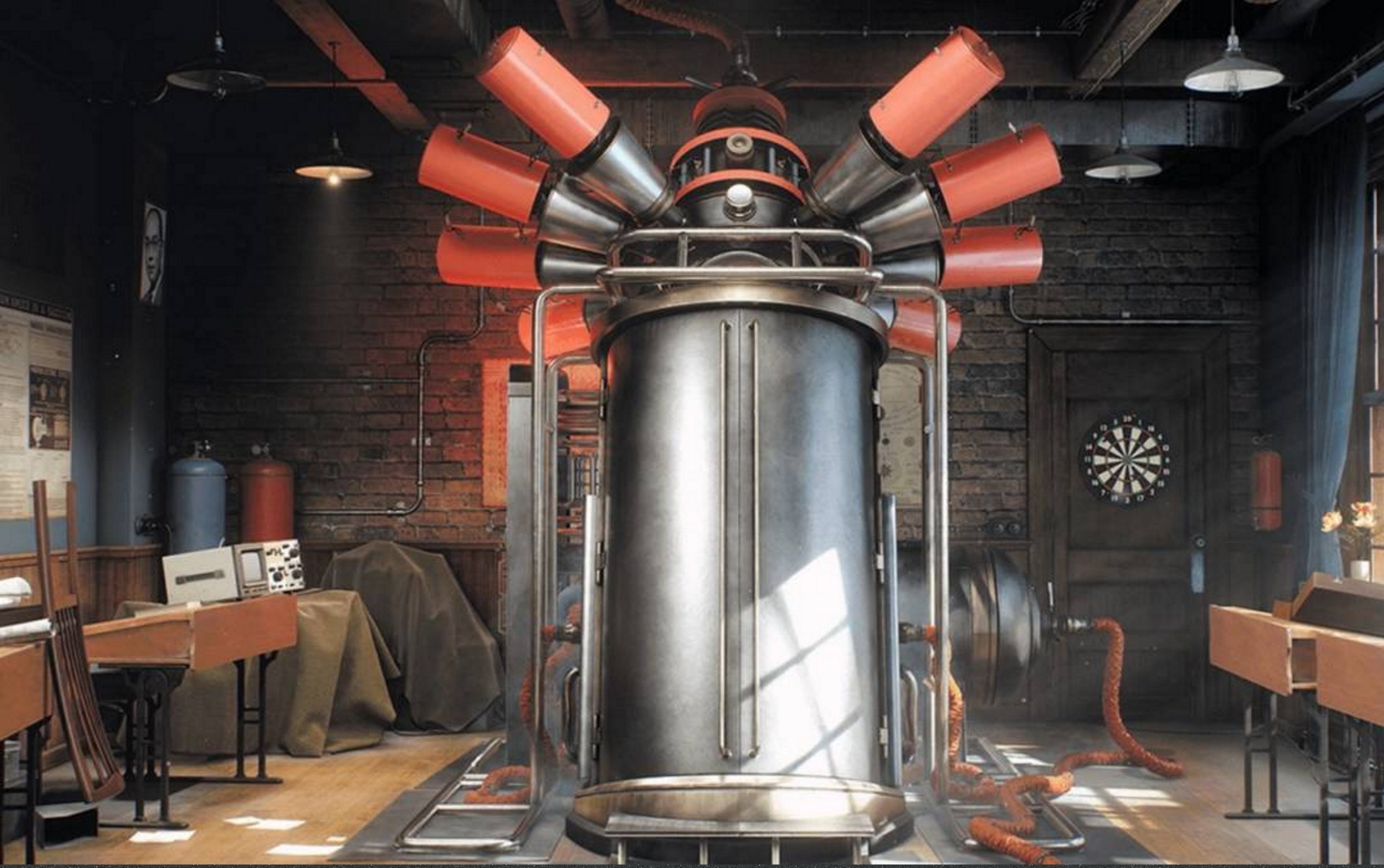


::UNIGINE

3D Game and Graphics Programming Using UNIGINE

C++, C# and CG Artist Perspective

Rohit Baptista Gonsalves

3D Game and Graphics Programming using UNIGINE

C++, C#, and CG Artist perspective

Rohit Gonsalves

This book is for sale at http://leanpub.com/unigine3d

This version was published on 2020-08-04

To my Family,

To my dad, Baptista, and mom, Lata to give me strength all the time. I Love you both.
To my wife, Harshada, she always stood by me and supported all my adventures and decisions.
To my children, Kaivalya, and Hridhan for inspiring me with questions like what is C++?
To my sister, to have fights that forced me to write the book chapters in anger.

I hope to see Kaivalya and Hridhan take this book to check what I had written about C++, once they grow little more to understand it.

Contents

CHAPTER 1: Install and Setup UNIGINE-2

UNIGINE and System Requirements

Installing UNIGINE SDK Browser and Licenses

Setup A Windows development environment

Summary

CHAPTER 2: Introduction of UNIGINE Engine

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Engine Features

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Engine Fundamentals and Build Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Logging, Start-up Options, Configuration files, and Console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

File Systems and Performance Analyzer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 3: Creating your first Project

We have installed the UNIGINE engine community edition and Microsoft Visual Studio community edition. We have also gone over understanding the features and fundamentals of the UNIGINE engine. It is time to smash the hammer and to create your first project. We will discuss a project creation, project configuration, files, and folder structure of the project. We will do an introduction to the UNIGINE editor, and open the content in the editor. We will understand the asset workflow and files created by it. We will conclude the chapter by running the project via code. Undoubtedly we will generate projects in suitable formats to work with C++ and C# language.

Create Project in SDK Browser

A project is an independent entity of the UNIGINE engine that contains all the data, source code, and binaries to run the content. A project comprises of conceptual design and organized in a set of directories (with typical folder structure). This folder structure remains identical for all projects created through the editor. As we have already installed the SDK and activated it, we can create content projects using the same SDK.

NOTE: UNIGINE releases new SDKs that appear in the SDK section of the browser. The recent releases are hotfixes, upgrades, or a completely new version of SDK. You can install the latest SDK and make it a default SDK. The new project creation function uses the default SDK for project configuration.

NOTE: By default, the UNIGINE SDK browser is installed at <user account>/UNIGINE SDK Browser/sdks. You will find all the installed SDK at this location.

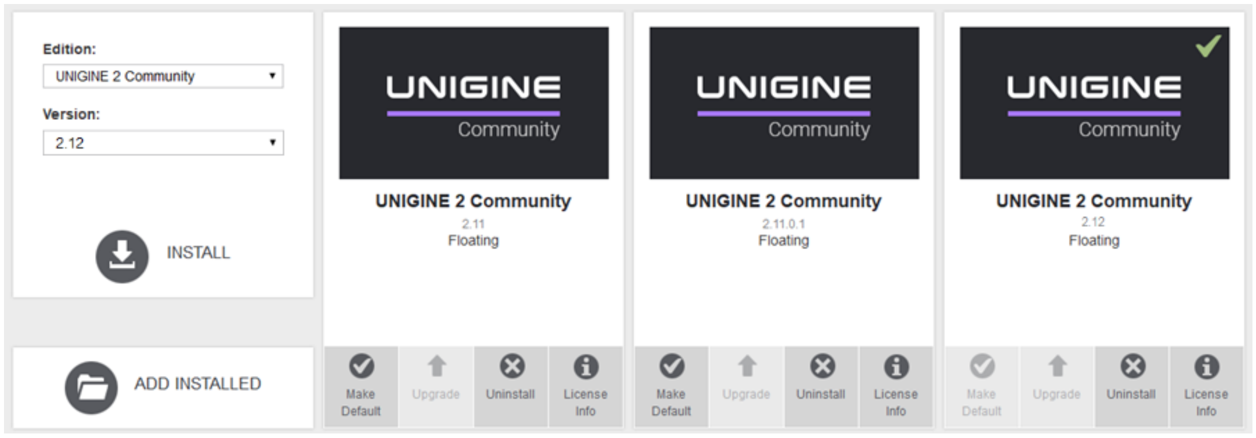
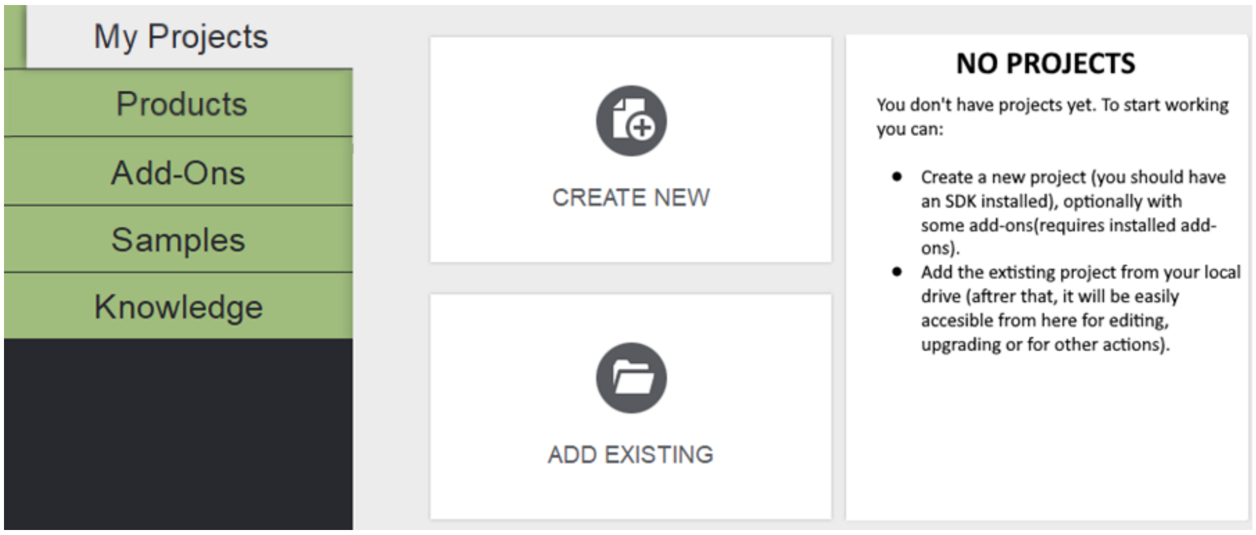


Figure 3.1: The picture showing the installed Product releases and default Product selected. The default selection of the Product is 2.12 version.

Let's hop to create the first project. Launch the UNIGINE SDK browser. Select My Projects tab.



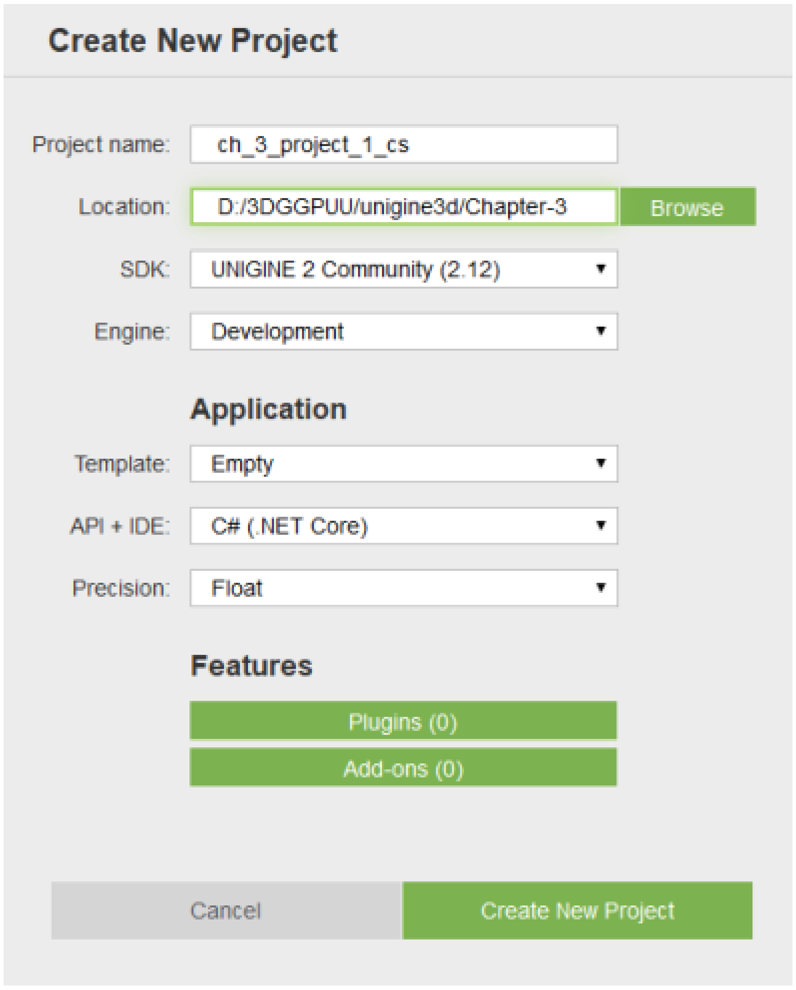
NO PROJECTS

You don't have projects yet. To start working you can:

- Create a new project (you should have an SDK installed), optionally with some add-ons(requires installed add-ons).
- Add the extisting project from your local drive (aftrer that, it will be easily accesible from here for editing, upgrading or for other actions).

Figure 3.2: My Projects section shows no project as we have not yet created any.

Click on the new project tab, the new project creation window will open. The new project creation window is a tool for configuring new projects. We select the project name, location to store it, SDK, and engine configuration type. For this first project, select the configuration options shown here. We will keep changing these configuration options later and create the projects as we need.



Create New Project

Project name: ch_3_project_1_cs

Location: D:/3DGGPUU/unigine3d/Chapter-3

SDK: UNIGINE 2 Community (2.12)

Engine: Development

Application

Template: Empty

API + IDE: C# (.NET Core)

Precision: Float

Features

Figure 3.3: New Project configuration window. Here you may specify name, location, SDK, and development build. There are a few more options to select templates, IDE, and precision. Ass plugins and add ons.

The first few parameters are easy to understand. I have given a name to the project, selected a location on my pc to save the project. I have installed only community edition SDK, 2.11.0.2, and chose it. If you have more than one SDK installed, you will see other options listed in this selection. The Engine parameter is very interesting. The configuration is the right term than the term engine. It is an important parameter to do any coding with the project. As we will code, we require to debug our code, to support it, we need debug engine linking files and executables with debugging metadata. We select the development option to build the development environment created by the project. It includes debugging as well as release build configurations and required files. To create our first project, we will select the Development option.

Watch that progress bar and have a project ready like this.

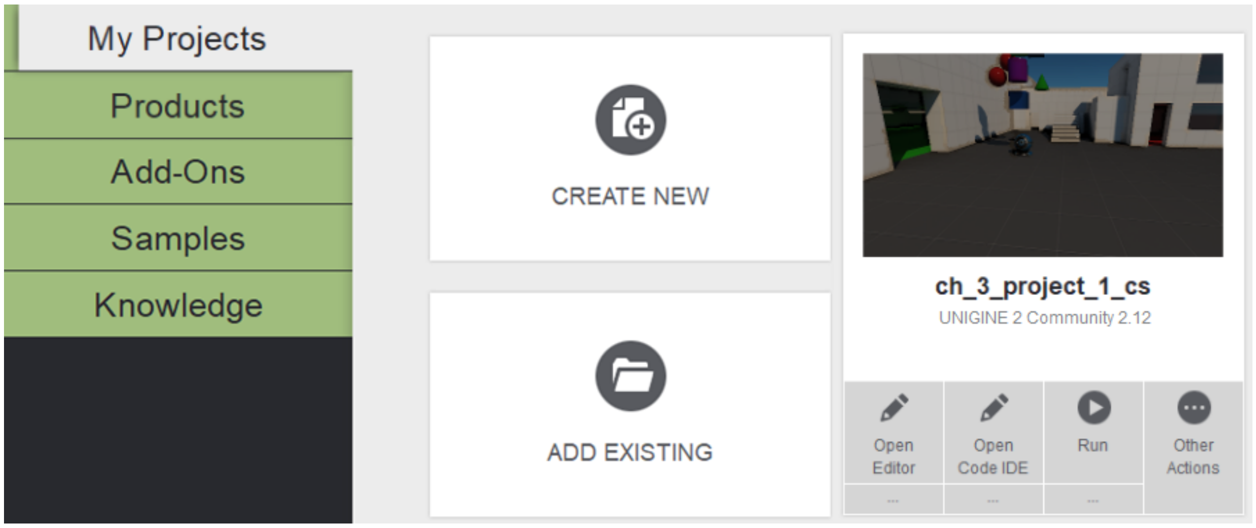


Figure 3.4: New Project configured and added to the list of My Projects tab.

