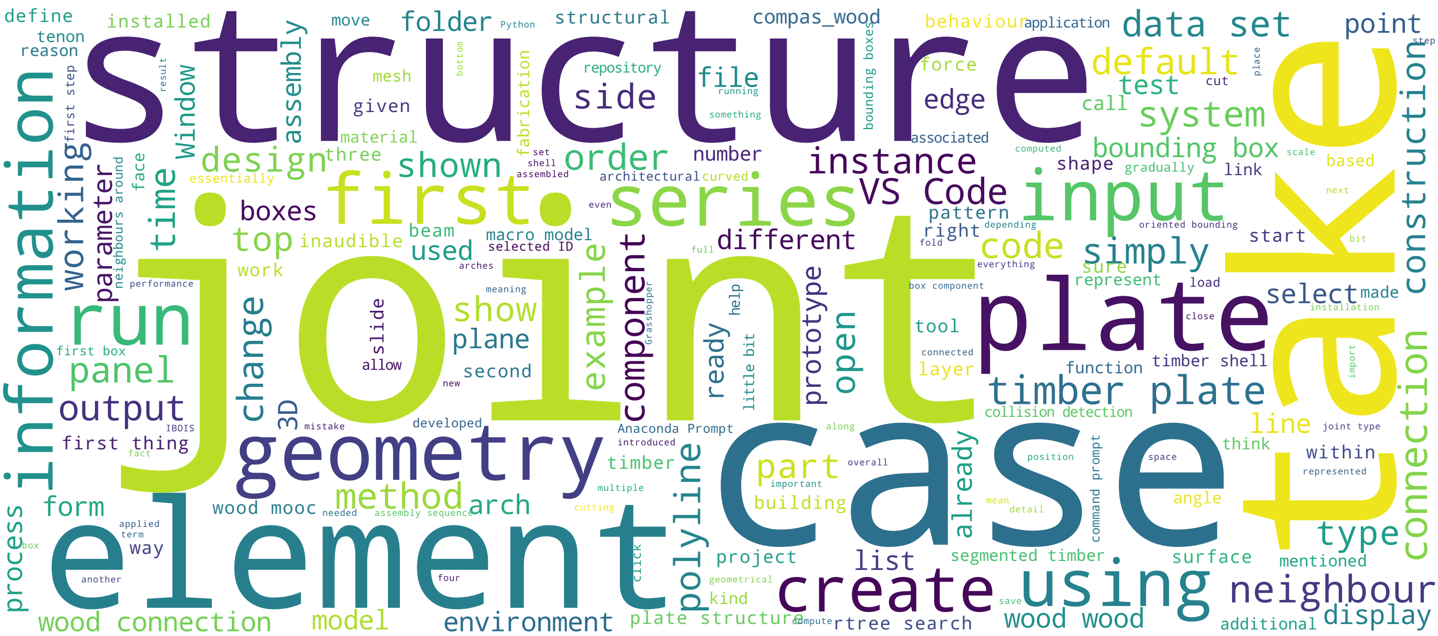
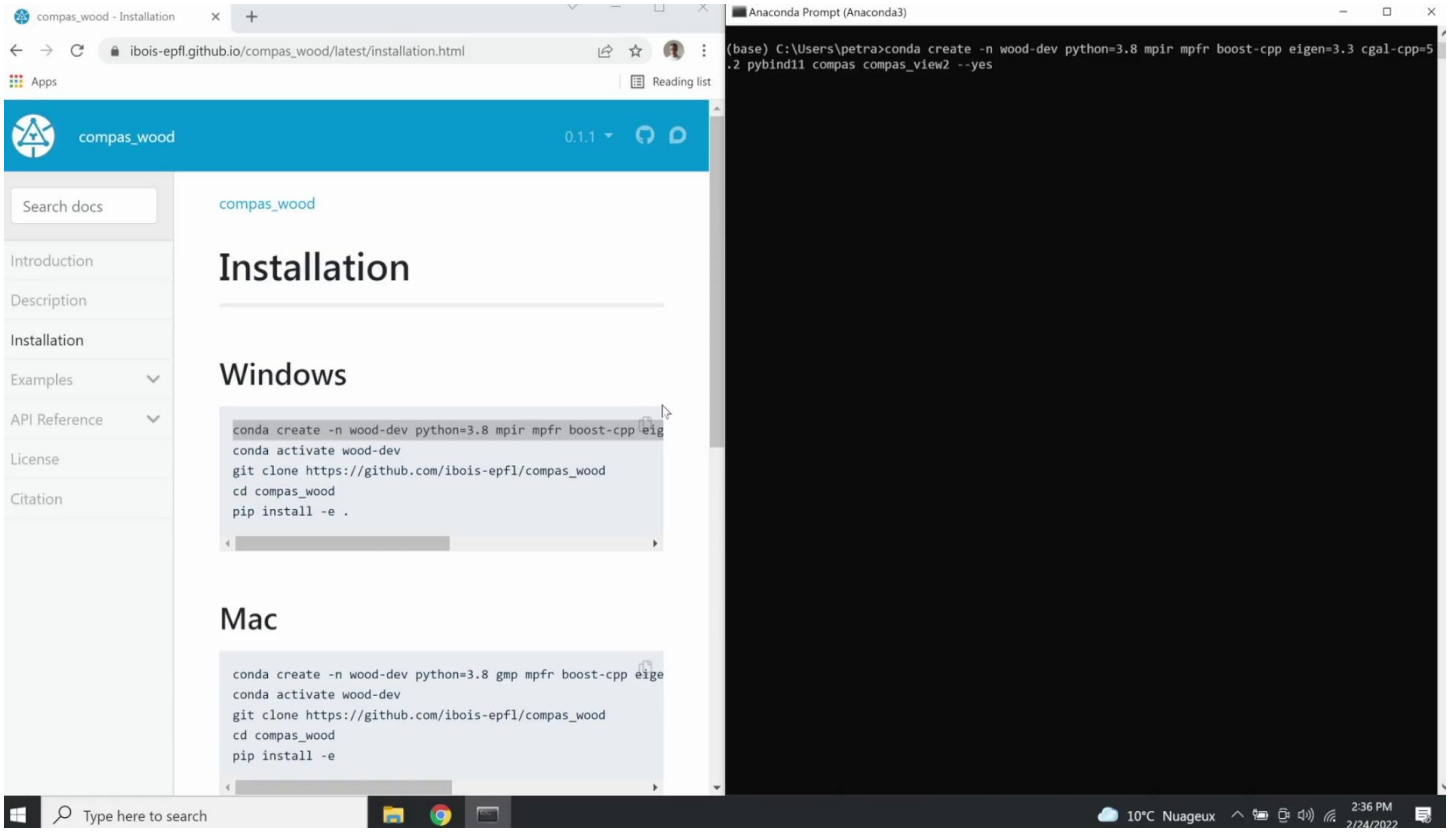


