
QES






Release 1.0

Fabien Margairaz

Aug 16, 2023

USER GUIDE

1	Getting Started	3
1.1	Introduction	3
2	Installation	5
2.1	Required Packages	5
2.2	Cloning QES-Winds from GitHub	5
2.3	Building QES	6
2.4	Build Types	8
3	Running QES	9
3.1	Running from the Command Line	9
3.2	slurm Template (for CUDA 11.4 build	10
3.3	Testing QES	10
4	QES-Winds	13
4.1	Introduction	13
4.2	QES-Winds Domain	14
4.3	Staggered Grid	14
4.4	Digital Elevation Model (DEM) and ESRI Shapefile	16
4.5	Initial Wind Field	21
4.6	Empirical Parameterizations	28
4.7	Mass Consistent Solver	46
5	QES-Turb	47
6	QES-Plume	49
6.1	The model	49
6.2	The XML file	49
7	Publications	53
8	Reference List	55
9	Acknowledgements	57
	Bibliography	59

Behnam Bozorgmehr , Jeremy A Gibbs , Fabien Margairaz , Eric R Pardyjak , Rob Stoll , ,

The Quick Environmental Simulation (**QES**) code is a low-computational-cost framework designed to compute high-resolution wind and concentration fields in complex atmospheric-boundary-layer environments.

The modules are the following:

- **QES-Winds** is the new wind model computing divergence-free steady-state 3D wind field in complex domain.
- **QES-TURB** is a stand-alone turbulence model that computes turbulence fields from the calculated wind field in QES-Winds.
- **QES-Plume** is a stand-alone Lagrangian dispersion model with the ability to calculate spatially and temporally varying scalar concentrations.
- **QES-Fire** is a fire-spread model.
- **QES-Transport** is a transport model (to be implemented)

Note: QES requires a NVIDIA GPU with Compute Capability of 7.0 (or higher).

GETTING STARTED

1.1 Introduction

A new dispersion modeling system based on the well-used FORTRAN-based QUIC (Quick Urban and Industrial Complex) dispersion modeling system originally developed by the University of Utah and Los Alamos National Laboratory [1], has been under development as collaboration between the University of Utah, the University of Minnesota, Duluth and Pukyong National University. Quick Environmental Simulation (QES) is a microclimate simulation platform for computing 3D environmental scalars in urban areas and over complex terrain. QES-Winds, QES-TURB and QES-Plume are mean wind modeling, turbulence, and plume dispersion modeling components of QUIC EnvSim (QES). Figure below shows a schematic of QES system and how different elements of the system interact with each other.

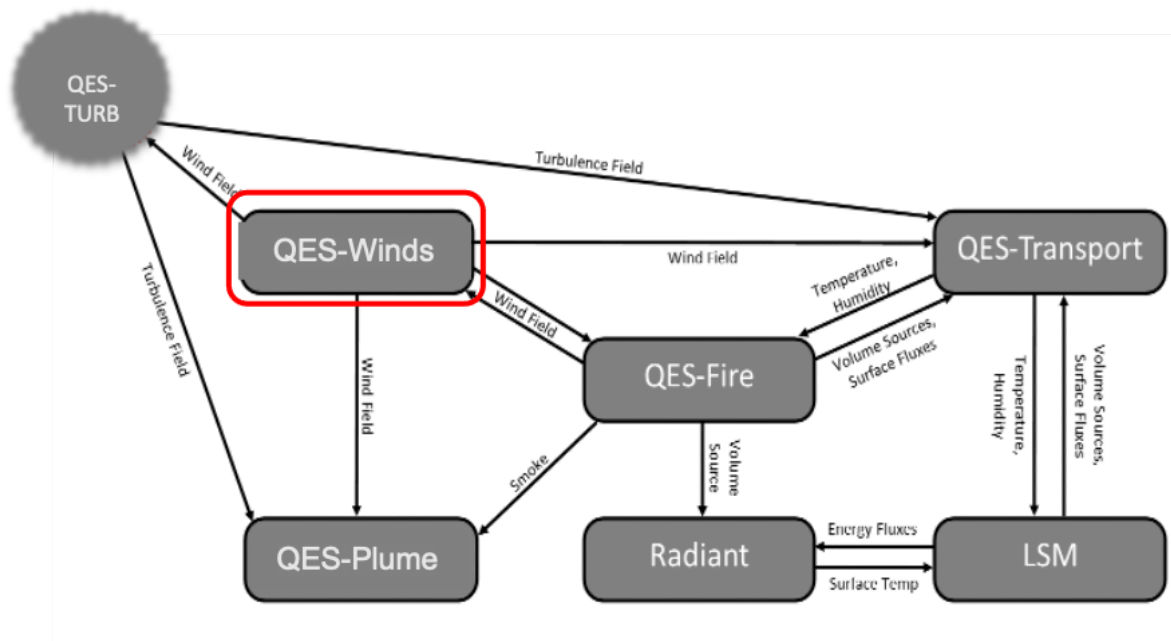


Fig. 1: Schematic of the QUIC EnvSim system and the relationship between different elements of the system including data flow from one element to the other

The QES code is a low-computational-cost framework designed to compute high-resolution wind and concentration fields in complex atmospheric-boundary-layer environments. QES is written in C++ and NVIDIA's CUDA for Graphics Processing Unit (GPU) acceleration. The code uses NVIDIA's dynamic parallelism API to substantially accelerate simulations. **QES requires a NVIDIA GPU with Compute Capability of 7.0 (or higher).**

1.1.1 QES-Winds

QES-Winds is a fast-response 3D diagnostic urban wind model using a mass-conserving wind-field solver [2]. QES-Winds uses a variational analysis technique to ensure the conservation of mass rather than slower yet more physics-based solvers that include the conservation of momentum. QES-Winds minimizes the difference between an initial wind field that is specified using empirical parameterizations and the final wind field. This method requires the solution of a Poisson equation for Lagrange multipliers. The Poisson equation is solved using the Successive Over-Relaxation (SOR) method (an iterative solver), which is a variant of the Gauss-Seidel method with more rapid convergence.

1.1.2 QES-Turb

QES-Turb is a turbulence model based on Prandtl's mixing-length and Boussinesq eddy-viscosity hypotheses. QES-Turb computes the stress tensor using local velocity gradients and some empirical non-local parameterizations.

1.1.3 QES-Plume

QES-Plume is a stochastic Lagrangian dispersion model using QES-Winds mean wind field and QES-Turb turbulence fields. QES-Plume solves the generalized Langevin equations to compute the fluctuations of the particle in the turbulent flow fluid. A time-implicit integration scheme is used to solve the Langevin equation, eliminating 'rogue' trajectories. The particles are advanced using a forward Euler scheme. QES-Plume is also a stand-alone dispersion model that can run using fields from diverse sources such as RANS or LES models.

1.1.4 QES-Fire

QES-Fire is a microscale wildfire model coupling the fire front to microscale winds. The model consists of a simplified physics rate of spread model, a kinematic plume-rise model, and a mass-consistent wind solver. The QES-Fire module is currently not publicly available.

INSTALLATION

This section is designed to serve as a step-by-step instruction of how to build and run QES-Winds. In the first part, packages required to build the code will be mentioned along with the oldest version of each package that satisfies the purpose. The next part will be interaction with the repository on GitHub in which the code is been stored to clone the code. Also, commands required for cloning the repository and building the executable of code, will be mentioned. The last part of this section will cover a brief description of how to change the input files of the code and run it.

2.1 Required Packages

The very first package needed to be installed is "git" package. It provides the ability to interact with GitHub and use commands to clone the repository, switch between different branches and etc. This package does not have any dependencies, so it is always recommended to install the latest version.

- The next package inline is "CMake" and its GUI version "CCMake". It finds all the packages required, links them together and creates the "makefile" for building the code. CMake should be any version greater than 3.10.
- QES-Winds also needs "boost" libraries in order to have access to C++ source libraries. Boost 1.66.0 is sufficient for the purpose of QES-Winds.
- "Gdal" libraries are necessary to read in Digital Elevation Models (DEM) and shapefile (for buildings). Version 2.3.1 of gdal libraries will do the job for our applications.
- The last library that needs to be installed is "netcdf-c" libraries along with netcdf interface with C++, version 4.6 is required. Netcdf libraries are essential for reading in WRF output files and writing QES-Winds results in netcdf format.
- Finally, since QES-Winds is written in C++ and CUDA, "gcc" and "CUDA" compilers needed to be installed. Because there is a compatibility issue between versions of CUDA, gcc and Operating System(OS) (for more information go to <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>), version of gcc that is compatible with the version of CUDA and OS is required. For CUDA, at least version 8.0 needs to be installed.

2.2 Cloning QES-Winds from GitHub

After making sure all the required packages are installed and ready to use, a copy of QES-Winds needs to be downloaded on the local computer (cloning process). To clone the code, go to the directory you want to have the code downloaded, open a terminal and type "git clone [address to the repository]". To get the address to the repository, go to the repository GitHub page, UtahEFD/QES-Winds-Public, click on the green button "Code" and copy the "HTTPS" address. It downloads a copy of the code in the "master" branch of the repository in your local directory.

2.3 Building QES

Next steps are:

- Go to the folder created with name QES-Winds: "cd QES-Winds-Public".
- Create a build directory: "mkdir build" or "sudo mkdir build".
- Go to folder build: "cd build".
- Type: "cmake ..".

There is a chance that cmake fails to find all the packages needed for running the code (packages installed on unconventional directories). In this case, you need to do cmake with appropriate flags that point to those packages.

- After cmake is done successfully, type: "build"
- A successful build will result in creating the executable named "qesWinds"

2.3.1 Linux

On a general Linux system, such as Ubuntu 18.04 or 20.04, the following packages need to be installed:

- libgdal-dev
- libnetcdf-c++4-dev
- libnetcdf-cxx-legacy-dev
- libnetcdf-dev
- netcdf-bin
- libboost-all-dev
- cmake
- cmake-curses-gui

If the system uses apt, the packages can be installed using the following command:

```
apt install libgdal-dev libnetcdf-c++4-dev libnetcdf-cxx-legacy-dev libnetcdf-dev ↵  
↵netcdf-bin libboost-all-dev cmake cmake-curses-gui
```

We separate the build

```
mkdir build  
cd build  
cmake ..
```

You can then build the source:

```
make
```

2.3.2 macOS

The packages can be installed using Homebrew (<https://brew.sh>)

```
brew install cmake boost gdal hdf5 netcdf netcdf-cxx
```

If openMP multithreading is desired:

```
brew install libomp
```

On intel silicon machines:

```
cmake -DNETCDF_LIBRARIES_CXX=/usr/local/lib/libnetcdf-cxx4.dylib -DENABLE_OPENMP=ON -
↳DCMAKE_PREFIX_PATH=/usr/local/Cellar/libomp/15.0.7/ ..
```

On apple silicon machines:

```
cmake -DCMAKE_PREFIX_PATH=/opt/homebrew/Cellar/libomp/15.0.7 -DENABLE_OPENMP=ON -DNETCDF_
↳LIBRARIES_CXX=/opt/homebrew/lib/libnetcdf-cxx4.dylib ..
```

2.3.3 University of Utah - CHPC

This is the preferred build setup on CHPC

The code does run on the CHPC cluster. You need to make sure the correct set of modules are loaded. Currently, we have tested a few configurations that use

- GCC 5.4.0 and CUDA 8.0
- CCC 8.1.0 and CUDA 10.1 (10.2)
- GCC 8.5.0 and CUDA 11.4

If you build with OptiX support, you will need to use CUDA 10.2 or newer configuration. Any builds (with or without OptiX) with CUDA 11.4 are preferred if you don't know which to use. Older configurations are provided in CHPC/oldBuilds.md.

After logging into your CHPC account, you will need to load specific modules. In the following sections, we outline the modules that need to be loaded along with the various cmake command-line calls that specify the exact locations of module installs on the CHPC system.

To build with GCC 8.5.0, CUDA 11.4, and OptiX 7.1.0 on CHPC. Please use the following modules:

```
module load cuda/11.4
module load cmake/3.21.4
module load gcc/8.5.0
module load boost/1.77.0
module load intel-oneapi-mpi/2021.4.0
module load gdal/3.3.3
module load netcdf-c/4.8.1
module load netcdf-cxx/4.2
```

Or use the provided load script, which will always load the latest tested configuration.

```
source CHPC/loadmodules_QES.sh
```

After completing the above module loads, the following modules are reported from 'module list':

Currently Loaded Modules:

```

1) cuda/11.4      (g)  3) gcc/8.5.0      5) intel-oneapi-mpi/2021.4.0  7) netcdf-c/4.8.
↪1
2) cmake/3.21.4   4) boost/1.77.0   6) gdal/3.3.3              8) netcdf-cxx/4.
↪2

```

After the modules are loaded, you can create the Makefiles with cmake. We keep our builds separate from the source and contain our builds within their own folders. For example,

```

mkdir build
cd build
cmake -DCUDA_TOOLKIT_DIR=/uufs/chpc.utah.edu/sys/installdir/cuda/11.4.0 -DCUDA_SDK_ROOT_
↪DIR=/uufs/chpc.utah.edu/sys/installdir/cuda/11.4.0 -DOptiX_INSTALL_DIR=/uufs/chpc.utah.
↪edu/sys/installdir/optix/7.1.0 -DCMAKE_C_COMPILER=gcc -DNETCDF_CXX_DIR=/uufs/chpc.utah.
↪edu/sys/installdir/netcdf-cxx/4.3.0-5.4.0g/include ..

```

Upon completion of the above commands, you can go about editing and building mostly as normal, and issue the ‘make’ command in your build folder to compile the source.

After you’ve created the Makefiles with the cmake commands above, the code can be compiled on CHPC:

```
make
```

Note you *may* need to type make a second time due to a build bug, especially on the CUDA 8.0 build.

2.4 Build Types

The code support several build types: *Debug*, *Release*, *RelWithDebInfo*, *MinSizeRel*. You can select the build type

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

Release is recommended for production

cmake options:

- build code without CUDA support automatically if CUDA is not supported by the system
- build code with openmp support for future multithread application, openmp is not automatically enabled. if openmp support is enable (-DENABLE_OPENMP=ON) the code will run a new red/black solver on the CPU. Unfortunately thread safety issues with some plume classes did not allow for an easy parallelization of the plume advection.
- default is *RELEASE* with most warning off, -O3 optimization. a dev mode was added -DENABLE_DEV_MODE=ON showing warnings, will build the code in *DEBUG* this option is slow and recommended only for active development.
- ClangTidy option was added
- use -DENABLE_TESTS=ON to enable unit, sanity, and regressions tests using Catch2 (<https://github.com/catchorg/Catch2>)

RUNNING QES

The command to run the QES-Winds executable created above is:

```
./qesWinds/qesWinds -q [address to XML file] -o [output file] -s [solver type] -z  
↳ [Visualization output]
```

At least three elements need to be addressed: input XML file, output file name and type of solver. The input XML file defines various variables necessary for running the code. Input files are usually located in QES-Winds/data/InputFiles and defined in command line by -q.

```
[address to XML file] = QES-Winds/data/InputFiles/XMLfilename
```

User can change the name of output file by -o outputname. QES-Winds has four solver types: solving on CPU (determined by -s 1), solving SOR equation on GPU using dynamic parallelism (determined by -s 2), GPU solver using global memory (determined by -s 3) and GPU solver using shared memory (determined by -s 4). GPU solvers are much faster than CPU solver and are highly recommended especially for large domains.

To run QES-Winds, you can take the following slurm template and run on CHPC. We'd suggest placing it in a run folder at the same level as your build folder. Make sure you change the various sbatch parameters as needed for your access to CHPC.

3.1 Running from the Command Line

QES is run from the terminal using arguments. For example:

```
./qesWinds/qesWinds -q ../data/InputFiles/GaussianHill.xml -s 2 -w -o gaussianHill
```

More info about the arguments supported by QES can be display using:

```
./qesWinds/qesWinds -?
```

3.2 slurm Template (for CUDA 11.4 build)

```
#!/bin/bash
#SBATCH --account=efd-np
#SBATCH --partition=efd-shared-np
#SBATCH --job-name=qesGaussian
#SBATCH --nodes=1
#SBATCH --mem=15G
#SBATCH --gres=gpu:titanv:1
#SBATCH --time=01:00:00
#SBATCH -e init_error.log
#SBATCH -o init_out.log
module load gcc/8.5.0
ulimit -c unlimited -s
./qesWinds/qesWinds -q ../data/InputFiles/GaussianHill.xml -s 2 -w -o gaussianHill
```

Note that if you build with a different GCC (e.g. 5.4.0), you will need to change the module load to use that version of GCC. Once the slurm file has been placed in the run folder, you can then send out the job. For example, assuming you are in the build folder and just built the code and we saved the slurm template above as a file `rGaussianHill_gpu.slurm`.

```
make clean
make
cd ../run
sbatch rGaussianHill_gpu.slurm
```

This will create the various NetCDF output files in the run folder, along with any output in the `init_error.log` and `init_out.log` files.

3.3 Testing QES

We are using `ctest` to conduct unit tests and sanity check on the code. Here are a few commands:

```
ctest          # launch all tests
ctest --verbose # launch all tests with verbose (see comment output)
ctest -N       # get list of tests
ctest -R $testname # launch only $testname
```

Here is a list of tests and testing option. Most test require manual inspection of the results. Recursive testing will be implemented in the future.

