Chapter 2

Installation

This chapter describes the steps required to install VTK on your computer system. The overall difficulty of this process depends on several factors. If you are developing using the precompiled, interpreted executables and libraries (e.g., Tcl), the installation is painless and fast. On the other hand, if you are compiling the VTK source code and building your own libraries, expect to spend one-half hour on faster, multi-processor systems, and several hours on slower, memory limited systems. Also, the installation depends on how many interpreted languages you wrap around the VTK C++ core, and your system configuration.

You may wish to refer to "System Architecture" on page 19 for an overview of the VTK architecture—this may make the compile process easier to follow. Also, if you run into trouble, you can contact the `vtkusers` list (see "Additional Resources" on page 4).

2.1 Overview

Installing VTK on your computer can range from the easy to difficult, depending on the specifics of your configuration. A simple PC binary installation might take less than a minute. A full source code compilation can take several hours. For some unfortunate people it has taken many days and considerable frustration. This chapter will help prevent you from falling into this category. Carefully following the instructions in this chapter should result in a successful installation of VTK with a minimal amount of effort.

The chapter is divided into two sections based on the type of operating system that you are installing on: either Windows or UNIX (for Macintosh OSX or Linux, follow the UNIX instructions). You only need to read the appropriate section for your installation. The *Visualization Toolkit* does not run on older versions of Windows such as Windows 3.1. It also does not run on any Macintosh OS older than OSX.

2.2 Installing VTK on Windows 9x/NT/ME/2000/XP

Under Windows there are two types of installations. The first is an binary/executable installation that lets you do development in C++, Java, Tcl, and/or Python by compiling and linking against pre-compiled libraries (C++), or running pre-compiled executables (e.g. Java, Tcl, or Python). The second type is a full source code installation requiring you to compile the VTK source code (to generate C++ libraries) and VTK wrapper code (to generate Java, Tcl, and Python executables). Of the two types of installations, the binary installation is much easier, and is recommended. The source code installation has the advantage that you can monitor, debug, and modify VTK code—which is probably what you want if you are a class developer. Note, however, that even if you choose the binary installation, you can still extend VTK in a variety of ways—creating your own class (see "Writing A VTK Class: An Overview" on page 208), using run-time programmable filters (see "Programmable Filters" on page 292), and replacing VTK classes at run-time with your own versions of the class (see "Object Factories" on page 211).

Binary Installation

To install the VTK libraries and executables, run `setup.exe`, which is located on the included CD ROM. (These instructions assume installation from the CD. Minor differences exist if you download from the web.) There are a few ways to do this. The easiest is to double click on the `My Computer` icon on your windows desktop. A window will appear containing icons for your hard drive, CD, and floppy, among others devices. Double click on the CD icon and you should get a window with a number of file icons in it. Double click on the `setup` or `setup.exe` icon and the VTK installation process will start, bringing up a little installation GUI something like that shown in **Figure 2–1**.

The first thing you'll need to decide is what parts of VTK you want to install. The binary installation process is packaged into five parts, as shown below.

1. `vtk40Core` — This part contains the VTK DLL for Windows 9x/NT/ME/2000/XP.
2. `vtk40Cpp` — This part contains include files and libraries for C++ development under Microsoft Visual C++ (version 6.0).
3. `vtk40Tcl` — This part contains the libraries and DLLs for Tcl.
4. `vtk40Java` — This part contains the libraries and DLLs for Java.

