Automate Radiance Workflows through Python

15th International Radiance Workshop
August 30, 2016

Mostapha Sadeghipour Roudsari
University of Pennsylvania

Sarith Subramaniam
Penn State University
Parametric Design

- What is [the History of] Parametric Design?
  Go read this link: [http://www.danielweiss.com/a-history-of-parametric/](http://www.danielweiss.com/a-history-of-parametric/)

Two centuries of developments. A fairly convoluted path that missed prothoses of theory and architecture in order to find past idealised technology. Ultimately this history wasn’t scholarly enough and wasn’t needed for the argument of my thesis. I deleted it.

What is Grasshopper? Why does it matter?

Visual Scripting

Pt = Rhino.Geometry.Point3D(X, Y, Z)

Parametric Analysis is the new ![Icon]!
Ladybug and Honeybee are two open source environmental plugins for Grasshopper to help designers and engineers create an environmentally-conscious building design.

Installation Instructions
- Download Ladybug and Honeybee
- Remove Old Version
- Ladybug Primer, Honeybee Primer
- Example Files for Ladybug
- Example Files for Honeybee
- Ladybug on GitHub
- Honeybee on GitHub

Use this Reference for your Publications.

Licensed under @license GPL-3.0+
http://sps.org/licenses/GPL-3.0+
+53,000 downloads from 2014.

1365 discussions, 5029 replies and 498 comments.

78.5% of the discussions get the first reply in less than a day.

95.0% of them will get the first reply in less than 3 days.
"When an expert network is functioning as its best, the smartest person in the room is the room itself."

- David Weinberger

Too Big to Know
Share Code.

Share Knowledge.

Share Examples.

Share Stories!
[discussion] Sky View Factor

Posted by Grasshopper on February 10, 2016 at 10:52am in Ladybug + Honeybee
+ Back to Ladybug + Honeybee Discussions

Dear bees and bugs,

I'd like to share and discuss with you my understanding of Sky View Factor, considering that it is an important concept frequently misunderstood.

Sky View Factor is first and foremost defined from the discussion of radiation exchange between urban surfaces and the sky in urban heat island research (see Ole's literature list below). It will be affected by the proportion of sky visible from a given calculation point on a surface (vertical or horizontal) as a result of the obstruction of urban geometry, but it is not entirely associated with the solid angle subtended by the visible sky patch/patches.

So, I think using “geometry way” to approximate Sky View Factor is not correct. Sky View Factor calculation shall be based on the first principle defining the concept: radiation exchange between urban surface and sky hemisphere:

(Image extracted from Johnson, G. T. & Watson, 1984)

We may now define the *sky view-factor* \( \psi_s \) for the surface element \( \Delta A \) as the fraction of radiant flux leaving \( \Delta A \) which is intercepted by the sky. Then

\[
\psi_s = \frac{1}{\pi R^2} \int_{S_o} \cos \phi dS.
\]  

(8)

It may be helpful to note that \( \psi_s \) is equal to the ratio of the radiant flux received by \( \Delta A \) from the visible sky, to that which would be received by \( \Delta A \) from an unobscured sky. This is shown by replacing \( F_o \) by \( F_s \) in (6) and integrating over \( S_o \).

Therefore, I always refer to the following “theoretical” Sky View Factors calculated at the centre of an infinitely long street canyon with different Height-to-width ratios in Ole’s original paper (1992) as the
Share Code.

Share Knowledge.

Share Workflows.

Share Stories!
IES Electric Lighting Grid-based calculations

Description
Indoor electric lighting calculations based on grid points
This file has been submitted by saniths
Check out this example on Hydral

Tags
IES Electric Lighting Grid-based calculations

Description

Indoor electric lighting calculations based on grid points

This file has been submitted by sanths

Check out this example on Hydral

Tags

#IES #Electric #Lighting #Grid-based #calculations
IES Electric Lighting Grid-based calculations

Description
Indoor electric lighting calculations based on grid points

This file has been submitted by sariths

Check out this example on Hydra!

