

Luigi Fusco

Period: Jul 5th – Sep 10th

Supervisors: Giulio Eulisse, Barthélemy von Haller

Providing a remote DebugGUI to DPL

Introduction:

[ALICE](#) (A Large Ion Collider Experiment) is a general purpose, heavy ion collision detector at the CERN LHC. The [Alice O2](#) software encompasses the framework and all code related to reconstruction, calibration, and simulation of runs 3 and 4 of the experiment.

The DPL (Data Processing Layer) of the O2 framework has knowledge of the O2 Data Model and is responsible for the validation, optimization, and scheduling of computations on a user-specified set of inputs. A DPL application consists of a Driver process, and one or more child processes called generically "data processors" or "devices". Each data processor runs asynchronously in a separate process doing some computation and sharing results with other data processors via shared memory. As they run, they report metrics to the Driver process, over a WebSocket connection. The Driver can then optionally display them in the DebugGUI. [DebugGUI](#) is developed as an external dynamic library and is based on Dear ImGui. [Dear ImGui](#) is a graphical user interface library for C++. It outputs optimized vertex buffers that can be rendered anytime in a 3D-pipeline enabled application. The task of rendering the vertex buffers and interfacing with the user is left to a series of backends. Users can easily interface with Dear ImGui data structures and create their own backends, with great flexibility and the possibility of integration in existing projects.

The task:

The objective of the project is to provide a remote view of the DebugGUI through a browser. With remote work becoming more and more common, this requirement has become critical. The initial step was to obtain the vertex data from Dear ImGui and then send it to a browser through a WebSocket connection. In the browser, the vertex data can

be rendered in order to display the DebugGUI. The rudimental application was then expanded to include support for interaction with the user, sending mouse and keyboard data to the Driver process, and updating the GUI accordingly.

The existing code:

DebugGUI:

The DebugGUI is developed as a project separate from the O2 core framework. It makes use of standard functions and shadow pointers in order to decouple it as much as possible from O2. The *PollGUI* function is called every time a new frame has to be rendered.

Websocket:

The data processors and the Driver communicate through a custom implementation of the WebSocket protocol. Websocket was chosen because of the possibility of connecting through a SOCKS proxy. Sockets are handled through libuv, providing support for asynchronous I/O and making heavy use of callbacks.

Developed features sum up:

- Remote display of the DebugGUI (*RemoteGUI*)
- Simultaneous connection of multiple RemoteGUIs
- Simultaneous remote input
- Remote keyboard input
- Remote mouse input
- Remote touch input
- Window resizing
- Rendering at multiple different frame rates

Development:

The final result consists of modifications of the source code of the O2 framework and the DebugGUI library, and the development of a small web app to show vertex data and send commands to the Driver process.

DebugGUI modifications:

The DebugGUI library works by setting up the graphical environment through *InitGUI* and then rendering the GUI every time the function *PollGUI* is called. Modifications were done to the OpenGL implementation only, and include the possibility of launching DebugGUI without opening any window and the split of *PollGUI* into separate functions, in order to isolate the event handling, vertex data creation, and rendering functionalities. These functionalities will then be separately called by the Driver process when needed. Functions added to the library include the transformation of the Dear ImGui vertex data format, which uses pointers and C++ data structures, into a binary format that can be transmitted over a network and later interpreted. The format is structured as follows:

--- HEADER ---

number of vertices	:	32 bit int
number of indices	:	32 bit int
number of commands	:	32 bit int

--- PAYLOAD ---

foreach vertex

- posX	:	32 bit float
- posY	:	32 bit float
- uvX	:	32 bit float
- uvY	:	32 bit float
- color	:	32 bit int

foreach index

- index	:	16 bit unsigned int
---------	---	---------------------

foreach command

- index count	:	32 bit int
- clipX	:	32 bit float
- clipY	:	32 bit float
- clipW	:	32 bit float
- clipZ	:	32 bit float

--- END ---

The index count of the commands is interpreted as an incremental range of the indices to be rendered with an associated clip scissor. The sum of the index count of all commands is equal to the total number of indices.

Webapp:

The web app has to establish a WebSocket connection with the Driver, render incoming vertex data, and send mouse inputs, keyboard inputs, and various commands. As the only data which is sent by the Driver is vertex data, the rendering function is implemented as the *onmessage* callback of the created WebSocket. The rendering is done with the support of the three.js library. [Three.js](#) is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using [WebGL](#). In order to render the vertex data, a single *BufferGeometry* is used. At each frame, all attributes of the *BufferGeometry* (vertices, uv, color, alpha, and indices) are updated. Commands are executed in order, each one modifying the scissor of the renderer and the index draw range of the geometry. The geometry is rendered using custom vertex and fragment shaders. Dear ImGui supports by default only one texture, containing characters and basic shapes. As it is always the same, it is loaded together with the web app as a png image instead of being sent by the Driver. Commands sent from the web app to the Driver use the following format:

```
--- HEADER ---
opcode      :      8 bit
--- BODY ---
payload     :      variable length
--- END ---
```

The supported functionalities are mouse position update, mouse click, mouse wheel, fit to window, set latency, keydown, keyup, and char input.

