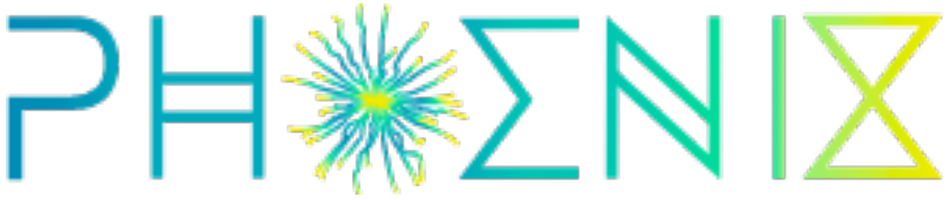
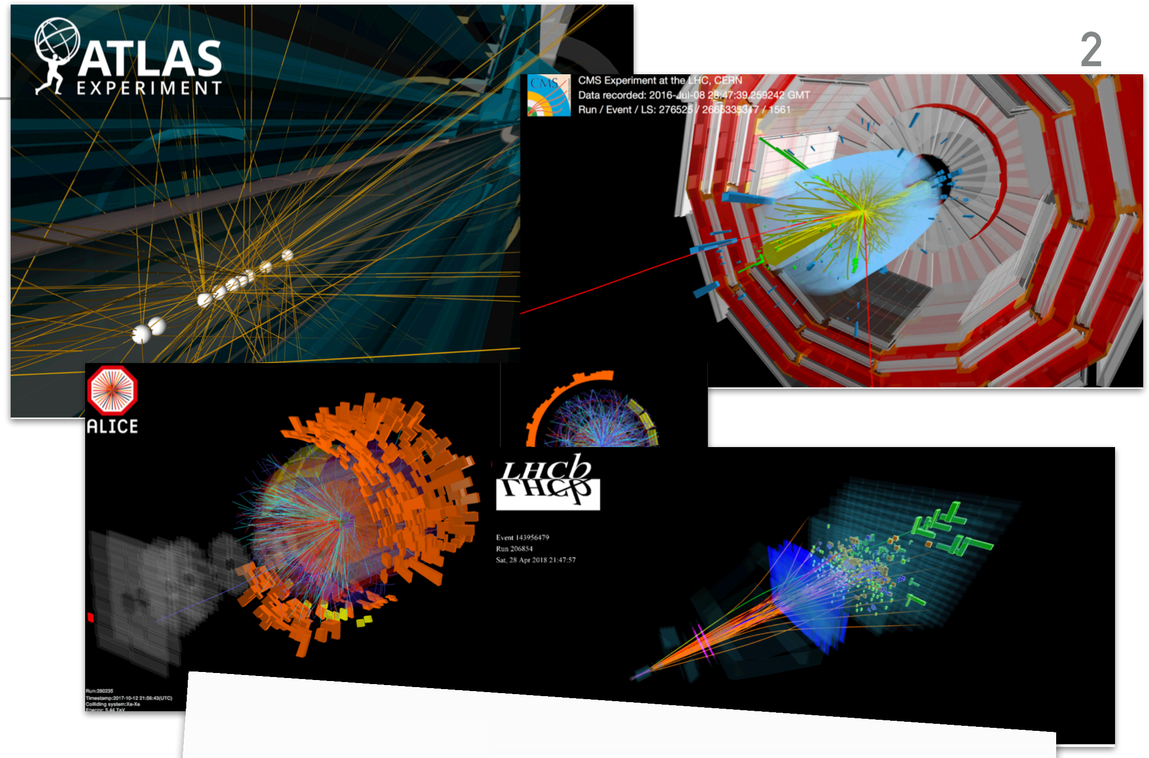


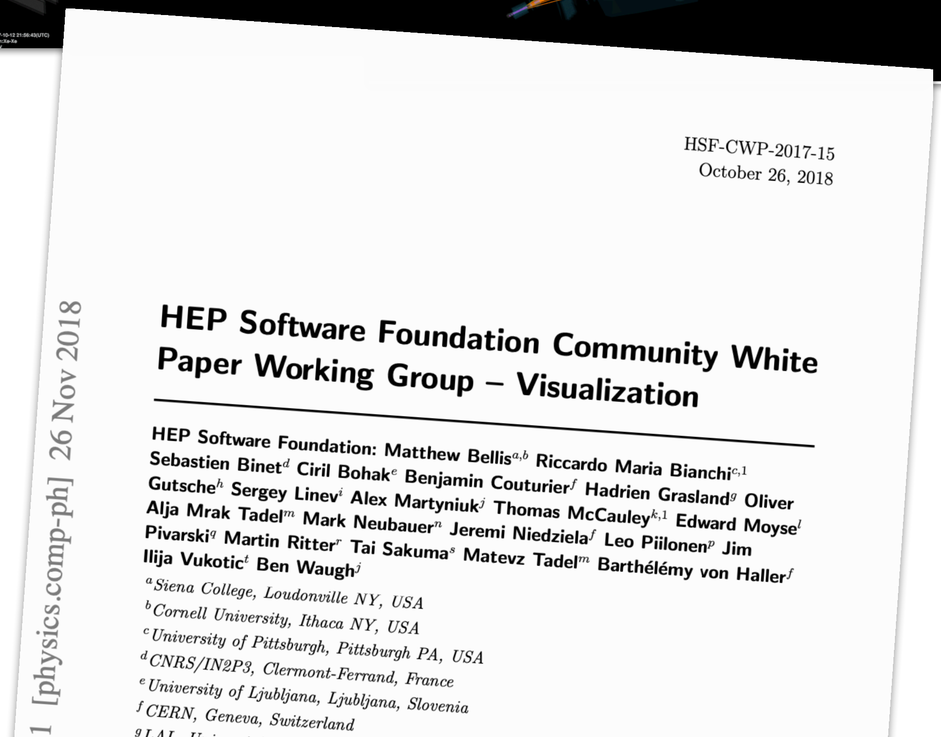
EDWARD MOYSE

PHOENIX

INTRODUCTION

- Event displays have tended to be experiment specific, requiring installation (often as part of a larger framework)
  - e.g. VP1, iSpy, ROOT EVE based displays etc
- In recent years, there has been a rise in browser based event displays,
  - inherently cross-platform
  - low entry barrier for users
  - e.g. iSpy WebGL
- In 2017 the HSF visualisation white paper identified the desirability of having a common event format, and a common tool





HSF-CWP-2017-15
October 26, 2018

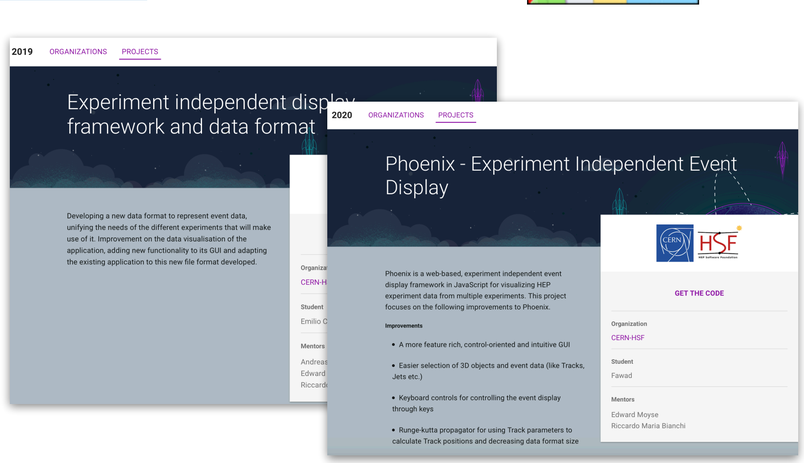
HEP Software Foundation Community White Paper Working Group – Visualization

HEP Software Foundation: Matthew Bellis[a,b] Riccardo Maria Bianchi[c,1] Sebastien Binet[d] Ciril Bohak[e] Benjamin Couturier[f] Hadrien Grasland[g] Oliver Gutsche[h] Sergey Linev[i] Alex Martyniuk[j] Thomas McCauley[k,1] Edward Moyse[l] Alja Mrak Tadel[m] Mark Neubauer[n] Jeremi Niedziela[f] Leo Piilonen[p] Jim Pivarski[q] Martin Ritter[r] Tai Sakuma[s] Matevz Tadel[m] Barthélémy von Haller[f] Ilija Vukotic[t] Ben Waugh[j]

[a] Siena College, Loudonville NY, USA
[b] Cornell University, Ithaca NY, USA
[c] University of Pittsburgh, Pittsburgh PA, USA
[d] CNRS/IN2P3, Clermont-Ferrand, France
[e] University of Ljubljana, Ljubljana, Slovenia
[f] CERN, Geneva, Switzerland

