RacetrackStats Application Overview

Maxime Lafrance

In the world of modern race car driving, data analysis plays an increasingly important role in being an effective driver. Modern racing vehicles record megabytes, or even gigabytes of data every second based on every facet of the race. Many different software solutions already exist to analyze this type of data, such as Podium and TrackAttack. RacetrackStats is designed to be a different software than its peers because of its ability to render a real-time simulation which reflects the data which has been recorded by a driver. While all telemetry analysis programs have the ability to display the data in the form of graphs and charts, RacetrackStats will allow the user to view a render of their vehicle in a scene mimicking the race track the data is recorded for. This can help drivers analyze their data with more context, and could lead to more useful insight.

As it is, the RacetrackStats software should be considered to still be in-development, and in a pre-alpha stage. Due to the constraints placed on the development of the software, the features RacetrackStats possesses in its current iteration are not as polished as those of similar programs, but it is still a proof-of-concept for this idea of integrated rendering combined with data analysis, and provides a solid foundation which can be extended in the future.

RacetrackStats is now under the MIT License, making it an open source project. In its present moment, the software has the ability to:

- Render a scene using objects parsed from files using the Wavefront OBJ and MTL file format, with diffuse shading, multiple lights, and a skybox
- Parse and maintain CSV data
- Draw Vectors driven by CSV data according to the user's specification
- Draw Graphs driven by CSV data according to the user's specification
- Render the scene from multiple points of view. At the moment; a perspective follow-cam, a cockpit view of the car, and an orthographic overhead camera.

This idea of pairing a simulation with telemetric data could be very useful, as it would allow the user to view their car from angles that would otherwise be impossible, at any given time during their race. Furthermore, as machine vision and 3d scanning technologies become more advanced, the mesh and texture data for the simulation could theoretically be generated on-the-fly by extra cameras attached to the car. While this implementation is far beyond the scope of the application at this time, this is just one of the ways this idea could be developed in the future.

User Guide

NOTE* The current version of the software has been tested on several Windows 10 machines, with modern OpenGL drivers and 16 GB of RAM. Other machine specifications may not be supported at this time. Running the program as 'Administrator' may also be necessary, if not running through the command line or through a build environment.

Installation

The source code for the application is available for download at https://github.com/mdLafrance/RacetrackStats. This repository also contains a dropbox link to a zip file containing the track data for Mosport racetrack. A Visual Studio solution is included in this repository, so building on windows should be fairly straightforward. Make sure that the Mosport track data folder, resources folder, and default display config file are present in the destination folder that the RacetrackStats.exe file will be created in. Symlinks to these items can be created in the exe destination folder as well, if building to multiple directories to test different builds.

Using The Software

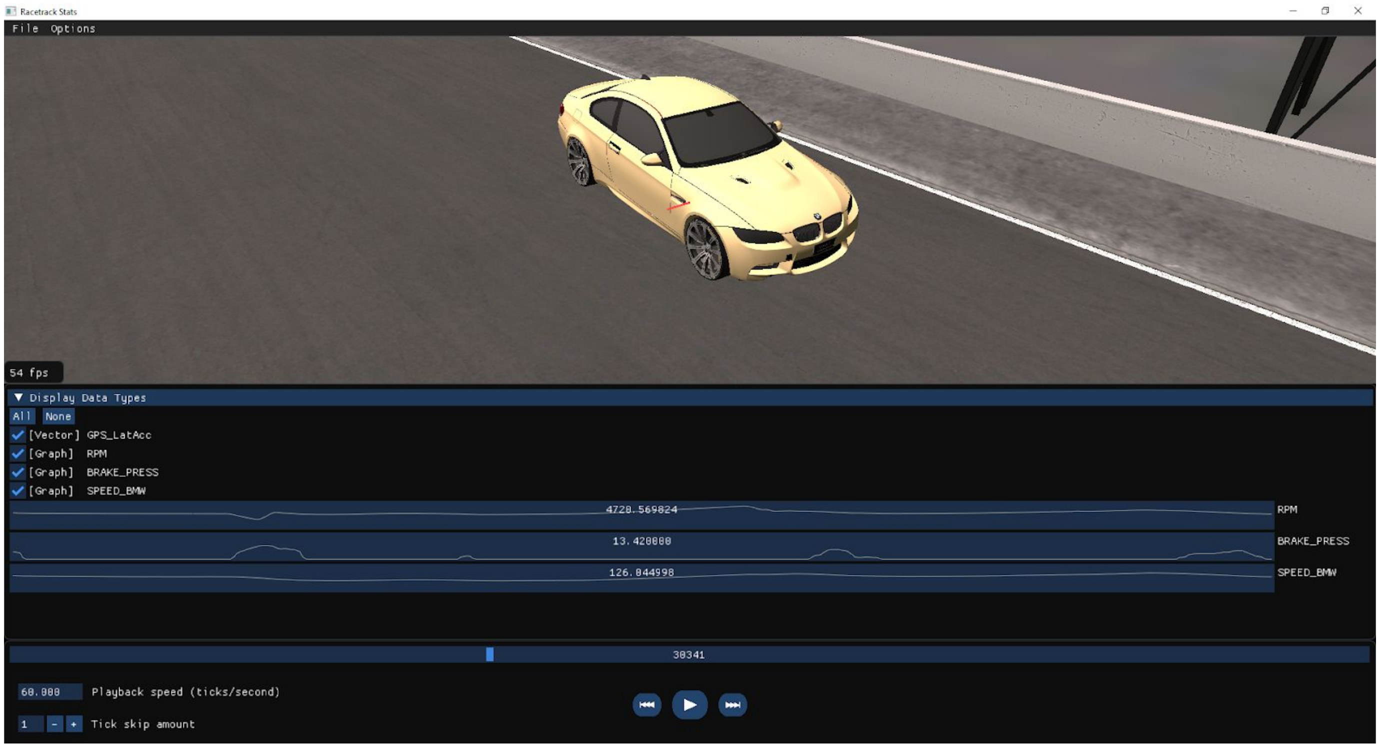
Upon launching the software, the user will be prompted to select a scene to use as the simulation for the racetrack. These scenes can be created and modified by the user, and are found in the resources/scenes folder (see .scene specification in the Developer Guide). After a scene has been selected, the software will load all necessary data, and then the main section of the application will begin running.