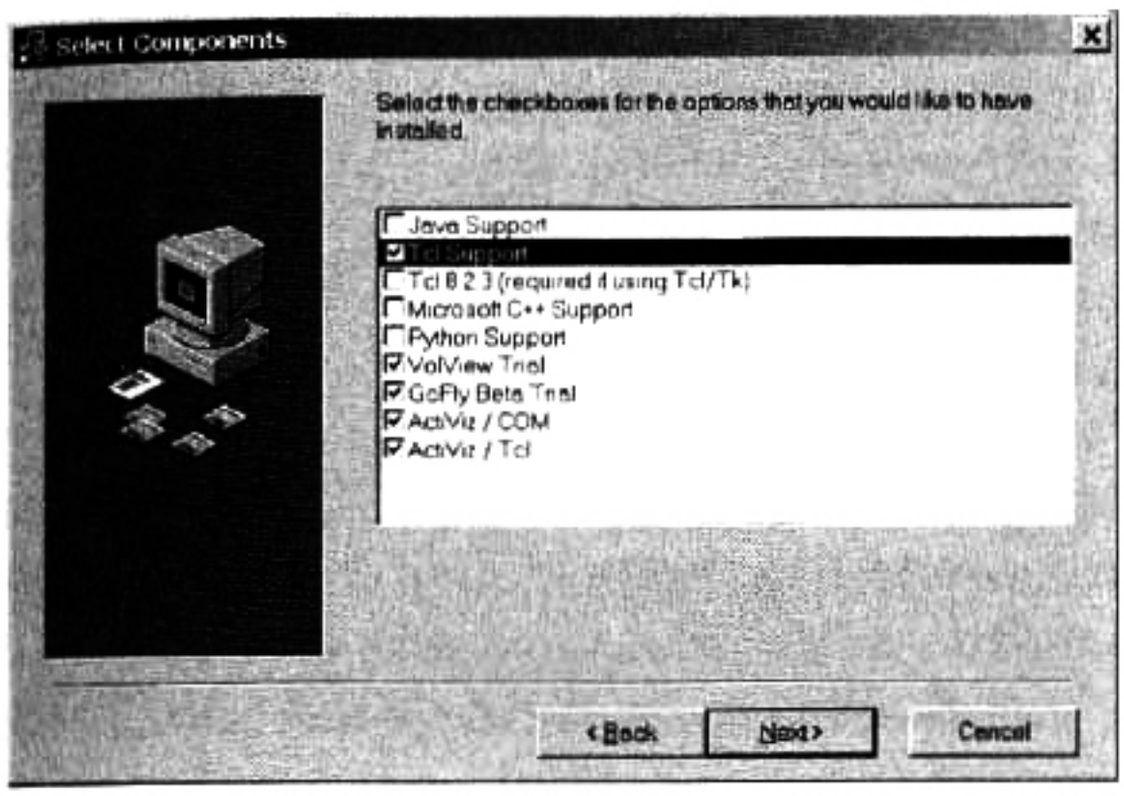

Figure 2–1 The VTK installation wizard for Windows 9x/NT /ME/2000/XP systems. Select the VTK components that you wish installed, and then proceed according to the directions given. You can use the same approach to install nightly or other releases found on the VTK home page `http://public.kitware.com/VTK`. Note that you also have the choice to install 30-day trial versions of ActiViz and VolView, commercial products available from Kitware.

5. `vtk40Python` — This part contains the libraries and DLLs for Python.
6. `CMake` - This part contains the source and Windows 9x/NT/ME/2000/XP binaries for CMake, a tool used to build VTK from source.

Depending on what you plan on doing with VTK, you'll want to install anywhere from two to five parts (`vtk40Core` should always be installed). For example, if you're doing C++ development, install `vtk40Core` and `vtk40Cpp`. If you're creating Tcl applications, you'll want to install `vtk40Tcl`; similarly for Java and Python. After you choose which parts you wish to install, answer the installation questions that follow (location of installation, etc.). If you are installing onto Windows98 or WindowsME/2000/XP then you should select Windows NT when prompted, not Windows95.

If you have installed Tcl, there should be a `vtk` option under your `Start/Programs` submenu. You can then run the examples in the installation directory, or you can also run the Tcl examples off of the CD ROM. To do this, just navigate to the CD ROM as before using either the Windows Explorer or `My Computer`. On the CD ROM in the VTK folder (`vtk-src-windows`), you will see a folder called `Examples`. Under the `Examples` folder, there are folders such as `GUI`, `MangledMesa` and `Parallel`; each of those folders will have a sub folder called `Tcl` that contains various Tcl examples. In addition to the `Examples` folder, there are library folders like `Graphics`, `Imaging`, and `Filtering`. Each of these folders contains a `Testing/Tcl` sub folder containing the regression tests for VTK. Try running any example by double clicking on the Tcl file. When you double click on a Tcl file (`.tcl` extension) for the first time, a dialog box may appear asking you what to use to open the file. This means that you need to create an association between Tcl files and an executable to run them. If this happens, click the `Other` button on the dialog. A

new dialog will appear. Use this dialog to go to the directory where you installed VTK. Normally this is `C:\Program Files\vtk` (or a similar variation, e.g., `vtk40` or even possibly `Visualization Toolkit`). In there you should see a `bin` folder which in turn contains a program called `vtk`. Double click on `vtk` and then select the `Ok` button on the original dialog. Your example should then run. In the future, double clicking on any Tcl scripts will automatically begin execution of `vtk`.

