal comfort. Source: LBNL.

Conducting gold standard, classic ray-tracing at each timestep, however, is computationally intensive and impractical for annual simulations. As a result, legacy daylight models, such as the split-flux and radiosity-based methods, are still available to end-users of EnergyPlus, despite their inaccuracies. Recently, Radiance matrix-based simulation methods have been developed and validated, enabling efficient and accurate ways to simulate annual, dynamic, daylighting, and solar control performance.

With matrix-based methods, the time needed to conduct annual simulations using ray-tracing tools has been reduced by several orders of magnitude (McNeil & Lee, 2013). Flux transport from the sky to interior points of interests can be broken down into two or more "phases" and stored in a matrix format to enable a parametric analysis of different parts of the scene (Figure 2). A simple case is to store the entire light transport from the points of interest to the discretized sky in a single matrix, known as the Daylight Coefficient (DC) method

(Tregenza and Waters, 1983; Littlefair, 1992). Multiplying such a matrix with a sky vector yields the irradiance of points under that specific sky condition. While generating the initial matrices is computationally intensive, matrix multiplication thereafter is almost instantaneous. Thus, simulating the annual performance for a specific location becomes trivial compared to conventional ray tracing once the matrices storing flux transport have been created. Software implementing the two-phase DC method (e.g., DAYSIM), however, cannot model optically-complex operable shades and dynamic glazing efficiently (Ward et al., 2011).

Three- and four-phase simulations enable a parametric analysis of planar and non-coplanar operable facade systems (McNeil & Lee, 2013; Wang et al., 2018). For tasks where specular transmission and solar peak preservation are critical such as for annual sunlight exposure (ASE) or visual comfort evaluations, five- and six-phase methods have been developed to increase accuracy; i.e., calculation of the direct sun component is separate from the diffuse sky component (Lee et al., 2018, Geisler-Moroder et al. 2017).

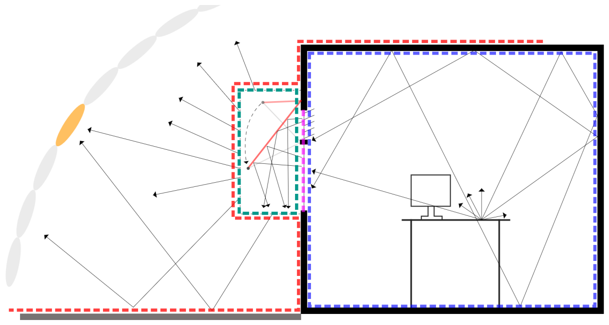

Figure 2. Flux transport can be broken down into four parts: daylight, facade, transmission, and view matrix.

While matrix-based methods offer extensive computational efficiency benefits, they are challenging to understand and deploy even with advanced tutorials (Subramaniam, 2017). Deploying a matrix-based simulation requires that the user have a basic understanding of Radiance hemispherical sampling mechanisms, the way rays are spawned and binned directionally at surfaces, and basic linear algebra. There are also, for matrix-generation, hemispherical sampling bases to choose from, determining how the rays are binned at spawning and receiving in terms of resolution and spatial distribution (McNeil, 2011). With five matrix methods now available and various sampling bases to choose from, it is up to users to choose the appropriate method for their particular application. Different methods yield different results and accuracy levels for standard daylight metrics such as spatial daylight autonomy (sDA) and ASE (Brembilla et al., 2019). Also, performing a matrix-based simulation requires meticulous book-keeping as the number of matrix files multiplies when multiple window groups and fenestration systems are involved. If not using a graphic user interface, a user usually resorts to scripting language to establish the workflow. The Radiance *rfluxmtx* tool was introduced in 2014 to automate some of the workflows required in the lower-level *rcontrib* program. However, users still need to manually decorate the sender and receiver files, making sure the surface normals are facing the correct direction, then make multiple calls to *rfluxmtx* while keeping track of which matrix file is associated with which "phase." During this process, user errors are difficult to avoid and diagnose.

