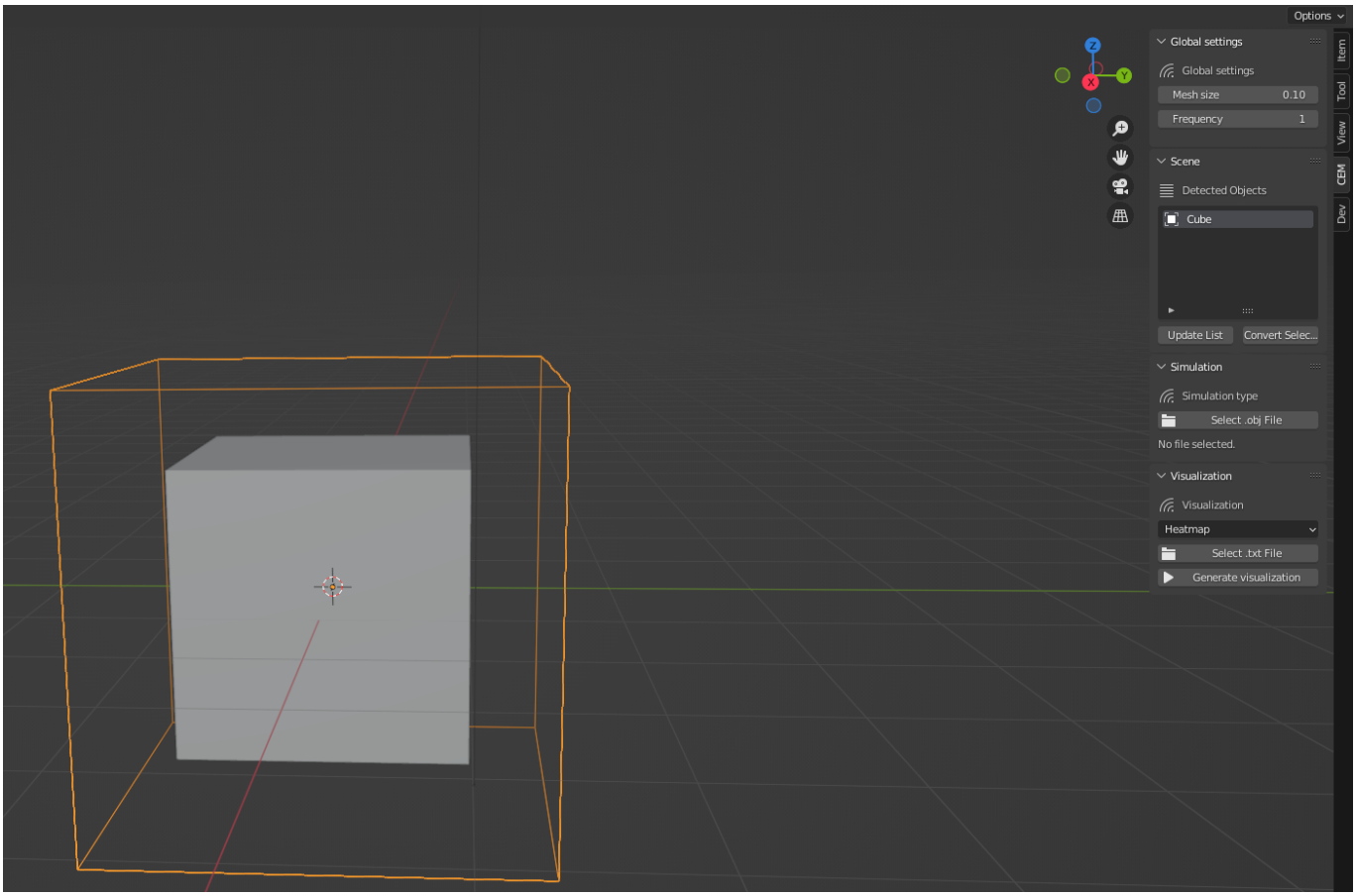
blender-cem-addon

Addon to convert object in the scene in voxels. Simulate and visualize stuff around.



Requirements

Install Blender from the Official Website the version 3.3 or above.

