

# Game Engine Architecture

Week1-Lab

OGRE

Object-Oriented **G**raphics **R**endering **E**ngine

# Ogre3D

- **O**bject-Oriented **G**raphics **R**endering **E**ngine
- We will be using Ogre in this class
- Ogre is not a game engine, but a rendering engine
- No physics, or anything else – just rendering
- Intentional design decision – do one thing well, use other libraries & plugins to do other things
- Since 2001, OGRE has grown to become one of the most popular open-source graphics rendering engines, and has been used in a large number of production projects, in such diverse areas as games, simulators, educational software, interactive art, scientific visualisation, and others.
- <https://www.ogre3d.org/>
- <https://www.ogre3d.org/download/sdk>

# What version to choose?

- OGRE is not one, but two “sister” projects.
- They are related, but they are not the same – as in being not compatible to one another.
- You may be confused which version to pick.
- The information below will help you to find the correct version for your needs:



# Ogre1 vs. Ogre2

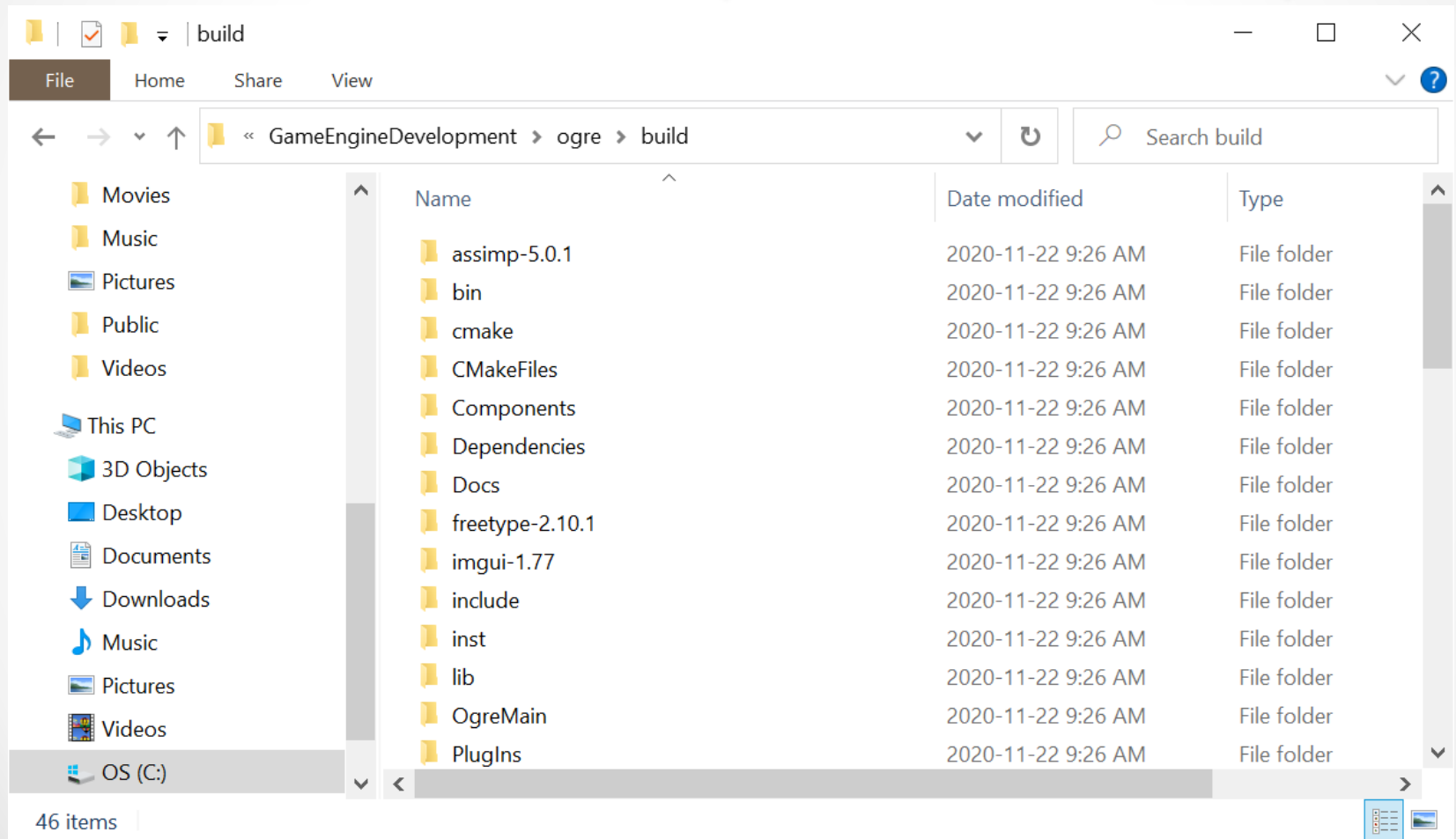
- **Ogre1**

- You need the 1st party components (e.g. Volume, Overlay, MeshLOD) and 3rd party plugins (e.g. GUI Solutions, Exporters)
- You target mobile platforms (Android, iOS, WebGL, Windows UWP)
- You only need a moderate amount (order of 1000) of objects per frame or are fine with manual Instancing
- You want to use a verified code (as in [Unit Tests](#))
- You need compatibility with the Ogre 1.7 API (e.g. legacy codebase)
- You want to learn from Wiki code snippets and a lot of existing resources found online (including books).
- You need broad hardware support including DX9 and OpenGL 1.5 class Hardware