INTRODUCTION

- Phoenix is the event display of the HSF visualisation group:
  - Repository: https://github.com/HSF/phoenix
  - Demo: http://hepsoftwarefoundation.org/phoenix/
  - Runs entirely in the browser, so scalable and cheap to host
    - Uses three.js and angular, nodeJS, NPM (+ other libraries)
      - (Also a demo using reactjs)
  - Experiment agnostic and extensible by design
    - Currently has support for LHCb, ATLAS, CMS and TrackML **geometry** and **event data**
    - And very recently, we saw a demo of **FASER** using Phoenix
- Accepted for Google Summer of Code (thanks to HSF!)
  - **2019**: Emilio Cortina Labra
  - **2020**: Fawad Ali
- **2020** selected by ATLAS as its official web event display



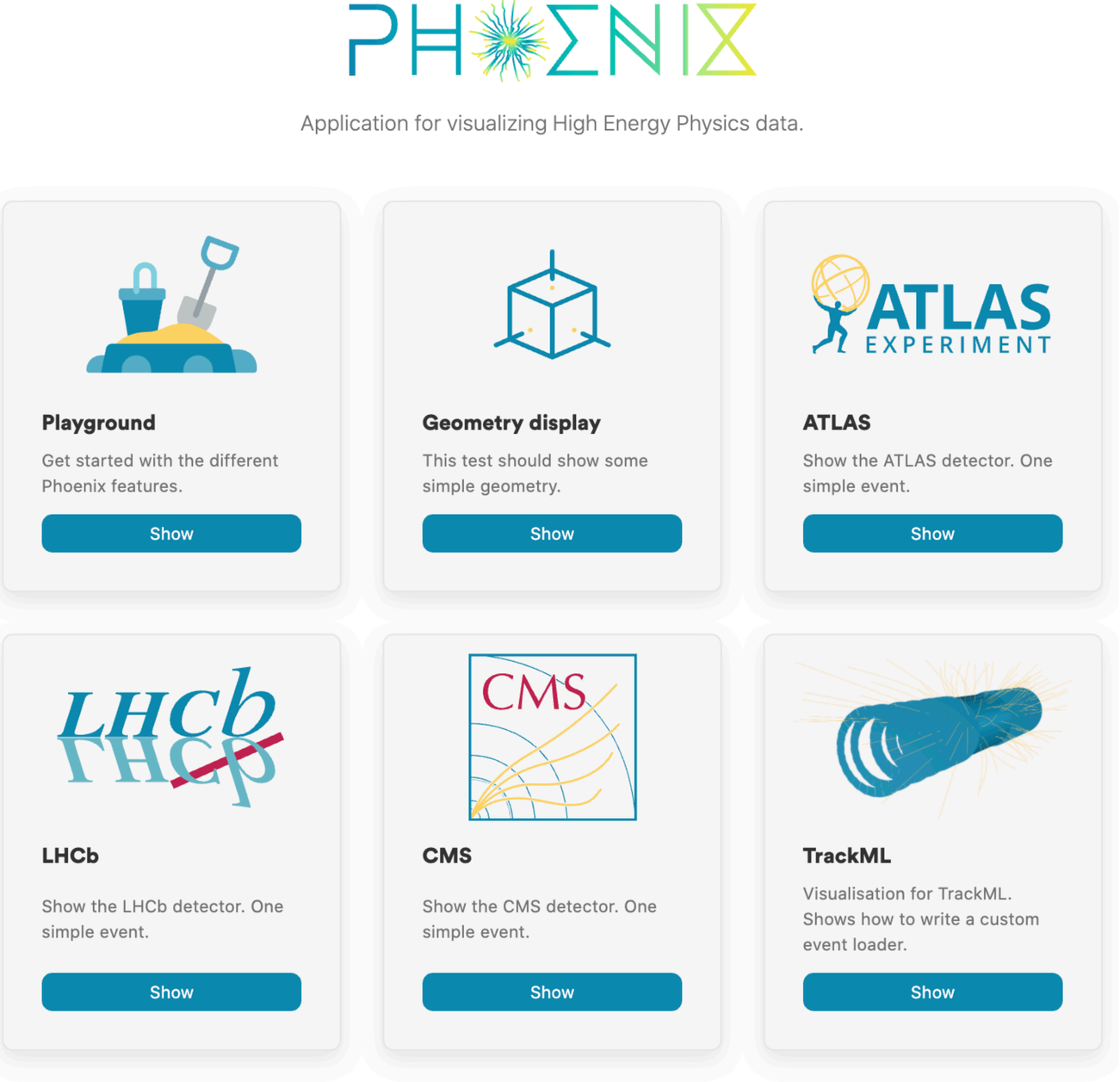
DESIGN CONCEPTS

- In order to support as many experiments as possible, some key goals:
  - **Permissive licence and open source** (Apache 2.0 Licence)
  - **Use industry standards**
  - **Simple standard format for Event Data**
  - **Good documentation**
  - **Don’t make experiment specific assumptions**
  - Make Phoenix **configurable, extendable and modular**

WALKTHROUGH

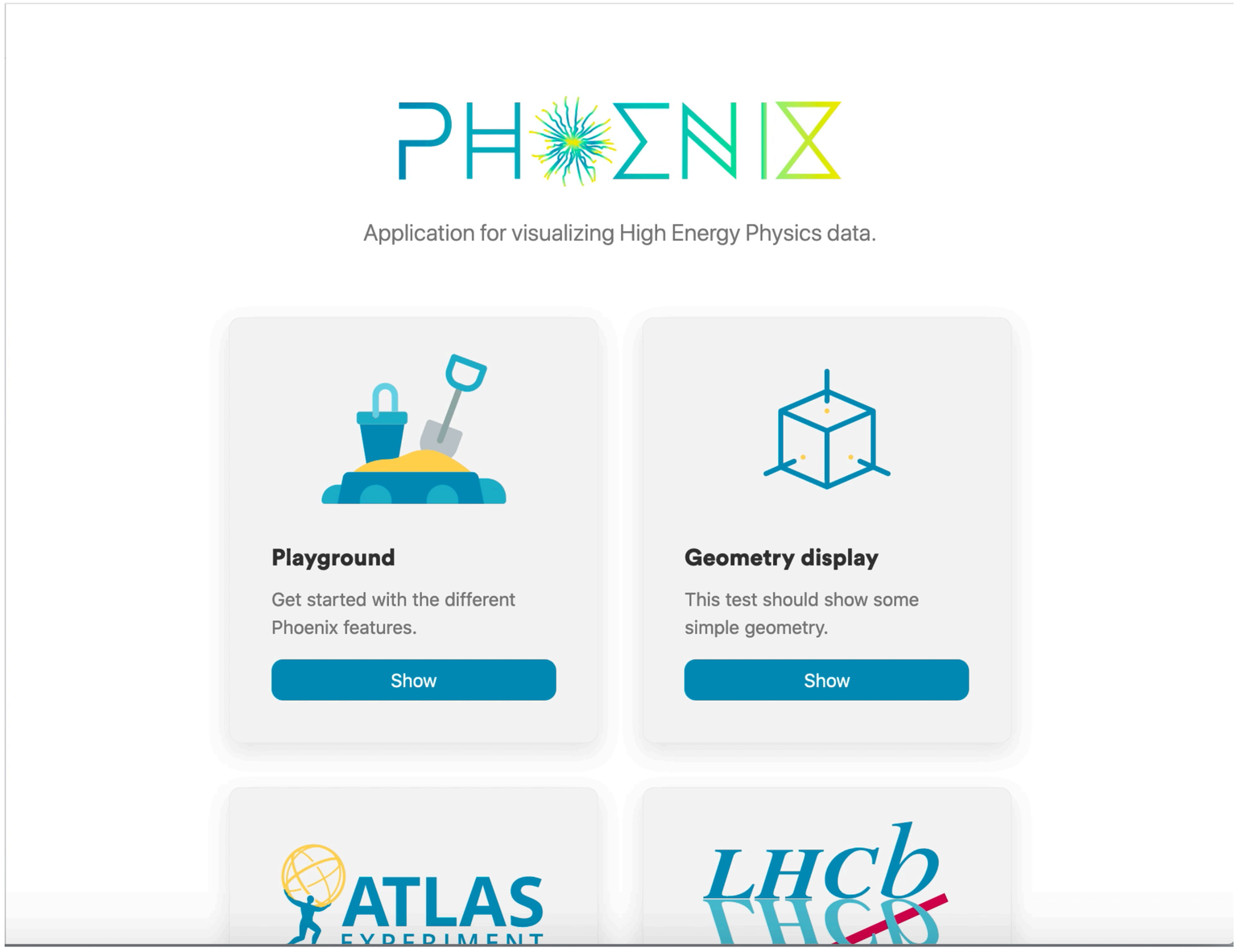
- This is the homepage, which you will see either:
  - At the online demo,
  - Or if you run locally
    - e.g. http://localhost:4200

- Let's do a very quick walkthrough some of the demos
  - Aim is to show **functionality**
  - For tutorials, please check the user manual



