

Importing and Mapping Model Load Data Using ANSYS DPF

Alex Grishin₁

Nathan Hays₂

1. Consulting Engineer, [PADT, Inc.](http://www.padtinc.com) (alex.grishin@padtinc.com)
2. Owner and Principle Researcher, [nuLUCA](http://www.nuluca.com) (nate@nuluca.com)



Background And Introduction

- In this blog post, we thought we'd do something a bit different
- Recently, PADT worked with Nathan Hays of [nuLUCA](#) to help analyze a unique bio-mimetic residential tile design (and develop a process for doing this frequently)
- nuLUCA uses the Rhino suite of tools for geometry and mesh generation.
- As ANSYS does not have specific interfaces for these tools, transferring the data may be a challenge.
- PADT thought that this was perfect opportunity to harness some of ANSYS' free Python DPF modules for this purpose.
- So, what follows are the fruits of the combined efforts of nuLUCA and PADT towards this goal.
- And for Rhino users who who are less familiar with Python, we outline an analgous procedure which avoids DPF entirely in the Appendix

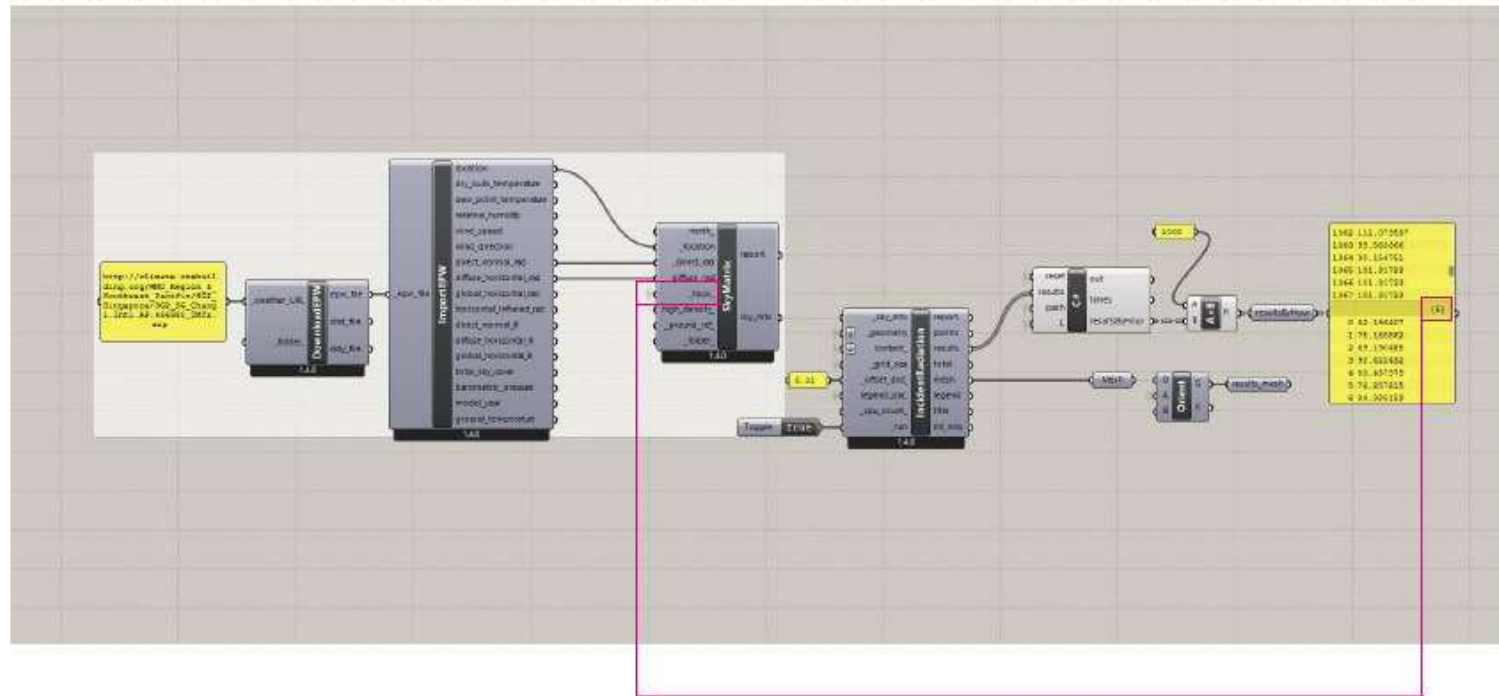
- Enjoy!



The Grasshopper Workflow

- Users of [Rhino's Grasshopper plugin](#) have access to powerful visual scripting tools to facilitate design. The example we're using in this post was created with this tool.
- An additional Grasshopper plugin we used is called [Ladybug](#). This is used to define environmental (climate) data for Grasshopper designs.
- The workflow for this example is shown below

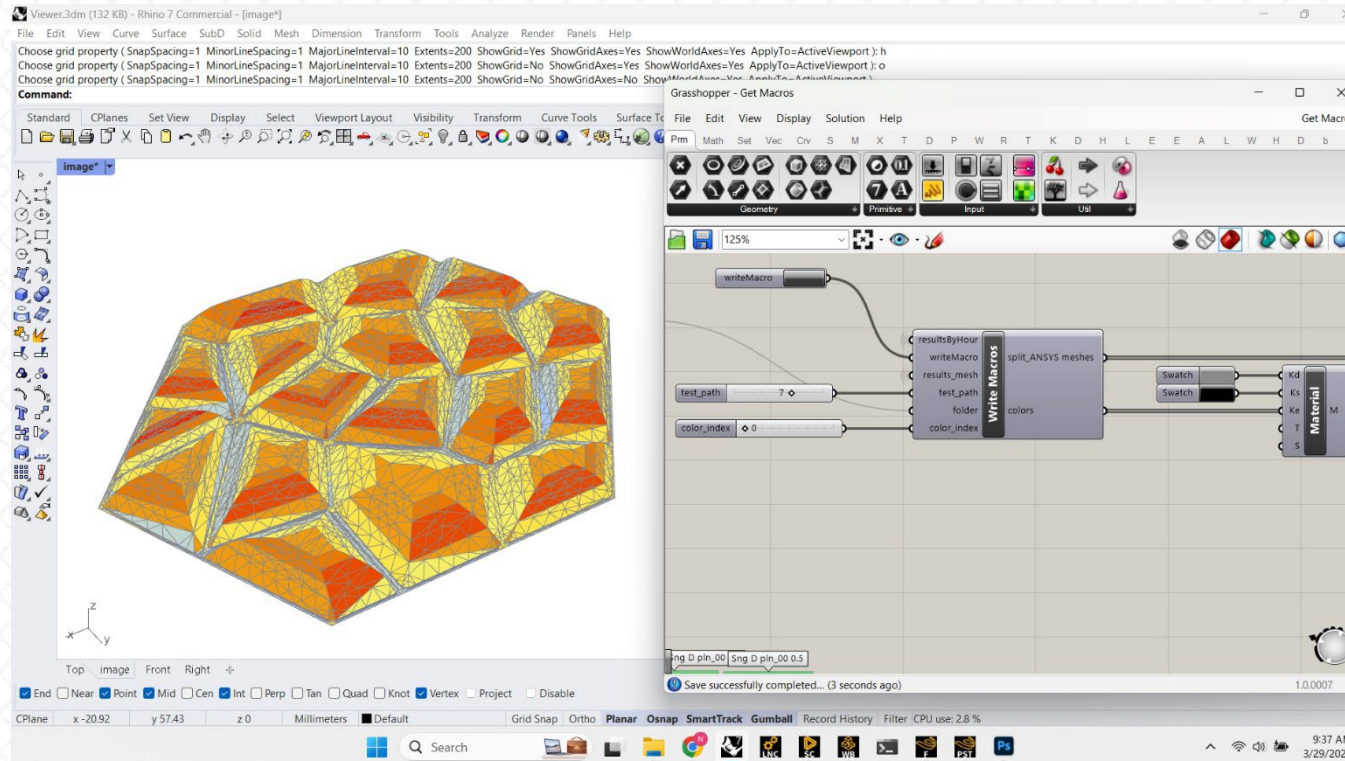
- Create a SkyMatrix to plug into Ladybug's IncidentRadiation component. This component gives us two sets of data we'll need: A new mesh created by the incidentRadiation components that represents the surface of the geometry we're analyzing, and a list of doubles –the indices of which correspond to the face indices of the new mesh, and the values of which are the incident heat flux values



- We can iterate through the variables hoys (hours of the year) – each time storing the results (IR values by face) in a DataTree, with each branch containing a list of IR values

The Grasshopper Workflow

- Below is an example of a design created by this workflow
- In what follows, we will describe the procedures involved in transferring this data to an ANSYS model (Nathan has translated the entire process in c# scripts executable from Rhino*, but we'll use ANSYS DPF instead)