Whole-building energy simulation practitioners, such as users of EnergyPlus, could leverage Radiance matrix-based methods to perform more accurate solar irradiance, daylighting, and electric lighting energy calculations. However, the complex nature and inconvenience of learning two software packages and building two separate models hinder real-world adoption. Third-party software provides such integration, but there can be a lag between new developments in simulation techniques and adoption by the software developers. Furthermore, the evaluation of advanced control strategies, such as model predictive control, requires real-time co-simulation between the energy and daylighting simulation engines (Gehbauer et al., 2020). While EnergyPlus version 9.3 introduced such capabilities through its Python application programming interface (API), there still needs to be a counterpart on the Radiance side to enable run-time co-simulation with EnergyPlus.

This work aims to enable users and software developers to adopt advanced Radiance matrix-based simulation methods without extensive knowledge and experience. General modeling knowledge is still required. In this study, we describe and demonstrate the Python-based, open-source library *frads* and accompanying command-line programs that automate and speed up the use of these advanced simulation methods. The methods utilize the tools developed under the U.S. Department of Energy building energy modeling program portfolio, enabling advanced co-simulation workflows (Wetter et al., 2015).

Methods

Frads is designed to satisfy two high-level goals:

1) extend the current Radiance simulation workflow (Unix-toolbox model) and implement high-level abstractions of the Radiance matrix-based simulation methods, and

2) provide the necessary infrastructure for seamless integration of Radiance matrix-based methods with other Building Energy Modeling tools, such as EnergyPlus.

Frads consists of a Python library where the matrix-based simulation workflow is implemented by individual modules, each handling parts of the workflow. *Frads* distribution also includes a toolbox that consists of several command-line programs, including two executive programs, *mrad* and *eprad*, developed using the Python library. Figure 3 shows *frads'* structure where each command-line program in the toolbox calls the Python library behind the scene to complete the task.

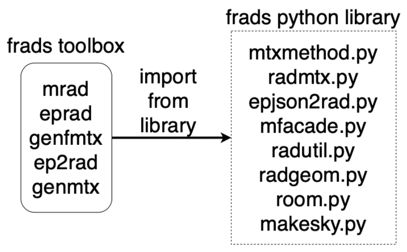

Figure 3. Each command-line program in the frads toolbox directly uses frads python library modules to perform tasks.

Frads toolbox

frads provide several high-level command-line tools for users who are familiar with the command-line interface to expedite standard workflow and minimize user error. The command-line tools also serve as examples of integrating the *frads* library with other software packages or graphic user interfaces.

The two main command-line tools are *mrad* and *eprad*. The former is an executive program that automates the Radiance matrix-based simulation methods for a given Radiance model. The latter is a tool that starts an EnergyPlus simulation using Radiance as the lighting, daylighting, solar radiation simulation engine with detailed shading operation modeling capability.

Mrad

Mrad is a program that controls *rfluxmtx* and *rcontrib* for managing different matrix-based methods. It eliminates most of the manual work of setting up the simulation workflow and keeping track of files. *Mrad* also has some built-in general knowledge that automatically determines which matrix-based method to apply then sets the associated simulation parameters, helping beginners learn the different matrix methods by observing *mrad's* behavior. However, *mrad* still requires the user to understand basic concepts underlying matrix-based simulation methods. End users will need to decide on accuracy-speed tradeoffs, such as selecting the resolution of the sampling basis, then supplying the required fenestration bidirectional scattering distribution function (BSDF) data based on the desired performance metric (e.g., discomfort glare).

The design of *mrad* follows the schematic of the Radiance *rad* program (Figure 4). It takes an input file of variables specifying Radiance scene files (i.e., zone geometry and materials), window (group) files, simulated points (e.g., grid of workplane illuminance sensors), and simulation control parameters. Based on the number of planar and non-coplanar shading systems specified in the configuration file and whether a separate direct solar calculation is needed, *mrad* then invokes the appropriate matrix phase method. Depending on the method used, it then generates the necessary "sender" and "receiver" objects for each subsequent *rfluxmtx* or *rcontrib* call with the appropriate variables, such as sampling basis (e.g., Klems, Tregenza). In the simplest case, the two-phase method is invoked for a point-in-time or annual calculation of daylight workplane illuminance for each zone with windows. If a separate, more accurate, direct solar contribution is required, such as in the five- or six-phase method, *mrad* then alters the model and calls *rcontrib* to carry out the calculation. When computing separate solar matrices, *mrad* speeds up the calculation by eliminating the solar positions that are not relevant to the specific site and window orientation. Finally, *mrad* multiplies the matrices in the correct groups and order for the final timestep result.

