Producing a Level Editor For 3D Environments

CS3D661

Alex Sampson

15015483

Carl Jones

Contents

[Project Summary 3](#_Toc31989789)

[Project Aim 3](#_Toc31989790)

[Project Objectives 3](#_Toc31989791)

[Pre-Existing Level Editors 4](#_Toc31989792)

[Professional Tools 4](#_Toc31989793)

[In-Engine Tools 4](#_Toc31989794)

[Methods of Production 6](#_Toc31989795)

[OpenGL 6](#_Toc31989796)

[Unity Game Engine 6](#_Toc31989797)

[Loading 3D Models 6](#_Toc31989798)

[Selecting Objects 7](#_Toc31989799)

[Project Planning 8](#_Toc31989800)

[Project Development 9](#_Toc31989801)

[User Camera 9](#_Toc31989802)

[Object Interaction 9](#_Toc31989803)

[Object Creation 9](#_Toc31989804)

[Input 11](#_Toc31989805)

[Keyboard and Mouse Inputs 11](#_Toc31989806)

[User Interface 12](#_Toc31989807)

[Example Scene 13](#_Toc31989808)

[References 14](#_Toc31989809)

[Project Repository 14](#_Toc31989810)

# Project Summary

## Project Aim

The aim of this project is to create a functioning 3D level editor tool that includes all of the required features able to produce and export a 3D level to a file that is then usable to load the scene.

## Project Objectives

The first objective for the project is to create an environment where 3D models/files can be loaded into. This 3D model will be imported into a local file to ensure the project has no external dependencies, so that it can still function correctly even if the external files are moved or deleted. The Level Editor will have a UI element to display all the available models which will load them from the local files, with a preview version on display. To save on performance, the local version could be either a low-resolution version of the model or a single image file, both options being generated when the model is imported.

The next objective is to be able to access the model file previously imported and position it into the scene. Objects that are placed into the scene must then be able to be selected to enable transformations to them individually. The selection process could be through a UI element which lists every active object in the scene for the user to select from, or directly with the mouse cursor using rays cast in a 3D space from the camera, through the cursors position relative to the world.

The objects, when selected must be allowed to be interacted with, including having the object be able to be translated within the environment by having features to change the objects scale, position and rotation. A secondary objective with the interaction may be to allow the objects colours and textures to be changed, as well as to add any dynamic features such as a constantly moving object, for example a train passing by the scene continuously.

The final primary objective is to export the completed environment to a single file that can be used to load the environment as a standalone file, as opposed to it being confined to the editor.

A secondary objective for the project file management is to implement a save/load feature so that the in-progress levels could be saved inside the editor to be loaded later. This feature isn’t required for the functionality of the level editor, however the ability to save and load the project is a feature that will improve the overall use of the program.

# Pre-Existing Level Editors

## Professional Tools

A picture containing text

Description automatically generatedSource SDK

The Source SDK is a collection of tools used for authoring content for the Source engine. Valve Hammer Editor, included in the Source SDK, is a 3D level creation tool for use with the Source and GoldSource Engines for titles such as Half-Life, Counter Strike and Team Fortress (Valve Corporation, 2019). Hammer is a professional tool used by the official developers as well as publicly available through their Steam platform.

Figure - Hammer Editor

GtkRadiant

Radiant Level Design Tools is a similar tool to Hammer in that it is used to create 3D environments for games powered by the id Tech engines such as Quake, Wolfenstein and Doom (GtkRadiant Team, 2016). GtkRadiant currently exists as an open-source level editor but is also the base of the level editor currently used by Call of Duty developers Treyarch and Infinity Ward as their engine is a modified variant of the id engine.

The tool contains the basic functions of a level editor such as placing objects, lights and entities down to create environments, however it also includes a more advanced toolset for creating set pieces such as triggers, which when supplied with proper key-value pairs, can create a dynamic environment with moving doors, hazards and other interactable objects.

In summary, the two tools referenced above are very refined in their design for a very specific use case, and contain a wide variety of features which allows the users to create intricate, professional levels however this comes at the cost of having a rather high learning curve, as using these features in conjunction with each other can be a burden without the proper experience.

## In-Engine Tools

Halo Forge

Halo’s Forge is an in-engine game mode where players can create and translate objects within the Halo Universe to create variants of pre-existing levels and game-modes. These tools are a lot more rudimentary than the professional tools previously mentioned as they are designed with a focus on user experience and ease of use. Halo’s forge is a mode in which the player can add and remove props from existing levels and place their own down in any configuration within the limitations of the map, as well as the prop count limit that exists for performance and design reasons. The player can switch between a grounded player perspective and a free moving monitor, which allows for instant creation and testing without having to export to an external tool.

Using the editors listed above, both professional and in-engine, helps outline which features are key to this project and how to design the level editor for the specified purpose. To keep the development focused and within schedule, the project will lean more towards the in-engine simplicity to ensure the planned features can be developed fully, with any excess time being used for the more in-depth features more typically found in the professional tools.

# Methods of Production

As the brief outlined above does not require the use of a specific language or technology to be used, the most suitable method must be determined for this project.