WALKTHROUGH 1: PLAYGROUND

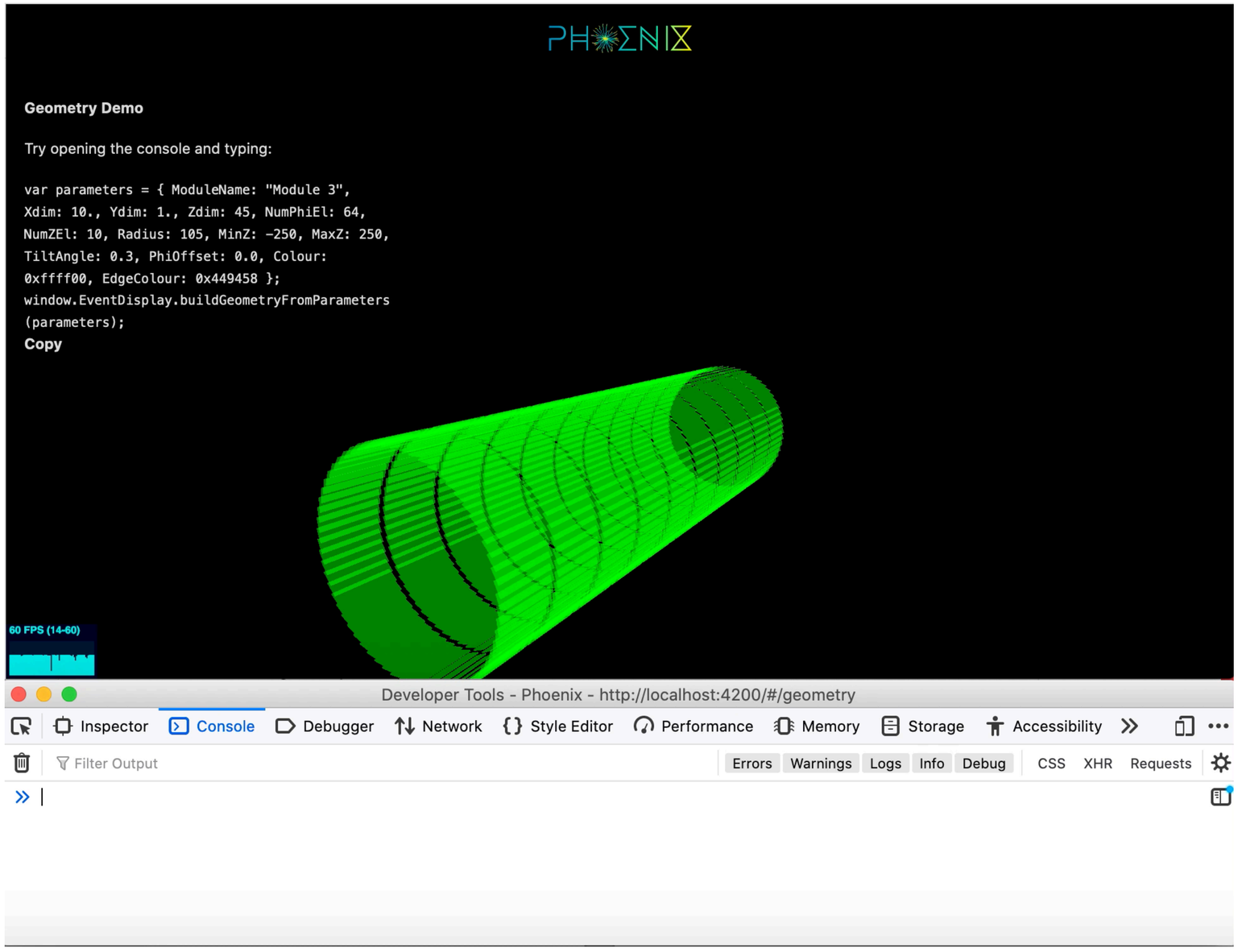
- In **Playground** [link], you can open a geometry file
  - Currently supported formats are OBJ, glTF , ROOT, json(gz)
  - However Threejs supports a huge number of 3D formats, so any of these could work too…



WALKTHROUGH 2: GEOMETRY

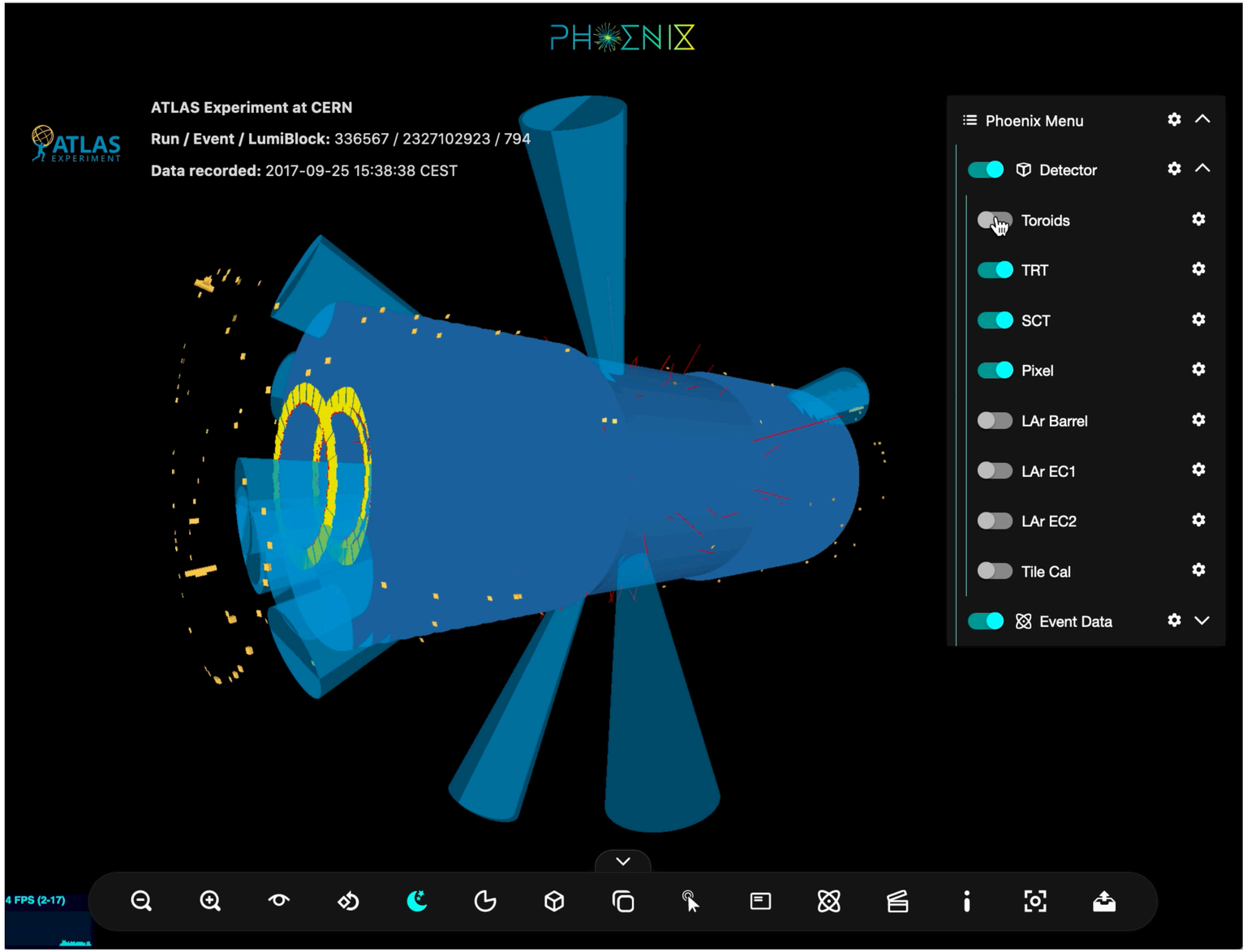
- In **Geometry** [link], you can open the javascript console in your browser and programmatically add a very simple detector
- e.g.

```
var parameters = { ModuleName: "Module 3", Xdim: 10., Ydim:
1., Zdim: 45, NumPhiEl: 64, NumZEl: 10, Radius: 75, MinZ:
-250, MaxZ: 250, TiltAngle: 0.3, PhiOffset: 0.0, Colour:
0x00ff00, EdgeColour: 0x449458 };

window.eventDisplay.buildGeometryFromParameters(parameters);
```