> - For Linux, please do not use any packet manager, install it from the source `tar.gz`
> - For MacOS and Windows use the official installer

First install

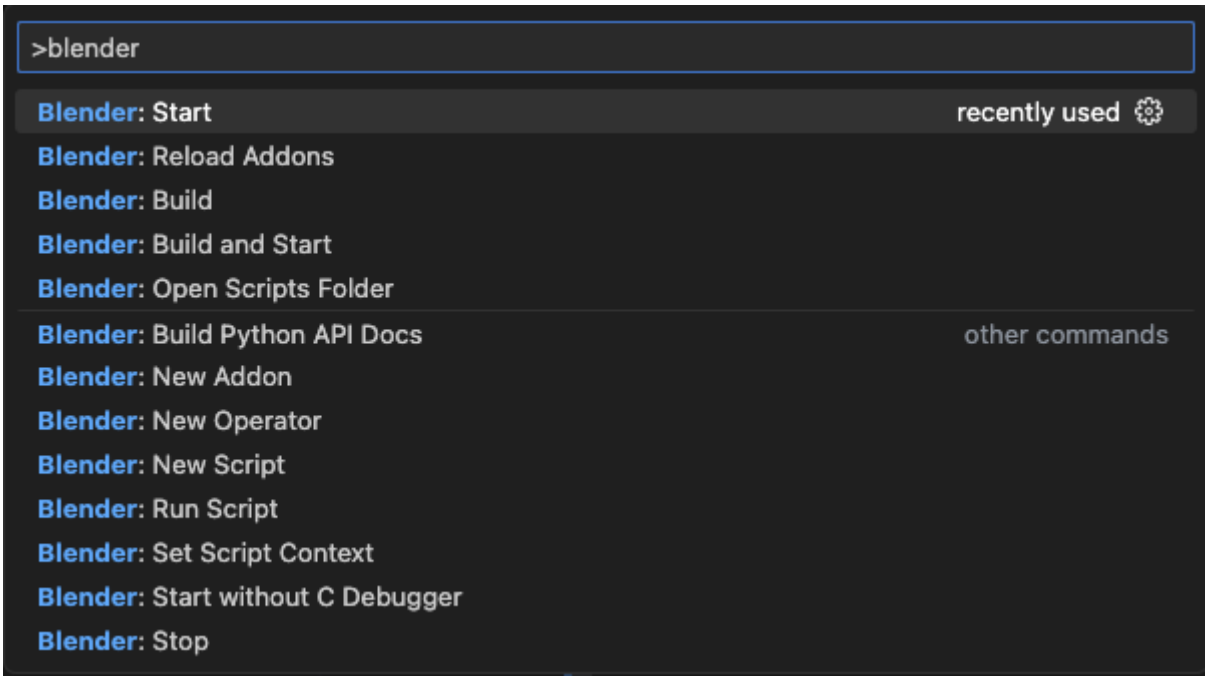
This section is a *step-by-step* guide for installing the addon in order to contribute to the project.

Configure Workspace

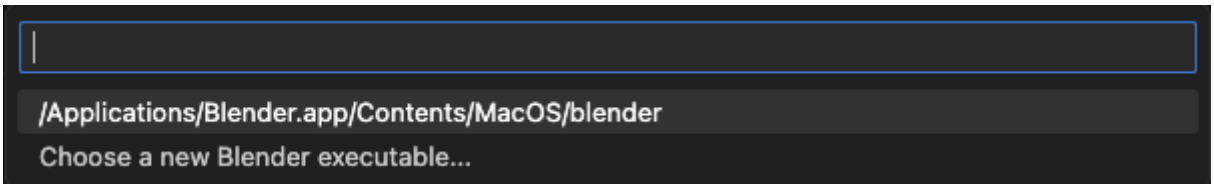
In order to develop the add-on, you should use VSCode. You can install the extension Blender Development to have a better experience.

Once you have installed the extension, you can open Blender by using `CTRL+SHIFT+P`.

