



Energy and Buildings

Design of an efficient and resilient building
considering global warming with **baterm**

author:

stephane.ploix@grenoble-inp.fr

Installing the `batem` application

First, check if your computer understand Python from the command line interface (CLI) by running `

- `PowerShell` for Windows
- `Terminal` for MacOS (it also runs on Linux):

Then `python --version`.

You should get 3.12 minimum.

Once this check done, you can launch the installation of the regular pypi module with

```
python -m pip install -U batem
```

The full sources at https://wwiw.gricad-gitlab.univ-grenoble-alpes.fr/batem/batem_all but you don't need the source code to follow the tutorials

For Windows users, if the commands `python3.exe` or `pip3.exe` are not found, it's likely because these files are not in the `PATH` environment variable.

You can add them to the `PATH` by:

1. Opening the System Properties
2. Clicking on the "Environment Variables" button
3. Adding the path of the Python installation directory (e.g., `C:\Python313`) to the "Path" variable
4. Clicking "OK" to save the changes
5. Open a new terminal and check if the commands are found.

This lecture is based on BATEM, but what is BATEM?

BATEM is a software framework dedicated to the teaching of many aspects of energy management related to buildings. It can also be used for developing researches and design of performant buildings or for learning and teaching purposes. It is a Python application (but you don't need to know Python to follow).

`batem` includes a command line interface to use it is named `baterm`

To launch `baterm`, type in the CLI: `python -m batem.baterm`

Note: if you want to update `batem` to the latest version, type again in the CLI:

```
python -m pip install -U batem
```

What is **baterm** ?

- a way to use batem
 - access to the lectures with `lectures` or `lecture <number>`
 - access to the workshops with `workshops` or `workshops <number>`
- a **command line tool** to design **efficient building systems** with a fast learning curve.
 - analyze a **context** i.e. what is around **a building and/or a PV plant**?
 - simulate the **building performance**
 - simulate a **PV plant**
 - analyze the joint performance: **building + PV plant**

batem or baterm ?

- `baterm` is part of `batem` .
- `batem` is a Python module containing:
 - tutorials, named `slidesXX-short_name.pdf`
 - workshops, named `workshopsXX-short_name.ipynb`
- Each tutorial has a corresponding workshop (a Jupyter Notebook) to learn by practice.
- `batem` is also a tool for research to experiment new ideas

part II - Intuitive approach of building design

Intuitive building design

Design of an efficient and resilient building considering global warming with **baterm**

author: stephane.ploix@grenoble-inp.fr

You are going to use **baterm** to discover the building design

baterm is a tool to easily design buildings but it is based on the precise simulation model of **batem**

- heat transfer by conduction, convection and radiation
- solar gains through windows, walls and PV panels considering solar masks
- close (building itself, surrounding buildings) and far (mountains) shadings
- air infiltration and forced ventilation
- HVAC system for cooling and heating
- types of roofs
- thermal inertia
- cell connections in photovoltaic panels and impact of temperature
- mounting type of PV panels

Understanding what matters in building design

Building efficiency is not just a matter of insulation

- location, shape and direction of the building (including roof)
- adjacency to other buildings
- location and glazing surfaces
- materials used
- tuning of the HVAC system
- occupancy and comfort demands
- PV plant and autonomy
- ...

Thanks to `baterm`, it's easy to understand what matters in building design

The basic commands of **baterm**

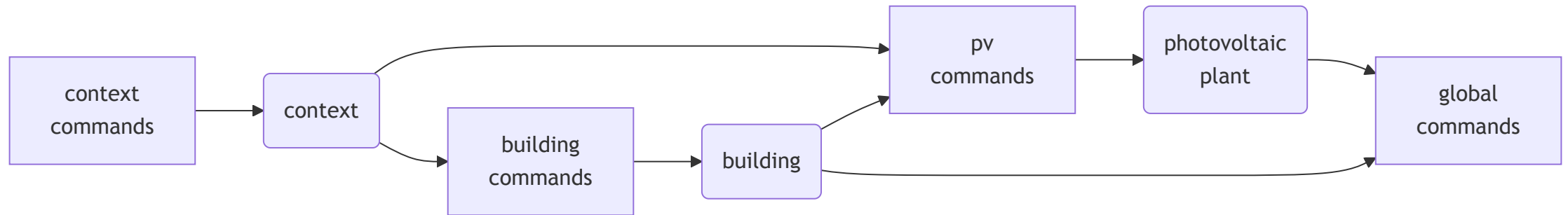
command	description
help	show help for a command (<code>help alias</code> , for instance)
alias	list all aliases (replace a string by another)
dir	list all directories in the working folder
cd	change the working directory
pwd	print the current working directory
history	show the history
manual	show the manual
quit or exit	quit the program

command	description
shell	run a shell command
macro	list all macros (batem scripts)
→ macro create	create or overwrite a new macro
→ macro delete	delete a macro
→ macro list	list all macros
load	load a macro (batem script)
run	run a macro (batem script)
shortcuts	list available shortcuts
ls	list all files
pyrun or run_pyscript	run a Python script

The batem commands

command	description
lectures	list all lectures
→ lectures 15	show the lecture 15
workshops	list all workshops
→ workshops 15	show the workshop 15

Commands are structured around 3 objects: context, building and photovoltaic plant.



A **context** is a location with a weather and possible solar **side masks** (modeling for instance, surrounding buildings).

The context commands

A context is mainly defined by a location:

- a latitude North and a longitude East in decimal degrees.

It is used to load the historical weather data from <http://open-meteo.com>, to get:

- the **elevation** of the location
- the far masks due to **surrounding mountains**.

A context can be created with the command `context new`.

Default values are proposed. Then it can be modified with `context edit`.

Note the tip on the right hand side of each value: it is a documentation for the value.

The name appearing in the location field is used to save the collected data to avoid re-downloading them each time.

The context data are shown below: the historical weather data (from January 1st, 1980 till nowadays) will be saved in `Grenoble.json`.

latitude_north_deg	45.1916	?
longitude_east_deg	5.7174	?
location	Grenoble	?
albedo	0.1	?
pollution	0.1	?
side_masks		?
simulated_year	2022	?
year_average_temperature	14.83 °C (used as TZ:ground)	?
Cancel OK		

Albedo is the average reflectivity of the surface: 0 means no reflection, 1 means full reflection. It is used to compute the solar radiation on the surface. (default value is 0.1)

The **pollution coefficient** is used to compute the solar radiation on surfaces.

The pollution level is a value between 0 and 1, 0 means no pollution, 1 means maximum pollution. (default value is 0.1)

Sidemasks are used to model the surrounding buildings. They are defined by a list of rectangular surfaces with their dimensions and angles.

The simulated year is the year of the historical weather data to be used. It should be between 1980 and the year before the current year because full year data is needed for the simulation.

The year average temperature is the average temperature of the year. It is used to compute the ground temperature but it can be changed if needed.