Tags
#IES #Electric #Lighting #Grid-based #calculations
Share Code.
Share Knowledge.
Share Examples.

Share Stories!
Yun Kyu Yi, Hyoungsub Kim, Agent-based geometry optimization with Genetic Algorithm (GA) for tall apartment’s solar right.
Yun Kyu Yi, Hyoungsub Kim, Agent-based geometry optimization with Genetic Algorithm (GA) for tall apartment’s solar right.
Make your own tool!
Naomi Keena, Mohamed Aly Etman, Nancy Diniz, Alexandra Rempel, Anna Dyson

Center for Architecture Science and Ecology (CASE), Rensselaer Polytechnic Institute
Max/Dec/Jun 21° for the extremely lazy people

Solar path that gives vectors for solar access simulations. (There is a second sun path there that runs for 21st June to get maximum hours for legend high)

Solar access with and without minimum mark cutoff (+ average passing points for regulations)

Geometry filter (when HBSSrf crashes). This is very useful for complex “dirty” models. It filters surfaces with very small sides (line-like surfaces) that make radiance crash

VSC only visual and then VSC for regulations were only non-horizontal surfaces are calculated and also overlapping surfaces are filtered. (This takes ages to run on more complex models)

Glare setup for fisheye and cylinder (360°) views & Visualization of those

Filters the mesh bellow and above target value and creates a contour line (with a smooth factor to make it visually better)
LadyBee

Contains a cluster of Grasshopper components

This cluster occurs **once** in this document.
The other side of the coin!
Hi

I'm wondering if there is a Dynamo node that can calculate the number of hours of direct sunlight a surface receives. Basically I want it to work like the 'Sunlight hours analysis' component found in Ladybug in Grasshopper. All I can currently see in Dynamo is daylighting and solar insolation. I would like to avoid constructing it from scratch if possible. Any thoughts?
At the last Radiance Workshop, there was some discussion regarding converting REVIT models to Radiance, with materials and named surfaces parsed into readable .rad datasets. As REVIT has essentially become the architectural industry modeling tool, the pipeline to Radiance is an important one. It seems that the latest version of Sketchup has some challenges with the very helpful su2rad, written quite awhile ago. I heard that others were using some other pipelines, perhaps integrating obj2rad, etc. I would appreciate your sharing conversion pipelines which interface with the most current releases of REVIT, SKETCHUP, etc. Hopefully a generous and expert code person would consider writing a direct REVIT plugin for this purpose. There might be modest financial encouragement to fill this gap. I do realize that it may be short-lived as AutoDESK releases are moving targets, yet to engage the full Radiance tool kit for lighting design and analysis on REVIT generated models, will keep Radiance on the forefront and accessible to a broader audience.

Appreciatively,
Rob Shakespeare
shakespe at indiana.edu

-------------- next part --------------
An HTML attachment was scrubbed...
URL: <http://www.radiance-online.org/pipermail/radiance-general/attachments/20151112/abe112fe/attachment.html>
Let the `bug fly free!
What does it really mean?
from honeybee.room import Room
from honeybee.radiance.material.glass import GlassMaterial
from honeybee.radiance.sky.certainIlluminance import SkyWithCertainIlluminanceLevel
from honeybee.radiance.recipe.gridbased import HBGridBasedAnalysisRecipe

room = Room(origin=(0, 0, 3.2), width=4.2, depth=6, height=3.2, rotationAngle=45)  # create a test room

room.addFenestrationSurface(wallName='back', width=2, height=2, sillHeight=0.7)  # add a window to the back wall

glass_60 = GlassMaterial.bySingleTransValue('tvis_0.6', 0.6)  # add another window with custom material.

room.addFenestrationSurface('right', 4, 1.5, 1.2, radianceMaterial=glass_60)  # This time to the right wall

# run a grid-based analysis for this room

sky = SkyWithCertainIlluminanceLevel(illuminanceValue=2000)  # generate the sky

testPoints = room.generateTestPoints(gridSize=0.5, height=0.75)  # generate grid of test points

rp = HBGridBasedAnalysisRecipe(sky=sky, pointGroups=(testPoints,), simulationType=0, hbObjects=(room,))