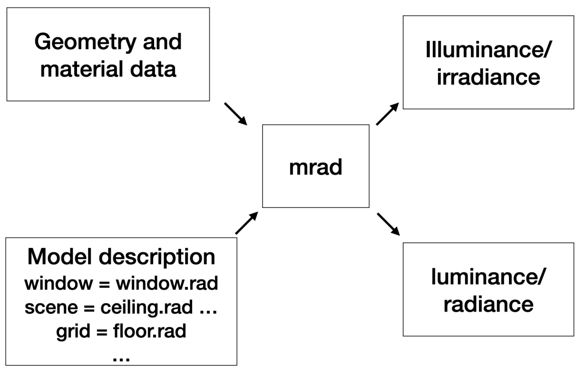

Figure 4. Mrad takes model files and model descriptions in the form of a configuration file and outputs the corresponding annual illuminance or luminance/image results.

Eprad

Eprad is a program that incorporates much of the functionality of *mrad* and leverages the EnergyPlus Python library to enable run-time data exchange. With the advent of version 9.3, EnergyPlus exposes its simulation engine through a Python library. EnergyPlus and the *frads* Python library enable integration between the two software packages at various levels, from simple annual schedule substitution, where data are pre-computed in Radiance and used by EnergyPlus through its schedule component, to run-time data exchange.

The input to *eprad* is a regular EnergyPlus model file in the epJSON format, which is then translated into Radiance format to prepare for the subsequent matrix generation procedure described above. Wall thickness is approximated in Radiance using the construction data from the EnergyPlus model. Surface optical properties need to be defined in the EnergyPlus model. Opaque surfaces are limited to Lambertian reflectance. Transparent surfaces can be defined as a simple glazing material represented as a single center-of-glass visible transmittance or imported from LBNL WINDOW software or Radiance *genBSDF*, including complex fenestration systems with associated BSDFs. Like *mrad*, the appropriate matrix-phase method is invoked automatically by *eprad*. For each zone, *eprad* generates a standard sensor grid from the zone geometry and computes the necessary matrices. Once all the necessary matrices are ready, *eprad* starts the EnergyPlus simulation using the EnergyPlus Python library.

For applications requiring run-time data exchange, such as when the façade control status is unknown before the simulation starts, EnergyPlus internal variables are replaced with values from *frads* using the "actuator" mechanism in the EnergyPlus Energy Management System module. An "actuator" includes zone lighting level, interior surface incident solar radiation, and other parameters. *Eprad* reads and edits the input EnergyPlus model and exposes the necessary "actuators" to be replaced later during the calculation through one of the callback functions implemented in *eprad*. After starting the simulation, EnergyPlus stops at pre-defined points of the calculation, such as before or after the zone heat balance calculation, to initiate the procedure that has been defined in the callback function. The procedure multiplies the corresponding matrices then uses the result to set the "actuator" value. The simulation then continues back to EnergyPlus (Figure 5).

One of the critical advantages of run-time data-exchange between Radiance and EnergyPlus is that it enables evaluations of advanced control of dynamic facades in a multizone building within the Spawn-of-EnergyPlus co-simulation environment. During run-time, at each timestep, EnergyPlus simulation pauses, then when given a dynamic facade control signal (e.g., via Spawn/ Modelica or manufacturer component model), Radiance computes facade energy transfer data, which are sent to EnergyPlus to complete the appropriate heat balance and daylight/ lighting calculations.

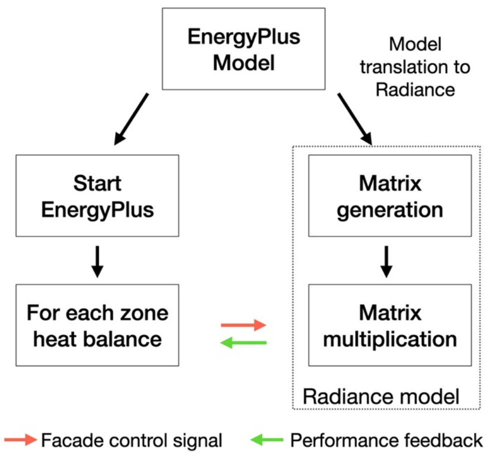

Figure 5. Example run-time data exchange between EnergyPlus and Radiance that happens at each zone heat balance calculation.

