Ultimate Train Simulator - CS324 report

Felix Bowyer

0. Contents

Running Instructions

To run the game, Python 3's HTTP server module can be used, as follows.

(1) Unzip the coursework submission, and navigate in a terminal to the root folder (if in the correct folder, you should see folders titled "assets", "extra_libraries" and "solution", and this report, by running `ls`)

(2) Run the command `python3 -m http.server` . If `python3` isn't a valid command on your system, install Python 3, make sure Python 3 is added to your PATH, and find the correct command for Python 3. Please note that this will not work if you are running the executable for Python 2.

(3) You should see output similar to:
`Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...`
In Google Chrome, navigate to the mentioned link, appending "solution" to the end. In the above example, you should navigate to `http://0.0.0.0:8000/solution`

(4) You should now be on the main menu screen of the game! Once you are done, close the Python http server with a keyboard interrupt (`ctrl+c` or `ctrl+d`) in your terminal.

Word count: 1455

Blender Modelling

For the project, 3 3D models were made in Blender - the train, rails, and station signs. For the sake of brevity, this section will focus on the train design.

2.1. Train Design—The train was chosen to be modelled on a Class 373 / Eurostar e300 train for two reasons; floor and side plans were easily available, and I happened to be on one when starting this project.

Firstly, a base to work on was created. This was made by resizing and lining up a cube with the floor plans and side texture.

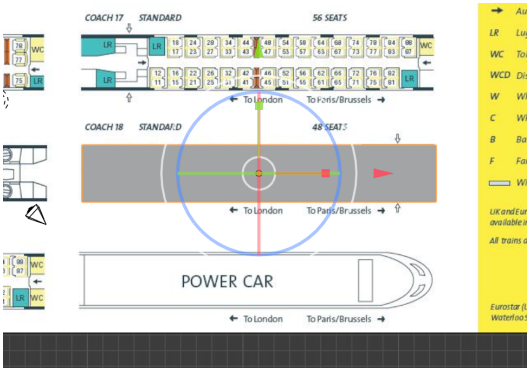

Fig. 1. Lining up the roof.

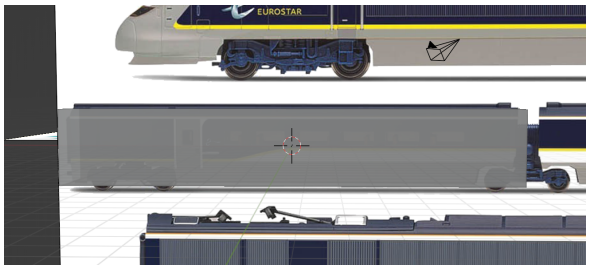

Fig. 2. Lining up the side.

Next, using Blender's orthographic view and knife tool (using "cut through" mode), features such as the windows, roof protrusion and door were cut out. The knife tool was also used to cut out the wheel arches, which were then extruded inwards to create the arch.

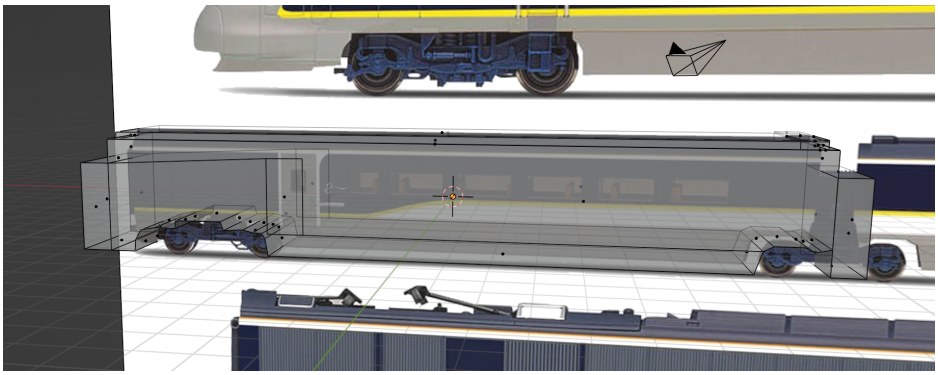

Fig. 3. The train, mid-knifing.