1. Select `Blender: Start`

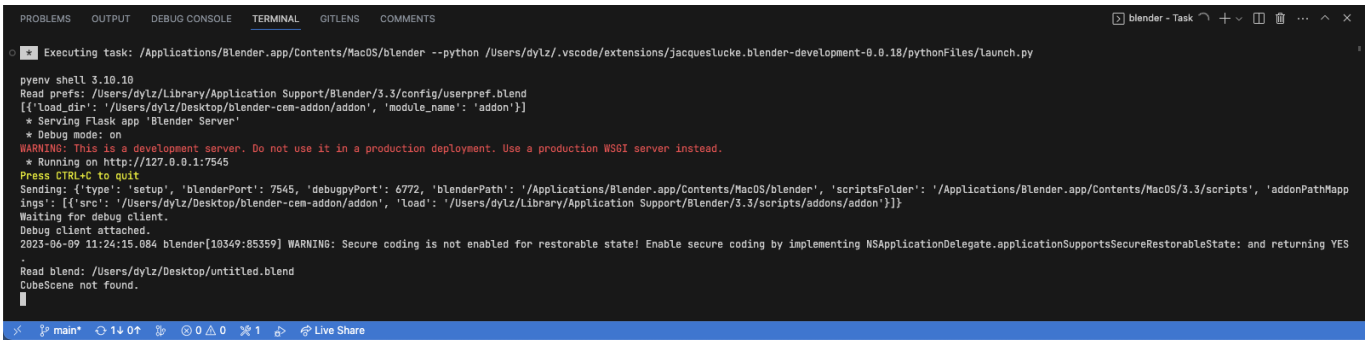


2. For the first run select `Choose a new Blender executable...`

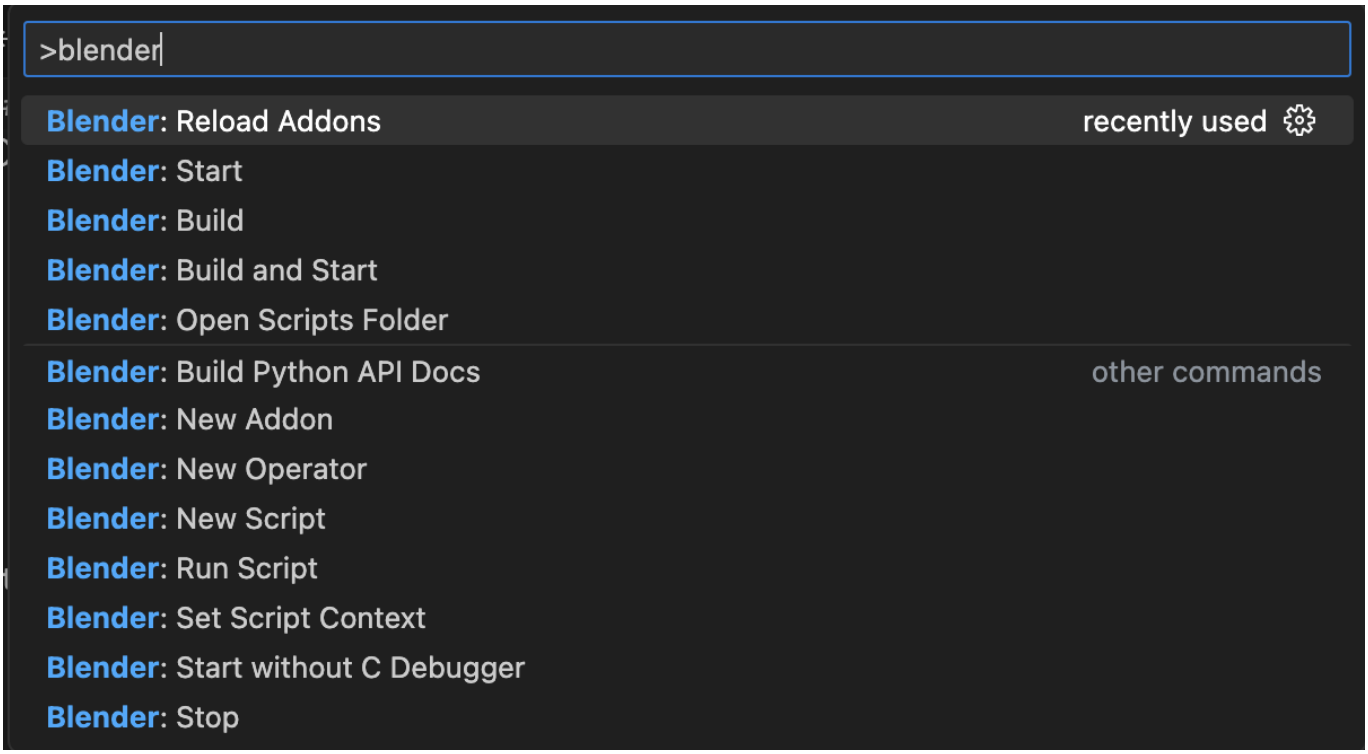


Now Blender is open in development mode.

