EPJ Web of Conferences **251**, 01007 (2021) https://doi.org/10.1051/epjconf/202125101007
CHEP 2021

The Phoenix event display framework

Edward Moyse[1,*], *Fawad* Ali, *Emilio* Cortina, *Riccardo Maria* Bianchi[2], and *Ben* Couturier[3]

[1]University of Massachusetts Amherst, USA
[2]University of Pittsburgh, Pittsburgh, USA,
[3]CERN, Meyrin, Switzerland.

Abstract. Visualising HEP experiment event data and geometry is vital for physicists trying to debug their reconstruction software, their detector geometry or their physics analysis, and also for outreach and publicity purposes. Traditionally experiments used in-house applications that required installation (often as part of a much larger experiment specific framework). In recent years, web-based event/geometry displays have started to appear, dramatically lowering the entry barrier to use, but which typically are still per-experiment. The Phoenix framework is an extensible, experiment-agnostic framework for event and geometry visualisation.

1 Introduction

High Energy Physics event displays are used to visualise experimental geometry, and event data. They will typically have many different use-cases and clients, from experts checking the detector description is correct, that the reconstruction software is producing the correct output, or that their analysis is performing as expected, to the general public looking at an outreach presentation.

Traditionally, event displays tended to be custom applications, developed separately by each experiment (and often requiring complicated software frameworks to run). However, modern web browsers are capable of showing sophisticated 3D content, even Augmented Reality (AR) and Virtual Reality (VR). This has lead to the rise of web-based event/geometry displays, which dramatically lower the entry barrier to use, but typically these displays are still developed separately for each experiment.

In 2017 the HSF visualisation white paper [1] identified the desirability of having a common event format and common tools to aid visualisation. The Phoenix framework was adopted as an extensible, experiment-agnostic framework for event and geometry visualisation, and is also the official web event display of the ATLAS experiment [2].

2 Design principles

In order to meet the goals of being experiment-agnostic and to provide the best support to the HEP community, Phoenix was created with the following principles:

- To have a permissive licence and be open source;

*e-mail: edward.moyse@cern.ch

- To use industry standards wherever possible;
- To provide a simple format for event data;
- To have excellent documentation;
- To avoid experiment-specific assumptions;
- To be configurable, extendable and modular.

Phoenix is licensed under the Apache 2.0 license [3] and its github repository[1] is accessible to all. It uses the ubiquitous `threejs` [4] visualisation library, and the example application is created with `angular` [5]. Additionally, Phoenix's continuous integration and deployment scripts all use industry standard tools. The simple format for event data is covered in Section 4.6. Phoenix is documented with extensive in-source guides[2] (including a user manual, developer and release guides), whilst the APIs are documented using `compodoc` [6]. The flexibility and configurability of Phoenix will become apparent as other features are discussed.

In addition to meeting these design principles, Phoenix must provide the relevant functionality to address its core goals of supporting outreach activities, as well as powerful geometry, reconstruction and analysis debugging.

Examples of required functionality are:

- loading and clearly displaying the required geometry and event data, and being performant enough whilst doing so;
- providing a common event format;
- and having a sophisticated and intuitive user interface (UI).

These will be covered in the following Sections.

3 Example Implementations

One of the ways in which Phoenix demonstrates its support for a variety of HEP experiments is by having several different built-in examples, which show various HEP experiments visualised using Phoenix. These real-world implementations also make it easier to test for regressions and incompatibilities as new features are added.

When a user visits the Phoenix online demonstration[3], they will see Figure 1, a webpage showing the six examples currently available. These consist of two technology demonstrations: a 'playground' for testing new experiments (where a user can load, scale, and translate geometry, whilst being able to load standard supported event data), and a 'geometry display', where you can try to design a detector parametrically. There are then three demonstrations that represent[4] actual experiments (ATLAS, LHCb [7], and CMS [8]) and finally the TrackML demonstration, that shows the imaginary HL-LHC detector created for the TrackML Kaggle challenge [9].

4 Phoenix functionality

4.1 Geometry

Visualising a HEP detector involves displaying very complicated detector geometry. Luckily, modern browsers can leverage WebGL to provide impressive performance on modest hardware, such as a mobile phone. Even scenes with many hundreds of thousands of triangles are

[1] https://github.com/hsf/phoenix
[2] https://github.com/HSF/phoenix/tree/master/guides
[3] https://hepsoftwarefoundation.org/phoenix/
[4] The geometry used is representative and may not be the current version.