*See the Appendix for more details



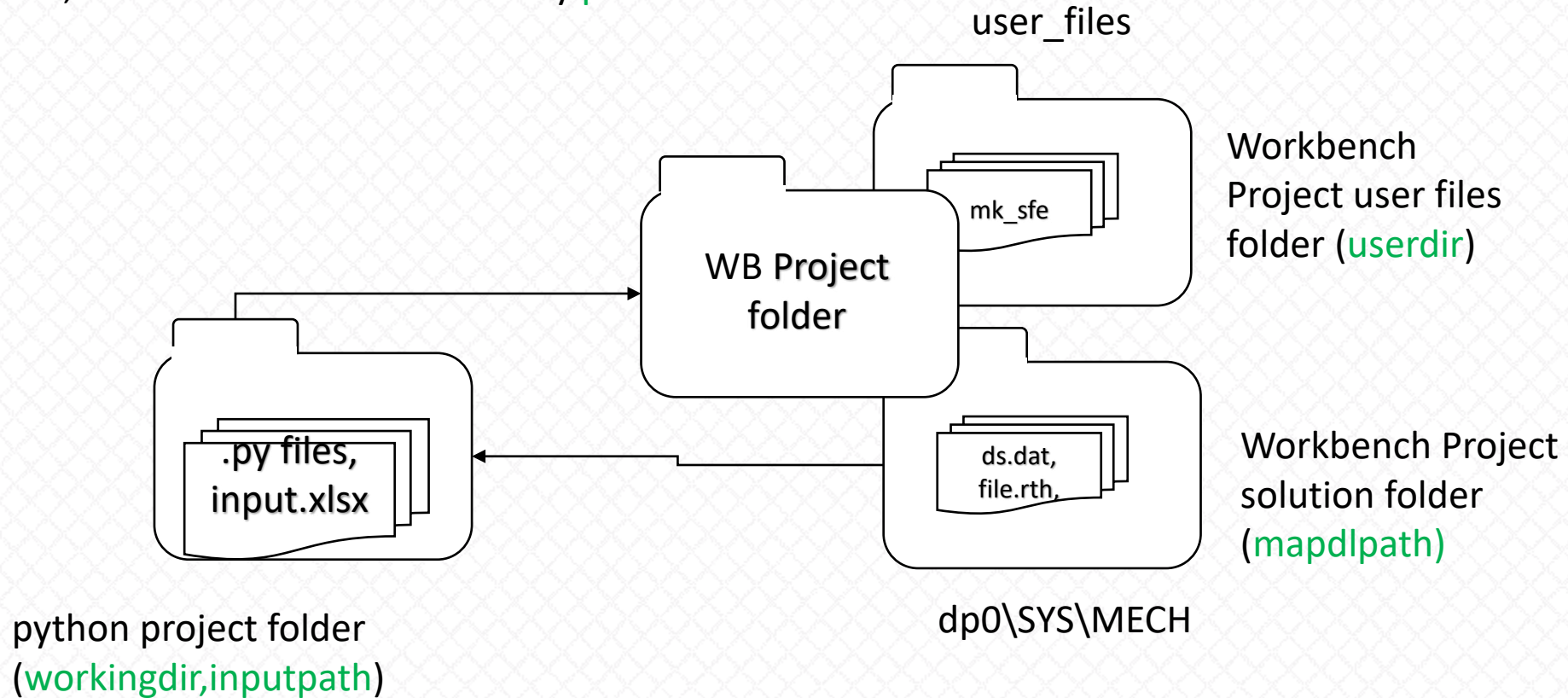
Importing (mapping) External Data in ANSYS

- In a [previous post](#) we demonstrated how to import a spatially and temporally varying load in ANSYS Workbench using the External Data Tool
- Among other things, readers should have learned that importing spatially varying data over multiple time steps can be quite time consuming and inefficient using the external data tool
- One alternative is to insert an MAPDL script to define the tabular loading, as shown [in this post](#)
- There are other cases, however, in which users may have to map spatially and temporally varying data in a non-structured form (unstructured mesh data, for example, in which constructing a simple table to describe the data is not possible) that doesn't conform to their mesh.
- In such cases, users may again turn to MAPDL scripting (see the MAPDL commands: CBDOF, BFINT, as well as *moper,,,map)
- However, there are cases where even the solutions above will prove difficult, such as when mapping element loads (whether volumetric or surface effect) from one mesh to another over multiple time steps (and this is especially true for very large models)
- In this article, we'd like to offer a relatively painless solution in such cases –especially when mapping data from outside ANSYS. And this method scales nicely with model size
- To do this, we'll use the powerful mesh utilities of PyVista which ships with pyMAPDL and pyDPF-Post (see our [previous articles on using PyMAPDL and PyDPF-Post](#))



The Project Setup

- As in previous posts, we'll use Python with an ANSYS Workbench model, and so our data pipeline will look something like the one below
- So, we'll first define all necessary **path variables**



Import Heat Flux Data

- So, launch a new Python session in your editor of choice
- We'll also use the "launch_mapdl" module of `ansys.mapdl.core` in order to directly read in the mesh
- In this example, we'll read and apply heat flux data provided in the form of a spreadsheet. In order to make this painless, readers should import a free module for this purpose, like [openpyxl](#) or [pandas](#). [Since pandas is by far the most popular Python module](#) for doing this, we'll use that.
- With Pandas installed, issue the following lines of Python to define file paths and import the spreadsheet data...

```
from ansys.mapdl.core import launch_mapdl
import numpy as np
import pyvista as pv
import pandas as pd
```

```
workingdir = r"C:\Users\alex.grishin\focus_article"
projectdir = workingdir + f"\example_files"
userdir = projectdir + f"\user_files"
mapdlpath = projectdir + f"\dp0\SYS\MECH\ds.dat"
inputpath = workingdir + f"\nuLUCA_tile IR data.xlsx"
```

```
#read in Rhino model mesh vertices
verts = pd.read_excel(inputpath, sheet_name='Mesh Vertex List', index_col=0).to_numpy()
#read in Rhino model surface facet connectivity
faces = pd.read_excel(inputpath, sheet_name='Mesh Face List', index_col=0).to_numpy()
#read in Ladybug radiation data
incrad = pd.read_excel(inputpath, sheet_name='Incident Radiation', index_col=0).to_numpy()
```



- Change these paths to your locations

Import Heat Flux Data

- The previous lines of code use Pandas to read in the data found in the spreadsheet worksheets “Mesh Vertex List”, “Mesh Face List”, and “incident Radiation”
- The code imports the data found there and stores it in a numpy array
- Note that we’re relying on Pandas’ default logic for determining the location and extent of our data (using the first row and column as header and indices, respectively)

Vertex Index	x (mm)	y (mm)	z (mm)
0	50.000015	0	-0.400979
1	50	0	-11.241008
2	49.096569	1.564809	-0.400946
3	49.096569	-1.564809	-0.400946
4	49.09656	1.564804	-11.241008
5	49.09656	-1.564804	-11.241008
6	48.820736	1.405559	-0.400983
7	48.820736	-1.405559	-0.400983
8	48.820724	1.405559	0.00003
9	48.820724	-1.405559	0.00003
10	48.581104	2.457635	-0.400952
11	48.581089	2.457626	-11.241008
12	48.305271	2.298385	-0.400989
13	48.30526	2.298384	0.000025
14	48.255589	-0.728443	2.107427
15	48.125195	-1.278988	2.366634
16	48.065636	3.350462	-0.400957
17	48.065616	3.350451	-11.241008
18	47.819599	1.537097	1.983334
19	47.789791	3.191212	0.000019
20	47.789776	3.191212	-0.400983
21	47.689793	2.129379	1.76254
22	47.57777	-4.195425	-0.401

Face Index	Pt A (Index)	Pt B (Index)	Pt C (Index)
0	562	563	597
1	529	530	563
2	462	463	506
3	388	389	463
4	505	506	530
5	316	317	344
6	291	292	317
7	225	226	256
8	171	172	199
9	198	199	226
10	255	256	292
11	343	344	389
12	597	596	562
13	563	562	529
14	506	505	462
15	463	462	388
16	505	530	529
17	344	343	316
18	317	316	291
19	256	255	225
20	199	198	171
21	198	226	225
22	255	292	291

Face Index	7am IR (kWh/m2)	8am IR (kWh/m2)	9am IR (kWh/m2)	10am IR (kWh/m2)	11am IR (kWh/m2)	12pm IR (kWh/m2)
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	0	0	0	0	0	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0
19	0	0	0	0	0	0
20	0	0	0	0	0	0
21	0	0	0	0	0	0
22	0	0	0	0	0	0
23	0	0	0	0	0	0
24	0	0	0	0	0	0

- So, we can query the arrays we just imported to instantly discover the size of the source mesh, as well as the the times we have heat flux data for



Import Heat Flux Data

- Use the numpy.array shape method on each of the imported arrays to ensure everything was read in:

```
In [10]: verts.shape
Out[10]: (2937, 3)
```

Vertex Index	x (mm)	y (mm)	z (mm)
0	50.000015	0	-0.400979
1	50	0	-11.241008
2	49.090569	1.564809	-0.400946
3	49.090569	-1.564809	-0.400946
4	49.09056	1.564804	-11.241008
5	49.09056	-1.564804	-11.241008
6	48.820736	1.405559	-0.400983
7	48.820736	-1.405559	-0.400983
8	48.820724	1.405559	0.000003
9	48.820724	-1.405559	0.000003
10	48.581104	2.457635	-0.400952
11	48.581089	2.457626	-11.241008
12	48.302271	2.298385	-0.400989
13	48.30526	2.298384	0.000025
14	48.255889	-0.728443	2.107427
15	48.125195	-1.278988	2.366634
16	48.065638	3.350462	-0.400957
17	48.065616	3.350451	-11.241008
18	47.819599	1.537097	1.983334
19	47.789791	3.191212	0.000019
20	47.789776	3.191212	-0.400983
21	47.689793	2.129379	1.76254
22	47.57777	-4.195425	-0.401