A documentation for the side-masks can be found in the file [doc_sidemasks](#).

Here is an example of a side-mask (several side-masks can be defined ; they just have to be separated by a comma):

```
{  "name": "left",
    "x_center": 2.5,
    "y_center": -20.0,
    "width": 15.0,
    "height": 6.0,
    "elevation": 0.0,
    "exposure_deg": 180.0,
    "slope_deg": 90.0,
    "normal_angle_with_south_deg": 0.0,
    "normal_rotation_angle_deg": null,
    "tangent_x": NaN,
    "tangent_y": NaN,
    "facade_normal_x": NaN,
    "facade_normal_y": NaN
}
```

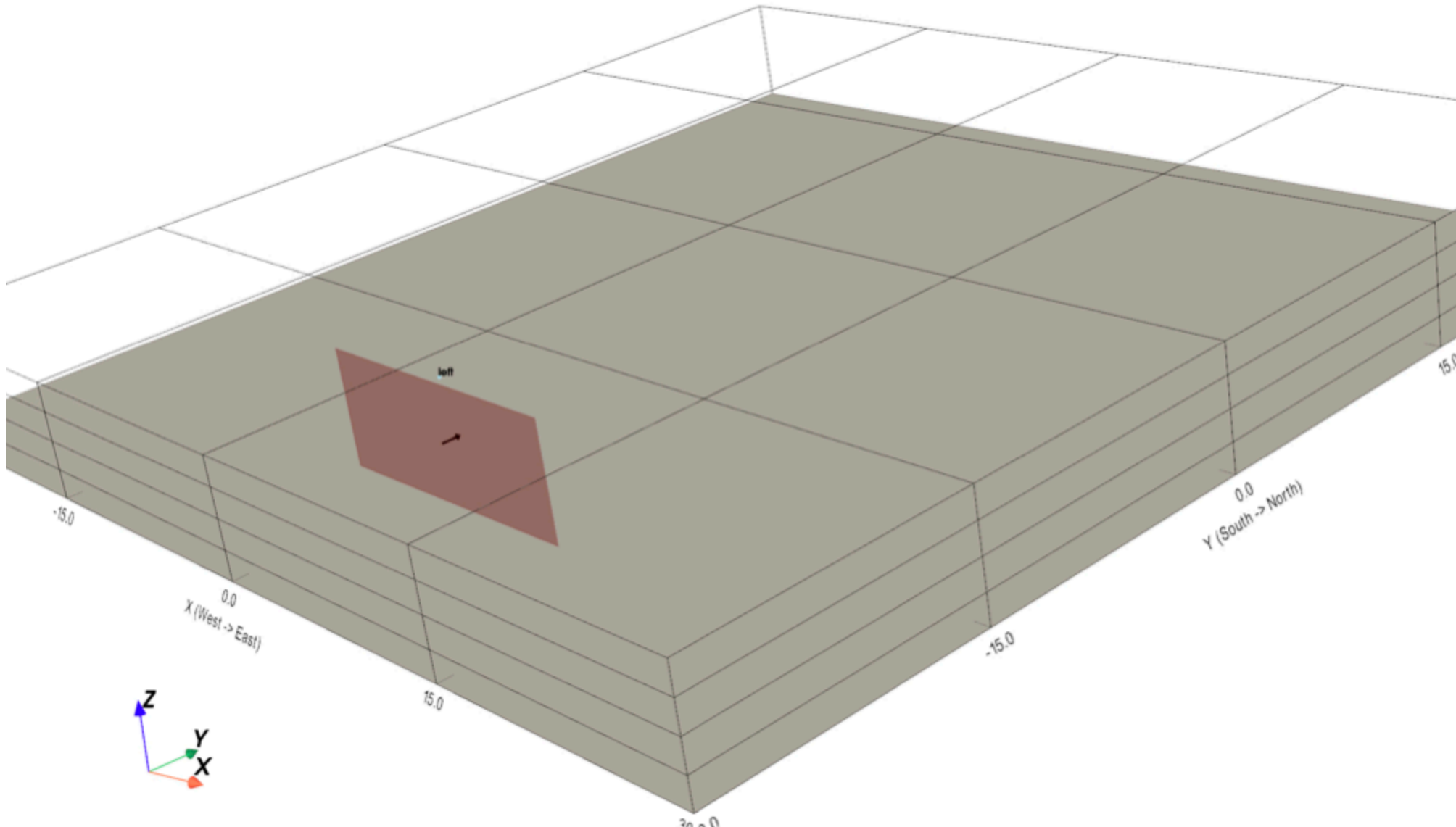
Here, side-masks are used to model vertical rectangular surfaces (slope = 90°) touching the ground (elevation = 0).

Only the lower middle position (at the center of the edge cutting the ground) has to be defined (x_center , y_center) and the dimension of the side (width, height). Finally the direction of the normal has to be defined: $exposure_deg=0$ (or $\pm 180^\circ$ or 360°) means South or North, $exposure_deg= \pm 90^\circ$ means East or West.

A side-mask for a wall requires only 5 values to be defined:

```
{ "name": "mask_name",  
  "x_center": ?,  
  "y_center": ?,  
  "width": ?,  
  "height": ?,  
  "elevation": 0.0,  
  "exposure_deg": ?,  
  "slope_deg": 90.0,  
}
```

