Fast design of building systems with baterm

author: stephane.ploix@grenoble-inp.fr

reviewer: mircea_stefan.simoiu@upb.ro

What is `baterm`?

`baterm` is a command line tool to design building systems with a focus on energy performance and comfort.

It has been thought for having a very fast learning curve.

It contains a set of tools to:

- describe and analyze a context i.e. what is around a building or a pv plant
- describe and simulate the energy performance of a building system
- describe and simulate the production of a pv plant
- analyze the joined performance of a building and a pv plant

It makes fast design of building systems possible although it is based on a precise simulation model that takes for instance thermal inertia, heat transfer by conduction, convection and radiation, gain through each window by solar radiation, close and far shadings due to building itself or photovoltaic panels themselves, weather effects on building and photovoltaic panels, air infiltration and forced ventilation, adjacent buildings and their effects on the building, HVAC system for cooling and heating, the cell connections in photovoltaic panels and the effect of shading on the production, ... into account.

batem or baterm ?

baterm is part of batem.

batem is a Python module containing a pedagogical part composed of:

- lectures, named slidesXX-short_name.pdf
- workshops, named workshopsXX-short_name.ipynb

Each lecture, except for the introduction, has a corresponding workshop (a Jupyter Notebook) to practice the concepts.

batem is also a tool for research and development to experiment new ideas and to validate them.

The lectures/workshops are organized in the following way:

- 00: introduction (no workshop, 4h)
- 01: modeling the Sun - Earth system and their effects (4h)
- 02: modeling climate changes and global warming (4h)
- 03: modeling human comfort and energy impacts (4h)
- 04: building physics (4h)
- 05: modeling inertia and dynamical phenomena (4h)
- 06: HVAC system for cooling and heating (4h)
- 07: fitting models of existing buildings (4h)
- 08: managing buildings to improve their energy performance (4h)
- 09: modeling photovoltaic plants (4h)

- 10: simulating energy communities (4h)
- 11: graph theory to model ski-resort systems (4h)
- 12: using multi-agent systems to model skiers (4h)
- 13: strategy for good compromises between comfort and energy performance in a ski-resort (4h)
- 14: pleiades for thermal dynamic simulation (8h)
- 15: fast design of building systems with `baterm` (8h)

Each session can be followed alone.

batem source code is available on GitLab at https://gricad-gitlab.univ-grenoble-alpes.fr/batem/batem_all

baterm is also available on PyPI at https://pypi.org/project/batem/

It's easy to install with Python/pip: `pip install baterm` (python 3.13 recommended)

The basic commands of `baterm`

command	description
help	show help for a command (`help alias` , 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.

```
flowchart LR
    A[context\ncommands] --> B(context)
    B --> C[building\ncommands]
    B --> D[pv\ncommands]
    D --> E(photovoltaic\nplant)
    C --> F(building)
    F --> D
    E --> G[global\ncommands]
    F --> G
```