- 2937 vertices

```
In [11]: faces.shape
Out[11]: (5870, 3)
```

Face Index	Pt A (Index)	Pt B (Index)	Pt C (Index)
0	562	563	597
1	529	530	563
2	462	463	506
3	388	389	463
4	505	506	530
5	316	317	344
6	291	292	317
7	225	226	256
8	171	172	199
9	198	199	226
10	255	256	292
11	343	344	389
12	597	596	562
13	563	562	529
14	506	505	462
15	463	462	388
16	505	530	529
17	344	343	316
18	317	316	291
19	256	255	225
20	199	198	171

- 5870 facets

```
In [13]: incread.shape
Out[13]: (5870, 12)
```

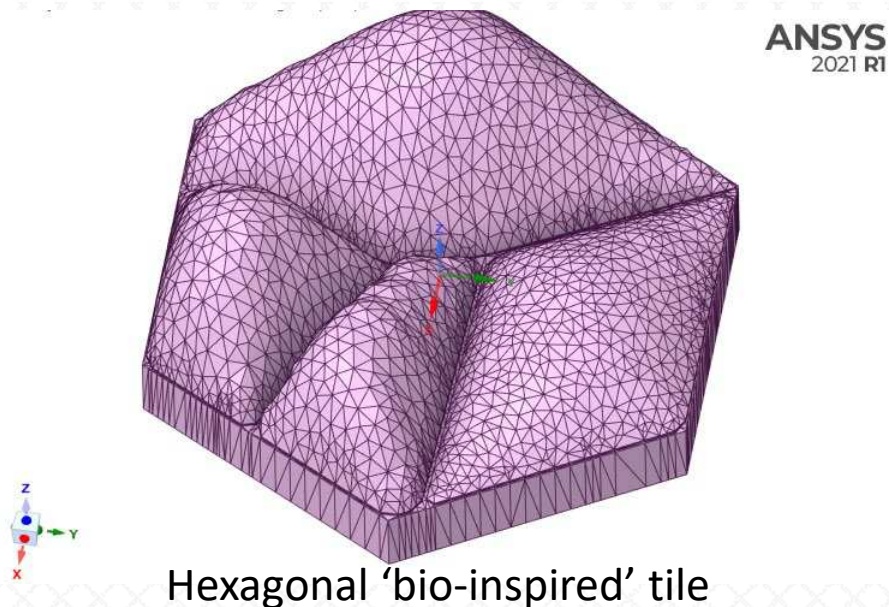
Face Index	7am IR (kWh/m2)	8am IR (kWh/m2)	9am IR (kWh/m2)	10am IR (kWh/m2)	11am IR (kWh/m2)	12pm IR (kWh/m2)
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	0	0	0	0	0	0
17	0	0	0	0	0	0
18	0	0	0	0	0	0
19	0	0	0	0	0	0
20	0	0	0	0	0	0
21	0	0	0	0	0	0
22	0	0	0	0	0	0
23	0	0	0	0	0	0
24	0	0	0	0	0	0

- Incident radiation over 5870 faces over a 12-hour period



Create The ANSYS Model

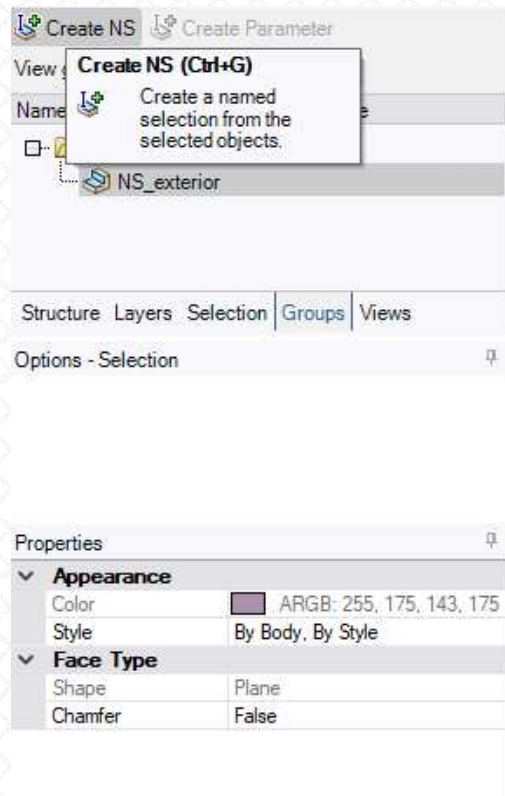
- Next, we have to create the ANSYS model before continuing
- In this example, we have a model of a bio-inspired outdoor tile design for residential housing
- The intent of the design is to dissipate heat more efficiently than more traditional designs
- The thermal load on the tile comes from the terrestrial solar radiant heat flux data we just read in from [Ladybug](#)
- The geometry exists in a CAD-neutral volumetric STEP file created in Rhino as a [manifold solid B-rep](#)
- Nathan at nuLUCA wraps the entire process we're about to describe on the ANSYS-DPF-side completely within `c#` scripts in Grasshopper (see the Appendix)
- But we want to describe the process on the ANSYS-DPF-side in some detail for our readers.



- Since the original geometric algorithms used to create this geometry (not uncommon in tools like Grasshopper) do not have a faithful and robust 'traditional' parametric CAD representation, the volume is represented by a surface-mesh structure as shown on the left (**note:** This is NOT an STL representation, but rather NURBS surface patches)

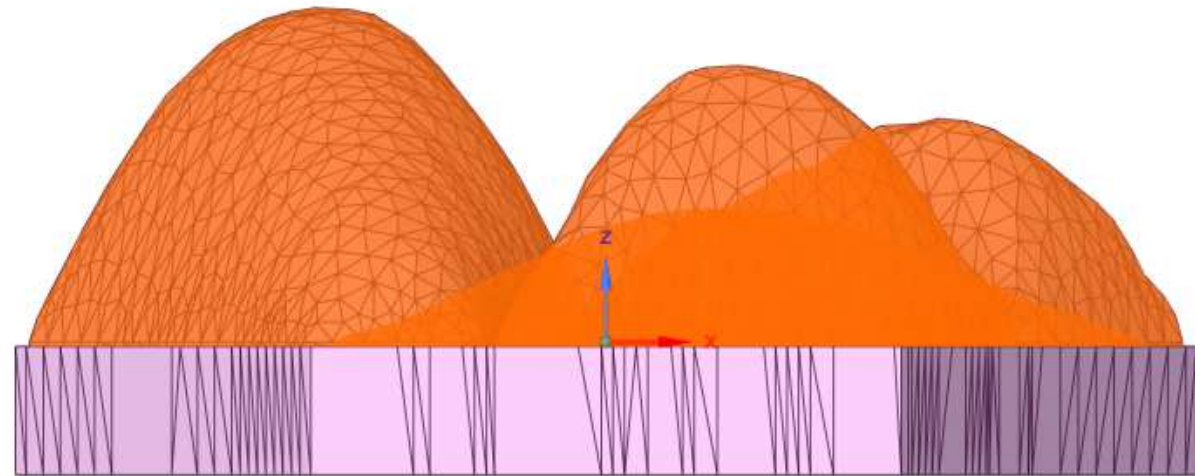
Create The ANSYS Model

- Because of the inconvenient nature of this manifold B-rep structure (each triangular facet is an individual surface), use SpaceClaim to make a named selection of the surfaces shown below (selected by box-select)
- We'll refer to this named selection to apply the thermal loading



Click an object. Double-click to select an edge loop. Triple-click to select a solid.

ANSYS
2021 R1



Create The ANSYS Model

- Once the geometry has been imported, define a Workbench transient heat transfer model
- Select the 'ceramic5' material from the thermal materials library

The screenshot shows the ANSYS Engineering Data Sources and Outline of Thermal Materials panels. The Engineering Data Sources panel is on the left, and the Outline of Thermal Materials panel is on the right. The Outline of Thermal Materials panel shows a list of materials, with 'Ceramic5' selected. Below the list, the Properties of Outline Row 18: Ceramic5 are displayed in a table.

