
Build and Installation Instructions for OpenVSP

INTRODUCTION

OpenVSP is a parametric aircraft geometry tool. OpenVSP allows the user to create a 3D model of an aircraft defined by common engineering parameters. This model can be processed into formats suitable for engineering analysis.

The predecessors to OpenVSP have been developed by JR Gloudemans and others for NASA since the early 1990's. In January 2012, OpenVSP was released as an open source project under the NOSA 1.3 license. The first open source version was 2.0.0.

LICENSE

OpenVSP is available under the terms of the NASA Open Source Agreement (NOSA) version 1.3. The complete terms of the license are specified in the LICENSE file.

DEPENDENCIES

OpenVSP relies on a variety of libraries and code made available by other authors. If you are a VSP user, you probably don't need to worry about this. Take this section as informational, providing credit where due.

If you wish to compile VSP or to use the VSP API, then you will need to know more. These dependencies fall into four categories: those assumed to be provided by the operating system; those that must be downloaded and installed by the developer; those that are bundled with OpenVSP, but the developer may want to use a different version installed on the system; and those that are bundled with OpenVSP.

Most users are familiar with the OpenVSP GUI. OpenVSP can also be accessed by other programs via an API, or in a batch mode with no user interaction. These 'headless' targets may be compiled without any of the graphics libraries installed. Although most developers will want to build the graphical version, those dependencies only needed by the GUI program and graphics-enabled API are marked GRAPHICS_ONLY.

Assumed provided by the operating system.

- OpenGL - 3D graphics library. OpenGL should be available on any platform capable of displaying 3D graphics. GRAPHICS_ONLY

Installed by the developer.

- C++ Compiler - OpenVSP requires a modern C++ compiler that supports some C++11 features. We try to support popular free compilers on the main platforms. For Windows, we require Visual Studio 2010 Express or newer. For Mac OSX or Linux/BSD, we support LLVM and GCC.
- CMake 3.1 - Cross-platform build system. CMake generates platform-native build files which control compilation of OpenVSP. CMake is available as a standard package on most Linux systems and binary installers are available for many other platforms including Windows and Mac OS X.
- SWIG - Simplified Wrapper and Interface Generator. Optional dependency used to build interface to API for Python and other scripting languages.
- Python - Scripting language. Optional dependency required if building Python API module.
- Doxygen - Documentation generator. Optional dependency used for generating documentation from the source. Can also utilize graphviz dot to generate improved figures in the documentation. (<http://www.graphviz.org>)

Bundled with OpenVSP, but system libraries may substitute.

- Code-Eli - Curve & surface library. Code-Eli was developed by David Marshall to meet OpenVSP's needs. It is a header-only library that must be available to build. It is not likely to be packaged on any system.
- Eigen3 - Template library for linear algebra. This header-only library is required by Code-Eli. Eigen3 is likely to be available as a standard package on Linux.
- CppTest - C++ Unit testing framework.
- Libxml2 - XML parser and toolkit. Libxml2 most likely comes pre-installed with most Linux distributions and Mac OS X systems. Windows users must download the source and build this library following the Libxml2 instructions.
- Clipper2 - An open source library for clipping and offsetting 2D lines and polygons.
- CMinpack - C/C++ port of standard Fortran Levenberg-Marquardt implementation.
- FLTK - Cross-platform windowing library. FLTK should be available as a standard package on most Linux systems and can be installed from source on Windows and Mac OS X according to the FLTK instructions. GRAPHICS_ONLY
- GLM - OpenGL math library. GRAPHICS_ONLY
- GLEW - OpenGL Extension Wrangler Library. This library is used to access any modern OpenGL features. GRAPHICS_ONLY

-
- libIGES - Library for working with IGES files. This is not likely to be packaged on any system. OpenVSP uses a version modified by Rob McDonald or Justin Gravett for CMake 2.8 compatibility
 - STEPcode - Library for working with STEP standard files. Based on an old library developed by NIST. This is a relatively new library, in constant development. Not likely to be packaged on any system.
 - exprparse - A simple c++ library to compute values from simple math expressions stored as strings.
 - delabella - A c++ Delaunay triangulation library.
 - OpenABF - A c++ Angle based flattening mesh parameterization library.

Bundled with OpenVSP.