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

The context commands

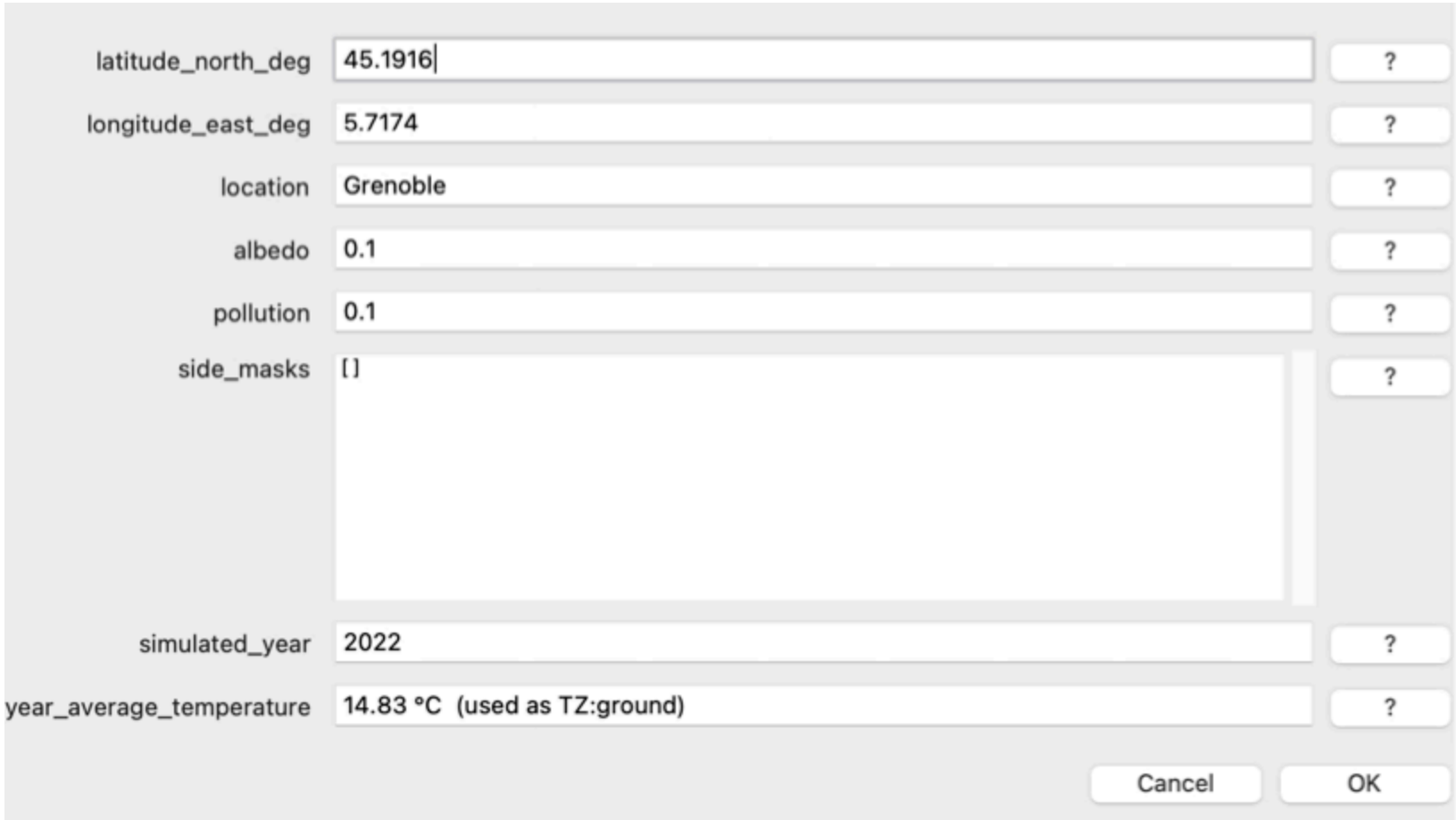
A context is mainly defined by a location: a latitude North and a longitude East in decimal degrees. It is used to load the historical weather data from http://open-meteo.com, to get the elevation of the location and to collect the far masks due to surrounding mountains.

A context can be created with the command `context new`. Default values are proposed. Then it can 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)