# write and run the analysis

rp.writeToFile(targetFolder='c:\ladybug', projectName='room')

rp.run(debug=False)

results = rp.results(flattenResults=True)

print 'Average illuminance level in this room is {} lux.' .format(sum(results) / len(results))
from honeybee.room import Room
from honeybee.radiance.material.glass import GlassMaterial
from honeybee.radiance.sky.certainIlluminance import SkyWithCertainIlluminanceLevel
from honeybee.radiance.recipe.gridbased import HBGridBasedAnalysisRecipe

# create a test room
room = Room(origin=(0, 0, 3.2), width=4.2, depth=6, height=3.2, rotationAngle=45)

# add a window to the back wall
room.addFenestrationSurface(wallName='back', width=2, height=2, sillHeight=0.7)

# add another window with custom material. This time to the right wall
glass_60 = GlassMaterial.bySingleTransValue('tvis_0.6', 0.6)
room.addFenestrationSurface('right', 4, 1.5, 1.2, radianceMaterial=glass_60)
# run a grid-based analysis for this room
# generate the sky

```python
sky = SkyWithCertainIlluminanceLevel(illuminanceValue=2000)
```

# generate grid of test points
```python
testPoints = room.generateTestPoints(gridSize=0.5, height=0.75)
```

# put the recipe together
```python
rp = HBGridBasedAnalysisRecipe(sky=sky, pointGroups=(testPoints,),
                              simulationType=0, hbObjects=(room,))
```

# write and run the analysis
```python
rp.writeToFile(targetFolder='c:\ladybug', projectName='room')
rp.run(debug=False)
```
results = rp.results(flattenResults=True)
print 'Average illuminance level in this room is {} lux.' \
    .format(sum(results) / len(results))

Number of total materials: 5
Number of total surfaces: 1
Files are written to: c:\ladybug\room\gridbased
C:\Users\Administrator\Documents\GitHub\hydrashare.github.io>c:
C:\Users\Administrator\Documents\GitHub\hydrashare.github.io>cd c:\ladybug\room\gridbased
c:\ladybug\room\gridbased>c:\radiance\bin\oconv -f c:\ladybug\room\gridbased\Uniform_CIE_2000.sky
c:\ladybug\room\gridbased\room.mat c:\ladybug\room\gridbased\room.rad 1>room.oct
c:\ladybug\room\gridbased>c:\radiance\bin\rtrace -aa 0.25 -ab 2 -dj 0.0 -ad 512 -ss 0.0 -h -dc 0.25 -st 0.85 -lw 0.05 -as 128 -ar 16 -lr 4 -l -dt 0.5 -dr 0 -ds 0.5 -dp 64 -e error.txt c:\ladybug\room\gridbased\room.oct 0<c:\ladybug\room\gridbased\room.pts 1>room.res
Average illuminance level in this room is 158.901223225 lux.
```python
import sys
sys.path.extend(IN[0])

try:
    from honeybee.radiance.recipe.gridbased import GridBasedAnalysisRecipe
    from honeybee import wrapper
except ImportError:
    raise ImportError("Failed to import Honeybee libraries. 
    Set-up path to Honeybee libraries in honeybee\_plugins\_core component")

HBsky, _testPoints, ptsVectors, _simType, _radParameters = IN[1:]

if HBsky and _testPoints and ptsVectors:
    _radParameters = _radParameters.unwrap()

analysisRecipe = GridBasedAnalysisRecipe(HBsky.unwrap(),_testPoints, ptsVectors, _simType, _radParameters)

OUT = wrapper.Wrapper(analysisRecipe)
```
Goals
Goals

Enable seamless Cross-OS compatibility
A 3 Phase Method Workflow in Windows

1. System Config: Remote Cluster, Xeon, 40-Core

2. `numProcessors=40`)

