Base Projects (Part I)

One disadvantage of using 3rd party components from a tutorial perspective is that we need to carefully configure our build system to find the libraries. We may also need to copy library files (.dlls) or resources into specific in order that they can be used. You may have already encountered this in the first year with SDL/SDL2 or with OpenGL in Real-time Graphics. We're going to use CMake to get around both these problems an allow us to work anywhere by simply cloning our project and re-running CMake to generate our build environment.

Overview

The process:

1. Create a fork of the project on Bitbucket.
2. Clone the base project.
3. Create a build directory.
4. Run CMake
5. Build
6. Install
7. Run/Debug

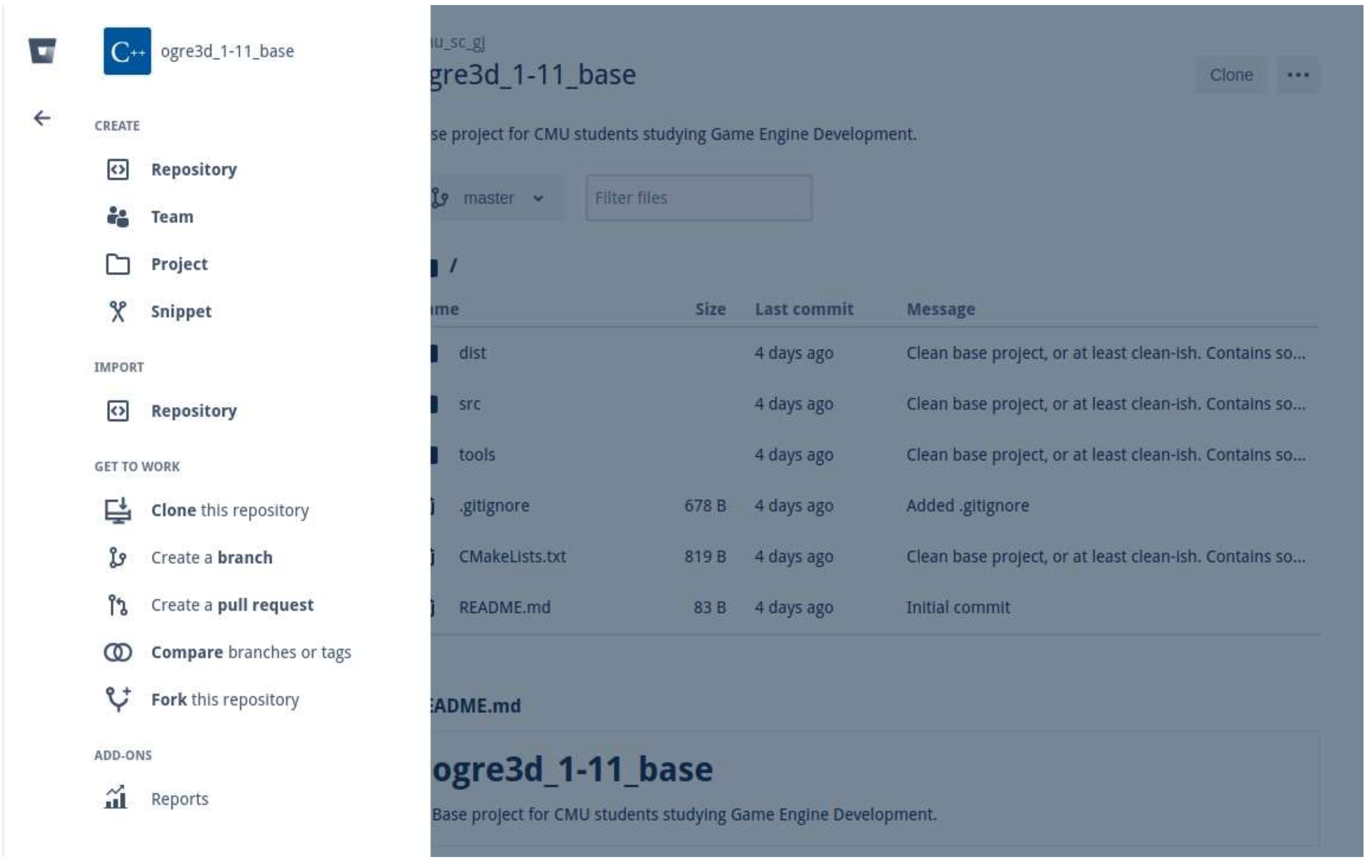
Well start with a base project which includes Ogre and add Bullet later.

Ogre Base Project

The Ogre Base Project is here. Last year I used branches from the main project for different features. This confused the student and was hard to maintain so I'm going to try separate projects which provide examples of loading meshes, scenes and animations. A second separate base project for Ogre + Bullet will be released later.

Creating a Fork

Though the Bitbucket Dashboard create a fork of the above repository, the fork option is not obvious. You'll click the **+** and select Fork the rest of the process is just like creating a new repository (and should be familiar).



u_sc_gj
gre3d_1-11_base

se project for CMU students studying Game Engine Development.

