Static and Dynamic drivers

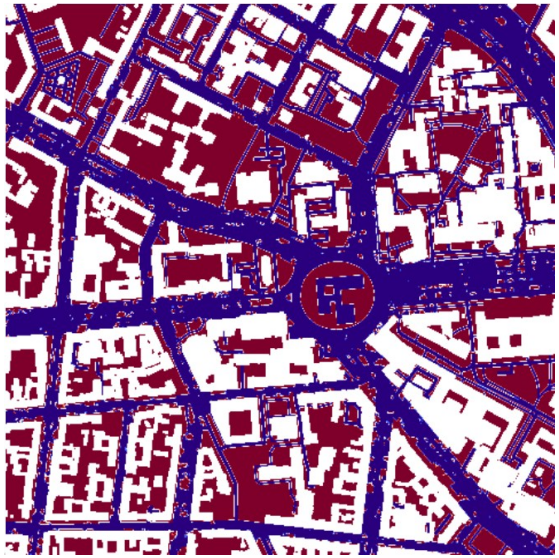
palmgroup

Institute of Meteorology and Climatology, Leibniz Universität Hannover

Content

- General information
- Static driver
- Dynamic driver
- Initialization
- Examples

Why is there a need for standardized input?



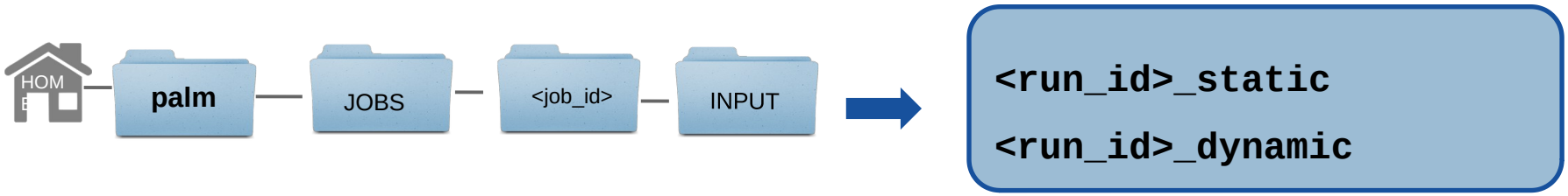
- More and more simulations require detailed realistic setups.
- Information required about terrain, buildings, surface types, soil properties, initial states, large-scale forcing, etc.
- ASCII file format would be bothersome to create and maintain; quick looks would be almost impossible

  → Standardized input via NetCDF files.

Static and dynamic drivers

General information

- Static and dynamic input files (driver files) are in NetCDF format.
- Drivers are optional; if not available, a simulation is initialized according to setup given in parameter file.
- Files must reside in input directory of a job and named
  - **`<run_id>_static`** for a static driver
  - **`<run_id>_dynamic`** for a dynamic driver



Static and dynamic drivers

Static driver

- Static driver: all data which is constant in time:
  - terrain height
  - building information (height, ID, type, surface properties)
  - tall vegetation (leaf and basal area density)
  - surface types and properties (non-building surfaces like water, vegetation, pavement)
  - geographical information (latitude, longitude, orientation)
  - classification between natural and built-type surfaces (handled by LSM and BSM)
- Data can be provided in different level of detail (lod):
  - lod 1: building heights for x/y position; only surface-mounted buildings
  - lod 2: 3D building geometry, overhanging structures like bridges or tunnels are possible
- NetCDF data must follow the **P**ALM Input **D**ata **S**tandard (PIDS).
- See full description and example file: http://palm-model.org/trac/wiki/doc/app/iofiles/pids

Extract of PIDS

[static] ***buildings_2d(y, x), buildings_3d(z, y, x)***

building topology or building height, depending on setting of attribute *lod*. z=0 refers to the highest point of terrain height occupied by that building.