Additional command-line tools are available in the *frads* toolbox. The *genfmtx* tool automates the matrix generation for operable and optically-complex, non-coplanar facade systems, such as drop-arm fabric awnings or expanded metal mesh retractable overhangs. This tool is used in the latest LBNL WinCalc engine for the generation of awning BSDF (LBNL WINDOW). *Ep2rad* program offers simple geometric model translation from EnergyPlus to Radiance format. The *genmtx* program is a generic matrix generator for any form of ray sender and receiver.

Frads library

The *frads* library exposes essential functions and classes for advanced users and third-party software developers to embed Radiance matrix-based simulation workflow into their existing software. The *frads* library consists of several modules situated at different abstraction levels (Figure 6). The *mtxmethod* module can be used to compute spatial daylight autonomy (sDA), ASE, and discomfort glare. More detailed workflow control is afforded using the *radmtx*, *makesky*, and *mfacade* modules. Also, the *epjson2rad* module implements the functionality of model translation from an EnergyPlus model to Radiance models representing each zone with exterior windows, and the *mtxmethod* module carries out the related Radiance simulations. With the *frads* Python library, embedding the Radiance matrix-based simulation methods can as simple as the following Python code using the *mtxmethod* module:

```
from frads import mtxmethod
msetup = mtxmethod.MTXMethod(config)
sky_mtx = msetup.gen_smx(config.smx_basis)
mtxmethod.three_phase(msetup, smx)
```

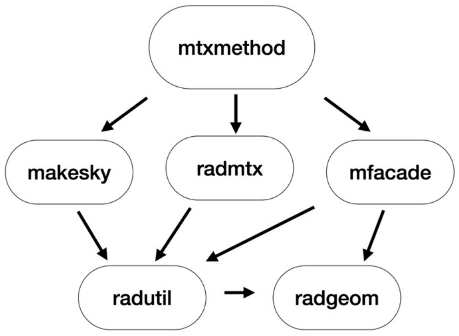

Figure 6. The mtxmethod module is at the highest level, implementing the workflow of each matrix-based method. radmtx module implements the generic sender and receiver objects and the matrix generation workflow of these objects. radgeom and radutil module provide the infrastructure for geometry manipulation and utilities such as file parsing.

Examples of use

Executive programs *mrad* and *eprad* are demonstrated in the following examples. In general, these programs require geometry and material data, either in Radiance or EnergyPlus format.

Example 1: Calculate workplane illuminance

The user wishes to compute a daylight-illuminance based metric (e.g., sDA) in a zone with upper clerestory and lower view windows. The simple room model with two window zones is shown in Figure 7.

The user can complete an entire matrix-based simulation using the *mrad* program, which takes a configuration file (i.e., room1.cfg) as input detailing the room model setup, including the material and geometry (i.e., "scene" variable) data as Radiance primitives. All surfaces in the model need to be in Radiance format, and each window zone in a separate file if the user wishes to treat them separately. The "grid_surface," "spacing," and "height" variables specify the location of the plane and grid spacing of sensor points. The basic input file looks like this (i.e., climate, run period details omitted):

```
#room1.cfg
material = material.mat
scene = walls.rad ceiling.rad floor.rad
window = UpperGlass.rad LowerGlass.rad
grid surface = floor.rad
spacing = 0.2
height = 0.3
```

Here, none of the zone descriptors are parameterized, so *mrad* automatically selects the two-phase method. For example, there is no "bsdf" variable defining BSDF files for each window, nor is there a parameterized exterior shading system defined. In this simple case, the window files can be included as part of the "scene" variable, leaving an empty "window" variable. To run the simulation, the user runs the following command in the terminal:

```
mrad room1.cfg
```

Mrad only runs the illuminance calculation because the view variable is not defined in the configuration file. It also uses a set of default simulation parameters suited for a conventional room, which the user can override in the configuration file. Grid sensor points are generated automatically for the floor surface using the specified spacing and height. The final illuminance results are as follows, from which the user can perform the subsequent processing or plotting:

```
datetime,pt1,pt2,pt3,...
2020-01-01 09:30,132.8,144.3,143.8,...
2020-01-01 10:30,149.9,144.1,152.9,...
2020-01-01 11:30,161.1,158.3,253.2,...
...
```