3.3.1 QES-Winds Tests

Test for QES-Winds are designed to check that to code is still running under a given set of parameters. These tests do not guarantee the validity of the results. To turn on the basic QES-wind test, use:

```
cmake -DENABLE_SANITY_TESTS=ON -DENABLE_GPU_TESTS=ON ..
```

The QES-Winds sanity tests are: - GPU-FlatTerrain: basic empty domain test - GPU-GaussianHill: basic terrain test - GPU-OklahomaCity: coarse resolution shapefile reader (without parameterization) - GPU-MultiSensors: test of multiple sensor and multiple timesteps - GPU-SaltLakeCity: test of high resolution urban setup with parameterizations - GPU-RxCADRE: test of high resolution and complex terrain (DEM)

3.3.2 QES-Turb Tests

There currently is no automated test available for QES-Turb.

3.3.3 QES-Plume Tests

There currently is no automated test available for QES-Plume. The following test cases are available - testing well-mixed condition: Sinusoidal3D Channel3D BaileyLES - testing against analytical solution: UniformFlow_ContRelease PowerLawBLFlow_ContRelease - testing against wind-tunnel data: EPA_7x11array

3.3.4 Unit Tests

Unit tests can be enable by settong the flag `ENABLE_UNITTESTS` to ON.

```
cmake -DENABLE_UNITTESTS=ON ..
```


QES-WINDS

4.1 Introduction

A new dispersion modeling system based on the well-used FORTRAN-based QUIC (Quick Urban and Industrial Complex) dispersion modeling system originally developed by the University of Utah and Los Alamos National Laboratory [1], has been under development as collaboration between the University of Utah and the University of Minnesota, Duluth. Quick Environmental Simulation (QES) is a microclimate simulation platform for computing 3D environmental scalars in urban areas and over complex terrain. Figure [fig:QES] shows a schematic of QES system and how different elements of the system interact with each other.

The fast-response three-dimensional diagnostic wind model written in C++, QES-Winds, is a rapid mass conserving wind-field solver. QES-Winds utilizes the concept of dynamic parallelism in CUDA to substantially accelerate wind simulations. Figure [fig:Winds] shows a high-level flowchart for QES-Winds code.

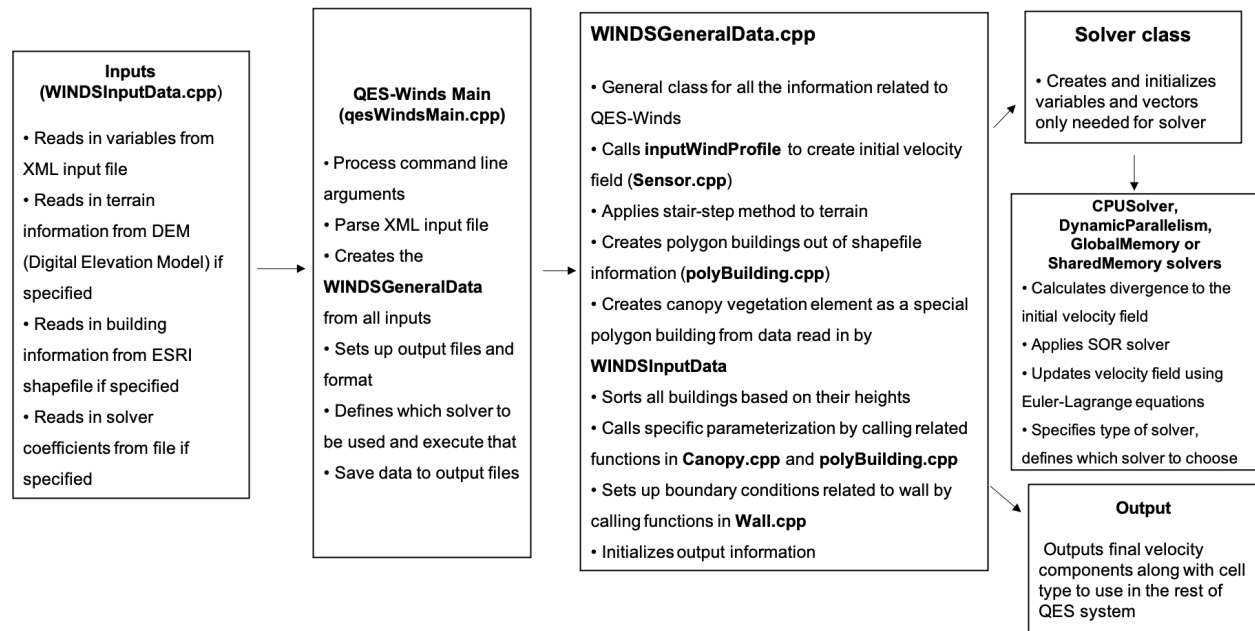


Fig. 1: Flowchart for the QES-Winds wind solver

4.2 QES-Winds Domain

The first step in every computational code or package is to define the computational domain. The user can define the domain by specifying the number of cells in x , y and z directions as well as the cell size in each direction in the input file (XML file).

4.2.1 XML Example

The domain information (number of cells and cell size) are defined under the `<simulationParameters>` part of the XML file. Following is an example of a domain with 2 km by 2 km by 200 m and resolution of 2 m by 2 m by 2 m:

```
<simulationParameters>
  <domain> 1000 1000 100 </domain>           <!-- Number of cells in x,y and
↪z directions-->
  <cellSize> 2.0 2.0 2.0 </cellSize>         <!-- Mesh resolution (meters)-->
</simulationParameters>
```

4.3 Staggered Grid

QES-Winds discretizes the computational domain using a staggered grid as shown in Figure [\[fig:staggered_grid\]](#). The velocity components (u , v and w) are face-centered values and Lagrange multipliers (λ), divergence of the initial wind field (R) and solver coefficients (e , f , g , h , m and n) are cell-centered variables. Because of nature of the finite difference method (depending on neighboring cells values), the first and last cells in x and y directions and the last cell in z direction, are not updated in the solving process (their velocity remains as the same as the initial velocity field). For the same reason, there is a layer of ghost cells under the bottom surface to make velocity calculation in the first layer above the bottom surface possible. The values of the Lagrange multipliers for the ghost cells are set to the ones for the layer above the bottom surface to create a zero gradient for the Lagrange multipliers (boundary condition) as well as providing the neighboring cell for the finite difference method.

4.3.1 Halo Region

If a solid element (building or terrain) overlaps with the QES domain boundaries, QES-Winds cannot model the wind field around the element correctly. In order to prevent this phenomenon, the user can add buffer zones to the sides of the domain when a terrain file or an ESRI shapefile is read into the code. Figure [\[fig:halo\]](#) represents how the halo region is added to the domain around a Digital Elevation Model (DEM) or a shapefile.

In order to define length of the halo zone in x and y direction, the user can use `<halox>` and `<haloy>` under `<simulationParameters>`. When the halo zone is defined, the length of the domain ($nx * dx$) and ($ny * dy$), should be greater than or equal to length of the DEM or shapefile in each direction plus twice the length of the halo in x and y directions, respectively.

```
<simulationParameters>
  <halo_x> 20.0 </halo_x>           <!-- Halo region added to x-direction of
↪domain (at the beginning and the end of domain) (meters)-->
  <halo_y> 30.0 </halo_y>           <!-- Halo region added to y-direction of
↪domain (at the beginning and the end of domain) (meters)-->
</simulationParameters>
```

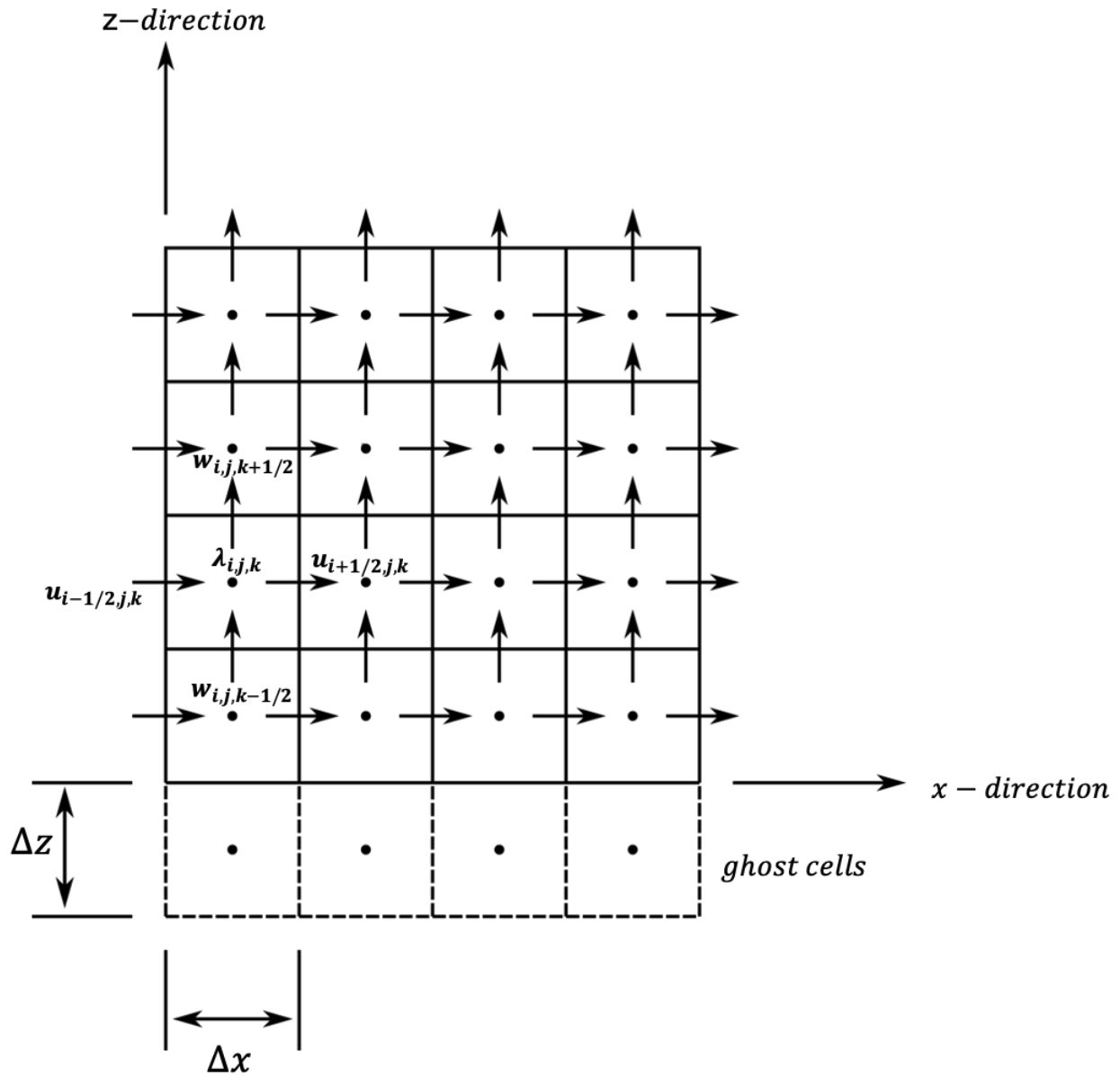


Fig. 2: Staggered grid representation of the domain and location of each variable.

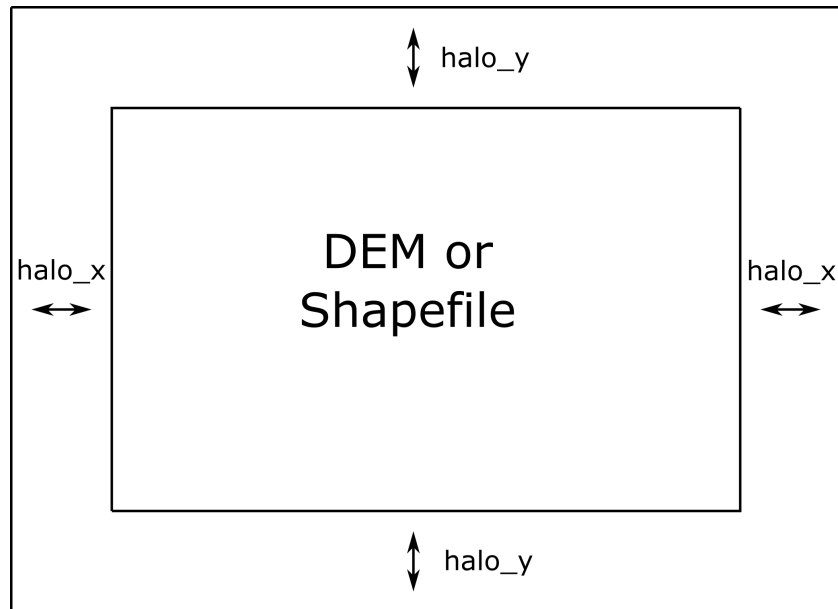


Fig. 3: Representation of halo region around the domain.

4.4 Digital Elevation Model (DEM) and ESRI Shapefile

The current version of QES-Winds has been written to allow commonly available terrain and building geometry datasets to be used for simulations. In this section, various input file formats for QES-Winds will be covered.

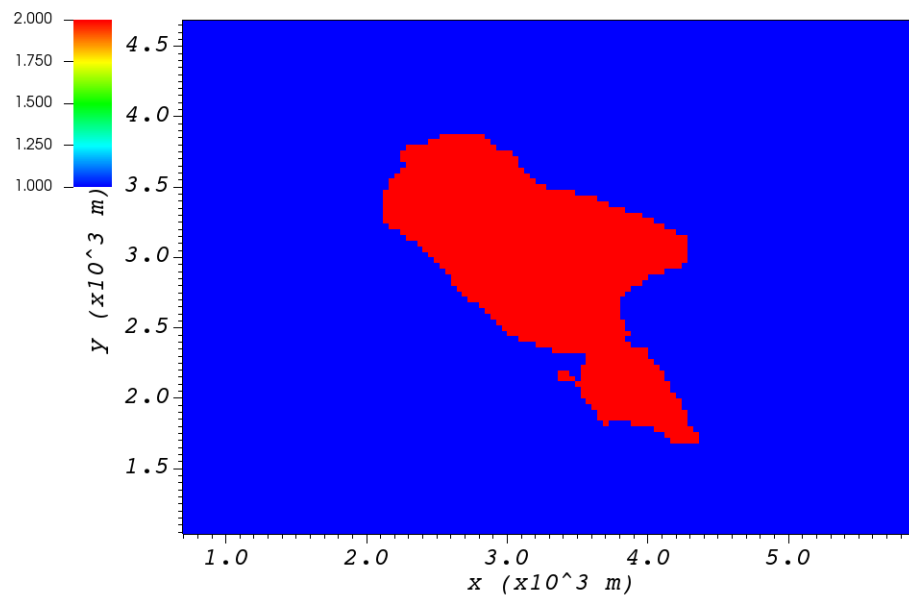
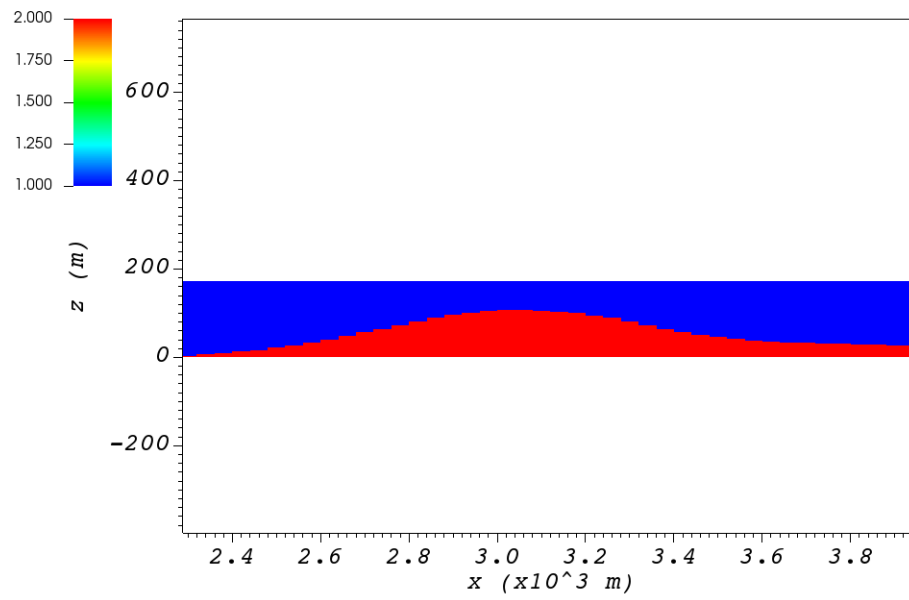
4.4.1 Terrain Features

Using the Geospatial Data Abstraction Library (GDAL; <https://www.gdal.org>), we are able to load geo-referenced datasets of terrain so that the simulations can include the effects of hills, valleys, and mountains. In the current version of the code, we can load Digital Elevation Model (DEM) files for different physical locations.

Using the Digital Elevation Model (DEM) file loaders in our code base, we have loaded and tested multiple different terrain data sets. As a first test, we loaded a DEM of Askervein Hill. This is an isolated hill in Scotland where field experiments have been conducted and data for testing and evaluation exists ([3, 4]). The simulation with Askervein Hill was run without any complex terrain flow parameterizations. The Askervein Hill dataset is 6023.43 m by 6023.43 m. The hill height is approximately 124 m tall. Figure [fig:askervein] indicates the cell type contour for the Askervein hill test case in a vertical plane at $y = 3000$ m (part (a)), and a horizontal plane at $z = 20$ m (part (b)). These plots show the ability of QES-Winds to read in and process DEM files. The cell type value 1 (blue) represents the air cells while value 2 (red) indicates the terrain cells.