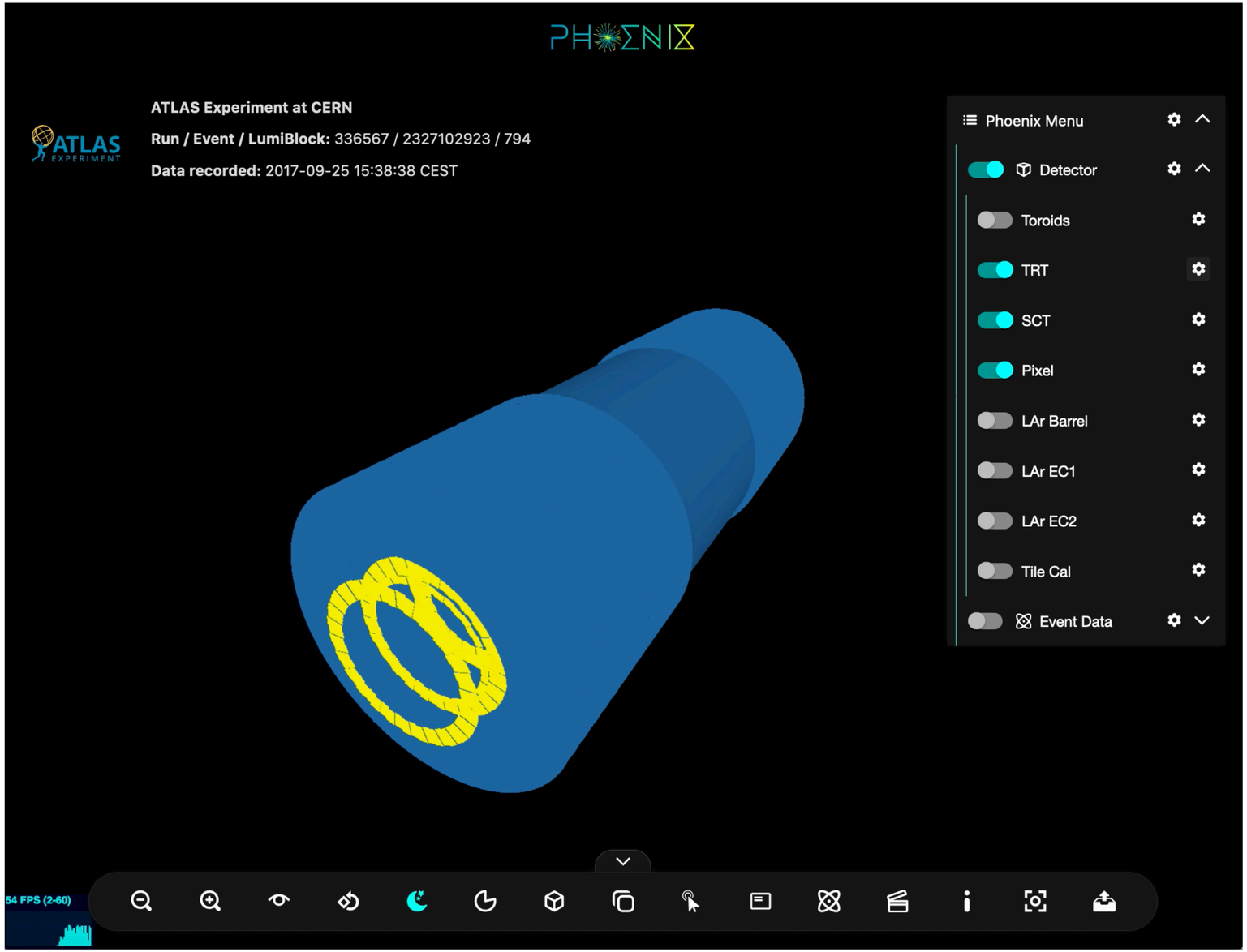
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - **Some geometry is hidden by default, but can be enabled**
  - How the geometry is rendered can be changed (and it can be sliced open)
  - Cuts can be applied to physics objects
  - Physics objects can be selected, further information displayed
  - Or you can search through a collection and find the objects this way
  - You can have an overlay view
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



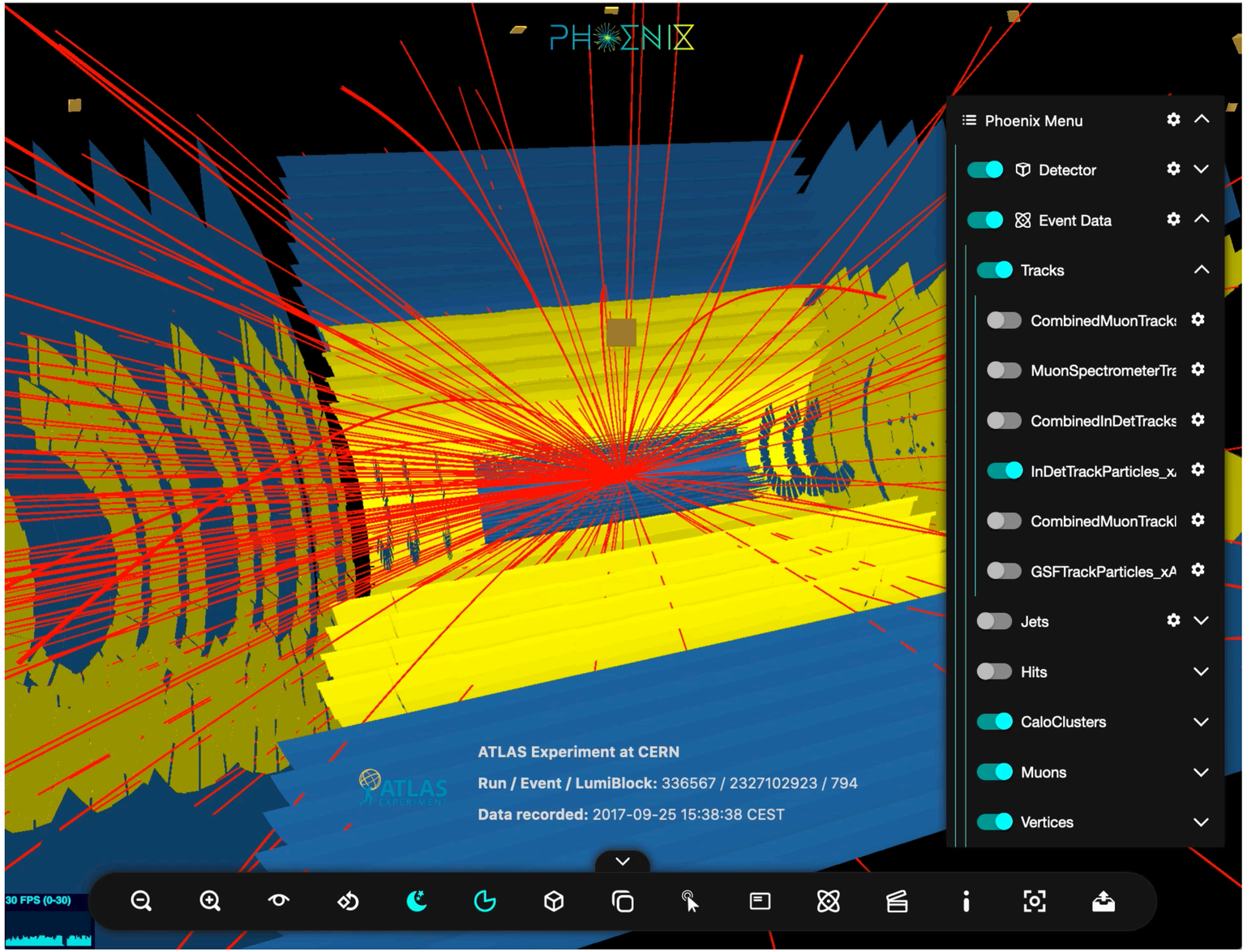
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - **How the geometry is rendered can be changed (and it can be sliced open)**
  - Cuts can be applied to physics objects
  - Physics objects can be selected, further information displayed
  - Or you can search through a collection and find the objects this way
  - You can have an overlay view
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