Example 2: Calculate discomfort glare

The user then wants to include an image-based glare analysis with a venetian blind and fabric roller shade on the windows. The configuration file can be adapted as:

```
#room2.cfg (option 1)
material = material.mat
scene = walls.rad ceiling.rad floor.rad
window = upperglass.rad lowerglass.rad
grid surface = floor.rad
spacing = 0.2
height = 0.3
bsdfs = blind.xml fabric.xml
separate_solar = True
view1 = -vf view.vf

#room2.cfg (option 2)
material = material.mat
...
bsdfs = blind.xml fabric.xml
dbsdfs = blindtt.xml fabrictt.xml
...
```

After running the same command as before, *mrad* invokes the five-phase method (5PM) because the user requests a separate solar calculation through the "separate_solar" variable. This method is then used for calculating both the workplane illuminance and rendering using the view location defined in the configuration file.

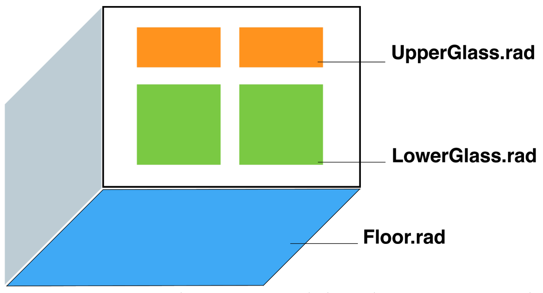

Figure 7. A simple room model with an upper and lower window zone.

With option 1, the BSDF files associated with each window group are described in the "bsdfs" variable, where the two entries, blind.xml and fabric.xml, correspond to the "upperglass.rad" and "lowerglass.rad" files defined in the "window" variable. Proxy geometry is modeled automatically if present in the blind.xml file, otherwise the peak extraction algorithm is used to model specular transmission in the direct solar calculation. The Klems basis is used for the entire calculation. With option 2, an optional "dsbsdfs" variable is used to specify high resolution tensor tree BSDF with peak extraction (or proxy geometry in the case of the blind) for the direct solar part of the 5PM calculation. The workplane illuminance values generated by the previous simulation will not be recomputed unless the user specifies in the configuration file to overwrite the previous result.

Example 3: Calculate illuminance in EnergyPlus

To carry out the first example's daylight illuminance calculation from EnergyPlus, the user runs the following command:

```
eprad -wp room.epJSON
```

where the -wp option invokes a workplane illuminance calculation. Radiance models are generated from the EnergyPlus epJSON zone model, and then matrices are generated following protocols similar to *mrad's*. Workplane illuminance values are then computed and used to determine the electric lighting system's dimming level if daylight controlled or output from EnergyPlus for user-defined analysis of daylight quality.

Example 4: Calculate surface irradiance in EnergyPlus

Interior surface incident irradiance and workplane illuminance can be calculated by running the following command:

```
eprad -wp -si room.epJSON
```

For surface irradiance, *eprad* identifies each interior surface in each zone with a window(s), computes the corresponding matrices for each interior surface, then, for each time step, performs matrix multiplication for each surface. The result is an average irradiance over the whole surface (room surfaces can be subdivided in smaller areas as shown in Figure 1). Incident irradiance values are then set in EnergyPlus prior to completion of the rest of the simulation. If the windows are modeled as complex fenestration systems in EnergyPlus, then they are modeled accordingly in Radiance. Control of the shade and Venetian blind would also be reflected if control logic was defined in the EnergyPlus model. Workplane illuminance results will not be recomputed if the results from example 3 still exist, but the user can force a re-run by using the -f option.

Discussion

The *mrad* program automates the simulation workflow and helps beginners learn Radiance matrix modeling processes. However, there are still a few challenges the user faces when using *mrad*. One challenge is deciding the appropriate sampling basis for each matrix generation process, especially when using the four-phase method for modeling non-coplanar shading systems, where selecting the appropriate sampling basis could affect the simulation's overall accuracy (Wang et al., 2018). The user also must decide how to group and divide windows. Sometimes it is evident to group windows by orientation, construction, or control system. In other situations, window division is not apparent with external obstructions such as overhangs and adjacent buildings. Dividing windows appropriately to account for external obstruction can significantly affect simulation results (Ward et al., 2011). Thus, creating a tool in the future to guide users towards the correct sampling basis and window division is paramount.