The doors and windows were then inwardly extruded, and wheel support axles created on the underside of the train. The wheels themselves were made as separate objects - cylinders with the inner ridge indented to the same width as the rails. The wheels were modelled as separate objects so that they could have a different material applied to them.

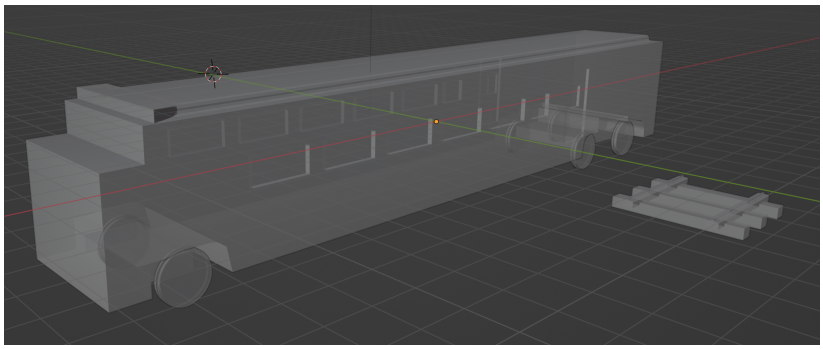

Fig. 4. The untextured train, pictured with the rails.

Next, materials and texture were applied to the train. For the wheels, a modified default texture with increased metallic property and specular reflections was used. For the main train body, the side texture used to carve details against was UV mapped onto the model. Since no texture was available for the top of the train, part of the side texture was repurposed for it.

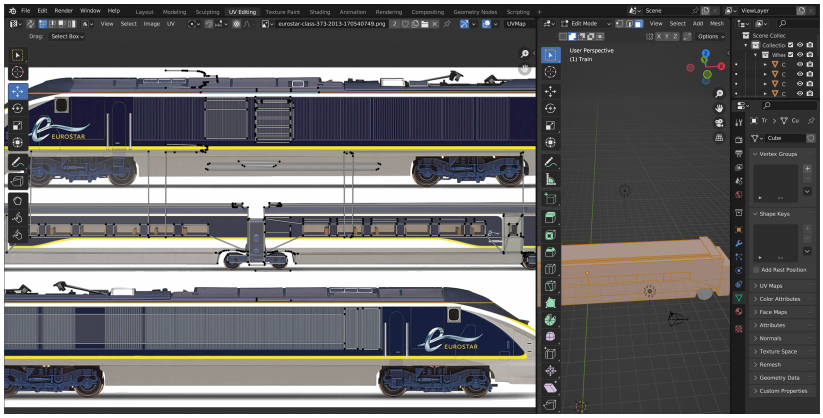

Fig. 5. UV mapping the texture.

Some final modelling touches were made (using the bevelling tool to round some sharp edges, smoothing the wheel arches, etc), and the train was then exported in glTF format to use in Three.js.

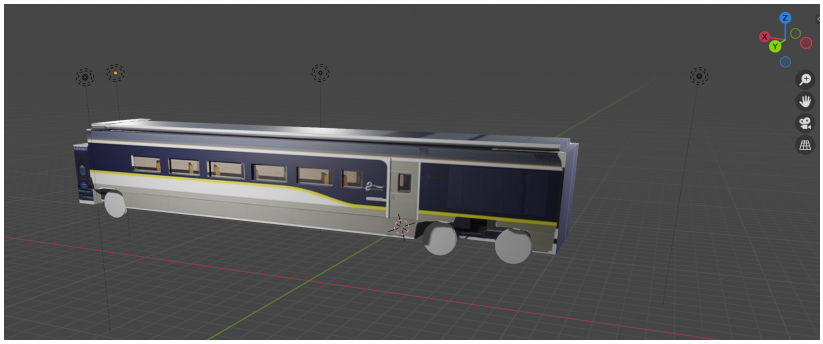

Fig. 6. The complete train.