Side-masks are used to model the surrounding buildings. They are defined by a list of rectangular surfaces with their dimension 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
    }
```

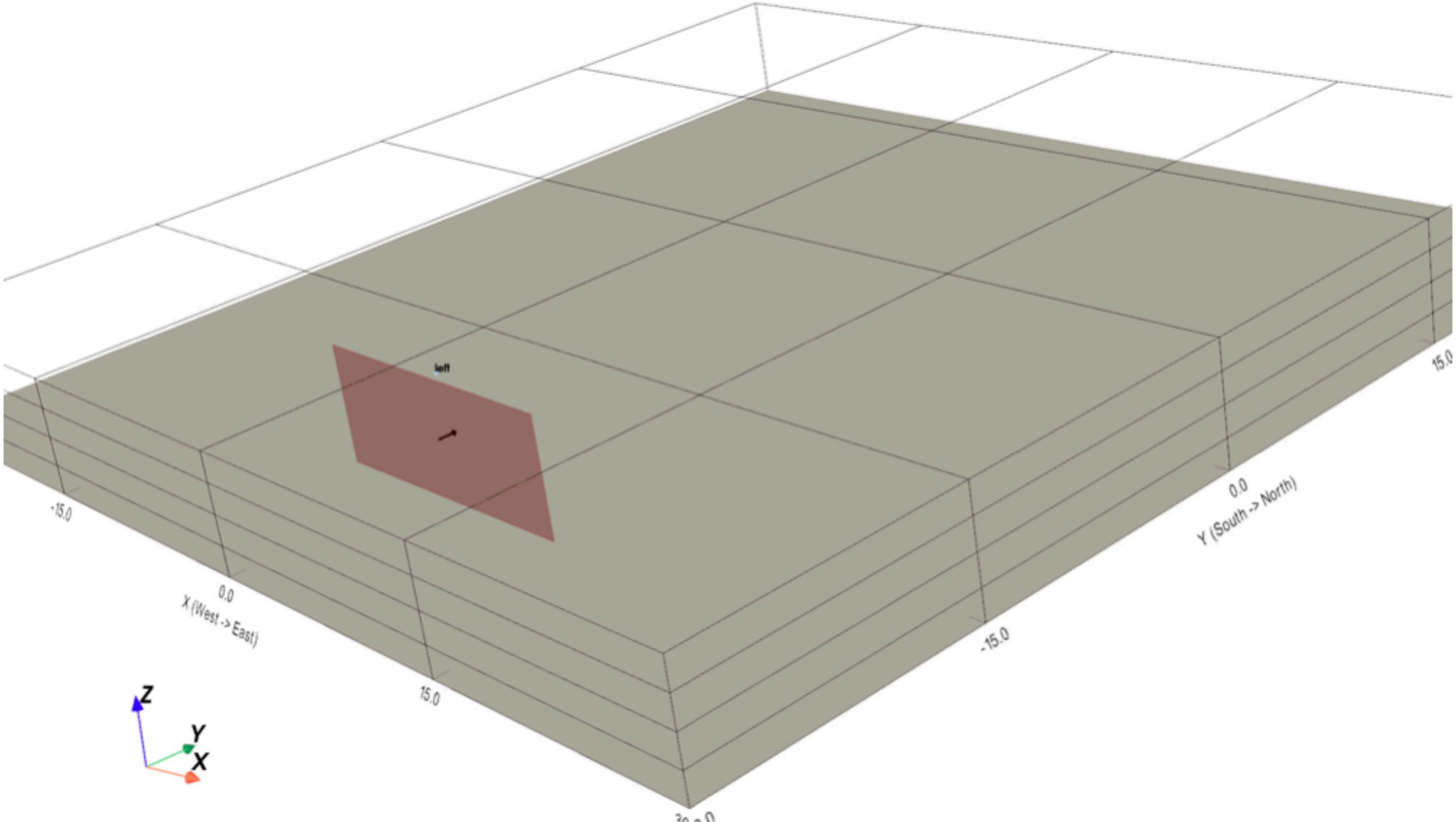
Here, side-masks are used to model vertical rectangular surfaces (slope = 90°) touching the ground (elevation = 0).

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 ± 180° or 360°) means South or North, exposure_deg= ± 90° 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`:



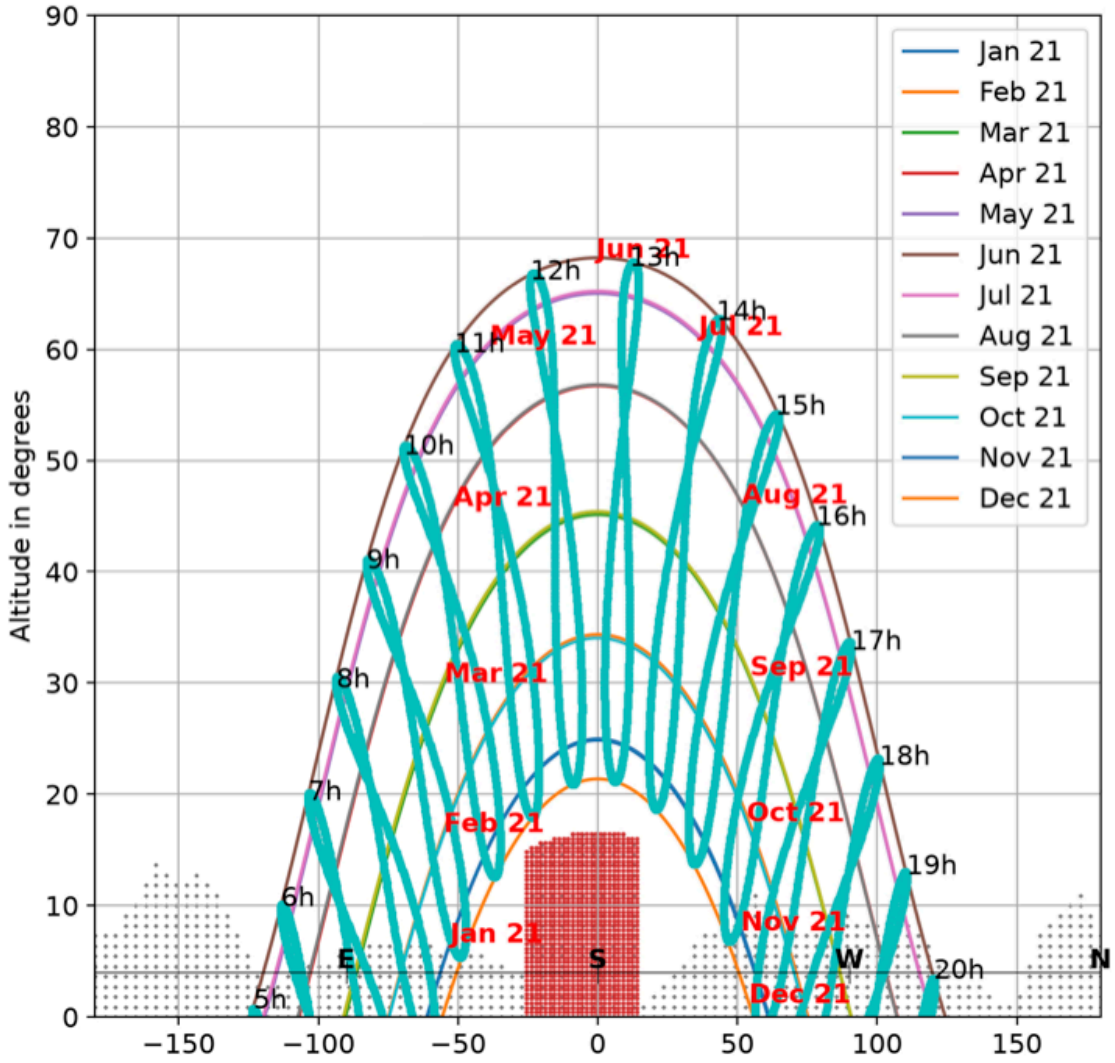