type: NC_FLOAT, NC_BYTE

coordinates

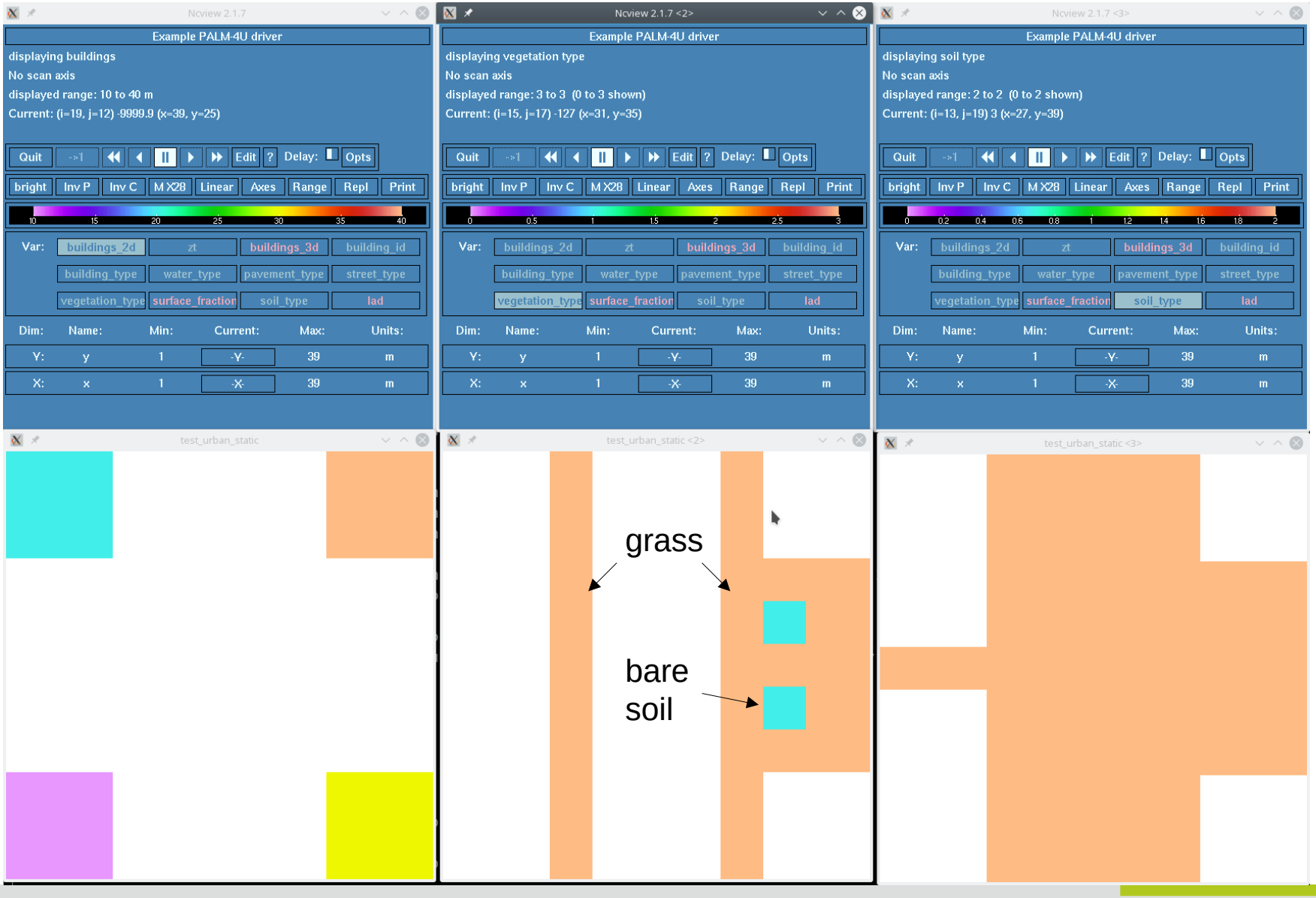
z	z-position (in m) (*lod* = *2* only)
y	y-position (in m)
x	x-position (in m)

attributes

(char)	*long_name*	*"building_height"* or *"building_flag"* (*lod* = 2)
(char)	*res_orig*	original resolution of the data in m
(char)	*source*	*data source, e.g., "satellite data"*
(char)	*units*	*"m"* or *"1"* (*lod* = 2)
(float/ byte)	*_FillValue*	*-9999.0* or *-127b* (*lod* = 2)
(char)	*coordinates*	*"E_UTM N_UTM lon lat"*
(char)	*grid_mapping*	*"crsUTM: E_UTM N_UTM crsETRS: lon lat"*
(int)	*lod*	Level of detail (1 or 2) *lod* = *1* surface-mounted buildings (no holes), the variable provides building heights in m *lod* = *2* 3D-topology, the variable provide either 1b (building) or 0b (no building)
(byte)	*valid_range*	*0b, 1b*; valid values (*lod* = *2* only)
(byte)	*flag_values*	*0b, 1b*; available values (*lod* = *2* only)
(char)	*flag_meanings*	„no building, building“ (*lod* = *2* only)