	A	B	C	D	E
1	Data Source		Location		Description
11	Magnetic B-H Curves				magnetic analysis.
12	Thermal Materials				Material samples specific for use i thermal analysis.
13	Fluid Materials				Material samples specific for use i analysis.
*	Click here to add a new library				

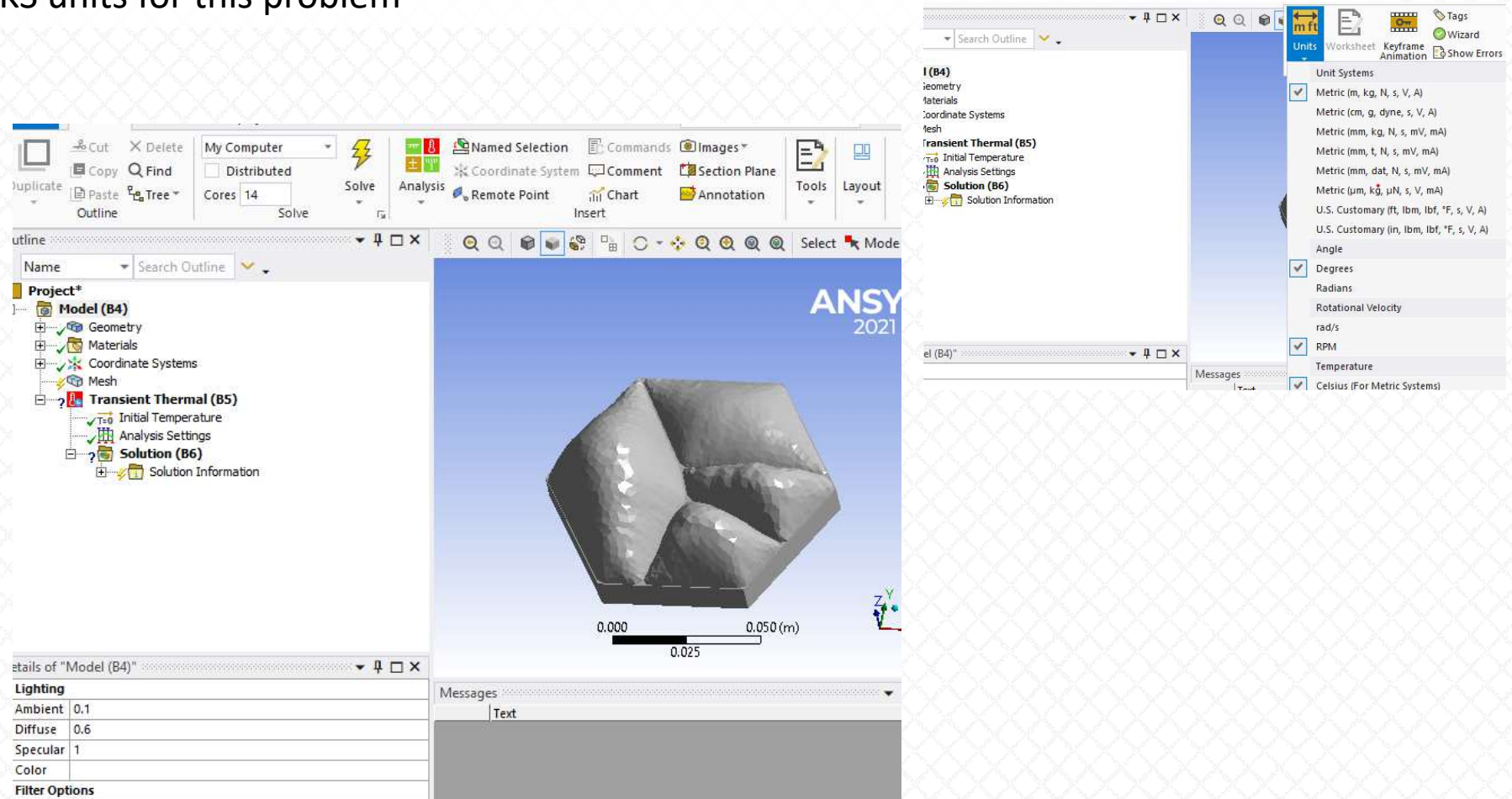
	A	B	C	D	E
1	Contents of Thermal Materials	Add	source		Description
13	Beryllium	+	=		
14	Beryllium Oxide	+	=		
15	Brass	+	=		
16	Bronze	+	=		
17	Cast Iron	+	=		
18	Ceramic5	+	=		

	A	B	
1	Property	Value	
2	Density	4900	kg m ⁻³
3	Isotropic Thermal Conductivity	4.5	W m ⁻¹
4	Specific Heat Constant Pressure, C _p	800	J kg ⁻¹



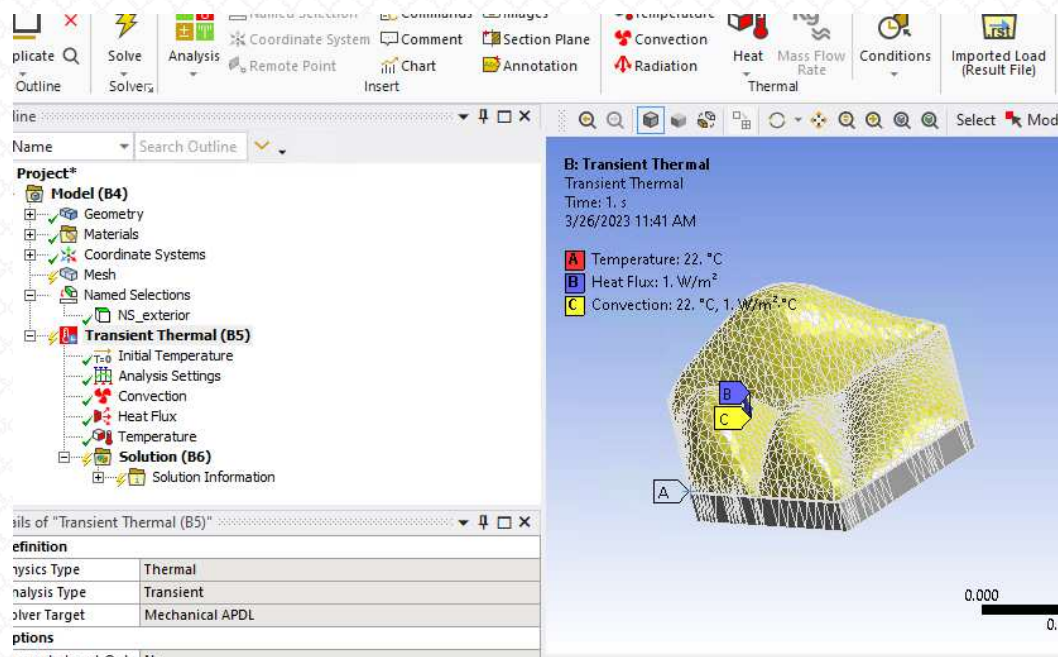
Create The ANSYS Model

- Make sure to use MKS units for this problem

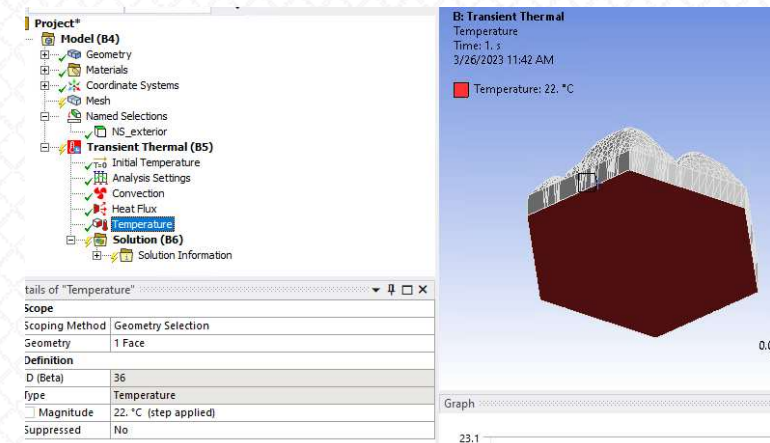


Create The ANSYS Model

- Define a Convection and Heat Flux on the named selection we created earlier
- These are just placeholders –these loads will be replaced with code. The reason for doing this is to create ‘surface-effect’ elements in ANSYS. We will use these elements to interpolate the external heat flux data later