The heliodon, representing the sun path each 21st of months of the year, is shown with the command `context heliodon` knowing that the observer is located at the (0,0,0) position i.e. at the specified gps coordinates.

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 heatwave	graphical tool to analyze the heatwave
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]`

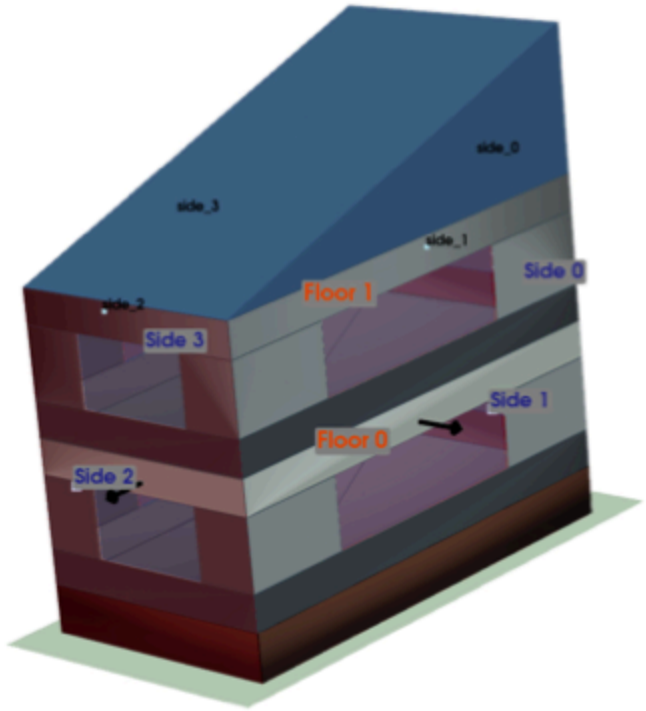
Material names come from `propertiesDB.xlsx` (see `material list thermal`) but only the ones with a star in front are loaded. If you load another one, use the command `material load <a_name> <row_number>`.

Roofs

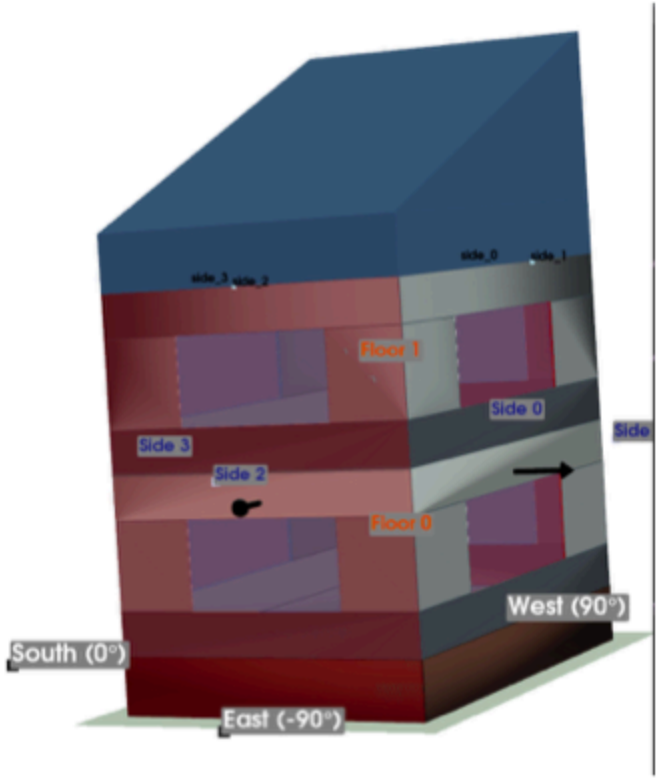