My projects tab will now have a tile with some simple scenery picture, Title of the project, SDK version, and few options to work. Hit the Run button, the obvious first choice. The new console window will open, alongside one more window that will pop up with UNIGINE rendering. You can use mouse and W, S, A, D keys to move the virtual camera, and navigating in the world. Look that camera is colliding quite nicely with walls and hopping on stairs if you climb on one. The camera will bump onto a few scattered objects, and they will react to the collision. All this is automatic. Wonderful Start!

The other console window will log all the messages, warnings, and errors discussed in the 2nd chapter. You may even click in the rendering window and Press the "' (situated before number 1) button to see the in-game console. We have created the first project and executed it.

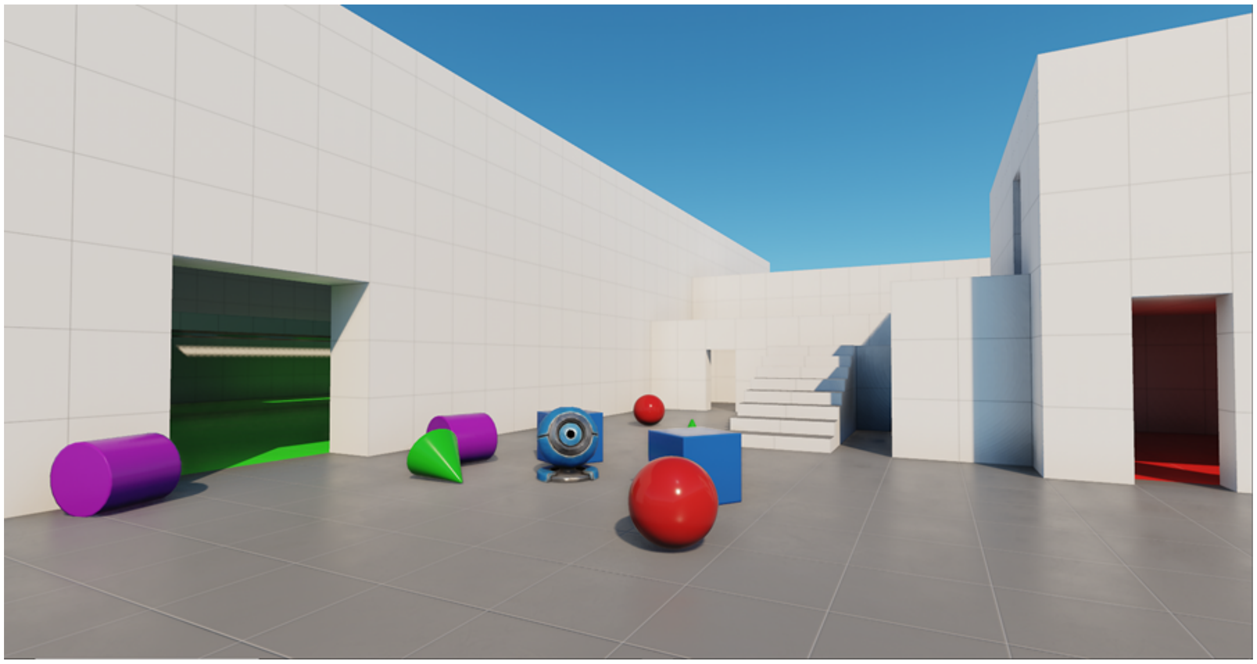


Figure 3.5: UNIGINE rendered default content after the creation of the project. There are some indoor areas, exterior areas with the sun, scattered objects to have a collision with, stairs and walls.

Close the engine. Select the Open Editor option from the project tile, and the editor will open up. The editor will start compiling shaders and then will show the complete editor with different panes. It is an excellent tool to create the virtual world, add assets, convert, and compress them. Make the composition of the world and add graphical objects. Create cinematics, generate builds, and do much more.

Once you work with an editor, close it. You will notice a change in the project tile. Now you will see a thumbnail of your project's content nicely painted over the project tile. It is an easy way to distinguish between many projects visually.

There is an 'Other action' option on the project tile. Click on it, and you will see three more options.
• Open Folder – It navigates to the physical location of project files.
• Configure Project – Opens a configuration window again to reconfigure the project settings.
• Delete – Deletes the project from the browser and may delete the data from the physical location if permitted.

There are many options we have not discussed yet. We will check them when the need will arise to change them or select them in the upcoming topics.

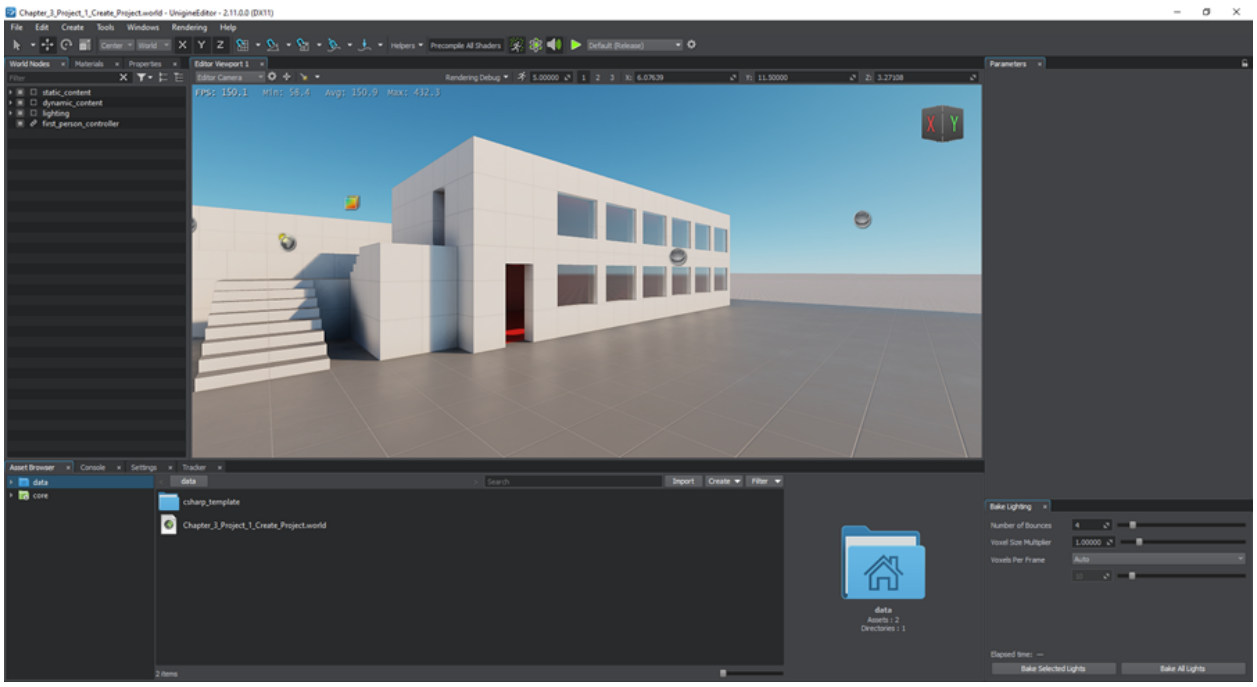


Figure 3.6: UNIGINE editor with main editor viewport, World hierarchy, asset browser, and parameter window.

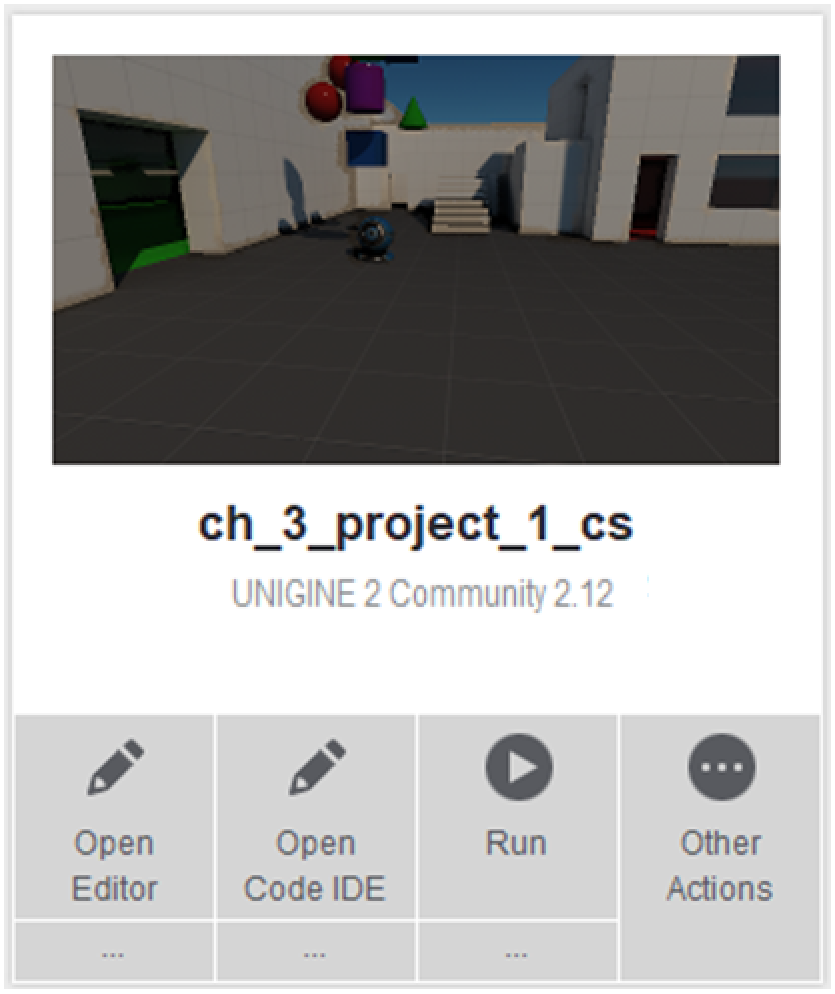
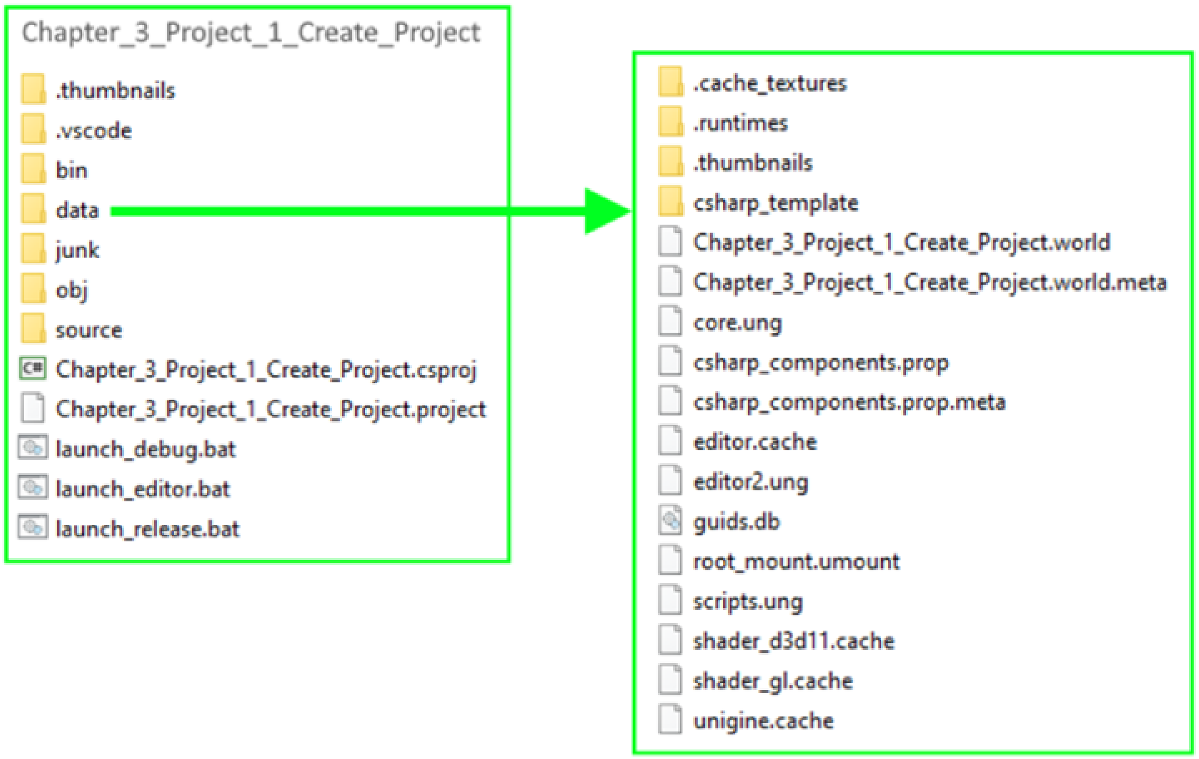


Figure 3.7: The project tile displays the thumbnail of the content of the project.

UNIGINE SDK Browser Project Format

Congratulations! You have just set up the first UNIGINE project. You have executed it and also ran it via the editor. We have seen the front-end. What has been created by the browser? What exactly happened in the background? The files and directories are generated automatically for the project, which are they? You had selected the location for the project. Navigate to that location or click on the project tile, select the other actions, and select the open folder option.
On the creation of a new UNIGINE project, you are creating a folder with the same name as your project. This folder includes the following:


Chapter_3_Project_1_Create_Project
.thumbnails
.vscode
bin
data
junk
obj
source
Chapter_3_Project_1_Create_Project.csproj
Chapter_3_Project_1_Create_Project.project
launch_debug.bat
launch_editor.bat
launch_release.bat

.cache_textures
.runtimes
.thumbnails
csharp_template
Chapter_3_Project_1_Create_Project.world
Chapter_3_Project_1_Create_Project.world.meta
core.ung
csharp_components.prop
csharp_components.prop.meta
editor.cache
editor2.ung
guids.db
root_mount.umount
scripts.ung
shader_d3d11.cache
shader_gl.cache
unigine.cache

Figure 3.8: The project directory structure. It lists folders for the asset workflow and data it contains.

Table 3.1 – The basic structure of the root folder of a UNIGINE project. It lists the directories and purpose of those directories.

.thumbnails/	It contains asset thumbnails to display in the asset browser. As you import new assets, the thumbnails are generated automatically by the editor.
.vscode/	It contains visual studio code user and workspace setting files. These settings are mandatory to launch our C# project inside the visual studio code.
bin/	It contains all necessary binary executable files and libraries of the UNIGINE engine, editor and plugins as well as binary executable files of your project.
data/	It includes all the assets imported and used in your project in both native and non-native formats. It is the folder where the project content is managed, processed, and stored.
data/.cache_textures	It contains texture cache files — minimized copies generated for all texture runtimes and used by the engine's data streaming system. The engine generates all these files automatically on the fly. You should not put the content of your project here.

data/.runtimes	It contains files in UNIGINE native formats used by the engine at run time as compressed DDS textures, mesh geometry, .anim animations, and other). UNIGINE editor generates these files for all assets that are not in the UNIGINE's native format (like .fbx, .obj, .hdr, etc.). It is an auto-generated directory, and you should not put the content of your project here.
data/.thumbnails	It contains the thumbnail of the content that displayed over the project tile in SDK Browser.
source/	It stores the source files of your application logic as the selected API. These files are editable in an IDE or any text editor. After any changes made to these files, it is necessary to rebuild the application or its parts. Upon the selection of C++ and C# API, the SDK browser creates this folder.
*.project	The project file includes all necessary metadata and settings for your UNIGINE project accessible from the SDK Browser.
launch_debug	Launcher for the debug version of your UNIGINE project (.bat or .sh file).
launch_editor	Launcher for your UNIGINE project with the UNIGINE Editor (.bat or .sh file).
launch_release	Launcher for the release version of your UNIGINE project (.bat or .sh file).

NOTE: We have created the first project using C# (.NET Core) API. There are more folders created viz. junk and obj. These folders are not part of the project when we create a C++ project. The C++ project has other folders viz. include folder, lib folder, and utils folder.