3. You can now see the console and the python editor to debug the add-on easily.



4. In development mode you can easily reload the add-on when you modified the files by using `CTRL+SHIFT+P` and select `Blender: Reload Scripts`



Install in production mode

1. First you need to generate the zip file of the add-on project. In the terminal, run the following command at the root path of the project :

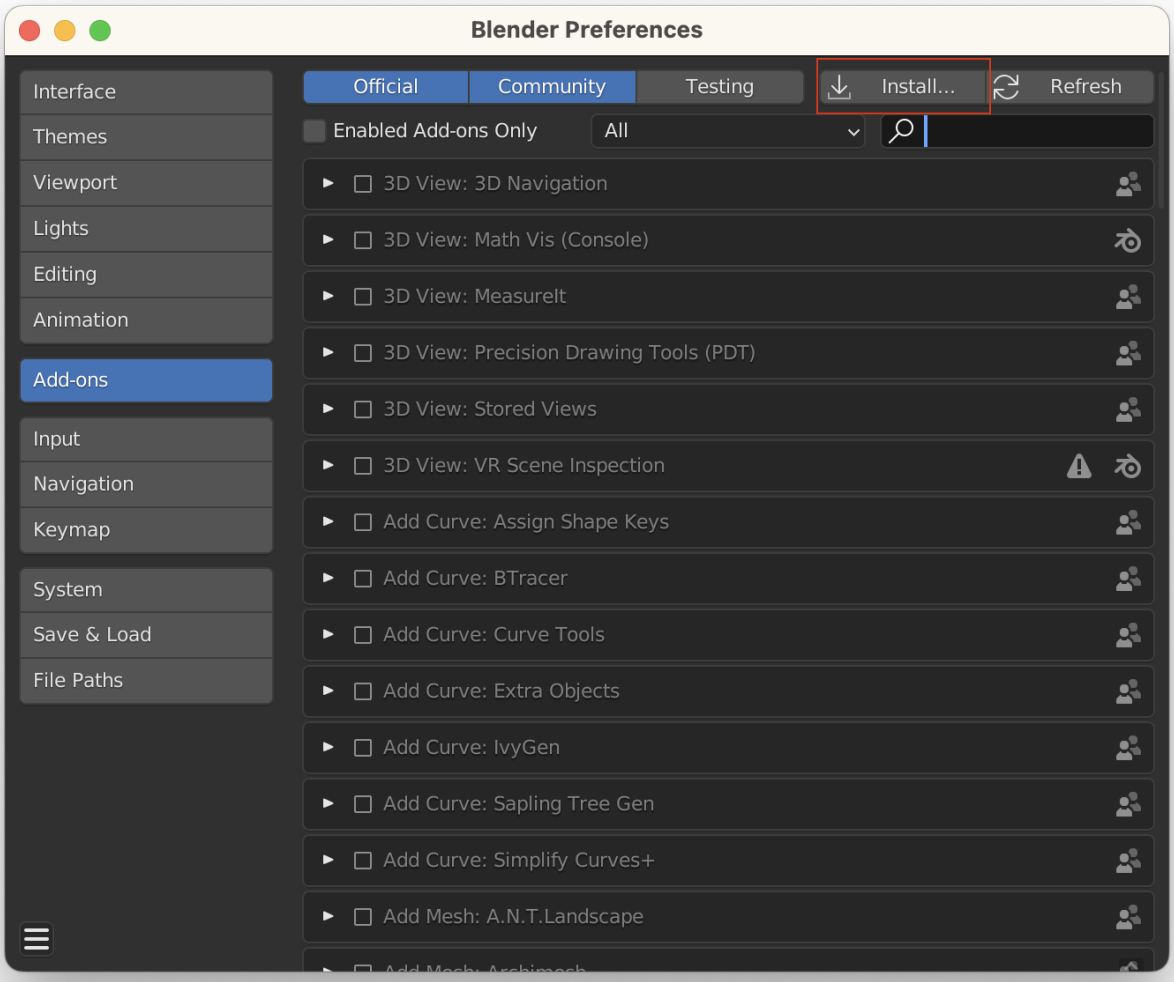
```
make
```

You should have this output:

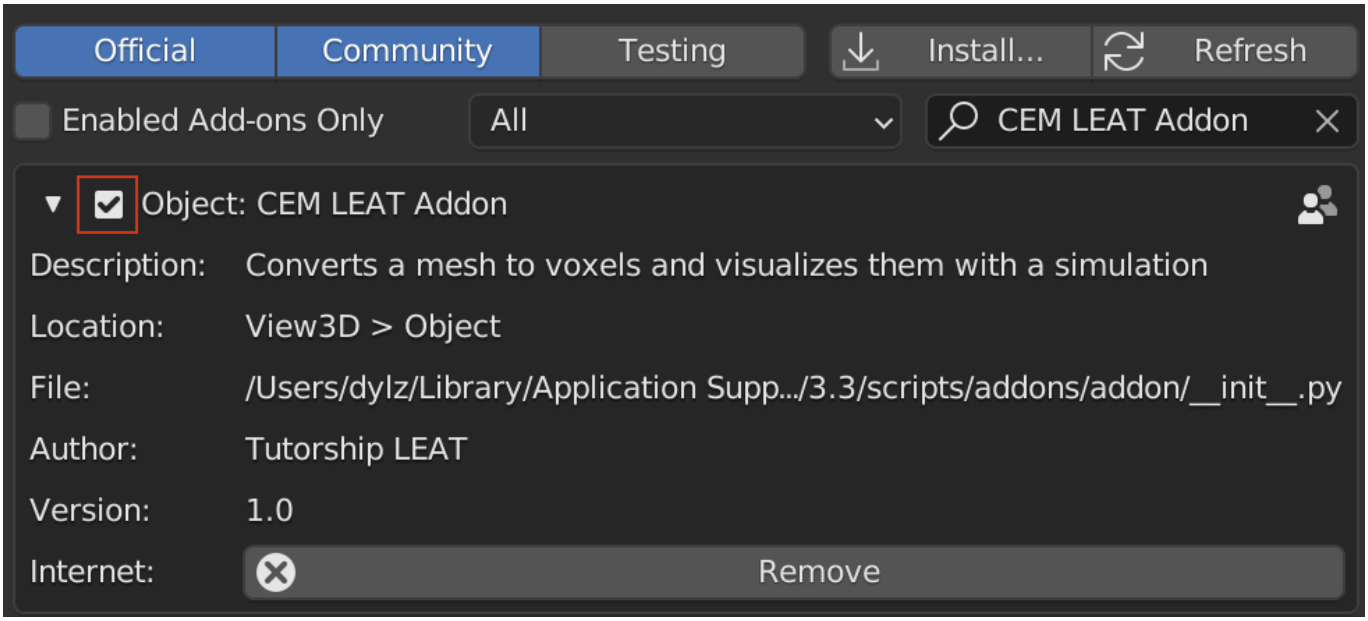
```
➜  blender-cem-addon git:(main) ✗ make
rm -rf blender-cem-addon.zip
rm -rf ./addon/__pycache__
zip -r blender-cem-addon.zip ./addon
  adding: addon/ (stored 0%)
  adding: addon/plot.py (deflated 70%)
  adding: addon/visualisations.py (deflated 25%)
  adding: addon/constants.py (deflated 26%)
  adding: addon/simulations.py (deflated 63%)
  adding: addon/__init__.py (deflated 53%)
  adding: addon/scene_opt.py (deflated 69%)
  adding: addon/voxelizer.py (deflated 72%)
  adding: addon/converters.py (deflated 69%)
  adding: addon/init.py (deflated 71%)
  adding: addon/visualization_opt.py (deflated 70%)
  adding: addon/settings_opt.py (deflated 64%)
  adding: addon/simulation_opt.py (deflated 70%)
```

The file `blender-cem-addon.zip` has been generated.

2. Now you can install the add-on in Blender. Open Blender and go to `Edit > Preferences > Add-ons > Install...` and select the zip file generated in the previous step.



Don't forget to check the box to activate the add-on.