A roof can be extruded along a given height: `roof_extrusion: 1.0`. There will be a additional 1m strip in roof material in the continuity of the walls.

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

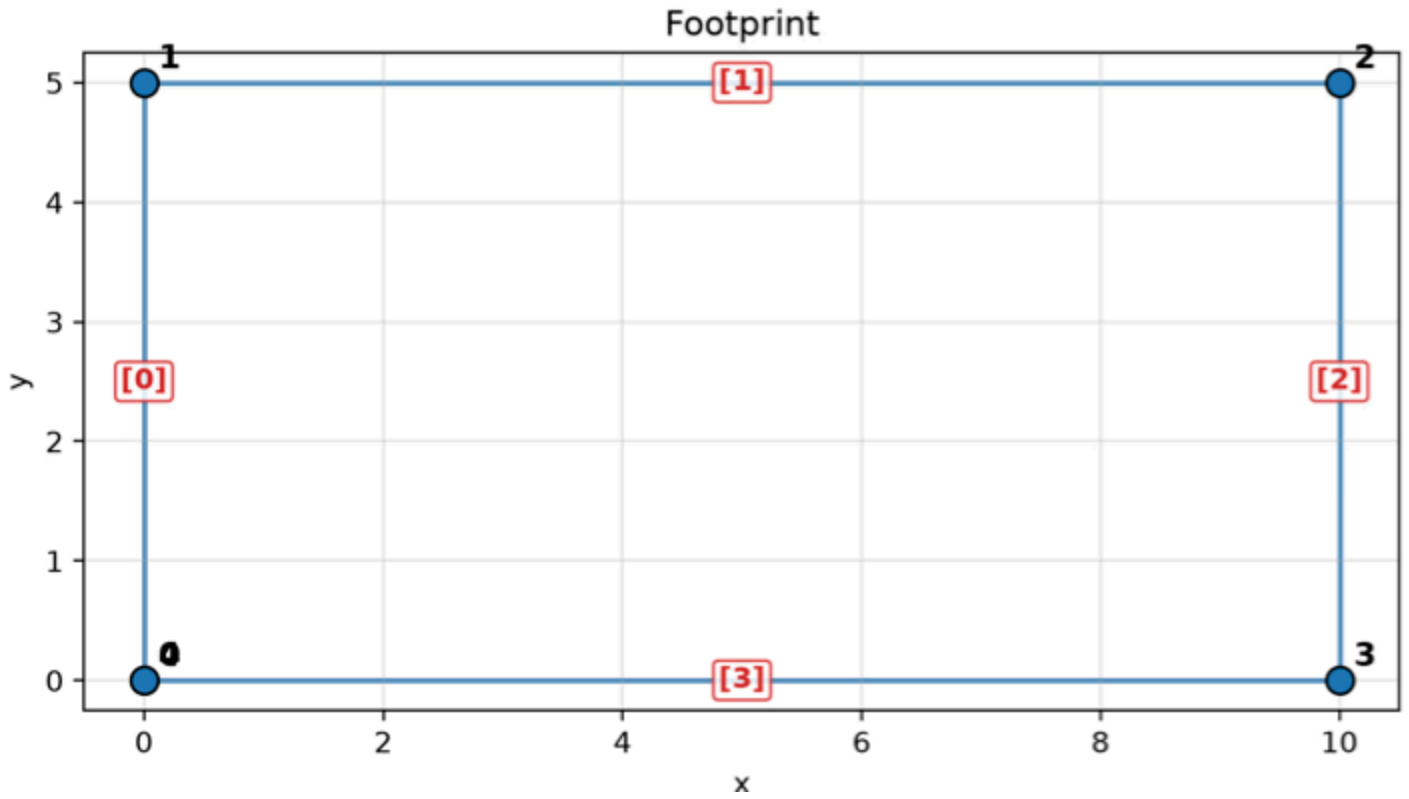
In the building form, 2 kinds of roof shapes can be defined: the aperiodic roofs (periodic=False) and the periodic roofs (periodic=True). Defaults keep a flat ceiling: `"roof_points": []` and do not depends on the kind of roof shape.

Aperiodic roof is defined by:

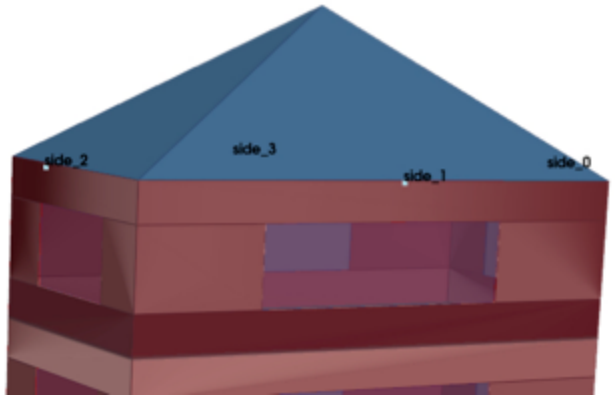
- `roof_points` — ridge / apex `(x, y, z)` in meters (same axes as the footprint in meters) define a convex hull on the roof surface where z=0 stands for the top of the building without roof but keeping the footprint.