We are going to learn the asset workflow and adding models, textures, animations, and materials to the editor. The physical assets that we add to the project are converted to runtime files automatically. Each newly generated file has assigned a globally unique identifier (GUID). This process helps to remove the dependency on physical assets. We will discuss in detail about this in Asset workflow discussion. The most important thing you need to remember is to respect the file system and folder structure of the project format. Import all your assets to the data folder of the project. All your custom assets must go in this folder. It is the strict rule from this point onwards.

Introduction to UNIGINE Editor

We have already run the editor once to execute the project in the editor. Probably taking a short look at the editor, you have closed it to learn further here. Now is the time to know more about the editor. The UNIGINE Editor provides the core functionality for the creation and editing of virtual worlds for UNIGINE-based applications based on the assets workflow. It allows you to easily view and modify virtual worlds by adding, transforming, and editing the nodes.
Besides, UNIGINEEditor also provides:

• Customizable UI layout with dockable panels
• Integration with the Asset Browser

Let us check the most necessary parts of the editor. We will discuss other parts of the editor in detail in the proceeding chapter. All kinds of layouts are possible, and this is more general to all readers here. So we are jumping to the topics of viewports.

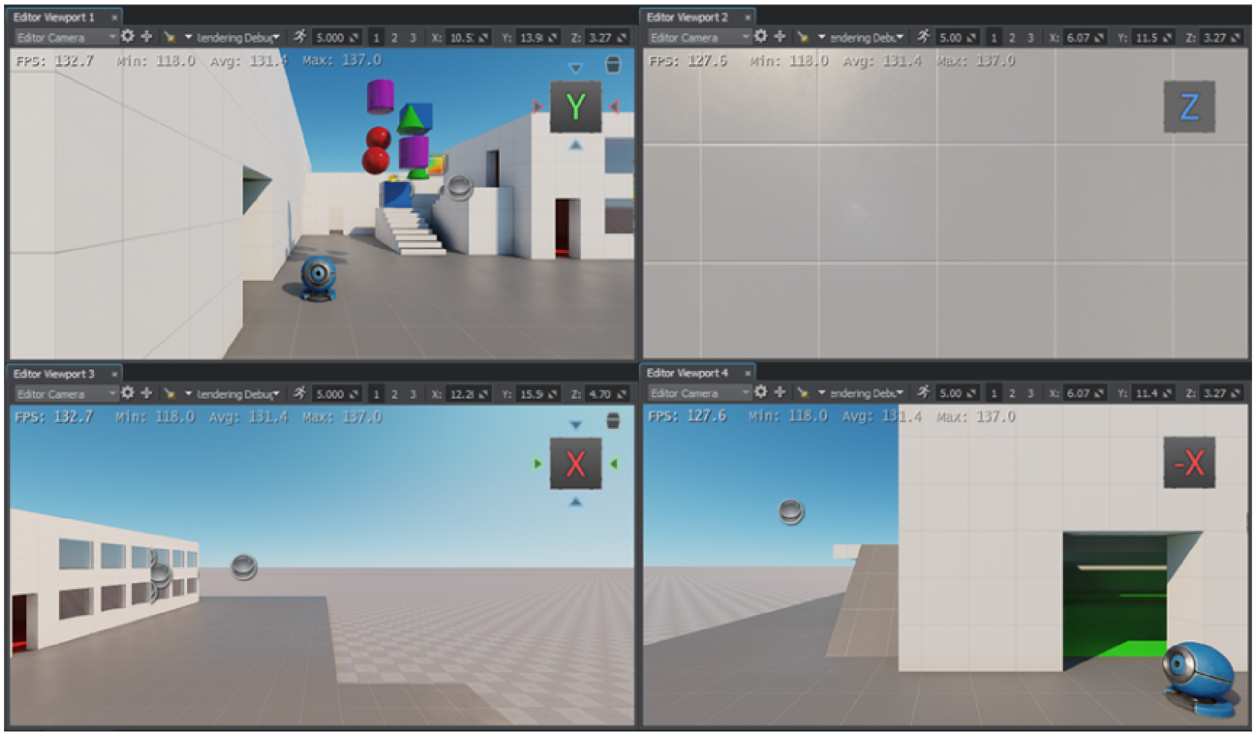


Figure 3.9: The Editor viewports with different views: front (top- left), top (top-right), right (bottom-left), and left or negative X (bottom-right)

The Add Editor Viewport is used to add the editor's viewport to the editor. You may customize the location for the added viewport. As you will add the viewport window, the window menu will list all of them on this menu. As shown in figure 3.10, you may hide or remove the viewport as your need.

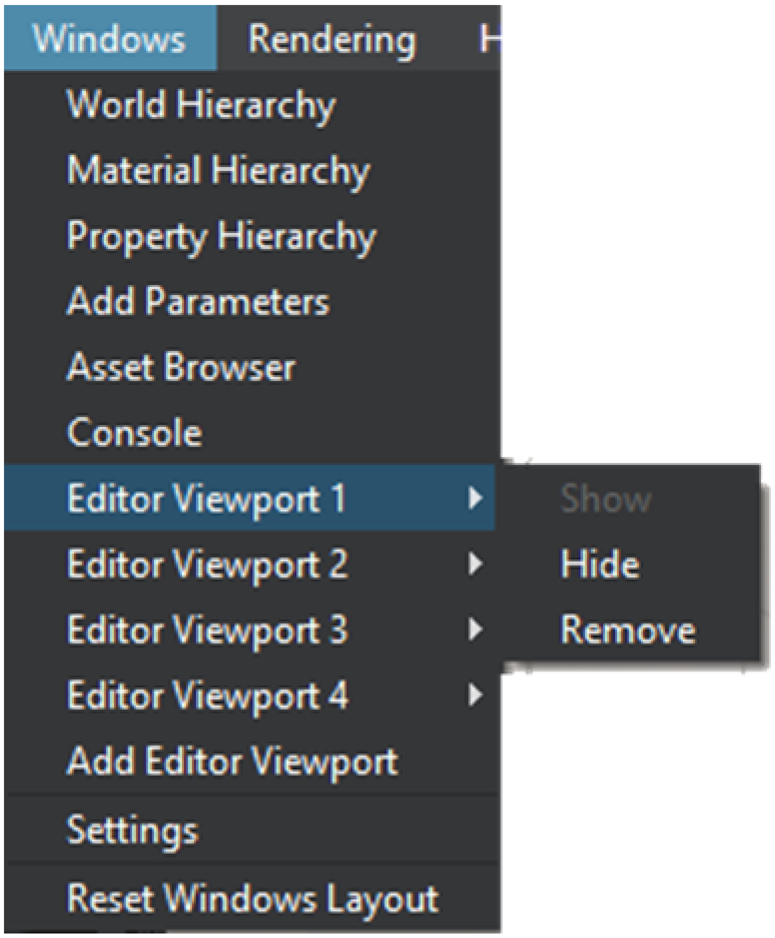


Figure 3.10: The Window menu in the editor have Add Editor Viewport menu item. Selecting this menu adds a new editor viewport to the editor. The window menu lists all the editor viewports added.

All editor viewports have a view-cube, to set up selected camera views. Further, you can change the type of projection to perspective or orthographic projection using the view cube. The editor toolbar has the most common node manipulators. Let us check them in short.

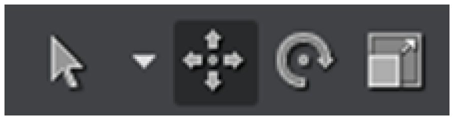


Figure 3.11: The editor toolbar, node manipulators. Node Selection, Translation, Rotation, and Scale manipulator.

The node selector manipulator helps to select nodes eighter in window mode or crossing mode.

- Window Mode requires the whole object to be inside the selection box.
- Crossing Mode makes the object selected if any part of it is inside the selection box.

NOTE: Inverse selection is useful for discard objects from selecting them. For example, if you need to select all objects in a scene except the few ones, you can select those specific objects and invert the selection. To do this, press Ctrl+I.

To move the node translation manipulator is used. After selection on the node, arrow keys can move nodes along the X and Y-axis. Using the mouse, you may select the axis, red, blue, or green and drag the mouse to add a translation. To move along a particular two-dimensional plane, you may use the rectangle manipulator.

NOTE: Besides, you may
1. Ignore the node's hierarchy when moving the selected node: press Alt and translate the node without its children
2. Translate the selected node to the camera by pressing Alt+X

To add orientation to the node, use rotation manipulator. After selection on the node, arrow keys can rotate nodes along the X and Y-axis. Using the mouse, you may select the sphere axis, red, blue, or green and drag the mouse to add a rotation along the chosen axis. Drag on the sphere to rotate freely around several axes.

To add scaling to the node, use a scale manipulator. After selection on the node, arrow keys can scale nodes along the X and Z-axis. Using the mouse, you may select the axis, red, blue, or green and drag the mouse to add scaling to respective axes. To scale along with a particular two-dimensional plane, you may use the rectangle manipulator. The white manipulator at the center can scale the node along three axes.

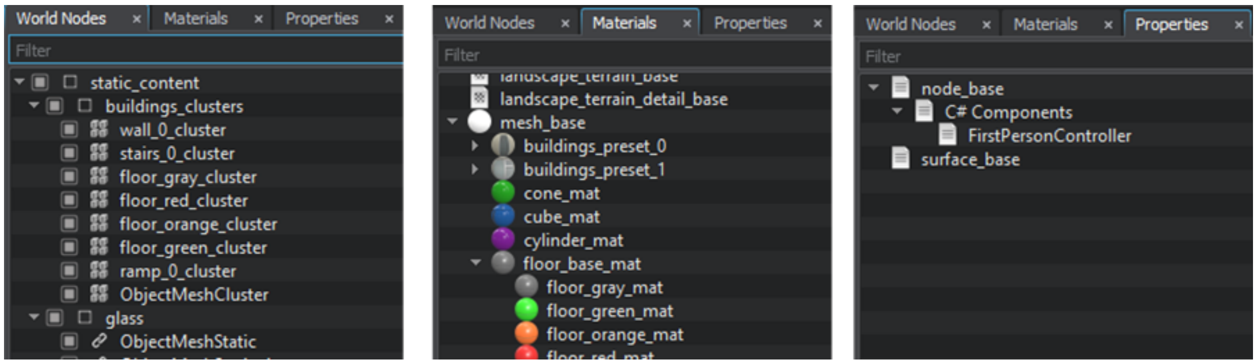
World, Material, and Properties Hierarchy.

World Hierarchy

The World Hierarchy window is a convenient tool for working with the hierarchy of nodes present in the scene. It allows you to:

• Organize nodes into a hierarchy
• Find a node by its name a quickly navigate to its location
• Perform basic operations with the selected nodes (clone, remove, and export)

To open the World Hierarchy window, choose Windows -> Toggle World Hierarchy in the Menu Bar.


World Nodes × Materials × Properties ×
Filter
static_content
buildings_clusters
wall_0_cluster
stairs_0_cluster
floor_gray_cluster
floor_red_cluster
floor_orange_cluster
floor_green_cluster
ramp_0_cluster
ObjectMeshCluster
glass
ObjectMeshStatic

World Nodes × Materials × Properties ×
Filter
landscape_terrain_base
landscape_terrain_detail_base
mesh_base
buildings_preset_0
buildings_preset_1
cone_mat
cube_mat
cylinder_mat
floor_base_mat
floor_gray_mat
floor_green_mat
floor_orange_mat
floor_red_mat

World Nodes × Materials × Properties ×
Filter
node_base
C# Components
FirstPersonController
surface_base

Figure 3.12: The Nodes Hierarchy, Material hierarchy, and property hierarchy windows.

The three elements make up the final UNIGINE composition for the world. Nodes Hierarchy, Material Hierarchy, and Property Hierarchy.

Material Hierarchy

Materials Hierarchy window organizes and modifies UNIGINE materials. It allows you to:

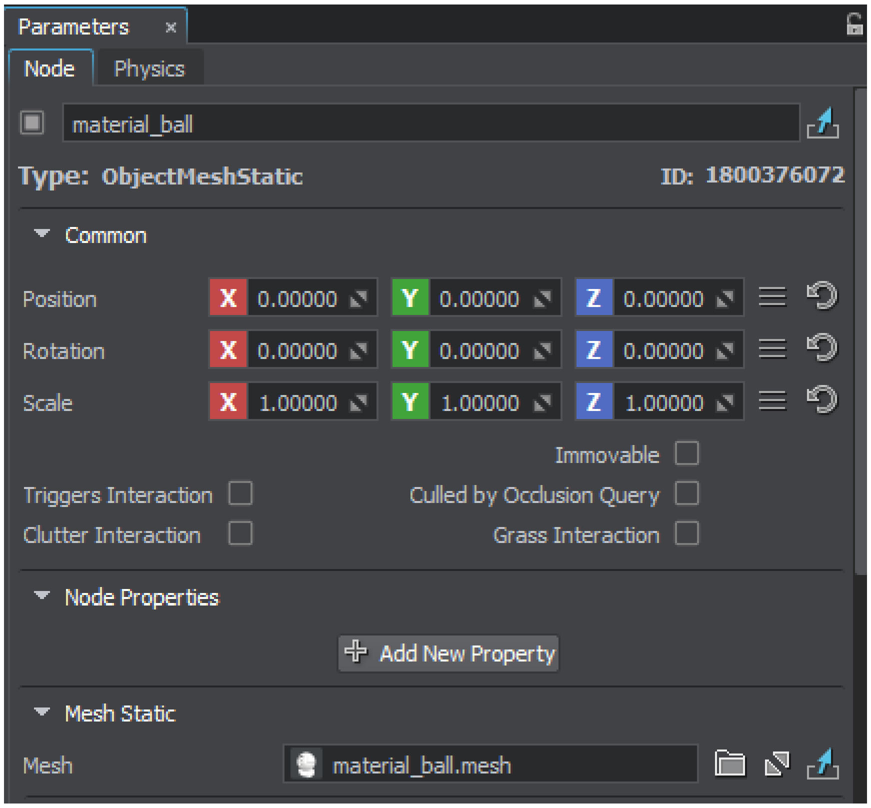
- Get instant access to built-in UNIGINE materials
- Inherit the materials and edit materials
- Organize the material entity into a hierarchy
- Perform basic operations on materials (clone, remove, rename and other)

To open the Materials Hierarchy window, choose Windows -> Toggle Materials Hierarchy in the Menu Bar or press M.

Properties Hierarchy

Properties Hierarchy window modifies and organizes nodes properties (sets of custom options). It allows you to:

- Organize properties into a hierarchy
- Perform basic operations (clone, remove, rename, and other.)
- Enable property-specific options


Parameters ×

Node | Physics

material_ball

Type: ObjectMeshStatic ID: 1800376072

Common

	X	Y	Z
Position	0.00000	0.00000	0.00000
Rotation	0.00000	0.00000	0.00000
Scale	1.00000	1.00000	1.00000

Immovable ☐
Triggers Interaction ☐ Culled by Occlusion Query ☐
Clutter Interaction ☐ Grass Interaction ☐

Node Properties

Add New Property

Mesh Static

Mesh material_ball.mesh

Figure 3.13: The Parameters window. It is a multi-purpose window, shows parameters of the selected asset type.