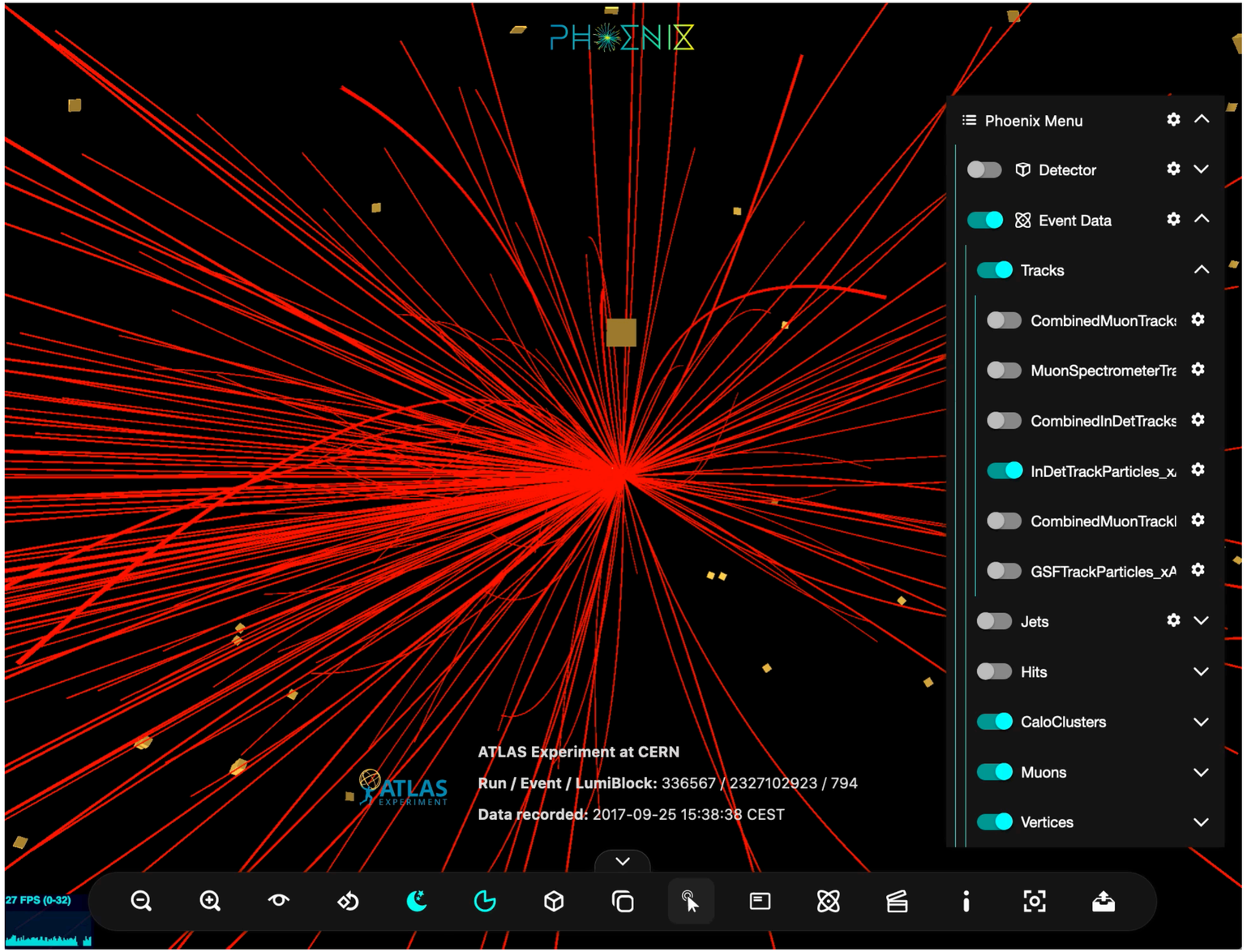
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - How the geometry is rendered can be changed (and it can be sliced open)
  - **Cuts can be applied to physics objects**
  - Physics objects can be selected, further information displayed
  - Or you can search through a collection and find the objects this way
  - You can have an overlay view
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



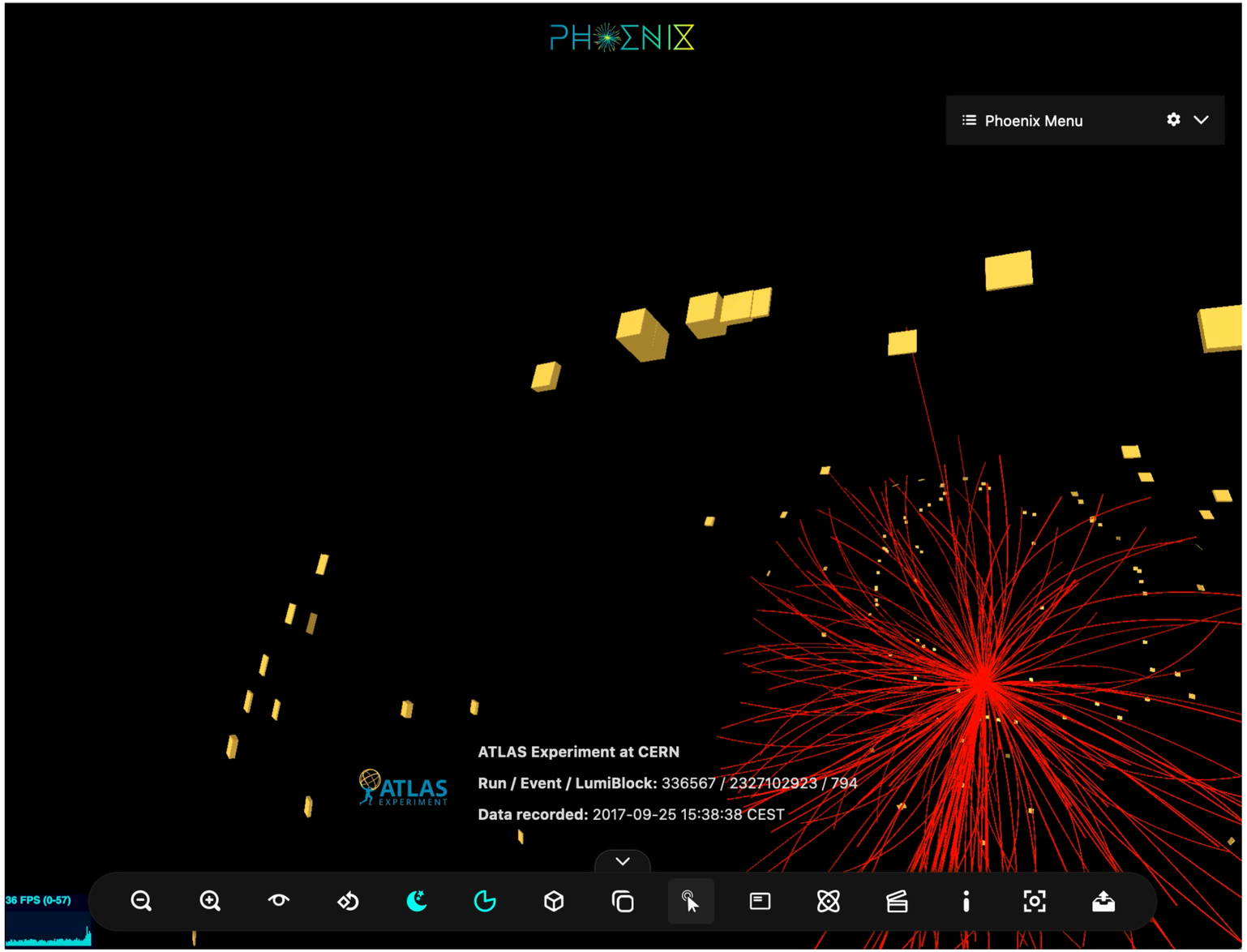
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - How the geometry is rendered can be changed (and it can be sliced open)
  - Cuts can be applied to physics objects
  - **Physics objects can be selected, further information displayed**
  - Or you can search through a collection and find the objects this way
  - You can have an overlay view
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



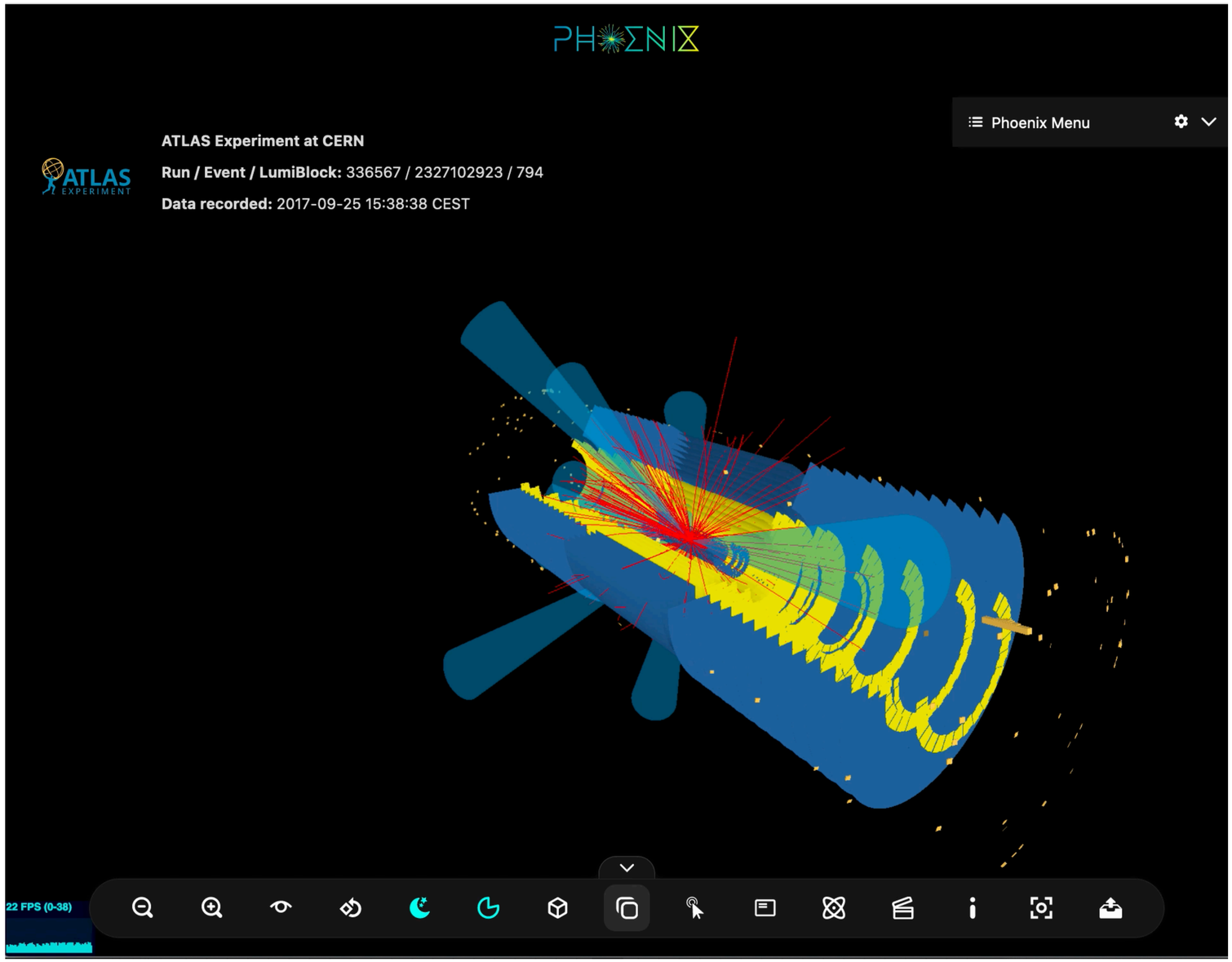
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - How the geometry is rendered can be changed (and it can be sliced open)
  - Cuts can be applied to physics objects
  - Physics objects can be selected, further information displayed
  - **Or you can search through a collection and find the objects this way**
  - You can have an overlay view
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