You can see the result with the command `context 3d :`



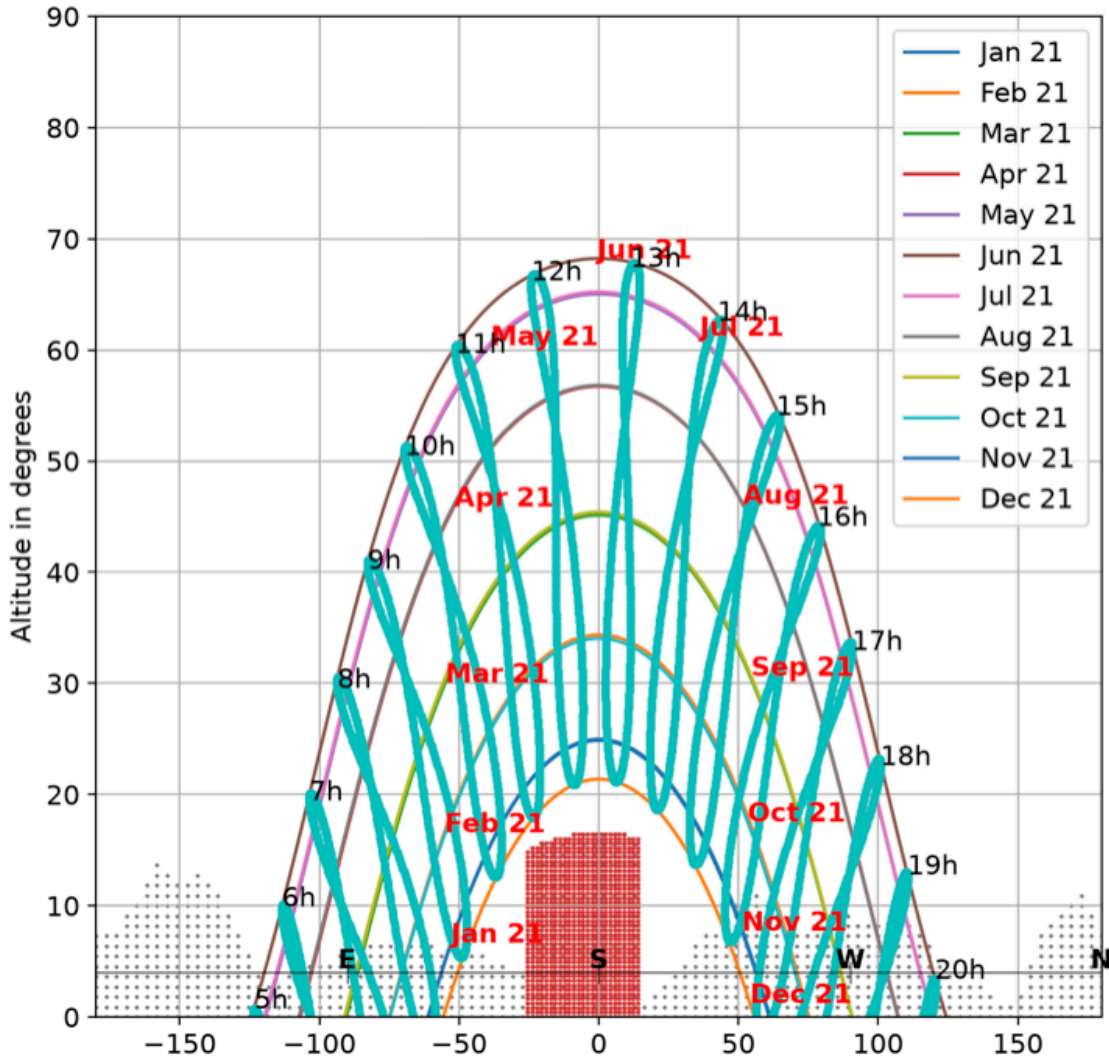
The heliodon, representing the sun path each 21st of months of the year, is shown with the command `context heliodon` knowing that the observer is located at the (0,0,0) position i.e. at the specified gps coordinates.

The red shape is the defined side-mask (sun is hidden behind it when it passes at this position).

The gray mask stands for the surrounding mountains.

As we can see, this mask is never hiding the sun.

Here is an example of heliodon for the context:



command	description
context	show the current context
context delete	delete a context
context new	create a new context
context edit	edit a context
context heliodon	show the heliodon
context 3d	show the 3d view of the context
context sidemasks	show the features of the defined side masks

Many other tools are available to analyze the context:

command	description
context rain	graphical tool to analyze the rain data
context temperature	graphical tool to analyze the temperature data
context wind	graphical tool to analyze the wind data
context solar	graphical tool to analyze the solar data
context psychro	psychrometric chart to analyze how the weather is comfortable
context evolution	graphical tool to analyze the evolution of the outdoor temperature since 1980

command	description
context vars	list all variables available in the context
context plot	plot any variables of the context

The building commands

A building is defined by its geometry (footprint, number of floors, heights), its envelope (wall / roof / floor / glazing compositions), its HVAC settings (setpoints, heating and cooling powers, air renewal) and its occupancy.

It needs a context first: the weather and the solar masks come from it.

A building can be created with `building new`. Default values are proposed. Then it can be modified with `building edit`.

As for the context, each field has a tip on the right-hand side.

The building is saved as a `.building` JSON file under `./data` (for instance `building save myhouse` → `./data/myhouse.building`).

The footprint is a list of 2D points in meters: `[0,0],[10,0],[10,5],[0,5]`. Axes: **+X East, +Y North, +Z up**. It can be visualized with the command `building footprint`.

Side indices start at 0 and follow the footprint edges. Use `building footprint` to check them. Sides can be visualized with the command `building footprint` too.

Glazing ratios can be one value same for all façades, or one value per side. It represents the fraction of the side that is glazed.

Compositions are stacks of layers **from indoor to outdoor**, for example:

```
["plaster",0.013],["insulation",0.15],["brick",0.1]
```

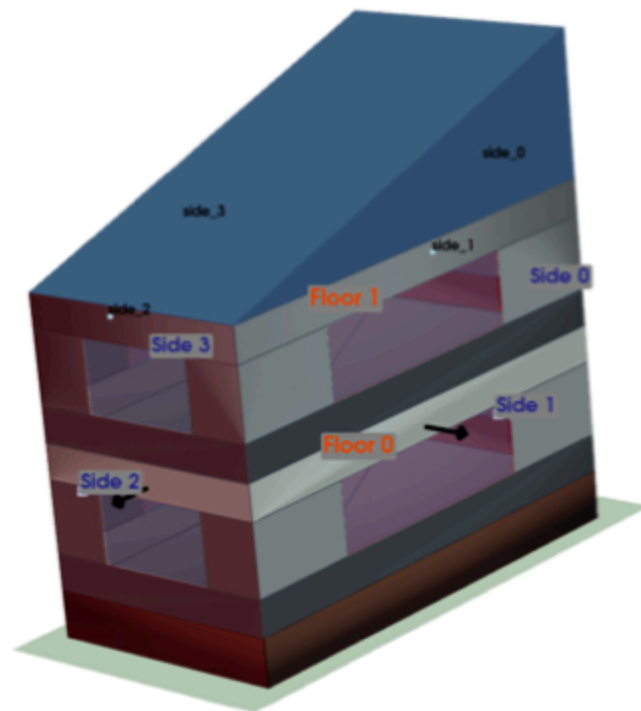
Material names come from `propertiesDB.xlsx` (see `material list thermal`) but only the ones with a star in front are loaded. If you load another one, use the command `material load <a_name> <row_number>`.

Roofs

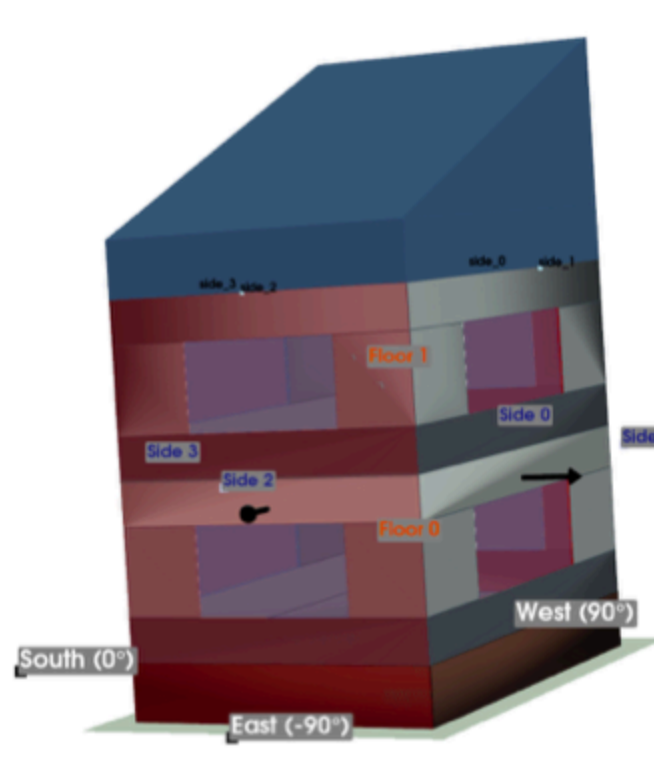
A roof can be extruded along a given height: `roof_extrusion: 1.0`. There will be an additional 1 m strip in roof material in the continuity of the walls.