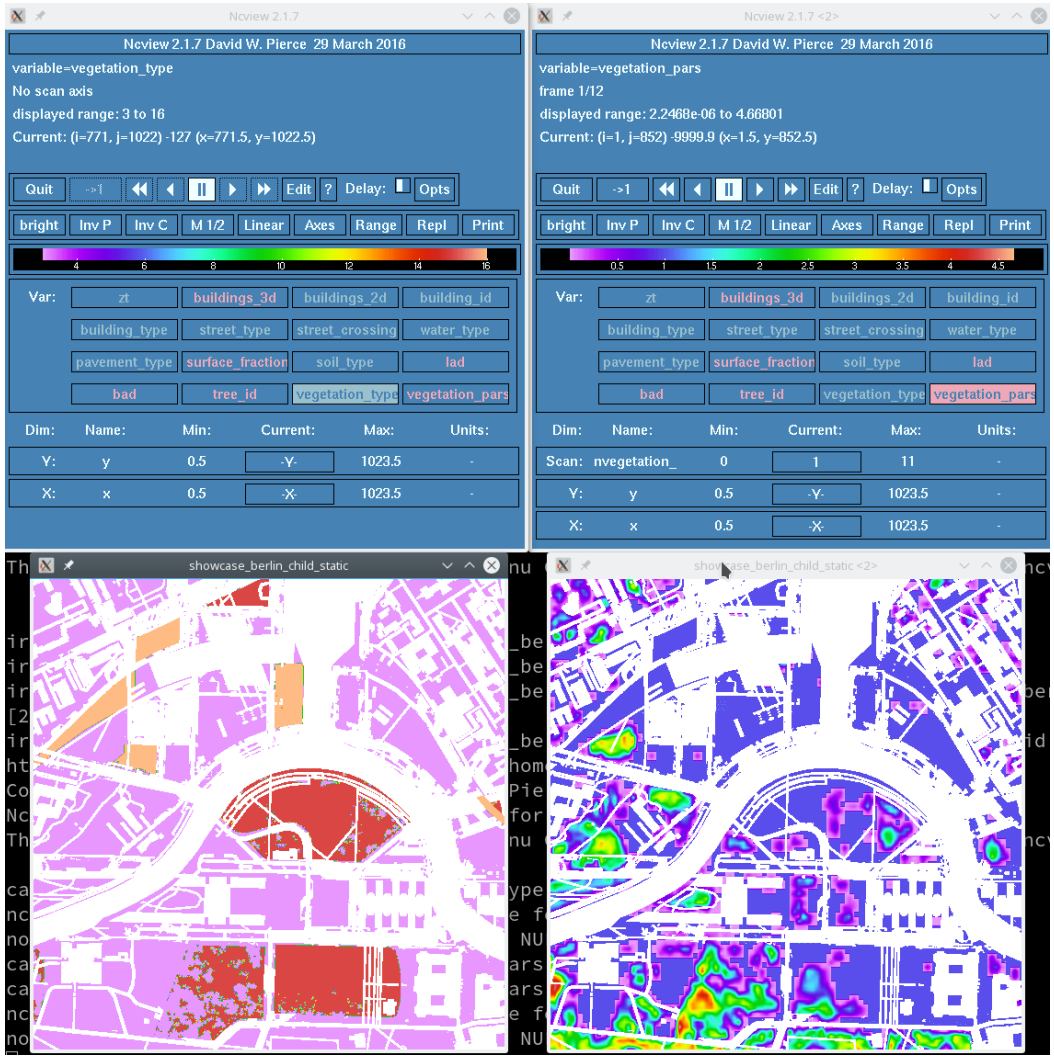
Static and dynamic drivers

Static driver



Static and dynamic drivers

Static driver

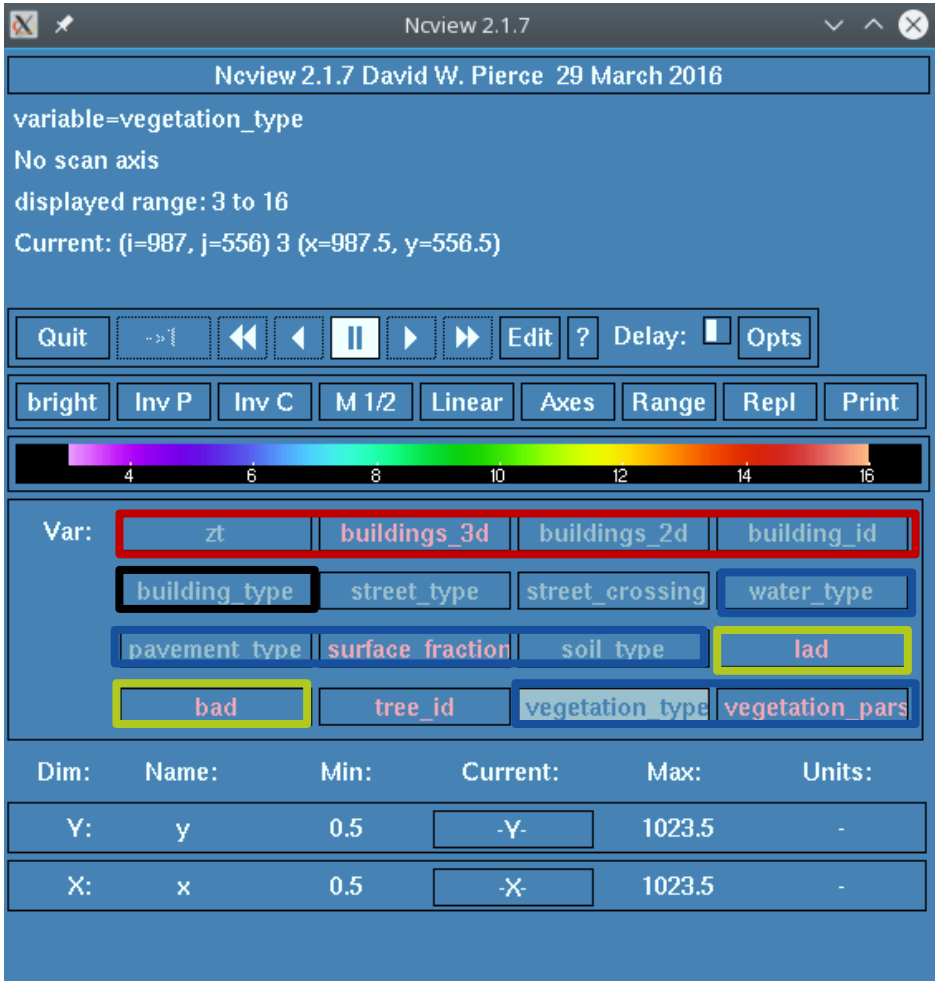


- **_FillValues** are set for each (x/y) location where no data is defined
- **_type** variables:
  - predefined list of bulk parameters for each type (roughness, albedo,...)
- **_pars** variables:
  - used to modify single surface properties at specified grid points
- Example static driver can be found at:

```
palm_model_system/packages/
palm/model/tests/cases/urba
n_environment/
```

Static and dynamic drivers

Static driver



- Variables will only be used if following parameters/namelists are set in the parameter file:
  - `topography = 'read_from_file'`
  - `&urban_surface_parameters`
  - `&land_surface_parameters`
  - `&plant_canopy_parameters`

Initialization of surface properties via static driver

- Initialization follows a 3-step hierarchy.
- Example: pavement surfaces
  1. Surface with pavement fraction are initialized via bulk parameters given by default type or namelist parameter.
  2. Surfaces are initialized via bulk parameters given by **`pavement_type`** from static driver.
  3. Single properties (roughness, albedo, emissivity, pavement depth) are overwritten via **`pavement_pars`** from static driver.
- Similar initialization for vegetation, and water surfaces and soil properties, etc.

Initialization of buildings

- Special case: buildings
  1. Surfaces are initialized via bulk parameters given by **`building_type`** from static driver.
  2. Single properties (wall fraction, window fraction, emissivity, heat capacities, ...) are overwritten via **`building_pars`** from static driver

Initialization of buildings

- Special case: buildings
  1. Surfaces are initialized via bulk parameters given by **`building_type`** from static driver.
  2. Single properties (wall fraction, window fraction, emissivities, heat capacities, ...) are overwritten via **`building_pars`** from static driver
     Note: building_pars does not allow to set properties for single walls or surface elements!