## OpenGL

OpenGL is an API (Application Programming Interface) that could be used in this project to handle the graphics and rendering of the level editor’s 3D environments. OpenGL can be integrated into programs written in many different languages, with C and C++ generally being used as OpenGL itself being written in C (OpenGL, 2019).

If the project were to be written in C++ from scratch, OpenGL would be a great tool to use for the rendering, however, this may take a considerable amount of time to fully utilise, leaving less time to develop the key features.

## Unity Game Engine

Unity Game Engine is a multi-platform engine used for a wide variety of productions. Unity contains a vast array of tools that can be used to create games, animations, applications and more. Unity has a wide variety of libraries and components available from the base editor that can be used during the production of this project to easily implement features that otherwise would take longer independently due to the nature of their complexity.

## Loading 3D Models

The first stage of producing the editor is to determine a suitable method of loading the 3D models from a file to be used in the editor. One of the biggest problems to overcome with this aspect is that 3D objects can come in many different file formats such as .OBJ, .FBX and .3DS. All of these file formats will need to either be supported or be stated that they aren’t available in the project as they will all require code that will handle them properly to allow them to be imported into the editor.

The most basic method for handling file formats would be to only include a parser for a single file type, such as a Wavefront OBJ file, and require the user to convert the model outside of the editor which will save time on development for the project however will limit the user functionality as well as the user experience.

Using an existing library such as Assimp, the Open Asset Import Library, would allow any file format supported by the library to be imported into the editor with relative ease, as the library parses the formats and converts them in a uniform manner so that all file formats would appear the same once in the code to be used by the program (Assimp, 2018). The use of Assimp would improve the functionality of the project but has the potential to add an overhead on the time it takes to produce the importer as it would include learning a new library I haven’t previously used. This could also cause issues with the code if not implemented correctly due to inexperience.

## Selecting Objects

When the objects are present in the scene, the editor must have a method of selecting the objects to be transformed. An ideal way to select objects would be to use the mouse and click on the objects in the scene. To achieve this, the program can cast a ray from the position of camera, through the mouse in world position and then select an object if the ray collides with the object. Adding ray casting to the program may increase the development time depending on how it is implemented.

An easier method of selection is to have a UI element that contains a dropdown menu with all of the available objects in the scene to be selected. This method would be a very simple way of implementing selections, however it would hamper the user experience.

# Project Planning

The planning of the project is broken down into two sections, the primary and secondary objectives, with each containing a set of objectives needed. The primary objectives outline the key features needed to produce a simple level editor that hits all the objectives for the project, however, will be lacking in features to create anything more than a basic scene, with very little support for a good user experience. Each section of the primary objectives shouldn’t ideally take more than a week to produce, from conceptual work to implementation, with the exception of the final objective, exporting the finished level, as that will require extra time for research and design.

The Secondary objectives are designed to improve the overall quality of the Level Editor, as well as the user experience of the product by implementing features that are designed to ease the workflow within the tool.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Figure - Estimated Time Taken Per Objective

Figure - Time Management Gantt Chart

# Project Development

The first stage of project development was to set up the initial environment for the project, as referenced in Figure 3. As the decision was made previously to develop the tools using Unity, the process of initialising the project was simplified, allowing for more development time within the project.

## User Camera

The first feature to implement into the project is the camera so that the user is able to move around the environment. The camera requires movement along the X and Z axis relative to its local rotation. It is vital that the local rotation is considered to allow for forward/backward and left/right movement. The third axis of movement, along the y axis, won’t take into account any local rotation and instead will simply move the camera up and down according to world coordinates.

The camera will also need to be able to rotate to allow the user to look around the 3D environment from different angles. The rotation can be handled in different ways, based on the function and position required. The simplest method of rotation is to parse a mouse or joysticks input to rotate. For accuracy and convenience, preset positions will be included to align with each axis to allow for more accuracy when translating objects.

## Object Interaction

A screenshot of a computer

Description automatically generatedObject interaction can be regarded as the main functionality of the project, including object selection and manipulation. When starting the development of the object interaction, a simple test object was created to ensure features, when implemented, function correctly. The object interaction uses the mouse cursor for most of the selection and uses Unity’s own input functions to handle all of the mouse button pressing as well as mouse dragging.

Figure 4 - A simple test object

The user object contains a state for the transformations which will be controllable by the user. The user object will simply use a switch statement to check the state to determine the function to perform.

## Object Creation

The next area to focus development on is to add new objects to the scene, as the current testing environment only contains a rudimentary button to spawn a specific object, in particular the cube as seen in Figure 4. The creation system will require features for the user to be able to set which object will be added, as well as allowing parameters to be set, such as scale or texturing.

For the initial version of the implementation, the UI functionality will be disconnected from the main scene, with functions in place to spawn the object a set position in front of the camera. This method will be the easiest to implement, however with the simple implementation comes a lack of functionality and style. Future design tweaks to this system will be to implement drag and drop functionality to the UI so that the objects can be selected and dragged into place with a single action.