Project dependencies

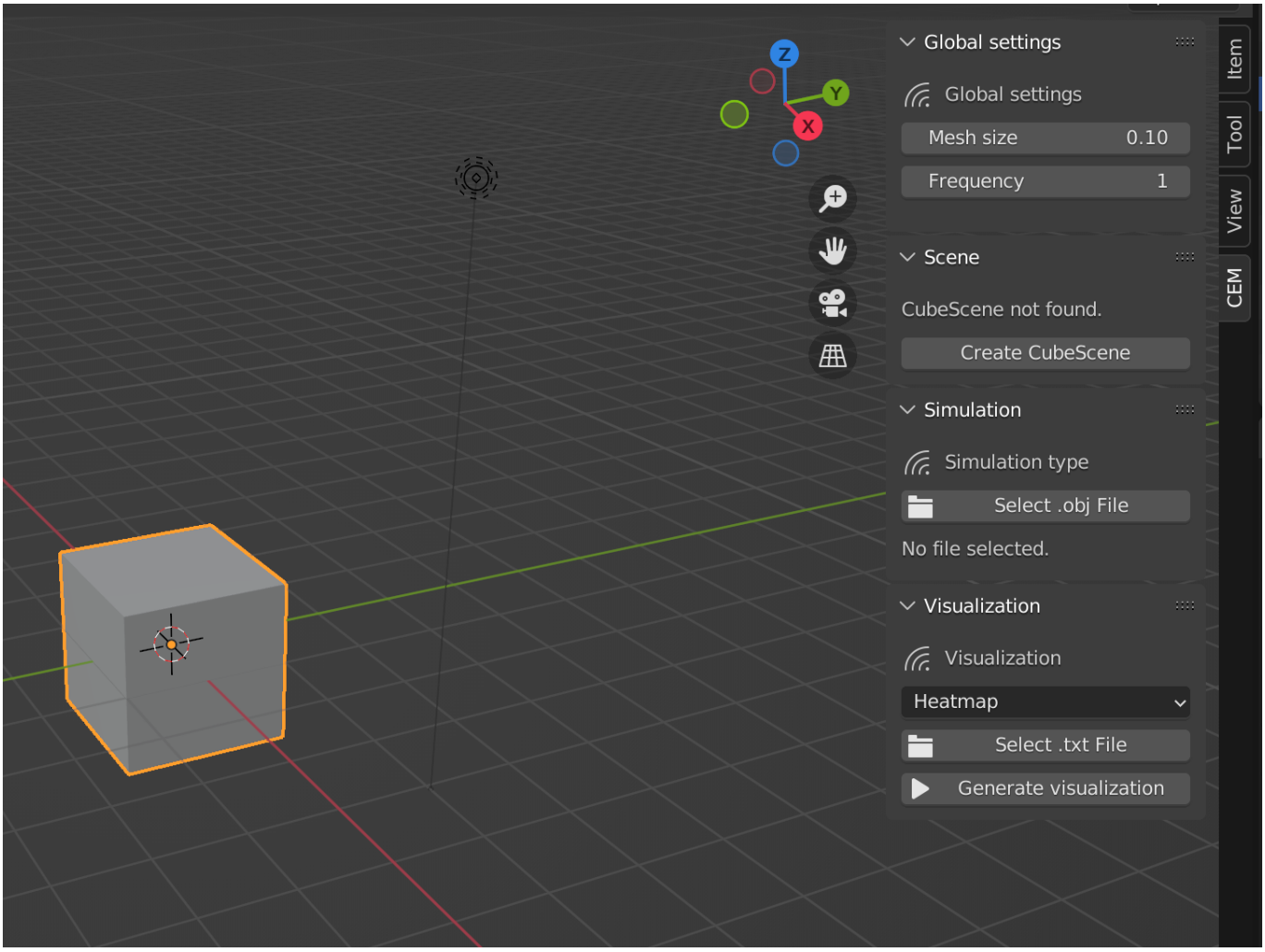
Dependencies are register in the constants.py file. You can add dependencies by adding new item to this list :

```
DEPENDENCIES = ['seaborn', 'trimesh', 'matplotlib', 'pandas', 'numpy',
'scipy']
```

How to use

Before starting, in order to use the addon, make sure to have saved your blender project. It's better to have an empty folder for the project, because the addon will create new folders inside the project folder.