/

me	Size	Last commit	Message
dist		4 days ago	Clean base project, or at least clean-ish. Contains so...
src		4 days ago	Clean base project, or at least clean-ish. Contains so...
tools		4 days ago	Clean base project, or at least clean-ish. Contains so...
.gitignore	678 B	4 days ago	Added .gitignore
CMakeLists.txt	819 B	4 days ago	Clean base project, or at least clean-ish. Contains so...
README.md	83 B	4 days ago	Initial commit

ADME.md

ogre3d_1-11_base

Base project for CMU students studying Game Engine Development.

Windows

Clone the Base Project

Make sure you in the directory you want to store you're working version of the project. Open GitBash and use this to clone your repository.

```
> git clone ...
```

Create a Build Directory

In explorer navigate to the directory into which you've cloned your project and inside the `ogre3d_1_14_base` directory create a `build` directory.

Run CMake

We'll use the CMake-Gui (as we have for Ogre), the makes configuring CMake to find Ogre much easier. Make sure you have **Grouped** checked!

Here's the options to change, I've included the full list at the end but hopefully you won't need it!

Description	Group	Key	Value
Location of the Ogre SDK	CMAKE	**OGRE_DIR**	`<where you cloned>/build/sdk/CMake`

The image above shows my configuration for the base project. Set the `OGRE_DIR` to the SDK path, this should be `<where you cloned>/ogre/build/sdk/CMake` your path will start differently to mine but should end the same unless you changed the SDK output directory when configuring Ogre.

Configure CMake and **Generate** the a Visual Studio project.

Build

Change the start up project to **SimpleGame**, then Build the project (select Build from the Build menu).

Install

You will need to build the **INSTALL target (project), this is not a real target but runs a build script. This copies the executable and the required configuration files, link libraries etc. to a sub-location in the build library.

Ogre3d relies on a number of config files and media files being present. To ease development placed these in a directory called `dist` the install step copies these into the build directory and place it with the executable.

You should have the following file structure:

```
dist
├── bin
│   ├── resources.cfg
│   └── SimpleGame
└── media
    ├── materials
    │   ├── scripts
    │   │   ├── Examples.material
    │   │   └── Ogre.material
    │   └── textures
    │       ├── GreenSkin.jpg
    │       ├── nskingr.jpg
    │       ├── rockwall.tga
    │       ├── spheremap.png
    │       └── tusk.jpg
    ├── models
    │   ├── cube.mesh
    │   ├── ninja.mesh
    │   ├── ninja.skeleton
    │   └── ogrehead.mesh
    └── packs
        └── SdkTrays.zip
```

The media directories are mostly for later tutorials which require loaded models. Packs are actually zip files as Ogre supports the loading of resources from one or more archives.

Run / Debug

CMake can set custom properties on different operating systems, in the case of Windows I've set a variable used by Visual Studio to identify its runtime library. As a result it should be possible to just run/debug as normal.

For Debugging Only

There are a huge number of options, the defaults are usually fine but in case you accidentally change a flag and need to check. Here's all the flags from my working setup, I've tried to group them as best I can.

Description	Group	Key	Value
Path to a program.	CMAKE	CMAKE_AR	FILEPATH
Semicolon separated list of supported configuration types, only supports Debug, Release, MinSizeRel, and RelWithDebInfo, anything else will be ignored.	CMAKE	CMAKE_CONFIGURATION_TYPES	STRING
Flags used by the CXX compiler during all build types.	CMAKE	CMAKE_CXX_FLAGS	STRING
Flags used by the CXX compiler during DEBUG builds.	CMAKE	CMAKE_CXX_FLAGS_DEBUG	STRING
Flags used by the CXX compiler during MINSIZEREL builds.	CMAKE	CMAKE_CXX_FLAGS_MINSIZEREL	STRING
Flags used by the CXX compiler during RELEASE builds.	CMAKE	CMAKE_CXX_FLAGS_RELEASE	STRING
Flags used by the CXX compiler during RELWITHDEBINFO builds.	CMAKE	CMAKE_CXX_FLAGS_RELWITHDEBINFO	STRING
Libraries linked by default with all C++ applications.	CMAKE	CMAKE_CXX_STANDARD_LIBRARIES	STRING
Flags used by the C compiler during all build types.	CMAKE	CMAKE_C_FLAGS	STRING
Flags used by the C compiler during DEBUG builds.	CMAKE	CMAKE_C_FLAGS_DEBUG	STRING