For best results on a PC, make sure the display setting are set to more than 256 colors. This can be done by going to the `Start` menu, selecting the `Settings/Control Panel` option. A window will appear with about twenty icons in it. Double click on the `Display` icon which will bring up another dialog with a `Settings` tab. Select that tab and then adjust the color palette to some value more than 256 colors. Click on the `Ok` button and then follow any additional instructions. You might have to restart your machine at this point.

That completes the binary installation process for Windows. In Chapter 3 we'll go into more detail on how to write your own C++, Tcl, Java and Python applications.

Source Code Installation

To develop C++ applications and extend VTK, you will need to do a source code installation. This is more challenging and will tie up your machine for a few hours as it compiles VTK. First you need to make sure your machine is capable of building a VTK source code release. You must be running Windows95, Windows98, Windows ME, Windows NT 4.0, Windows 2000 or Windows XP. You will need a C++ compiler installed on your machine. The instructions in this guide are oriented towards Microsoft Visual C++ Version 6.0 or Microsoft Visual C++ .NET which work well with VTK. We also support (to a lesser degree) the Borland C++ compiler. If you have not installed a C++ compiler, then you should do this first.

The next issue to consider is what additional tools you plan to use. If you plan to do development in Java then you must download and install the Java JDK which is available from Sun Microsystems at `http://www.java.sun.com`. If you plan on using Tcl/Tk and you are not using Microsoft Visual C++, then you will need to download and build the source code version of Tcl/Tk from `http://www.scriptics.com`. (Note: download Tcl/Tk version 8.3.2.)

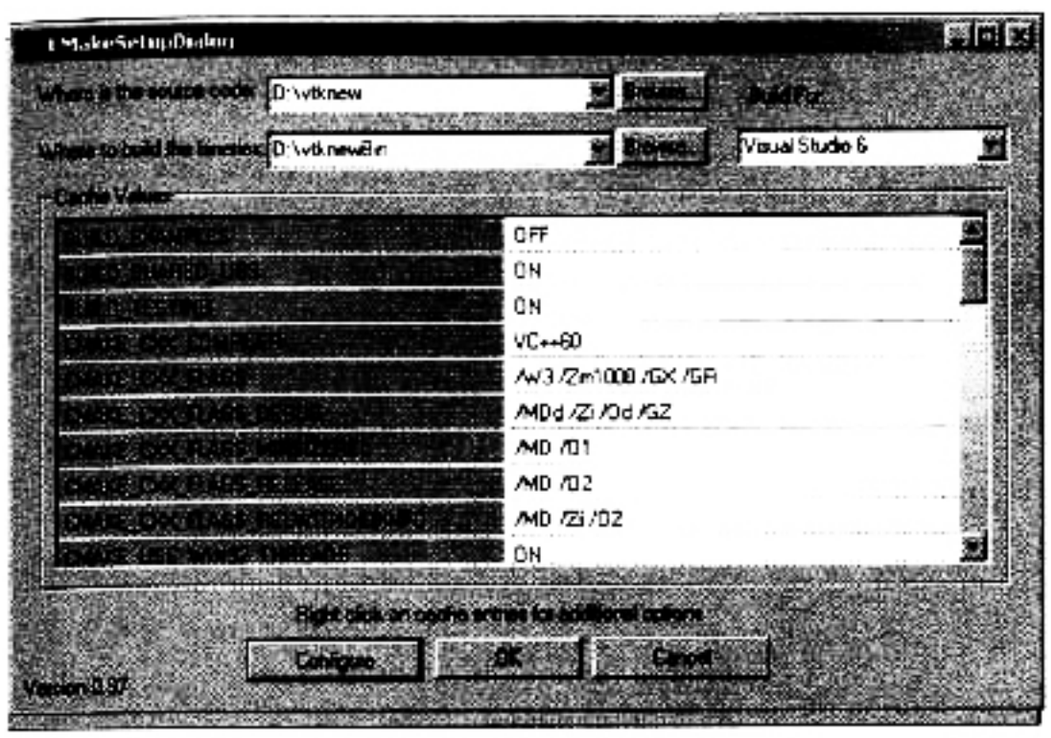

Figure 2–2 CMake is used to generate projects, makefiles, or workspaces for different compilers and operating systems. CMake is cross-platform, and on Windows, runs with a GUI that allows you to easily set compile flags. These flags can be set by left-mouse clicking in the build flag value.