It can have a wall ratio: `roof_wall_ratio: 0.4`. It represents the fraction of the side that is considered as a wall. The rest is considered as a roof.

Moreover, if `attic` is set to `True`, a passive attic zone (no HVAC, glazing, or occupants) is defined: it constitutes a buffer zone between the last inhabited floor and the attic: the last floor composition is `lowup_floor_composition`.



attic: False
extrusion: 0
wall_ratio: 0.0



attic: True (not visible on the 3D plot)
extrusion: 1
wall_ratio: 0.4

In the building form, 2 kinds of roof shapes can be defined: the aperiodic roofs (periodic=False) and the periodic roofs (periodic=True). Defaults keep a flat ceiling:

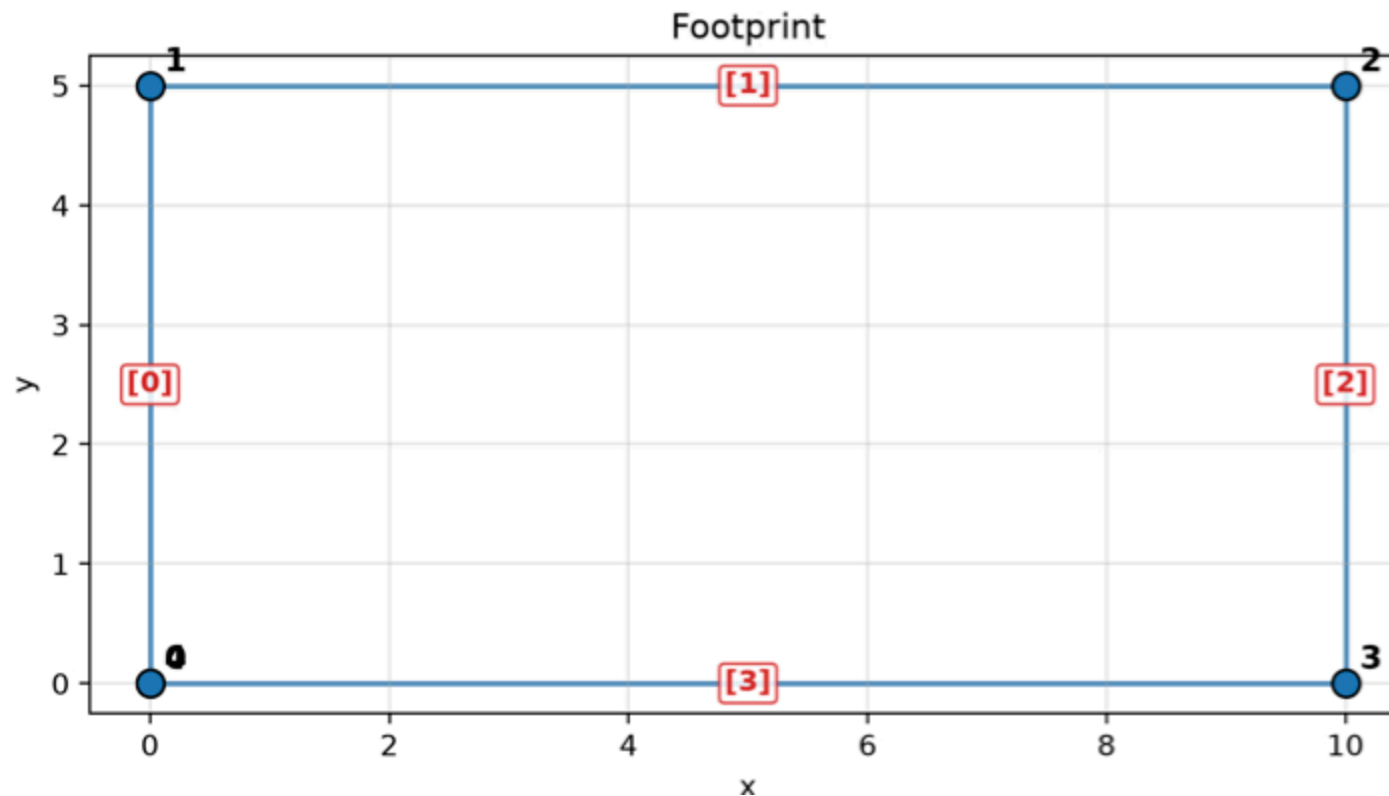
`"roof_points": []` and do not depend on the kind of roof shape.

Aperiodic roof is defined by:

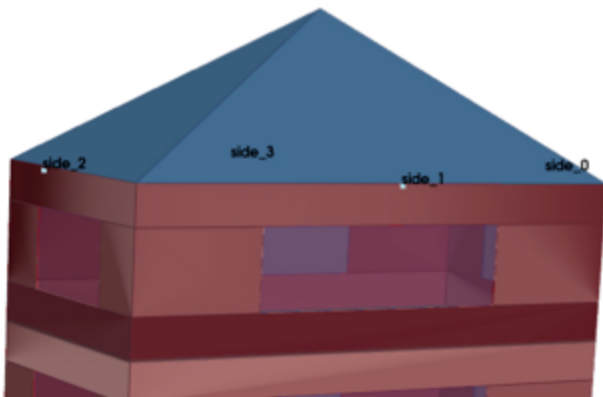
- `roof_points` — ridge / apex `(x, y, z)` in meters (same axes as the footprint in meters) define a convex hull on the roof surface where `z=0` stands for the top of the building without roof but keeping the footprint.
- `roof_extrusion` — height of the vertical roof base above the top storey [m]
- `roof_wall_ratio` — lower fraction of each vertical roof side counted as **wall** (0..1); the rest is **roof**
- `attic` — if true, a passive attic zone (no HVAC, glazing, or occupants)

Check the shape with `building 3d` .