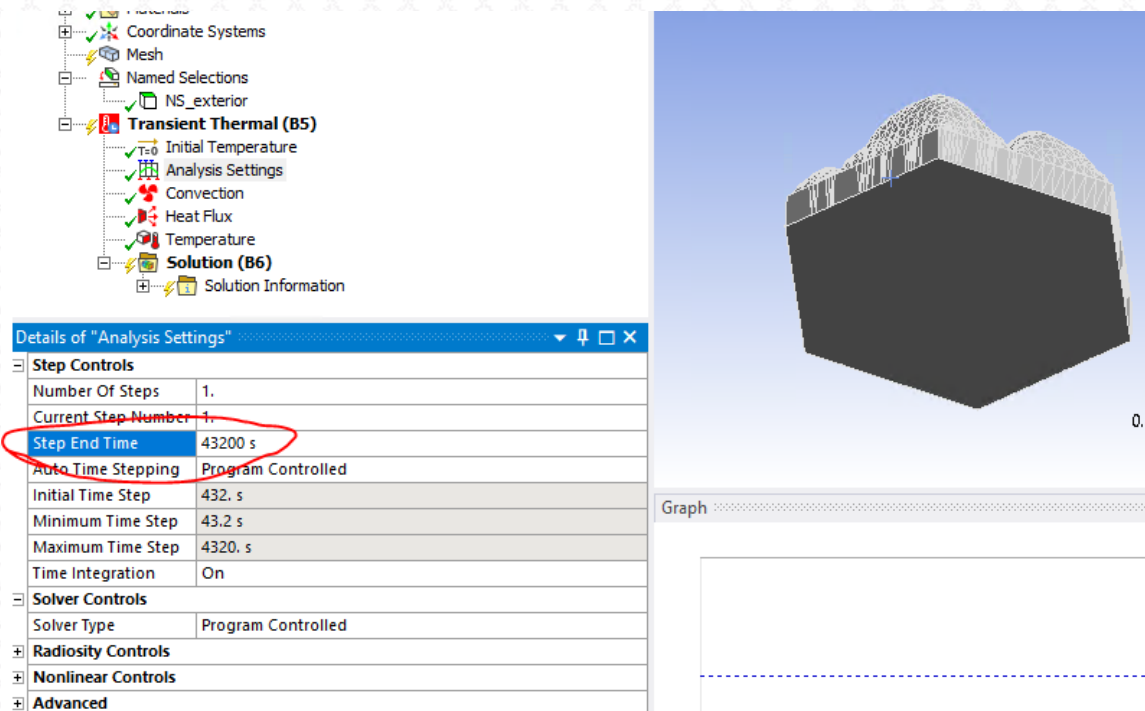


- Also define a fixed ‘interior’ temperature on the flat surface

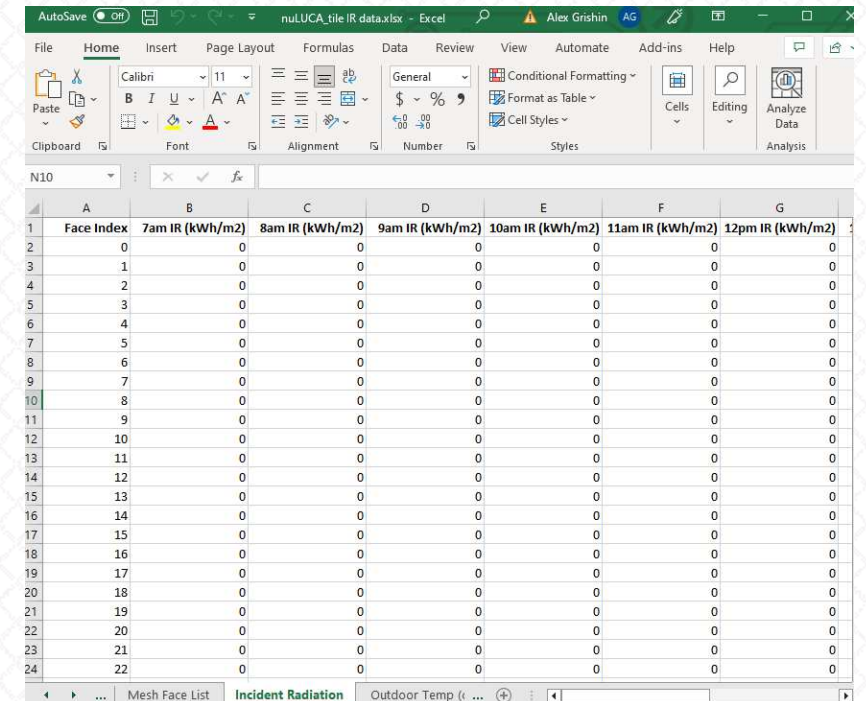


Create The ANSYS Model

- Finally, define the solution end-time to coincide with the end of the heat flux data we imported
- Reviewing that data, we see that we have heat flux data for each of 12 one-hour periods (starting at 7am)
- To keep the project simple, we'll stick with temporal units in seconds, but this means that our analysis will start at 0 seconds and end at $12 * 3600 = 43200$ seconds



The image shows the ANSYS Workbench interface. On the left, the project tree includes 'Transient Thermal (B5)' with sub-items for 'Initial Temperature', 'Analysis Settings', 'Convection', 'Heat Flux', 'Temperature', and 'Solution (B6)'. Below the tree is the 'Details of "Analysis Settings"' panel. In the 'Step Controls' section, the 'Step End Time' is set to 43200 s and is circled in red. Other settings include 'Number Of Steps: 1', 'Current Step Number: 1', 'Auto Time Stepping: Program Controlled', 'Initial Time Step: 432. s', 'Minimum Time Step: 43.2 s', 'Maximum Time Step: 4320. s', and 'Time Integration: On'. The 'Solver Controls' section shows 'Solver Type: Program Controlled'. The 'Radiosity Controls', 'Nonlinear Controls', and 'Advanced' sections are also visible. In the center, a 3D model of a building is shown with a mesh. Below the model is a 'Graph' area.



The image shows a screenshot of an Excel spreadsheet titled 'nuLUCA_tile_IR_data.xlsx'. The spreadsheet contains a table with 24 rows and 7 columns. The columns are labeled 'Face Index', '7am IR (kWh/m2)', '8am IR (kWh/m2)', '9am IR (kWh/m2)', '10am IR (kWh/m2)', '11am IR (kWh/m2)', and '12pm IR (kWh/m2)'. All the data cells in the table contain the value '0'. The spreadsheet is displayed in a window with the ribbon showing 'File', 'Home', 'Insert', 'Page Layout', 'Formulas', 'Data', 'Review', 'View', 'Automate', 'Add-ins', and 'Help'. The 'Home' ribbon is active, showing options for 'Clipboard', 'Font', 'Alignment', 'Number', and 'Styles'.

	A	B	C	D	E	F	G
1	Face Index	7am IR (kWh/m2)	8am IR (kWh/m2)	9am IR (kWh/m2)	10am IR (kWh/m2)	11am IR (kWh/m2)	12pm IR (kWh/m2)
2	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	2	0	0	0	0	0	0
5	3	0	0	0	0	0	0
6	4	0	0	0	0	0	0
7	5	0	0	0	0	0	0
8	6	0	0	0	0	0	0
9	7	0	0	0	0	0	0
10	8	0	0	0	0	0	0
11	9	0	0	0	0	0	0
12	10	0	0	0	0	0	0
13	11	0	0	0	0	0	0
14	12	0	0	0	0	0	0
15	13	0	0	0	0	0	0
16	14	0	0	0	0	0	0
17	15	0	0	0	0	0	0
18	16	0	0	0	0	0	0
19	17	0	0	0	0	0	0
20	18	0	0	0	0	0	0
21	19	0	0	0	0	0	0
22	20	0	0	0	0	0	0
23	21	0	0	0	0	0	0
24	22	0	0	0	0	0	0

Interpolate the Imported Flux Data onto the ANSYS model

- Insert an empty command object
- In the details view, select 'Issue Solve Command, No'
- Run this model (let Workbench create a default mesh)
- Don't worry about the red lightning bolt. Workbench is complaining that it doesn't have any results to postprocess. That's fine (once again, this is just a placeholder for now)!
- The purpose is to just create an input file we can use to import the ANSYS model into Python

The screenshot displays the ANSYS Workbench interface. The Outline panel on the left shows a hierarchy of objects: Model (B4) containing Geometry, Materials, Coordinate Systems, Mesh, Named Selections, and NS_exterior; Transient Thermal (B5) containing Initial Temperature, Analysis Settings, Convection, Heat Flux, and Temperature; Commands (APDL); and Solution (B6) containing Solution Information. The Commands panel on the right shows a list of commands with a red lightning bolt icon next to the 'Issue Solve Command' entry. The Details of 'Commands (APDL)' panel at the bottom left shows the 'Issue Solve Command' property set to 'No'. The Messages panel at the bottom right displays several error and warning messages, including 'An unknown error occurred during solution' and 'The initial time increment may be too large for this problem'.

Text	Association
Error	An unknown error occurred during solution. Check the Solver Output on the Solution In Project>Model>Trans
Warning	The initial time increment may be too large for this problem. Check results carefully. Refe Project>Model>Trans
Warning	The solve command has not been issued for the last step since Issue Solve Command has Project>Model>Trans
Warning	Failed to move file from solver directory to scratch directory: . tmp .inp Project>Model>Trans

Interpolate the Imported Flux Data onto the ANSYS model

- Now, execute the following lines in the open Python session (this will expose the Workbench model for us in Python):

```
#convert units to meters:
verts = verts/1000

#create grid starting at face 705 (the first one whose radiation isn't zero)
radmesh = pv.UnstructuredGrid({5:faces[705:]},verts)

mapdl = launch_mapdl(run_location=mapdlpath,override=True)
mapdl.input("ds.dat")
#mapdl.mesh.ekey #lists the element types
sindex = np.where(mapdl.mesh.grid['ansys_etype']==2)[0]
ansmesh = mapdl.mesh.grid.extract_cells(sindex)

for i in range(incrad.shape[1]):
    dstr = 'F'+str(i)
    #starting at row 705 because rows 0 thru 704 are side and back (no radation)
    radmesh[dstr] = incrad[705:,i]*1000 #convert from kw/m^2 to w/m^2

#Need to extrapolate cell data to vertices before interpolation --that's what the line
below does
radmesh = radmesh.cell_data_to_point_data(pass_cell_data=True)
Fsampled = ansmesh.interpolate(radmesh,sharpness=2,radius=0.005)
Fsampled = Fsampled.point_data_to_cell_data(pass_point_data=True)
```