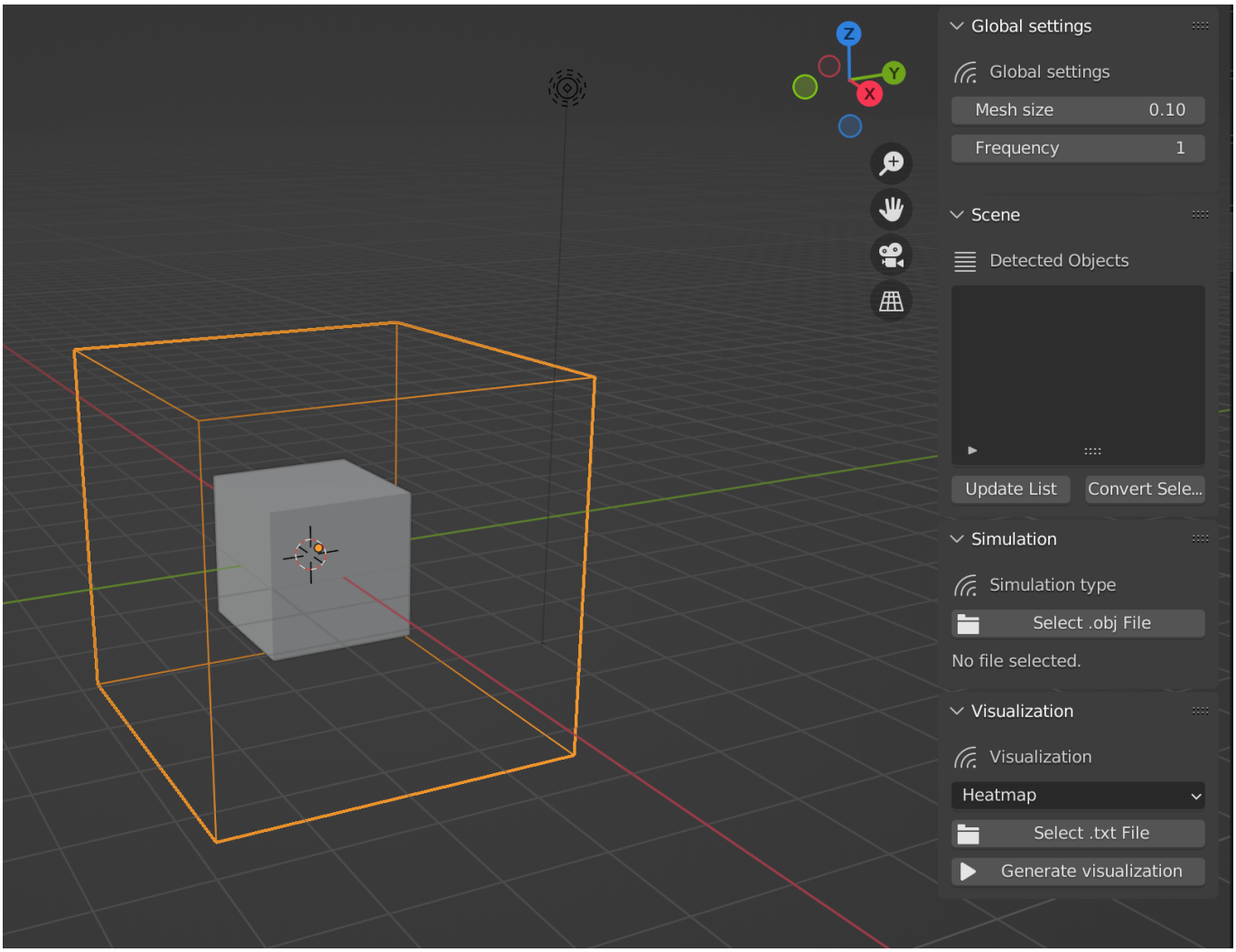
1. Open the CEM tab in the Layout workspace in Blender. You should face the following screen :



2. You can noz configure the global settings of the addon. Mesh size determines the size of the voxels. Frequency will be used later on the simulation.

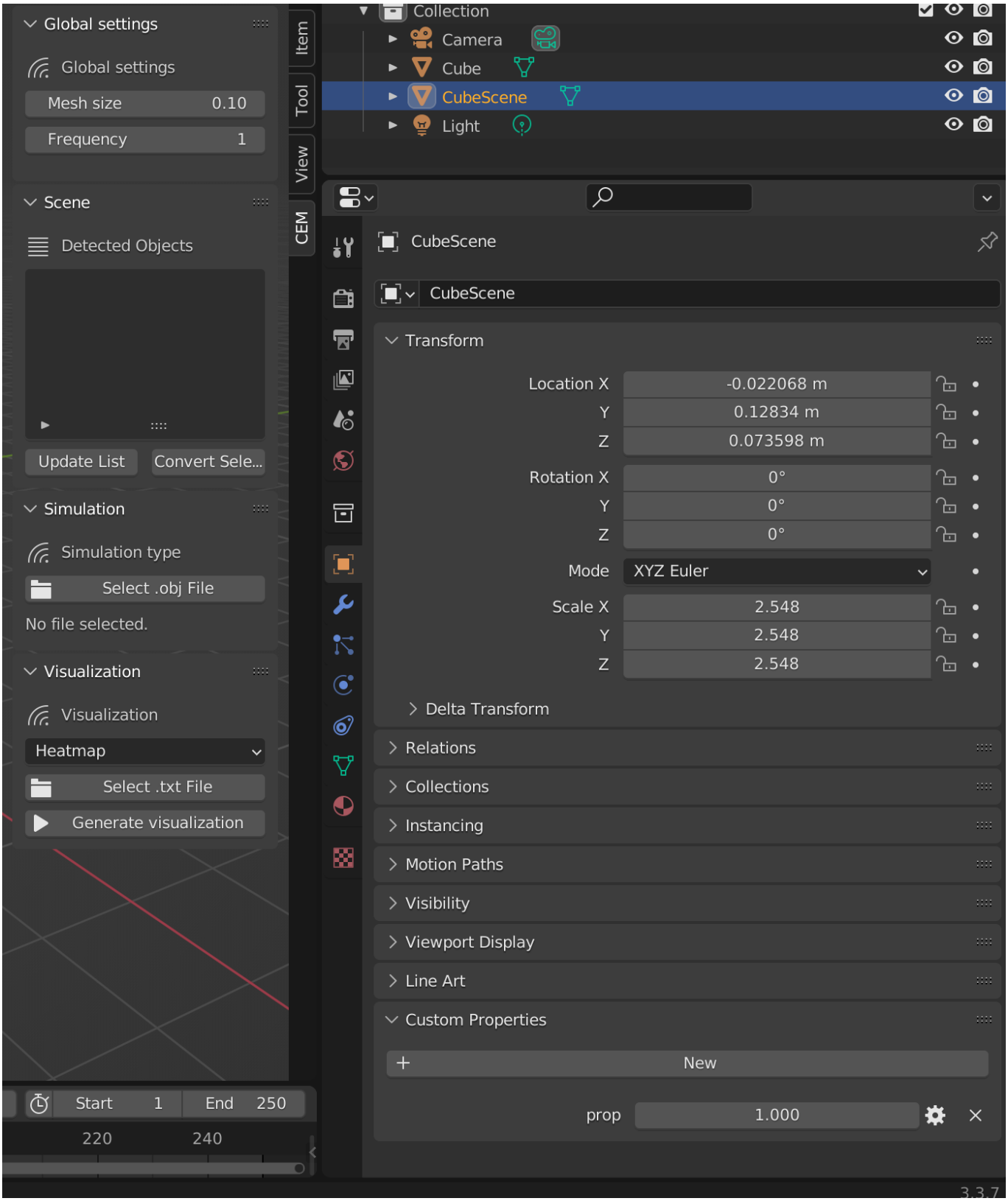
3. Create the cube Scene. This cube will be used to define the simulation space. You can move it, scale it, rotate it.