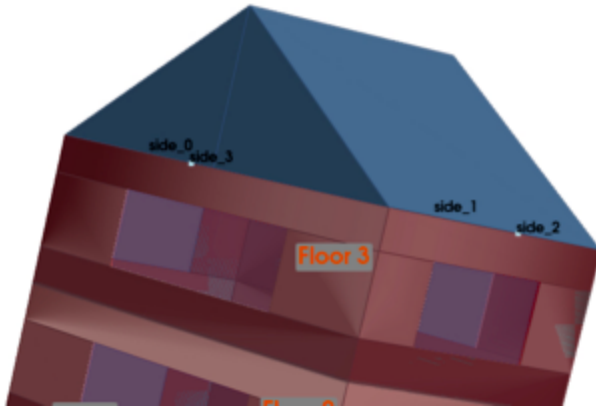
For example, the following roofs are defined for a building whose footprint is `[0,0,0]` , `[10,0,0]` , `[10,5,0]` , `[0,5,0]` .



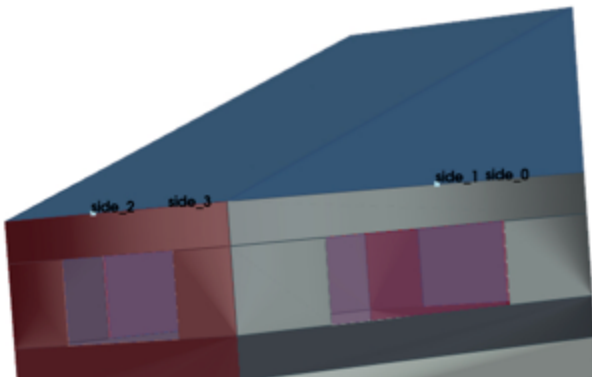
A pointed roof can be defined by one point: `[5, 2.5, 3]` .



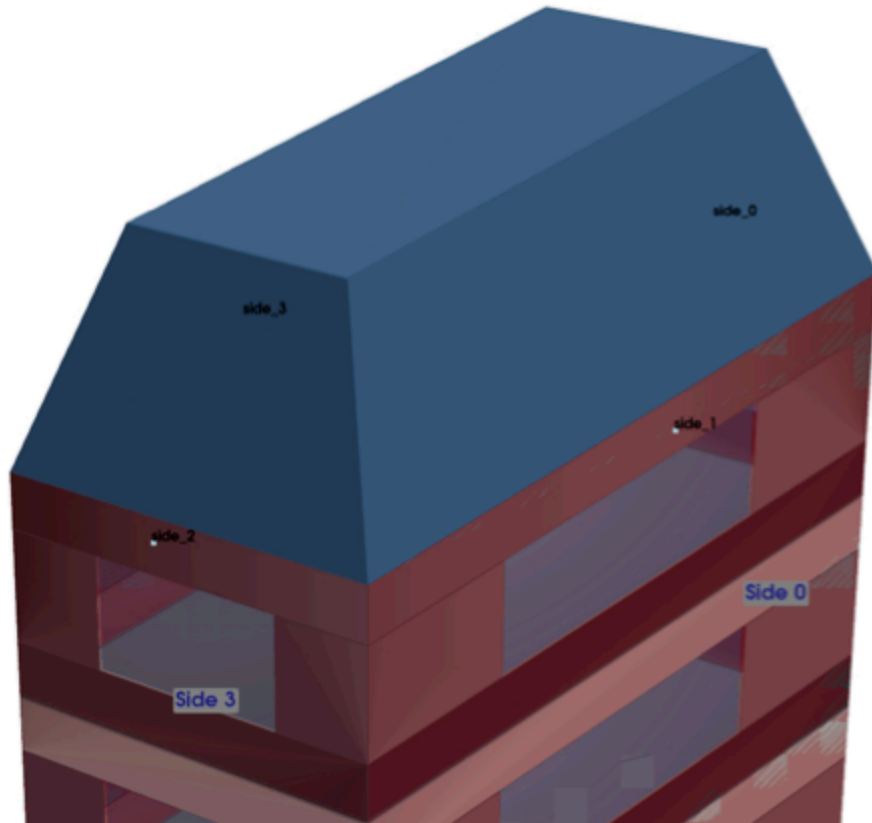
A hip roof can be defined by two points: $[5, 0, 3]$, $[5, 5, 3]$.



Another hip roof: $[0, 0, 3]$, $[0, 5, 3]$:



A ridge roof can be defined by 4 points: $[1, 1, 3]$, $[9, 1, 3]$, $[9, 4, 3]$, $[1, 4, 3]$.



Periodic roof shapes

Set `periodic: True` to build a repeating height profile instead of a convex hull.

In this mode, `roof_points` are **not** `(x, y, z)` apexes. They are breakpoints of **one period**:

```
(fraction, z)
```

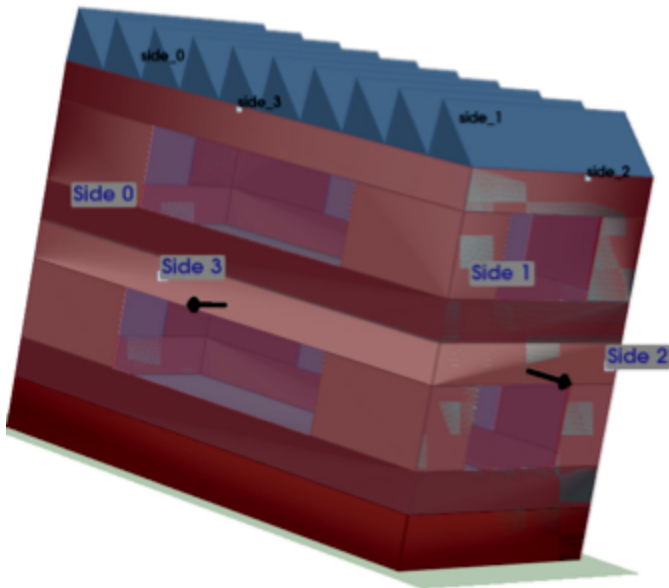
- `fraction` — position along one period (`0` = start, `1` = end of the period)
- `z` — roof height above the top of the extrusion [m]

That profile is repeated `periodic_n_repeats` times along `periodic_direction` (`"x"`, `"y"`, or a 2D vector `[dx, dy]`).

`roof_extrusion`, `roof_wall_ratio` and `attic` work as for aperiodic roofs.