Search MOOC

Video

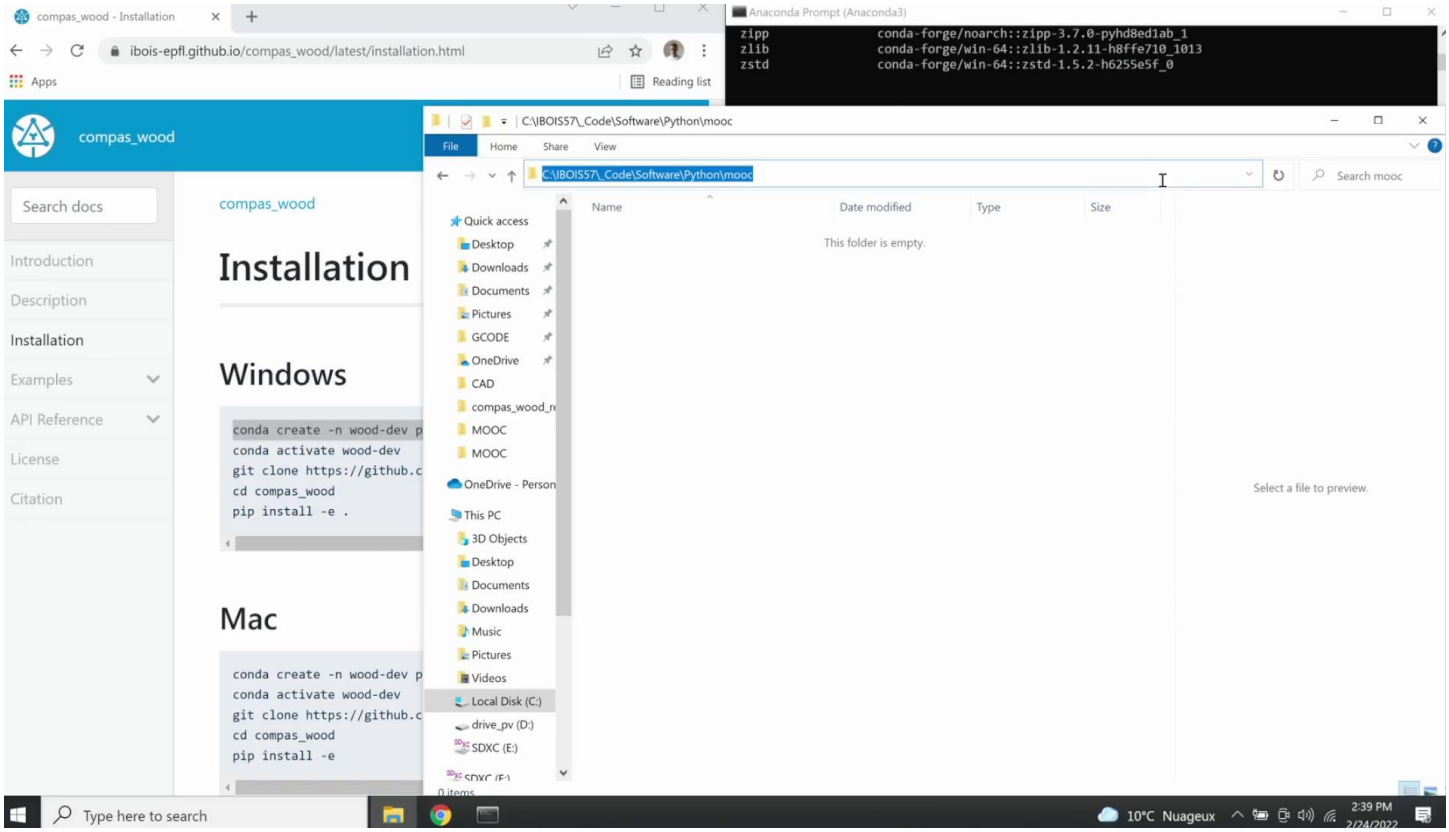


The first thing what we need to do is we need to clone repository of the main code and try to run it in VS Code editor. For this, as I mentioned before, we need to be ready to be installed the Anaconda Prompt and also VS Code. If you already do that, we can do the installation process of the compas_wood package. First thing what you need to do is to open the command prompt. We can do that like this. We can go to Start, if you're on Windows, then go to Anaconda Prompt. Then you can move one window to one side, another window to another side. If you are in the website of the compas_wood, you can go to installation. In installation, there are a few cases we talked about trying on Grasshopper version in the last lecture, and now we are going to talk about installation for Windows or for Mac. There are slight differences in the way you configure the conda environment. In my case, we on Windows. What I would going to do, I would simply copy each line by line into the Anaconda Prompt and try to run it. The first thing is that you need to set up the environment of conda. Let's copy this first line. You can change the environment name as you would like to have. It doesn't matter.