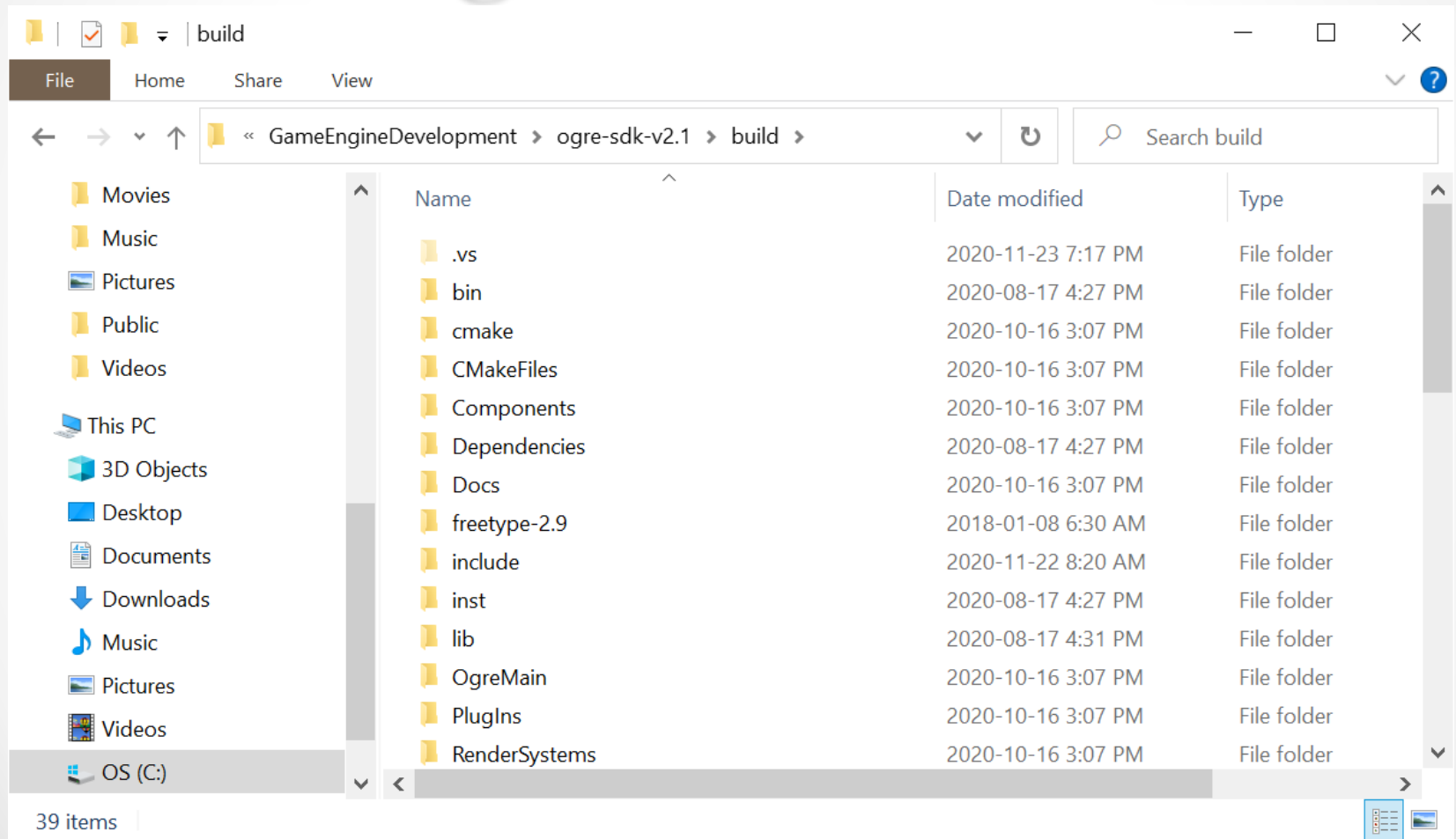
- **Ogre Next**

- You need many objects (order of 10 000) per frame.
- You want the huge performance boost in the above usecase. Your main target is Windows and/or Linux.
- You like bleeding edge.
- You want VR Support out-of-the-box
- You already know your way around Ogre or can learn the API from source code as Documentation is still scarce
- You don't need the 1st party components (Overlay, Terrain, Volume) and 3rd party plugins that haven't been ported yet (like GUI Solutions, Exporters). Check [the 2.0+ Forum](#) to find what has been ported by the community already.
- You don't care about Android support (or anything older than the iPhone 5S)