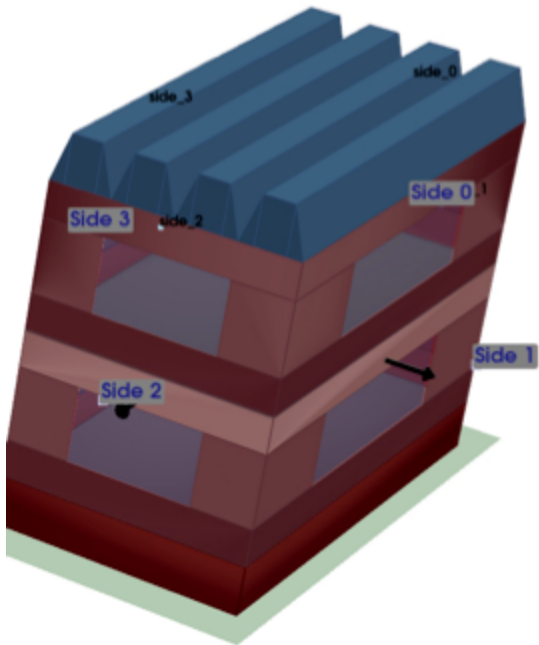
Saw-tooth / shed periods (10 repeats along `x`):

```
"periodic": true,  
"periodic_n_repeats": 10,  
"periodic_direction": "x",  
"roof_points": [[0, 0], [0.1, 1.0], [1, 0]],  
"roof_extrusion": 0
```



Trapezoid periods (4 repeats):

```
"periodic": true,  
"periodic_n_repeats": 4,  
"periodic_direction": "y",  
"roof_points": [[0, 0], [0.25, 1], [0.75, 1], [1, 0]]
```



Check with `building 3d`. Prefer this mode for industrial saw roofs or any shape that repeats along one axis.

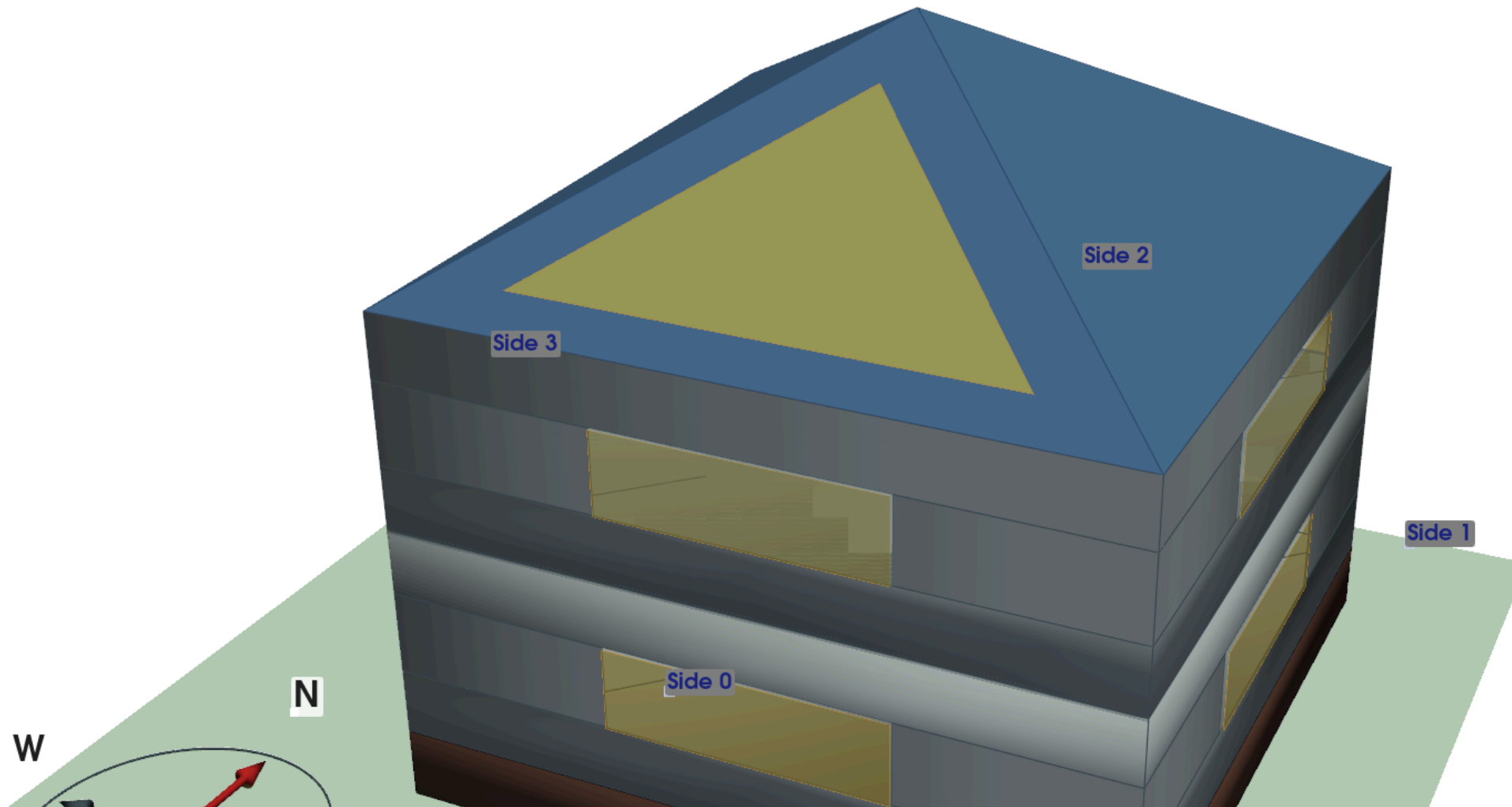
Roof windows

Optional glazing on a **non-periodic** pitched roof (`periodic: false` , with `roof_points`).

Enter a full JSON list in the building form (outer `[]` included), or use the  editor:

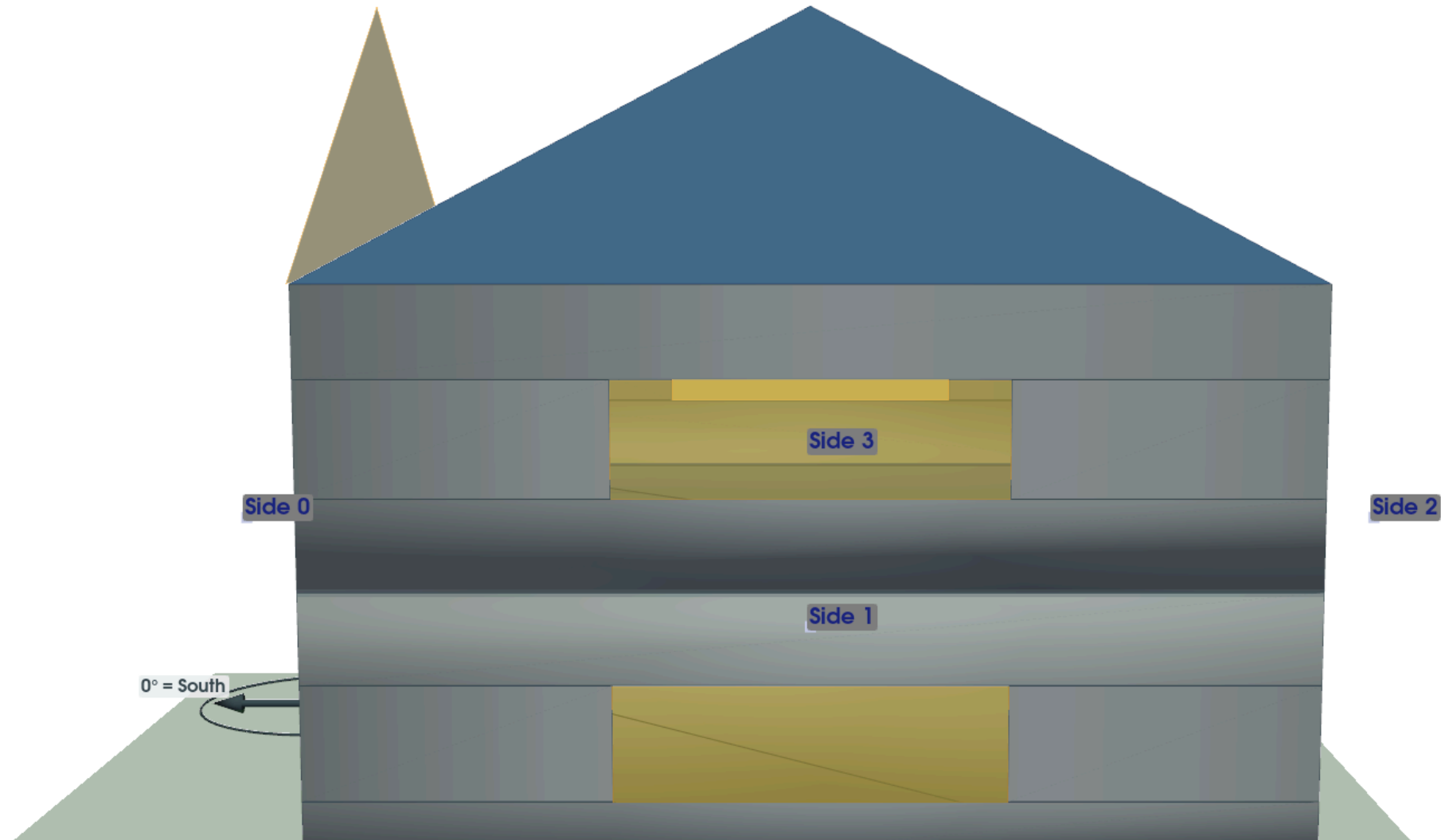
```
"roof_windows": [
  {
    "facet": ["$0", "$1", "$4"],
    "glazing_ratio": 0.5,
    "aligned": true,
    "solar_factor": 0.5
  }
]
```

- `facet` — coplanar points on **roof mesh nodes**, or shortcuts `"$N"` (same indices as `building footprint: $0...$n-1 = footprint (X,Y,0)` , then `roof_points`)

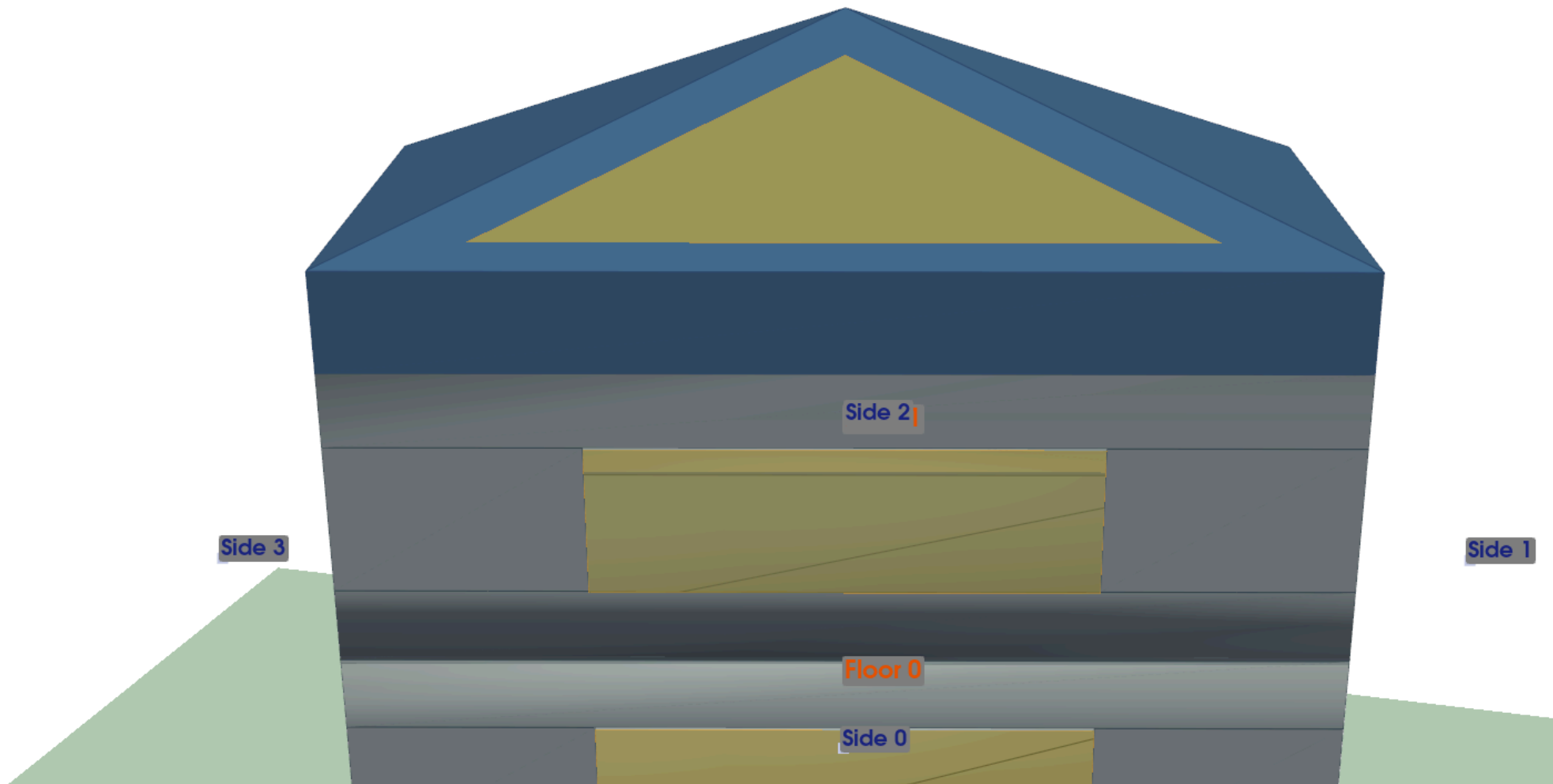


`aligned: false` — **vertical** glazing (slope 90°) on the lowest facet edge. Opaque roof unchanged; solar aperture = `glazing_ratio` × **vertical projection** of the facet (sill × rise).

In the next slide, one can see that the vertical windows are coming out of the roof by are not fully connected to the roof. Vertical windows are not changing the surface of the roof whereas the aligned windows are replacing the covered surface by the glass.