Notes

Summary

0m 04s

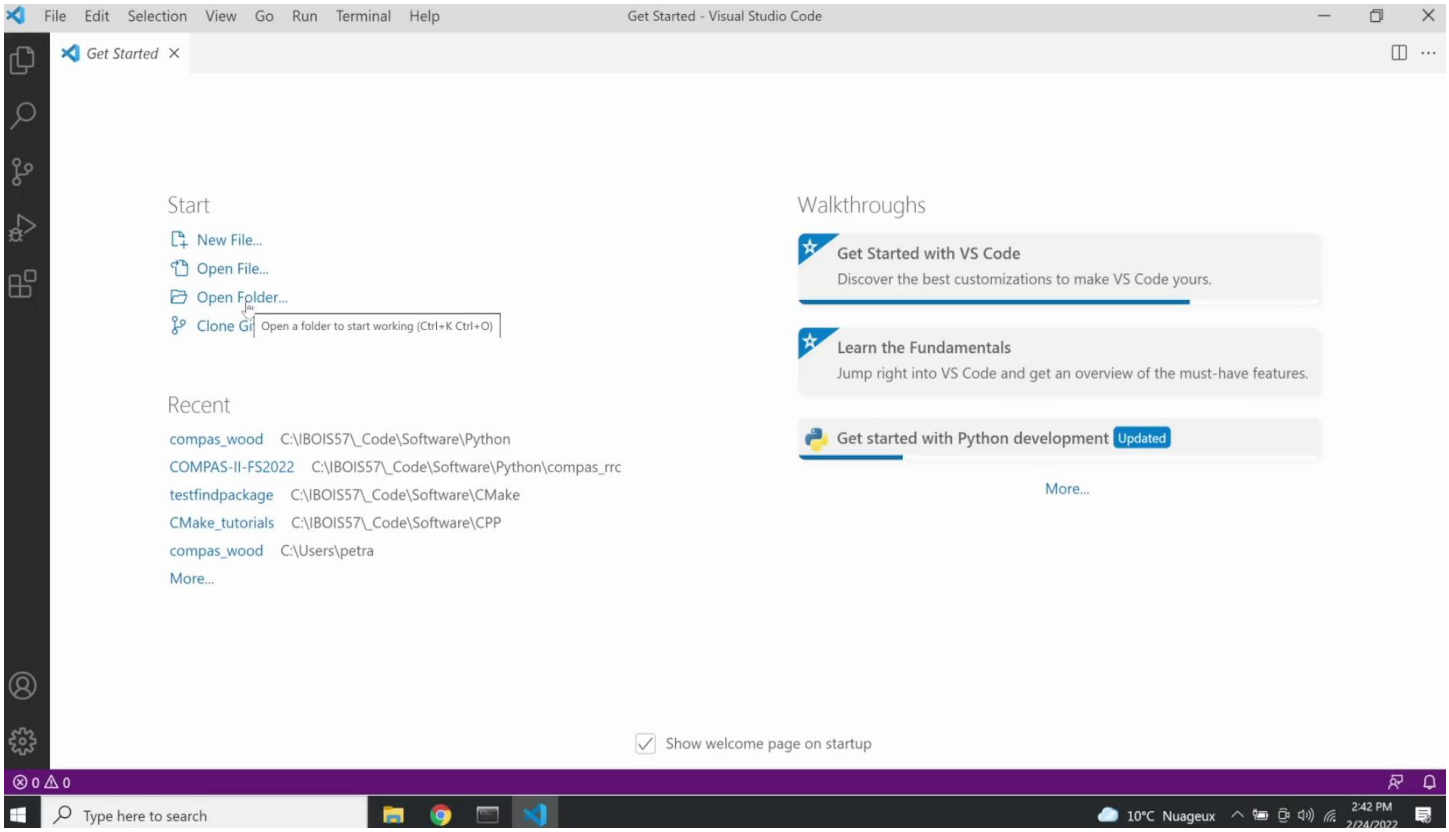


For instance, in my case, let's say wood mooc. Say, something like this. This environment will install special libraries like Boost, CPP, C++ Libraries, and then Eigen, then CGAL, pybinds, compas, and compas u2. It helps you to set up environment with all these things. You click Enter, and then you need to wait quite some time until it is installed. It's usually quite a long process until it is ready for the next line. Okay, if everything is ready to go, the environment is set up. By default, it is deactivated. As you can see here, you are in the base environment by default. Therefore, you need to change it to the wood mooc environment, so let's do that. Let's type conda activate wood mooc. Now what we need to do, we download a series of libraries that you can run in Python. Then you need to actually clone the compas_wood repository. For this, you need to have a specific folder in case you would like to be tidy and clean. In my case, I have a special folder, where I keep my codes, and I create a folder called mooc for today lecture. I copy this link, I can go to this directory, see that I am in the mooc directory, and then I will clone the whole directory in this folder.

Notes

Summary

1m 36s

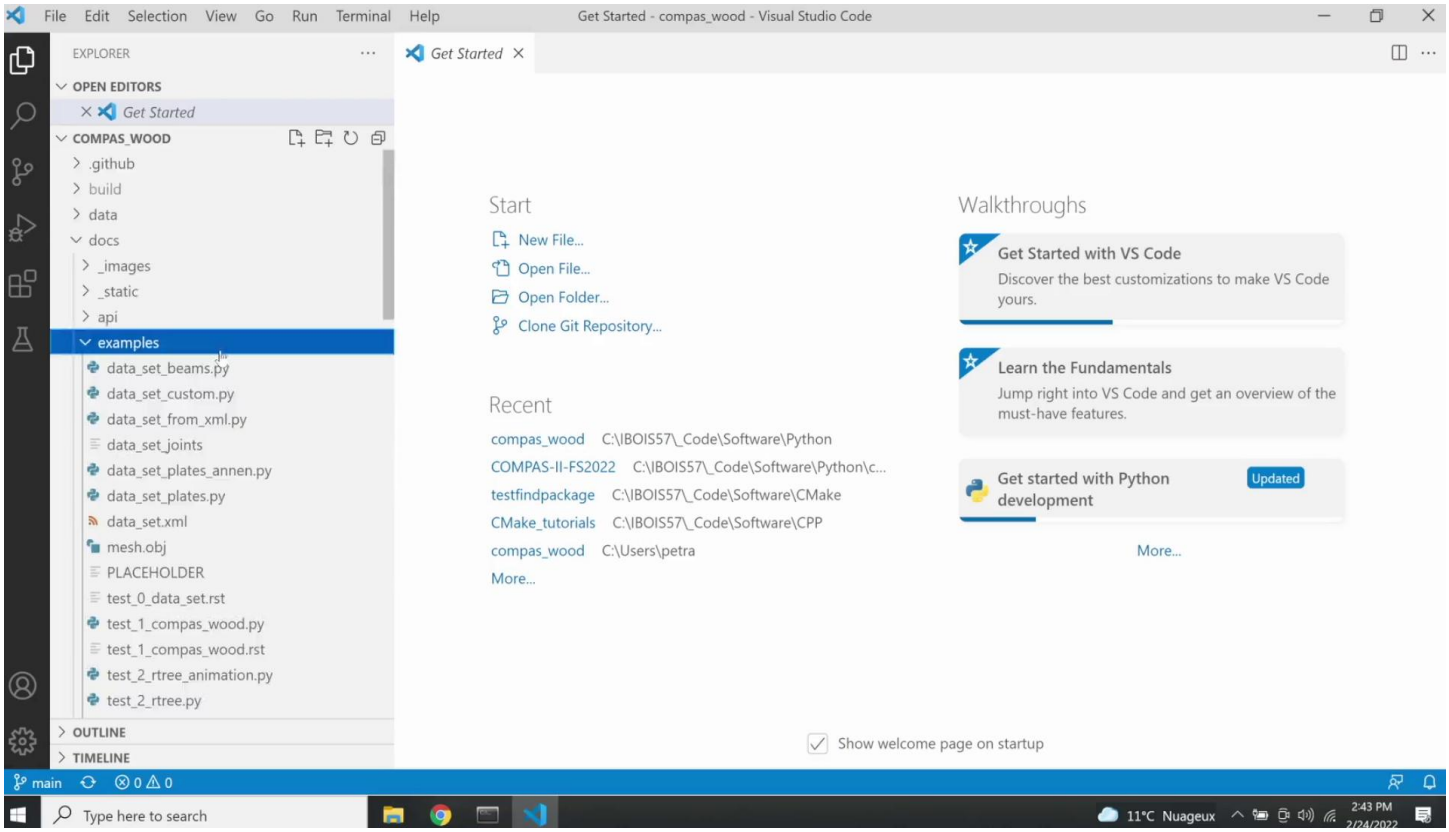


It will again take some time because you are downloading the full development repository from GitHub. Afterwards, we are going to actually go to this repository and try to compile the code that we can run it directly in VS Code. Okay, this stuff was quite fast, and now we can go to the compas folder. So then I'm writing compas_wood. See the compas_wood is actually just goes one directory there. The reason it works is that we created a folder called mooc. Then in this folder, the full repository of GitHub was downloaded essentially. Yu have the full file of all the files of this repository, test development, and so on. Now what you're going to do is you're going to compile the code because it's a C++ code, it has to be specially compiled for Python, so we write pip install minus e. This process again takes a little bit of time because it is C++ compilation. Once we're ready, we can go to the VS Code. Okay, great. If everything is successful, we can start working. I will open a VS Code that I hope that you installed before. So go to VS Code. Then by default, you should have a white page or you have already your project open.