Interpolate the Imported Flux Data onto the ANSYS model

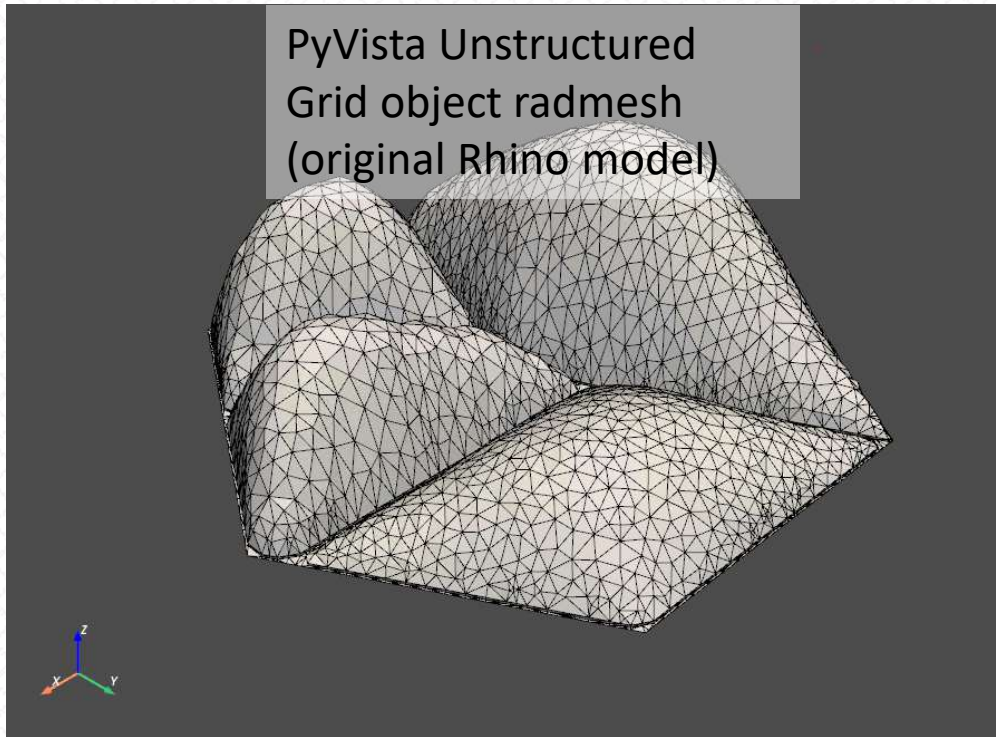
- The lines on the previous slide (lines 20 through 38) do all the 'heavy lifting' for us. So, we'll explain them

```
19 #convert units to meters:
20 verts = verts/1000
21
22 #create grid starting at face 705 (the first one whose radiation isn't zero)
23 radmesh = pv.UnstructuredGrid({5:faces[705:]}),verts)
24
25 mapdl = launch_mapdl(run_location=mapdlpath,override=True)
26 mapdl.input("ds.dat")
27 #mapdl.mesh.ekey #lists the element types
28 sindex = np.where(mapdl.mesh.grid['ansys_etype']==2)[0]
29 ansmesh = mapdl.mesh.grid.extract_cells(sindex)
30
31 for i in range(incrad.shape[1]):
32     dstr = 'F'+str(i)
33     #starting at row 705 because rows 0 thru 704 are side and back (no radiation)
34     radmesh[dstr] = incrad[705:,i]*1000 #convert from kw/m^2 to w/m^2
35
36 #Need to extrapolate cell data to vertices before interpolation --that's what the line below does
37 radmesh = radmesh.cell_data_to_point_data(pass_cell_data=True)
38 Fsampled = ansmesh.interpolate(radmesh,sharpness=2,radius=0.005)
39 Fsampled = Fsampled.point_data_to_cell_data(pass_point_data=True)
```

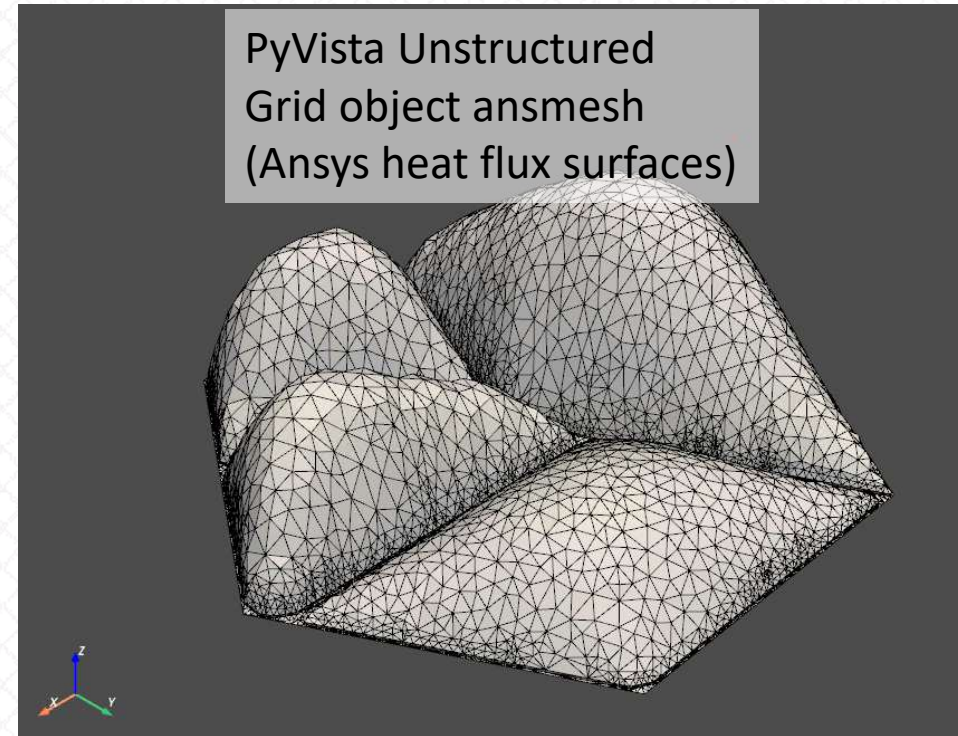
- Create a PyVista unstructured grid object (radmesh) from the Rhino tile model defined in the spreadsheet
- Create a second PyVista unstructured grid object from the elements on which to apply a heat flux in the Workbench model (ansmesh)
- Fill the radmesh object with the incident radiation data
- Interpolate the data from the Rhino mesh to the ANSYS mesh

Interpolate the Imported Flux Data onto the ANSYS model

- You can view the Pyvista unstructured grid objects...



```
In [30]: radmesh.plot(show_edges=True,color='white')
```



```
In [31]: ansmesh.plot(show_edges=True,color='white')
```


Interpolate the Imported Flux Data onto the ANSYS model

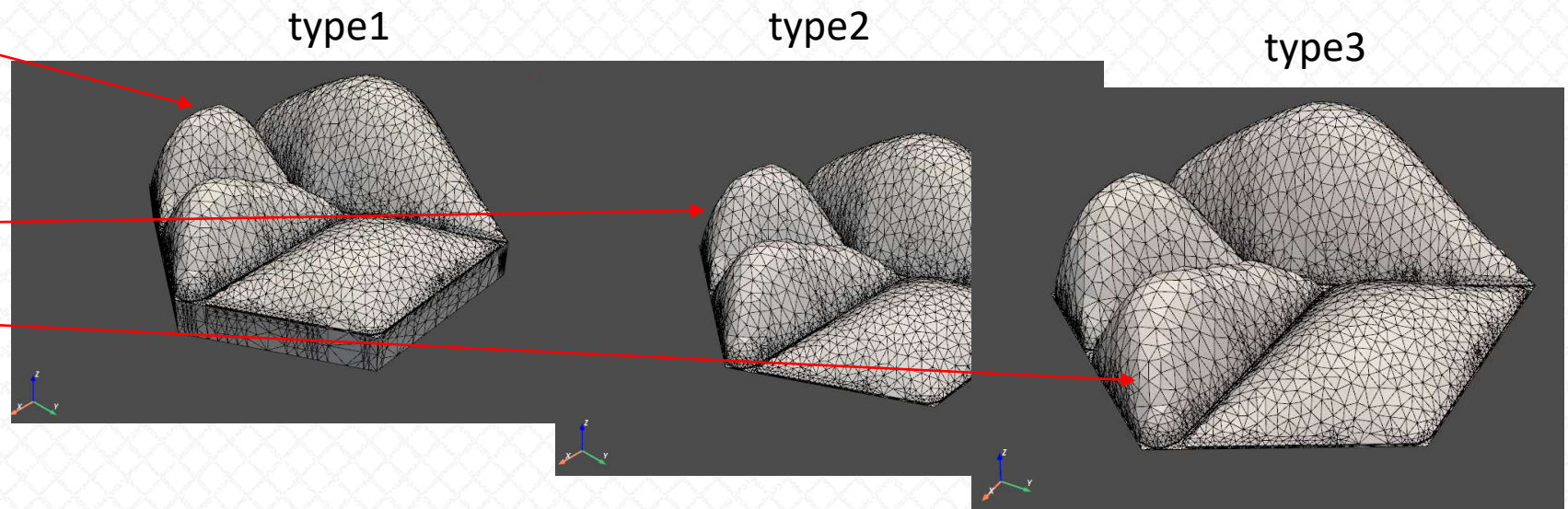
- From the ANSYS model, we're only interested in the 'surface effect' elements
- Lines 28 and 29 of slide 18 isolate and extract these from the larger ANSYS model.
- After executing line 26 (reading the Workbench ds.dat file), ansys.mapdl.core re-creates a faithful PyVista representation of the ANSYS model stored in the 'mesh' object
- We can quickly query various properties of the mesh. Users may be curious how we knew that we only need extract element type 2, for example. The mesh.ekey array contains an array listing of element types in the model
- Element type 152 are surface effect elements. Users can quickly select any or all of the element types in the model the way we do in lines 28 and 29 and plot them. It should then be clear by visual inspection which element types are the ones we're after

```
In [79]: mapdl.mesh.ekey
Out[79]:
array([[ 1, 291],
       [ 2, 152],
       [ 3, 152]])
```

```
In [93]: type1index = np.where(mapdl.mesh.grid['ansys_etype']==1)[0]
In [94]: type2index = np.where(mapdl.mesh.grid['ansys_etype']==2)[0]
In [95]: type3index = np.where(mapdl.mesh.grid['ansys_etype']==3)[0]
```

```
In [90]: type1 = mapdl.mesh.grid.extract_cells(type1index)
In [91]: type2 = mapdl.mesh.grid.extract_cells(type2index)
In [92]: type3 = mapdl.mesh.grid.extract_cells(type3index)
```