Same idea on a hip roof (`aligned: true` on one facet):



Optional features:

- attic: set `attic: true` in the roof fields; needs the intermediate floor composition `lowup_floor`
- basement: when `base_elevation > 0`, a `basement_ground` composition is required
- wall contacts: neighbouring heated volumes on a façade (force glazing to 0 on the contacting floors)
- solar protections: `max_unshaded_protections` per side; 90° means no solar protection on that axis and 45° means the sun is hidden when it passes this altitude
- roof windows: `roof_windows` list on non-periodic roofs (see above)

Typical design loop:

```
context load project
building new
building 3d
building simulate --quick
building results
building edit
building simulate
building save project
```

command	description
building	show a summary of the current building
building new	create a new building (GUI)
building edit	edit the current building (GUI)
building load <path>	load a <code>.building</code> file from <code>./data</code>
building save <path>	save the building to <code>./data</code>
building delete	forget the building in memory (or delete a file)

command	description
building footprint	plan view with side indices
building 3d	3D view of the building
building heliodons	sun paths seen from central glazing of building façades
building simulate	hourly thermal simulation over the year specified in the context
building results	HVAC totals + comfort indicators
building vars	list building-related variables (+ <code>PV:power_W</code> after <code>pv simulate</code>)
building plot	plot building variables (+ <code>PV:power_W</code> after <code>pv simulate</code>)
building optimize	comfort-first tune of fields marked Tune in the building form

Parametric design is done with `building variation ...`.

It sweeps one design parameter (or a full set) and opens figures with:

- left: HVAC energy (kWh/year)
- right: comfort percentiles on a reference floor

`glazing_ratio` sweeps include **all façades** (even currently opaque ones).

Examples:

```
building variation
building variation heat_recovery
building variation glazing_ratio side 0
building variation solar_protection side 0
```

Scalars include heat recovery, air renewal, solar factor, setpoints, rotation, ...

Figures are also saved under `.baterm/variation_figures/`.

Full syntax: `help building variation`

command	description
building variation	full parametric sweep
building variation <param>	sweep one parameter
building variation ...	limit to a few cases (fast)
building optimize	comfort-weighted optimize of Tune fields
material list thermal	browse thermal materials
material list glazing	browse glazing materials
material thermal load <name> <row>	load a material into the session

The pv (for photovoltaic) commands

A photovoltaic plant is defined by its orientation, tilt, mounting type, module efficiency and sizing.

It also needs a **context** (site + weather). If no context is loaded, `pv new` opens the context editor first.

A plant can be created with `pv new`, then modified with `pv edit`.

It is saved as a `.pv` JSON file under `./data` (for instance `pv save roof` → `./data/roof.pv`).

The plant embeds a copy of the context used at creation time. When you already have a session context, baterra keeps that period so building and PV stay on the same horizon.

Orientation conventions:

- `exposure_deg` : **0 = South**, +90 = West, -90 = East, 180 = North
- `slope_deg` : **0 = facing the ground**, 90 = vertical, 180 = facing the sky
(a classic 35° tilt facing the sky corresponds to `slope_deg = 145`)

Sizing: give **exactly one** of:

- `number_of_panels`
- `peak_power_kw`
- `ground_surface_m2`

Mounting type (`flat` , `plan` , `back2back`) changes self-shading between rows. See `slides01-sun.pdf` for more details.

Typical PV loop:

```
context load project
pv new                # Tune selected fields
pv 3d
pv best angles
pv optimize           # max yield (no building PELEC yet)
building simulate
pv optimize           # then match daily PV to daily PELEC
pv simulate
building plot         # PELEC / HVAC + PV:power_W
balance
pv save project
```

`pv best angles` searches exposure/slope for maximum annual DC production of the current plant.

`pv optimize` has two modes:

- **no building PELEC yet** — maximize annual PV kWh (exposure, tilt, spacing, mount)

command	description
pv	show a summary of the current PV plant
pv new	create a new plant (GUI; needs context)
pv edit	edit the current plant (GUI; Lock/Tune on design fields)
pv load <path>	load a <code>.pv</code> file from <code>./data</code>
pv save <path>	save the plant to <code>./data</code>
pv delete	forget the plant in memory (or delete a file)
pv dir	list <code>.pv</code> files in <code>./data</code>

command	description
pv 3d	3D layout of the arrays
pv heliodon	sun path for the plant site
pv best angles	best exposure / slope for annual yield
pv simulate [--quick]	hourly PV production (building sim optional)
pv optimize [--passes K]	max yield, or match daily PELEC after building sim
pv results	annual totals + building–PV indicators
pv vars	list PV-related variables
pv plot	plot PV variables

After both `building simulate` and `pv simulate`, `pv results` prints indicators such as:

- self-consumption
- self-sufficiency (self-production)
- dependency
- year autonomy
- NEEG (net energy exchange with the grid)
- autonomous day ratio

These indicators need the building electricity demand `PELEC:building#sim` and the PV production `PV:power_W`.

The global commands

Global commands join the **building** and the **PV plant** on the same DataProvider.

They answer questions such as:

- when is production larger than demand?
- what is the annual electricity bill with a given tariff?
- how to export all series for further analysis?

```
flowchart LR
    B[building simulate] --> D[DataProvider]
    P[pv simulate] --> D
    D --> E[enercost]
    D --> F[balance]
    D --> X[export]
```

balance

`balance` draws a **day × month heatmap** of the daily net balance:

balance = PV production – building electricity demand

- **green**: surplus (PV covers the day)
- **red**: deficit (grid import needed)

Needs both simulations first:

```
building simulate  
pv simulate  
balance
```

enercost

`enercost` computes the annual electricity cost from:

- `PELEC:building#sim` (building demand)
- `PV:power_W` (PV production)

Default tariff is French **HP/HC** (off-peak roughly 22:00–06:00).

It writes series such as `ENERCOST:hour#sim`, `GRID_IMPORT:kWh#sim`,
`GRID_EXPORT:kWh#sim`.

```
enercost
enercost hp_hc
enercost base --base 0.25 --export 0.12
```

Typical joined workflow:

```
context load mysite  
building load myhouse  
building simulate  
pv load roof  
pv simulate  
pv results  
balance  
enercost  
export my_study
```

`export <name>` writes all DataProvider series to `./<name>.xlsx` in the working directory.

command	description
balance	day×month heatmap PV – PELEC
enercost	annual electricity bill (default HP/HC)
enercost hp_hc	French peak / off-peak tariff
enercost base --base ... --export ...	flat buy / sell prices (€/kWh)
export <name>	export all series to Excel
status	show loaded context / building / PV
clear	forget context, building and PV in memory
vars	list variables (session DataProvider)
plot	plot any session variables

Wrap-up

Remember the dependency chain:

1. **context** (site + weather + masks)
2. **building** and/or **pV** (both need the context)
3. **simulate** each side
4. **global** tools: `balance`, `enercost`, `export`

Use `help <command>` at any time, and `manual` to open the PDF user manual.