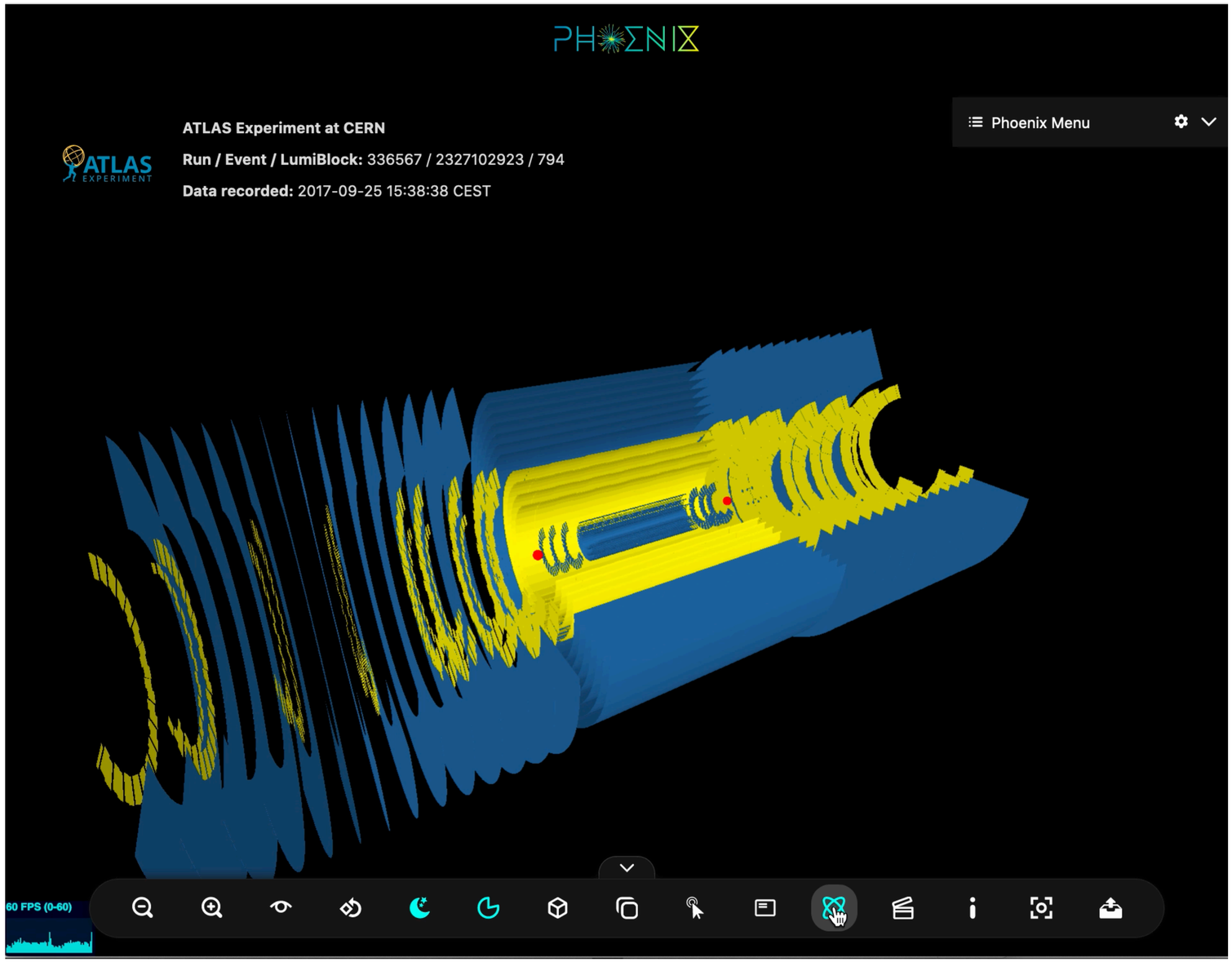
WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - How the geometry is rendered can be changed (and it can be sliced open)
  - Cuts can be applied to physics objects
  - Physics objects can be selected, further information displayed
  - Or you can search through a collection and find the objects this way
  - **You can have an overlay view**
  - And we have some builtin animations
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)



WALKTHROUGH 3: ATLAS

- In **ATLAS** [link], a default event is opened, and geometry is loaded
  - Some geometry is hidden by default, but can be enabled
  - How the geometry is rendered can be changed (and it can be sliced open)
  - Cuts can be applied to physics objects
  - Physics objects can be selected, further information displayed
  - Or you can search through a collection and find the objects this way
  - You can have an overlay view
  - **And we have some builtin animations**
- All of this configuration can be saved, loaded again, applied by default or passed in by URL (next slide)