- heat flux
- convection



Interpolate the Imported Flux Data onto the ANSYS model

- In line 38 on slide 18, we interpolated the incident radiation data (which exists in the form of 12 arrays within the unstructured grid object) onto the ansys mesh and renamed the result to a new unstructured grid object called 'Fsampled'
- We can verify this by typing 'Fsampled.array_names'. You should see all the original array data (supplied by dpf.mapl.core), as well as the twelve radiation arrays...

```
36 | #Need to extrapolate cell data to vertices before interpolation --that's what the line below does
37 | radmesh = radmesh.cell_data_to_point_data(pass_cell_data=True)
38 | Fsampled = ansmesh.interpolate(radmesh,sharpness=2,radius=0.005)
```

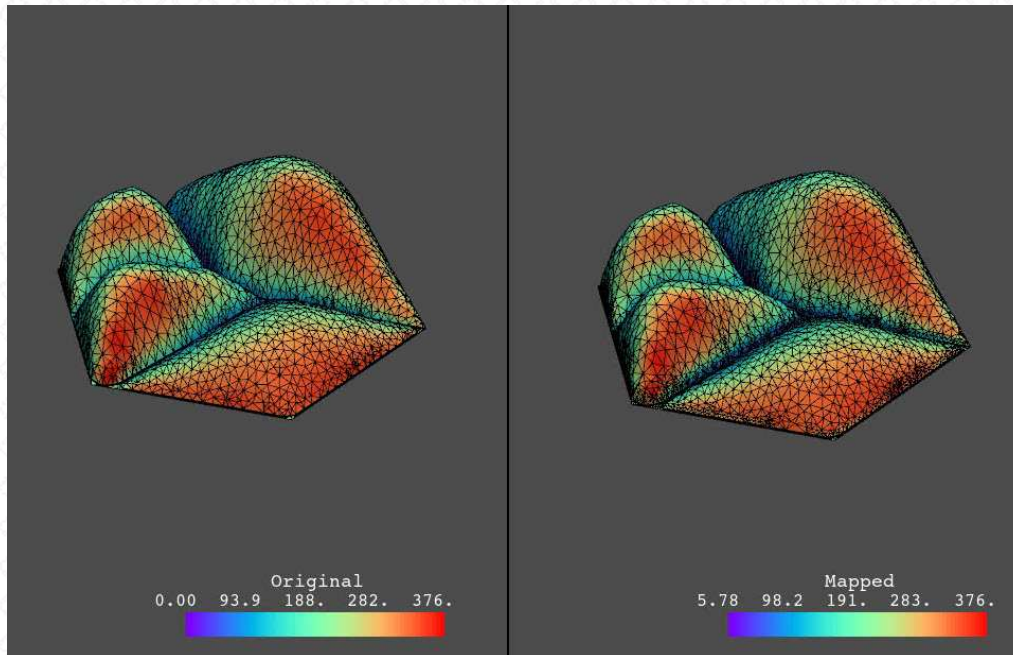
```
In [37]: Fsampled.array_names
Out[37]:
['F11',
 'F0',
 'F1',
 'F10',
 'F2',
 'F3',
 'F4',
 'F5',
 'F6',
 'F7',
 'F8',
 'F9',
 'ansys_node_num',
 'vtkOriginalPointIds',
 'origid',
 'VTKorigID',
 'ansys_elem_num',
 'ansys_real_constant',
 'ansys_material_type',
 'ansys_etype',
 'ansys_elem_type_num',
 'vtkOriginalCellIds']
```

- The 'radius' field of the interpolate method refers to the expected distance over which interpolation occurs between points. This should generally correspond to element size
- See [the documentation](#) for more on these options

Interpolate the Imported Flux Data onto the ANSYS model

- Finally, we can directly compare plots of the original incident radiation data and mapped data side-by-side by executing the following code

```
p = pv.Plotter(shape=(1,2))
p.subplot(0,0)
leg1 = {'title':'Original'}
p.add_mesh(radmesh,scalars=radmesh.point_data['F4'],cmap='rainbow',scalar_bar_args=leg1,show_edges=True)
p.subplot(0,1)
leg2 = {'title':'Mapped'}
p.add_mesh(Fsampled,scalars=Fsampled.point_data['F4'],cmap='rainbow',scalar_bar_args=leg2,show_edges=True)
p.show()
```



- You can compare different times by replacing the 'F4' (the fourth time step) above with 'F#' where # ranges from 1 through 12

Export the Interpolated Heat Fluxes to the User_Files folder

- Cut-and-paste the final bit of Python code into your session to write the heat fluxes to the Workbench user_files folder.

```
for time in range(incrad.shape[1]):  
    fname = f"mk_sfe{str(time+1)}.mac"  
    fpath = userdir + f"\\{fname}"  
    with open(fpath, 'w') as f:  
        tstr = f"F{str(time)}"  
        for i, fval in enumerate(Fsampled.cell_data[tstr]):  
            anstr = f"sfe,{str(ansmesh['ansys_elem_num'][i])},1,hflux,,{str(fval)}\n"  
            f.write(anstr)
```

- This writes the interpolated heat fluxes to a set of macros that applies these using the APDL SFE command
- Within the 12 macros, each SFE command applies the heat flux as a flux over each surface element face individually



Transfer Hourly Surface Convection Data

- Referring to the spreadsheet, we see that there is a worksheet called 'Outdoor Temp (dry bulb)'
- We'll use this to apply the surface convection terms to the same surfaces that receive incident radiation
- Although we could use the same techniques we've shown here (write macros using the imported data), it may be more appealing to some users to cut-and-paste the worksheet inputs directly into the Workbench
- Simply do some additional calculations in the Worksheet to get the data into the 3-column form needed by Workbench

7am (K)	8am (k)	9am (K)	10am (K)	11am (K)	12pm (K)	1pm (K)	2pm (K)	3pm (K)	4pm (K)	5pm (K)	6pm (K)
298.15	297.75	297.15	297.15	296.15	296.15	297.15	298.05	298.15	299.15	299.35	299.15

• This is data we're given

	time	convectio	bulk temperature
1	3600	5	25
2	7200	5	24.6
3	10800	5	24
4	14400	5	24
5	18000	5	23
6	21600	5	23
7	25200	5	24
8	28800	5	24.9
9	32400	5	25
10	36000	5	26
11	39600	5	26.2
12	43200	5	26

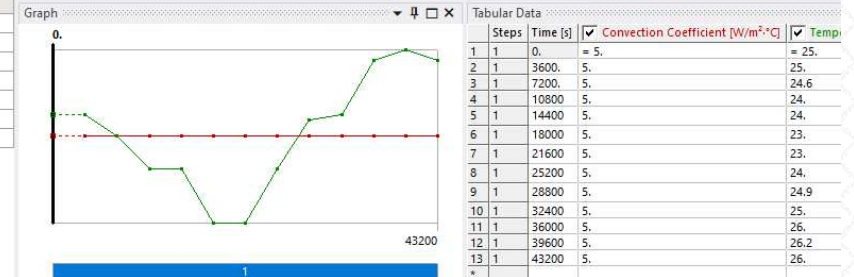
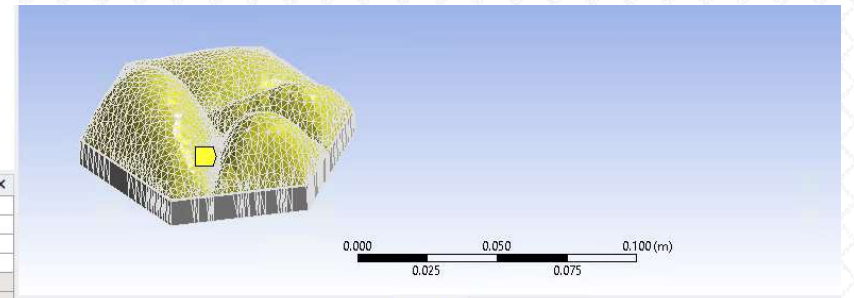
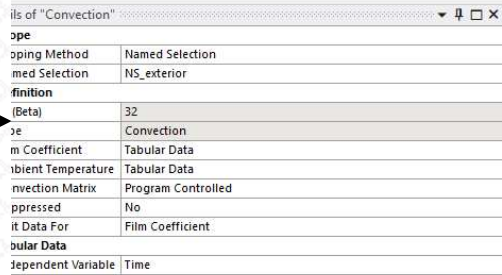
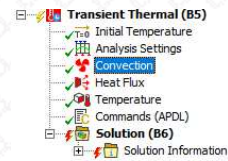
• this is the form we'll need to apply these as bulk temperatures in a convection load (let's assume a constant 5 W/m² C heat transfer coefficient)



Transfer Hourly Surface Convection Data

- After cutting-and-pasting...

	time	convection	bulk temperature
1	3600	5	25
2	7200	5	24.6
3	10800	5	24
4	14400	5	24
5	18000	5	23
6	21600	5	23
7	25200	5	24
8	28800	5	24.9
9	32400	5	25
10	36000	5	26
11	39600	5	26.2
12	43200	5	26



Tie everything together with a Command Object...