Initialization of buildings

- Special case: buildings

1. Surfaces are initialized via bulk parameters given by **`building_type`** from static driver.
2. Single properties (wall fraction, window fraction, emissivities, heat capacities, ...) are overwritten via **`building_pars`** from static driver
   Note: building_pars does not allow to set properties for single walls or surface elements!
3. Single properties for individual surface elements can be overwritten via
4. **`building_surface_pars(nbuilding_surface_pars,ns)`**
   `ns:` number of surface element
5. **`nbuilding_surface_pars:`** number of parameter
   → 1D array with:
6. **`x(ns), y(ns), z(ns), zenith(ns), azimuth(ns)`**

Minimum requirements and example script

- If 2D/3D buildings are present
  - **`building_id`** at each (x,y) position where buildings are defined
  - **`building_type`** if the urban-surface model is used (**`&urban_surface_parameters`**)
- If land-surface model is used (**`&land_surface_parameters`**)
  - **`pavement_type`**, **`vegetation_type`**, **`water_type`**, **`soil_type`**
  - At least one of **`pavement_type`**, **`vegetation_type`**, **`water_type`**, must be set at each location where no building is defined.
  - If **`pavement_type`** or **`vegetation_type`** is set, **`soil_type`** must be set at the same positions.
  - Tile approach: If more than one of **`pavement_type`**, **`vegetation_type`**, **`water_type`** is defined at a single location, **`surface_fraction`** must also be set for this position.

    NOTE: tile approach is not fully implemented and should not be used at the moment
- An example static input file and a script to create an input file can be found at:
  `palm_model_system/packages/palm/model/tests/cases/urban_environment/INPUT/`

Minimum requirements and example script

- If 2D/3D buildings are present
  - **`building_id`** at each (x,y) position where buildings are defined
  - **`building_type`** if the urban-surface model is used (**`&urban_surface_parameters`**)
- If land-s
  - **`paver`**
  - At lea… must be set at each l…
  - If **`pav`**… be set at the same …
  - Tile ap… **`pe`**, **`water_type`** is defined at a single location, **`surface_fraction`** must also be set for this position.

    NOTE: tile approach is not fully implemented and should not be used at the moment

Be very careful with creating static driver files / data!
Always make consistency checks!
Currently, badly configured static drivers may cause PALM to crash. Such crashes are very difficult to debug.
We try to improve and extend our automatic driver all the time!

- An example static input file and a script to create an input file can be found at:
  `palm_model_system/packages/palm/model/tests/cases/urban_environment/INPUT/`

Using `palm_csd`

- **`palm_csd`** (palm create static driver) can be used to create static drivers for pre-processed data (i.e. rastered NetCDF data containing the required information)
- Requires configuration file: **`csd_config.yml`** in your PALM main directory
- Example available at

  ```
  palm_model_system/packages/static_driver/palm_csd/share/
  ```

- Currently, suitable open source data are available for Berlin and Hamburg (on request)
- Alternatively, use own data and Python template available at

  ```
  palm_model_system/packages/static_driver/
  create_basic_static_driver/create_basic_driver.py
  ```

Using `palm_csd`

- Step 1: edit configuration file

```
attributes:
  author: Bjoern Maronga, maronga@muk.uni-hannover.de
  contact_person: Bjoern Maronga, maronga@muk.uni-hannover.de
  acronym: LUHimuk
  comment: created with palm_csd
  data_content:
  dependencies:
  keywords:
  source:
  campaign:
  location: B
  site: Berlin Mitte
  institution: Leibniz University Hannover
  palm_version: 6.0
  rotation_angle: 0.0
  references:
```