A multi-purpose Parameters window allows you to modify parameters of any element selected in the World, Materials, or Properties window, as well as in the Asset Browser. It offers the following features:

- Quick access to all parameters of the selected object
- Simultaneous editing of multiple assets types of the same type
- Copying and pasting of transformation parameters and surface parameters by using the context menu

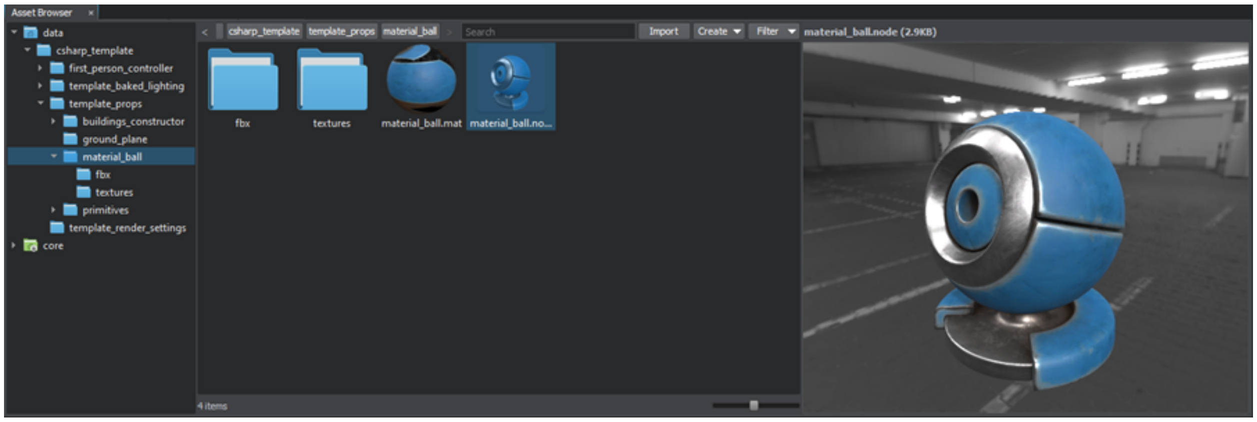


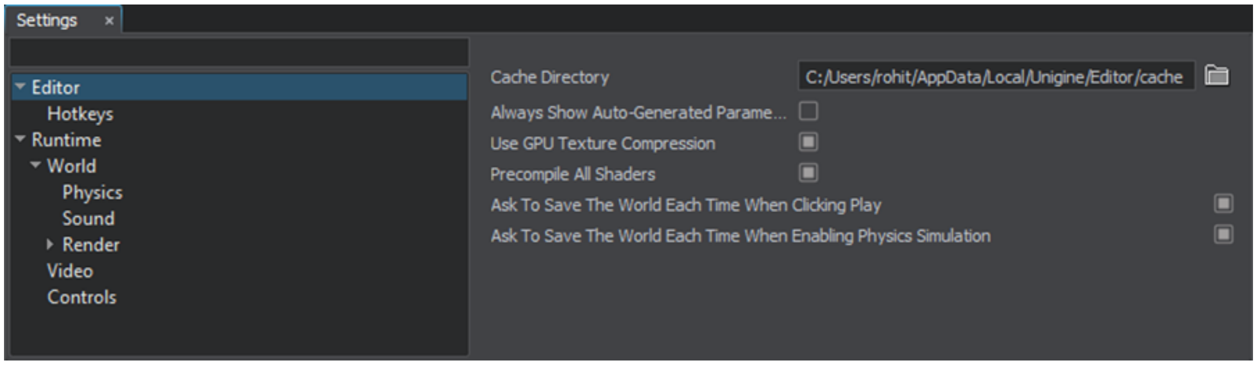
Figure 3.14: The Asset Browser window showing, assets directories, asset files, and asset type preview.

Asset Browser organizes content in your project. The Assets System keeps all links and dependencies between the resources when you edit, rename, or move them within the project. It allows you to:

- Create and import assets
- Organize your assets: rename, move them between the folders and manage their hierarchy
- View assets and add them directly to the scene

The Editor Settings Window

The editor settings window is an essential commodity in the editor. It allows us to set up setting parameters for most of the global settings as well as all render settings for the rendering engine.



Cache Directory	C:/Users/rohit/AppData/Local/Unigine/Editor/cache
Always Show Auto-Generated Parame...	☐
Use GPU Texture Compression	✓
Precompile All Shaders	✓
Ask To Save The World Each Time When Clicking Play	✓
Ask To Save The World Each Time When Enabling Physics Simulation	✓

Figure 3.15: The Editor Settings Window. Editor, world, render, global physics, global sound, and control settings.

The most common settings configured from this window:

1. Editor Settings – General editor settings and hotkeys selection for the editor
2. World Settings – World related settings
3. Global Physics Settings – Physics global parameters. All physical objects in the scene take the values from the global physics parameters from here
4. Global Sound Settings – Sound global parameters are assigned to all sounds in the scene
5. Render Settings – The Render section of the Settings window allows adjusting the rendering settings of the world

6. Video Settings – Vertical synch of the monitor or display
7. Controls Settings – The settings for the keyboard control keys and defining the mouse behavior

There are many other editing-related information and elements. We will take a look at them as the need arises of those tools, parts, or windows. Now you are aware of the necessary stuff to work within the editor. It is time to leap towards asset workflow and how to import assets in the editor.

Asset Workflow: GUID, Runtime files creation, and handling of assets.

The first thing we make with the editor is the project; it loads it and runs it. The complexity from this point onwards will increase with each additional feature of the editor explained. Exactly, like other software, it does have a learning curve. The learning curve is not steep, and it is fun to learn the editor's features. The next feature we are targeting is asset workflow. Asset Workflow is a mechanism to unify file management within the UNIGINE project. The file management revolves around the content of the project. The content is a collection of many elements like textures, geometries, materials, nodes, and many other things. At the core level, these elements are called assets. Some assets are compound assets, collection of base assets. The asset is the unit of work; it represents an item that is inside your world or project. An asset may come from a third-party application, such as a 3D model, an audio file, an image, or any other type supported by the UNIGINE Engine. They are non-native assets. The UNIGINE editor creates native-assets viz, a node, a material, or a property.

Let us dig a little to know what it is? Revisit figure 3.14. The picture shows the Asset Browser window in the UNIGINE editor. Asset Browser has a folder view, file view, and preview panel to view the selected asset. It provides functions to import different types of assets, conversion and managing them. We will check the asset workflow with the help of an image.
Now select the project tile and click other options. Open the folder to explore the base location of the project. Navigate to the data folder. Find the file guid.db and open it in any editor. Open it in either Visual Studio editor or Notepad++.

Listing 3.1: The guid.db file contents. The contains a listing of pairs in terms of guid and file location.

```
{
        "c3d83ed67a59683d40768c787623e577f24e046a": "core/materials/default/water/water_glo\
bal_beaufort_12.mat.meta",
        "33c598a780993ddadd25f3715c64561c50927acb": ".cache_textures/cc/cc9700e0e0533ebfec5\
03d122b993949182af214.dds",
        "1e6132e97a685f3455be810317cdb286d618682d": ".cache_textures/fd/fdce52252458c90169b\
884999ed107ed4f612ffe.meta",
        "3b3794ebee8a254e08edec5590fad1b002e7b94f": "core/systems/tracker/editor/images/tra\
cker_line_add.png",
.
.
.
.
        "ec34947401557120a3f8bd54853468ffadf930d6": "core/materials/default/water/water_glo\
bal_beaufort_3.mat",
        "2fd41edf32485eb2df38ae69300ea4b498610883": ".cache_textures/f1/f1806859da38b5e4ad0\
d07f3fdfe2cb6c5edbc9d.dds"
}
```

The file has a listing of pairs in terms of guid and file path. Let us consider one listing as an example from the above listings.

`"ec34947401557120a3f8bd54853468ffadf930d6":`
`"core/materials/default/water/water_global_beaufort_3.mat",`

UNIGINE uses a 40 byte-long guid created using the SHA1 algorithm. The above example shows the guid `"ec34947401557120a3f8bd54853468ffadf930d6"` and it is linked to the file `"core/materials/default/water/water_global_beaufort_3.mat"`. Now one questions arise here

1. Where is the core folder in the project?

Do you remember the rule? Import all your assets to the data folder of the project. All your custom assets must go in this folder only. It is the strict rule from this point onwards. Every project has a data folder, and this data folder is a root for all kinds of content. All the relative paths you will encounter in the UNIGINE file system are concerning the data folder. Back to question asked. Where is the core folder in the data folder then? You probably need to find the file "core/materials/default/water/ water_global_beaufort_3.mat". Here is an explanation.

The UNIGINE SDK is used to create the project. We have specifically chosen the SDK at the time of the creation of the project. Navigate to `<user account>/UNIGINE SDK Browser/sdks` path and explore the SDK you have selected. In my case it is `<user account>/UNIGINE SDK Browser/sdks/community_windows_2.11.0.2`. At this path, you will find the data folder, and when you navigate to this folder, you will see the core folder. You will also locate the file describe above if you browse through the relative path.

The project creation process picks up the core folder and compresses it using the archiver tool from UNIGINE SDK. Archiver is a tool for data archiving, which handles UNG files. It encrypts the given data to avoid unauthorized data access. UNG files are transparent to the engine, and their content viewed as not packed. You may compress, archive, and store the ung file in the data folder, the editor and engine can still browse it as an open file structure. The core.ung file contains all the core elements, and you may use them in your projects. You may browse this file in the asset browser of the editor.

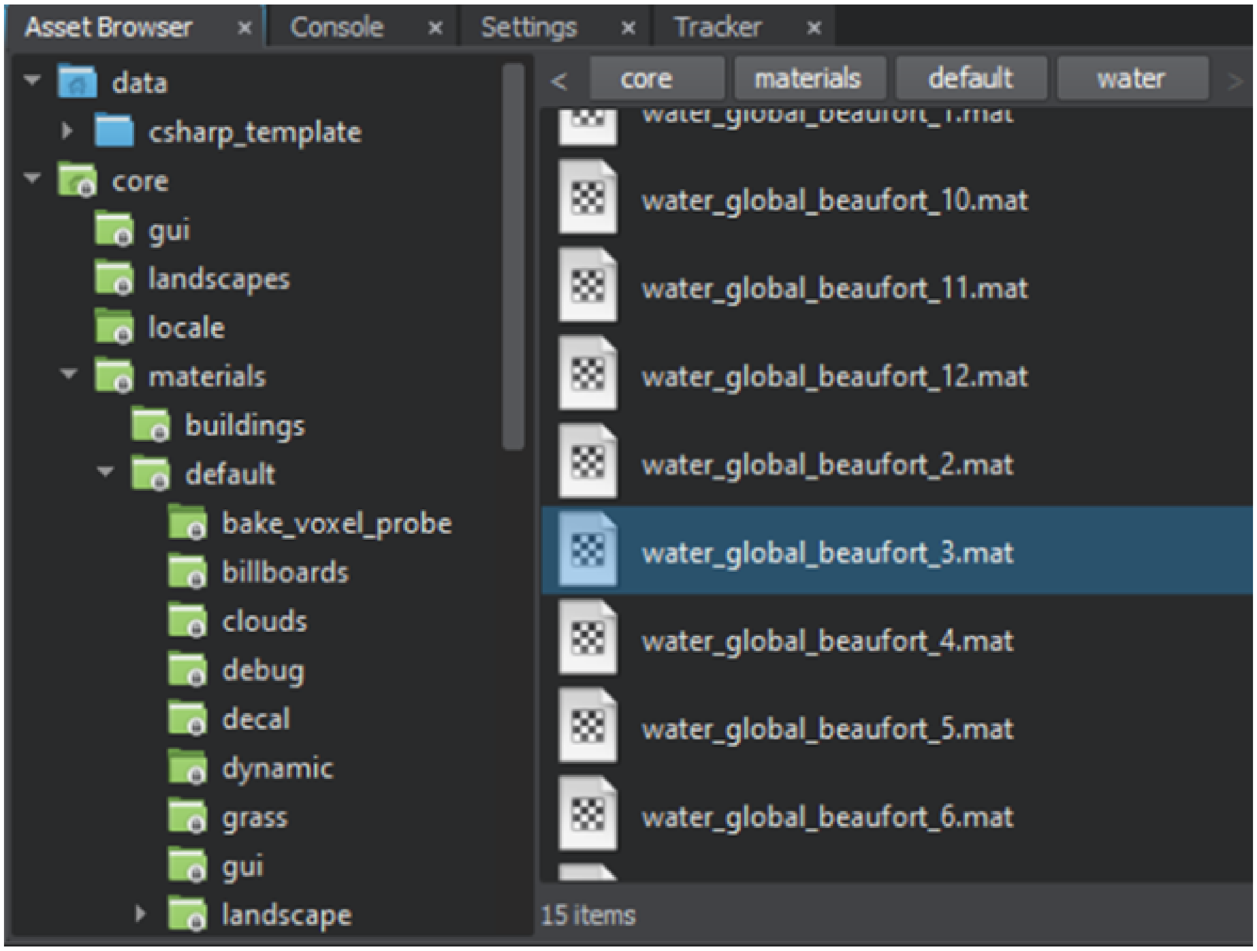

Asset Browser
Console
Settings
Tracker
data
csharp_template
core
gui
landscapes
locale
materials
buildings
default
bake_voxel_probe
billboards
clouds
debug
decal
dynamic
grass
gui
landscape
core
materials
default
water
water_global_beaufort_10.mat
water_global_beaufort_11.mat
water_global_beaufort_12.mat
water_global_beaufort_2.mat
water_global_beaufort_3.mat
water_global_beaufort_4.mat
water_global_beaufort_5.mat
water_global_beaufort_6.mat
15 items

Figure 3.16: The asset browser window. Browsing of materials from core.ung file that is transparent to editor and engine.

Let us change some content in our project. Let us understand the workflow by adding an image (2-dimensional texture) to the content. I have an image file flower_image_1.png. The file's physical path can be anywhere in my storage. I will import this to the project we have created.

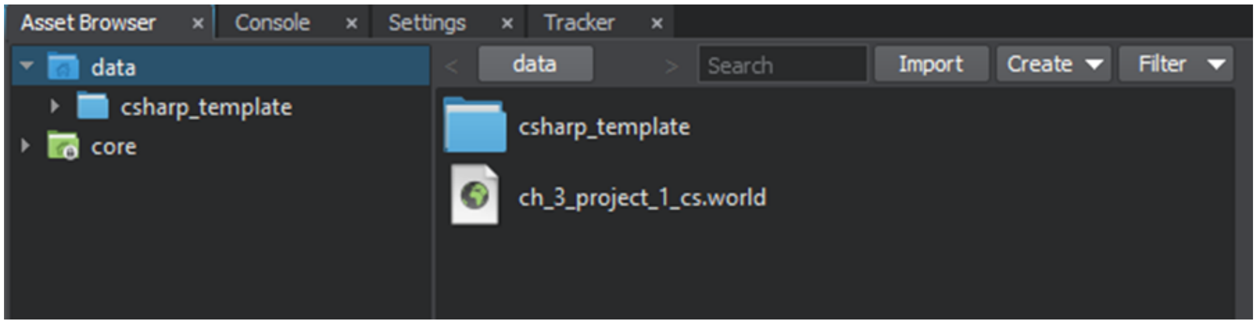
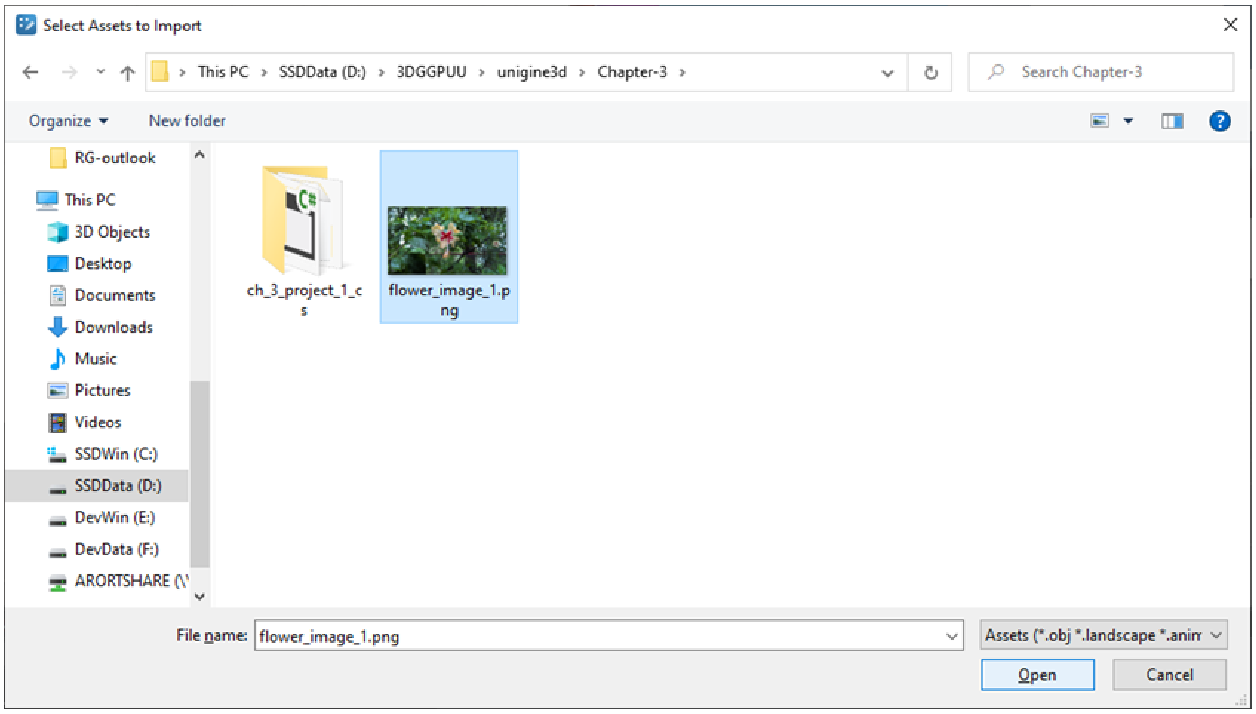


Figure 3.17: The asset browser window. The data root folder is selected. The import action imports the assets into the run-time.