Notes

Summary

3m 29s

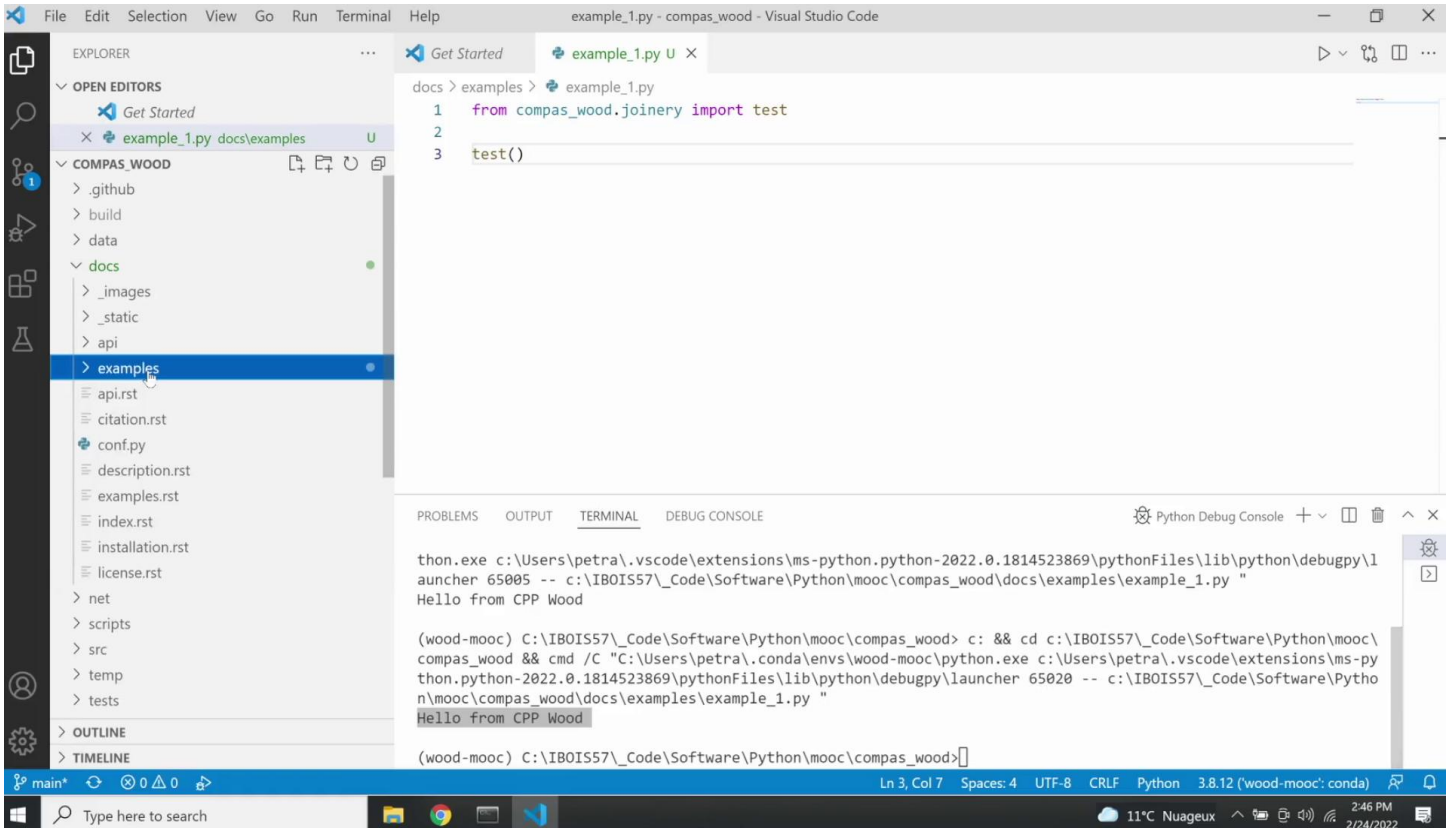


If you have a project open, just close it and open a new folder, and just simply navigate to the folder that you actually downloaded the files. This whole thing, copy the link here and select the folder. VS Code is going to open the full folder and will essentially allow to run the code that was developed for the segmented timber shell of applications as well. For the first time, you need to do two special things. As mentioned in the slides, you need to select the conda environment to ask you to select the interpreter. Then we can click on wood mooc. This is the name that we created initially. Then what we need to do is we also need to be sure that audio code for Windows users only, you need to be sure that it's running on command prompt. For Windows, you can click Control Shift P, and then select the terminal, and then command prompt. Again, this only is needed to do only for Windows users, but not for Mac users. For Mac users, it should be correct and fine. We are going to be working in the docs folder. By default, I create all the examples that I can think of and during development in the examples folder. You can already test different examples that is here or will be here in this folder.

Notes

Summary

5m 19s

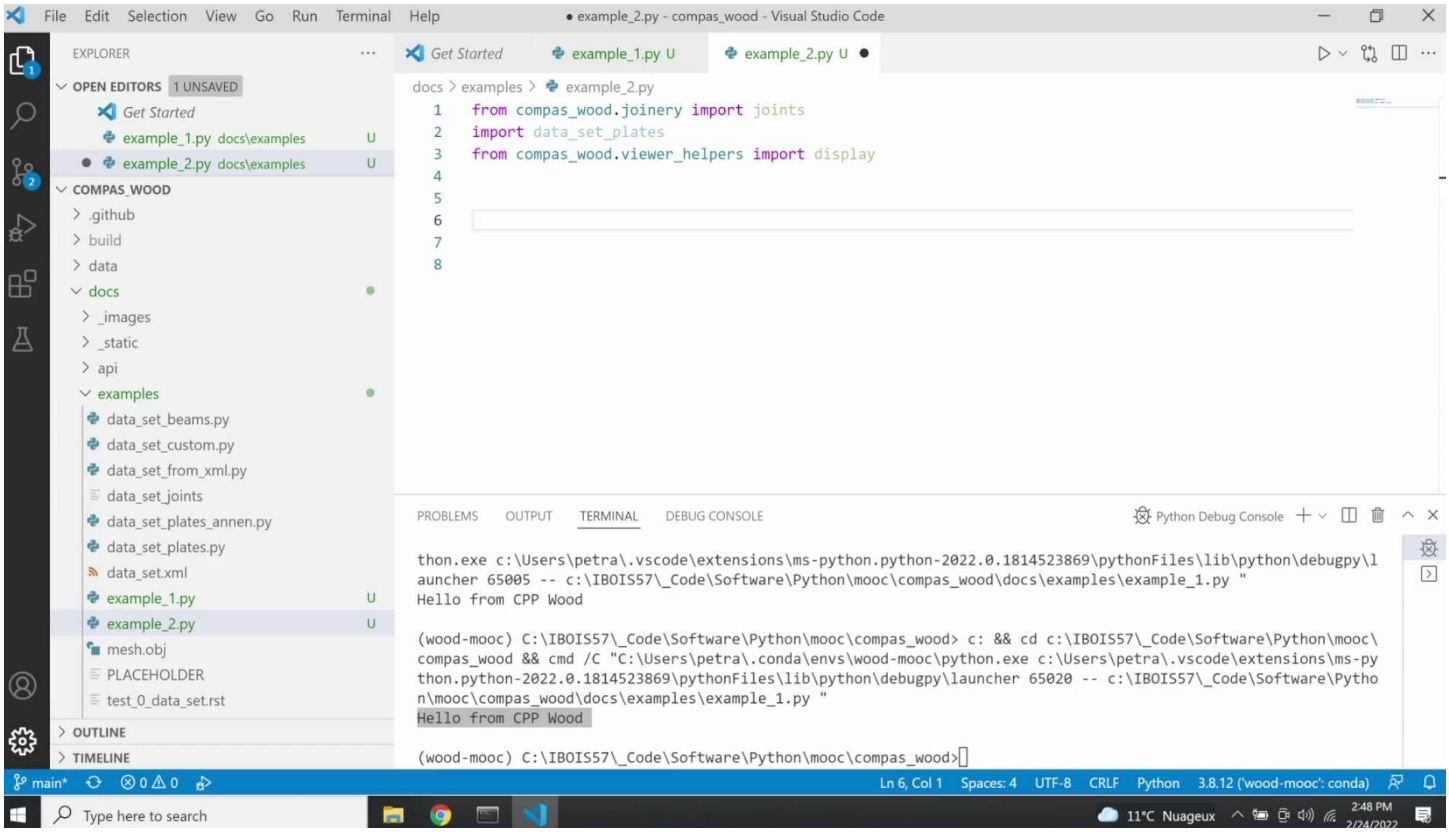


But the first step that we are going to do, we will try to test if compas_wood is actually running on your computers. Let's create a new file, and let's type, let's say, example_1.py, just an example file. The first step what you need to do, you need to reference compas_wood and run a test function. Let's import it, let's import compas_wood, and then we can import joinery. We can say that we don't need to import all the functions, just save it from compas_wood import test method. Let's test if it's actually working or if you need to install something else or not. Let's just run this function. We can run without debug. By default, I'm using formatter black. If you need, you can install it, if not, just skip it. It takes some time until it installs, but after the step, the whole installation process is actually ready. Let's test again this method. Let's go to Run and then Run Without Debugging, and you see that it's actually printed Hello from CPP Wood. That means that the repository is working and we can actually move on to the practical examples, and show you the few definitions that can be used within this repository as a complete basis. I'll create another file called example_2.py.