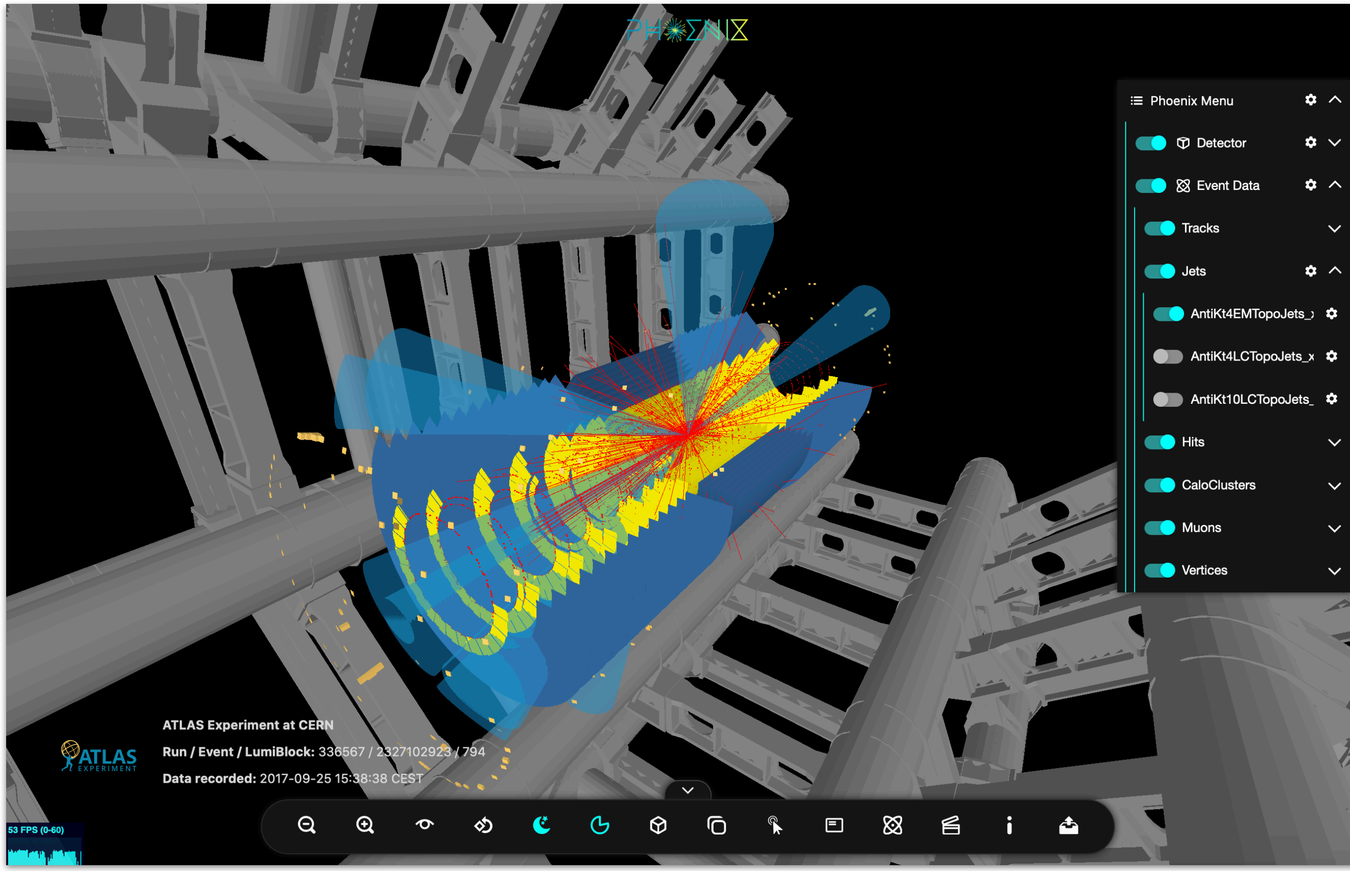


OTHER FEATURES

- One possible use-case is that analysers (or outreach) would like to give people a URL which when opened, loads a specified event and specified configuration
  - e.g highlight the **physics objects** used in this analysis and **arrange the detector** to make this as clear as possible
  - Currently this requires pre-loading the event data to e.g. an EOS folder, and then you use a URL
    - e.g. http://MYSERVER.web.cern.ch/?file=EVENTDATA&type=jivexml&config=CONFIG.json

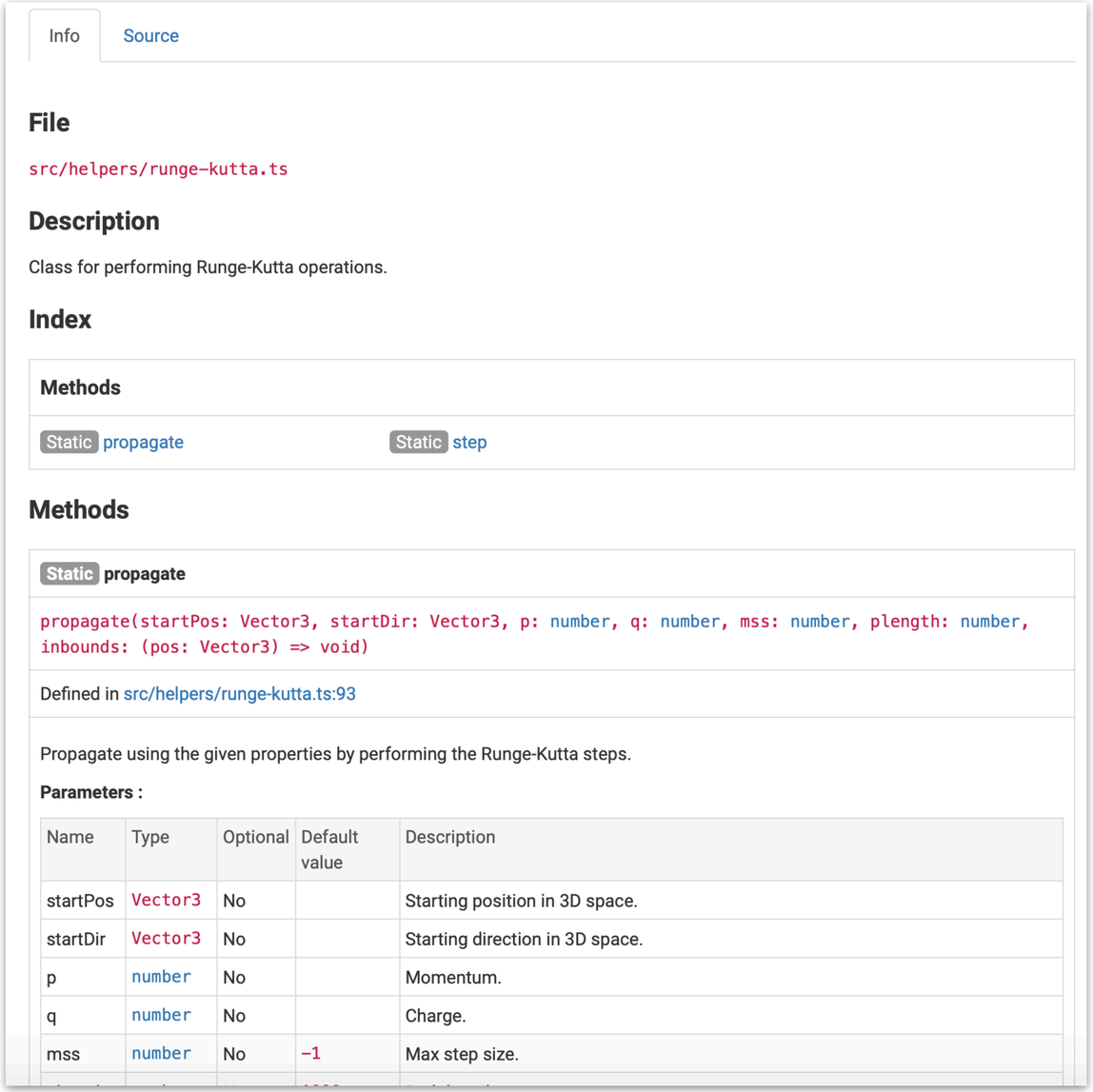
EVENT

CONFIGURATION



MENUS AND HELPERS

- Phoenix provides lots of functionality to help developers
  - e.g Phoenix has its own menu system [phoenix-ui-components]
- Phoenix also has many classes to help render physics data e.g.
  - Many experiments only store limited numbers of track parameters, so cannot draw a complete curve
  - Phoenix provides a **RungeKutta** propagator
  - You just need to supply the magnetic field!



Info Source

File

`src/helpers/runge-kutta.ts`

Description

Class for performing Runge-Kutta operations.

Index

Methods

Static propagate Static step

Methods

Static propagate

```
propagate(startPos: Vector3, startDir: Vector3, p: number, q: number, mss: number, plength: number,
inbounds: (pos: Vector3) => void)
```

Defined in src/helpers/runge-kutta.ts:93

Propagate using the given properties by performing the Runge-Kutta steps.

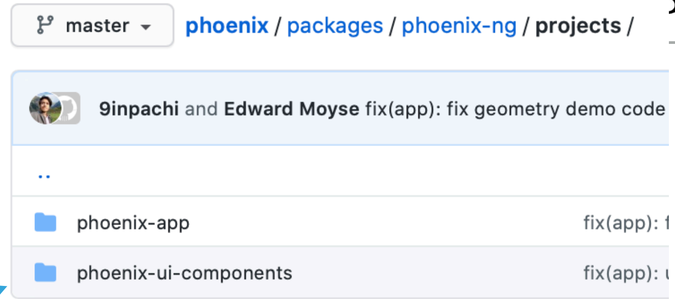
Parameters :

Name	Type	Optional	Default value	Description
startPos	Vector3	No		Starting position in 3D space.
startDir	Vector3	No		Starting direction in 3D space.
p	number	No		Momentum.
q	number	No		Charge.
mss	number	No	−1	Max step size.

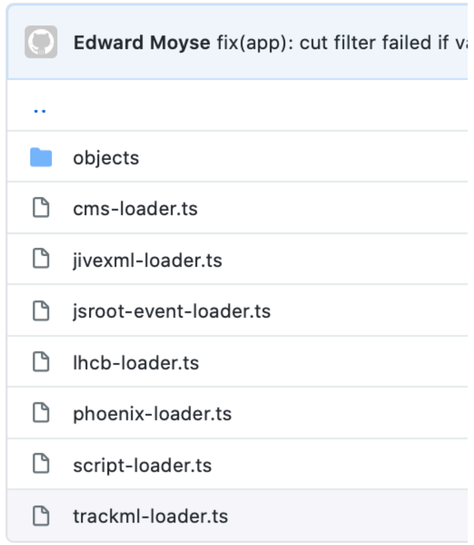
[https://hepsoftwarefoundation.org/phoenix/api-docs/classes/RungeKutta.html]

LAYOUT AND EXTENSIBILITY

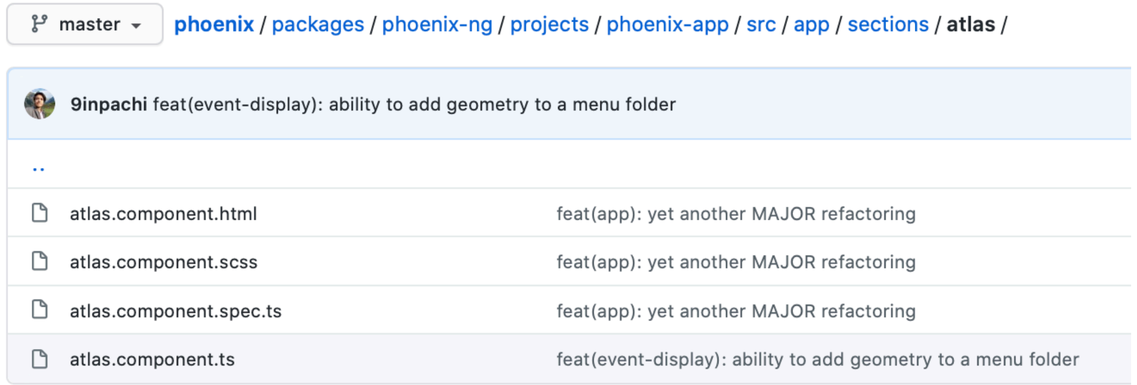
- Phoenix is divided into
  - a core library (`phoenix-event-display`)...
  - and `phoenix-ng`, which contains the UI and the demo app you have seen today
    - **N.B.** The UI is split into a separate package, so users can use a different UI if they prefer
- In order to support a new experiment, we need to add the geometry
  - As mentioned earlier, Phoenix supports many standard geometry formats
- And add a new **loader** to convert event data to phoenix's internal format
- And finally add a new section
  - See next slide!