The user can define the address to the DEM using <DEM> variable under the <simulationParameters> part in the XML file:

```
<simulationParameters>
  <DEM>../scratch/DEM/askervein.tif</DEM>          <!-- Address to DEM location-->
</simulationParameters>
```



Process Part of DEM

In some cases, user wants to load a giant DEM but only process part of the file. This is possible in QES-Winds by defining the origin of QES domain inside the DEM borders and the size of the QES domain. Figure [fig:DEM_cut] shows a schematic of how the QES domain can be defined inside a DEM file and only process that part.

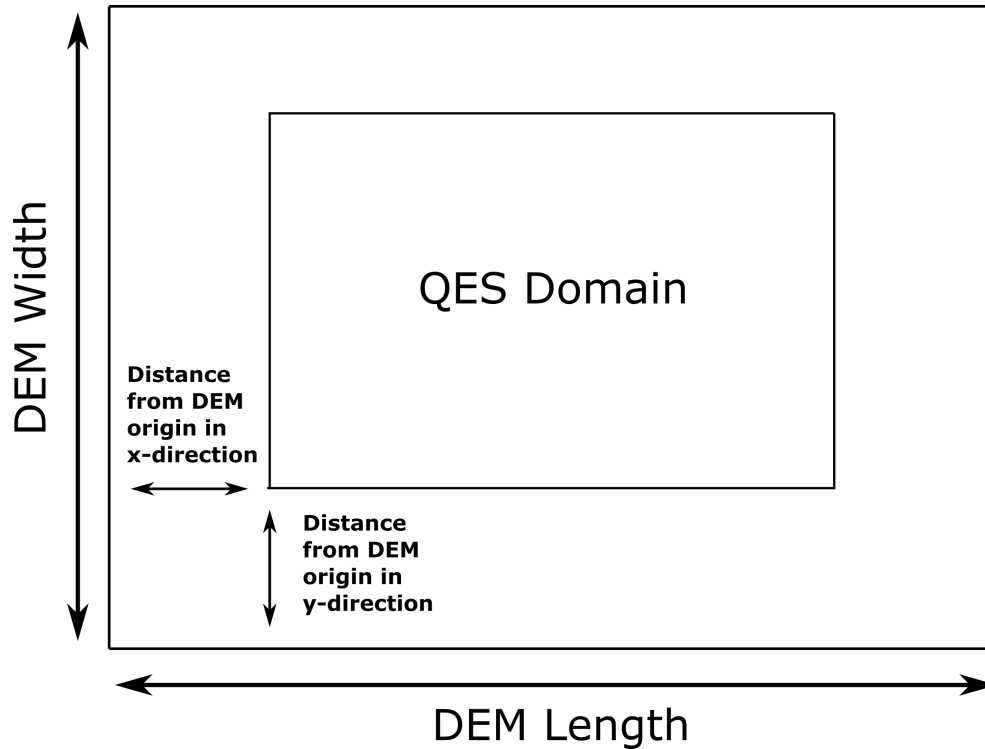


Fig. 4: Schematic of how the QES domain can be defined inside a DEM file and only process that part.

There are two options to determine the location of the origin of QES domain inside the DEM borders:

1. Specifying the distance of the QES origin with respect to bottom left corner of the DEM file. This can be done by setting the value of `<originFlag>` to 0 and defining distances (in meters) in *x* and *y* directions using `<DEMDistancex>` and `<DEMDistancey>`, respectively.

```
<simulationParameters>
  <originFlag> 0 </originFlag>                                <!-- Origin flag (0- DEM_
  ↪coordinates (default), 1- UTM coordinates) -->
    <DEMDistancex> 1000.0 </DEMDistancex>                      <!-- x component (m) of_
  ↪origin in DEM coordinates (if originFlag = 0) -->
    <DEMDistancey> 1000.0 </DEMDistancey>                      <!-- y component (m) of_
  ↪origin in DEM coordinates (if originFlag = 0) -->
</simulationParameters>
```

2. Defining the location of the QES domain origin in the Universal Transverse Mercator (UTM) coordinates by setting the value of `<originFlag>` to 1 and determining `<UTMx>` and `<UTMy>` of the origin in *x* and *y* directions, respectively.

```
<simulationParameters>
  <originFlag> 1 </originFlag>                                <!-- Origin flag (0- DEM_
  ↪coordinates (default), 1- UTM coordinates) -->
```

(continues on next page)

(continued from previous page)

```

<UTMx> 595469.6122881 </UTMx>          <!-- x component (m) of_
↪origin in UTM DEM coordinates (if originFlag = 1)-->
<UTMy> 6336281.9538635 </UTMy>        <!-- y component (m) of_
↪origin in UTM DEM coordinates (if originFlag = 1)-->
</simulationParameters>

```

4.4.2 Automated City Building

A new shapefile reader function has been added to QES-Winds, which provides the capacity to load the ESRI shapefiles using GDAL (Geospatial Data Abstraction Library) libraries. After the building footprints and heights are loaded from ESRI shapefiles, QES-Winds creates polygon buildings and applies appropriate parameterization to them. Figure [fig:okc_qgis] shows an example ESRI shapefile can be read into QES-Winds, Central Business District (CBD) of Oklahoma City shapefile, subject to JU2003 experimental campaign [5], plotted using the freely available software QGIS (<https://qgis.org>). The cell type contour for the Oklahoma City test case in a horizontal plane at $z = 3$ m is shown in Figure [fig:okc_icell]. This plot indicates the ability of QES-Winds to read in and process ESRI shapefiles. The cell type value 0 (blue) represents the building cells while value 1 (red) indicates the air cells.



Fig. 5: Central Business District (CBD) of Oklahoma City shapefile, subject to JU2003 experimental campaign [5], plotted using the freely available software QGIS.

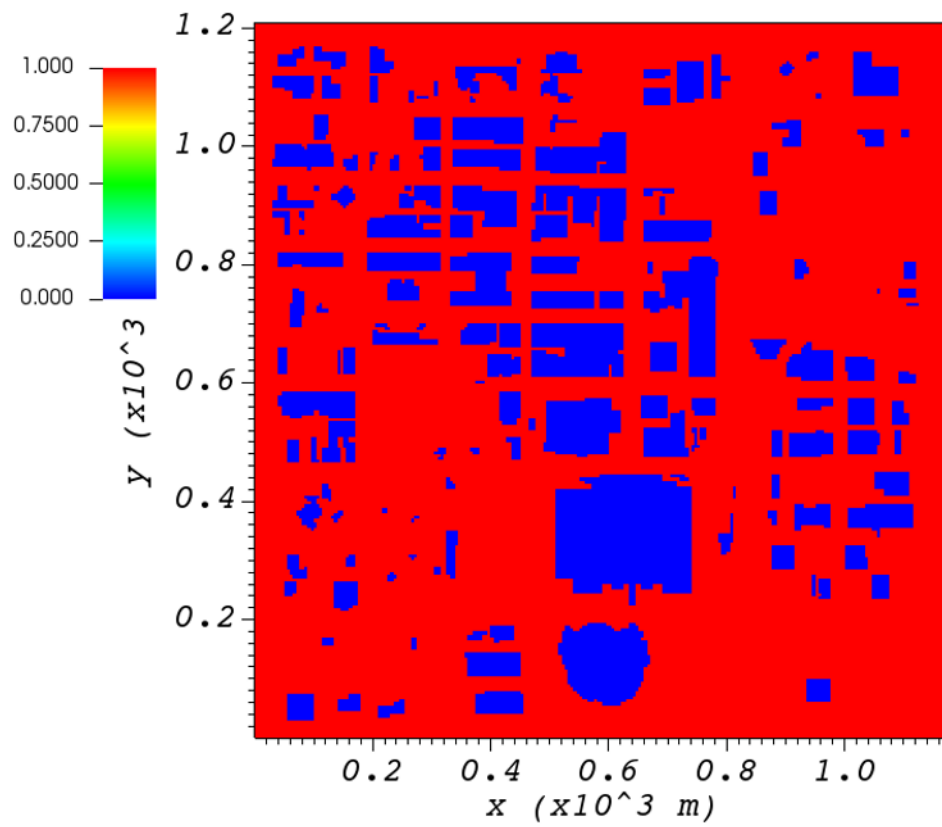


Fig. 6: Cell type contour for the Oklahoma City test case in a horizontal plane at $z = 3$ m. The cell type value 0 (blue) represents the building cells while value 1 (red) indicates the air cells.

The user can define the address to the shapefile using <SHP> variable as well as the name of the shapefile using the <SHPBuildingLayer> and the correlation factor between the height field of the shapefile and the actual height of the buildings using the <heightFactor> under <simulationParameters> part in the XML file:

```
<simulationParameters>
  <SHP>../data/GISFiles/OKCSmallDomain/OKCSmallDomainJU2003.shp</SHP> <!-- Address to
↳shapefile location-->
  <SHPBuildingLayer>OKCSmallDomainJU2003</SHPBuildingLayer>
  <heightFactor> 1.0 </heightFactor> <!-- Height factor multiplied by the
↳building height read in from the shapefile (default = 1.0)-->
</simulationParameters>
```

4.4.3 Import Building From XML

Instead of reading in a ESRI shapefile, the user can import building information manually through the XML file. This can be done by using the <buildings> section of the XML file. The only option available for now is the rectangular building. Information required for defining a rectangular building are height, base height, length, width, location of the closest corner to the origin of domain and building rotational angle. Following is an example of a rectangular building with 40 m as height, 0 m as base height, 20 m as length and width, closest corner to the origin located at 90 m in x and y directions, and 0° as rotation angle with respect to the North-South line. Also, 0.1 m is defined as the surface roughness for all the building walls.

```
<buildings>
  <wallRoughness> 0.1 </wallRoughness>
  <rectangularBuilding>
    <height> 40.0 </height>
    <baseHeight> 0 </baseHeight>
    <xStart> 90.0 </xStart>
    <yStart> 90.0 </yStart>
    <length> 20.0 </length>
    <width> 20.0 </width>
    <buildingRotation> 0.0 </buildingRotation>
  </rectangularBuilding>
</buildings>
```

4.5 Initial Wind Field

QES-Winds can read a single or multiple sensors for a specific test case. In this context, sensor means the velocity magnitude and direction at a single point or a single velocity profile to initialize the wind field. If there is only the wind velocity and direction at a single point, the user should specify what type of velocity profile they want to build from the measurement. There are three options available for the type of profile:

1. a logarithmic profile [6]:

$$u_{log}(z) = u_{ref} \cdot \frac{\ln(z/z_0)}{\ln(z_{ref}/z_0)}$$

2. a power law profile [6]:

$$u_{pow}(z) = u_{ref} \cdot (z/z_{ref})^{z_0}$$

3. an urban canopy profile [6, 7]:

$$u_{uc}(z) = \begin{cases} u(H) \cdot \exp(\alpha(\frac{z}{H} - 1)) & \text{if } z \leq H \\ u(H) \cdot \exp(\alpha(\frac{z}{H} - 1)) & \text{if } z > H. \end{cases}$$

where u_{ref} is the measured velocity at measured height z_{ref} , z_0 is the surface roughness. The lower portion of the urban canopy profile calculated where α is a factor that depends on canopy element density (attenuation coefficient) and $u(H)$ is the computed velocity at height H . The upper portion of the urban canopy is a different form of a logarithmic profile where u_* is the friction velocity, κ is the von Karman constant at 0.4 and d is the zero plane displacement.

If there is only one sensor available in the computational domain, the code will extend the profile for that sensor uniformly to the whole domain. On the occasion of multiple sensors, QES-Winds utilizes a two-dimensional Barnes interpolation scheme [8, 9] to interpolate velocity components at each cell height of the domain based on the weighted distance from each sensor.

4.5.1 XML Setup

There are two options available for defining sensor information:

1. The user can put all the sensor information in a separate XML file and define the address to the location of the sensor file using the `<sensorName>` variable.

```
<metParams>
  <z0_domain_flag> 0 </z0_domain_flag>           <!-- Distribution of_
↪ surface roughness for domain (0-uniform (default), 1-custom -->
  <sensorName>../data/InputFiles/sensor.xml</sensorName> <!-- Name of the sensor_
↪ file with information for the sensor included -->
</metParams>
```

2. The user can define all information required for creating a sensor by using the `<sensor>` variable inside the `<metParams>` section of the XML file.

The first part of the sensor information is the location of the sensor in domain. There are three options for it: 1) define the location in local coordinates of the QES domain.

```
<metParams>
  <sensor>
    <site_coord_flag> 1 </site_coord_flag>         <!-- Sensor site coordinate_
↪ system (1=QES (default), 2=UTM, 3=Lat/Lon) -->
    <site_xcoord> 1.0 </site_xcoord>               <!-- x component of site_
↪ location in QES domain (m) (if site_coord_flag = 1) -->
    <site_ycoord> 1.0 </site_ycoord>               <!-- y component of site_
↪ location in QES domain (m) (if site_coord_flag = 1)-->
  </sensor>
</metParams>
```

3. The user can define the location in the Universal Transverse Mercator (UTM) coordinates. In this case, user also needs to define the origin of computational domain in the UTM coordinates.

```
<simulationParameters>
  <UTMx> 634173 </UTMx>                           <!-- x component (m) of origin in_
↪ UTM -->
  <UTMy> 3925360 </UTMy>                           <!-- y component (m) of origin in_
```

(continues on next page)

(continued from previous page)

```

↪ UTM -->
    <UTMZone> 14 </UTMZone>                                <!-- UTM zone that domain_
↪ located -->
</simulationParameters>

```

```

<metParams>
  <sensor>
    <site_coord_flag> 2 </site_coord_flag>                  <!-- Sensor site coordinate_
↪ system (1=QES (default), 2=UTM, 3=Lat/Lon) -->
    <site_UTM_x> 634175 </site_UTM_x>                        <!-- x components of site_
↪ coordinate in UTM (if site_coord_flag = 2) -->
    <site_UTM_y> 3925362 </site_UTM_y>                       <!-- y components of site_
↪ coordinate in UTM (if site_coord_flag = 2)-->
    <site_UTM_zone> 14 </site_UTM_zone>                      <!-- UTM zone of the sensor_
↪ site (if site_coord_flag = 2)-->
    </sensor>
  </metParams>

```

4. The user can define the location in Latitude and Longitude coordinates. In this case, user also needs to define the origin of computational domain in the UTM coordinates.

```

<simulationParameters>
  <UTMx> 634173 </UTMx>                                     <!-- x component (m) of origin in_
↪ UTM -->
  <UTMy> 3925360 </UTMy>                                    <!-- y component (m) of origin in_
↪ UTM -->
  <UTMZone> 14 </UTMZone>                                    <!-- UTM zone that domain_
↪ located -->
</simulationParameters>

```

```

<metParams>
  <sensor>
    <site_coord_flag> 3 </site_coord_flag>                  <!-- Sensor site coordinate_
↪ system (1=QES (default), 2=UTM, 3=Lat/Lon) -->
    <site_lat> 35.46270 </site_lat>                          <!-- x components of site_
↪ coordinate in Latitude (if site_coord_flag = 3) -->
    <site_lat> -97.52130 </site_lat>                         <!-- y components of site_
↪ coordinate in Longitude (if site_coord_flag = 3)-->
    </sensor>
  </metParams>

```

The second part of sensor definition is choosing type of profile for different time steps, if applicable. The <timeSeries> variable is designed to define type of sensor profile in the sensor section for several time steps. There are four options for the input profile in QES-Winds:

1. Logarithmic velocity profile, based on Eq. [eq:log_law]:

```

<metParams>
  <sensor>
    <timeSeries>                                              <!-- Start of timestep information for_
↪ a sensor -->
    <boundaryLayerFlag> 1 </boundaryLayerFlag>               <!-- Site boundary_

```

(continues on next page)

(continued from previous page)

```

→ layer flag (1-log (default), 2-exp, 3-urban canopy, 4-data entry) -->
    <siteZ0> 0.1 </siteZ0>                                <!-- Site z0 -->
    <reciprocal> 0.0 </reciprocal>                          <!-- Reciprocal Monin-
→ Obukhov Length (1/m) -->
    <height> 20.0 </height>                                <!-- Height of the sensor -->
    <speed> 5.0 </speed>                                    <!-- Measured speed at the
→ sensor height -->
    <direction> 270.0 </direction>                          <!-- Wind direction of
→ sensor -->
    </timeSeries>
  </sensor>
</metParams>

```

Figure [fig:log_profile] shows velocity magnitude contour with overlaying velocity vectors of initial velocity field created by the aforementioned example of the logarithmic profile.

2. Exponential (power law) velocity profile, based on Eq. [eq:power_law]:

```

<metParams>
  <sensor>
    <timeSeries>                                           <!-- Start of timestep information for
→ a sensor -->
    <boundaryLayerFlag> 2 </boundaryLayerFlag>            <!-- Site boundary
→ layer flag (1-log (default), 2-exp, 3-urban canopy, 4-data entry) -->
    <siteZ0> 0.1 </siteZ0>                                <!-- Site z0 -->
    <reciprocal> 0.0 </reciprocal>                          <!-- Reciprocal Monin-
→ Obukhov Length (1/m) -->
    <height> 20.0 </height>                                <!-- Height of the sensor -->
    <speed> 5.0 </speed>                                    <!-- Measured speed at the
→ sensor height -->
    <direction> 270.0 </direction>                          <!-- Wind direction of
→ sensor -->
    </timeSeries>
  </sensor>
</metParams>

```

Figure [fig:exp] shows velocity magnitude contour with overlaying velocity vectors of the initial velocity field created by the aforementioned example of the exponential (power law) profile.

