IN4089 - Data Visualization - Volume Visualization - Debugging

Debugging with Breakpoints (Visual Studio)

Breakpoints are a great way to debug your code. If you set a breakpoint in your code and run your application in *Debug Mode*, the application will stop at the corresponding position in the code and you will be able to inspect all variables that are available in the current scope.

Visual Studio Code

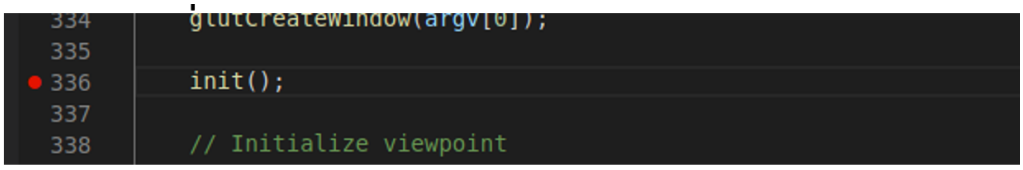

Figure 1: To set breakpoints in your C++ code. Click left of the number of the line that you want to set a breakpoint on.

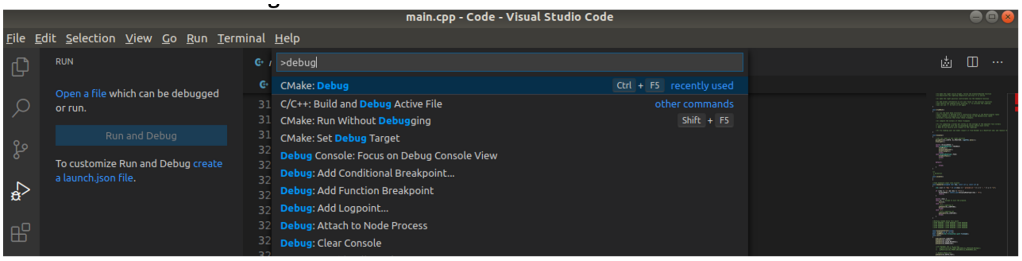

Figure 2: To run the project and attach a debugger, use the Command Palette (`Ctrl/Cmd + shift + p`) and search for *CMake: Debug* OR use the shortcut `Ctrl/Cmd + F5`.

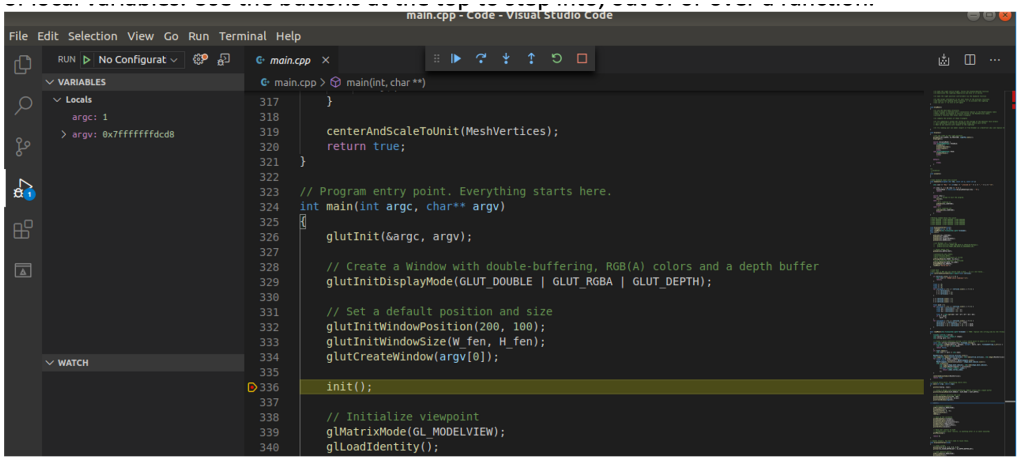

Figure 3: When you run your code it should now stop at the breakpoint and show you the values of local variables. Use the buttons at the top to step into, out of or over a function.

Visual Studio

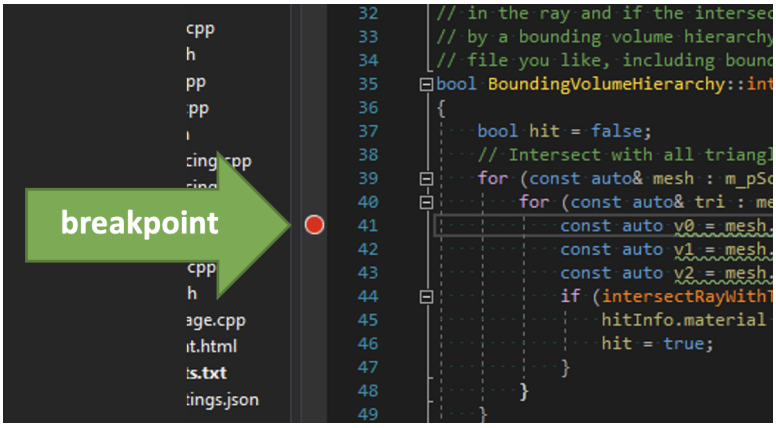

Figure 4: In Visual Studio, you can set a breakpoint by clicking on the light gray bar to the left of the line numbers. A red dot will appear at your breakpoint. Usually, you set a breakpoint before the program runs, but it is also possible to set or remove breakpoints while the program is running as long as you are running in Debug Mode.