# ogre-master Folder Structure (OGRE V1)

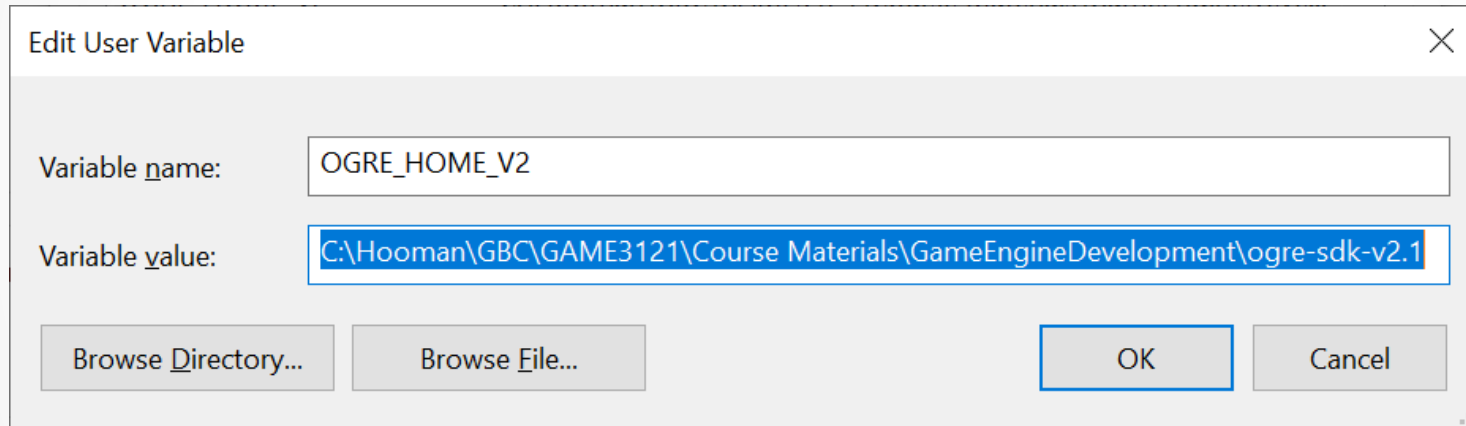


# Ogre V2 SDK



# Setting your Environment Variable

- C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1



Edit User Variable

Variable name: OGRE\_HOME\_V2

Variable value: C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1

Browse Directory... Browse File... OK Cancel

# Ogre Project Folder

- Executables for every sample project will be output in the bin/debug or bin/release folders depending on the project's build configuration. These folders also contain the following required DLLs and configuration files:
  - OgreMain.dll Main Ogre DLL.
  - RenderSystem\_Direct3D9.dll DirectX 9 Ogre render system DLL. This is necessary only if you want Ogre to use the DirectX 9 graphics library.
  - RenderSystem\_Direct3D11.dll DirectX 11 Ogre render system DLL. This is necessary only if you want Ogre to use the DirectX 9 graphics library.
  - RenderSystem\_GL.dll OpenGL Ogre render system DLL. This is necessary only if you want Ogre to use the OpenGL graphics library.
  - Plugin\_OctreeSceneManager.dll Octree scene manager Ogre plugin DLL.
  - Plugin\_ParticleFX.dll Particle effects Ogre plugin DLL.
  - resources.cfg Ogre resource configuration file that contains paths to all resource locations. Resources include graphics files, shaders, material files, mesh files, and so on.
  - plugins.cfg Ogre plugin configuration file that contains a list of all the plugins we want Ogre to use. Typical plugins include the Plugin\_OctreeSceneManager, RenderSystem\_Direct3D9, RenderSystem\_GL, and so on.
  - **ogre.cfg** This file is generated by the Render Settings dialog that appears when you run your application. **Do not** distribute this file with your application. This file will be specific to your own setup. This file will contain your choices for things like screen resolution. Do not modify this file directly. Change the settings with the dialog and it will be automatically updated. In Windows, it will be saved under: **Documents/\$subdir/**



# OGRE Engine Rendering

- If you need to change the default rendering to a different GL renderer, you'll have to delete ogre.cfg file under **Documents/\$subdir/**



# Ogre.cfg

- RenderSystem=Direct3D11 Rendering Subsystem
- [Direct3D11 Rendering Subsystem]
- Allow NVPerfHUD=No
- Backbuffer Count=Auto
- Driver type=Hardware
- FSA=1
- Floating-point mode=Fastest
- Full Screen=No
- Information Queue Exceptions Bottom Level=Info (exception on any message)
- Max Requested Feature Levels=11.0
- Min Requested Feature Levels=9.1
- Rendering Device=(default)
- Reversed Z-Buffer=No
- VSync=Yes
- VSync Interval=1
- Video Mode=800 x 600 @ 32-bit colour
- sRGB Gamma Conversion=No
- [OpenGL Rendering Subsystem]
- Colour Depth=
- Display Frequency=N/A
- FSA=0
- Fixed Pipeline Enabled=Yes
- Full Screen=No
- RTT Preferred Mode=FBO
- VSync=Yes
- VSync Interval=1
- Video Mode= 480 x 640
- sRGB Gamma Conversion=No
- [OpenGL 3+ Rendering Subsystem]
- Colour Depth=
- Display Frequency=N/A
- FSA=0
- Full Screen=No
- RTT Preferred Mode=FBO
- Reversed Z-Buffer=No
- Separate Shader Objects=No
- VSync=Yes
- VSync Interval=1
- Video Mode= 480 x 640
- sRGB Gamma Conversion=No

# Additional Include Directories

Additional Include Directories

\$(OGRE\_HOME)/include/OGRE/Bites/  
\$(OGRE\_HOME)/include/OGRE/  
\$(OGRE\_HOME)/include/OGRE/Overlay  
\$(OGRE\_HOME)/include/OGRE/Paging/  
\$(OGRE\_HOME)/include/OGRE/Terrain/  
\$(OGRE\_HOME)/include/OGRE/RTShaderSystem/

Evaluated value:

C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/Bites/  
C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/  
C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/Overlay  
C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/Paging/  
C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/Terrain/  
C:\Hooman\GBC\GAME3121\ogre-master\include\OGRE/RTShaderSystem/  
%(\AdditionalIncludeDirectories)

Inherited values:

☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# Additional Include Directories (V2)