3. Urban canopy velocity profile, based on Eq. [eq:urban_canopy_low] and [eq:urban_canopy_up]:

```

<metParams>
  <sensor>
    <timeSeries>                                           <!-- Start of timestep information for
→ a sensor -->
    <boundaryLayerFlag> 3 </boundaryLayerFlag>            <!-- Site boundary
→ layer flag (1-log (default), 2-exp, 3-urban canopy, 4-data entry) -->
    <siteZ0> 0.1 </siteZ0>                                <!-- Site z0 -->
    <reciprocal> 0.0 </reciprocal>                          <!-- Reciprocal Monin-
→ Obukhov Length (1/m) -->
    <height> 20.0 </height>                                <!-- Height of the sensor -->
    <speed> 5.0 </speed>                                    <!-- Measured speed at the
→ sensor height -->

```

(continues on next page)

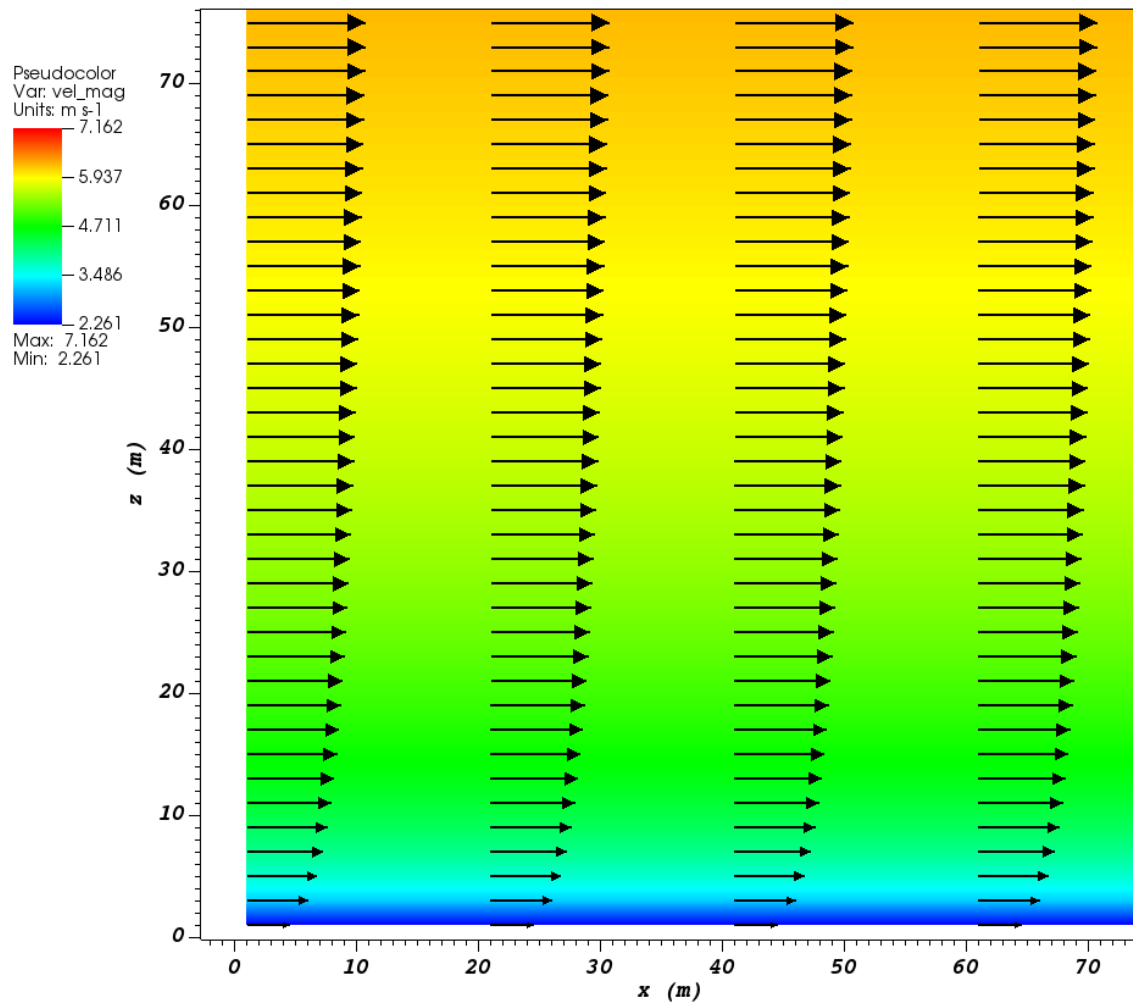


Fig. 7: Velocity magnitude contour with overlaying velocity vectors in a vertical plane at $y = 101$ m for initial velocity field created by the logarithmic profile.

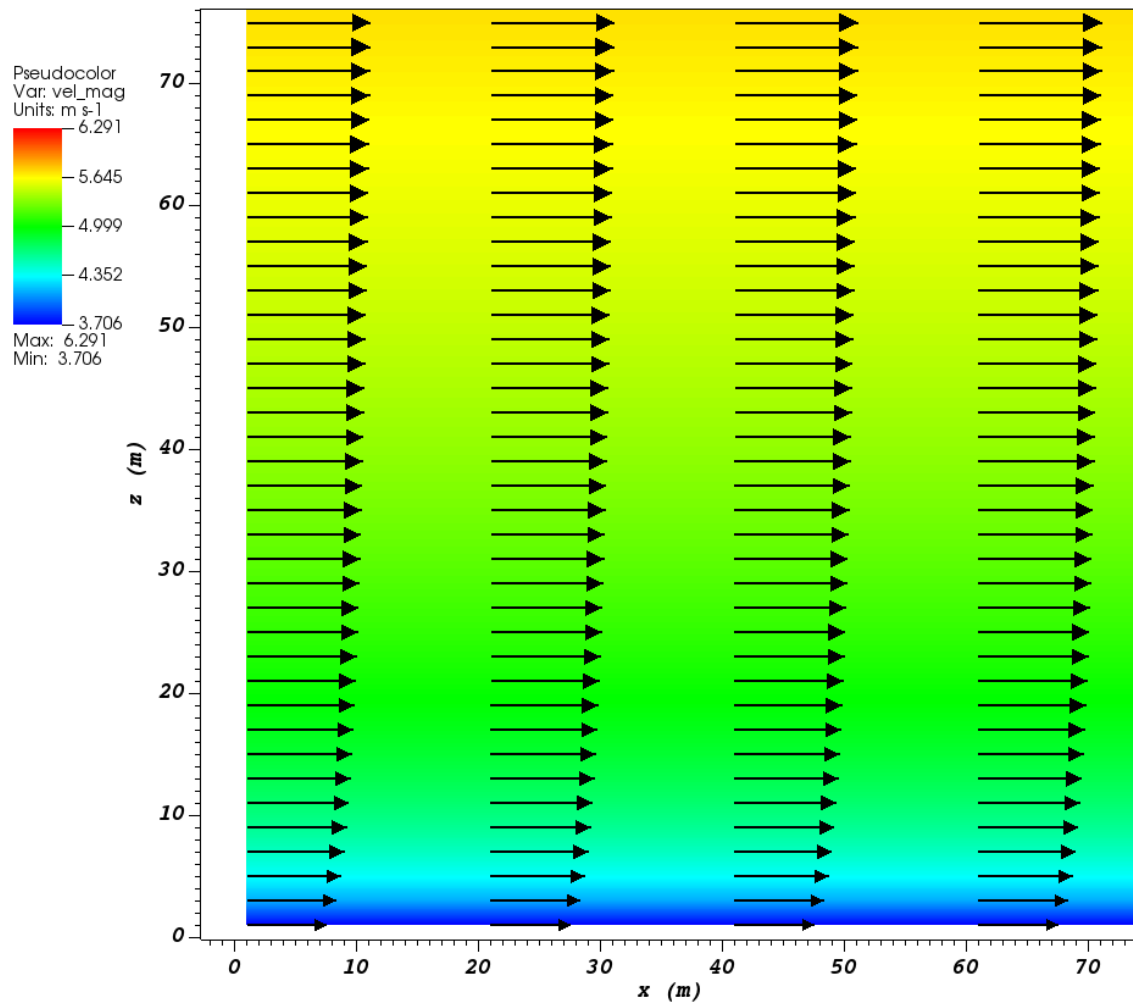


Fig. 8: Velocity magnitude contour with overlaying velocity vectors in a vertical plane at $y = 101$ m for initial velocity field created by the exponential (power law) profile.

(continued from previous page)

```

<direction> 270.0 </direction>                                <!-- Wind direction of u
->sensor -->
    <canopyHeight> 10.0 </canopyHeight>
    <attenuationCoefficient> 1.0 </attenuationCoefficient>
  </timeSeries>
</sensor>
</metParams>

```

Figure [fig:canopy] shows velocity magnitude contour with overlaying velocity vectors of the initial velocity field created by the aforementioned example of the urban canopy profile.

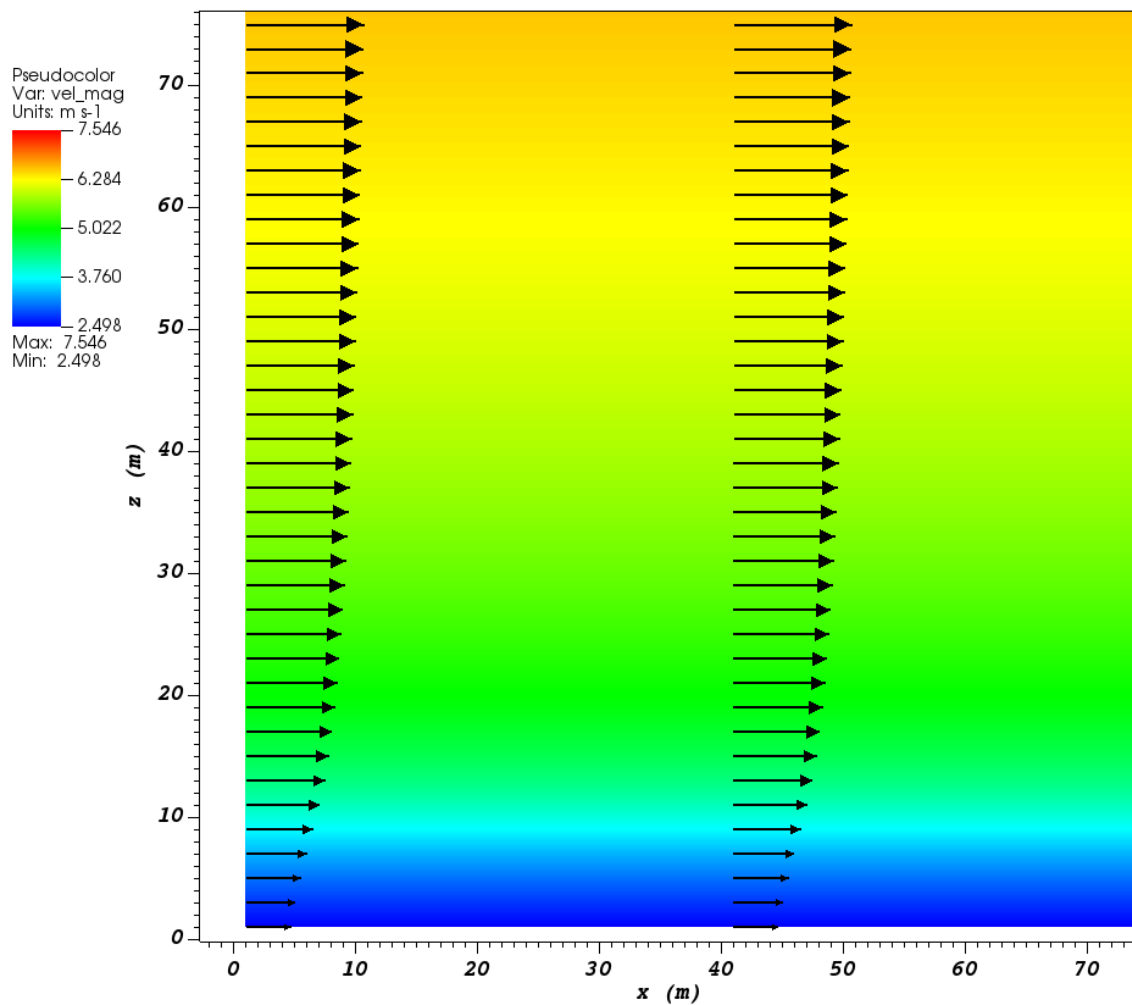


Fig. 9: Velocity magnitude contour with overlaying velocity vectors in a vertical plane at $y = 101$ m for initial velocity field created by the urban canopy profile.

4. Data entry of the profile from an experimental tower with multiple sensors or from a numerical mesoscale weather

prediction model like WRF [10]:

```

<metParams>
  <sensor>
    <timeSeries>                                <!-- Start of timestep informastion for_
    ↪ a sensor -->
      <boundaryLayerFlag> 4 </boundaryLayerFlag>          <!-- Site boundary_
    ↪ layer flag (1-log, 2-exp, 3-urban canopy, 4-data entry) -->
      <siteZ0> 0.1 </siteZ0>                                <!-- Site z0 -->
      <reciprocal> 0.0 </reciprocal>                        <!-- Reciprocal_
    ↪ Monin-Obukhov Length (1/m) -->
      <height> 30.7015 </height>                            <!-- Height of the_
    ↪ sensor -->
      <height> 74.4169 </height>
      <height> 144.644 </height>
      <height> 197.455 </height>
      <height> 268.468 </height>
      <speed> 2.56922 </speed>                                <!-- Measured speed at_
    ↪ the sensor height -->
      <speed> 2.55532 </speed>
      <speed> 2.33319 </speed>
      <speed> 2.16058 </speed>
      <speed> 1.98843 </speed>
      <direction> 323.283 </direction>                        <!-- Wind direction of_
    ↪ sensor -->
      <direction> 327.377 </direction>
      <direction> 332.676 </direction>
      <direction> 337.649 </direction>
      <direction> 344.273 </direction>
    </timeSeries>
  </sensor>
</metParams>

```

4.6 Empirical Parameterizations

QES-Winds only conserves mass and no momentum equation is solved. As a result, the solution is a potential-flow solution (no shear effects). In order to add shear effects to our solution, empirical parameterizations are needed. These parameterizations are designed using results of experiments and computational simulations (e.g. [1, 11]). Buildings are the most important elements in urban areas. There are several parameterizations developed for different areas around the building. This section covers available parameterizations in QES-Winds along with their effects on the wind field.

4.6.1 Upwind Cavity

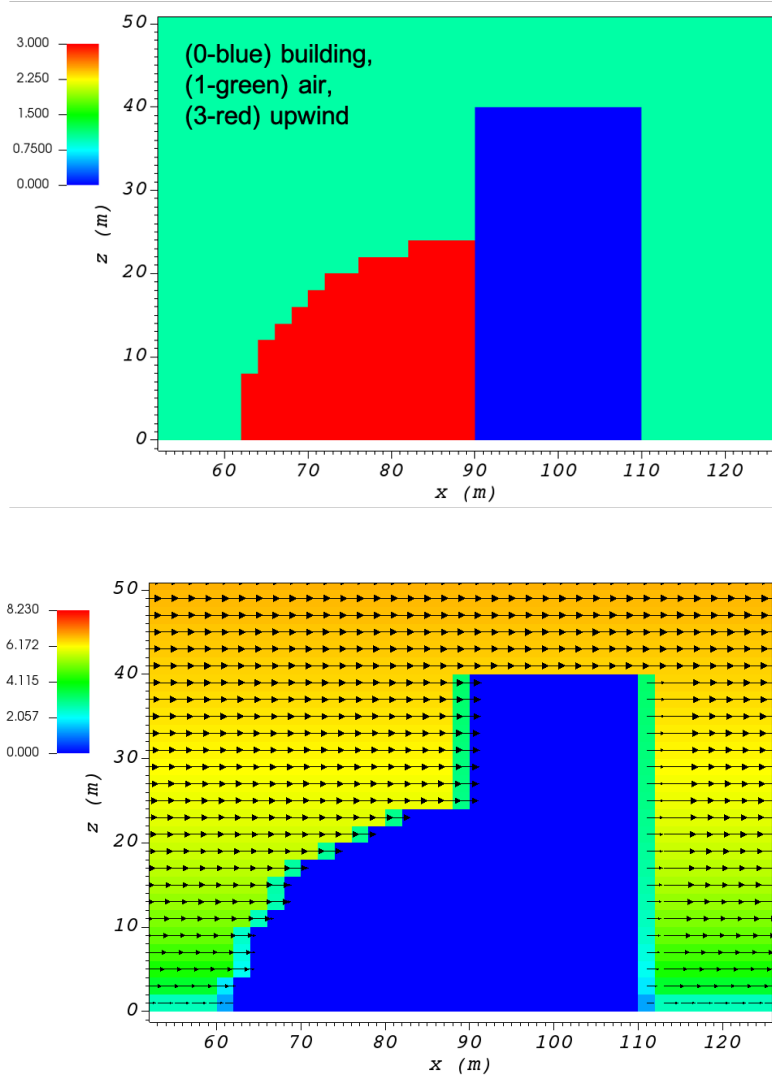
Upwind cavity as described in [12, 13, 14] is the parameterization representing upwind and stagnation effects of the building on the fluid flow. There are three options available for this type of parameterization in QES-Winds. The first option based on the parameterization proposed by Röckle [15] and later Kaplan and Dinar [16]. They defined an ellipsoid to represent what they call is the displacement zone in front of the building. The length of the displacement zone, L_F , is defined by Eq. [eq:lf]. The shape of the ellipsoid is estimated by Eq. [eq:upwind]. Finally, the initial velocity components in the displacement zone are set to zero.