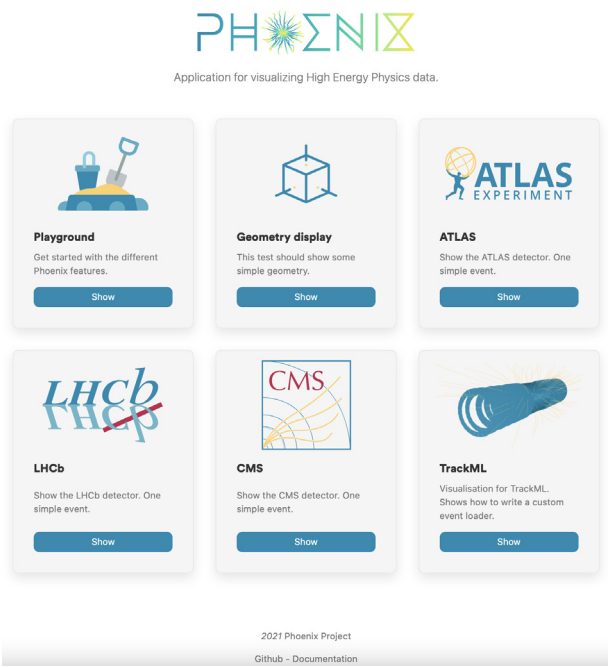

Figure 1. The Phoenix landing page, showing the six built-in demonstrations. One is for testing new geometry and event data, another is for building geometry parametrically, three show three real experiments (ATLAS, LHCb and CMS) and finally, one pseudo HL-LHC detector created for the TrackML Kaggle [9] challenge. There are also links to the repository and to the documentation.

easily possible. Care must still be taken to simplify the models as much as possible, not just to maintain frame-rate, but also because downloading excessively large geometry files can hinder the responsiveness of the application, depending on the available bandwidth. There are also practical limitations to the amount of memory that can be consumed by a browser.

Since Phoenix uses the `threejs` library, it can support an enormous number of industry standard formats out-of-the-box (the recommendation is to use glTF [10] however). The CMS demonstration also shows that Phoenix can be extended to support HEP-specific formats, in this case TGeo via `jsroot` [11].

4.2 Physics objects

The second task of any event display is to clearly display the physics objects. Care must be taken to display these in a way that is accurate, but that also intuitively expresses the underlying physics.

Phoenix currently supports the following physics objects:

- Tracks: representing charged particles moving in a magnetic field, Phoenix can draw these as either lines or tubes, depending on the configuration.
- Jets: Phoenix represents these as simple geometric cones pointing in the direction of the energy flow.
- Hits: individual measurements from e.g. silicon detectors, drawn as a point in space.
- Clusters: energy depositions in a calorimeter, drawing a pointing cuboid.

- Vertices: the common origin of other physics objects.

Moreover, the ATLAS and CMS demonstration applications add:

- MuonChambers - the details of the Muon chamber geometry passed by tracks.
- Compound objects (such as Muons, Photons, Electrons) - objects that are mostly defined (and visualised) by their links to other objects e.g. Tracks, Clusters.

4.3 User interface

The user interface of any application is extremely important, but especially so for an application where the users will potentially span from novice (outreach) to expert (reconstruction and geometry debugging).

The most basic (and immediate) way of interacting with Phoenix is through the display using a mouse (or headset, if in VR mode). Here you can zoom, rotate and move, using mouse-wheel, click-and-drag and shift-click-and-drag respectively (there are also keyboard shortcuts for various actions, as explained in the user manual).

However for more complicated operations, a graphical user interface (GUI) is required. Phoenix currently support two GUIs, the lightweight `dat.GUI` [12] and the much more powerful `PhoenixUI`, which this paper will concentrate on.

Exactly what is shown in terms of GUI can be configured, but Figure 2 shows a typical setup (which the caption explains in more detail). For the sake of brevity, this paper will focus on the parts of the UI labelled 3), the Phoenix menu and 4), the icon bar.

The menu allows the user to enable/disable geometry, change how the geometry is rendered (i.e. its colour, opacity and whether to show it as wire frame). Similar options are available for event data, with the addition of cuts, (for example, to show only tracks above a certain transverse momentum). Finally, there are visualisation options for labels, which are discussed below. The menu also allows users to save and load the current state, or configuration.