Click on the import button to open the import asset dialog. Browse the path where the image is stored and select it to import. Look at the file filters in the open file dialog. Many file types present there that are considered assets. We are interested in the image at this moment. Once you select the required file, depending on the asset type, import asset dialog will pop up.


Select Assets to Import

This PC > SSDData (D:) > 3DGGPUU > unigine3d > Chapter-3 >

Search Chapter-3

Organize ▾ New folder

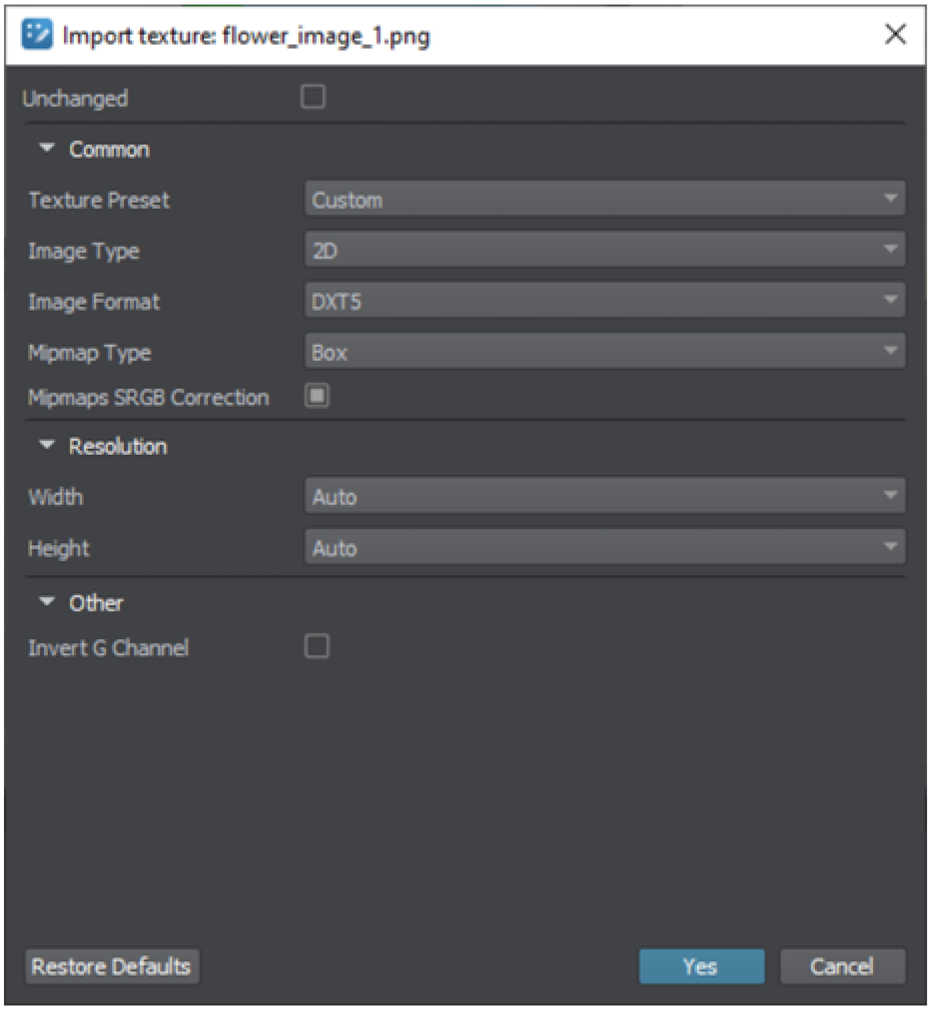
RG-outlook
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
SSDWin (C:)
SSDData (D:)
DevWin (E:)
DevData (F:)
ARORTSHARE (\\

ch_3_project_1_cs
flower_image_1.png

File name: flower_image_1.png
Assets (*.obj *.landscape *.anim

Open Cancel

Figure 3.18: The asset import dialog window. The file type filters contain lots of asset types to import to the UNIGINE asset workflow.


Import texture: flower_image_1.png

Unchanged	☐
Common	
Texture Preset	Custom
Image Type	2D
Image Format	DXT5
Mipmap Type	Box
Mipmaps SRGB Correction	■
Resolution	
Width	Auto
Height	Auto
Other	
Invert G Channel	☐

Figure 3.19: The asset we are importing is an image. UNIGINE editor identified the type, and the asset import dialog box will pop up. Here we need to specify the conversion parameters for run-time.

We will keep the default values to the import parameter and click on yes to import the image. The asset workflow will import the image with the necessary conversions required for the run-time. The original raw image will also get added to the selected folder. The asset workflow will make few changes to the file systems by this point. Let us check it.

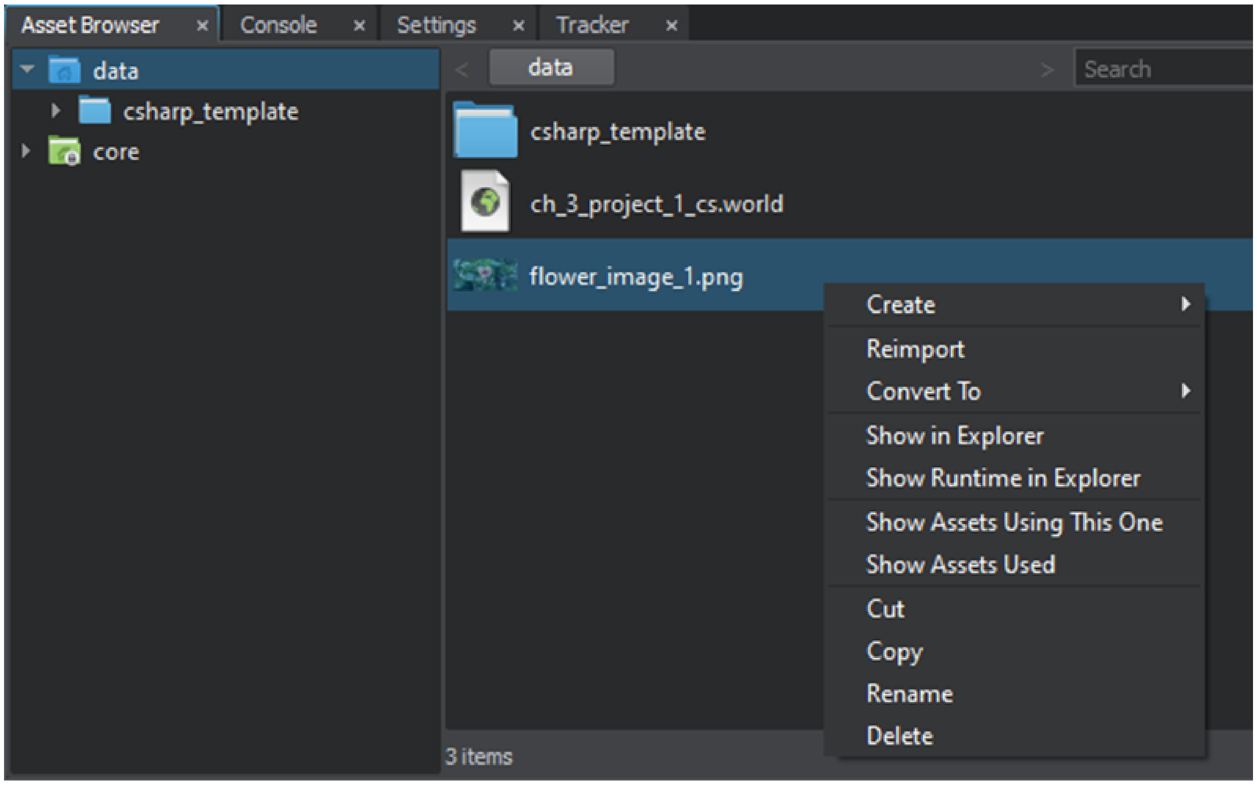


Figure 3.20: The asset browser with the newly imported asset.

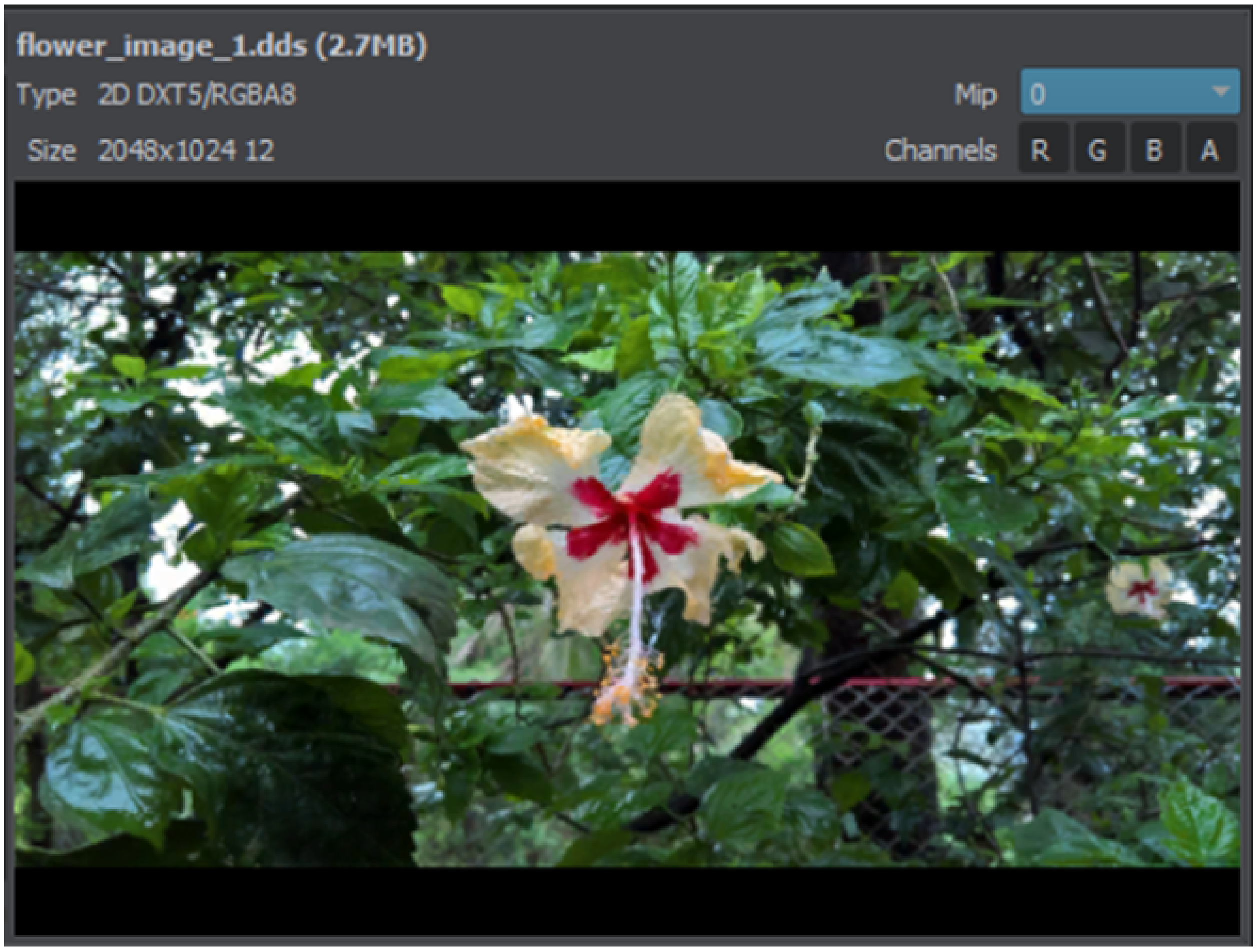


Figure 3.21: The asset browser, preview panel showing the runtime texture.

Figure 3.20 shows the original imported texture entry to the data folder. Click to select the file and right-click it for context menu and select Show in Explorer option. The explorer will open at the project's physical path. Navigate to the data folder, and you will find your flower_image_1.png. The asset workflow has copied the original image file from the source location to the data folder as we have selected it.
Open the context menu again and select "Show runtime in the explorer" menu. The explorer will open <project path>/data/.runtimes/<some two digits>. It is becoming interesting now. What has happened actually?

1. We imported an image using asset workflow.
2. We have chosen some values for import texture dialog parameters.
3. The editor has copied the original file from its source location to the data folder location we have selected.
4. The asset workflow has created a meta file against imported file flower_image_1.png.meta.
5. The asset workflow created a guid for the original file and created a link for it in .meta file.
6. The asset workflow has converted the original file to the .dds file as per selected import parameters
7. The workflow creates the .runtime folder with the first two digits of the guid that is a link to the

.dds file.
8. The converted .dds file is given that guid as a name and saved at data/.runtimes/<first two digits of guid>/<guid>.dds
9. The <guid>.dds.meta file contains the link to the original asset.
10. The flower_image_1.png.meta. files save all of this information about the original and run time asset.

Listing 3.2: The .meta file contents. The metafile contains the guid, links for original, and run-time created files. It also preserves the import parameters.

```xml
<?xml version="1.0" encoding="utf-8"?>
<asset version="2.11.0.0">
        <guid>2057d4529b7f67890188cb54ac98bea37a1093a6</guid>
        <type>texture</type>
        <hash>2726d38b</hash>
        <runtimes>
                <runtime id="2057d4529b7f67890188cb54ac98bea37a1093a6" name="flower_image_1.png" l\
ink="0"/>
                <runtime id="755096cd07a149c8178d6287630dbe3a5196b851" name="flower_image_1.dds" l\
ink="1"/>
        </runtimes>
        <parameters>
                <parameter type="string" name="mipmap_type">box</parameter>
                <parameter type="string" name="image_format">DXT5</parameter>
                <parameter type="bool" name="invert_g">0</parameter>
                <parameter type="string" name="image_type">2d</parameter>
                <parameter type="string" name="width">auto</parameter>
                <parameter type="string" name="height">auto</parameter>
                <parameter type="bool" name="srgb_correction">1</parameter>
                <parameter type="bool" name="unchanged">0</parameter>
                <parameter type="string" name="preset_name">custom</parameter>
        </parameters>
</asset>
```

Check figure 3.21. The preview panel shows the run-time file created for the image imported prior. On selecting the asset, the parameter window shows all the asset import parameters we have chosen at the time of asset import. We may change the values of import parameters and reimport the image as many times as we need. Whatever is the size of the imported texture, when run-time generates texture assets dynamically, the dimensions resize to powers of two and even creates mipmaps. The preview panel gives options to see per channel data of run-time assets. It is the reason to generate run-time assets. As original textures will not come with mipmaps, but for performance and to achieve rendering quality, the engine creates mipmaps for all types of imported images. At this point, the imported textures are ready to map onto 3D geometries.

NOTE: In computer graphics, mipmaps are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the same picture. The height and width of each image or level, in the mipmap, is a power of two smaller than the previous level.

There are no changes or additions visible in the file, guid.db. Please close the editor and reopen it; the asset workflow recreates the file, guid.db. We will learn more about this file during build creation.

Listing 3.3: The guid.db file with entries linked to the added image and its meta file using guid.

```
"3976c9762842325d92f190180474d497c5d0e90a": "flower_image_1.png.meta",
"2057d4529b7f67890188cb54ac98bea37a1093a6": "flower_image_1.png",
```

Table 3.2 – The explanation of parameters and their values of import texture dialog

Unchanged	Defines whether the texture is to be used "as is" or runtime will be created for it following the specified import options.
Texture Preset	This preset determines the kind of the texture generation pattern, and it defines compression algorithms and color channels to use.
Image Type	Type of texture generated, 2D, 3D, or Cube map.
Image Format	It defines the image pixel format: bit depth and channels used. Compressed formats are supported.
MipMap Type	It defines the filtering used for the mipmap generation. Box, Point, or Combined.
sRGB Correction	It enables sRGB gamma correction for mipmaps.
Width	Texture Width in the power of 2.
Height	Texture Width in the power of 2.
Invert G Channel	It enables the inversion of the green channel of the imported image. Normal maps creation differs depending on the game engine or 3D software package used to prepare it. This option simplifies the conversion of normal maps from different source platforms.