$$\frac{L_F}{H} = \frac{2(W/H)}{1 + 0.8W/H}$$

$$\frac{X^2}{L_F^2 (1 - (Z/0.6H)^2)} + \frac{Y^2}{W^2} = 1$$

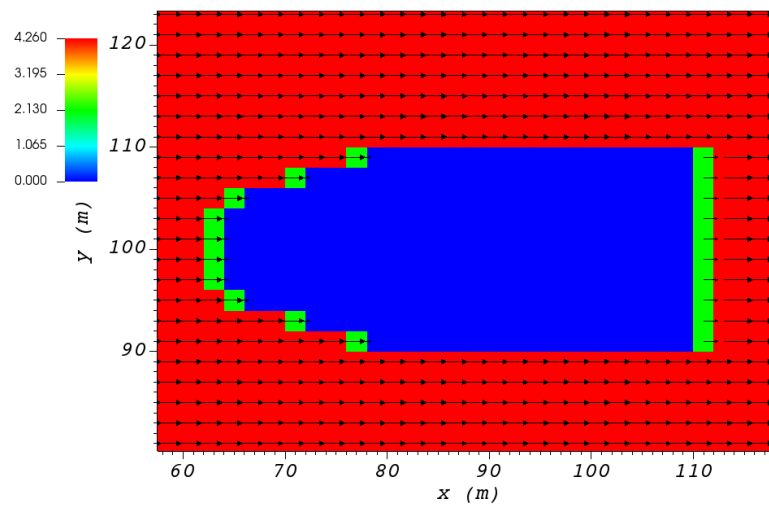
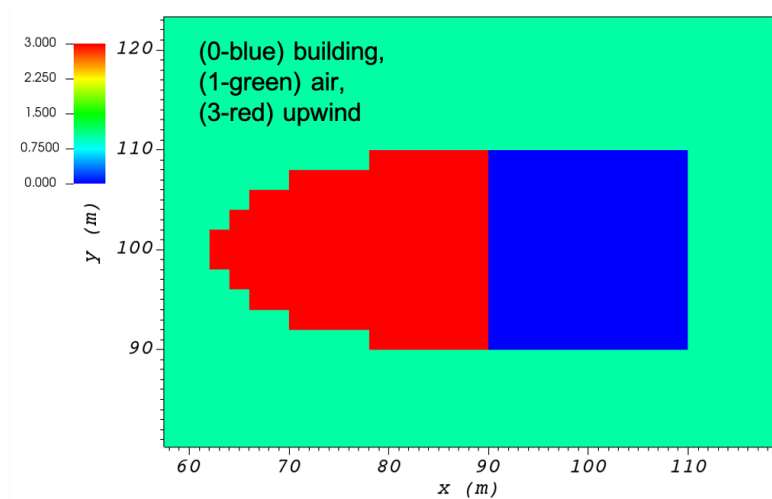
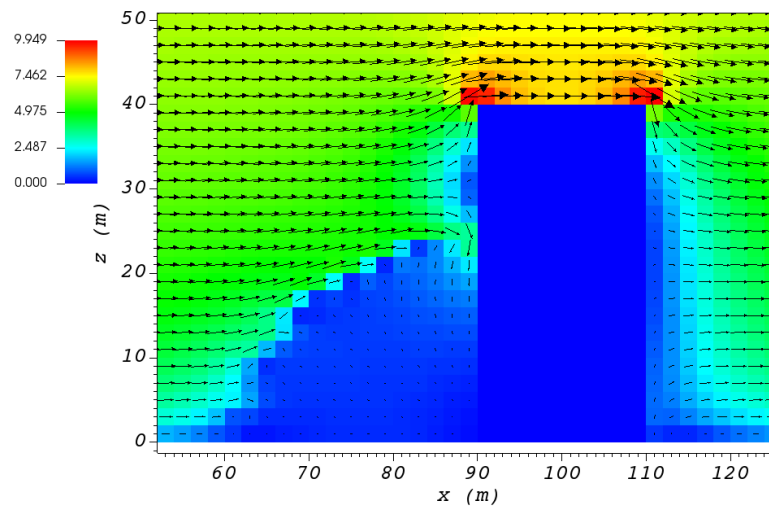
where L , H and W are length, width and height of the building, receptively.

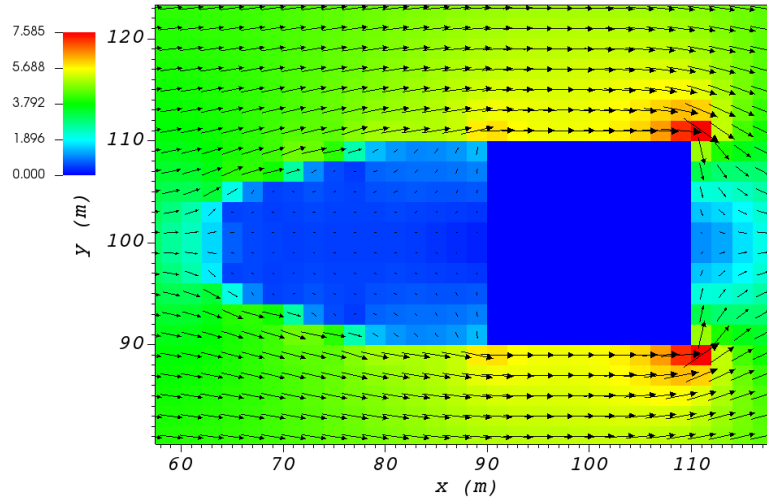
Part (a) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] show cell type contour to represent the area of effect of the Röckle upwind cavity parameterization in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively. The upwind parameterizations is applied to a rectangular building defined in Section 1.4.3. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part(c)) velocity fields in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively.



The second option is called the Modified Vortex Parameterization (MVP) and created by Bagal et al. [13]. In this parameterization, the length of the displacement zone, L_F , is calculated by Eq. [eq:lf_MVP]. The MVP parameterization defines two ellipsoids instead of one: In the outer ellipsoid, velocities are reduced to 40% of their initial values while in the inner region, velocity components are set to zero [12]. Both ellipsoids are extended to 0.6 of the building height.

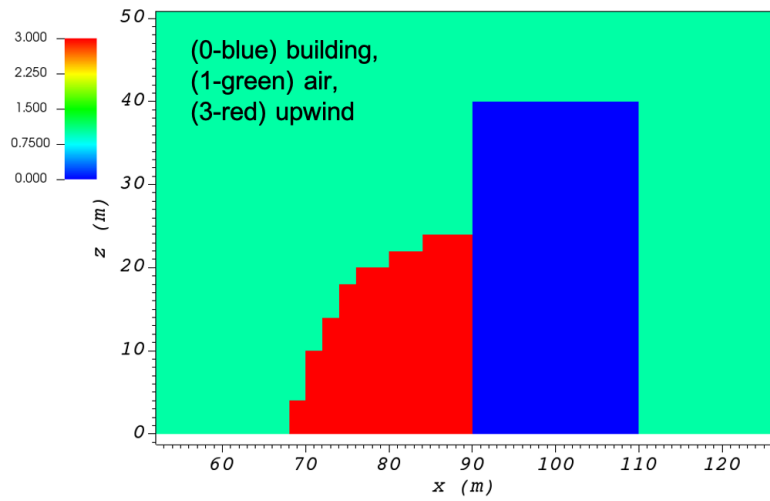
$$\frac{L_F}{H} = \frac{1.5(W/H)}{1 + 0.8W/H}$$





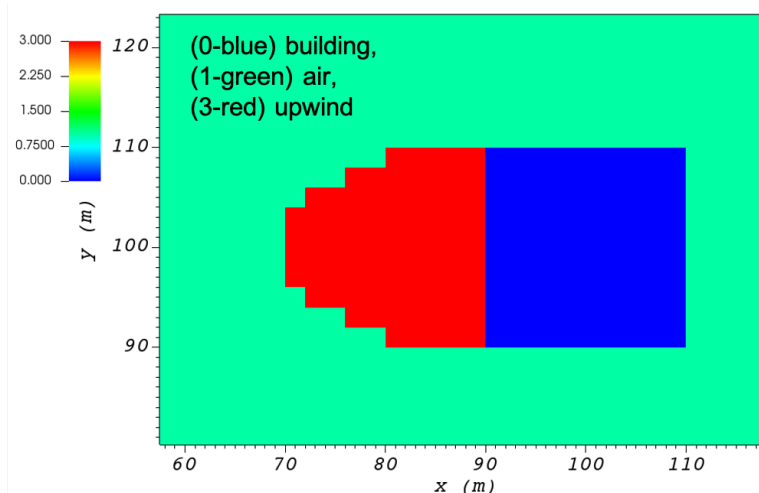
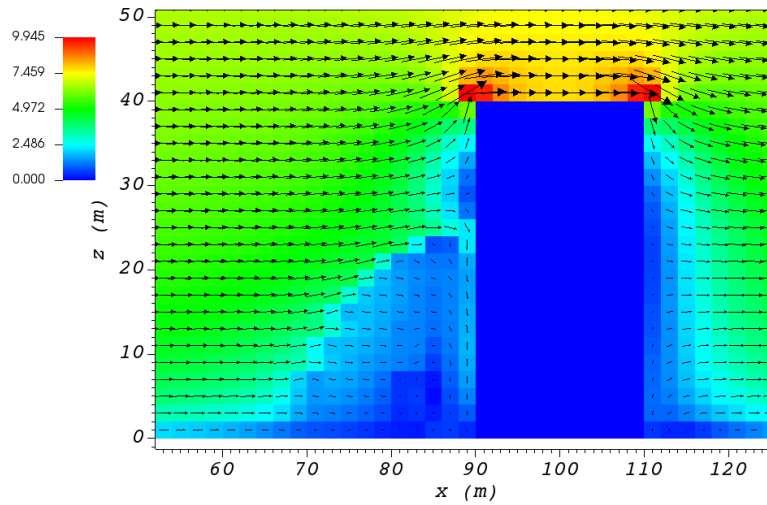
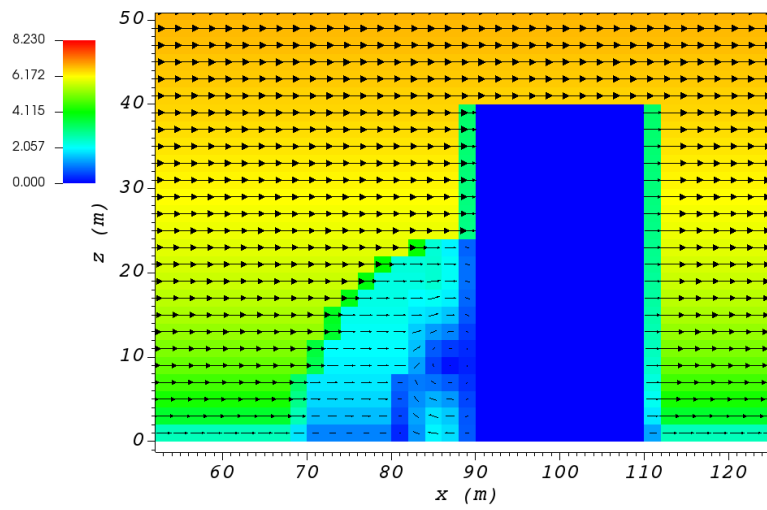
where L , H and W are length, width and height of the building, receptively.

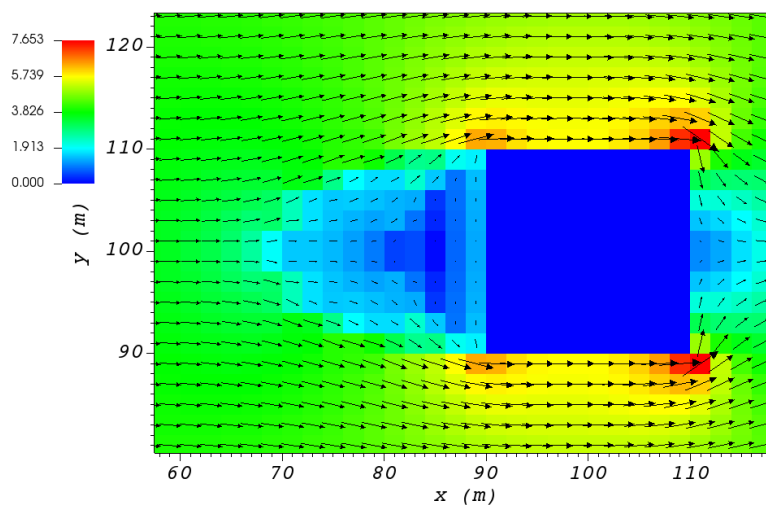
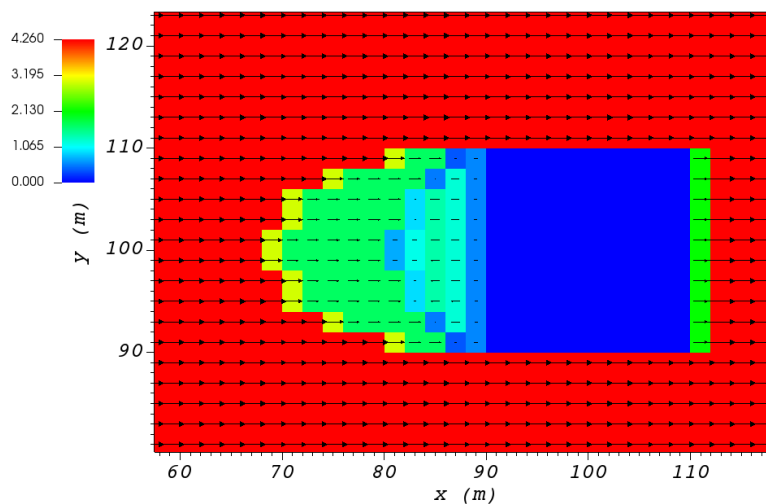
Part (a) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] show cell type contour to represent the area of effect of the MVP upwind cavity parameterization in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively. The upwind parameterizations is applied to a rectangular building defined in Section 1.4.3. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part (c)) velocity fields in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively.



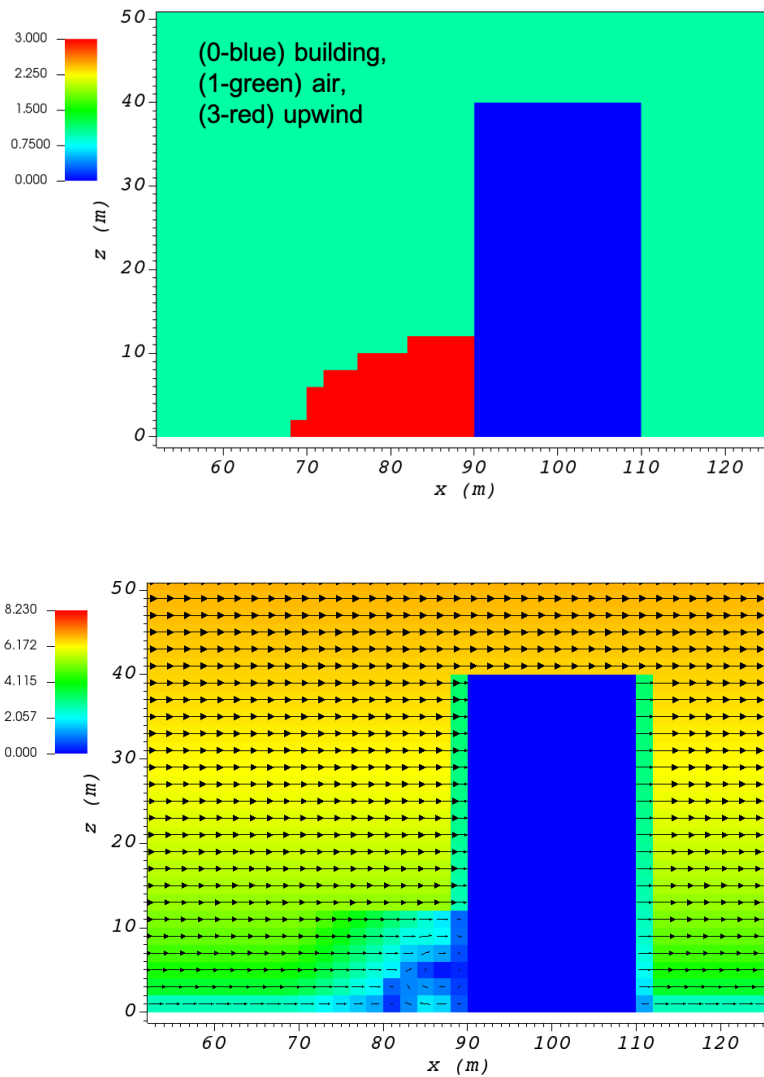
The third option is called the high-rise MVP algorithm (HMVP) and is designed to address the shortcomings of the previous models when it comes to tall buildings [12]. The length of the displacement zone is calculated the same as Eq. [eq:lf_MVP]. The HMVP algorithm creates two ellipsoids with the difference that the inner region only extends to 60% of the minimum of building height and building width. In addition, the algorithm linearly reduces the velocities in the outer region from their upwind values at the outer surface to 40% of the initial values on the inner region.