Another crucial challenge is modeling the window with a complex fenestration system (e.g., fabric shade, venetian blind, daylight-redirecting film) using BSDFs. There are two aspects to this challenge: data access and modeling. The data access challenge, not related to the current *frads* implementation, refers to the lack of high-quality BSDF data suitable for a wide range of applications, from solar heat gain to visual comfort analysis. For each class of shading or daylighting products and each performance metric, measurement standards are needed to produce adequate tabular BSDF data. There are several ongoing international efforts to address this issue (Geisler-Moroder & Lee, 2020). The modeling challenge refers to the complicated nature of modeling an optically complex fenestration system. The user needs to assemble a multi-layer BSDF if the fenestration system consists of multiple layers (e.g., double-pane window with indoor fabric shade). The user also needs to place the BSDF in the correct orientation (facing indoor vs. outdoor) and planar rotation, depending on how the system is physically measured and how the BSDF is generated. The usual asymmetrical nature of the fenestration system makes the modeling process difficult and prone to user error. Additional toolchains need to be developed to assist users so as to ensure a successful endeavor.

Table 1 lists *frads'* existing and planned features. EnergyPlus and Radiance integration through their respective Python API (*frads* and EnergyPlus Python API) enables long- and short-wave radiation simulation in complex spaces. Advanced thermal comfort models can now receive *frad*-generated solar irradiance data on each surface of a multi-node manikin, for example, at a fraction of the time needed by conventional ray-tracing or radiosity methods. Incident solar irradiance on exterior envelope surfaces (e.g., BIPV, phase change materials) with attached shading or in complex urban environments can be modeled more efficiently. In the future, *frads* will include occupant health- and alertness-related metric analysis enabled by spectrum or equivalent modeling capabilities.

Table 1. frads' existing and planned features for integrated EnergyPlus simulation

	Implementation
Daylight illuminance	Implemented
Solar irradiance on interior surfaces	Implemented
Solar irradiance on window and shading layers	Implemented
Solar irradiance on outdoor surfaces	Implemented
Detailed luminance map for visual comfort	Planned
Melanopic illuminance calculation	Planned

For operable or automated shading and daylighting systems, *Eprad* can model energy performance when conventional rule-based supervisory control sequences are defined via the energy management system (EMS) feature in EnergyPlus or when more advanced controls are modeled using Spawn-of-EnergyPlus functional mockup units (FMU). When the control signal (i.e., position of the dynamic facade) is sent by other means to the EnergyPlus Python API or if the user wishes to send the control signal to *frads* directly (e.g., to compute impacts such as detailed visual or thermal comfort metrics that are then sent to the EnergyPlus Python API), then additional scripting is required by the end user. In the case when Radiance models are used *within* the controller logic, *eprad* cannot be used: additional scripting is required to incorporate matrix models generated by *frads* into the controller. Gehbauer et al., for example, used the *frads* Python library to model solar heat gains, daylight, and glare within a model predictive controller (Gehbauer et al., 2020). Control designers can leverage *frads* to develop and fine-tune the various building components' control strategies to ensure energy efficiency and occupant comfort.

Using programs like *mrad* and *eprad* requires familiarity with the command-line interface. Existing Radiance users who are used to calling programs through the command-line interface will find adopting *frads* a relatively smooth process. However, achieving wide adoption of Radiance matrix-based simulation methods requires graphical interface developers to integrate *frads* through either the command-line interface or *frads* Python library. Some existing commercially available graphical programming interfaces such as Grasshopper have the capability to import external Python modules. Grasshopper users, thus, can directly import and incorporate *frads* into their workflows in Grasshopper. Individual users who are used to programming in Python can also develop customized workflow or specialized tools using the *frads* Python library.

The *frads* library is open-source and actively maintained by developers of Radiance and EnergyPlus engines. The library will be routinely tested using a suite of test cases developed by the Illuminating Engineering (IESNA) Society of North America, Daylight Metrics Committee (DMC) (IESNA 2020). The test cases consist of multiple common building types with different shading systems and the results are generated using classic backwards ray-tracing using Radiance. Results generated from the frads Python library using different matrix-based methods will be compared to this dataset to validate the code. Issue reports and feature suggestions are welcomed as the claimed functionalities certainly do not cover all Radiance and EnergyPlus use cases. See https://github.com/LBNL-ETA/frads.