Additional Include Directories

\$(OGRE\_HOME\_V2)/Components/Bites/Include  
\$(OGRE\_HOME\_V2)/OgreMain/Include  
\$(OGRE\_HOME\_V2)/Components/Overlay/Include  
\$(OGRE\_HOME\_V2)/Components/Paging/Include  
\$(OGRE\_HOME\_V2)/Components/Terrain/Include  
\$(OGRE\_HOME\_V2)/Build/Include  
\$(OGRE\_HOME\_V2)/Components/RTShaderSystem/Include

Evaluated value:

C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\Components/Bites/Include  
C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\OgreMain/Include  
C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\Components/Overlay/Include  
C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\Components/Paging/Include  
C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\Components/Terrain/Include  
C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\Build/Include

Inherited values:

☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# Additional Library Directories

Additional Library Directories

\$(OGRE\_HOME)/lib/

Evaluated value:

C:\Hooman\GBC\GAME3121\ogre-master\lib/  
%(AdditionalLibraryDirectories)

Inherited values:

☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# Additional Library Directories (v2)

Additional Library Directories

\$(OGRE\_HOME\_V2)/build/lib/Debug

Evaluated value:

C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1/build/lib/Debug  
%(AdditionalLibraryDirectories)

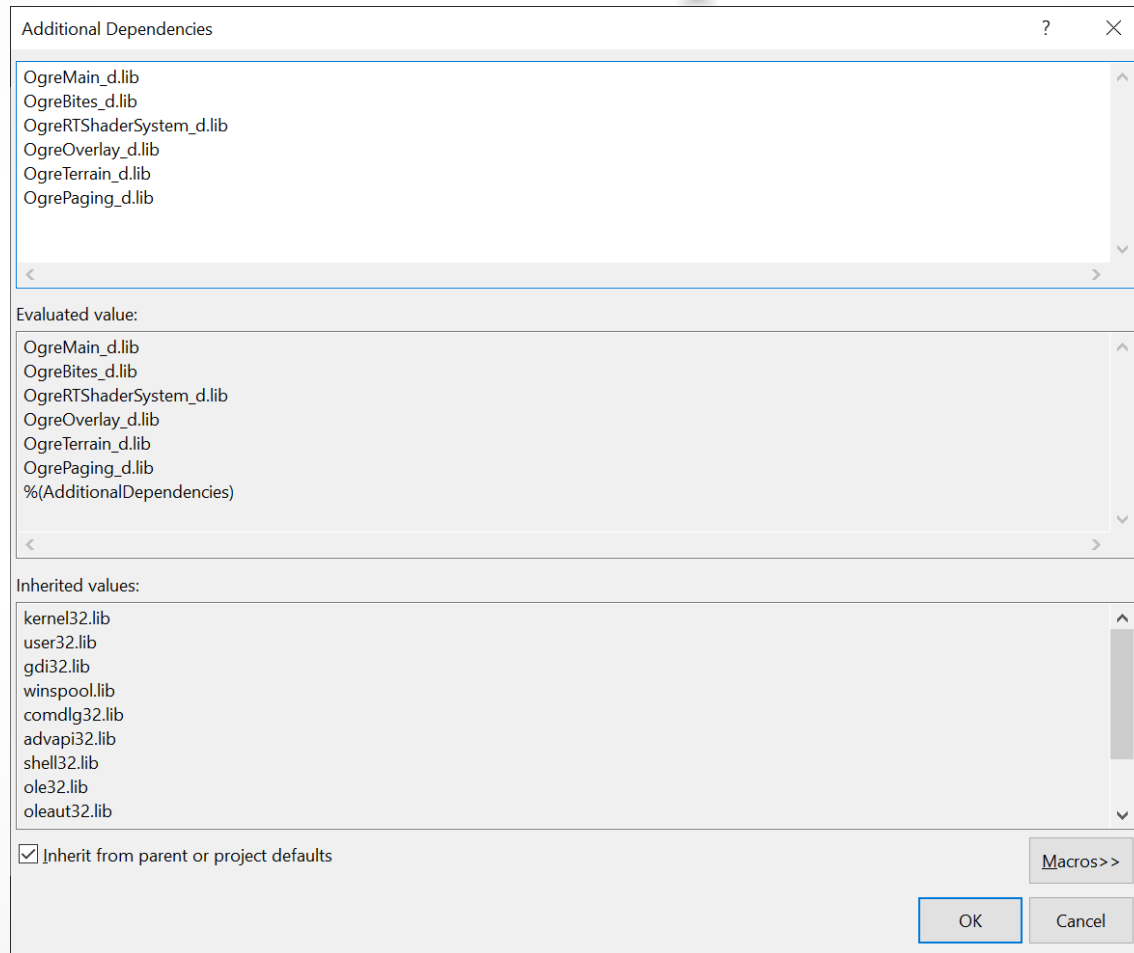
Inherited values:

☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# Additional Dependencies



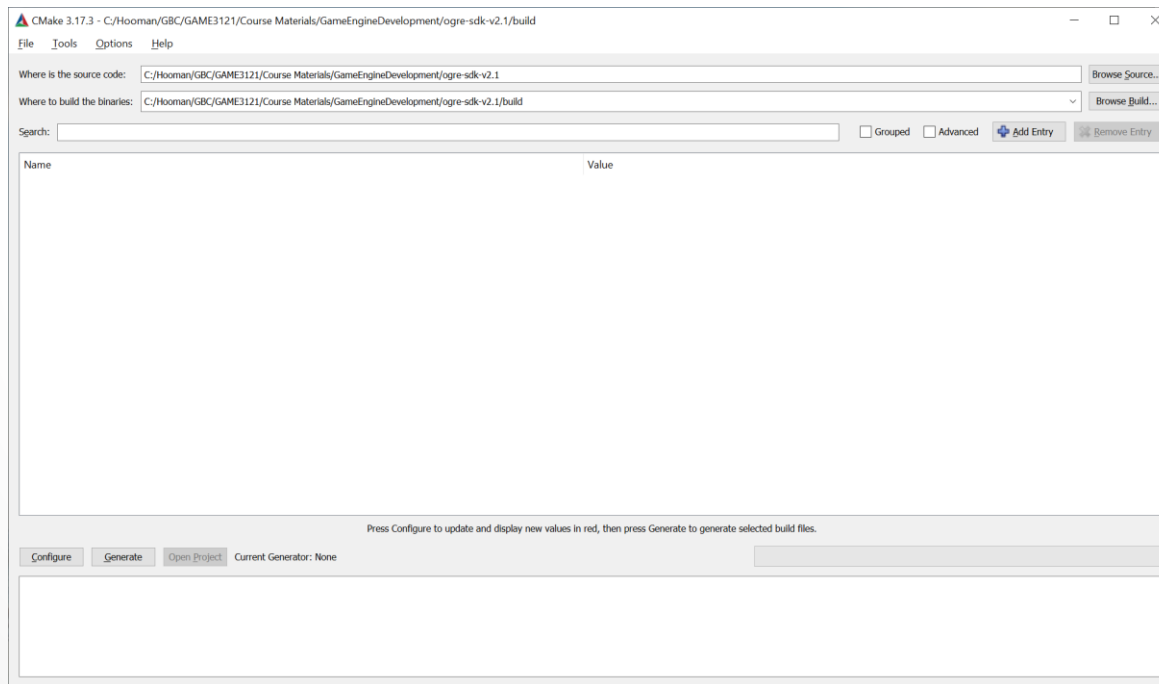
# V2 SDK

- <https://github.com/hsalamat/GameEngineDevelopment>
- Create this folder structure under your C drive
- C:\Hooman\GBC\GAME3121
- Copy [GameEngineDevelopment](#) folder from Github under C:\Hooman\GBC\GAME3121
- Go to C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\build
- Open Ogre.sln solution file
- Rebuild the solution