Part (a) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] show cell type contour to represent the area of effect of the HMVP upwind cavity parameterization in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively. The upwind parameterization is applied to a rectangular building defined in Section 1.4.3. The



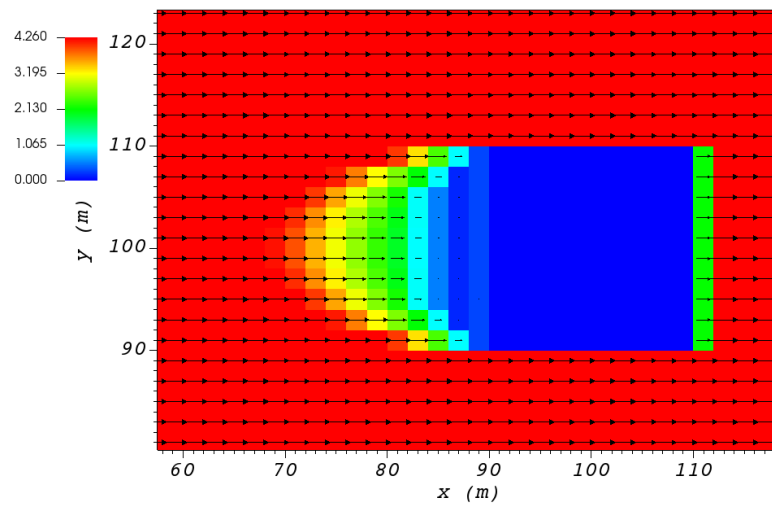
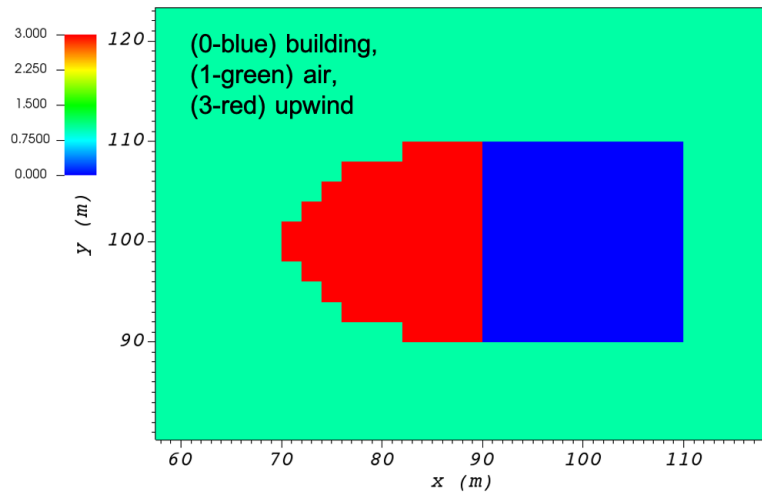
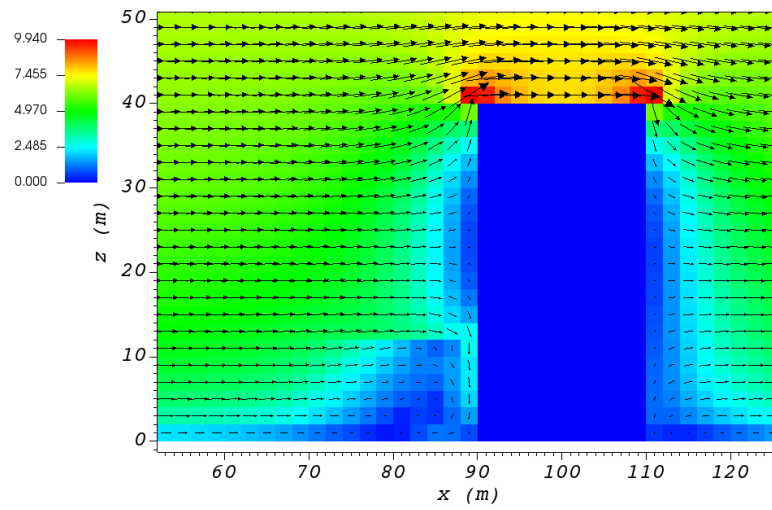


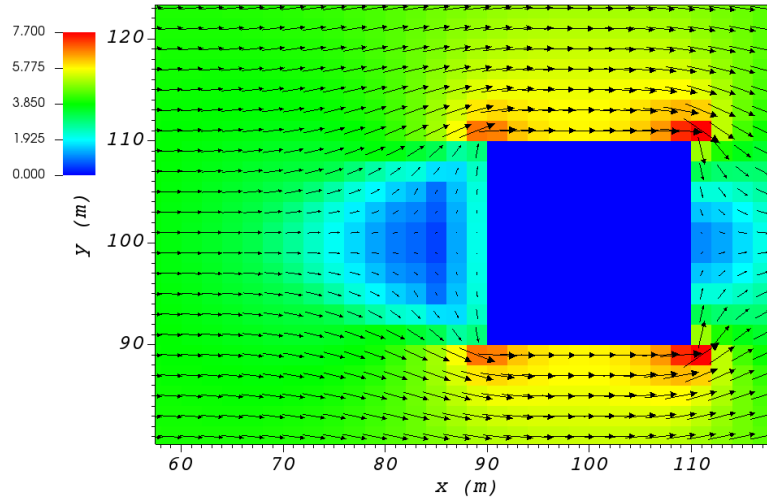
initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:upwind_1_vert] and Figure [fig:upwind_1_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part(c)) velocity fields in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively.



In order to choose between these three upwind models, the user needs to change the value of “upwindCavityFlag” in the XML file.

```
<simulationParameters>
  <upwindCavityFlag> 2 </upwindCavityFlag>           <!-- Upwind cavity flag (0-none, 1-Rockle, 2-MVP (default), 3-HMVP) -->
</simulationParameters>
```





4.6.2 Leaside Cavity and Far-Wake

The far-wake and cavity parameterization described in [17, 18] are a significant part of the building parameterizations. The one available in QES-Winds is based on the parameterization proposed by Röckle [15] and later Kaplan and Dinar [16]. The Röckle parameterization defines two ellipsoids to represent the shape of the reversed flow cavity and the far-wake region. The reversed flow cavity extends to the along-wind cavity length (L_R), which is calculated as Eq. [eq:Lr], and wake is assumed to be approximately 3 cavity lengths long (i.e., $3L_R$). After calculating L_R , the cavity length, d in the stream-wise direction was defined by an ellipsoid shape using Eq. [eq:d]. Finally, the velocity in the reversed cavity zone is defined using Eq. [eq:cavity] and in the wake region, the velocity field is estimated by Eq. [eq:wake].

$$\frac{L_R}{H} = \frac{1.8 \frac{W}{H}}{\left(\frac{L}{H}\right)^{0.3} \left(1 + 0.24 \frac{W}{H}\right)}$$

$$d = L_R \sqrt{\left(1 - \left(\frac{z}{H}\right)^2\right) \left(1 - \left(\frac{y}{W}\right)^2\right)} - \frac{L}{2}$$

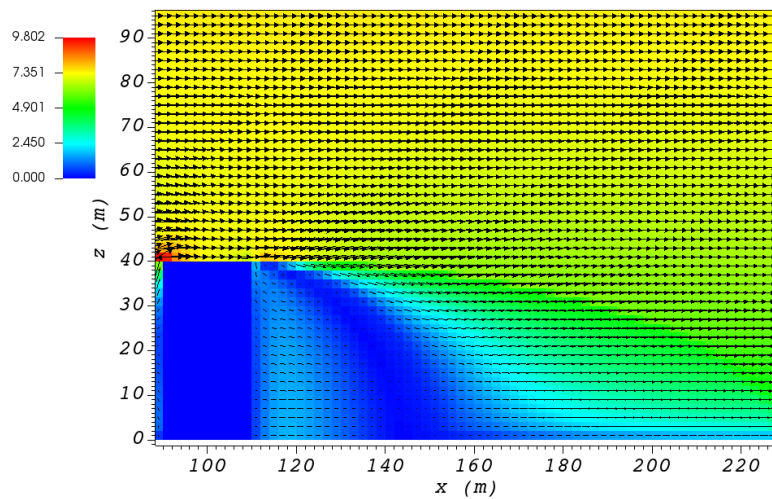
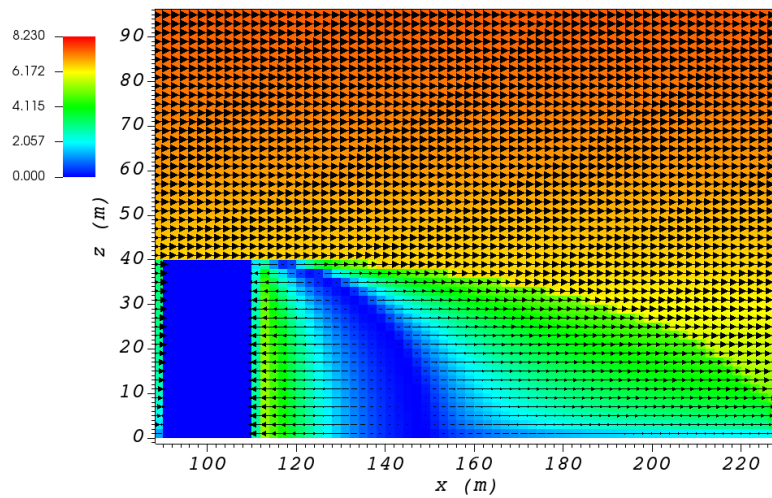
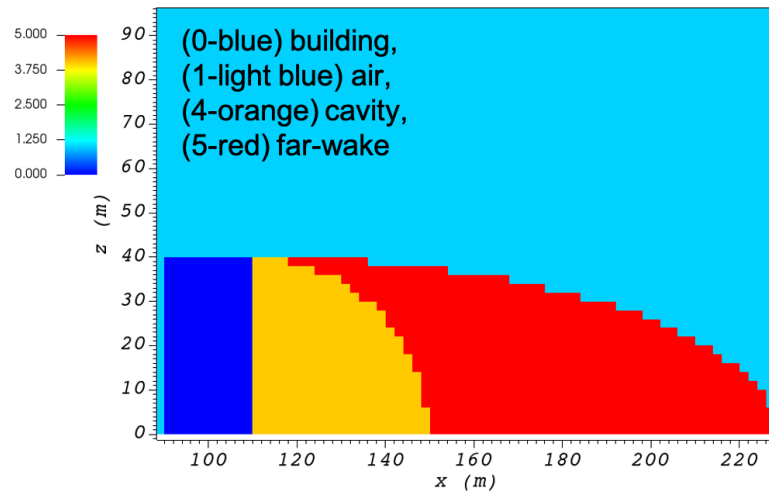
$$\frac{u(x, y, z)}{U(H)} = -\left(1 - \left(\frac{x}{d}\right)^2\right)$$

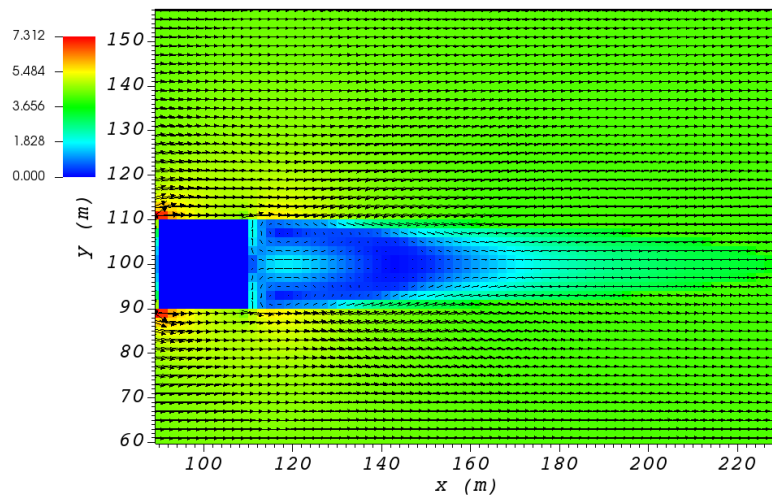
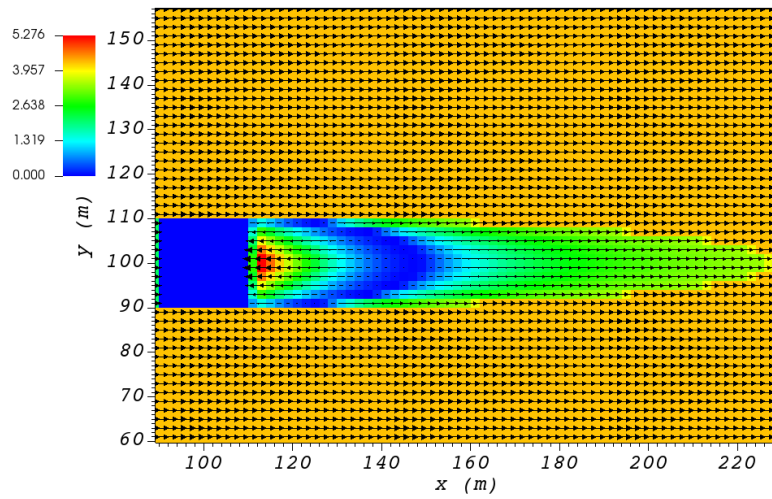
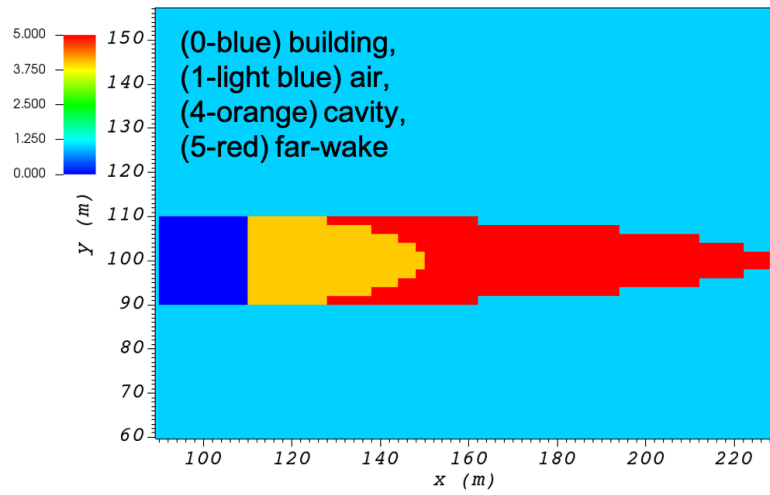
$$\frac{u(x, y, z)}{U(H)} = \left(1 - \left(\frac{d}{x}\right)^{1.5}\right)$$

where L , H and W are length, width and height of the building, respectively. $u(x, y, z)$ is the velocity at point (x, y, z) , $U(H)$ is the reference velocity at height of the building and x is the distance from the building in the stream-wise direction.

Part (a) of Figure [fig:wake_vert] and Figure [fig:wake_horiz] show cell type contour to represent the area of effect of the Röckle wake parameterization in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively. The wake parameterization is applied to a rectangular building defined in Section 1.4.3. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:wake_vert] and Figure [fig:wake_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part(c)) velocity fields in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively.

In order to turn on the wake model, the user needs to change the value of “wakeFlag” in the XML file.





```

<simulationParameters>
  <wakeFlag> 1 </wakeFlag>                                <!-- Wake flag (0-none, 1-Rockle (default)) -
  <->
</simulationParameters>

```

4.6.3 Street Canyon

The street canyon parameterization detailed in [11] represents the effects of two buildings in close vicinity to each other, on the fluid flow. Röckle [15] Introduced velocity parameterizations for the stream-wise components as in Eq. [eq:u_can] and the vertical component as in Eq. [eq:w_can].