We learn about texture imports using a 2-dimensional image. There are some images with specific texture features viz. albedo, normal, cube textures, and so many other types. We will check other texture types in the following chapters and sections once we come across the need for one. We will also look for different types of native and non- native assets, importing, and working with them in the coming chapters.

NOTE: The best feature of the Asset System is real-time tracking of changes. The changes made to the asset through the Windows file system will reflect itself inside the UNIGINE editor. It means while the editor is open, Browse to the file and open it in any image editor. Make some changes and save. The same changes will reflect automatically to the image in the UNIGINE editor. Bravo!

The asset workflow is a file system that relies on virtual connections using GUID. It creates .meta files and run-time assets to suit the native architecture. The whole system performs real-time tracking and keeps the content up to date. The entire workflow will revolve around the asset browser, and the regular file explorer should no longer be necessary when working on a UNIGINE project.

We looked deep in the file system to understand the guid, .meta files, and we learn about how UNIGINE manages them. Sometimes, at an advanced level to solve some problems, this knowledge is necessary and essential.

Run via Code

Finally, we are coming closer to finish our chapter with the first project. We have created the project, we edited it, and we have learned a little about the asset workflow system.

NOTE: The coding is not required to run the UNIGINE project until you want to put interactivity and behaviors into the content.

All the work described in this book is available through Github. Please download or clone the GitHub repository that contains all the projects.

The path to the book's chapter-wise projects and code is https://github.com/rohitgonsalves/unigine3d[1]

NOTE: After downloading the GitHub projects. Add them to the editor. For example, download the ch_3_project_1_cs project. Use the SDK browser and select Add Installed. Locate the directory and add the project. It will add the project, but the project will not contain all the run-time files as they are not committed to the GitHub. That's why the opening of the editor will give you an error. Please select the Configure Project option from other Actions menu. Select the Update Configuration to recreate the run-time and project-specific files from the existing data and source folder.

Run code via C#

Programmers, let us run our code using C# and C++ based UNIGINE projects. We have already created the C# (.NET Core) UNIGINE project. Just select the project tile and chose the option open code IDE. We have installed the Visual Studio IDE, and the c# projects will default to open in visual studio. The visual studio will launch with the C# project. Where is this c# project located?
Open the project in explorer and check the files at the location. You will find <project_name>.csproj file. It is the project file for the c# project.

[1]https://github.com/rohitgonsalves/unigine3d

Please download or clone the Github repository. From chapter 3, load the project ch_3_project_1_cs. The project has the .csproj file as we have created a project with C# bindings. For the c++ language support, we require to create the projects with c++ API binding.

NOTE: The book will follow the project naming convention as ch_<no>project<no><Langauange binding>.
For example, chapter 3 c# project that is the first project will take a name like ch_3_project_1_cs, and the next c++ project will take a name like ch_3_project_2_cpp.

The visual studio will open with the c# project from our UNIGINE project. Check figure 3.22. The C# project will load the default build configurations and platforms. The project supports debug and release configuration for development, and it has Any CPU build platform. It is a necessary and sufficient start to the code, and compile to a specific platform like 32-bit or 64-bit processors installed on your machine. You may change the build platform or add more configurations as your programming environment demands. But the default one is good for us.

NOTE: You may find more information about configurations and platforms at Microsoft documentation of Visual Studio IDE.
Build Configuration: https://docs.microsoft.com/en-us/visualstudio/ide/understanding-build-configurations?view=vs-2019[2]
Build Platform: https://docs.microsoft.com/en-us/visualstudio/ide/understanding-build-platforms?view=vs-2019[3]

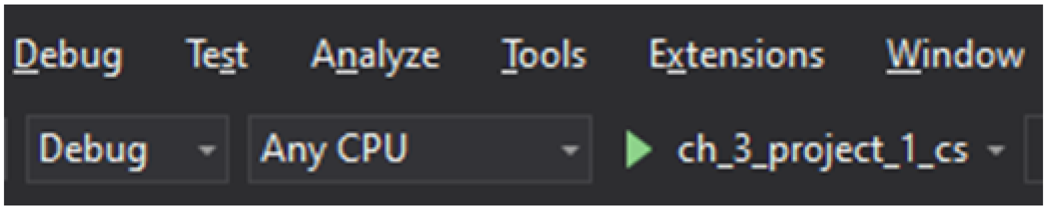
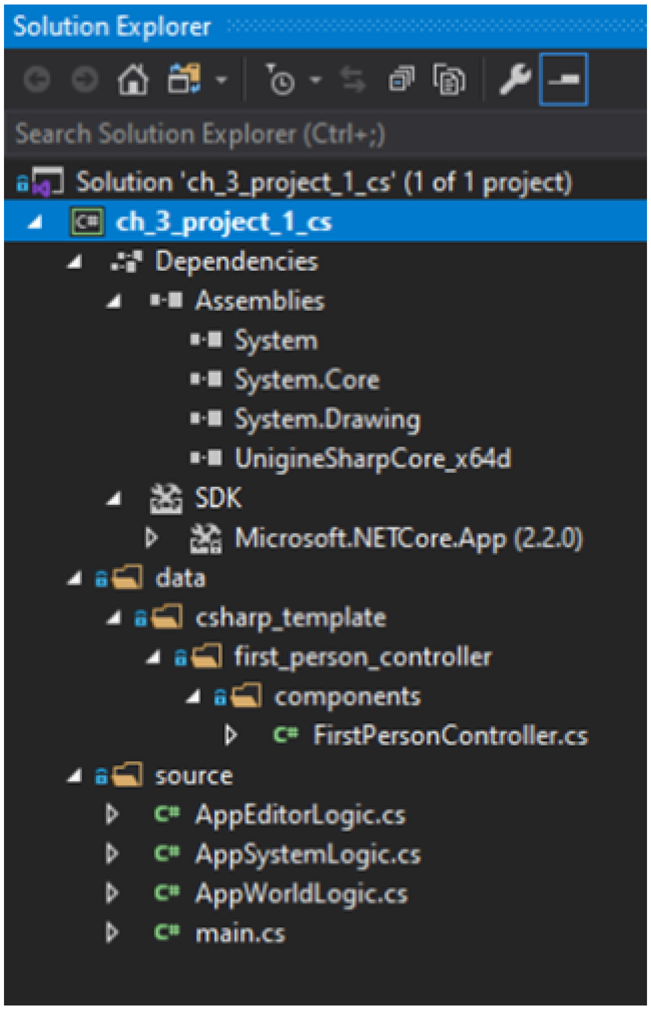


Figure 3.22: The c# project builds configurations and builds platforms

[2]https://docs.microsoft.com/en-us/visualstudio/ide/understanding-build-configurations?view=vs-2019
[3]https://docs.microsoft.com/en-us/visualstudio/ide/understanding-build-platforms?view=vs-2019


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'ch_3_project_1_cs' (1 of 1 project)
- ch_3_project_1_cs
  - Dependencies
    - Assemblies
      - System
      - System.Core
      - System.Drawing
      - UnigineSharpCore_x64d
    - SDK
      - Microsoft.NETCore.App (2.2.0)
  - data
    - csharp_template
      - first_person_controller
        - components
          - FirstPersonController.cs
  - source
    - AppEditorLogic.cs
    - AppSystemLogic.cs
    - AppWorldLogic.cs
    - main.cs

Figure 3.23: The c# project solution explorer. It adds required assemblies and dependencies. There are components and sources for further development already installed.

In this chapter, we will only compile the code and run it to see the UNIGINE project in action through coding. We will do an in-detail discussion in further chapters and sections. Please change the configuration to release, and compile the project.

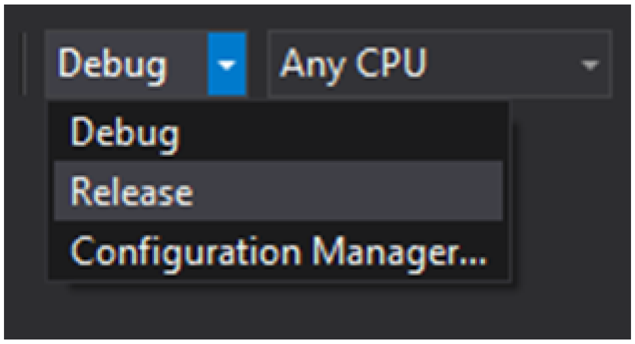


Figure 3.24: Changing the c# project configuration to release.

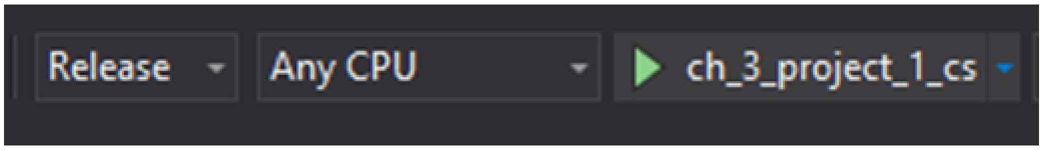
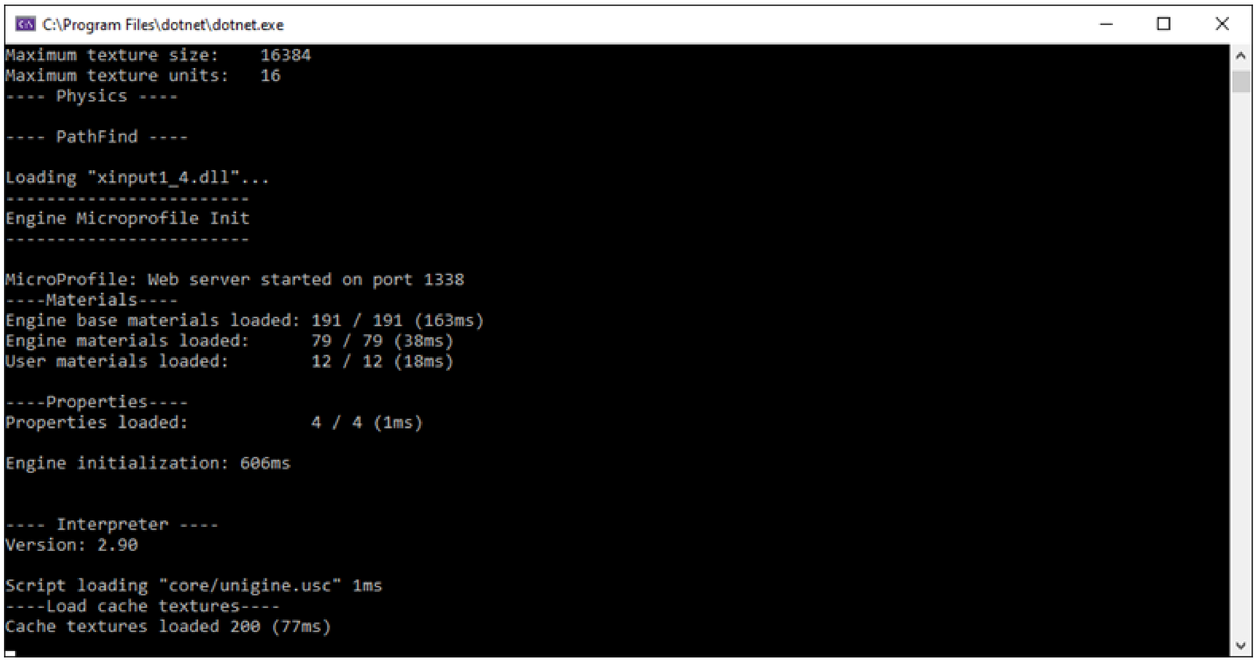


Figure 3.25: Run the project using the Run button from the toolbar. Or press Ctrl + F5 to run without debugging.

Press F7 or Choose Build menu and select Build Solution or Rebuild Solution. Or you may right-click on the Solution from solution explorer and choose Build or Rebuild. Once the compiled and built project is ready, click on the Run button or press Ctrl + F5 to run the build without debugging.


C:\Program Files\dotnet\dotnet.exe

```
Maximum texture size:    16384
Maximum texture units:   16
---- Physics ----

---- PathFind ----

Loading "xinput1_4.dll"...
-----------------------
Engine Microprofile Init
-----------------------

MicroProfile: Web server started on port 1338
----Materials----
Engine base materials loaded: 191 / 191 (163ms)
Engine materials loaded:      79 / 79 (38ms)
User materials loaded:        12 / 12 (18ms)

----Properties----
Properties loaded:            4 / 4 (1ms)

Engine initialization: 606ms


---- Interpreter ----
Version: 2.90

Script loading "core/unigine.usc" 1ms
----Load cache textures----
Cache textures loaded 200 (77ms)
```

Figure 3.26 The console application executed by dotnet.

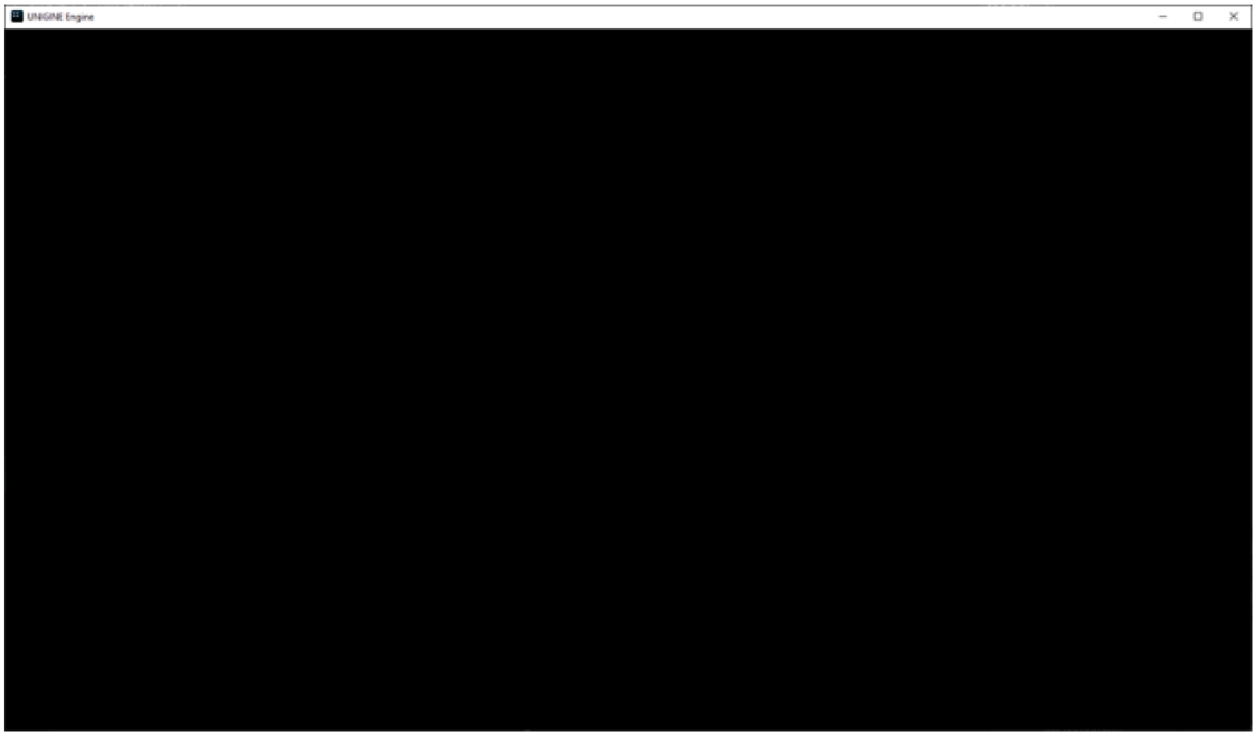


Figure 3.27 The console application initiates the UNIGINE engine with the black screen.

As shown in figure 3.27, we see the black screen and not the world we have created with our first project. To understand the solution to this problem, take a look at one more project file. Open the project location and open the launch_release.bat file in the editor to check. Double click it and see the result as in figure 3.28.