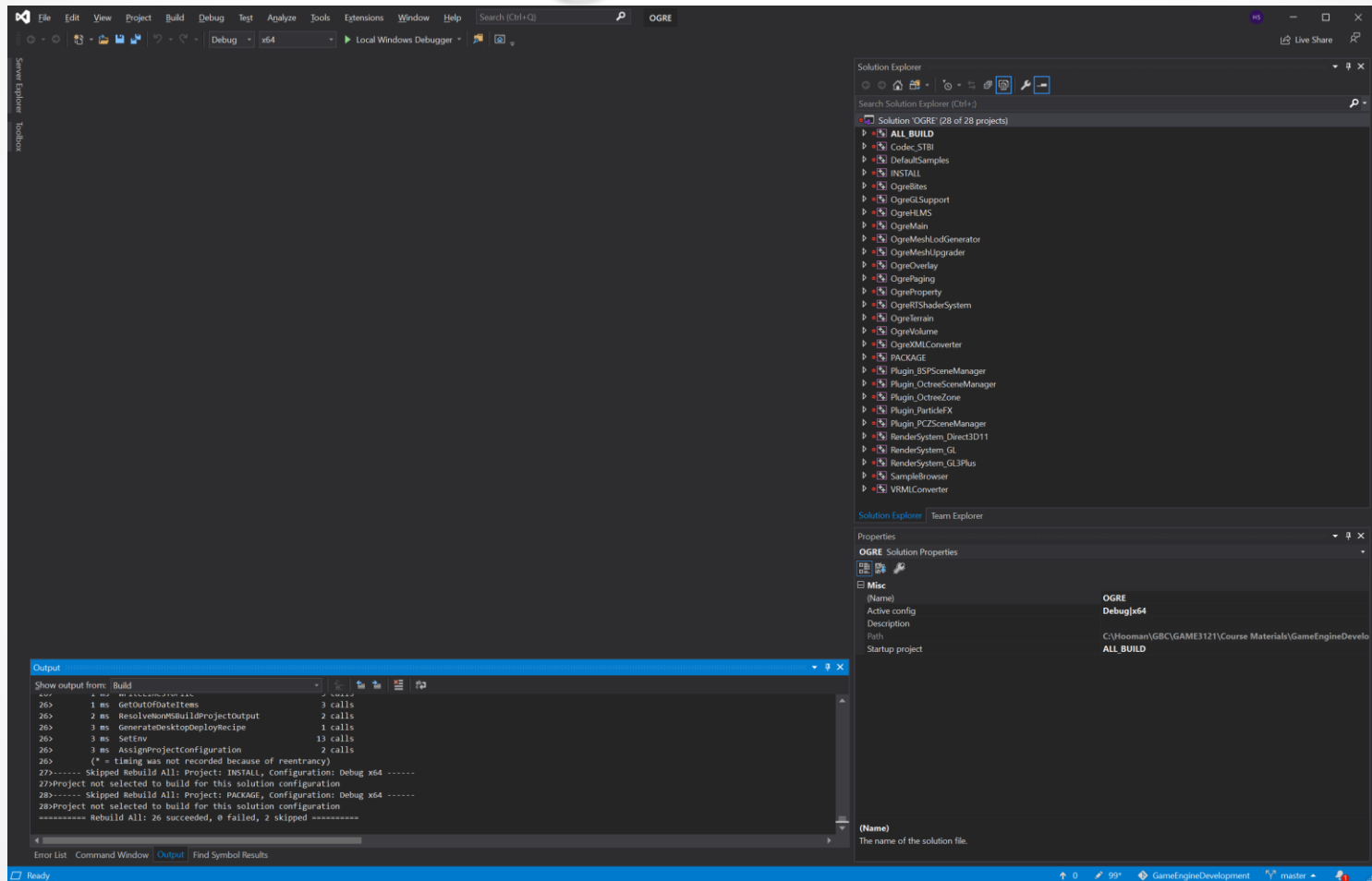


# OgreV2

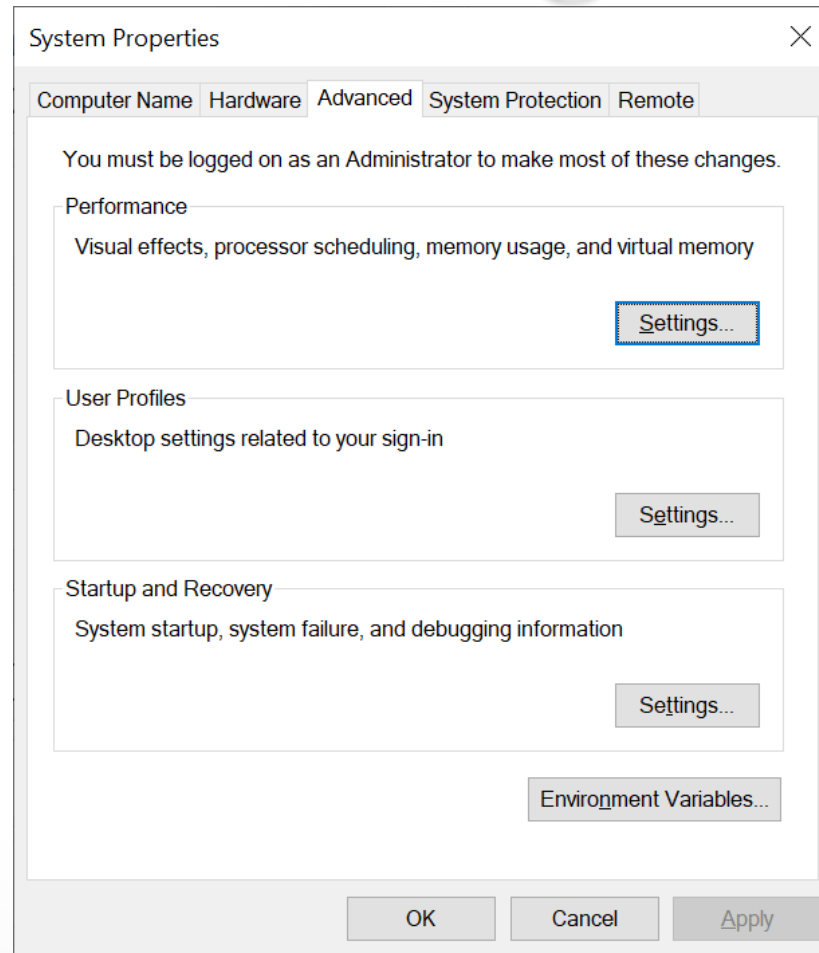
- I already build the solution and packages for you using Cmake. If you are curious, you can do it yourself as well!