- AngelScript - Embedded scripting language for C++. Used for user-defined components.
- nanoflann - Fast nearest neighbors library.
- sixseries - NASA 6-Series airfoil generation Fortran code released to the public domain from NASA TM X 3069 September, 1974. Converted to C using F2C.
- Triangle by Jonathan Shewchuk 2D Delaunay triangulator.
 - CMake Library version used by OpenVSP
- tri_tri_intersect by Philippe Guigue and Olivier Devillers - Fast and Robust Triangle-Triangle Overlap Test using Orientation Predicates. An upgraded version of Tomas Möller's fast triangle-triangle intersection routines included with earlier versions of OpenVSP. While largely based on Möller's algorithms, this code is more numerically robust and efficient.
 - Old URL
 - Philippe Guigue, Olivier Devillers, "Fast and Robust Triangle-Triangle Overlap Test using Orientation Predicates", Journal of Graphics Tools, Vol. 8, Iss. 1, 2003.
- glFont2 by Brad Fish (was: students.cs.byu.edu/~bfish/glfont2.php) - 2D OpenGL font tool. gl-Font2 has been included in this source distribution. GRAPHICS_ONLY
- cartesian - Simple 2D plotting library for FLTK. GRAPHICS_ONLY
- Pinocchio - The pinocchio auto-rigging / weighting tool.
- stb - Single-file libraries for C/C++ for reading and writing image files. GRAPHICS_ONLY
- wavedragEL - Standalone version of the Eminton-Lord procedure for obtaining the zero-lift wavedrag D/Q written by Sriram Rallabhandi

BUILD INSTRUCTIONS

If you want to build OpenVSP on a Debian based Linux computer (Debian, Ubuntu, Mint, etc.), step-by-step instructions are included on the OpenVSP Wiki here: [Debian Based Build](#).

If you want to build OpenVSP on a RPM based Linux distribution, step-by-step instructions are included on the OpenVSP Wiki here: [RPM Based Build](#).

All of the supporting libraries and code described above in DEPENDENCIES categories 1 and 2 must be properly installed. Those in categories 3 and 4 can be satisfied by OpenVSP itself. Any dependencies from category 3 that the developer prefers to use a system library must be properly installed.

We only expect two common cases where developers would prefer to use the system libraries over the bundled ones. First, on Linux where system libraries are readily available and distributions strongly prefer they are used. Second, when a developer is simultaneously working on OpenVSP and the dependency in question.

OpenVSP is packaged for compilation into three CMake projects: the main OpenVSP project with all the OpenVSP source and the category 4 dependencies; a Libraries project with all the category 3 dependencies; a SuperProject that can unify building the other two projects.

Casual developers new to OpenVSP will want to use the SuperProject; building it should take care of everything. More involved developers will want to use the OpenVSP and Libraries projects directly; the OpenVSP project integrates better than the SuperProject with most IDEs. Developers who want to use only system libraries for the category 3 dependencies have no need for the Libraries project; they can work only with the main project.

CMake is used to construct the build files for a project. CMake supports out-of-tree builds which keep the source tree free of generated files. Create a build directory to contain the generated files, say `openvsp/build`. Then, change to the build directory and run CMake - passing a relative path to `openvsp/src`, say `cmake ../src`.

Once CMake has generated the files for your favorite build system, load them in the IDE and compile or launch the build from the command line.

The OpenVSP CMake system can be configured by defining a handful of variables. These variables may be relevant to some or all of the projects.

All project variables:

- `VSP_NO_GRAPHICS` – Set this variable to disable everything to do with graphics. The build system will not search for or build any graphics libraries. It will only build the headless batch-mode VSP, API, and bindings. This is ideal for building VSP on a HPC machine with limited access.

-
- `VSP_NO_VSPAERO` – Set this variable to disable everything to do with VSPAERO. The build system will not build VSPAERO or any of its associated utilities.
 - `XXX_OMP_COMPILER` – Set these variables to point at secondary compilers to use when the primary compiler does not support OpenMP. This will allow the VSPAERO solver to be built as a multithreaded program. Expected versions of this variable are:
 - `C_OMP_COMPILER`
 - `CXX_OMP_COMPILER`
 - `VSP_INSTALL_API_TEST` – Set to include the `apitest` executable in the installation package.

OpenVSP project variables:

- `VSP_LIBRARY_PATH` – Set this variable to point at the directory where the Libraries project was built. The SuperProject sets this path automatically. The Libraries project writes a file `VSP_Libraries_Config.cmake` containing numerous settings to this path.
- `XXXX_INSTALL_DIR` – Set this variable to point to a directory where a library has been installed. This provides a suggestion to a corresponding `FIND_PACKAGE(XXXX)`. Do not use these variables with `VSP_LIBRARY_PATH`, it will overwrite them. Acceptable versions of this variable are:
 - `Adept2_INSTALL_DIR`
 - `CLIPPER2_INSTALL_DIR`
 - `CMINPACK_INSTALL_DIR`
 - `CODEELI_INSTALL_DIR`
 - `CPPTTEST_INSTALL_DIR`
 - `DELABELLA_INSTALL_DIR`
 - `EIGEN_INSTALL_DIR`
 - `EXPRPARSE_INSTALL_DIR`
 - `FLTK_INSTALL_DIR`
 - `GLEW_INSTALL_DIR`
 - `GLM_INSTALL_DIR`
 - `LIBIGES_INSTALL_DIR`
 - `LIBXML2_INSTALL_DIR`
 - `OPENABF_INSTALL_DIR`
 - `PINOCCHIO_INSTALL_DIR`
 - `STEPCODE_INSTALL_DIR`
 - `TRIANGLE_INSTALL_DIR`

