

---

## libfive

*Infrastructure for solid modeling.*

[Homepage](#) | [API Examples](#) | [Downloads](#)

**libfive** is a framework for solid modeling using functional representations.

It includes several layers, ranging from infrastructure to GUI:

- The **libfive** shared library contains functions to build, manipulate, and render f-reps. A great deal of work has gone into the meshing algorithm, which produces watertight, manifold, hierarchical, feature-preserving triangle meshes. The library is written in C++ and exposes a C API in `libfive.h`.
- The **libfive** standard library is a library of common shapes, transforms, and CSG operations. It is implemented in C++ and exposes a C API in `libfive/stdlib/stdlib.h`
- The standard library is parsed and used to generate bindings for both Guile Scheme and Python, for use in the REPL or as part of larger applications.
- **Studio** is a GUI application in the style of OpenSCAD. It uses the Python and Guile bindings and allows for live-coding of solid models. The interface also includes direct modeling, where the user can push and pull on the model's surface to change variables in the script.

## Other projects using libfive

### Language bindings

- Tovero: A 3D modeling system for Common Lisp
- **libfivepy**: A Python CAD library (work in progress)
- Bindings for Unity
- High level Rust bindings
- Unpublished Stanza bindings (email for details)

### Viewers

- Inspekt3D: Lightweight pure-Guile viewer
- PyFive3D: Lightweight pure-Python viewer (work in progress)
- C5H12 (Pentane): Lightweight C viewer

---

## Research

- Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces: a technical paper extending [libfive](#) to render on the GPU (reference implementation)

## Community

For [libfive](#)-specific discussions, consider opening a topic in the Github Discussions tab.

There's also a [libfive](#) subforum in the SDF User Group Discourse, which is a good place for general discussion of modeling with signed distance fields.

## License

(c) 2015-2021 Matthew Keeter

Different layers of this project are released under different licenses: - The [libfive](#) library, [libfive-stdlib](#) library, and Python bindings are released under the Mozilla Public License, version 2. - [libfive-guile](#) and [Studio](#) are released under the GNU General Public License, version 2 or later.

Contact the author to discuss custom development, integration, or commercial support.

## Compiling from source

[libfive](#) and Studio are compatible with macOS, Linux, and Windows.

## Dependencies

### libfive

- [cmake](#) 1.65 or later
- [pkg-config](#)
- Eigen 3.3.x
- [libpng](#)
- Boost 1.65 or later

### Guile bindings (optional, macOS and Linux only)

- Guile 2.2.1 or later

---

### Python bindings (optional)

- Python 3.7 or later

### Studio (optional, requires Guile or Python bindings)

- Qt 5.12 or later

When `cmake` is first run, it will check for all dependencies and print details of what will be build, e.g.

```
1 Checking dependencies:
2   libfive: ✓
3   Guile bindings: ✓
4   Python bindings: ✓
5   Studio: ✓           (Python + Guile)
```

### Mac

With `homebrew` installed, run

```
1 brew install cmake pkg-config eigen libpng boost guile python3 qt
```

Omit `guile`, `python3`, or `qt` to avoid building bindings and/or the UI.

Then, from the `libfive` folder, run something like:

```
1 mkdir build
2 cd build
3 cmake -DCMAKE_PREFIX_PATH=/usr/local/Cellar/qt5/5.12.0 ..
4 make
```

(adjust based on your Qt installation, and consider using `ninja` for faster builds.

### Ubuntu

`libfive` should build out of the box on the latest Ubuntu LTS (currently 20.04). If you find that's not the case, please open an issue!

Start by installing dependencies through the package manager:

```
1 sudo apt-get install g++ cmake pkg-config libeigen3-dev libpng-dev
   libboost-all-dev guile-3.0-dev qtbase5-dev python3
```

Omit `guile-3.0-dev` and/or `qtbase5-dev` if you do not want Guile bindings and/or Studio to be built too.

Building is similar as on Mac: clone the repository, then run something like

---

```
1 mkdir build
2 cd build
3 cmake ..
4 make -j4
```

Once building is complete, run Studio with `./studio/Studio`.

Running `sudo make install` will install components to system-wide destinations, e.g. `/usr/local/bin/Studio` for the main executable. This will let you invoke `Studio` from anywhere in the system, rather than just the `build` directory. If you are using this workflow, `sudo make install` must be run after changes to the repository to update the system-wide installation of the executable and libraries. `Studio.desktop` may be used to put a shortcut on your desktop.

If you don't want the Python bindings installed under `/usr/lib`, you can specify the install directory using the `cmake` variable `PYTHON_SITE_PACKAGES_DIR`, e.g.

```
1 cmake -DPYTHON_SITE_PACKAGES_DIR=/usr/local/lib/python3.9/dist-packages
   .
```

Ubuntu releases before 20.04 are not officially supported; if you insist, there are hints here and a discussion of Python linking issues here.

## Windows (VS2022)

Install Git, choosing settings so that it can be invoked from a Windows *Command Prompt* (the defaults should be fine).

Install VS2022 (Community Edition), configured for “Desktop development with C++”. You only *need* MSVC, Windows 10 SDK, and C++ CMake tools for Windows, so feel free to uncheck other optional packages in the right sidebar, then run the installation!

Next, install dependencies using `vcpkg`.

(This step touches many files, so you may want to disable the *Antimalware Service Executable*, which will otherwise scan every single file and slow things down dramatically: in “Windows Security → Virus & threat protection settings”, uncheck “Real-time protection”.)

In a Windows *Command Prompt*:

```
1 git.exe clone https://github.com/libfive/libfive
2 cd libfive
3 git clone https://github.com/Microsoft/vcpkg.git
4 .\vcpkg\bootstrap-vcpkg.bat
5 .\vcpkg\vcpkg.exe install --triplet x64-windows eigen3 boost-container
   boost-bimap boost-interval boost-lockfree boost-functional boost-
   algorithm boost-math libpng qt5-base python3
```

---

Go get some coffee or something - this will take a while.

Once this is done installing, you're ready to actually build `libfive` and Studio!

```
1 mkdir build
2 cd build
3 & "C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\
  Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\cmake.exe" -
  DCMAKE_TOOLCHAIN_FILE="..\vcpkg\scripts\buildsystems\vcpkg.cmake" -
  DVCPKG_TARGET_TRIPLET="x64-windows" -G"Visual Studio 17 2022" ..
4 & "C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\
  Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\cmake.exe" --
  build . --config Release --target Studio --
5 .\studio\Release\Studio.exe
```

At this point, you can also double-click on `Studio.exe` to launch it, and create a shortcut to put it on your desktop.

(don't move it out of the `build` directory, or the precarious house of cards that finds Python will come tumbling down)

When changes are made, you *should* only need to re-run the build step, i.e.

```
1 & "C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\
  Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\cmake.exe" --
  build . --config Release --target Studio --
```