Notes

Summary

6m 59s

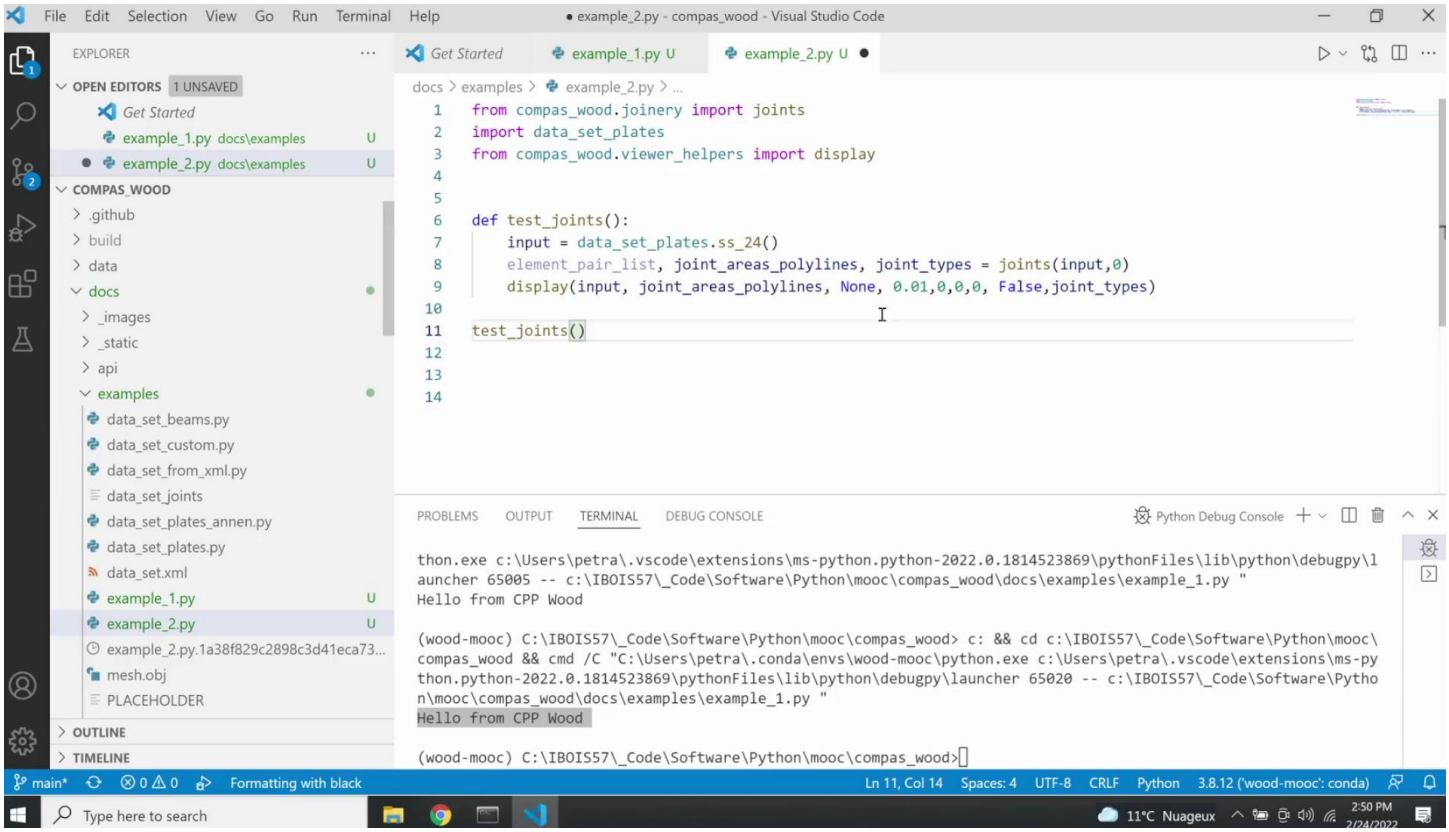


What we are going to do right now, we actually going to test simple segmented shell in this Python environment, and then we are going to use a few functions that can be visualised because by default, we installed the compas view, which extremely is useful for displaying a geometry in 3D. Let's start. I will import compas wood method just going from the joinery namespace, import joints. For sure, it has to be from this method. Now we need to also input some data sets. In this case, I simply have one file that has a lot of data sets, and for this case, we're just going to test that. We use data sets plates, which has hardcoded simply a list of power lines and the property so we can debug and test the different methods during the development. Then we also need to display the geometry in 3D. Therefore, we are going to import the viewer from compas_wood viewer helpers import display function simply to display a series of polylines. Now we create a simple function to actually execute the code. Let's create a method called def.test_joints, and then we need to have an input, which will be data sets, plates and ss_24 will be the segmented timber shell in this case without other ones.