- The user can pan around the car using right click and drag, zoom in and out with the mouse wheel, and switch between the three main cameras ('follow', 'cockpit', and 'overhead') by pressing the 1, 2, and 3 buttons.
- Opening the 'File' menu, the user can then load any CSV track data they might have, and these data will become available to display if they are matched in the current data display config file. The user can open this same menu and load a different config file, or reload the same file if making changes on-the-fly.
- Opening the 'Options' menu exposes various parameters the user can change to alter the appearance and behavior of the software. These include some parameters to alter the cameras, graph behavior, and UI behavior.
- Pressing the 'spacebar' will pause and play the current CSV data, if any is loaded.
- Pressing the left and right arrow keys will nudge the timeline by the amount specified in the 'tick skip amount' field.
- If CSV data is loaded, opening the 'Display Data Types' dropdown will allow the user to view which data in the CSV has been defined to have some kind of visual display, and enable these displays.
- Pressing 'escape' at any time will close the software.

It is recommended that the user has the terminal accompanying the app open to view the status of the program over the course of its runtime.

RacetrackStats Developer Guide

1. **Overview**
2. **External Libraries**
3. **Code Structure and Dependencies**
4. **Next Steps**
5. **Specifications for Custom File Types**



✓ [Vector] GPS_LatAcc
✓ [Graph] RPM
✓ [Graph] BRAKE_PRESS
✓ [Graph] SPEED_BMW

```
4720.569824   RPM
13.420000     BRAKE_PRESS
126.044998    SPEED_BMW
30341
60.000   Playback speed (ticks/second)
1 - +    Tick skip amount
```

Overview

This guide is intended to give insight into the structure of the codebase, and the motivations for this structure.

RacetrackStats is an app designed to allow a user to view telemetry data analytically through graphs, and visually through vectors and a render of a scene. Various modules are used to this effect, and handle the whole pipeline, from parsing mesh and telemetry data, to handling user input. The work done by the application can be split into two major parts: rendering components, and UI/data components. The rendering components are responsible for rendering the simulation of the track, and the UI/data components are responsible for displaying the user's data, and allowing the user to interact with the app.

As the app is largely still in development, all of the intended features of the full build are not yet realized, but the infrastructure for them is in place. The key functionality that is missing as of the current build is a proper mapping of GPS location coordinates to world space coordinates for the user's car in the simulation. However, all necessary functionality exists to fetch the user's GPS data at required time points, and transformation functionality is available to move the car within the scene; the matrix to convert the GPS data to world space points is currently incorrect. All other features are present; rendering a scene, loading user data, configurable display settings, a timeline system, a (early-stage) graphing system, a vector drawing system, and configurable fidelity for scenes.

As you can see from the still image, the racetrack and user's car are being rendered, selectable data fields are driving graphs, and a vector which can be seen overlayed on top of the car. The timeline is also visible at the bottom.

External Libraries

The app uses 3rd party libraries in order to render the scene (i), draw the ui (ii), perform matrix math (iii), and load images (iv).