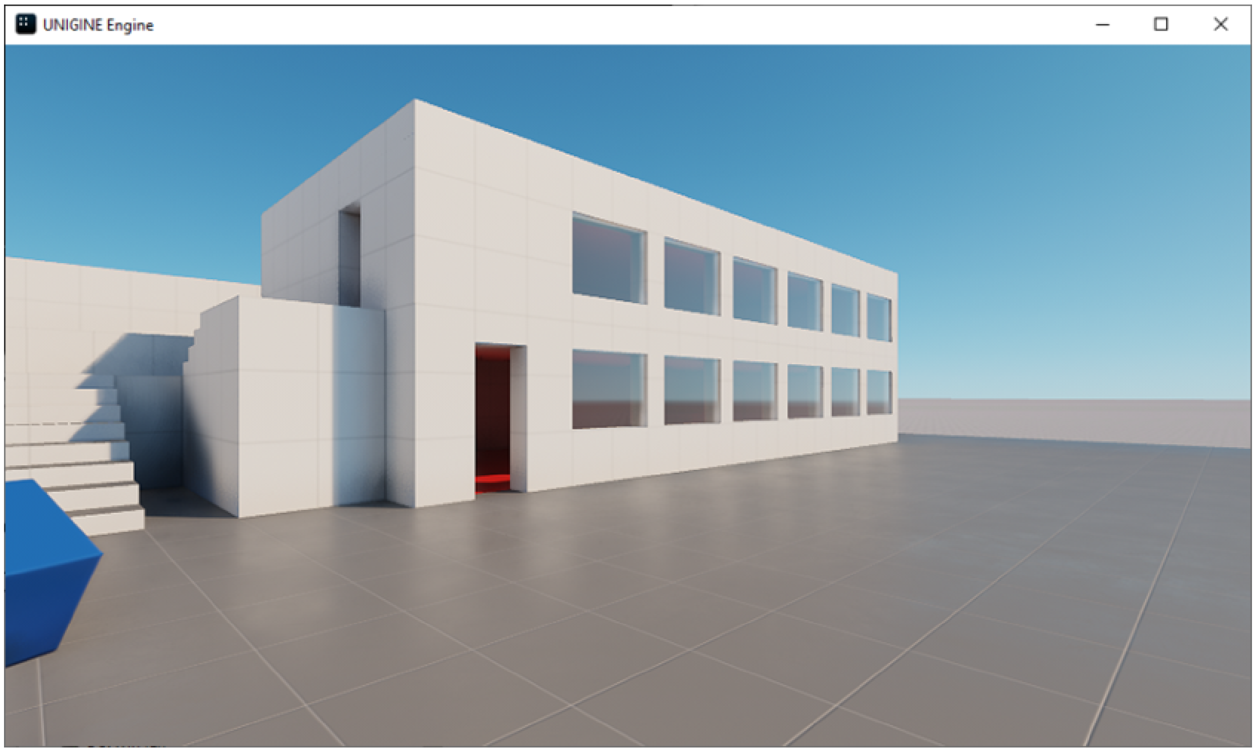


Figure 3.28 The UNIGINE Engine with loaded world content.

What is happening here? First of all, when we compile and build the solution, the project creates an assembly with type console application in the bin folder. For our first project, the program assembly is ch_3_project_1_cs_x64.dll for release configuration and ch_3_project_1_cs_x64d.dll for debug configuration.

NOTE: UNIGINE only supports the x64 or 64bit processor architecture for 3D game and Graphics development.

The .bat file contains the command to run dotnet with ch_3_project_1_cs_x64.dll assembly located in the bin folder. Remember, we discussed start-up options in chapter 2. This command contains many start-up options like video_mode that sets the window width to 960 and height to 540. The .bat file command also has an option for loading a world, world_load ch_3_project_1_cs. The engine looks for this world in the data folder. In command, we only give the name of the world. But engine searches for ch_3_project_1_cs.world file in the data folder. It is how we built the program and executed it. Now our program from Visual studio is also ready. But we see the black screen as there is no world loaded. Let us load the world, and we run to see the effect. Open the AppSystemLogic.cs from the source. Add just one line to load the world on init.

Listing 3.5: The init method is modified to load the world using the LoadWorld method of the World Class.

```
public override bool Init()
{
// Write here code to be called on engine initialization.
        World.LoadWorld("ch_3_project_1_cs");

        return true;
}
```

Let us build again and run. You will see the UNIGINE engine started from the code, and it has loaded the content we had in our first project. Bravo! Save your project. The IDE will ask you to save the solution. If you wish, do it. No harm. Let us discuss the starting point for our Program.

Listing 3.6: The main.cs file from the project.

```
using Unigine;

namespace UnigineApp
{
        class UnigineApp
        {
                [STAThread]
                static void Main(string[] args)
                {
                        Engine.Init(args);

                        AppSystemLogic systemLogic = new AppSystemLogic();
                        AppWorldLogic worldLogic = new AppWorldLogic();
                        AppEditorLogic editorLogic = new AppEditorLogic();

                        Engine.Main(systemLogic, worldLogic, editorLogic);

                        Engine.Shutdown();
                }
        }
}
```

The main.cs file contains the startup code for .NET assembly. The program starts the execution of the program assembly from the main routine. We see that the file uses the namespace Unigine.
The routine defines Engine's Init, Main, and Shutdown method. Any graphics or game engine program uses a rendering loop. The rendering loop is a loop that runs indefinitely to render frames to the output. UNIGINE also has a built-in rendering loop. Let us understand a little about the rendering loop.

1. Initialization: During this stage, the required resources are prepared and initialized. As soon as these resources are ready for use, the engine enters the main loop. It is window creation, DirectX or OpenGL device creation, and many other startup functions.
2. Main loop: The UNIGINE engines repeat the three-stage rendering process in brief. After UNIGINE enters the main-loop, all three stages are repeated one by one in a cycle:
• Update stage contains the logic of your application that executes every frame
• Rendering stage includes all the rendering-related operations, physics simulation calculations, and pathfinding
• Swap stage contains all synchronization operations performed to switch between the buffers
• This cycle repeats every frame until the application runs

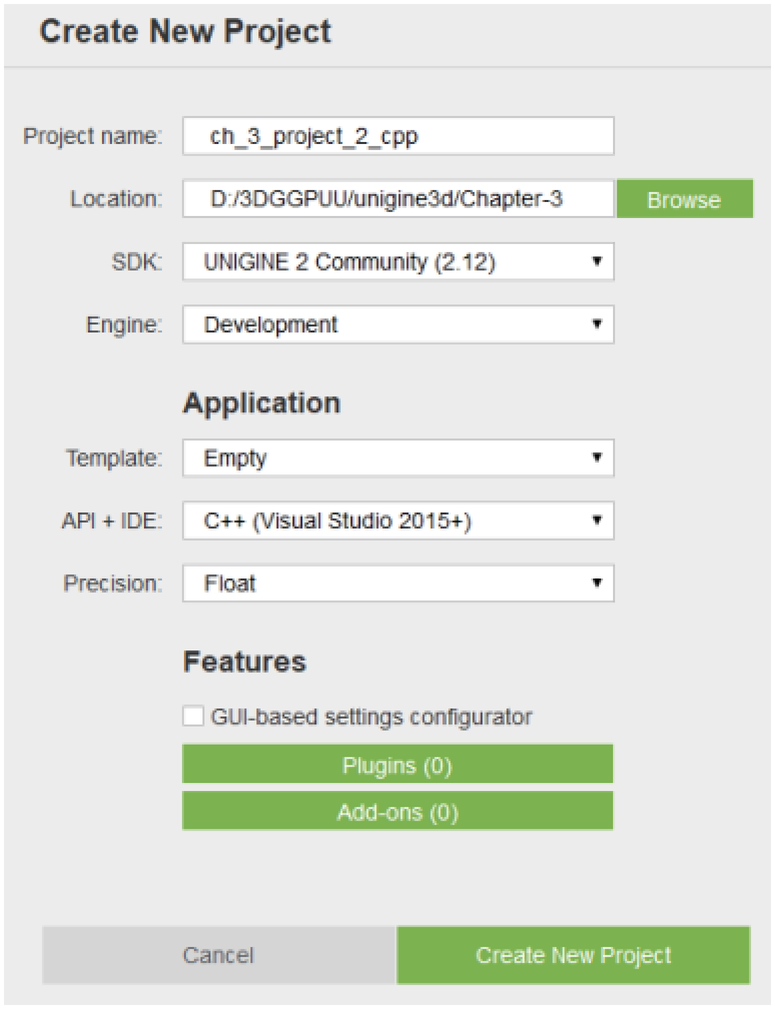
3. Shutdown: When UNIGINE stops execution of the application, it performs operations related to the application shutdown and resource cleanup.

From our listing 3.6, it is clear that we are initializing an engine. Then enter the main loop, and it repeats itself until the end of the application. Once we close the window, the main loop leaves the repeating cycle, and the shutdown routine takes over to end an application. The Main() routine takes three parameters, object of type AppSystemLogic, AppWorldLogic, and AppEditorLogic. You may just delete them, build the solution, and run. It is a simple code to initiate the engine and run it with a repeating rendering loop.

Engine.Main(); routine will also work, and the engine loads but with a black screen. It is because we didn't supply the AppSystemLogic to the engine's run-time that loads the world on initialization. Change the Main routine to have the only object of AppSystemLogic. Engine.Main(systemlogic); build and run. The engine will load, and subsequently, it will load the world file. Let us see the same action in C++.

Run code via C++

It is time for C++ programmers like me to leap with UNIGINE projects. The best part about the UNIGINE engine, it is developed in the C++ language. The UNIGINE developers have prepared the managed wrappers and used other third-party libraries to give support of .NET and the C# language. We had C# (.NET Core) based project to do C# programming. To program in c++, we require to create projects with C++ language bindings.


Create New Project

Project name: ch_3_project_2_cpp

Location: D:/3DGGPUU/unigine3d/Chapter-3

SDK: UNIGINE 2 Community (2.12)

Engine: Development

Application

Template: Empty

API + IDE: C++ (Visual Studio 2015+)

Precision: Float

Features

GUI-based settings configurator

Figure 3.29 The UNIGINE project creation windows. For C++ project binding, we select C++ (Visual Studio 2015+) API and IDE.

There are two options for C++. C++ (Visual Studio 2015+) and C++ (CMake). We select to use Visual Studio to make the UNIGINE browser create a Visual studio compatible project file. After project creation, Open the editor and let the engine configure run time and other files. There are a few differences. Choose to open the folder of the project. You will see that you have include and lib folders now. Few folders are different from the C# project. The include folder contains the header files to use UNIGINE APIs. There are debug and release configuration lib files present in the lib folder that you may link to compile and build your C++ based projects. Even the world contents are completely different when you create a project using C++ API. Let us chose the open code IDE option to load the C++ project now. Please note that the C++ project resides in the source folder and not just outside like .csproj.

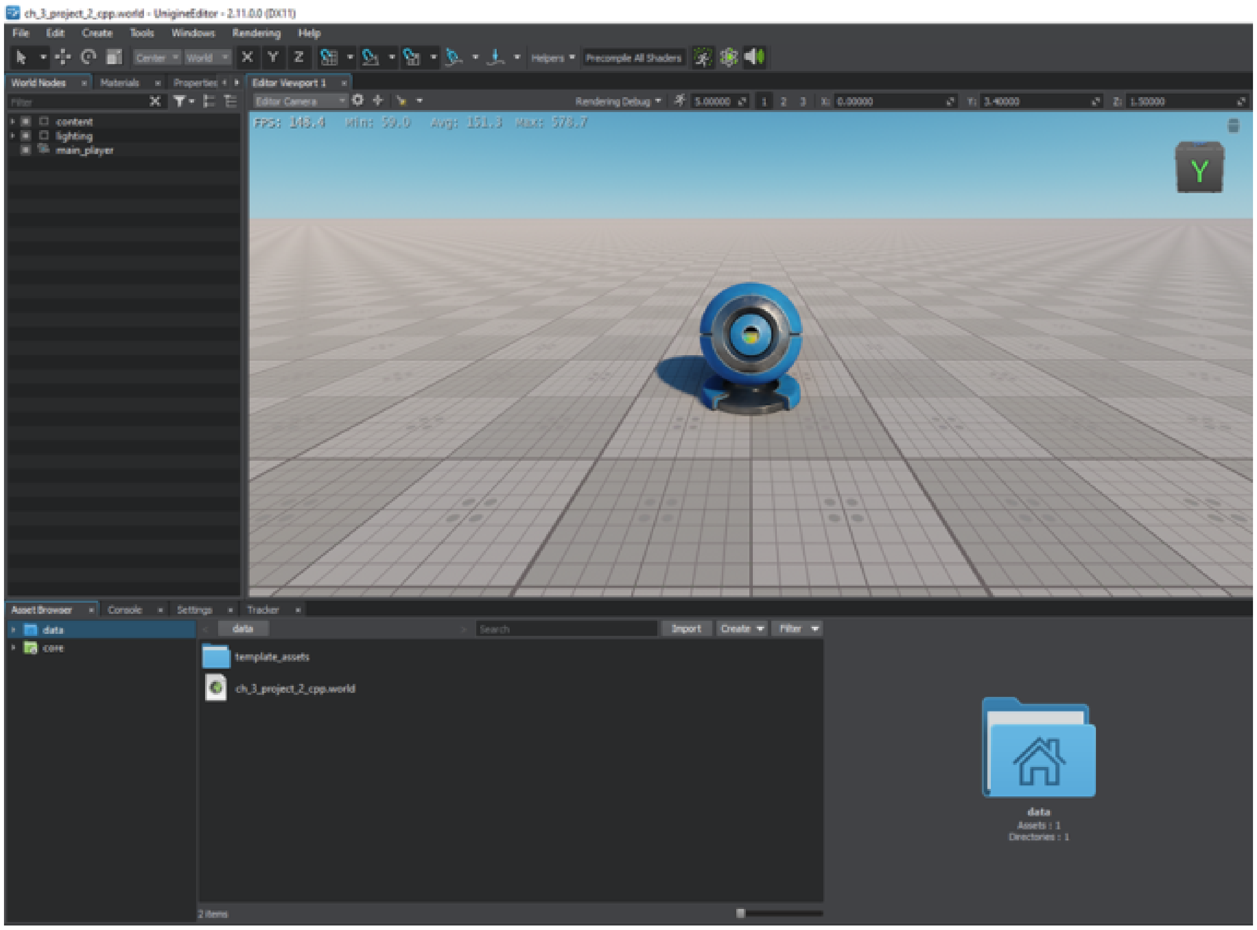
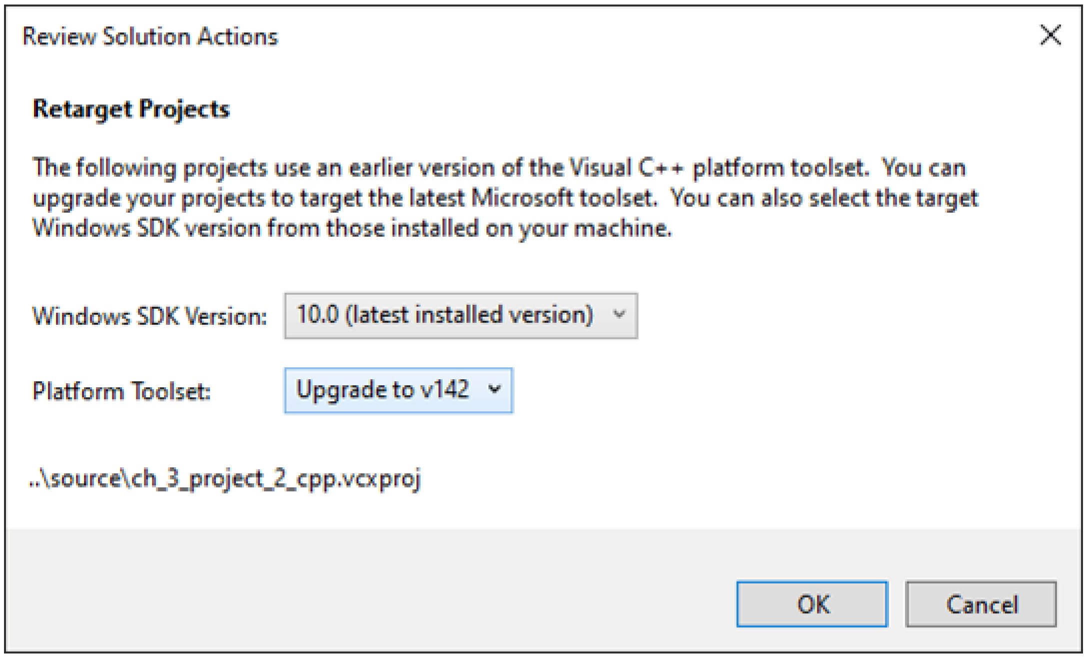


Figure 3.30 The UNIGINE C++ project with default world and assets.