# Ogre.sln

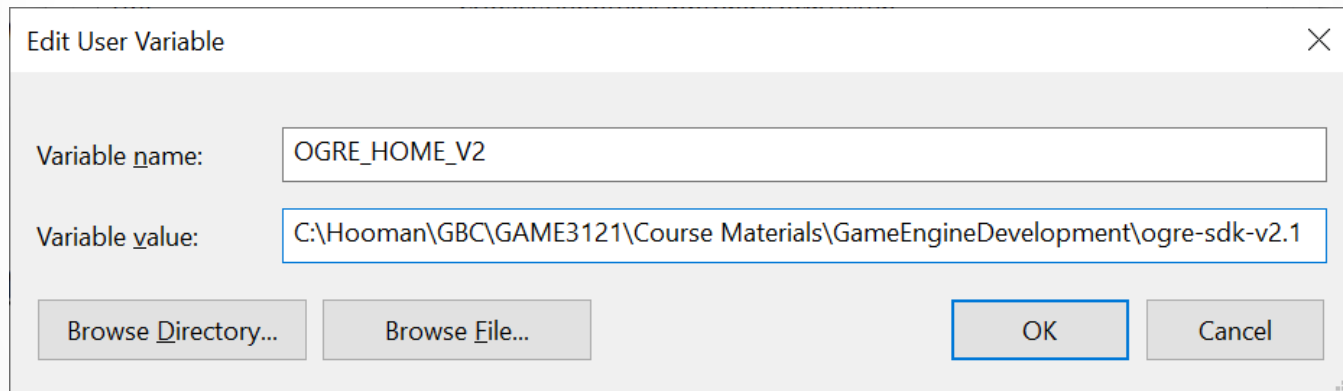


# Environment Variables Settings



# OGRE\_HOME\_V2

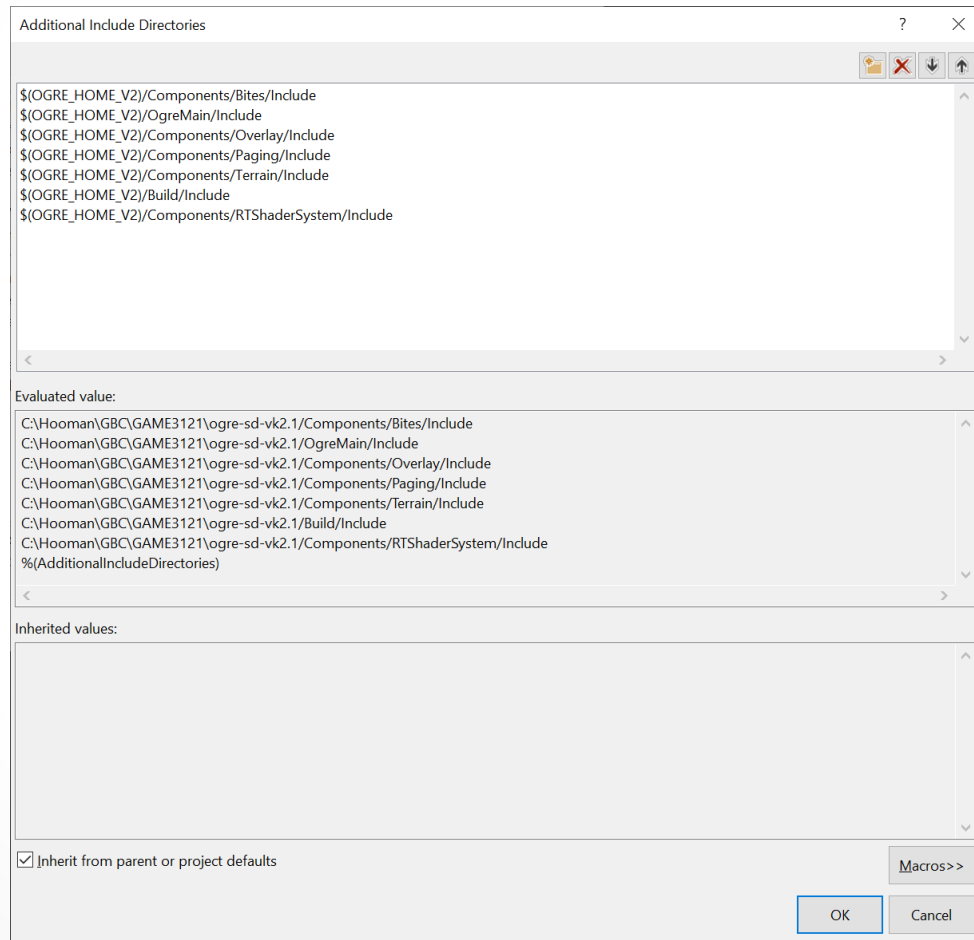
- Add OGRE Home environment variable. If you are using a different folder structure, this must be changed!



The screenshot shows a Windows 'Edit User Variable' dialog box. The title bar reads 'Edit User Variable' with a close button (X) on the right. The dialog has two main input fields: 'Variable name:' and 'Variable value:'. The 'Variable name' field contains the text 'OGRE\_HOME\_V2'. The 'Variable value' field contains the path 'C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1'. Below these fields are four buttons: 'Browse Directory...', 'Browse File...', 'OK', and 'Cancel'. The 'OK' button is highlighted with a blue border. The dialog box has a standard Windows XP-style appearance with a light gray background and a white border.

Field	Value
Variable name:	OGRE_HOME_V2
Variable value:	C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1

# V2 Additional Include Directories



# V2 Additional Library Directories

Additional Library Directories

\$(OGRE\_HOME\_V2)/build/lib/Debug

Evaluated value:

/build/lib/Debug  
%(AdditionalLibraryDirectories)

Inherited values:

☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# V2 Additional Dependencies

Additional Dependencies

OgreMain\_d.lib  
OgreBites\_d.lib  
OgreRTShaderSystem\_d.lib  
OgreOverlay\_d.lib  
OgreTerrain\_d.lib  
OgrePaging\_d.lib

Evaluated value:

OgreMain\_d.lib  
OgreBites\_d.lib  
OgreRTShaderSystem\_d.lib  
OgreOverlay\_d.lib  
OgreTerrain\_d.lib  
OgrePaging\_d.lib  
%(AdditionalDependencies)

Inherited values:

kernel32.lib  
user32.lib  
gdi32.lib  
winspool.lib  
comdlg32.lib  
advapi32.lib  
shell32.lib  
ole32.lib  
oleaut32.lib

