Creating a Game Level for Export:

While hardly impossible, a custom level creation/design tool can be a time-consuming thing to create. We can get most of the object placement benefits of such a tool by leveraging existing 3D design software like Blender, Maya or 3ds Max.

In fact, many studios take the approach of integrating their game engines closely with such tools using their extensive scripting and plug-in capabilities.

<https://www.autodesk.com/campaigns/autodesk-for-games>

<https://www.gamedeveloper.com/design/using-blender-3d-as-a-3d-map-editor-rather-than-programming-your-own-from-scratch>

This can be a significant benefit when hiring artists and designers already familiar with industry standard modeling software to work on your game.

While many different tools could be leveraged here, this guide will show you how to use the popular & free 3D modeling software Blender to populate a game level. We won’t be using this 3D software for direct mesh modeling (though it can do that too), instead we will use it to import & place pre-existing 3D models. Once we are happy with the level, we will leverage a python script to traverse the Blender scene and export the names and locations of the models making up our level to a text file.

The purpose of the Level Renderer being to parse this file, loading the various 3D models onto the GPU and then rendering them to the locations designated by the text file. Anyway, let’s get started:

1. Download and install Blender: <https://www.blender.org/download/> (Get v.4.0 or newer)

Graphical user interface, text, application

Description automatically generated

1. Adjust the default control scheme to industry compatible (A.K.A Maya) controls. (optional)

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

1. Go grab some pre-made **\*.obj** 3D models. I recommend models that only have simple materials and no textures to start with. Here are some websites that may be useful: <https://quaternius.com>, <https://turbosquid.com>, <https://opengameart.org>

If you find a model you really want that is not in WAVEFRONT\_OBJ format you can always use Blender or Maya to convert it to the format you need.

Diagram

Description automatically generated

1. Inside Blender, **save your empty level** with an appropriate name in your project folder/GIT repo. (Use of a private GitHub or GitLab highly recommended)

A screenshot of a computer

Description automatically generated with medium confidence

1. Copy the models you wish to use into their own directory in your project (ex: ./Assets) and use the **Import** feature of Blender to load in a 3D \*.obj model.

A screenshot of a computer

Description automatically generated with medium confidence

1. When importing a Wavefront \*.obj file, make sure to **uncheck “Split by Object”** under the **“Geometry”** submenu. This will make it much easier to **identify the 3D model by name** you need to load into the Level Renderer when parsing the GameLevel.txt file.

A screenshot of a computer

Description automatically generated with medium confidence

1. Once your model is imported, its name should appear in the **Scene Collection** menu on the right-hand side. From here you can **delete the default cube** and use the various manipulation tools on the left-hand side to **Translate**, **Rotate** and **Scale** the imported model into position.

Graphical user interface

Description automatically generated

1. Take some time to place a few 3D models into your level. **Feel free to duplicate models** in the level. When authoring the Level Renderer, we can always filter/reuse any duplicates and just re-render something at a different location. (Or optimally use an API feature called instancing)

A screenshot of a computer

Description automatically generated with medium confidence

1. Once you are happy with your initial testing level, save your blender project and click on the **Scripting Tab** near the top all the way to the right side (You may need to enlarge your window to see it). This is how you access the programmable side of Blender.

Graphical user interface

Description automatically generated

1. **Drag & Drop** the Python level exporter script I provided in this handout to the text editor panel on the right side. Then **run the script** by pressing the **Play Symbol** near the top.

Text

Description automatically generated

1. This should run the script which will **create a text file**(GameLevel.txt) next to wherever you saved this **.blend** file. If you inspect the contents of this file, you will notice that it contains the **names of all the models** you imported into your level. Additionally, each instance of a model also contains the exact **matrix** required to reproduce its **world location** in the Blender level.

Graphical user interface, text, table

Description automatically generated with medium confidence

**IMPORTANT: You cannot HARDCODE the models used in your Level Renderer! (You will get a ZERO) You must read (File I/O) the exported game level text file to determine which H2B models to load (using H2BParser.h) and where they need to be drawn (Matrices).**

Summary:

* In addition to **\*.obj** files you gathered earlier; you should now have all the information you need to reproduce this level outside of Blender in your chosen hardware API. (The Level Renderer)
* While it is possible to parse the **\*.obj** file format on your own, supporting features like material(**\*.mtl**) files and **non-triangulated geometry** is non-trivial and would take a while.
* In **Assignment 2** you acquired a tool called **“obj2Header.exe”** this executable was used to convert a 3D model file into a C/C++ header file so it could be more easily utilized in that lab.
* Header files are not a good fit for the Level Renderer because you should not need to **recompile** it each time someone creates or edits a game level. The game should instead detect any model from GameLevel.txt and load it in at **run-time** from the Assets/Models folder.
* To that end I have added a feature to **obj2Header.exe** that will also generate a **\*.h2b** file in addition to the standard header file. This file format is called **“Header 2 Binary”** and is essentially just a byte dump of the contents of the generated header file.
* This means when parsing the **GameLevel.txt** file you can use the included names to find the appropriate (**model name).h2b** file and then read its contents without parsing the **\*.obj** format or using the model’s header file.
* Because **\*.h2b** is a proprietary format intended for simple rendering tasks, I also provide a **h2bParser.h** file that contains a small class capable of reading this format very efficiently. Use it when loading a model’s data for the first time. (You can then reuse that data for each instance)
* The provided level exporter script is quite basic, it gets the job done but not much more. If you want to edit and **customize it** to better fit your needs, feel free to do so. Just be sure to **provide a copy of it** for grading purposes.

Hints/Tips/Bonus:

* We will never use the **.blend** file directly in the Level Renderer. Blender’s primary purpose is to just move **\*.obj** files around and produce the **GameLevel.txt** file. (A Basic Level Editor)
* The **obj2Header.exe** tool can triangulate most wavefront meshes but struggles with a few edge cases. If you run the tool via the command line it will **report any issues** it has when converting meshes to headers/h2b files. Problematic meshes can be imported, **triangulated in Blender**, and re-exported as it has a much more robust triangulation system. (Then re-run the tool)

A screenshot of a computer

Description automatically generated with medium confidence

* Unlike Hardware graphics APIs, **Blender has back-face culling disabled by default**. You can account for this by enabling it on a **per-material** basis in Blender, or just disabling it in your hardware rendering API. (The latter is simpler, the former more optimal)

Graphical user interface

Description automatically generated

* Blender has a **“Walk”** Navigation mode! Once your level has a “floor” of some type, use this mode to walk around and look at things from a first-person perspective.

Graphical user interface

Description automatically generated

* Blender can also be used to **preview lighting effects** in your game level. **Switch the shading mode to “Rendered”** to see how any created light sources interact with the materials. Exporting these light sources and their attributes for use in your Level Renderer is very much a possibility.

A screenshot of a video game

Description automatically generated

* **Really want to use Maya instead?** Maybe just looking for a much more **robust exporting script** for Capstone or your own personal game/engine development? I have not tested these personally, but they do seem like more polished scripts than the one I threw together:
  + <https://jhocking.itch.io/dropper-for-maya>
  + <https://jhocking.itch.io/dropper-for-blender>
  + **IMPORTANT:** For this course stick to the provided script, unless you don’t mind being a Guinea Pig for everybody else. (And you don’t mind the risk to your grade!)