2.2. OTHER MODELS—Two more models were created for this game - the rail model, and station sign model.

The rail model was created from intersected cuboids, with the top cuboids modelled to the shape of a rail. The protrusion on the top is the same width as the wheel model's indentation. A wood texture was applied to the bottom, and a metal texture to the top. The station sign was created using a text object intersected with some cuboids, and a basic black and white texture applied via UV mapping.

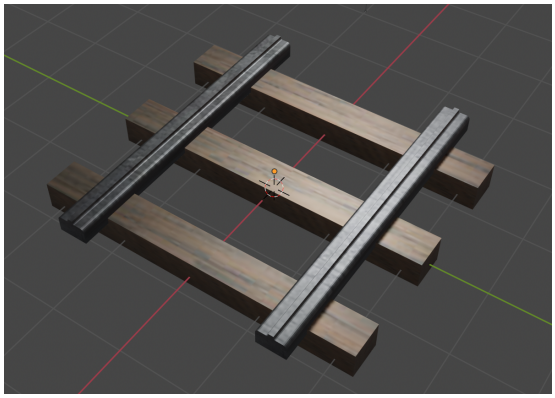

Fig. 7. Rail model.

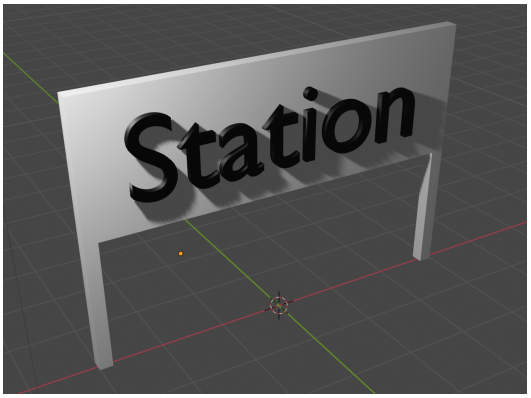

Fig. 8. Station sign model.

Some models were used in the game that were not created from scratch. These are the building and street models used in towns, and the trees placed around the map. These were all created by Quaternius[1] , and used under a Creative Commons license.

Game Development

The game was made using Three.js[2] . The map controls demo[3] was used as a starting point for the game.

3.1. INITIATION—Before we can do anything, the game environment has to be set up. This is done in the `init` function. Some of the items set up are:

- The Scene, `PerspectiveCamera`, and `Renderer` (the three things needed to display anything[4])
- A `Raycaster`, to get intersections later on
- Controls (`MapControls`), attached to the camera and renderer
- Sound elements using `AudioListener` and `AudioLoader`
- The `floorMesh` and lights (ambient lighting, and 2 directional lights - one daylight coloured, and another from the opposite angle sunser-coloured)

In `init`, 3D models are also loaded in. Most of the models (including all of the custom made ones) are loaded with `GLTFLoader`, but the street and tree models are FBX models and therefore loaded with `FBXLoader`. Once the buildings and streets are loaded, `generateTown` is called - more on this later.

3.2. RAIL DRAWING—To draw rails, the user enters rail placement mode, and inputs several points (obtained by raycasting the mouse position from the camera to the `floorMesh`). As these points are selected, they are pushed to the `splinePoints` array (after having the prospective curve's validity checked in the `splineBendValid` function). Two rail splines can be joined together - this is done by drawing from the active spline to the end of an inactive one.

When the user leaves rail placement mode, the spline curve to draw the rails along is calculated, as well as some evenly spaced points along the curve, and is pushed to the `railSplines` array (which contains all sections of rail, not just the one currently being drawn).

In the render loop, the `drawRails` function is called on each rail spline, which places copies of the loaded rail model along the evenly spaced points calculated earlier, and rotates them accordingly.

3.3. TOWN GENERATION—As mentioned, towns generation is called from the `init` function. Every town has a common structure, but is procedurally generated - it has 1 or 2 houses on the left and right of the station, and may have a street in front of the station. This takes place in a few stages:

(1) Generate the locations to place buildings and their corresponding streets, relative to the centre point of the town
(2) Place the buildings and streets by copying the loaded in meshes, rotating if need be
(3) Add the station sign behind the station, and calculate the end points of the 2 platform rails
(4) Add the end points to `railSplines`

Additionally, a name is generated for the town, and an entry is created for passenger demand.