- Finally, cut-and-paste the following into your Command object window...

```
fini                !get out of solution mode
/psearch,_wb_userfiles_dir(1) !search for macros in user_files folder
/prep7              !get into pre-processor mode
esel,s,type,,2     !select elements with applied flux
nsle                !select all nodes attached to these elements
sfdelete,all,all   !delete all previously applied fluxes on selected nodes
allsel,            !re-select everything
/solu              !get back into solution mode...
*do,i,1,12         !loop through 12-hour day (defined now in seconds)...
  time,i*3600      !define time (in seconds) for this load step
  /nopr           !suppress writing to log file
  mk_sfe%i%       !apply mapped incident radiation on elements with previously applied flux
  /gopr           !un-suppress writing to log file
  solve           !solve this load step
*enddo            !complete loop
```



Solve...

- One thing to pay attention to is that we're solving this model with the command object with 12 load steps (review the code in the last slide)
- If you keep the default 'Output Controls' settings under 'Analysis Settings', ANSYS will provide you with additional output substeps. If only want 12 results corresponding to each load step, select 'Store Results At Last Time Point'

The screenshot displays the ANSYS Workbench interface during a simulation. On the left, the 'Details of "Analysis Settings"' panel is open, showing various solver and output controls. The 'Store Results At' option is highlighted in red and set to 'Last Time Point'. In the center, a 'Graph' window shows a plot of Temperature (°C) versus Time (s), with a red curve that rises and then levels off. On the right, a 3D model of a mechanical part is shown with a color-coded temperature distribution. A legend indicates a range from 22.2 Min to 22.257 Max. Below the 3D model, a 'Graph' window shows a plot of Temperature (°C) versus Time (s) with a red curve. To the right of the graph is a 'Tabular Data' window showing a table of results.

Time [s]	Minimum [°C]	Maximum [°C]	Average [°C]
1 3600	22.	23.417	22.45
2 7200	22.	23.959	22.61
3 10800	22.	24.491	22.75
4 14400	22.	24.567	22.76
5 18000	22.	24.267	22.67
6 21600	22.	24.603	22.75
7 25200	22.	24.583	22.74
8 28800	22.	24.608	22.75
9 32400	22.	24.205	22.64
10 36000	22.	23.258	22.38
11 39600	22.	22.611	22.19
12 43200	22.	22.257	22.09



Final Notes

- Importing and mapping loading and other mesh-based model inputs from other (dissimilar) models –whether within ANSYS or from without –can be very challenging. Especially when considering multiple analysis time-steps
- Fortunately, the relatively new DPF (Data-Processing Framework) come to the rescue!
- We feel that not only is this a reasonable and efficient choice for transferring model data from Rhino to Ansys, but the use of PyVista unstructured grid object allows for very efficient data mapping over multiple time steps
- For the model in this example, the actual transfer occurs in seconds. And this scales well with model size
- Without DPF, this last capability is challenging even within ANSYS Workbench
- In this article, we provide all the files and code used in the example for the ANSYS-DPF workflow
- We also provide all the jpeg images (some of which are described in the Appendix) that Nathan created for the alternative workflow



Appendix

An Alternative Workflow

- Some users (especially Rhino users) may not feel comfortable installing and learning the ANSYS DPF Python tools
- For those users, Nathan suggests a workflow similar to the one we outlined, but replacing the ANSYS DPF with programming tools available to Rhino users
- We compare the steps in each below

ANSYS DPF Workflow

1. Import heat flux data in Python (slide 7)
2. Create the Ansys model (slide 11)
3. Import and interpolate heat flux (slide 17)
4. Export heat fluxes and other loads to ANSYS (slide 23)
5. Incorporate imported loads and run with command snippet (slide 25)

Alternative Workflow

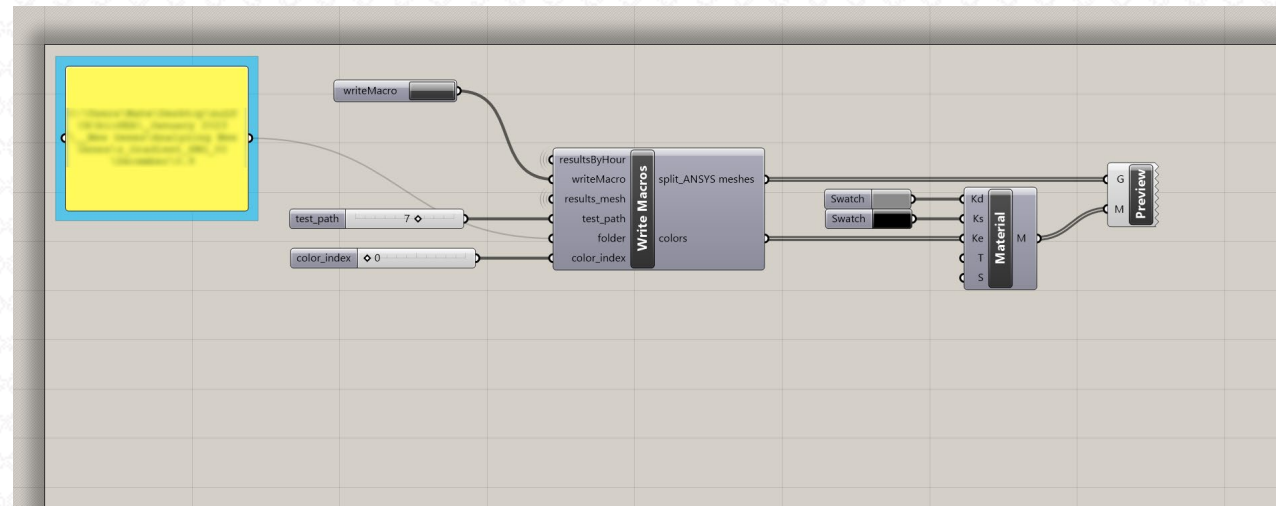
1. Export Grasshopper model without radiation
2. Create the ANSYS model (slide 11)
3. Import ANSYS surface mesh into Grasshopper
4. Generate and export heat fluxes and other loads to ANSYS
5. Incorporate imported loads and run with command snippet (slide 25)



Appendix

An Alternative Workflow

- There are some key differences in the two workflows
- In the ANSYS-DPF workflow, we start by importing a mesh-based model created in Ladybug into ANSYS (it always creates its own mesh. And that's what we originally imported in this example)
- In the alternative workflow, we export the Grasshopper-generated data directly and read that into ANSYS (bypassing Ladybug at first. This is step 1)
- We then create the placeholder ANSYS model the same way we did in slides 11 thru 16 (step 2) and export. Experienced programmers may simply parse the ensuing ds.dat file, but we've supplied another macro called 'mk_exportmesh.mac' to export the surface effect elements (it can be invoked as a command object or from a separate MAPDL session)
- Then, we we import the ANSYS mesh into Grasshopper (step 3)



Appendix

An Alternative Workflow

- Nathan provides the following helpful c# to import the ANSYS model (starting with the text files crated by 'mk_exportmesh.mac')

```
private void RunScript(string extfaces_path, string nodes_path, ref object A, ref object B)
{
    string[] faceLines = System.IO.File.ReadAllLines(extfaces_path);
    string[] nodeLines = System.IO.File.ReadAllLines(nodes_path);

    var pts = new List<Point3d>();
    for (int i = 0; i < nodeLines.Length; i++)
    {
        var strPt = nodeLines[i].Split(',');
        if (strPt.Length == 3)
        {
            double X = 1000 * double.Parse(strPt[0]);
            double Y = 1000 * double.Parse(strPt[1]);
            double Z = 1000 * double.Parse(strPt[2]);
            pts.Add(new Point3d(X, Y, Z));
        }
    }

    Mesh mesh = new Mesh();
    var elementNumbers = new List<int>();

    for (int i = 0; i < pts.Count; i++)
    {
        mesh.Vertices.Add(pts[i]);
    }

    for (int i = 0; i < faceLines.Length; i++)
    {
        var strFace = faceLines[i].Split(',');
        if (strFace.Length == 4)
        {
            mesh.Faces.AddFace(Int32.Parse(strFace[1]) - 1, Int32.Parse(strFace[2]) - 1, Int32.Parse(strFace[3]) - 1);
            elementNumbers.Add(Int32.Parse(strFace[0]));
        }
    }

    var strPt1 = nodeLines[0].Split(',');

    A = mesh;
    B = elementNumbers;
}
```



Appendix

An Alternative Workflow

- Nathan also provides the following helpful c# to export the heat flux data from Ladybug:

```
private void RunScript(bool writeMacro, string filepath, DataTree<double> results, List<int> eNumber, List<int> faceIndex, ref object A)
{
    DataTree<string> irMacro = new DataTree<string>();

    for (int i = 0; i < results.BranchCount; i++)
    {
        GH_Path p = new GH_Path(i);

        for (int j = 0; j < eNumber.Count; j++)
        {
            string nextLine = "sfe," + eNumber[j].ToString() + ",1,hflux,," + results.Branch(i)[faceIndex[j]].ToString();
            irMacro.Add(nextLine, p);
        }
    }

    if (writeMacro)
    {
        for (int i = 0; i < results.BranchCount; i++)
        {
            int step = i + 1;
            string fullPath = filepath + @"\mk_sfe" + step.ToString() + ".mac";

            File.WriteAllLines(fullPath, irMacro.Branch(i));
        }
    }
}
```