- `roof_extrusion` — height of the vertical roof base above the top storey [m]
- `roof_wall_ratio` — 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`.

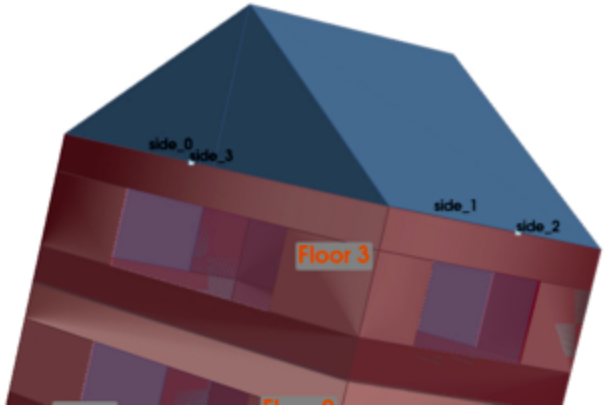
For example, the following roofs are defined for a building whose footprint is `[0,0,0], [10,0,0], [10,5,0], [0,5,0]`.



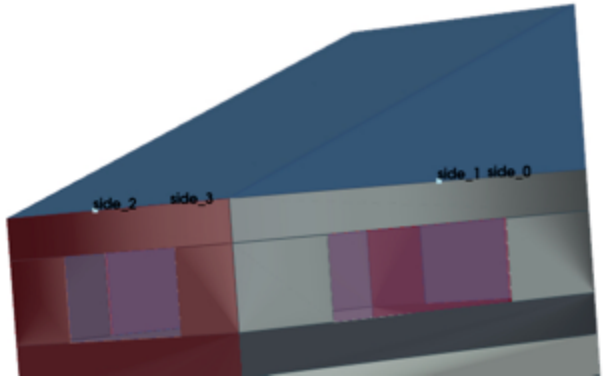
A pointed roof can be defined by one point: `[5, 2.5, 3]`.



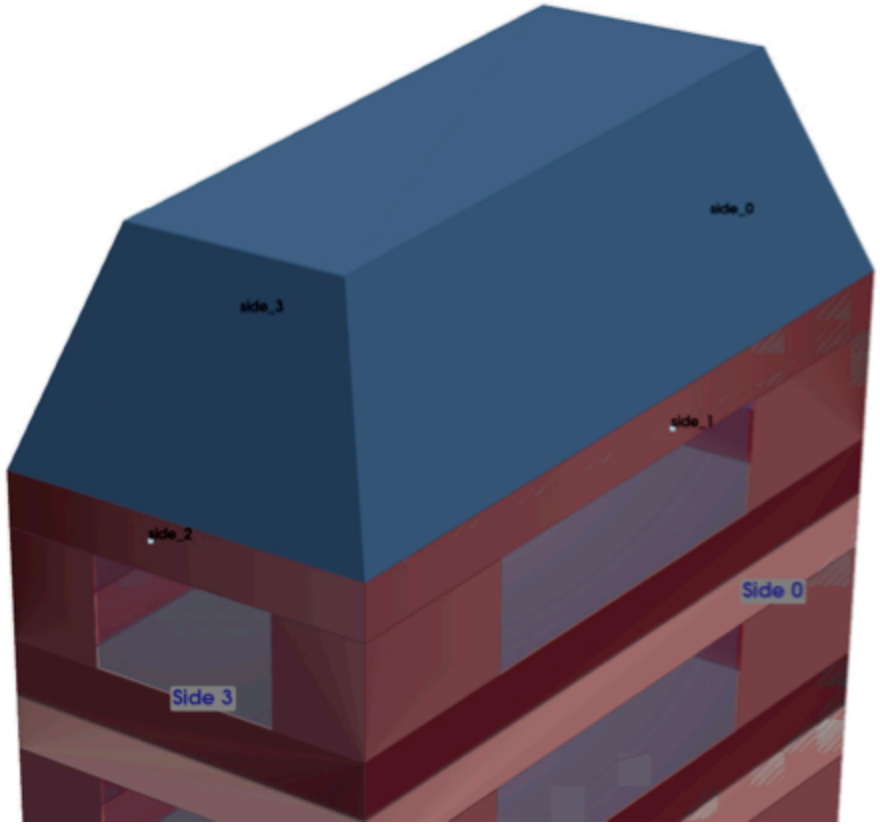
A hip roof can be defined by two points: `[5,0,3], [5,5,3]`.



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)
```