Review Solution Actions

Retarget Projects

The following projects use an earlier version of the Visual C++ platform toolset. You can upgrade your projects to target the latest Microsoft toolset. You can also select the target Windows SDK version from those installed on your machine.

Windows SDK Version: 10.0 (latest installed version)

Platform Toolset: Upgrade to v142

..\source\ch_3_project_2_cpp.vcxproj

Figure 3.31 The Visual Studio project upgrade dialog with retargeting of Windows SDK.

The project opens up with the visual studio IDE. The IDE will probably popup the upgrade dialog. We are using the latest Visual Studio IDE, and projects created by UNIGINE are still supporting the older version. It is safe to upgrade the projects, to the newest Windows SDK and v142 platform toolchain.

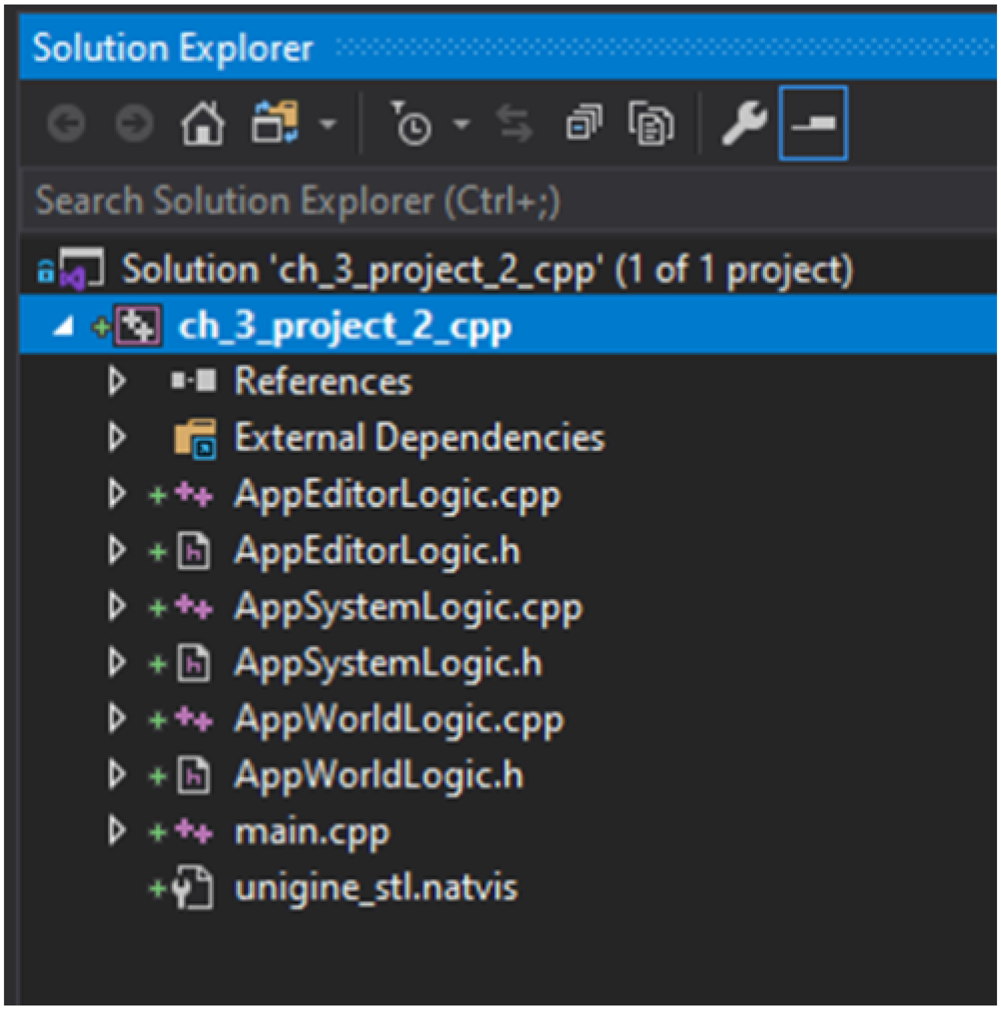


Figure 3.32 The C++ project solution explorer with the headers and CPP files.

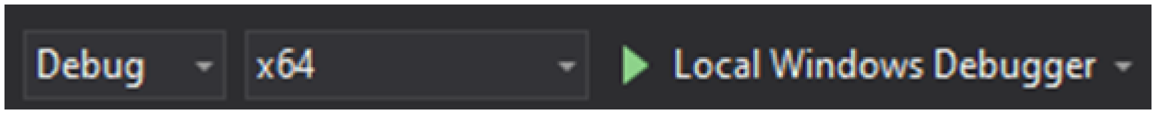
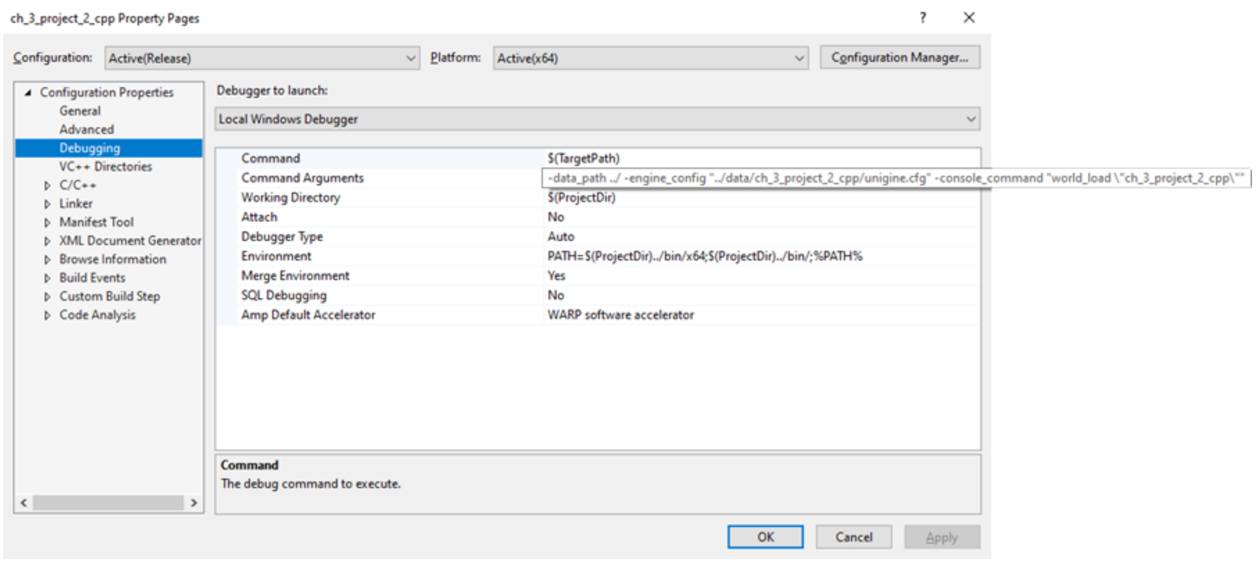


Figure 3.33 The C++ project debug and release configuration with the x64 platform.

As in the C# project, the C++ project has Debug and release configuration, and x64 build platform. Change the build configuration to release option, and compile the code using the F7 key or right-click on the project and chose the Rebuild option. As the compilation finishes, you have ch_3_project_2_cpp_x64.exe file ready in the bin folder. Now open this project launch_release.bat file and observe that the command directly calls the executable file here. Execute the code using the run button. Surprise! The engine loads the world automatically without specifying it anywhere. In the C# project, we manually added world loading functionality. How is this happening? There is no magic here; the UNIGINE developers created the C++ project with some extra parameters and support. Figure 3.34 will answer all questions.



Configuration: Active(Release) Platform: Active(x64)

Debugger to launch:
Local Windows Debugger

Command	$(TargetPath)
Command Arguments	-data_path ../ -engine_config "../data/ch_3_project_2_cpp/unigine.cfg" -console_command "world_load \"ch_3_project_2_cpp\""
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	PATH=$(ProjectDir)../bin/x64;$(ProjectDir)../bin/;%PATH%
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

Command
The debug command to execute.

Figure 3.34 The C++ project properties windows showing debugging parameters.

The UNIGINE developers have added the command line arguments to the debugging of the project. Right-click the project listing and chose property to see the project property window. The command arguments have a world_load command with the name of the world. That is why when we run C++ projects via code, it loads the project world. Even the C++ project has files called, AppSystemLogic, AppWorldLogic, and AppEditorLogic header and CPP files.

Listing 3.7: The main.cpp file from the c++ project

```
#include <UnigineEngine.h>

#include "AppEditorLogic.h"
#include "AppSystemLogic.h"
#include "AppWorldLogic.h"

#ifdef _WIN32
int wmain(int argc, wchar_t *argv[])
#else
int main(int argc, char *argv[])
#endif
{
        // UnigineLogic
        AppSystemLogic system_logic;
        AppWorldLogic world_logic;
        AppEditorLogic editor_logic;
        // init engine
        Unigine::EnginePtr engine(UNIGINE_VERSION, argc, argv);

        // enter main loop
        engine->main(&system_logic, &world_logic, &editor_logic);

        return 0;
}
```

The listing 3.7 shows the same kind of code like c#, we initiated the engine and started the main loop. The UnigineEngine.h header included referring to the engine object. Once the main loop ends, the application ends with all correct resource cleanup. Lets us see a little bit about application logic systems.

Logic Systems in UNIGINE

We may program out UNIGINE projects using C# and C++. We can write an application or game logic using C# and C++. UNIGINE architecture adds special interface classes to support different life cycles of your game and world. The application logic has three components that have different life cycles.
1. **Application System logic**: It exists during the application life cycle. It means, once you provide the object of this application logic component to the engine, it will call the specific methods throughout the life cycle of the engine. You can inherit SystemLogic class and implement your logic in C++ or C#. Naturally, it is the main game container to shift between levels of game or load and unload different worlds as per game logic goes.
2. **Application World logic**: It takes effect only when the world is loaded. These logic system objects hold the game-specific code and logic. The application world logic objects will not see callbacks from the engine if system logic unloads the world. We may add many world logic objects to the engine runtime. Thus we can add application world logic when we load a specific level of the game and remove it on the level unload. It is the best way to keep the level-specific code separate.
3. **Application Editor logic**: It takes effect only when UNIGINE Editor is loaded. You can also inherit the EditorLogic class and implement your logic in C++ or C#. It is a specific behavior if you want to add some editor related functionality to the content. Very few specific and limited implementations will require this kind of application logic.

In the coming chapters, we will understand more about the application logic system.

Version Control

The Version Control Systems or VCS (Git, SVN, Perforce, Mercurial, Bazaar, CVS, TFS, and other) are used to manage changes to code and data and enable teams to coordinate their development efforts. To make tracking of changes efficient and reduce the amount of disk space required, you should know which files must be tracked by the VCS and that need to be ignored.

Ignore List: Any UNIGINE project contains files and folders that are not subject to version control. These can be files created by the compiler, *.obj, *.tlog, or an output folder used to store binary executables. Ignore the following files and folders from the project's root folder, as they have generated automatically on your disk:

1. .thumbnails
2. bin

3. obj
4. junk
5. launch_debug.bat
6. launch_editor.bat
7. launch_release.bat
8. data/configs/default.user

Files and Folders to Add to Version Control: The following files and folders should be subject to version control:
1. data folder with all its contents except ignoring data/configs/default.user
2. source folder contents except for the elements mentioned above
3. *.project file and *.csproj for C# projects)

NOTE: You will have to Update the configuration of the project and recreate the run time files from the downloaded project from the GitHub.

Summary

Quite a long chapter. It gave us insights into many things about UNIGINE projects. We have created one project in C# and C++ each. We executed it in an editor; we learn about project configuration update. We have seen the asset workflow and understood how it behaves? We will certainly move through the asset workflow equation to know why we need it? We have executed the project code using C# and C++. We have an overview of the Application logic system. Now moving further, we will look at building blocks of UNIGINE content, Nodes, and Players. We will do some little mathematics to add transformations to geometry. By the end of this chapter, you must be feeling good to know more about the UNIGINE and internals of it.

CHAPTER 4: Virtual Worlds: Nodes, Surfaces, and Players

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Coordinate System and 3D models and Export

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Importing 3D Models: Geometry, Animations, Textures, Lights, and materials

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Nodes and Surfaces: LOD's, Nodes, Converted Engine Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Players: Virtual Camera and properties, View Bitmasking

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Sample Project with Models and Players (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Rotate the node continuously, Start the animation after a pre-defined time. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 5: File System and Render Settings

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Unigine File System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Asynchronous Data Streaming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Render Settings: Environment, Camera Effects, Antialiasing, Textures, color correction, DOF.

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Sample Project with render settings (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Change the rendering settings at run time (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

CHAPTER 6: Paint the Surfaces: Textures and materials, Metalness Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Textures: Colors and transparencies, mipmaps

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Unigine materials and materials inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

PBR workflow and creating new material

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Material Mapping and material Inheritance (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Access material parameters and change them at run time. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

CHAPTER 7: Lightning and Reflections: Global Illumination, Lights, and Day of Time

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Types of lights

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Global Illumination

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Types of Reflections and settings

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Different types of Lights and reflection implementation (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Add Lights programmatically and program the light settings and change the time of day. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 8: Behaviors and Logic: Properties and Component Systems

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Unigine Properties: Files, structure and usage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Component system

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Add Components to existing nodes (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Program a component to jump if any other node comes into its vicinity. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 9: Cinematics using tracker

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Introduction to Unigine Tracker

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Cinematic using players and other tracks. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Program a tracker using UnigineScript. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 10: Exteriors: Terrain, Vegetation, Grass, roads, and Fields

Introduction Terrain Objects in Unigine

Vegetation Add-on and Grass Objects.

Road Tool and Fields.

Project: Exterior scene with the terrain. (GitHub)

Programming: Roll the sphere on terrain.(GitHub)

Summary

CHAPTER 11: Decorate your world: Water, Volumes, Clouds, and Sky

Water

Volumes

Clouds

Sky

Project: Scene with water and clouds. (GitHub)

Programming: Float an object over water with waves. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 12: Effects: Particles and Decals

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Particle System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Decals

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Decals on roads and smoke and fire particles. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Shooting decals Example. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 13: Making it responsive: Collisions, Intersections, Rigid Body Physics, and Physicals

World Collisions, and intersections. Properties and API

Physics: Rigid body physics, Shapes, and Joints.

Project: World with Physicals and Rigid Body Physics. (GitHub)

Programming: Physical callbacks and physics triggers. (GitHub)

Programming: Intersection and collision detections. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Object Picking. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 14: Pathfinding and obstacles

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Navigation objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Obstacles

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Writing a simple implementation for the crowd. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 15: Spatial Sound

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Sound Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: 3D Sound effect world (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Playing sound on collisions and playing ambient sound. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 16: User Interfaces: Unigine Widgets Graphics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming GUI: Containers, Widgets, and Localisation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

RC Files and Skin Layouts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Project: Small world with User interface and skins. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Create User interface and handle the callbacks (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 17: Creating your build

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Create Build

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 18: Virtual Reality and Mixed Reality

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Virtual Reality Support

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Mixed Reality Support

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Properties and component for Lens calibration for mixed reality (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 19: Debugging and Optimisations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Debugging using micro profiler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Techniques for Content Optimizations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Microprofiler programming (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 20: Extending Unigine Functionality using Plugins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Plugins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: Writing a simple Plugin (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

CHAPTER 21: Procedural Shading

UUSL

Vertex, Pixel, and Compute shaders

Programming: Write a custom material to change the scene to black and white. (GitHub)

Summary

CHAPTER 22: Boids Example

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Programming: BOIDS Program to simulate birds flock. (GitHub)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/unigine3d.

Acknowledgement