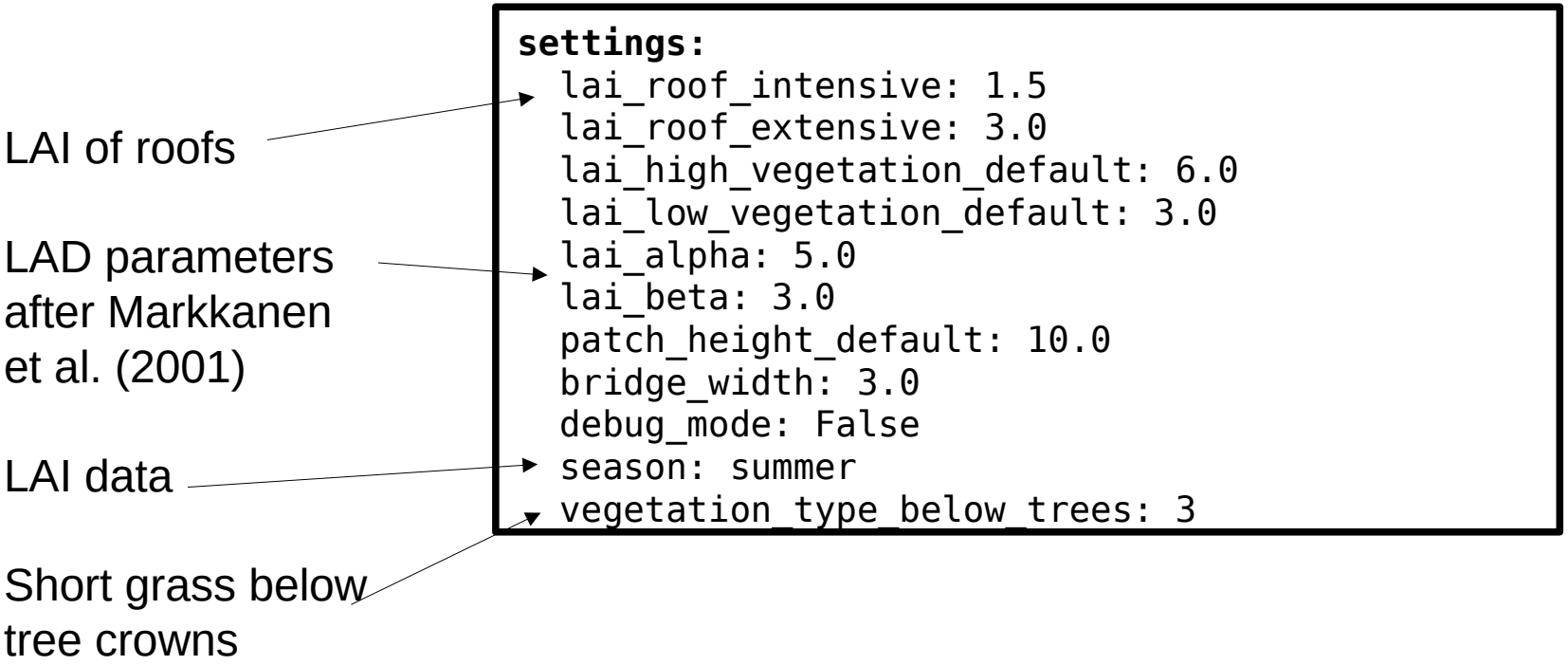
Using `palm_csd`

- Step 1: edit configuration file

```
settings:
  lai_roof_intensive: 1.5
  lai_roof_extensive: 3.0
  lai_high_vegetation_default: 6.0
  lai_low_vegetation_default: 3.0
  lai_alpha: 5.0
  lai_beta: 3.0
  patch_height_default: 10.0
  bridge_width: 3.0
  debug_mode: False
  season: summer
```

Using `palm_csd`

- Step 1: edit configuration file



Using `palm_csd`

- Step 1: edit configuration file

```
input_01:
  path: /ldata2/MOSAIK/Berlin_static_driver_data
  pixel_size: 15.0
  file_x: Berlin_CoordinatesUTM_x_15m_DLR.nc
  file_y: Berlin_CoordinatesUTM_y_15m_DLR.nc
  file_x_UTM: Berlin_CoordinatesUTM_y_15m_DLR.nc
  file_y_UTM: Berlin_CoordinatesUTM_x_15m_DLR.nc
  file_lon: Berlin_CoordinatesLatLon_x_15m_DLR.nc
  file_lat: Berlin_CoordinatesLatLon_y_15m_DLR.nc
  file_zt: Berlin_terrain_height_15m_DLR.nc
  file_buildings_2d: Berlin_building_height_15m_DLR.nc
  file_building_id: Berlin_building_id_15m_DLR.nc
  file_building_type: Berlin_building_type_15m_DLR.nc
  file_bridges_2d: Berlin_bridges_height_15m_DLR.nc
  file_bridges_id: Berlin_bridges_id_15m_DLR.nc
[...]
```

Using `palm_csd`

- Step 1: edit configuration file

```
output:
  path: /ldata2/MOSAIK/
  file_out: showcase_berlin
  version: 1
```

Using `palm_csd`

- Step 1: edit configuration file

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: False
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Create no 3D buildings (grid too coarse)

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: False
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Allow land surface types that represent tall vegetation. Not allowed for fine grid spacings!