The icon bar controls how the user interacts with the view and the data. From left-to-right clicking on the buttons will change the current event, zoom in/out, open a view dialog (accessing some configurable pre-defined views), auto-rotate the view, toggle dark mode, open the cut dialogue, toggle perspective mode, toggle the overlay view, toggle object selection (and show the related window), toggle an info panel (showing messages from Phoenix), toggle collections view (see below), animate the event with a simulation collision, animate the event by flying through it, enable performance mode (for slower hardware), enable VR mode (if available, see below), take a screenshot, and finally, show the import/export dialogue (allows loading of event data and geometry, and exporting the scene).

This paper cannot address the full functionality of this framework, but will cover a few highlights. Firstly, the overlay allows users to toggle an additional draggable window, which shows a different view of the detector. This can be set to perspective or orthogonal mode, irrespective of the settings of the main view, and can be fixed (or frozen) once it is arranged correctly. One use case might be to show a transverse slice of a colliding beam experiment (as shown in Fig 3) but many other options are possible. Secondly, the object selection and collection view offer two ways to display extra information about the objects in the display. Once object selection is enabled, any 3D object that is clicked will display information about itself. The collection view shows a list of similar information about all the objects in a chosen collection. The two are linked, so if an event data object is selected in the display, it will also be highlighted in the collection view. Similarly, if an object is selected in the collection view, it will be highlighted in the display. The collection view also allows users to focus the display view on the selected object, which can be very useful for complicated events. The collection

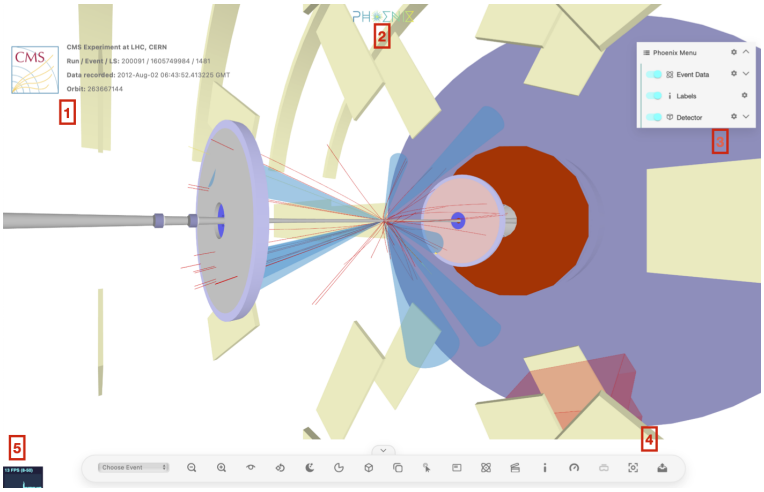

Figure 2. A typical view of the Phoenix user interface. In the centre is the actual display, showing geometry and event data. Around this display are: 1) the (configurable) experiment logo and event data, 2) the Phoenix logo (which is also a link back to the demonstrations grid), 3) the Phoenix menu (which controls how geometry and event data are visualised), 4) the Phoenix icon bar (which controls interactions with this data), and finally 5) the performance graph, here showing frames-per-second (FPS).

view is also where users can label objects - for example, label a track "the leading muon", for analysis presentation or outreach.

Finally, if the correct hardware is available, which can range from a simple mobile phone with VR lenses (e.g. Google cardboard) to a dedicated VR headset, then it will be possible to enter a VR view of the event. With appropriate controllers, the user can move around in the virtual space. Interacting with objects in VR is a development goal for 2021.

4.4 Tools provided by Phoenix

In addition to the UI already mentioned, Phoenix provides many tools to help display geometry and event data. One notable example is the Runge-Kutta propagator. Many experiments save space by only storing the first kinematic quantities of charged objects. This makes visualising them correctly challenging, unless you can propagate them correctly through the detector. The Runge-Kutta propagator tool allows developers to do this, as is shown in the ATLAS demonstration for the `InDetTrackParticles` track collection.

4.5 Outreach

Outreach is a special use case for Phoenix; a recent developments have been focused on related functionality. One example, is adding the ability to set up a Phoenix server and open a display by passing configuration and the required event data via URL. One possible use-case of this is for an analysis presentation: the analysers (or outreach representative) would give the audience a URL which when opened, loads a representative event for the analysis, correctly configured to highlight the primary physics objects and to arrange the detector to make this as clear as possible.