Description	Group	Key	Value
Flags used by the C compiler during MINSIZEREL builds.	CMAKE	CMAKE_C_FLAGS_MINSIZEREL	STRING
Flags used by the C compiler during RELEASE builds.	CMAKE	CMAKE_C_FLAGS_RELEASE	STRING
Flags used by the C compiler during RELWITHDEBINFO builds.	CMAKE	CMAKE_C_FLAGS_RELWITHDEBINFO	STRING
Libraries linked by default with all C applications.	CMAKE	CMAKE_C_STANDARD_LIBRARIES	STRING
Flags used by the linker during all build types.	CMAKE	CMAKE_EXE_LINKER_FLAGS	STRING
Flags used by the linker during DEBUG builds.	CMAKE	CMAKE_EXE_LINKER_FLAGS_DEBUG	STRING
Flags used by the linker during MINSIZEREL builds.	CMAKE	CMAKE_EXE_LINKER_FLAGS_MINSIZEREL	STRING
Flags used by the linker during RELEASE builds.	CMAKE	CMAKE_EXE_LINKER_FLAGS_RELEASE	STRING
Flags used by the linker during RELWITHDEBINFO builds.	CMAKE	CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	STRING
Install path prefix, prepended onto install directories.	CMAKE	CMAKE_INSTALL_PREFIX	PATH
Path to a program.	CMAKE	CMAKE_LINKER	FILEPATH
Flags used by the linker during the creation of modules during all build types.	CMAKE	CMAKE_MODULE_LINKER_FLAGS	STRING
Flags used by the linker during the creation of modules during DEBUG builds.	CMAKE	CMAKE_MODULE_LINKER_FLAGS_DEBUG	STRING

Description	Group	Key	Value
Flags used by the linker during the creation of modules during MINSIZEREL builds.	CMAKE	CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL	STRING
Flags used by the linker during the creation of modules during RELEASE builds.	CMAKE	CMAKE_MODULE_LINKER_FLAGS_RELEASE	STRING
Flags used by the linker during the creation of modules during RELWITHDEBINFO builds.	CMAKE	CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO	STRING
Path to a program.	CMAKE	CMAKE_MT	FILEPATH
RC compiler	CMAKE	CMAKE_RC_COMPILER	FILEPATH
Flags for Windows Resource Compiler during all build types.	CMAKE	CMAKE_RC_FLAGS	STRING
Flags for Windows Resource Compiler during DEBUG builds.	CMAKE	CMAKE_RC_FLAGS_DEBUG	STRING
Flags for Windows Resource Compiler during MINSIZEREL builds.	CMAKE	CMAKE_RC_FLAGS_MINSIZEREL	STRING
Flags for Windows Resource Compiler during RELEASE builds.	CMAKE	CMAKE_RC_FLAGS_RELEASE	STRING
Flags for Windows Resource Compiler during RELWITHDEBINFO builds.	CMAKE	CMAKE_RC_FLAGS_RELWITHDEBINFO	STRING
Flags used by the linker during the creation of shared libraries during all build types.	CMAKE	CMAKE_SHARED_LINKER_FLAGS	STRING
Flags used by the linker during the creation of	CMAKE	CMAKE_SHARED_LINKER_FLAGS_DEBUG	STRING

Description	Group	Key	Value
shared libraries during DEBUG builds.			
Flags used by the linker during the creation of shared libraries during MINSIZEREL builds.	CMAKE	CMAKE_SHARED_LINKER_FLAGS_MINSIZEREL	STRING
Flags used by the linker during the creation of shared libraries during RELEASE builds.	CMAKE	CMAKE_SHARED_LINKER_FLAGS_RELEASE	STRING
Flags used by the linker during the creation of shared libraries during RELWITHDEBINFO builds.	CMAKE	CMAKE_SHARED_LINKER_FLAGS_RELWITHDEBINFO	STRING
If set, runtime paths are not added when installing shared libraries, but are added when building.	CMAKE	CMAKE_SKIP_INSTALL_RPATH	BOOL
If set, runtime paths are not added when using shared libraries.	CMAKE	CMAKE_SKIP_RPATH	BOOL
Flags used by the linker during the creation of static libraries during all build types.	CMAKE	CMAKE_STATIC_LINKER_FLAGS	STRING
Flags used by the linker during the creation of static libraries during DEBUG builds.	CMAKE	CMAKE_STATIC_LINKER_FLAGS_DEBUG	STRING
Flags used by the linker during the creation of static libraries during MINSIZEREL builds.	CMAKE	CMAKE_STATIC_LINKER_FLAGS_MINSIZEREL	STRING
Flags used by the linker during the creation of static libraries during RELEASE builds.	CMAKE	CMAKE_STATIC_LINKER_FLAGS_RELEASE	STRING
Flags used by the linker during the creation of	CMAKE	CMAKE_STATIC_LINKER_FLAGS_RELWITHDEBINFO	STRING

Description	Group	Key	Value
static libraries during RELWITHDEBINFO builds.			
If this value is on, makefiles will be generated without the .SILENT directive, and all commands will be echoed to the console during the make. This is useful for debugging only. With Visual Studio IDE projects all commands are done without /nologo.	CMAKE	CMAKE_VERBOSE_MAKEFILE	BOOL
The directory containing a CMake configuration file for OGRE.	Ungrouped	OGRE_DIR	PATH