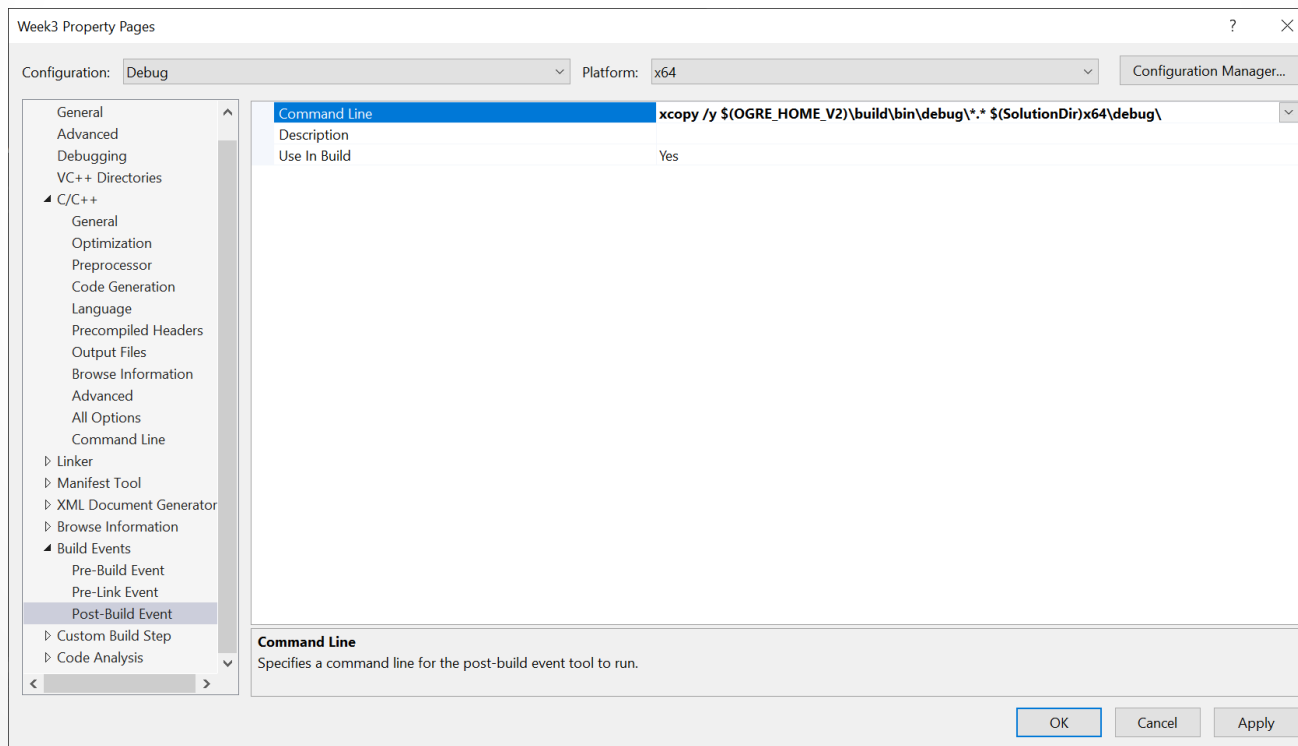
☒ Inherit from parent or project defaults

Macros>>

OK Cancel

# Post-Build Event


- `xcopy /y $(OGRE_HOME_V2)\build\bin\debug\*. * $(SolutionDir)x64\debug\`





# Export Template Wizard

Export Template Wizard



Choose Template Type

This wizard will allow you to export a project or project item from the current solution to a template which future projects can then be based upon.

Which type of template would you like to create?

☒ Project template

A project template will allow a user to create a new project based on your exported project. A user will be able to utilize your template from the New Project dialog box for client projects and from the New Website dialog box for websites.

☐ Item template

An item template will allow a user to add your item to one of their existing projects. Your template will be available to the user from the Add New Item dialog box.

From which project would you like to create a template?

OgreTemplateV2

< Previous

Next >

Finish

Cancel

# Ogre::Root

- Open `Ogre.sln` under `C:\Hooman\GBC\GAME3121\Course Materials\GameEngineDevelopment\ogre-sdk-v2.1\build`
- `OgreApplicationContextBase.cpp`
- An instance of `Ogre::Root` must exist before any other Ogre functions are called.
- The first parameter to the constructor is the plugins configuration filename, which defaults to `plugins.cfg`.
- The second parameter is the main configuration filename, which defaults to `ogre.cfg`.
- The third parameter is the name of the log file where Ogre will write the debugging and the hardware information.
- ```
mRoot = OGRE_NEW Ogre::Root(pluginsPath, mFSLayer->getWritablePath("ogre.cfg"), mFSLayer->getWritablePath("ogre.log"));
```

- Once the `Ogre::Root` instance has been created, it can be globally accessed by `Root::getSingleton()`, which returns a reference or `Root::getSingletonPtr()`, which returns a pointer.
- `// get a pointer to the already created root`
- `Root* root = getRoot();`

# \*.cfg files

- The ogre.cfg configuration file contains Ogre 3D engine graphics settings.
- Once the main configuration file is loaded, we load the correct render system plugin and tell Ogre which render system to use.
- Next, we load the resources.cfg configuration file.
- The resources.cfg file contains a list of all the paths where Ogre should search for graphic resources.
- Then, we go through all the sections and settings in the resource configuration file, and add every location to the Ogre resource manager.

# Ogre scene manager

- Every graphics object in the scene including all meshes, lights, and cameras are managed by the Ogre scene manager.

```
SceneManager* scnMgr = root->createSceneManager();
```

- The Run Time Shader System or RTSS for short is the Ogre way of managing Shaders and their variations.

```
// register our scene with the RTSS
```

```
RTShader::ShaderGenerator* shadergen =  
RTShader::ShaderGenerator::getSingletonPtr();  
shadergen->addSceneManager(scnMgr);
```

# Camera

- If we want to see anything, we need to create a camera, and add it to our scene. The next bit of code does just that.

```
SceneNode* camNode = scnMgr->getRootSceneNode()->createChildSceneNode();
Camera* cam = scnMgr->createCamera("myCam");
    cam->setNearClipDistance(5);
    cam->setAutoAspectRatio(true);
    camNode->attachObject(cam);
    camNode->setPosition(0, 0, 140);
// tell it to render into the main window
    getRenderWindow()->addViewport(cam);
```