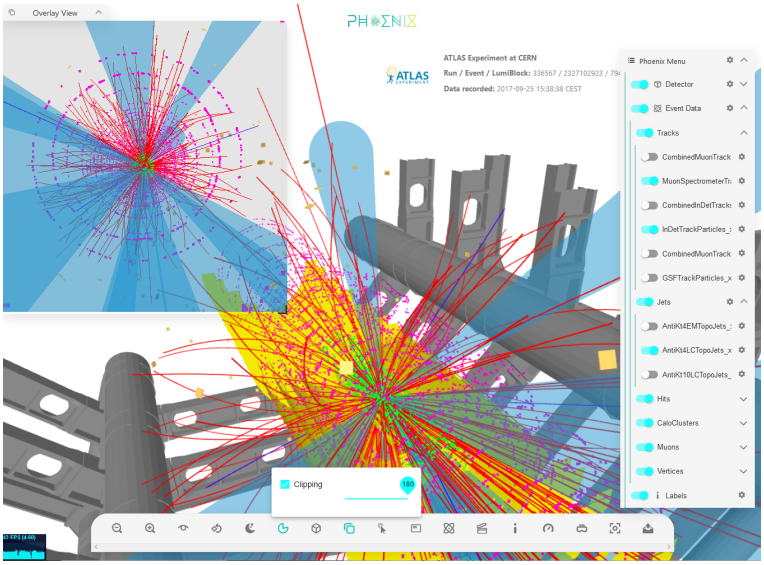

Figure 3. A view of the ATLAS detector, showing (top-left) an orthogonal transverse view overlaid on the main display, and dynamic clipping in action, where a configurable slice of geometry is removed to make it easier to see the event data. The menu on the right-hand side controls what event data is being visualised.

Currently this is implemented as follows: before the presentation the event data and configuration is made accessible to the server, e.g. via an EOS folder, and then you use a URL e.g. http://myserver.com/?file=evt.xml&type=jivexml&config=cfg.json

This setup can also be used to visualise live events (as long as they are in the appropriate directory), and so could be used for a real-time display.

Another recently added option is the ability to minimize the application and therefore directly embed a simple Phoenix view in an outreach announcement, or physics discovery briefing. This currently disables all UI elements, except for the experiment logo and event details.

4.6 Common Event format

Internally Phoenix uses a JSON-based event data model, known as Phoenix-format. This is purely for event-display purposes and is not intended for storing large numbers of events. The primary design goal is to be human readable and experiment-agnostic. It is structured as follows:

```
{
  "EVENT_KEY_1": event_object,
  "EVENT_KEY_2": event_object,
  ...
  "EVENT_KEY_N": event_object
}
```

where `EVENT_KEY` is the event identifier (i.e. a string explaining what it contains) and the format of `event_object` is of the form:

```
{
  "event number": XXX,
  "run number": YYY,
  "OBJECT_TYPE_1": {
    "COLLECTION_NAME_X" : [ PHYSICS_OBJECTS ]
  },
  "OBJECT_TYPE_2": {
    "COLLECTION_NAME_Y" : [ PHYSICS_OBJECTS ],
    "COLLECTION_NAME_Z" : [ PHYSICS_OBJECTS ]
  }
}
```

where `PHYSICS_OBJECTS` are the supported physics objects, as described in Section 4.2. The exact formats of these `PHYSICS_OBJECTS` are outside the scope of this document, but several examples are provided with Phoenix.

5 Structure of the code

Phoenix is split into two packages, `phoenix-event-display` and `phoenix-ng`, which is the angular application used for the online demonstrations. The two packages can be browsed online[5] but in the next two sub-sections we will briefly explain the contents.

The package `phoenix-event-display` and sub-package `phoenix-ui-components` (part of `phoenix-ng`) are both published to `npm` and can therefore be installed very simply:

```
npm install phoenix-ui-components
npm install phoenix-event-display
```

This is explained in more depth in the Phoenix developer guides[6].

5.1 phoenix-event-display

This is the heart of Phoenix, and where all the core code lives. As already mentioned, it is an independent package which can be installed via `npm` or `yarn`. The package documentation contains examples for using it as a module, as a standalone (minimised) bundle, and as a react application (and of course, `phoenix-ng` is a very complete example of its use in an angular app).

Inside `packages/phoenix-event-display/` there are several sub-directories, and `event-display.ts` which defines the main `EventDisplay` class of Phoenix, and which is responsible for all of the core functionality, such as configuration, loading geometry, enabling VR mode, etc etc. The complete API documentation can be found here[7].