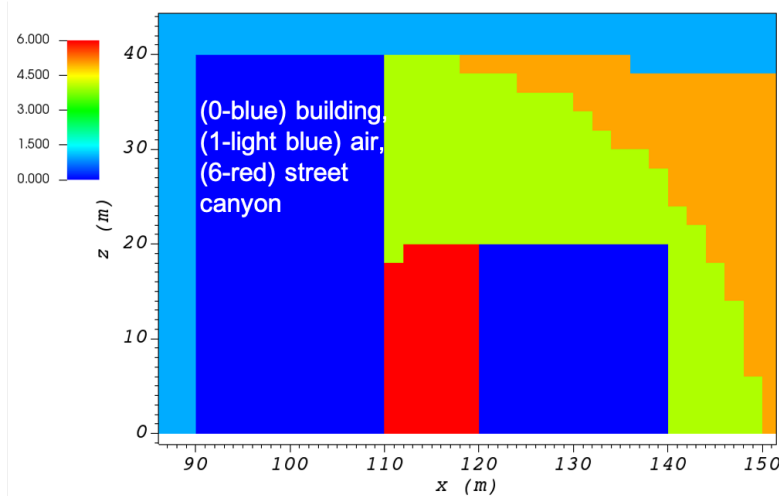
$$\frac{u(x, y, z)}{U(H)} = -\frac{x_{\text{can}}}{(0.5S)} \left(\frac{S - x_{\text{can}}}{0.5S} \right)$$

$$\frac{w(x, y, z)}{U(H)} = -\left| \frac{1}{2} \left(1 - \frac{x_{\text{can}}}{0.5S} \right) \right| \left(1 - \frac{S - x_{\text{can}}}{0.5S} \right)$$

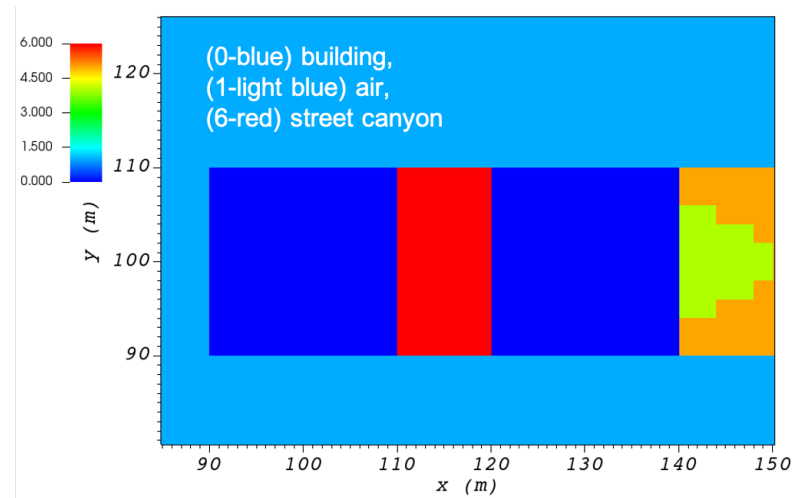
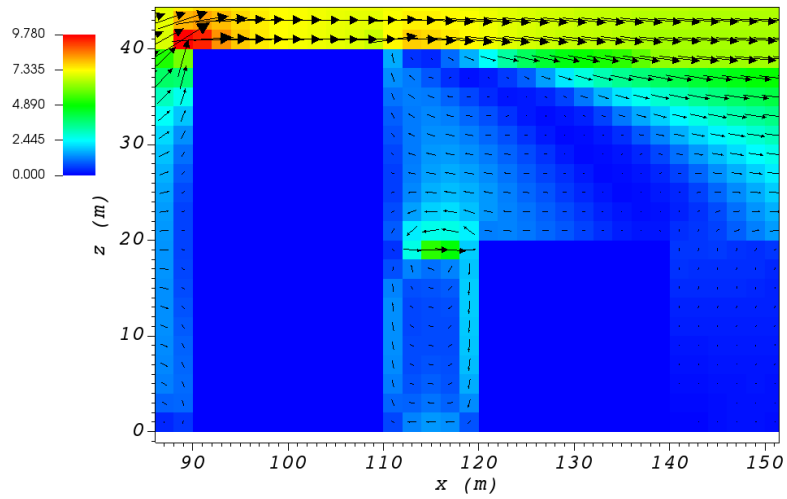
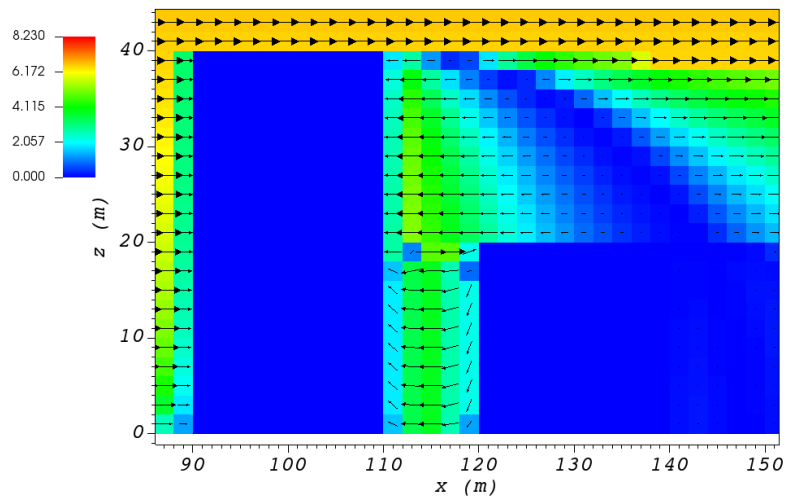
where S is the spacing between two buildings and x_{can} is the distance from the backwall of the upwind building.

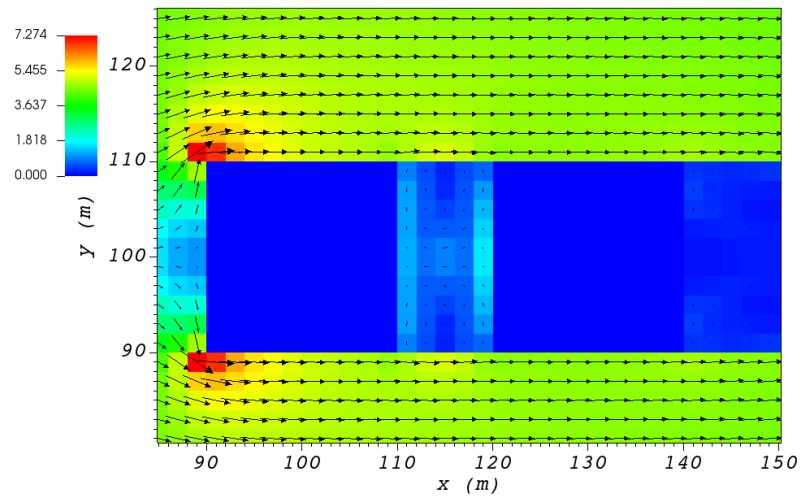
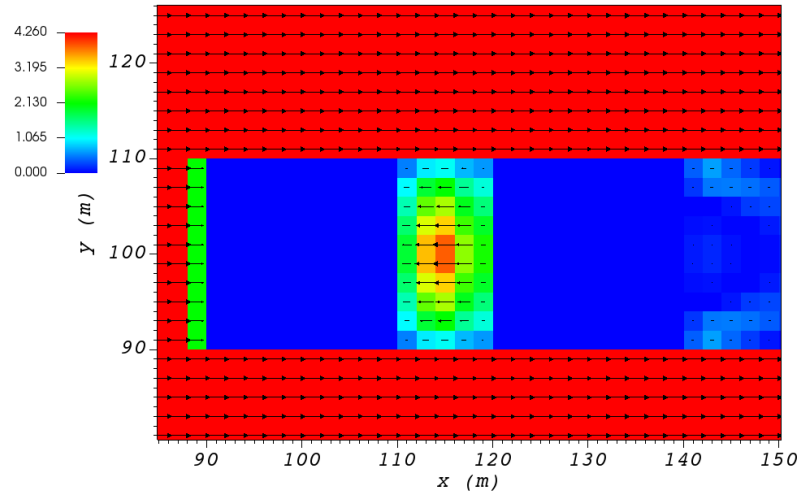
In order to identify the criteria to determine the existence of a street canyon, Singh et al. [11] utilized the cavity length, L_R (Eq. [eq:Lr]), for the upwind building. If $S < L_R$, the street canyon parameterization is applied, otherwise, the upwind building is considered as an isolated building.

Part (a) of Figure [fig:street_vert] and Figure [fig:street_horiz] show cell type contour to represent the area of effect of the street canyon parameterization in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively. The street canyon parameterization is applied to an area between two rectangular buildings. The upwind building is same as the one defined in Section 1.4.3. The downwind building is a rectangular building with 20 m as height, 0 m as base height, 20 m as length and width, closest corner to the origin located at 90 m in x and 120 m in y directions, and 0° as rotation angle with respect to the North-South line. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:street_vert] and Figure [fig:street_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part(c)) velocity fields in a vertical plane at $y = 100$ m and a horizontal plane at $z = 5$ m, respectively.



To turn on the street canyon parameterization, the user needs to change the value of “streetCanyonFlag” in the XML file.





```

<simulationParameters>
  <streetCanyonFlag> 1 </streetCanyonFlag>          <!-- Street canyon flag (0-none, 1-
  <!-- 1-Roeckle w/ Fackrel (default)) -->
</simulationParameters>

```

4.6.4 Rooftop Recirculation

The rooftop parameterization described in [19, 20], captures the separation of the flow from the leading edge of the building. It first checks if the incident flow is in $\pm 15^\circ$ of perpendicular to the front face. The parameterization then creates an ellipsoidal region above the building with height of H_c (height of the vortex, calculated by Eq. [eq:Hc]) and length of L_c (length of the vortex, calculated by Eq. [eq:Lc]). It applies a logarithmic profile in the whole vortex area and finally, reverses the velocity in region 1. Region 1 is an ellipsoidal zone with the same length as the vortex and half of the height.

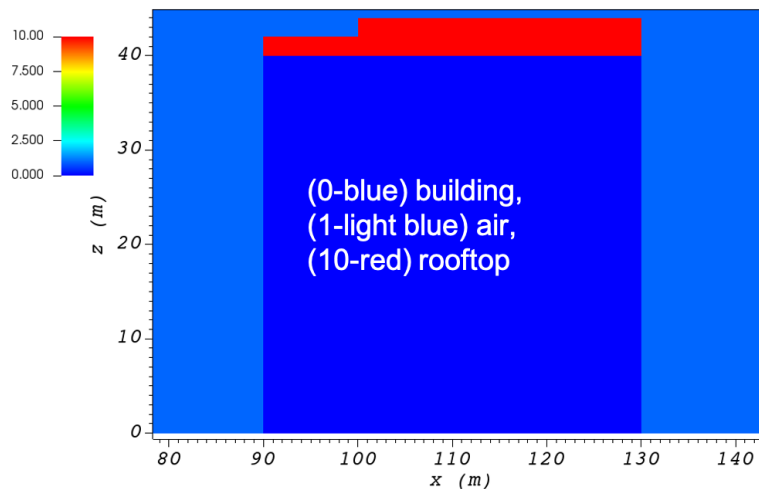
$$R = B_s^{2/3} B_l^{1/3}$$

$$L_c = 0.9R$$

$$H_c = 0.22R$$

where B_s is the smaller of the height (H) and the effective width (W_{eff}) of the building, B_l is the larger of H and W_{eff} , R is the vortex size scaling factor.

Part (a) of Figure [fig:street_vert] show cell type contour to represent the area of effect of the rooftop parameterization in a vertical plane at $y = 100$ m. The rooftop parameterization is applied to a rectangular building with 40 m as height, 0 m as base height, 40 m as length and width, closest corner to the origin located at 90 m in x and y directions, and 0° as rotation angle with respect to the North-South line. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:street_vert] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part (c)) velocity fields in a vertical plane at $y = 100$ m.

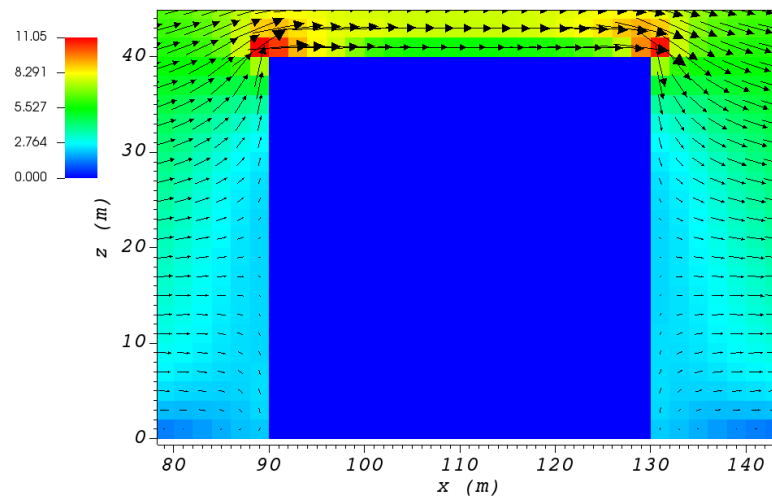
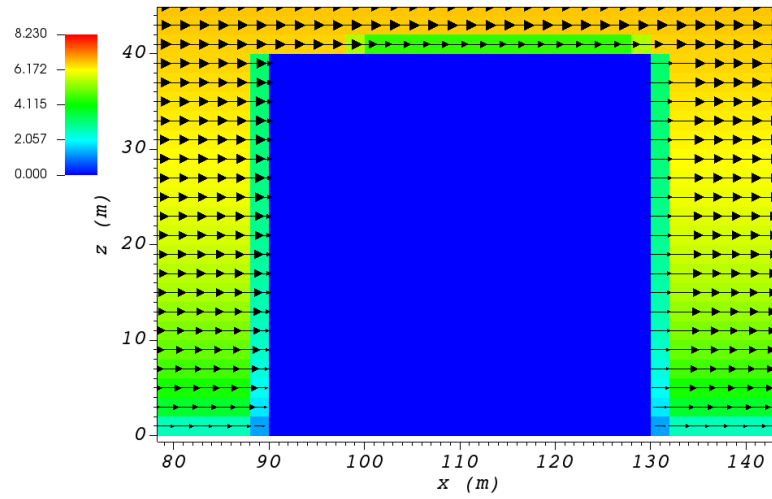


To turn the parameterization on, the user needs to change the value of “rooftopFlag” in the XML file.

```

<simulationParameters>
  <rooftopFlag> 1 </rooftopFlag>          <!-- Rooftop flag (0-none, 1-log_
  <!-- profile (default)) -->
</simulationParameters>

```



4.6.5 Sidewall Recirculation Zone

The sidewall parameterization is designed to represent the effects of the edge of the building on the upwind field [21]. It first checks if a face has an outward normal vector nominally ($\pm 10^\circ$) perpendicular to the local wind vector. The important parameters controlling the sidewall vortex strength and geometry are:

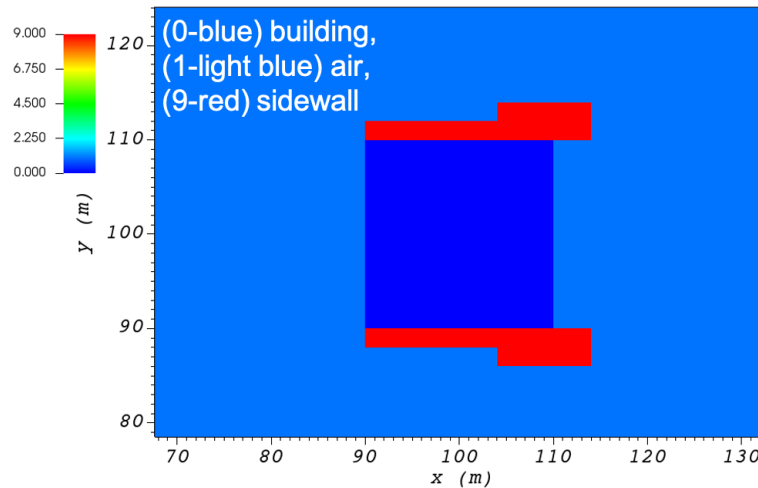
$$R = B_s^{2/3} B_l^{1/3}$$

$$L_c = 0.9R$$

$$W_c = 0.22R$$

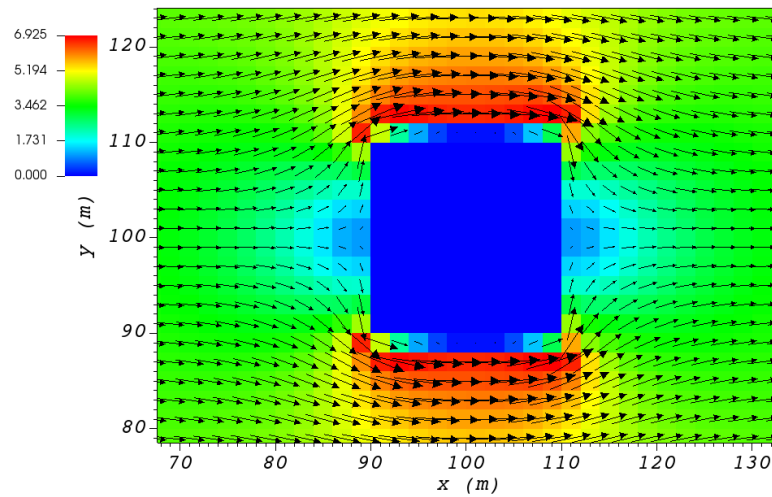
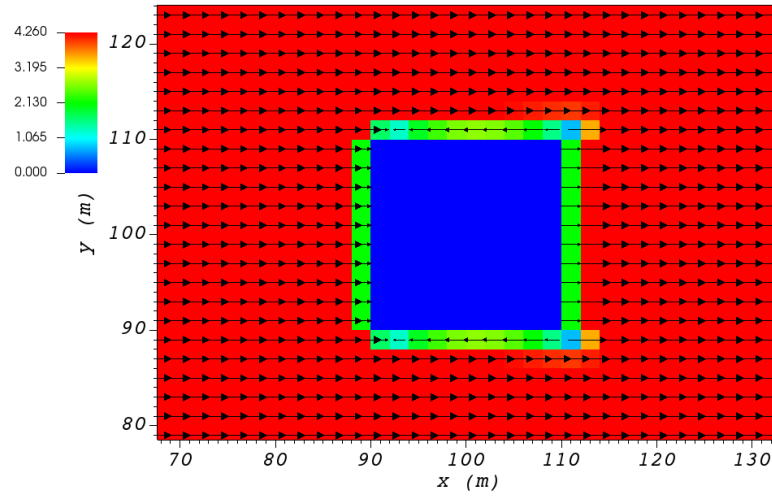
where B_s is the smaller of the height (H) and the effective width (W_{eff}) of the building, B_l is the larger of H and W_{eff} , R is the vortex size scaling factor, L_c is the downwind length of the half-ellipse that defines the vortex recirculation region, and W_c is the lateral width of the elliptical recirculation region. Within the recirculation zone, the velocity is reversed and scaled linearly from the reference wind speed near the wall to zero at the edge of the ellipse.

Part (a) of Figure [fig:street_vert] show cell type contour to represent the area of effect of the sidewall parameterization in a horizontal plane at $z = 5$ m. The rooftop parameterization is applied to a rectangular building defined in Section 1.4.3. The initial guess field is constructed using a single sensor with logarithmic profile as defined in 1.5.1. Parts (b) and (c) of Figure [fig:sidewall_horiz] indicate velocity magnitude contour with overlaying velocity vectors of initial (part (b)) and final (part(c)) velocity fields in a horizontal plane at $z = 5$ m.