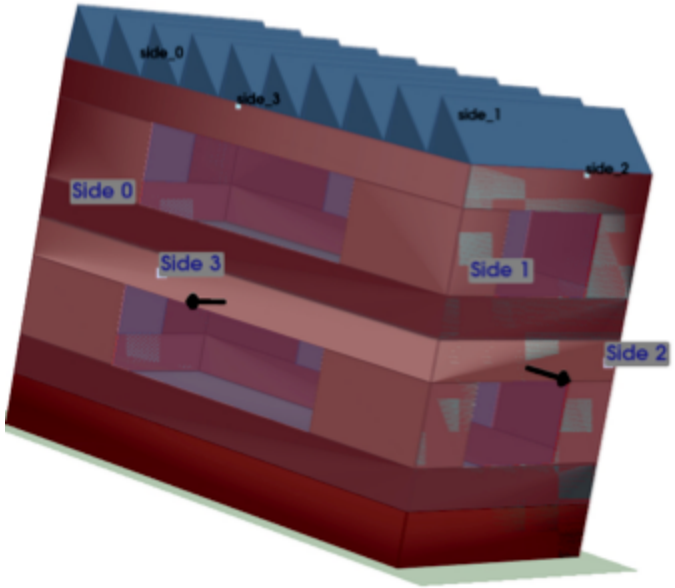
- `fraction` — position along one period (`0` = start, `1` = end of the period)
- `z` — roof height above the top of the extrusion [m]