Notes

Summary

8m 55s

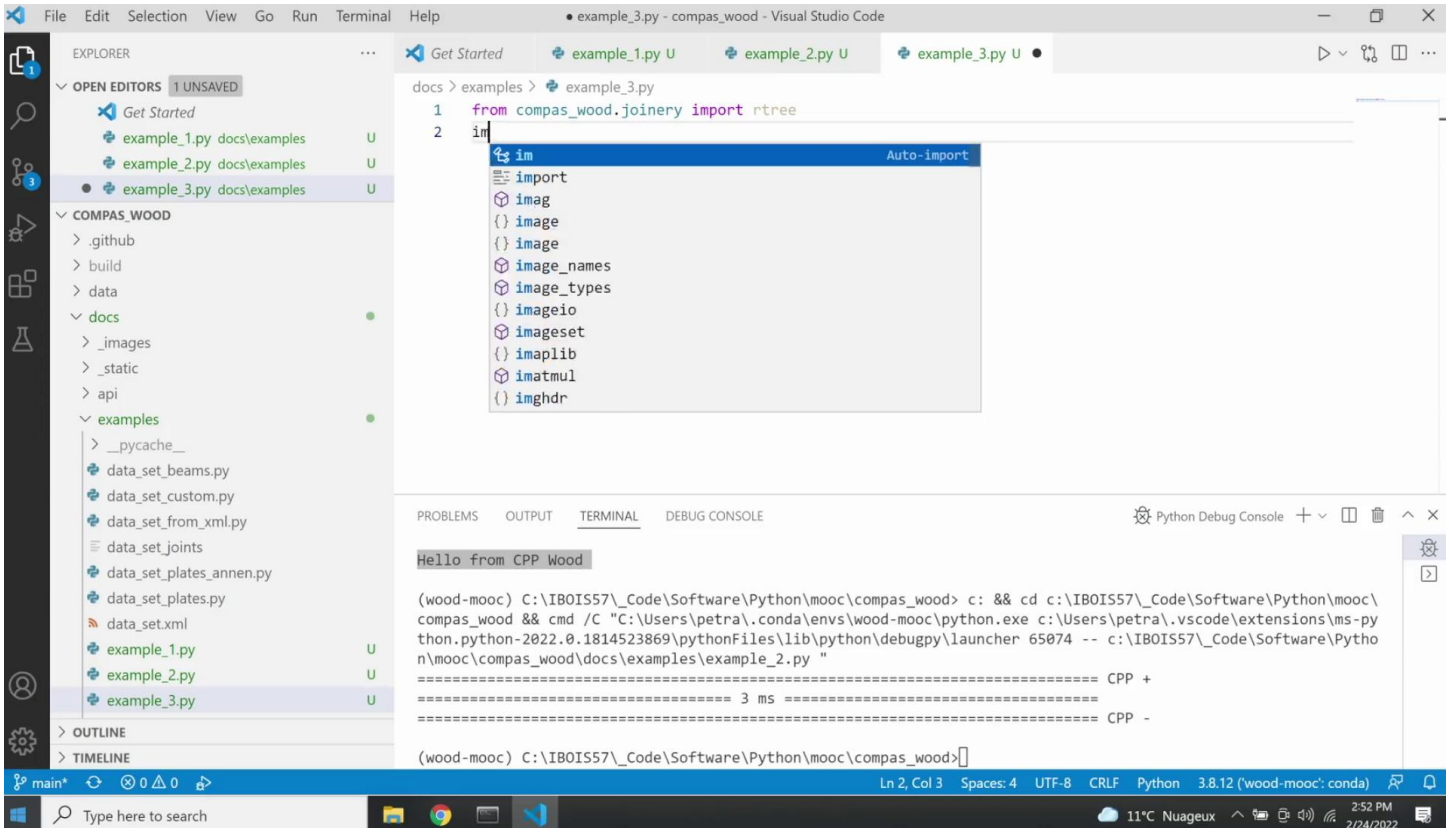


As you can see here, if I click that you can actually see a lot of different cases. I will just use to be consistent for the segmented shell only. Then we need to call a method joints, which has an input and by default, we are going to use the search method zero, which actually says that we're going to try to detect face-to-face connection. Plain-to-face detection is more for cross joints. Let's take the first one, the zero. What this method outputs is a series of neighbours, simply indices and the bounding boxes for visualisation, so a couple of three items. Then we can say that these three items are called this: element pair list, then joint areas polylines. Actually it outputs a series of element pairs, where the connections are and the joint types in order to colour those information in 3D. Then finally, we simply need to display the information. We recall a display method from the viewer display input, joint area, poly lines, and we try simply to use the default values as they arem something like this. Then finally, we can call this method as joints in order to run. This is a simple example that will test if it actually has no errors.

Notes

Summary

11m 00s

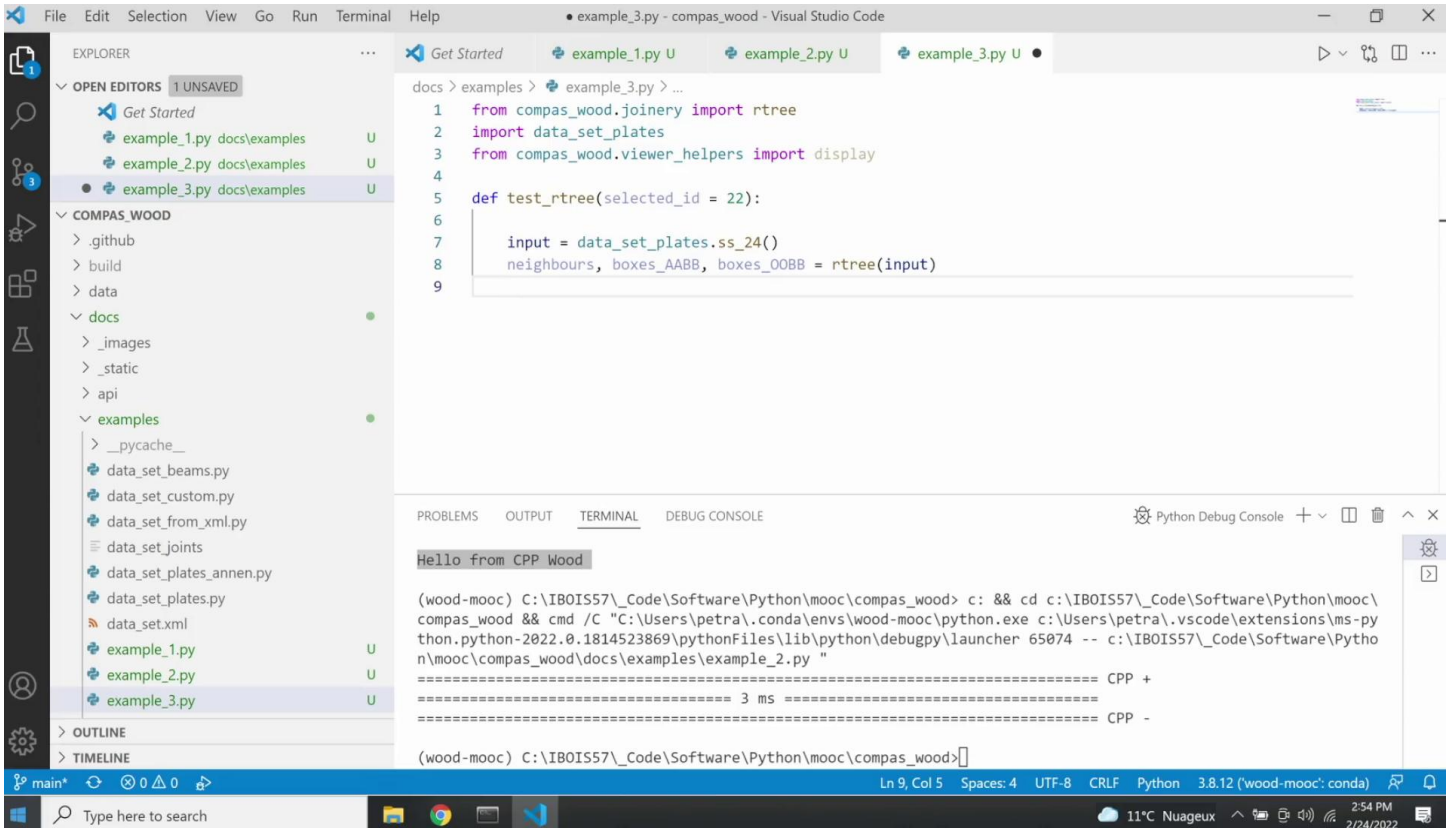


It's working, and I see that as in the last lecture, we had those plates that represents each plate object, and generated connection areas, blue areas simply says the draw connections side to top. We don't define here what type of connections because there could be multiple different connections here as well. Then we have those red areas that essentially says that there could be a possibility to define [inaudible 00:13:27] joint, which is a site out of plane connection. This was the first example. The next example will explore the arch research method that is independent of the joinery solver. But I think it will be interesting to see how the neighbours between plates can be detected without using any underlying graph method. It's simply by using the collision detection. Let's create the third example called example_3.py. Again, what we are going to do is from compas_wood and joinery import rtree, so it's a C++ implementation in the Python rtree search. Then you're going to import again the data sets of plates because we are working with the plates at the current stage, and then we are also going to import the viewer.