Copying the Source Code. The VTK CD ROM comes with a complete copy of the source code stored in the VTK folder. If you don't plan on modifying the source code or creating your own classes then you can leave the source code on the CD ROM. Otherwise, we suggest that you copy the `VTK-src-windows` folder from the CD onto your hard disk. This has the added benefit that it will improve your compile time. (Be careful;: using the standard copying tools code on a Windows system from a CD will result in write-protected files. You may wish to use the DOS xcopy command or Unix tools such as Cygwin.)

Installing CMake. To compile VTK, you will first need to install a program called CMake. CMake is an open source, cross platform build tool (`http://public.kitware.com/CMake`). The use of CMake allows VTK to be configured and built on a variety of machines using the same source tree and build files. You can obtain CMake from `http://public.kitware.com/CMake/HTML/Download.html`. For Microsoft and Borland compilers there is a pre-compiled binary that you can download and install, which is the preferred installation method.

Running CMake. After you have setup your C++ compiler, installed CMake, and installed any additional packages such as Tcl, Java, and Python, you are ready to run CMake. To run CMake, there should be a CMake entry in the Start menu under `Programs->CMake->CMakeSetup`. The `CMakeSetup.exe` interface (**Figure 2–2**) is a simple interface that allows you to customize the build to your particular machine and desired options for VTK. First you must tell `CMakeSetup` where the source tree for VTK is located and where you want to put the VTK binaries (these are generated as a result of compilation of the source code). You can specify those directories with the Browse buttons or by typing in the paths manually. The next step is to choose the build system that you are going to use (Microsoft Visual Studio 6, Microsoft Visual Studio .NET or Bor-

land). Once the source, binary and build system have been selected, you should click on the Configure button. This will fill the CMakeSetup GUI with a list of variables and values found in the CMake cache. When first run, all the variables will be colored red. The red indicates that the cache entry was generated or changed during the previous configure step.

At this point, you can customize your VTK build. For example, if you want to enable the Tcl wrapping feature of VTK, scroll down in the cache values editor to the entry `VTK_WRAP_TCL`, and click on the value to toggle it from `NO` to `YES`. After that, click the Configure button again. This will cause most of the values to change to gray, and any new values to appear in red. If you installed Tcl/Tk from binary, none of the new values should have `NOTFOUND` as values, if they do, you will have to specify those paths manually with the CMake interface. To set any value in the CMake interface, you click to the right of the variable where the value is displayed. Depending on the type of variable, there may be a file chooser, edit box or pull down that will allow you to edit the value.

These are some important cache values for VTK are:

- `BUILD_SHARED_LIBS` — If this Boolean value is set to yes, then DLLs or shared libraries will be built. If it is no, then static libraries will be built. The default is static libraries. The static libraries are somewhat easier to work with, since they do not need to be in your path when executables are run. The executables will be self-contained. This is preferred for distribution of VTK based applications.
- `VTK_WRAP_TCL` — This determines if Tcl wrapping will be built.
- `VTK_WRAP_PYTHON` — This determines if Python wrapping will be built.
- `VTK_WRAP_JAVA` — This determines if Java wrapping will be built.

To get on-line help for any variable in CMake, simply click right over the value and select "Help for Cache Entry". Most of the defaults should be correct.

Continue to click on `Configure` until there are no longer any red values and you are happy with all of the values. At this point, you can click the `OK` button. This will cause CMake to write out the build files for the build type selected. For Microsoft, a project file will be located in the binary path you selected. Simply load this project file `VTK.dsw` into Visual Studio, and select the configuration you want to build in the `Build->Set Active Configuration` menu of Visual Studio. You will have the choice of Debug, Release, MinsizeRel (minimum size release), and RelWithDebInfo (release with debug information). You can select the `ALL_BUILD` project, and compile it as you would any other

Visual Studio project. For Borland, makefiles are generated, and you have to use the command line make supplied with that compiler. The makefiles are located in the binary directory you specified.

Once VTK has been built all libraries and executables produced will be located in the binary directory you specified to CMake in a sub-folder called bin (unless you changed the `EXECUTABLE_OUTPUT_PATH`, or `LIBRARY_OUTPUT_PATH` variables in CMake.

(**Note:** Do not use the MSVC++ "Rebuild All" menu selection to rebuild the source code. This deletes all `CMakeLists.txt` files which are then automatically regenerated as part of the build process. MSVC will then try reloading them and an error will result. Instead, to rebuild everything, remove your VTK binary directory, rerun CMake, and then rebuild.)