3. File list with sizes:
   - `contrib.exe`: 4,512K
   - `confhost.exe`: 2,492K
   - `cmd.exe": 1,576K
   - `tasking.exe`: 3,330K
   - `rfuxmltv.exe`: 3,404K
The same script in a Unix environment

System Config: Desktop, i7, 8-core
Behind the scenes: genskyvec(.pl)

```perl
genskv = Genskyvec()
genskv.inputSkyFile = r'temp/sky.rad'
genskv.outputFile = r'temp/sky.vec'
genskv.skySubdivision = 4
genskv.execute()
```

"C:\Program Files\OpenStudio 1.11.0\strawberry-perl-5.16.2.1-32bit-portable-reduced\perl\bin\perl.exe"
"C:\Program Files\OpenStudio 1.11.0\share\openStudio\Radiance\bin\genskyvec.pl" -m 4  < temp\sky.rad > temp\sky.vec

/gpfs/home/sxs1106/work/Radiance/bin/genskyvec  -m 4  < temp/sky.rad > temp/sky.vec

PERL Interpreter  Genskyvec  Inputs
Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions
Human readable inputs for infrequently used commands

Code completion on compatible IDEs. (API remembers and prompts for inputs)

Look-up class documentation while coding workflows.
A self-documenting API

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>honeybee.radience</td>
<td>Honeybee Radiance libraries.</td>
</tr>
<tr>
<td>honeybee.radience.command</td>
<td>Descriptors, factory classes etc for the Radiance library.</td>
</tr>
<tr>
<td>honeybee.radience.datatype</td>
<td>A collection of auxiliary functions for working with radiance files and objects.</td>
</tr>
<tr>
<td>honeybee.radience.geometry</td>
<td>A collection of methods for writing Radiance geometry file.</td>
</tr>
<tr>
<td>honeybee.radience.material</td>
<td></td>
</tr>
<tr>
<td>honeybee.radience.parameters</td>
<td></td>
</tr>
<tr>
<td>honeybee.radience.postprocess</td>
<td></td>
</tr>
<tr>
<td>honeybee.radience.properties</td>
<td></td>
</tr>
</tbody>
</table>

Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions

Type and Error Checking
Gensky Manpage
radsite.lbl.gov/radiance/man_html/gensky.1.html

"Values less than 1.0 are physically impossible."

Gensky Command

genskPar = GenskyParameters()
genskPar.turbidity = 0.01
gensk = Gensky()
gensk.genskyParameters = genskPar
gensk.monthDayHour = (11,11,11)
gensk.outputFile = 'temp/sky.rad'
gensk.execute()

Error Message

Traceback (most recent call last):
  File "C:\Users\Arith\Projects\honeybee\scripts\ThreePhaseIllum.py", line 153, in <module>
    phasesToCalculate=phases, epwFile=epwFiles[1])
  File "C:\Users\Arith\Projects\honeybee\scripts\ThreePhaseIllum.py", line 113, in run3pha:  
    genskPar.turbidity = 0.01
  File "C:\Users\Arith\Projects\honeybee\scripts\Radiance\phases\parameter_frozen.py", line 63, in object.__setattr__
    key, value)
  File "C:\Users\Arith\Projects\honeybee\scripts\Radiance\data\types.py", line 368, in __setitem__
    raise ValueError(msg)

ValueError: The specified input for t (turbidity) is 0.01. This is below the valid range. The
Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions

Type and Error Checking

Leverage Python to reduce effort
rfluxPara = RfluxmtxParameters()
rfluxPara.I = True
rfluxPara.aa = 0.1
rfluxPara.ad = 4096
rfluxPara.ab = 12
rfluxPara.lw = 0.0000001
rflux = Rfluxmtx()
skyFile = rflux.defaultSkyGround(r'temp/rfluxSky.rad',skyType='r4')
rflux.receiverFile = skyFile
rflux.sender = '-'
rflux.rfluxmtxParameters = rfluxPara
rflux.radFiles = [r'room.mat', r'room.rad']
rflux.pointsFile = r'indoor_points.pts'
rflux.outputMatrix = r'temp/dayCoeff.dc'
rflux.execute()