# Input

An intuitive input system is an integral part of the project that needs to be thoroughly planned out to ensure the projects user experience is satisfactory.

## Keyboard and Mouse Inputs

A widespread adoption of WASD for movement in most software tools and games means that anyone with familiarities in similar projects will be able to grasp the implementation very easily. The WASD movement will be applied to the camera’s movement controls, with W and S for forwards and backwards respectively, and A and D for left and right respectively. Two additional buttons, Q and E will be assigned to control the cameras Y axis, moving it down and up along the axis to allow for easier navigation around the environment.

To control the rotation of the camera, a very clear and obvious option is the mouse to look around the scene in a 3D view. By parsing the mouse’s movements in its X and Y axis, the rotation of the camera can be set. This is an effective measure of handling the camera rotation, however the mouse will be the main input for most of the features, which means the mouse cannot be enabled for rotation at all times. The most effective way of handling this is to add a simple modifier key that enables the rotation when held. This is achieved by simply placing all of the logic for the rotation inside an if statement for when the modifier key is down. For both ease of use and logic reasons, this key is currently set to be the left CTRL key which is easily accessible in relation to the other movement-based keys.

The transformation states of the user are the next features to consider on the key mapping. The user can toggle between three states, being translation, rotation and scale. When considering how to approach the switching of states, there are three options when it comes to key mapping. The first one being the use of a single key that switches the state to the next one in line as if the states were contents in an array being cycled through. This method is good for conserving keys for future features however the user experience of a system like this will be rather tedious when constant switching between the states is required. A slight improvement to this system would be to add a second key for going backwards which allows every state to be accessible within a single button press, assuming the three-state system is implemented. While this method is a lot easier to use than the single key method, it still doesn’t achieve the best possible level of intuition as the keys aren’t directly mapped to access a state and instead change function based on the current state. This can mean that the users target state, and in turn target key, will change based on context which may be confusing or frustrating to users. From both personal opinion and based on pre-existing software implementations, the most user-friendly, straightforward implementation would be to bind each state to a different key so that the user knows that when the particular state key is pressed, the transformation state is guaranteed to be in that state. The obvious downside to this implementation is that it takes up more keys that could be used for other features, however this is very unlikely to cause any future problems in development. For this implementation the keys R, T and Y have been used for rotation, translation and scale respectively. These keys have been assigned to the state switching as it will be likely that the user will require easy access to these keys, with these features being in constant use, and to have the keys located next to each other to ensure the key mapping overall is cohesive and rationally designed.

## User Interface

Alongside the keyboard and mouse inputs, an effective and clear user interface is an important toolset to improve on the workflow and ease of use as well as enabling many features that couldn’t easily be mapped to keys.

One of the features that the UI will allow for is an object inspector panel. This panel will be able to display information about an object such as current position and rotation of the object, as well as allowing the user to input specific parameters via text boxes.

The User Interface will contain a menu, similar to those found in most production tools, that includes project management functions such as the save/load feature, file and importing settings as well as exporting the final product.

The following figure is an example toolbar from Visual Studio which contains the familiar features and tabs such as File, Edit and Help. Keeping in line with the industry standard bar will improve user familiarity and allow them to identify feature locations easily.

A screenshot of a social media post

Description automatically generated

Figure 5 - Visual Studio's Toolbar Menu

## Coordinate Snapping

To ensure the maps objects are placed in an orderly manner and able to fit together, objects positions, rotations and scale are rounded to the nearest integer. This allows objects to be easily positioned correctly without clipping

# Example Scene

The following figures contain some example scenes created with the first iteration of the transformed tools during development. As demonstrated the objects can be moved and rotated on all three axes as well as scaled. At this stage in development, the scaling is linearly across the axis, however the planned implementation of axis locking and precise value settings.

A screenshot of a computer

Description automatically generated

Figure 6 - An example cube utilising all three transformation functions

A screenshot of a computer

Description automatically generated

Figure 7 - Three cubes manipulated into rudimentary steps

A screenshot of a computer

Description automatically generated

Figure 8 - Multiple objects positioned in 3D space

A screenshot of a computer screen

Description automatically generated

Figure 9 - Alternative angle of Figure 7

# References

Assimp, 2018. *The Open Asset Importer Lib.* [Online]   
Available at: http://www.assimp.org/  
[Accessed 22 November 2019].

GtkRadiant Team, 2016. *Radiant Level Design Tools.* [Online]   
Available at: https://icculus.org/gtkradiant/  
[Accessed 18 November 2019].

OpenGL, 2019. *OpenGL API Documentation.* [Online]   
Available at: https://www.opengl.org/documentation/  
[Accessed 21 November 2019].

Valve Corporation, 2019. *Valve Developer Community.* [Online]   
Available at: https://developer.valvesoftware.com/wiki/Valve\_Hammer\_Editor  
[Accessed 29 October 2019].

# Project Repository

The project files can be found [here](https://github.com/AlexSDevDump/3D-Level-Editor). Please note that the project is developed using Unity 2019.3 and will require access to said platform or newer.