Once VTK has been built, you need to let Windows know where to find the DLLs, if you turned on `BUILD_SHARED_LIBS`. There are a couple ways to do this. First you can simply copy the resulting DLLs into a directory Windows normally checks such as `Windows/System` or `Winnt/System`. The other choice is to modify your `PATH` environment variable to include the directories where the libraries are stored. If you decide to copy the DLLs and executables, you will need to copy all the files in the `bin/selected configuration/` directory (where `selected configuration` is the build configuration that you chose earlier).

If you choose not to copy the DLLs, then you will need to edit your `PATH` environment variable. Under Windows95 and Windows98 you can do this using `sysedit` to add a line to the `autoexec.bat` file. Four examples are given below that correspond to the four different configurations listed earlier. All four examples assume that you build VTK in `C:\vtkbin`.

1. `PATH=C:\vtkbin\bin\Debug`
2. `PATH=C:\vtkbin\bin\Release`
3. `PATH=C:\vtkbin\bin\MisizeRel`
4. `PATH=C:\vtkbin\bin\RelWithDebInfo`

The same idea can be used in Windows NT/ME/2000/XP except that it should be done by right clicking on the `My Computer` icon. Select the `Properties` option and a dialog should pop up. Select the `Environment` tab (it may be found under the `Advanced` tab). Either modify or add a `PATH` environment variable with the path as described above. If a

PATH variable already exists, add the VTK paths to the front of it. For example, if the original path was

```
C:\winnt\system;C:\some\other=dir;%PATH%
```

change it to

```
C:\vtkbin\bin\Debug; C:\winnt\system;C:\some\other=dir;%PATH%
```

If you've made it this far, you've successfully built VTK on a PC. It can be a challenging process, partly due to limitations in the current compilers, and partly due to the size and complexity of the software. Please pay careful attention to the instructions given earlier. If you do run into problems, you may wish to join the vtkusers mailing list (see "Additional Resources" on page 4) and ask for help there. Commercial support is also available from Kitware.

2.3 Installing VTK On Unix Systems

There are a wide variety of flavors of Unix systems. As a result you will have to compile the VTK source code to build binaries and executables.

Source Code Installation

This section will walk you through the steps required to build VTK on a UNIX system. Unlike the PC, pre-compiled libraries and executables are not available for most Unix systems, so it's likely that you'll have to compile VTK yourself. (Note: check the vtkusers list and other resources as described in "Additional Resources" on page 4—some users maintain binaries on the Web.) Typically, it is a fairly simple process and it should take about one to four hours depending on the size of your machine. (High-end, large-memory multi-processor machines using parallel builds can build the C++ and Tcl libraries and executables in under 10 minutes!) Most of this time is spent waiting for the computer to compile the source code. Only about 10-30 minutes of your time will be required. The first step is to make sure you have the necessary resources to build VTK. To be safe, you will need about 300 megabytes of disk space. On some systems, such as the SGI, you may need more space, especially if compiling a debug version of VTK. You will also need a C++ compiler since VTK is written in C++. Typically the C++ compiler will be called CC,

g++, or ACC. If you are not sure that you have a C++ compiler, check with your support staff.

If you are planning to use VTK with Tcl/Tk, Python, or Java, then you will first need to download and install those packages. The Java JDK is available from Sun Microsystems at `http://www.java.sun.com`. If you plan on using Tcl/Tk then you will need to download and build the source code version of Tcl/Tk from `http://www.scriptics.com`. Do not download the pre-compiled version of Tcl/Tk as the binary files will not work with VTK. Python can be downloaded from `http://www.python.org`. Follow the instructions in these packages to build them.

Next, we recommend that you copy the VTK source code from the CD ROM onto your hard disk. Typically, a UNIX command like

```
/bin/cp -r /Your/CDROM/vtkunix /yourdisk/vtk
```

is used. Now you need to decide if you are doing an in-place build. An in-place build puts the object files into the same directory as the source code. If you do not have `gmake` (GNU make) installed on your computer, you will need to do an in-place build. If you have `gmake` and will be building VTK for a few different types of machines, you can choose to do a multiple architecture build. Instructions for the in-place build are provided below, followed by instructions for a multiple architecture build.