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Generate 3D LAD fields for canopies (tall vegetation canopies)

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Output data is (not) rastered on the staggered PALM grid

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Terrain heights are not interpolated to the PALM grid (PALM does that...)

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Do not allow green roofs

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

Create 3D LAD/LAB fields for individual trees

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Using `palm_csd`

- Step 1: edit configuration file

```
domain_root:
  pixel_size: 15.0
  lower_left_x: 0
  lower_left_y: 0
  nx: 3119
  ny: 2573
  buildings_3d: False
  dz: 15.0
  allow_high_vegetation: True
  generate_vegetation_patches: True
  use_palm_z_axis: False
  interpolate_terrain: False
  vegetation_on_roofs: False
  street_trees: True
  overhanging_trees: False
```

Do not render tree crowns hanging over buildings (can cause issues)

Using `palm_csd`

- Step 1: edit configuration file

You can define multiple domains for PALM self-nesting configurations!

- Domain: N02
- Parent of N02
- Terrain height is interpolated to match the root domain (this avoids thresholds between domain heights)

```
domain_N02:
  pixel_size: 1.0
  origin_x: 19605
  origin_y: 20895
  nx: 1023
  ny: 1023
  domain_parent: root
  buildings_3d: True
  dz: 1.0
  allow_high_vegetation: False
  generate_vegetation_patches: True
  use_palm_z_axis: True
  interpolate_terrain: True
```

Using `palm_csd`

- Step 2: run **`palm_csd`**

```
palm_csd path/to/csd_config.yml
```

```
Writing global attributes to file...
Shift terrain heights by -30.879060745239258
Writing attribute origin_z to file...
Writing dimension x to file...
Writing dimension y to file...
Writing array lat to file...
Writing array lon to file...
Writing array E_UTM to file...
Writing array N_UTM to file...
Writing crs to file...
Writing array zt to file...
...
```

- Step 3: check static driver (here: **`/ldata2/MOSAIK/static_driver_example.nc`**)

Using `palm_csd`

- Step 2: run **`palm_csd`**

```
palm_csd path/to/csd_config.yml
```

```
Writing global attributes to file...
Shift terrain heights by -30.879060745239258
Writing attribute origin_z to file
Wr
Wr
Wr
Wr
Writing array E_UTM to file...
Writing array N_UTM to file...
Writing crs to file...
Writing array zt to file...
...
```

Attention: A documentation of palm_csd is still missing and work in progress!

- Step 3: check static driver (here: **`/ldata2/MOSAIK/static_driver_example.nc`**)

- Remarks / Outlook: **palm_csd**
  - A more general tool based on python3 and qgis will be available in the future (in combination with palm_csd)
  - For now, check palmpy: https://github.com/stefanfluck/palmpy
  - palm_csd is currently under development to allow for setting up the DCEP model (urban parameterization for coarse grid that do not allow to resolve individual buildings) based on local climate zone (LCZ) classification

Dynamic driver

- Dynamic driver: all data which is variable in time
  - initialization data for atmosphere (wind, moisture, potential temperature) and soil state (temperature, moisture)
  - geostrophic wind
  - boundary data for offline nesting
- Initialization data can be provided in different level of detail (lod):
  - lod 1: profiles of, e.g., potential temperature
  - lod 2: 3D initialization data of, e.g., potential temperature for each grid point
- At the moment, dynamic drivers are created via **`inifor`** (to be replaced by **promet**)
- **`inifor`** processes COSMO output data for a given point in time and given coordinates and creates a dynamic driver fitting for PALM.
- Alternatively there is a wrf interface
- See full description at: http://palm-model.org/trac/wiki/doc/app/iofiles/pids

Dynamic driver

- To enable initialization with dynamic driver, set **`initializing_actions = 'read_from_file'`**
- To enable offline nesting with dynamic driver, set NAMELIST:
- &nesting_offl_parameters /
- Dynamic driver code (inifor / WRF interface) is located at:

```
palm_model_system/packages/dynamic_driver/
```

- Remarks / Outlook: **promet**
  - Flexible interface for COSMO, ICON, ICON-Art, WRF, WRF-Chem

Static and dynamic drivers

Dynamic driver