When opened, the user is prompted to insert the IP and the port of the Driver. As the application is launched, it will start rendering the incoming data, and a [dat.GUI](#)

menu will give the user the possibility of enabling the interactive mode, fitting the GUI to the screen, and updating the frame delay time.

O2:

Several modifications were done to the main O2 framework. These modifications regard both connection management and GUI management.

Websocket implementation:

A bug in the WebSocket resulting in altered header formatting and preventing the connection from external WebSocket implementations was fixed. The bug was caused by the incorrect ordering of data in the header due to the incorrect use of bitfield structs to read and write the header.

Remote connection:

The old assumption that all connections come from data processors does not hold anymore. The *WSDPLHandler* was modified in order to detect a connection from a RemoteGUI if the header *x-dpl-pid* is not present. The creation of the *mHandler* attribute was moved inside the *endHeaders* method of *WSDPLHandler*. If the connection comes from a RemoteGUI, a *GUIWebSocketHandler* is created instead of a *ControlWebSocketHandler*, and rendering related data structures are set up. The *GUIWebSocketHandler* class is responsible for interpreting and executing commands coming from the RemoteGUIs.

GUI:

Connected RemoteGUIs are kept track of in the *GuiCallbackContext* (atm this is used only for counting the number of RemoteGUIs currently connected). Every RemoteGUI is associated with a *GuiRenderer*. A *GuiRenderer* contains a *uv_timer_t*, a reference to the *GuiCallbackContext*, and a reference to the RemoteGUI's *WSDPLHandler*. The *GuiCallbackContext* stores the *GuiRenderers* in a set. Whenever a connection with a RemoteGUI is established, a *GuiRenderer* is initialized and the *uv_timer_t* is started, calling *remoteGuiCallback* at intervals of 200ms, passing the *GuiRenderer* as context. A reference to the *GuiRenderer* is kept in the *GUIWebSocketHandler*. Whenever the connection with a RemoteGUI is lost, the relative *WSDPLHandler* is destroyed, which in turn destroys its

`GUIWebSocketHandler`, which removes the `GuiRenderer` from the set and stops the `uv_timer_t`.

Possible future developments:

- Browser support: the only officially supported browsers for the web app are Firefox and Safari. Other browsers (Chrome) fail to connect to the Driver. This is most likely due to the limited amount of features of the custom WebSocket implementation, with the browser not receiving all data that it expects. The path to fixing this problem would be to analyze the traffic between the Driver and the browser and comparing the messages sent to the WebSocket standard.
- Data compression: the size of the vertex data for one frame can be over 100KB. A more efficient method for data transmission would implement a differential algorithm, sending to the browser only the data that actually changed from one frame to the other. Its implementation would require the storage in the Driver of each remote connection's last frame, and implement a more complex message format for data sent by the Driver. Another approach is to analyze the different draw commands issued by Dear ImGui and sending only the updated ones.
- Dear ImGui advanced features: the current implementation does not support custom textures and frames with more than 65536 indices. These limitations do not affect in any way the current implementation of the DebugGUI (the average frame has a number of indices an order of magnitude lower than the limit) but they may be an obstacle in the future.
- DebugGUI Metal support: the changes made to the DebugGUI API only refer to the `DebugGUI.cxx` file, not the `DebugGUI.mm` file. This means that there is currently no support for compiling on MacOS (using the Metal implementation).
- Additional commands: additional commands (like stopping the application) can be trivially added by extending the communication protocol between the web app and the Driver.

Notes:

- The current implementation of the WebSocket protocol does not support the connection close command. The Driver side connection is closed on TCP timeout.
- For security reasons browser implementations of WebSocket do not support custom headers. At the moment the Driver differentiates the type of remote client (data processor or RemoteGUI) based on the presence of the *x-dpl-pid*. In order to support different types of browser based remote clients a more complex message based differentiation method is needed.

Acknowledgments:

CERN has given me the invaluable opportunity of contributing to such an amazing project. I had the chance of applying my knowledge to something truly useful, and I learned a lot in the meantime.

Thank you to Giulio and Barthélémy, for your help, guidance, and for making me feel valued. Working with you has truly been a pleasure.

Thank you to the CERN Summer Students Team for your continuous support.