Notes

Summary

12m 57s

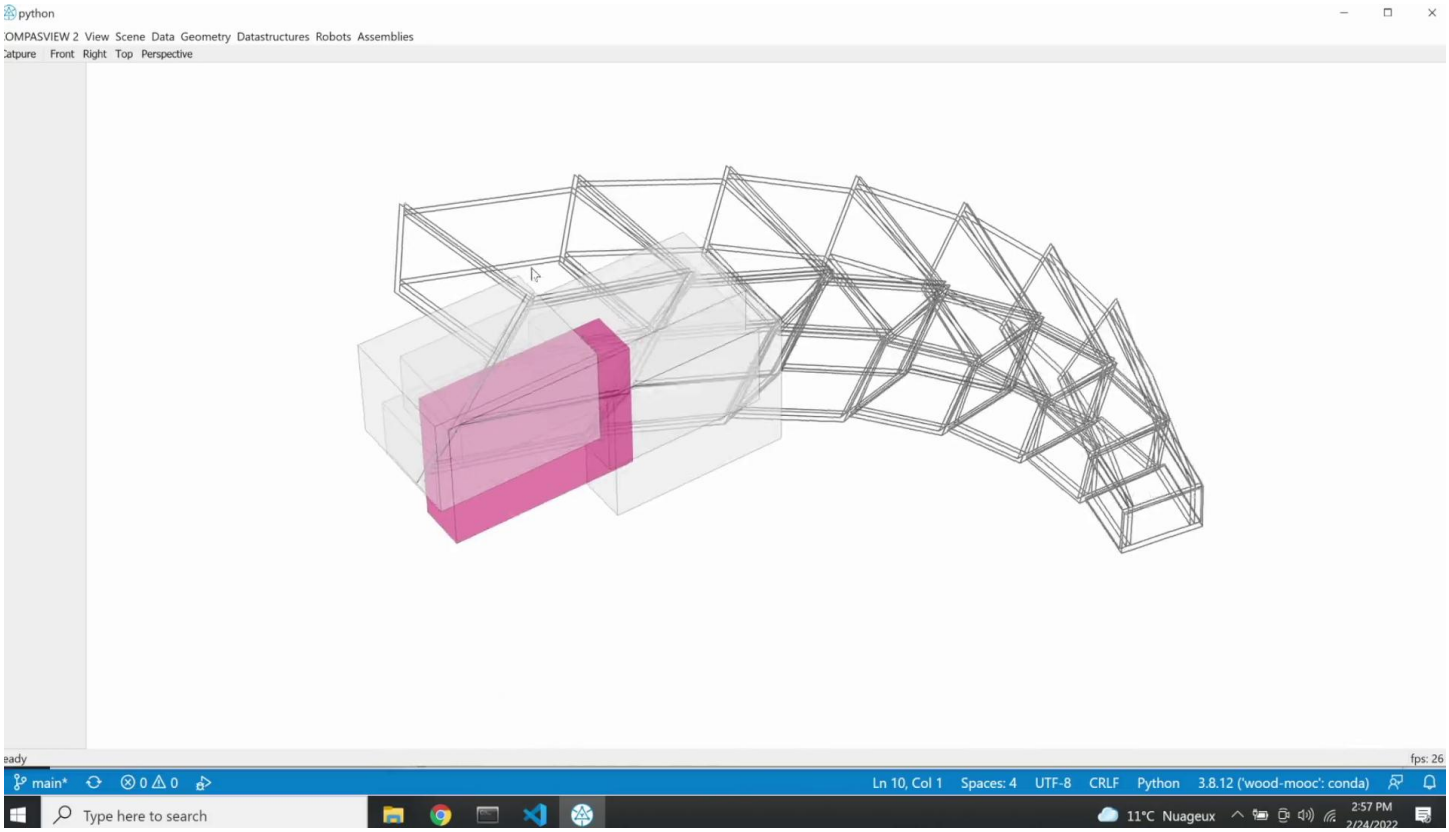


You can also use a compas default viewer, which is viewer 2, this small function that helps to clear display geometry based on compas_wood joinering methods. Then let's again define a function that has a series of operations to call the rtree search, and then visualise it in 3D. Let's write test_rtree, and let's just give an input of ID which element of the neighbour we want to display. For instance, we can say that selected ID would be 22. We would like to display a neighbour the 22nd plate. The first thing, as before, we need to get the input information. Let's write input equals data set plates ss 24. Now we can compute an rtree from this geometry. For instance, we can call rtree on the input, and then we would see that the rtree search actually outputs also a couple of three elements. It outputs the neighbours, it outputs bounding box that is access-aligned, and then bounding box that is object-aligned. Let's define those inputs. The first one are neighbours, the second one is boxes.AABB and then boxes object-oriented, something like that. Then we can actually display a geometry. By default, there are boxes of all individual plates. Let's just select the first one and the neighbours of that.