In order to turn the algorithm on, the user needs to change the value of “sidewallFlag” in the XML file.

```
<simulationParameters>
  <sidewallFlag> 1 </sidewallFlag>          <!-- Sidewall flag (0-off, 1-on,
  ↪(default)) -->
</simulationParameters>
```

4.7 Mass Consistent Solver

QES-Winds have mass conserving wind field solvers that rapidly compute wind fields using a variational method rather than slower yet more physics based solvers that include conservation of momentum [22]. While the QES-Winds method uses reduced order physics in the numerical solution of urban flow problems, the solutions are rapid and compare quite well higher order physics-based models in both idealized [21] and realistic urban cities [23]. The method minimizes the difference between an initial wind field that is specified using empirical parameterizations [11] and the final wind fields. The empirical parameterizations account for complex wind fields around buildings such as wake cavities downstream of a building. To obtain a quasi-time-averaged velocity field, QES-Winds uses a variational analysis technique [11]. This method requires the solution of a Poisson equation for Lagrange multipliers, λ (Equation [poisson]) in the following form:

$$\frac{\partial^2 \lambda}{\partial x^2} + \frac{\partial^2 \lambda}{\partial y^2} + \left(\frac{\alpha_1}{\alpha_2}\right)^2 \frac{\partial^2 \lambda}{\partial z^2} = R$$

Where R is divergence of the initial wind field and is defined as:

$$R = -2\alpha_1^2 \left[\frac{u_{i+1/2}^0 - u_{i-1/2}^0}{\Delta x} + \frac{v_{j+1/2}^0 - v_{j-1/2}^0}{\Delta y} + \frac{w_{k+1/2}^0 - w_{k-1/2}^0}{\Delta z} \right]$$

The final velocity field is updated using Euler-Lagrange equations:

$$\begin{aligned} u &= u^0 + \frac{1}{2\alpha_1^2 \Delta x} [\lambda_{i+1,j,k} - \lambda_{i,j,k}] \\ v &= v^0 + \frac{1}{2\alpha_1^2 \Delta y} [\lambda_{i,j+1,k} - \lambda_{i,j,k}] \\ w &= w^0 + \frac{1}{2\alpha_2^2 \Delta z} [\lambda_{i,j,k+1} - \lambda_{i,j,k}] \end{aligned}$$

The Poisson equation is solved using the Successive Over-Relaxation (SOR) method which is a variant of Gauss-Seidel method with faster convergence. Applying SOR to Equation [poisson] results in:

$$\begin{aligned} \lambda_{i,j,k} &= \frac{\omega \left[(\Delta x)^2 R_{i,j,k} + e \lambda_{i+1} + f \lambda_{i-1} + A(g \lambda_{j+1} + h \lambda_{j-1}) + B(m \lambda_{k+1} + n \lambda_{k-1}) \right]}{e + f + g + h + m + n} \\ &\quad + (1 - \omega) \lambda_{i,j,k} \end{aligned}$$

Where e,f,g,h,m,n are boundary condition coefficients and A and B are domain constants. $\omega = 1.78$ is the SOR relaxation factor. The boundary condition for solid surfaces is ($\frac{\partial \lambda}{\partial n} = 0$) and for inlet/outlet surfaces it is $\lambda = 0$.

4.7.1 Solver Types

QES-Winds has four options for solving the SOR equation discussed above, the first option is to solve the equation on the CPU and the rest use the GPU for computations. The GPU solvers are called: the dynamic parallel, the global memory and the shared memory. The CPU solver is quite rapid, but slow in comparison to the GPU solvers since it is a serial solver and does not have parallel computing capabilities, especially for large domains. For more information regarding different types of solvers available in QES-Winds, read [2].

QES-TURB

QES-PLUME

6.1 The model

6.2 The XML file

6.2.1 Simulation Parameters

The parameters below are the parameters used to run the Plume model.

```
<simulationParameters>
  <simDur> 1000.0 </simDur>    <!-- Total simulation time -->
  <timeStep> 0.1 </timeStep>    <!-- time step -->
  <CourantNumber> 0.5 </CourantNumber>
  <invarianceTol> 1e-10 </invarianceTol>
  <C_0> 4.0 </C_0>
  <updateFrequency_particleLoop> 10000 </updateFrequency_particleLoop>
  <updateFrequency_timeLoop> 100 </updateFrequency_timeLoop>
</simulationParameters>
```

6.2.2 Collection Parameters

The parameters below are the parameters used to calculate the concentration of particle (in #particles/m³). All parameters are in SI units by default (second and meters). The size of collection area should be set smaller or equal to the domain.

```
<collectionParameters>
  <timeAvgStart> 0.0 </timeAvgStart>    <!-- time to start calc of concentration -->
  <timeAvgFreq> 60.0 </timeAvgFreq>    <!-- averaging period -->
  <boxBoundsX1> 0.0 </boxBoundsX1>    <!-- beginning of collection area -->
  <boxBoundsX2> 200.0 </boxBoundsX2>    <!-- end of collection area -->
  <boxBoundsY1> 0.0 </boxBoundsY1>
  <boxBoundsY2> 200.0 </boxBoundsY2>
  <boxBoundsZ1> 0.0 </boxBoundsZ1>
  <boxBoundsZ2> 200.0 </boxBoundsZ2>
  <nBoxesX> 200 </nBoxesX>            <!-- resolution of concentration -->
  <nBoxesY> 200 </nBoxesY>
  <nBoxesZ> 200 </nBoxesZ>
</collectionParameters>
```

6.2.3 Sources

```
<sources>
  <numSources> 1 </numSources> <!-- Number of active sources -->
  <!-- HERE COME THE SOURCES -->
</sources>
```

Source types

```
<SourcePoint>
  <!-- HERE COMES THE RELEASE TYPE -->
  <posX> 40.0 </posX>
  <posY> 80.0 </posY>
  <posZ> 30.0 </posZ>
</SourcePoint>
```

```
<SourceLine>
  <!-- HERE COMES THE RELEASE TYPE -->
  <posX_0> 25.0 </posX_0>
  <posY_0> 175.0 </posY_0>
  <posZ_0> 40.0 </posZ_0>
  <posX_1> 50.0 </posX_1>
  <posY_1> 25.0 </posY_1>
  <posZ_1> 40.0 </posZ_1>
</SourceLine>
```

```
<SourceCube>
  <!-- HERE COMES THE RELEASE TYPE -->
  <minX> 75.0 </minX>
  <minY> 25.0 </minY>
  <minZ> 70.0 </minZ>
  <maxX> 80.0 </maxX>
  <maxY> 35.0 </maxY>
  <maxZ> 80.0 </maxZ>
</SourceCube>
```

```
<SourceCircle>
  <!-- HERE COMES THE RELEASE TYPE -->
  <posX> 40.0 </posX>
  <posY> 80.0 </posY>
  <posZ> 30.0 </posZ>
  <radius> 30.0 </radius>
</SourceCircle>
```

```
<SourceFullDomain>
  <!-- HERE COMES THE RELEASE TYPE -->
</SourceFullDomain>
```

Release types

```
<ReleaseType_continuous>
  <parPerTimestep>10</parPerTimestep>
</ReleaseType_continuous>
```

```
<ReleaseType_duration>
  <releaseEndTime>0</releaseEndTime>
  <releaseEndTime>5</releaseEndTime>
  <parPerTimestep>10</parPerTimestep>
</ReleaseType_duration>
```

```
<ReleaseType_instantaneous>
  <numPar>100000</numPar>
</ReleaseType_instantaneous>
```

6.2.4 Boundary Conditions

```
<boundaryConditions>
  <xBCtype>exiting</xBCtype>
  <yBCtype>exiting</yBCtype>
  <zBCtype>exiting</zBCtype>
  <wallReflection>stairstepReflection</wallReflection>
</boundaryConditions>
```

Here are the option of the boundary conditions types:

- exiting particle exit the domain
- periodic particle reenter the domain at the other side
- reflection particle is reflected from the domain boundary (works only of domain ends)

Here are the option of the wall reflections methods

- doNothing nothing happen when particle enter wall
- setInactive (default) particle is set to inactive when entering a wall
- stairstepReflection particle use full stair step reflection when entering a wall

6.2.5 Full XML Example

```
<simulationParameters>
  <simDur> 1000.0 </simDur>
  <timeStep> 0.1 </timeStep>
  <CourantNumber> 1 </CourantNumber>
  <invarianceTol> 1e-10 </invarianceTol>
  <C_0> 4.0 </C_0>
  <updateFrequency_particleLoop> 10000 </updateFrequency_particleLoop>
  <updateFrequency_timeLoop> 100 </updateFrequency_timeLoop>
</simulationParameters>
<collectionParameters>
```

(continues on next page)

(continued from previous page)

```

<timeAvgStart> 0.0 </timeAvgStart>
<timeAvgFreq> 60.0 </timeAvgFreq>
<boxBoundsX1> 0.0 </boxBoundsX1>
<boxBoundsX2> 200.0 </boxBoundsX2>
<boxBoundsY1> 0.0 </boxBoundsY1>
<boxBoundsY2> 200.0 </boxBoundsY2>
<boxBoundsZ1> 0.0 </boxBoundsZ1>
<boxBoundsZ2> 200.0 </boxBoundsZ2>
<nBoxesX> 200 </nBoxesX>
<nBoxesY> 200 </nBoxesY>
<nBoxesZ> 200 </nBoxesZ>
</collectionParameters>
<sources>
  <numSources> 1 </numSources>
  <SourcePoint>
    <ReleaseType_continuous>
      <parPerTimestep>10</parPerTimestep>
    </ReleaseType_continuous>
    <posX> 10.0 </posX>
    <posY> 100.0 </posY>
    <posZ> 50.0 </posZ>
  </SourcePoint>
</sources>
<boundaryConditions>
  <xBCtype>exiting</xBCtype>
  <yBCtype>exiting</yBCtype>
  <zBCtype>exiting</zBCtype>
  <wallReflection>stairstepReflection</wallReflection>
</boundaryConditions>

```


PUBLICATIONS

- QES-Winds: Dynamic Parallelism Solver

Bozorgmehr, B., Willemsen, P., Gibbs, J.A., Stoll, R., Kim, J.-J., Pardyjak, E.R., 2021. Utilizing dynamic parallelism in CUDA to accelerate a 3D red-black successive over relaxation wind-field solver. *Environ Modell Softw* 137, 104958. <https://doi.org/10.1016/j.envsoft.2021.104958>

- Isolated Tree Model

Margairaz, F., Eshagh, H., Hayati, A.N., Pardyjak, E.R., Stoll, R., 2022. Development and evaluation of an isolated-tree flow model for neutral-stability conditions. *Urban Clim* 42, 101083. <https://doi.org/10.1016/j.uclim.2022.101083>

- QES-Fire: wildfire model

Moody, M.J., Gibbs, J.A., Krueger, S., Mallia, D., Pardyjak, E.R., Kochanski, A.K., Bailey, B.N., Stoll, R., 2022. QES-Fire: a dynamically coupled fast-response wildfire model. *Int J Wildland Fire* 31, 306–325. <https://doi.org/10.1071/wf21057>

- Row-Oriented Canopy Model

Ulmer, L., Margairaz, F., Bailey, B.N., Mahaffee, W.F., Pardyjak, E.R., Stoll, R., 2022. A fast-response, wind angle-sensitive model for predicting mean winds in row-organized canopies. *Agric. For. Meteorol.* 329, 109273. <https://doi.org/10.1016/j.agrformet.2022.109273>

REFERENCE LIST

ACKNOWLEDGEMENTS

This work was partly supported by grants from:

- The National Institute of Environment Research (NIER), funded by the Ministry of Environment (MOE) of the Republic of Korea (NIER-SP2019-312). In addition, we would like to acknowledge Dr. Jae-Jin Kim from Department of Environmental Atmospheric Sciences, Pukyong National University, Republic of Korea, as the main Principal Investigator (PI) on the grant from the National Institute of Environment Research (NIER).
- The United States Department of Agriculture National Institute for Food and Agriculture Specialty Crop Research Initiative Award No. 2018-03375.
- The United States Department of Agriculture Agricultural Research Service through Research Support Agreement 58-2072-0-036.

BIBLIOGRAPHY

- [1] Michael J Brown, Akshay A Gowardhan, Mathew A Nelson, Michael D Williams, and Eric R Pardyjak. Quic transport and dispersion modelling of two releases from the joint urban 2003 field experiment. *International Journal of Environment and Pollution*, 52(3-4):263–287, 2013.
- [2] Behnam Bozorgmehr, Pete Willemssen, Jeremy Gibbs, Rob Stoll, Jae-Jin Kim, and Eric Pardyjak. Utilizing dynamic parallelism in cuda to accelerate a 3d red-black successive over relaxation wind-field solver. *Environmental Modelling & Software*, 137:104958, 01 2021. doi:10.1016/j.envsoft.2021.104958.
- [3] PA Taylor and HW Teunissen. The askervein hill project: overview and background data. *Boundary-layer meteorology*, 39(1-2):15–39, 1987.
- [4] RE Mickle, NJ Cook, AM Hoff, NO Jensen, JR Salmon, PA Taylor, G Tetzlaff, and HW Teunissen. The askervein hill project: vertical profiles of wind and turbulence. *Boundary-Layer Meteorology*, 43(1-2):143–169, 1988.
- [5] K Jerry Allwine and Julia E Flaherty. Joint urban 2003: study overview and instrument locations. Technical Report, Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2006.
- [6] Tony Favaloro, Eric R Pardyjak, and Michael J Brown. *Toward Understanding the Sensitivity of the QUIC Dispersion Modeling System to Real Input Data*. PhD thesis, Department of Mechanical Engineering, University of Utah, 2008.
- [7] ER Pardyjak, SO Speckart, F Yin, and JM Veranth. Near source deposition of vehicle generated fugitive dust on vegetation and buildings: model development and theory. *Atmospheric Environment*, 42(26):6442–6452, 2008.
- [8] Steven E Koch, Mary DesJardins, and Paul J Kocin. An interactive Barnes objective map analysis scheme for use with satellite and conventional data. *Journal of climate and applied meteorology*, 22(9):1487–1503, 1983.
- [9] Thomas Michael Booth and ER Pardyjak. *Validation of a data assimilation technique for an urban wind model*. Department of Mechanical Engineering, University of Utah, 2012.
- [10] Jordan G Powers, Joseph B Klemp, William C Skamarock, Christopher A Davis, Jimmy Dudhia, David O Gill, Janice L Coen, David J Gochis, Ravan Ahmadov, Steven E Peckham, and others. The weather research and forecasting model: overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717–1737, 2017.
- [11] Balwinder Singh, Bradley S Hansen, Michael J Brown, and Eric R Pardyjak. Evaluation of the quic-urb fast response urban wind model for a cubical building array and wide building street canyon. *Environmental fluid mechanics*, 8(4):281–312, 2008.
- [12] **missing journal in nelson20085**
- [13] N Bagal, ER Pardyjak, and MJ Brown. Improved upwind cavity parameterization for a fast response urban wind model. In *84th Annual AMS Meeting*. Seattle, WA. 2004.
- [14] Akshay A Gowardhan, Michael J Brown, and Eric R Pardyjak. Evaluation of a fast response pressure solver for flow around an isolated cube. *Environmental fluid mechanics*, 10(3):311–328, 2010.

- [15] Rainer Röckle. *Bestimmung der Strömungsverhältnisse im Bereich komplexer Bauungsstrukturen*. na, 1990.
- [16] H Kaplan and N Dinar. A lagrangian dispersion model for calculating concentration distribution within a built-up domain. *Atmospheric Environment*, 30(24):4197–4207, 1996.
- [17] Balwinder Singh. *Testing and development of a fast response Lagrangian dispersion models*. PhD thesis, Department of Mechanical Engineering, University of Utah, 2005.
- [18] B Singh, ER Pardyjak, MJ Brown, and MD Williams. Testing of a far-wake parameterization for a fast response urban wind model. In *Sixth Symposium on the Urban Environment/14th Joint Conference on the Applications of Air Pollution Meteorology with the Air and Waste Management Association, Atlanta, January J*, volume 8. 2006.
- [19] Nilesh Bagal, Balwinder Singh, Eric R Pardyjak, and Michael J Brown. Implementation of rooftop recirculation parameterization into the quic fast response urban wind model. Technical Report, Los Alamos National Laboratory, 2004.
- [20] **missing journal in pol2006implementation**
- [21] Arash Nemati Hayati, Rob Stoll, JJ Kim, Todd Harman, Matthew A Nelson, Michael J Brown, and Eric R Pardyjak. Comprehensive evaluation of fast-response, reynolds-averaged navier–stokes, and large-eddy simulation methods against high-spatial-resolution wind-tunnel data in step-down street canyons. *Boundary-Layer Meteorology*, 164(2):217–247, 2017.
- [22] Jae-Jin Kim, Eric Pardyjak, Do-Yong Kim, Kyoung-Soo Han, and Byung-Hyuk Kwon. Effects of building-roof cooling on flow and air temperature in urban street canyons. *Asia-Pacific Journal of Atmospheric Sciences*, 50(3):365–375, 2014.
- [23] Marina Neophytou, Akshay Gowardhan, and Michael Brown. An inter-comparison of three urban wind models using oklahoma city joint urban 2003 wind field measurements. *Journal of Wind Engineering and Industrial Aerodynamics*, 99(4):357–368, 2011.