Running CMake

For Unix, VTK uses CMake for the build process just like in the windows environment (see "Running CMake" on page 11). However, there are no pre-compiled binaries for CMake. So, to build VTK, you will first have to build and install CMake. In addition, CMake is typically not run from a GUI on UNIX platforms. If you want a GUI for CMake on UNIX, you will need to install FLTK prior to building CMake. To build and install CMake, simply untar the sources into a directory, and then run:

```
./configure
make install
```

If you do not have root privileges, you can skip the install target, and just type `make`. The command line CMake executable will be located at `CMake/Source/cmake`.

To run CMake on Unix, you use the command line interface to CMake. If you installed FLTK, there will be a GUI version in `CMake/Source/CMakeSetup`. However, these instructions are for the command line interface, see the Windows installation for information about the GUI version.

It is a good idea to tell CMake which C++ and C compilers you want to use. On most systems, you can set the information this way:

```
setenv CXX /your/c++/compiler
setenv CC /your/c/compiler
```

or

```
export CXX=/your/c++/compiler
export CC=/your/c/compiler
```

Otherwise CMake will default to the GNU compiler (gcc). Once you've done this, run CMake in the source directory simply as

```
cmake
```

(This is called an in-place build because you are generating object code and binaries in the VTK source directory. To support multiple platforms, or place binaries in a separate directories, see "Building VTK On Multiple Platforms" on page 17

UNIX developers familiar with configure scripts will notice that CMake and configure are similar in their functionality. However, configure takes command line arguments to control the generation of makefiles. CMake is controlled from its CMakeCache.txt file. To modify the operation of CMake, edit this file as necessary, keeping in mind that every time you modify CMakeCache.txt you need to rerun CMake.

Customizing the Build. Once you run CMake, it will create a file call `CMakeCache.txt` that will contain all of the information that CMake could figure out about your system. It is a good idea to look at the contents of this file and make sure CMake was able to find important things like your OpenGL library. (Note: the CMake process is iterative. After you run it the first time, you change files in `CMakeCache.txt` and rerun until no changes are made.) If you want to build one of the wrapped languages like Tcl or Python, you will have to turn on `VTK_USE_TCL` in the cache file. Then re-run CMake. This will cause your cache to be filled with more values, and you should check the cache to make sure it found the correct Tcl installation you wanted to use. Each time you edit the

cache, you should re-run CMake. Note, after the first run, it is no longer necessary to specify a compiler to CMake, and CMake can be run from `make`, with the command: `make rebuild_cache`. When you are done editing the `CMakeCache.txt` file, all your makefiles should be generated and VTK is ready to build.

Compiling the Source Code

Once CMake has completed running, you can type `make` (or `gmake`, if you are using GNU make) and VTK should compile. Some make utilities support parallel builds (e.g., `gmake` with the `-j` option). Use parallel make if possible, even if on a single processor system, because usually the process is IO bound and the processor can handle multiple compiles. If you do run into problems, you may wish to join the `vtkusers` mailing list (see "Additional Resources" on page 4) and ask for help there. Commercial support is also available from Kitware.

Building VTK On Multiple Platforms

If you are planning to build VTK for multiple architectures then you can either make a copy of the entire VTK tree for each architecture and follow the instructions above; or, if you have `gmake`, you can have one copy of the VTK source tree and compile it in a different manner. Instead of running CMake in the VTK source directory, create a new directory where you have some free disk space (not in the VTK tree), such as `vtk-solaris`. Change directory (`cd`) into this directory and then run CMake similar to the following example:

```
cd /yourdisk
ls      (output is: vtk vtk-solaris vtk-sgi)
cd vtk-solaris
cmake ../vtk
```

This will create makefiles and a `CMakeCache.txt` file in the `vtk-solaris` directory. Then you can edit `CMakeCache.txt` and `make rebuild_cache` until the values are correct as you did for the in-place build. Note: It is possible to pre-load the cache for CMake. To do this, you can take some important values that CMake can not figure out on its own, like the path to your Tcl installation, and create a `CMakeCache.txt` file in your build directory before running CMake. CMake will use those values and add more values to the cache file as needed.

Installing VTK

Now that VTK has been built, the executables and libraries will be located in the build directory, in the sub-directory `bin/`. If you plan to share the build with more than one developer on the UNIX system, it is often a good idea to run the make install command. This will install VTK into `/usr/local`, unless you changed the cache value `CMAKE_INSTALL_PREFIX` to another location. Running make install will copy all the files you need to compile and run VTK into a directory that more people can use.

This concludes the build and installation guide for VTK under UNIX. In Chapter 3 more details will be provided on how to run examples and create your own applications.