Radiance

rfluxmtx" -y 6 -aa 0.1 -ab 12 -ad 4096 -lw 1e-07 -I - temp\rfluxSky.rad room.mat room.rad < indoor_points.pts > temp\dayCoeff.dc
Automate predictable tasks: Create a sky definition

1. Create sky definition
2. Add rfluxmtx comments
3. Assign sky as “sender”
Automate predictable tasks: Count a points-file

1. Count the number of points

```python
rfluxPara = RfluxmtxParameters()
rfluxPara.I = True
rfluxPara.aa = 0.1
rfluxPara.ad = 4096
rfluxPara.ab = 12
rfluxPara.lw = 0.0000001
rflux = Rfluxmtx()
skyFile = rflux.defaultSkyGround(r'temp/rfluxSky.rad',skyType='r4')
rflux.receiverFile = skyFile
rflux.sender = '-'
rflux.rfluxmtxParameters = rfluxPara
rflux.radFiles = [r"room.mat",r"room.rad"]
rflux.pointsFile = r"indoor_points.pts"
rflux.outputMatrix = r"temp/dayCoeff.dc"
rflux.execute()
```

2. Assign the number of points in the command
```bash
rfluxmtx -y 6 -aa 0.1 -ab 12 -ad 4096 -lw 1e-07 -I - temp\rfluxSky.rad room.mat room.rad < indoor_points.pts > temp\dayCoeff dc
```
Make the workflow more flexible

rfluxPara = RfluxmtxParameters()
rfluxPara.I = True
rfluxPara.aa = 0.1
rfluxPara.ad = 4096
rfluxPara.ab = 12
rfluxPara.lw = 0.000001
rflux = Rfluxmtx()
skyFile = rflux.defaultSkyGround(r'temp/rfluxSky.rad',skyType='r4')
rflux.receiverFile = skyFile
rflux.sender = '-'
rflux.rfluxmtxParameters = rfluxPara
rflux.radFiles = [r'room.mat', r'room.rad']
rflux.pointsFile = r'indoor_points.pts'
rflux.outputMatrix = r'temp/dayCoeff.dc'
rflux.execute()

Inputs can be in any order.

rfluxmtx" -y 6 -aa 0.1 -ab 12 -ad 4096 -lw 1e-07 -I - temp\rfluxSky.rad room.mat room.rad < indoor_points.pts > temp\dayCoeff.dc

The ordering of inputs needs to follow a certain order.
rfluxmtx [ -v ][ rcontrib options ] { sender.rad | - } receivers.rad [ -i system.oct ] [ system.rad .. ]
Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions

Type and Error Checking

Leverage Python to reduce effort

Interoperability between simulation platforms
from honeybee.energyplus import filemanager, geometryrules
from honeybee.hbsurface import HBSurface
from honeybee.hbzonename import HBZoneName
from honeybee.hbconstruction import HBConstruction
from honeybee.hbzone import HBZone
import os

def idfToRadString(idfFilePath):
    """Convert an idf file geometry to a radiance definition."

    Radiance materials are assigned based on surface types and not from
    EnergyPlus materials or construction. You can create and map your
    own Radiance materials by adding a few number of lines to the code.
    """
    objects = filemanager.getEnergyPlusObjectsFromFile(idfFilePath)

    # if the geometry rules is relative then all the points should be added
    # to X, Y, Z of zone origin
    geoRules = geometryrules.GlobalGeometryRules(
        "objects["globalgeometryrules"].values()[0][1:4]
    )

    hzoObjects = {'zone': {}, 'buildingsurface': {}, 'shading': {}}

    # create zones
    for zonename, zoneData in objects['zone'].iteritems():
        # create a HBZone
        zone = HBZone.fromEPString("", join(zoneData), geometryRules=geoRules)
        hzoObjects['zone'][zonename] = zone

    # create surfaces
    for surfacename, surfaceData in objects['buildingsurface: detailed'].iteritems():
        surface = HBSurface.fromEPString("", join(surfaceData))
        surface.parent = hzoObjects['zone'][surfacedata[0]]
        hzoObjects['buildingsurface'][surfacedata[1]] = surface
Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions

Type and Error Checking

Leverage Python to reduce effort

Interoperability between simulation platforms

Apply Object Oriented Programming (OOP) where required
Radiance syntax alludes to OOP patterns

NAME
rcontrib - compute contribution coefficients in a RADIANCE scene

SYNOPSIS
rcontrib [ -n nprocs ] [ -V ] [ -c count ] [ -fo ] [ -r ] [ -e expr ] [ -f source ] [ -o ospec ] [ -p p1=V1,p2=V2 ] [ -b binc ] [ -bn nbins ] [ -m mod ] [ -M file ] [ @file ] [ rtrace options ] octree
rcontrib [ options ] -defaults

NAME
rfluxmtx - compute flux transfer matrix(es) for RADIANCE scene

SYNOPSIS
rfluxmtx [ -V ] [ rcontrib options ] { sender.rad [- ] } receivers.rad [ -i system.oct ] [ system.rad .. ]

DESCRIPTION

NAME
vwrays - compute rays for a given picture or view

SYNOPSIS
vwrays [ -i -u -f[a|f|d] -c rept | -d ] { view opts .. | picture [zbuf] }

DESCRIPTION
Vrays takes a picture or view specification and computes the ray origin and direction corresponding to
Honeybee “wraps” Radiance syntax through OOP

**Base Class**
Implements `rtrace` options ab, ad, as etc.

```python
@frozen
class GridBasedParameters(AdvancedRadianceParameters):
    ..."""Radiance Parameters for grid based analysis...."""

    def init (self, quality=None):
```

**Subclass**
Implements `rcontrib`.

```python
@frozen
class RcontribParameters(GridBasedParameters):
    ..."""Radiance Parameters for rcontrib command including rtrace parameters.

    def init (self, modFile=None):
```

**Subclass**
Implements `rfluxmtx`.

```python
@frozen
class RfluxmtxParameters(GridBasedParameters):
    ..."""Init parameters."""
    ...def init (self, sender=None, receiver=None, octree=None, systemFile
    ...,...):
```
Goals

Enable seamless Cross-OS compatibility

Simplify syntax with abstractions

Type and Error Checking

Leverage Python to reduce effort

Interoperability between simulation platforms

Apply Object Oriented Programming (OOP) where required

Recipes!
Use the API to create “recipes” for common workflows

```python
def runPhase(phasesToCalculate='v':True,'t':True,'d':True,'s':True,
calculationType='annual',epwFile=None,matrixFile=None):
    if phasesToCalculate['v']:
        # Step 1: Create the view matrix.
        rfluxPara = RfluxMatrixParameters()
        rfluxPara.I = True
        rfluxPara.sS = 0.1
        rfluxPara.ab = 10
        rfluxPara.sD = 65B36
        rfluxPara.lw = 1E-5

        # Step 1.1 Invert glazing surface with xform so that it faces inwards
        xfrPara = XformParameters()
        xfrPara.invertSurfaces = True

        xfr = Xform()
        xfr.xformParameters = xfrPara
        xfr.radFile = 'glazing.rad'
        xfr.outputFile = 'glazingI.rad'
        xfr.run()

        rflux = RfluxMatrix()
        rflux.sender = '-'

        # Lens full basis sampling and the window faces +z
        recCtrlPar = rflux.ControlParameters(hemiType='kf',hemiUpDirection='+z')
        rflux.receiverFile = rflux.addControlParameters('glazingI.rad',
                                                        {'Exterior Window': recCtrlPar})
        rflux.rfluxMatrixParameters = rfluxPara
        rflux.pointsFile = 'indoor_points.txt'
```
Use “recipes” independently or through 3D Environments

ThreePhaseIllum.py

ThreePhaseImage.py

DayCoeffImage.py

DayCoeffImage.py

PhotonMapImage.py

FivePhaseImage.py
Live Demo
How to get involved?

https://github.com/ladybug-analysis-tools
Thank You!