Notes

Summary

14m 37s

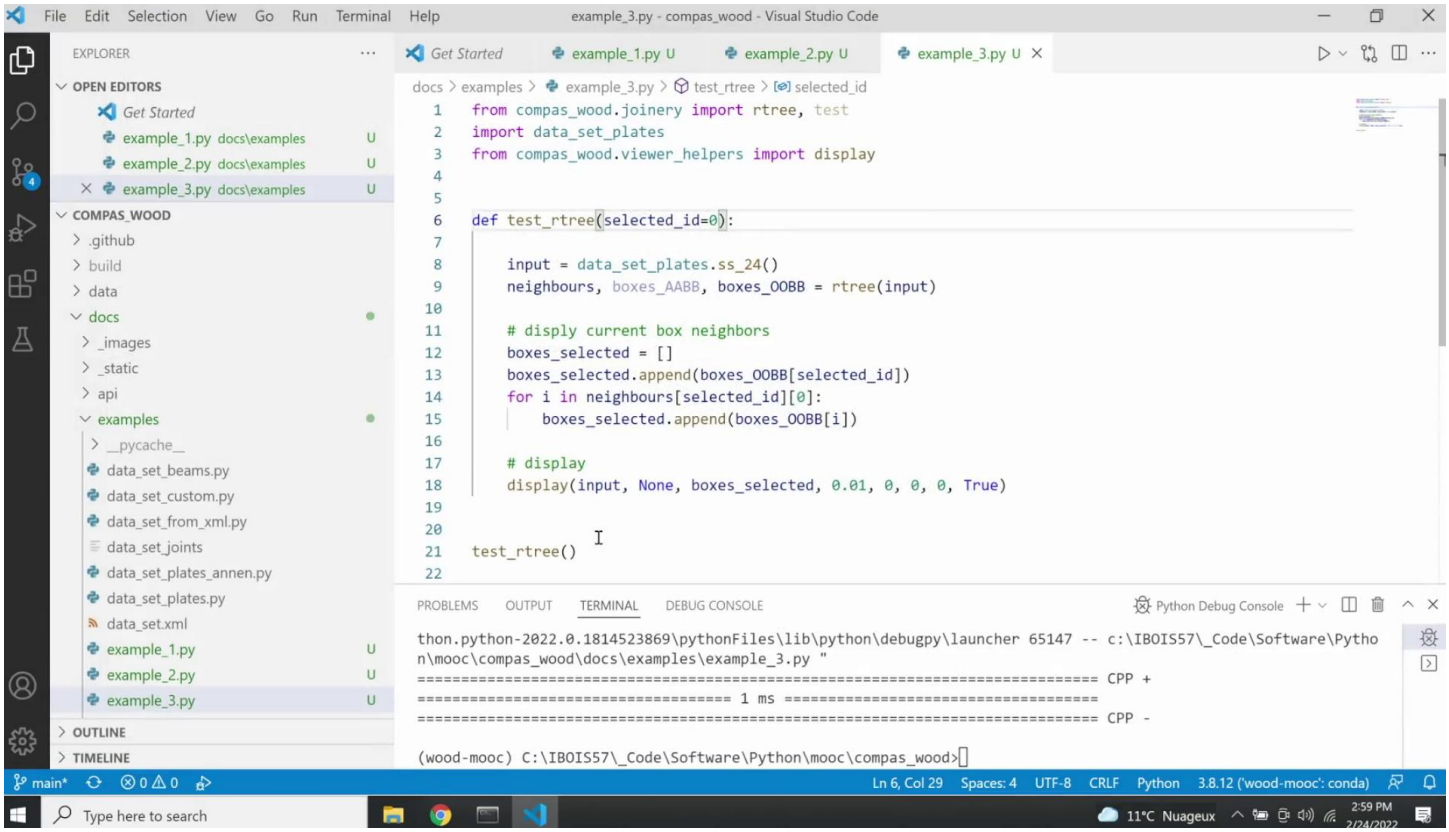


Let's write a very simple loop for actually displaying the geometry. Let's say box selected. Let's create just an empty list, and I would say that display current box neighbours. Then we actually will take the box, select it, and then add one element from the box ABBB, which is going to be selected ID, so we are going to take one box from the computed bounding boxes to the selected ones. Now we need to take the neighbours, Let's trade for i in neighbours selected ID. Since it's a num py re, we need to take through the first list, and then we can actually gradually add all elements to the box list, so selected, and then we can take that boxes ABB, and then take the index of that box. Now we can simply take the display case and try to display this geometry a series of meshes or boxes. What we'll take, we'll take the input, and then we take the box meshes. Then we gradually, again, output some information, and then try to call the first box. We can actually output all the information, but here, we just test the visualisation part and test rtree, and see if you don't have any typos, any mistakes. Let's try to run. Okay, so no mistakes. That's good. You see here that we computed bounding box around one plate.

Notes

Summary

16m 38s

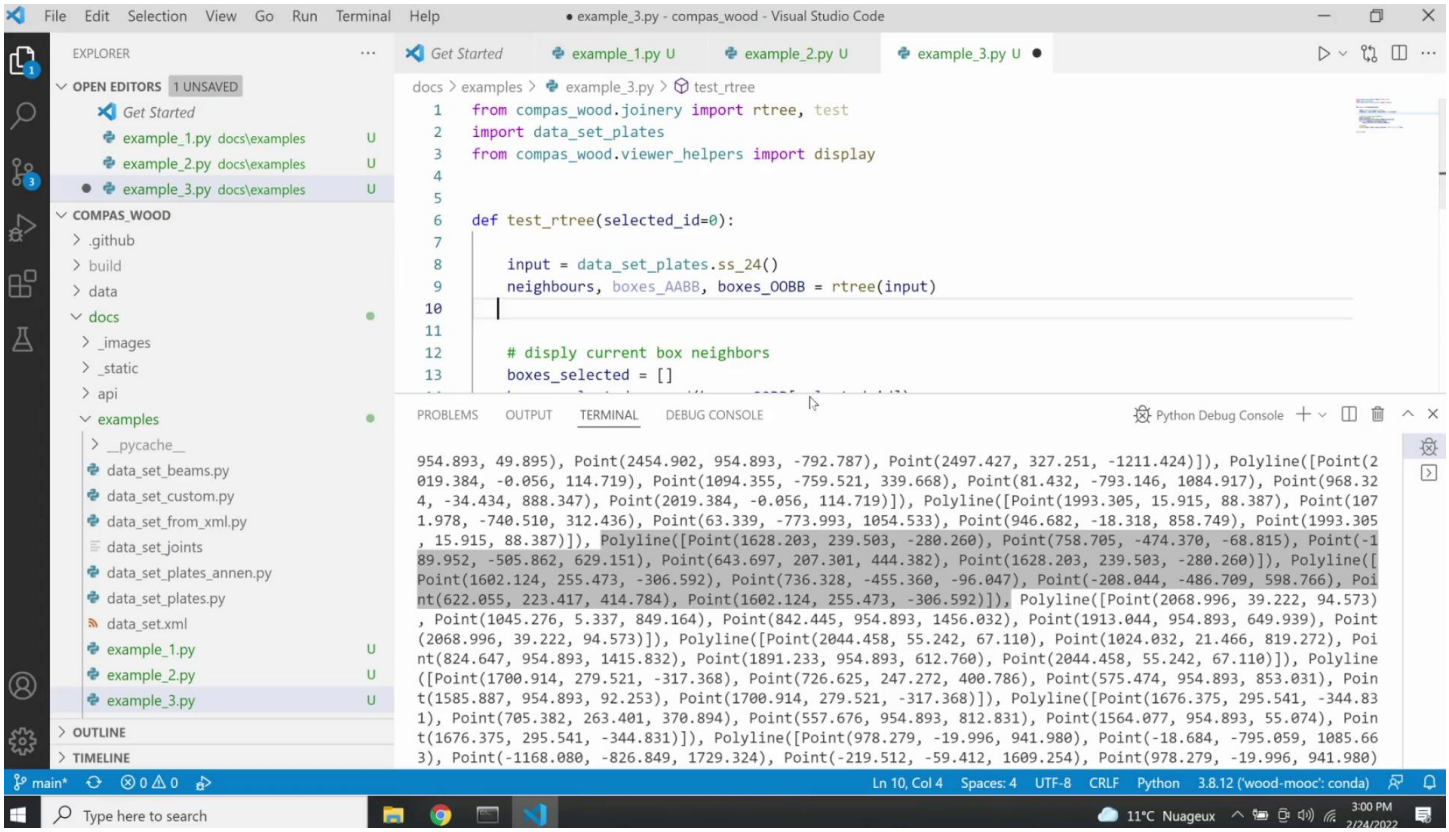


You see that it's a bounding box of this inclined plate and then neighbours around it were actually selected. The issue with this rtree search is that it can slow a little bit, be a bit more slower because we are not computing the object-oriented bounding boxes. You can see that can be multiple collisions even that there's actually none in some other cases. For this reason, it is actually sped up by using the oriented-bounding box collision. For instance, to visualise that the actual implementation is using oriented-bounding box, I would save it and run it. You can see that the actual collision detection is made around these object-aligned bounded boxes. If you want to take another box, you can run it again, let's say, let's take the first box. Then it computed neighbours around this first box, all these grey neighbours around it. This is the very first step than actually doing the code detection of joints based on face-to-face intersection that was shown in the last example just to show how actually the graph data structure is computed here just based on collision detection. If you would like to actually see what kind of neighbours we got, we can simply print the neighbours.

Notes

Summary

19m 05s



I would just simply print this information. You can see here that these are actually the neighbours of each individual element based on how many collisions there are. If you just reiterate what is the input, the input of this information is actually a series of polylines. This is the standard compas polyline input. I just close it for now. Here you can see that there's just a polyline list, and essentially, each plate, again, is represented by a list of pair of polylines. Okay, this was a second example.

Notes

Summary

21m 03s