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.

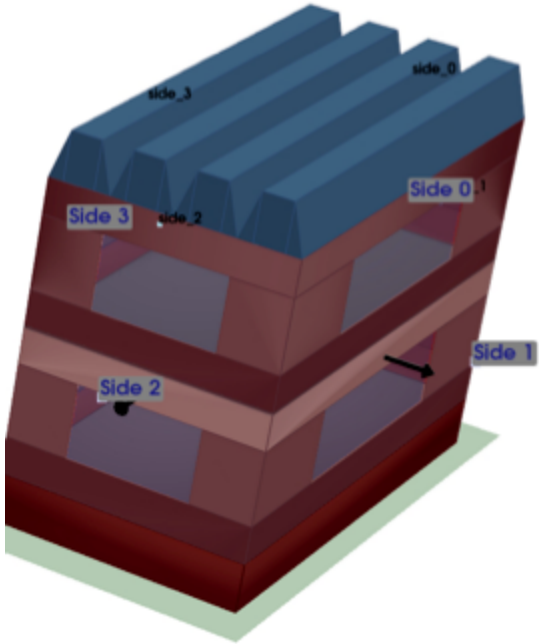
Saw-tooth / shed periods (10 repeats along `x`):

```
"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.

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

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 `.building` file from `./data`
building save <path>	save the building to `./data`
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
building plot	plot any building variables

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

Examples:

```
building variation --quick
building variation heat_recovery --quick
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 ... --quick	limit to a few cases (fast)
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, baterm 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` , `saw` , `arrow` , …) changes self-shading between rows. See slides01-sun.pdf for more details.

Typical PV loop:

```
context load project
pv new
pv 3d
pv best angles
pv simulate --quick
pv simulate
pv save project
```

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

`pv simulate` writes collector powers and the total `PV:power_W` on the shared DataProvider.

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)
pv load <path>	load a `.pv` file from `./data`
pv save <path>	save the plant to `./data`
pv delete	forget the plant in memory (or delete a file)
pv dir	list `.pv` files in `./data`

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
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:

$$\text{balance} = \text{PV production} - \text{building electricity demand}$$

- **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.