You should have now the following screen :



4. You can noz manipulate the scene and add objects to the scene. Be sure to scale, rotate and move the cube scene to fit the simulation space.

5. The Object Properties Tab is where you can add properties of the object. You have the option to add multiple properties to the object and configure them.



6. You can now simulate the scene. You can choose the simulation type and the frequency of the simulation. The simulation will be saved in the project folder.

7. Lastly you can modify the visualization of the simulation. You can choose the type of visualization and the frequency of the visualization. The visualization will be saved in the project folder.

Contribution

Available visualization

Several visualizations are included in the add-on. There are *Unidimensional*, *Bidimensional* and *Tridimensional* visualization.

Available visualizations are :

Visualization	Link
Heatmap	link
3D Scatter plot	link
3D Surface chart	link
Bubble plot	link

Add a visualization

In order to extend available visualization, it is necessary to modify 3 different files.

First, you need to edit the file plot.py.

In this file you need to create a class for the type of visualization you want to add. The new class **must** extend the abstract class **AbstractPlot**. Each plotting class should contains at least one method called *create_plot* which contains all the steps needed to create the plot.

Here is the class template:

```
class ClassName(AbstractPlot)

    """
        x, y, z parameters depend on the visualization (1D, 2D, 3D)
    """
    def create_plot(x, y, z):
        #plot instructions here
```

Once you have finished the class, you need to add the new type of visualizaiton to existing ones.

To do so, go into the file init.py.

When you are in the file, you need to edit the variable `bpy.types.Scene.visualization_types`.

Edit by adding your plot in the same format as other: `('PLOTNAME', 'Plotname', 'Plotname visualization')`

Finally, the last file to edit is visualization_opt.py.

In the file, find the class `class VISUALIZATION_OT_generate_visu(bpy.types.Operator)`, then in the method `execute`, find the *if* condition on `context.scene.visualization_types` and add a *elif* condition as it is done for the other options.

Here is the template:

```
elif context.scene.visualization_types == 'PLOTNAME':
    plot = ClassName() #it is the class instanciation
    df = read_csv(context.scene.data_file_path) #reads data
    plot.create_plot(df["x"], df["y"]) #call the method you coded before,
add here all needed parameters
    plot.save_to_png(image_path)
```

Add new global settings

You can configure the global settings of the add-on. Mesh size determines the size of the voxels. Frequency will be used later on the simulation.

These settings can be easily modified in the file `settings_opt.py`.

```
class GlobalSettings(PropertyGroup):
    mesh_size: FloatProperty()
    frequency: IntProperty()

    # for example here is the new settings to add
    custom_settings: IntProperty()
```

You also need to edit the class `OBJECT_PT_parameters_section` in the `draw_parameters_section` method.

```
layout.prop(global_settings, "mesh_size")
layout.prop(global_settings, "frequency")
# the settings needs to be added in the layout
layout.prop(global_settings, "custom_settings")
```