Particularly interesting subdirectories are: `loaders`, where the event data processing occurs (and the physics objects which are later rendered); `helpers` which contains the Runge-Kutta propagator, `managers`, which has code to handle state (or configuration); and `three`, which contains managers specifically designed to handle important interactions with `threejs`. Examples are:

- **RendererManager**: controls the renderer, including extra features such as the overlay.
- **VRManager** : responsible for handling the change to VR mode.
- **SceneManager** : handles everything related to the scene.

[5] https://github.com/HSF/phoenix/tree/master/packages
[6] https://github.com/HSF/phoenix/blob/master/guides/developers/set-up-phoenix.md
[7] https://hepsoftwarefoundation.org/phoenix/api-docs/classes/EventDisplay.html

5.2 phoenix-ng

`phoenix-ng` is the angular project used to create the example shown at the online demonstration. It consists of two main components, `phoenix-app`, which houses the landing screen and the various examples, and `phoenix-ui-components`, which is where the custom Phoenix user-interface is defined. This is further split into two main sections, `ui-menu`, which houses all the functionality of the icon bar (such as animating the event, changing geometry cuts, turning off and on overlays etc), and `phoenix-menu`, which houses the code to draw the right-hand menu, which determines which geometry and event data is visible (and saving and loading configuration).

6 Customising Phoenix

In order to add a new experiment, the developer needs a few things: the detector geometry, event data, and an interface.

The geometry is in many ways the easiest step. As explained in Section 4.1, the `threejs` library used by Phoenix supports an enormous number of industry standard formats, and so Phoenix can display any of these. However, geometry loaders can be written to read any format.

As already mentioned in Section 5.1, Phoenix uses `loaders` to convert from arbitrary event data structures into Phoenix format (Section 4.6), and from these construction representations of the physics objects (Section 4.2).

The `EventDataLoader` interface works as a base for implementing all the loaders including the `PhoenixLoader` (and other loaders, such as `JiveXML` and `LHCbloader` in turn implement `PhoenixLoader`)

The next step is to add an experiment directory to the 'demos' folder: `phoenix-ng/projects/phoenix-app/src/app/sections`. The two most important components to add are: `experiment.component.html`, which specifies what Phoenix UI components appear on screen, and the experiment specific implementation `experiment.component.ts`, where the developer can define the configuration (or load a previously saved configuration file), load geometry, specify event data loaders, etc.

7 Conclusions

Visualising event and geometry data has typically been done separately per experiment. However recent developments in web technology make it feasible to run a powerful, complete event display in a browser. Phoenix is an open-source, experiment-agnostic framework that makes this possible, supporting many geometry and event data formats, as well as providing its own event data format. Since the core functionality is complete, recent developments have concentrated on outreach functionality, and it has a busy development schedule for 2021.

References

[1] M. Bellis, R.M. Bianchi, S. Binet, C. Bohak, B. Couturier, H. Grasland, O. Gutsche, S. Linev, A. Martyniuk, T. McCauley et al., *Hep software foundation community white paper working group — visualization* (2018), `1811.10309`

[2] G. Aad (ATLAS Collaboration), JINST **3**, S08003. 437 p (2008), also published by CERN Geneva in 2010

[3] *Apache 2.0 license*, https://www.apache.org/licenses/LICENSE-2.0 (2021), accessed: 2021-02-26

[4] *Threejs, javascript 3d library*, https://threejs.org (2021), accessed: 2021-02-26
[5] *Angular*, https://angular.io (2021), accessed: 2021-02-26
[6] *Compodoc*, https://compodoc.app/ (2021), accessed: 2021-02-26
[7] A.A. Alves, Jr. et al. (LHCb), JINST **3**, S08005 (2008)
[8] S. Chatrchyan et al. (CMS), JINST **3**, S08004 (2008)
[9] *Trackml particle tracking challenge*, https://www.kaggle.com/c/trackml-particle-identification (2021), accessed: 2021-02-26
[10] *gltf*, https://www.khronos.org/gltf/ (2021), accessed: 2021-02-26
[11] R. Brun, F. Rademakers, *Root - an object oriented data analysis framework* (1997), 10.5281/zenodo.848818
[12] *dat.gui, a lightweight graphical user interface*, https://github.com/dataarts/dat.gui/tree/master (2021), accessed: 2021-02-26