-
- `VSP_ENABLE_MATLAB_API` – Set this variable to ON to compile the OpenVSP MATLAB API. To be successful, `SWIG_EXECUTABLE` must be set to the swig executable compiled off of the unreleased MATLAB branch of SWIG.

Libraries & SuperProject project variables:

- `VSP_USE_SYSTEM_XXXX` – Set this variable to search for the particular library rather than use the bundled versions. Acceptable versions of this variable are:
 - `VSP_USE_SYSTEM_ADEPT2`
 - `VSP_USE_SYSTEM_CLIPPER2`
 - `VSP_USE_SYSTEM_CMINPACK`
 - `VSP_USE_SYSTEM_CODEELI`
 - `VSP_USE_SYSTEM_CPPTTEST`
 - `VSP_USE_SYSTEM_DELABELLA`
 - `VSP_USE_SYSTEM_EIGEN`
 - `VSP_USE_SYSTEM_EXPRPARSE`
 - `VSP_USE_SYSTEM_FLTK`
 - `VSP_USE_SYSTEM_GLEW`
 - `VSP_USE_SYSTEM_GLM`
 - `VSP_USE_SYSTEM_LIBIGES`
 - `VSP_USE_SYSTEM_LIBXML2`
 - `VSP_USE_SYSTEM_OPENABF`
 - `VSP_USE_SYSTEM_PINOCCHIO`
 - `VSP_USE_SYSTEM_STEPCODE`
 - `VSP_USE_SYSTEM_TRIANGLE`

Build Instructions for Windows 10 and Visual Studio Express 2017

These instructions are for building the Libraries and OpenVSP separately. It is assumed the user has successfully installed CMake, Python, SWIG, and Doxygen and those utilities can be called from the command line.

Building Libraries

1. Create a new directory called build located outside the locally cloned OpenVSP repo
2. In the new build directory, create new directories “Libraries” and “vsp”

-
3. Using the CMake gui (cmake-gui.exe), click “Browse Source” and browse to .../OpenVSP/Libraries where OpenVSP is the top level of the cloned repo for the source code.
 4. Click “Browse Build” and browse to the newly created “Libraries” directory (.../build/Libraries) for the build location of the new binaries.
 5. Click “Configure” and select “Visual Studio 2017” for the project generator and “x64” for the Optional Platform Generator. Click “Finish”.
 6. Click the “Add Entry” button. For Name, enter “CMAKE_BUILD_TYPE”. For Type, select STRING, and Value type “Release”. Click OK.
 7. Click “Generate”.
 8. Open Visual Studio Express 2017 and select File > Open Project and browse to .../build/Libraries/VSP_LIBRARIES.sln.
 9. In the Solutions Configurations pulldown, change from “Debug” to “Release”.
 10. Right-click on “ALL_BUILD” in the Solution Explorer and select “Build”. When complete, repeat this step to verify no errors.

Building OpenVSP

1. Open the CMake Gui (cmake-gui.exe) and browse to .../OpenVSP for the source code and .../build/vsp for the build location for the binaries.
2. Click “Add Entry” and type “CMAKE_BUILD_TYPE” for Name, select “String” for Type and enter “Release” for Value.
3. Click “Add Entry” again and input “VSP_LIBRARY_PATH” for Name, select PATH for Type and browse to .../build/Libraries
4. Click “Configure” and select “Visual Studio 2017” for the project generator and “x64” for the Optional Platform for Generator. Click “Finish”.
5. Click “Generate”.
6. Open Visual Studio Express 2017 and select File > Open Project and browse to .../build/vsp/VSP_TOP.sln.
7. In the Solutions Configurations pulldown, change from “Debug” to “Release”.
8. Right-click on ALL_BUILD in the Solution Explorer and select Build. When complete, repeat this step to verify no errors.
9. Right-click on PACKAGE in the Solution Explorer and select Build. This packages all the directories and executables in a zip file. This zip file will be placed in .../build/vsp/.