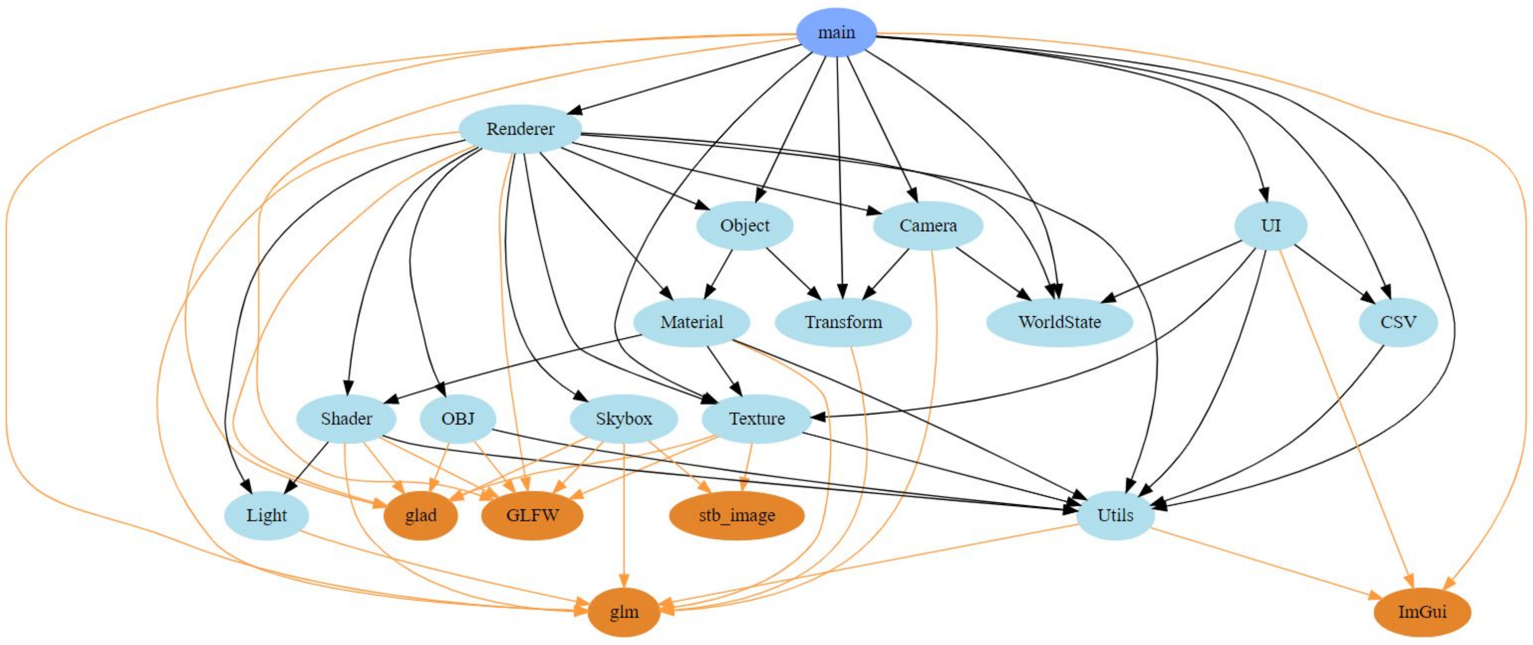
i. The rendering component was written in OpenGL. OpenGL was chosen over other alternatives (such as DirectX or Vulcan) because of the ease of development, portability, and vast amount of online support.

ii. UI functionality is implemented using an open-source library called ImGui. ImGui was a good choice for this early stage of development because of its speed and ease of use. Iterating in ImGui is very, very fast, and this was advantageous for the early stages of the app.

iii. glm was used for linear math. The choice to use glm was because of its good compatibility with OpenGL.

iv. Image loading was done using the stbi header library. This was not a very flexible choice, but it was very quick to integrate into the code.

<u>**Code Structure and Dependencies**</u>



Above is the dependency graph for RacetrackStats (as of the writing of this document). 3rd party libraries, and the connections to their dependents, are colored in orange, while modules written for this program are colored in blue.

As mentioned previously, the app consists of a rendering component, and a data handling component. This can be seen in the dependencies of the program; UI and CSV (the two modules responsible for drawing the UI and handling user data) are isolated from the rendering components. Their only point of contact is WorldState, which is a small struct used by several modules to share the state of the scene. In this way, changes to either system can take place without affecting the other.

Most of the modules in the rendering component were designed with the idea that they could be removed and used in other applications, so have been isolated as much as possible from details specific to this software.

- Almost all modules in the software use the Utils module, which is a centralized place for different kinds of functions which have no one home (such as string manipulations, file path parsing, numerical utility functions, undefined operators, etc.).
- In the current state of the software, UI is a header-only file which is only used in main, so it is effectively a part of main, which has been separated to keep the code clean, and to account for future changes.
- OBJ optionally depends on Material, but this dependence can be disabled if OBJ is not being included in the RacetrackStats app.