3.4. JOURNEYS—Now that we have stations, and connections between those stations, we can set up journeys. A journey is a route between stations on the same rail spline, as well as a frequency with which trains should set off. The process to create a journey is as follows:

(1) The user enters journey placement mode. This causes platforms at stations to be rendered with a white block.
(2) The user selects platforms. Similarly to rail placement, this adds them to the `currentJourney` array. Before being added, it is checked if the platform is on the same rail spline as the previous platform (i.e. the journey is connected)
(3) On exiting journey placement mode, `currentJourney` is added to the `journeys` array.

On each render cycle, each journey's frequency is checked against the timer - if the time is a multiple of the journey's frequency, then a new train sets off.

3.5. UI—The purpose of the UI in this game is to enter and leave placement modes, manage journeys, and navigate the game (starting the game, quitting the game). The UI is accomplished by placing HTML divs over the game, providing a HUD, and is themed with London Underground design and typefaces using CSS.

The main menu is presented to the user at the start of the game, and allows the user to navigate to the level selection screen, or an explanation of how to play. All screens provide back / exit buttons. The level selection screen has 2 main buttons, to get to both levels of the game.

Over the actual game, ”Place Rails”, ”Create Journey”, and ”Manage Journeys” buttons are present. These interact with the game through event listeners in JavaScript, and activate/deactivate the relevant modes by toggling globals used elsewhere in the program.

”Manage Journeys” doesn't activate a mode per se, but instead shows the management UI. When a journey is created, this menu is regenerated to include the new journey. If any exist, the user can choose one from the drop down and fetch the info, where the user can change the journey frequency, or delete the journey. This dynamic menu generation is done programatically, and buttons are handled through their relevant event listeners. The UI also displays the current score, time, and provides pause and quit buttons, as well as an ”info” box which shows relevant information (eg. ”invalid curve” when placing rails) when required.

3.6. GAMIFICATION, PASSENGERS & LEVELS—To complete the game, a few additions were made. Firstly, when a train sets off, the player loses 50 points (as a train cost). When a passenger is delivered, the player earns one point. The game ends when the timer reaches 500 seconds, as which point the game remains ”paused” and the final score is shown.

Passengers gradually appear at stations probabilistically over the game. When the stations are initially generated, demand probabilities for each other station are randomly generated. Per these probabilities, a ”waiting passenger” count is incremented. Each train also has a passenger count. Passengers will leave stations and enter trains iff the train is going directly to their destination station. The number of passengers waiting at a station can be seen by entering journey placement mode, and hovering over a platform.

A second winter themed level was created, which provides an additional, unique challenge to players. In this mode, trains cost twice as much to run, and there are twice as many passengers, encouraging a unique play strategy. Additionally, it uses vastly different textures, objects and theming - the randomly placed trees now have snow on top, the directional lighting is darker and warmer (reflecting a winter's night), the floor is snowy, and each train has its own light source, making for pretty graphics.

3.7. ENHANCEMENTS—As mentioned, the game includes a HUD, showing the user time remaining and score. Sound effects are also used - a sound is played when the player gains score. Collision detection between the mouse cursor and the floor is implemented using `Raycaster`, and the extra lighting for each train in the winter level enhances the game's aesthetics.

Notes and References

[1] Quaternius. Quaternius - Free Game Assets. https://quaternius.com/, 2023. [Online; accessed 01/23].

[2] MrDoob. Three.js. https://threejs.org/, 2023. [Online; accessed 01/23].

[3] MrDoob. three.js webgl - map controls. https://threejs.org/examples/misc_controls_map.html, 2023. [Online; accessed 01/23].

[4] MrDoob. Creating a scene - three.js docs. https://threejs.org/docs/#manual/en/introduction/Creating-a-scene, 2023. [Online; accessed 01/23].