Conclusion

The *frads* Python-based toolbox and library facilitate the use of Radiance ray-tracing based, time-efficient, matrix algebraic calculation methods, improving the accuracy of illuminance, luminance, and irradiance-related performance measures. Critical Radiance workflows were automated to eliminate user error and integrated with EnergyPlus in a co-simulation environment. Some basic knowledge of matrix-based methods is needed to determine setup requirements for applications with attached exterior shading or complex urban environments. This integration of Radiance and EnergyPlus is expected to significantly improve building energy simulations' speed and accuracy involving light-scattering shading and daylighting systems and smart, operable fenestration.

Acknowledgment

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Office of the US Department of Energy, under Contract No. DE-AC02-05CH11231, and by the California Energy Commission under the Electric Program Investment Charge (EPIC) Program, Solicitation Number: PON-13-301, entitled "Developing A Portfolio of Advanced Efficiency Solutions: Technologies and Approaches for More Affordable and Comfortable Buildings," that was awarded to Lawrence Berkeley National Lab for the work herein.

References

Brembilla, E. & Mardaljevic, J. (2019). Climate-Based Daylight Modelling for compliance verification: Benchmarking multiple state-of-the-art methods. *Building and Environment*, 158, 151-164.

Gehbauer, C., Blum, D. H., Wang, T., & Lee, E. S. (2020). An assessment of the load modifying potential of model predictive controlled dynamic facades within the California context. *Energy and Buildings*, 210, 109762.

Geisler-Moroder, D., et al. (2020). White paper on BSDF generation procedures for daylighting systems. T61.C.2.1: A Technical Report of Subtask C. IEA SHC Task 61 / EBC Annex 77. IEA SHC.

Geisler-Moroder, D., Lee, E.S., Ward, G. (2017). Validation of the Five-Phase Method for Simulating Complex Fenestration Systems with Radiance against Field Measurements. Proceedings of Building Simulation 2017, San Francisco, 7-9 August 2017.

IESNA 2020. IES Technical Memorandum: Daylight Modeling and Simulation Methods and Standards and Test Cases for the Evaluation of Daylighting Analysis Software, draft under development.

Lee, E.S., Geisler-Moroder, D. and Ward, G., 2018. Modeling the direct sun contribution in buildings using matrix algebraic approaches: Methods and validation. *Solar Energy* 160: 380-395.

Lee, E.S., Fernandes, L.L., Coffey, B., McNeil, A., Clear, R., Webster, T., Bauman, F., Dickeroff, D., Heinzerling, D. and Hoyt, T., 2013. A post-occupancy monitored evaluation of the dimmable lighting, automated shading, and underfloor air distribution system in The New York Times Building. LBNL-6023E.

Littlefair, P. J. (1992). Daylight coefficients for practical computation of internal illuminances. *Lighting Research & Technology*, 24(3), 127-135.

McNeil, A. & Lee, E. S. (2013). A validation of the Radiance three-phase simulation method for modeling annual daylight performance of optically complex fenestration systems. *Journal of Building Performance Simulation*, 6(1), 24-37.

McNeil, A. (2011). On the sensitivity of daylight simulations to the resolution of the hemispherical basis used to define bidirectional scattering distribution functions. DOE/ LBNL FY11 Technical Report.

Subramaniam, S. (2017). Daylighting Simulations with Radiance using Matrix-based Methods. Lawrence Berkeley National Laboratory.

Tregenza, P. R., & Waters, I. M. (1983). Daylight coefficients. *Lighting Research & Technology*, 15(2), 65-71.

Wang, T., Ward, G., & Lee, E. S. (2018). Efficient modeling of optically-complex, non-coplanar exterior shading: Validation of matrix algebraic methods. *Energy and Buildings*, 174, 464-483.

Ward, G., Mistrick, R., Lee, E. S., McNeil, A., & Jonsson, J. (2011). Simulating the Daylight Performance of Complex Fenestration Systems Using Bidirectional Scattering Distribution Functions within Radiance, LEUKOS, 7:4, 241-261, DOI: 10.1080/15502724.2011.10732150.

Wetter, M., Nouidui, T. S., Lorenzetti, D., Lee, E. A., & Roth, A. (2015, December). Prototyping the next generation energyplus simulation engine. 13th IBPSA Conference. International Building Performance Simulation Association.