Next Steps

This section will highlight some potential areas for improvement for the software.

1. **Proper GPS Mapping:** Implement the proper GPS to World Coordinate mapping values to transform the user's car according to GPS data. The framework for this change is in the codebase, but the exact values for this transformation need work.
2. **UI Overhaul:** Switching to a different UI library would improve the quality of the UI, and could enable more advanced graphing tools. In the meantime, an ImGui extension header called imgui_plot is being considered for advanced plotting within ImGui.
3. **Parallelized Loading:** Parallelizing the loading of obj and png texture files would drastically reduce the amount of time it takes to load a scene, and make a smoother experience for the user.
4. **Improved Texture Module:** Using a different image loading library would add support for more types of image formats, as currently only png images are supported.
5. **Improved Mesh Loading:** Adding support for more modern file formats such as fbx could reduce loading times, and size of data folder.
6. **Deferred Rendering:** The application only uses forward rendering, but a switch to deferred rendering could give a performance boost.
7. **Improved Light Workflow:** Upgrading the lighting data structure used by the shaders and the Light module to be a more well-defined struct to be buffered in a Uniform Buffer would be more readable and performant than the current solution (See Light.h).

Custom File Type Specifications

The app uses two types of custom files to allow the user to modify the behavior of the app; .scene files and .config files.

Scene Files define the objects, textures, and lights for a scene. Each line in the file defines some facet of the scene (described below)

-Lines can start with '#', and will be ignored.

<name> : indicates the name of the scene
ex. "name Mosport Low Resolution Scene"

<mesh> : Indicates some mesh file to be loaded into the scene.
ex. "mesh mosport_trees.obj"

<material> : Indicates a mtl library to be loaded for some matching mesh. The “-t” flag after ‘material’ indicates this material needs transparency.
ex. "material -t mosport_trees.mtl"

<ambient> : Defines ambient light to be used in the scene, followed by rgb values for the ambient light (between 0-1)
ex. "ambient 0.2 0.2 0.2"

<light> : Defines a light to be used in the scene (can be more than one). Followed immediately by either 'directional' or 'point' (only directional is implemented atm).
-i defines the intensity (any floating point value)
-v defines the direction vector of the light
-c defines the rgb color of the light (between 0 and 1)
ex. light directional -i 2.5 -v -2 -10 -5 -c 0.65 0.6 0.6

Example .scene file:

```
# Low res mosport track for smaller machines
name Mosport (Low)

# Meshes
mesh jr_mosport.obj
mesh mosport_horizon.obj
mesh mosport_vehicles.obj
mesh mosport_objects.obj
mesh mosport_layers.obj
mesh mosport_trees.obj
mesh BMW.obj

# Materials
material mtl_low/jr_mosport.mtl
material mtl_low/mosport_horizon.mtl
material mtl_low/mosport_vehicles.mtl
material mtl_low/mosport_objects.mtl
material -t mtl_low/mosport_layers.mtl
material -t mtl_low/mosport_trees.mtl
material -t BMW.mtl

ambient 0.43 0.43 0.4

light directional -i 2.5 -v -2 -10 -5 -c 0.65 0.6 0.6
```

Config files define which data types from loaded CSV files should be interpreted or displayed. The application will attempt to load a file called config.txt at launch, that defines which data from the CSV the app will use to draw vectors and graphs

Lines can start with '#', and will be ignored.

<longitude> : which CSV data defines the longitude of the car
ex. "longitude GPS_Longitude"

<latitude> : which CSV data defines the latitude of the car
ex. "latitude GPS_Latitude"

<elevation> : which CSV data defines the elevation of the car
ex. "elevation GPS_Elevation"

<heading> : which CSV data defines the heading of the car
ex. "heading GPS_Heading"

<vector> : Define a vector to be drawn whose length is determined by CSV data, followed by origin vector (relative to position of the car), direction vector, color rgb, CSV data type
ex. "vector 0 0 0 -1 0 0 0.8 0.8 0.2 INLINE_ACC"

<graph> : Define type to be drawn to a graph, followed by the CSV data type, then color of the line r g b,.
ex. "graph BRAKE_PRESS 1.0 0.2 0.2"

Example .config file:

```
longitude GPS_Longitude
latitude GPS_Latitude
elevation GPS_Elevation
heading GPS_Heading

vector  0 0 0  -1 0 0  1.0 0.3 0.3  GPS_LatAcc

graph RPM 1.0 1.0 1.0
graph BRAKE_PRESS 1.0 1.0 1.0
graph SPEED_BMW 1.0 1.0 1.0
```