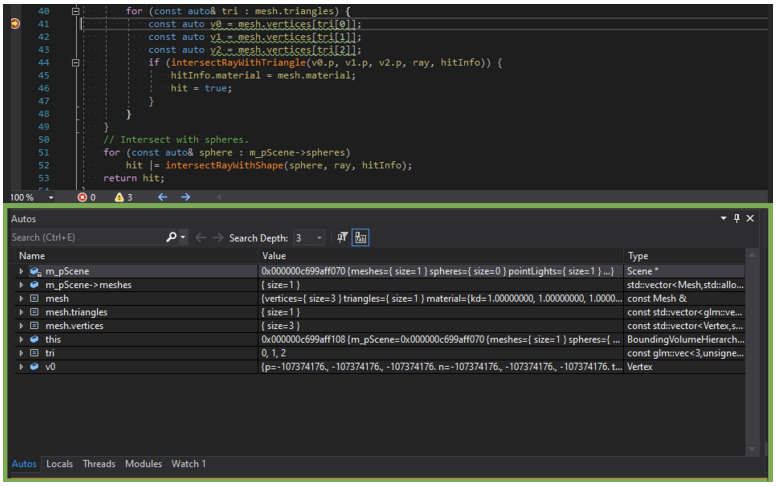

Figure 5: **Running the program** Now run the program. For the best debugging experience, make sure your compiler is set to debug (x64-Debug) mode. If you have just built the program the first time this is already the case. Below we describe how to add *Release Mode* to the framework to run the app faster at the expense of optimal debug capabilities. We run the program by pressing the green triangle play-button When the program hits a breakpoint, a yellow arrow will appear at that breakpoint (red dot). At the bottom of your screen, you should see a window with *Autos* which stores all variables that the compiler thinks are useful to use. Use the tabs at the bottom to switch to *Locals* to view (almost) all local variables. This is the most important window to look at. It will show you the values of the variables at the current point in the program.

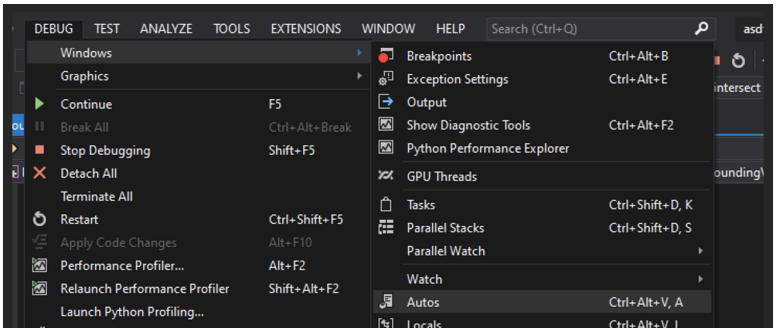

Figure 6: If the *Autos* or *Locals* window does not appear, you can manually open them by going to DEBUG ⇒ Windows ⇒ Locals. Additionally, you can hover over any variable in scope to see its value, or right-click it to manually add a watch.

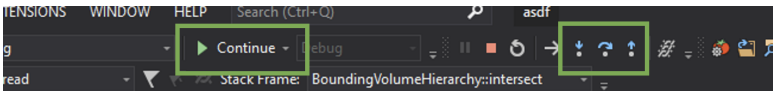

Figure 7: After stopping at a breakpoint, you can step through your code line-by-line using the F10 key or the buttons at the top. This is very useful to track what the changes the program makes to your local variables. Are the changes to the variables in line with what you expect? To continue running, click the big *Continue* button (same as the button you used to run the program.

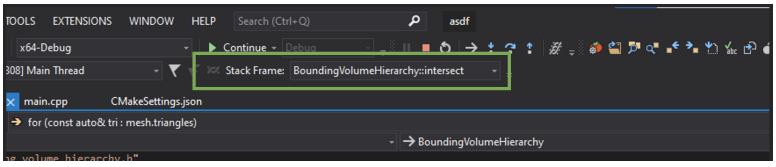

Figure 8: While at a breakpoint or a thrown exception, you can access the full call stack using the *stack frame* dropdown. From there you can navigate between the different function invocations in your call stack. When you select an invocation the (local) variables at that point in time will be come available in the Locals/Autos window. This functionality is especially useful when debugging an exception thrown in third party code (such as the standard library) or a stack overflow. Navigating to the calling code will help you understand the exception.