[link]



[link]



EXTENSIBILITY

- Two important components here: the `experiment.component.html` file, which specifies what is used…

```
1  <app-nav></app-nav>
2  <app-ui-menu></app-ui-menu>
3  <app-experiment-info experiment="atlas" experimentTagline="ATLAS Experiment at CERN"></app-experiment-info>
4  <app-phoenix-menu [rootNode]="phoenixMenuRoot"></app-phoenix-menu>
5  <div id="eventDisplay"></div>
```

atlas.component.html

EXTENSIBILITY

- Two important components here: the `experiment.component.html` file, which specifies what is used…

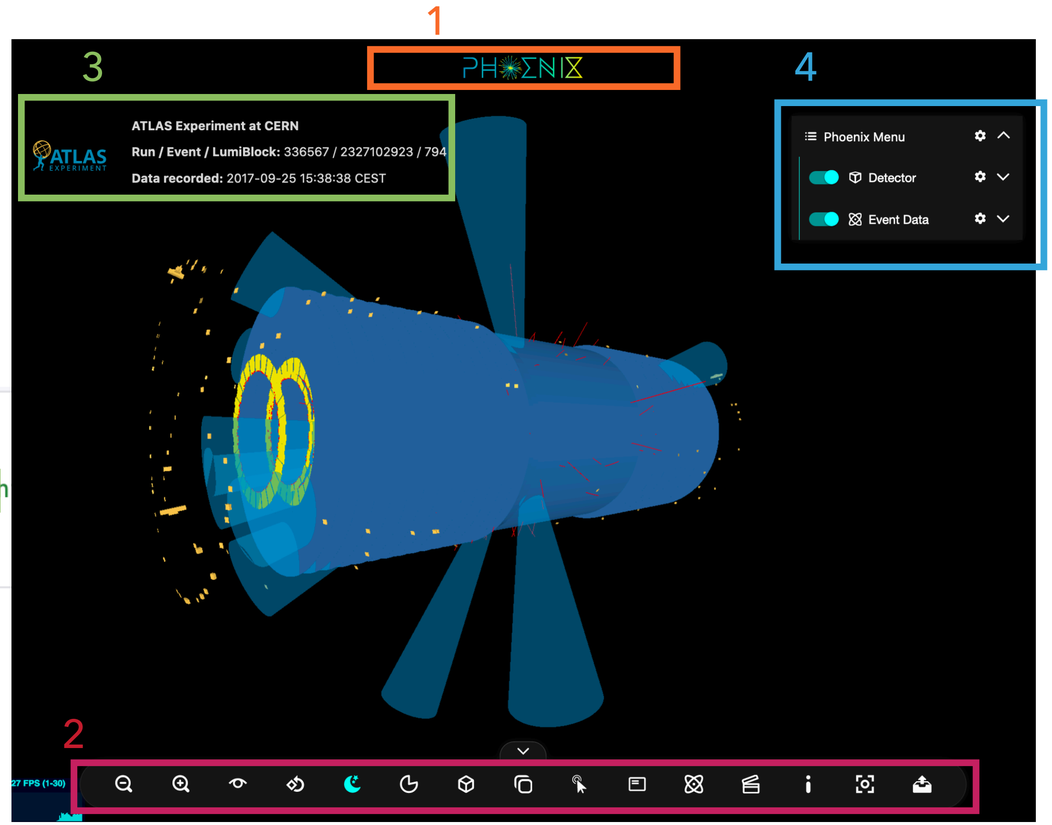
1. Link back to main Phoenix page

```
1  <app-nav></app-nav>
2  <app-ui-menu></app-ui-menu>
3  <app-experiment-info experiment="atlas" experimentTagline="ATLAS Experiment at CERN"></app-experiment-in
4  <app-phoenix-menu [rootNode]="phoenixMenuRoot"></app-phoenix-menu>
5  <div id="eventDisplay"></div>
```

2. Phoenix row menu

3. Experiment logo, link and info

4. Phoenix geometry/event data menu



atlas.component.html

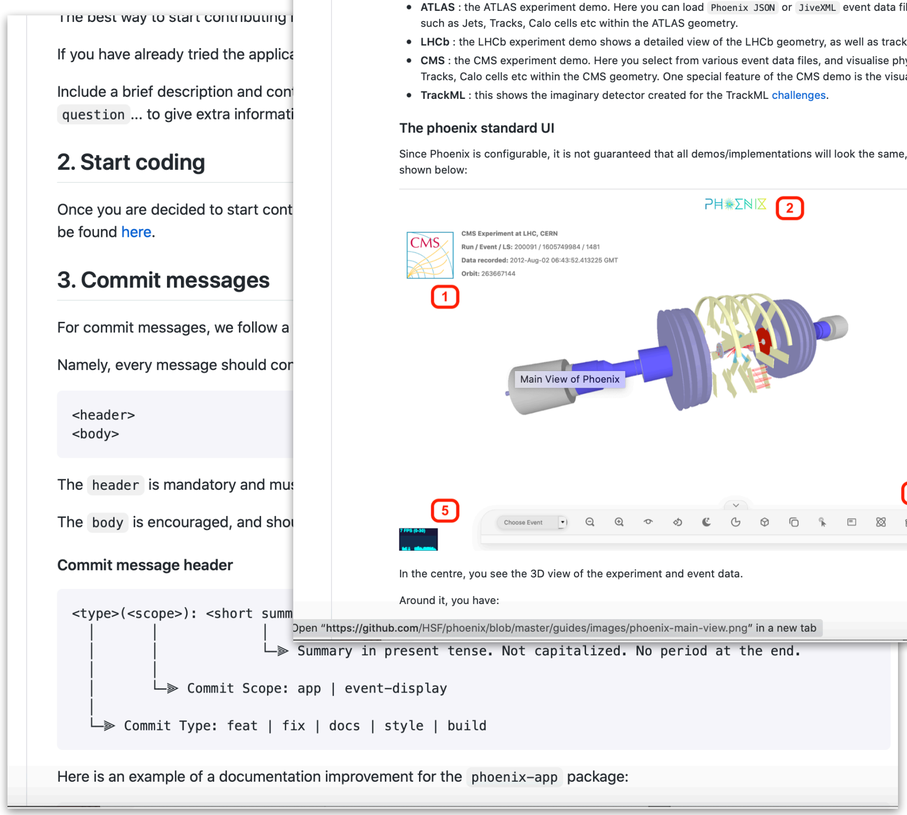
EXTENSIBILITY

- Two important components here: the `experiment.component.html` file, which specifies what is used and the experiment specific **implementation** i.e. `experiment.component.ts` file
  - Contains e.g.
    - The default configuration,
    - Default event,
    - Loaders required,
    - Geometry etc
- And that is it!

```typescript
import { Component, OnInit } from '@angular/core';
import { EventDisplayService } from 'phoenix-ui-components';
import { Configuration, PresetView, PhoenixMenuNode, PhoenixLoader } from 'phoenix-event-
import { environment } from '../../../environments/environment';
import eventConfig from '../../../../event-config.json';

@Component({
  selector: 'app-atlas',
  templateUrl: './atlas.component.html',
  styleUrls: ['./atlas.component.scss']
})
export class AtlasComponent implements OnInit {
  phoenixMenuRoot = new PhoenixMenuNode('Phoenix Menu', 'phoenix-menu');

  constructor(private eventDisplay: EventDisplayService) { }

  ngOnInit() {
    let defaultEvent: { eventFile: string, eventType: string };
    // Get default event from configuration
    if (environment?.singleEvent) {
      defaultEvent = eventConfig;
    } else {
      defaultEvent = {
        eventFile: 'assets/files/JiveXML/JiveXML_336567_2327102923.xml',
        eventType: 'jivexml'
      }
    }

    // Define the configuration
    const configuration: Configuration = {
      eventDataLoader: new PhoenixLoader(),
      presetViews: [
        new PresetView('Left View', [0, 0, -12000], 'left-cube'),
        new PresetView('Center View', [-500, 12000, 0], 'top-cube'),
        new PresetView('Right View', [0, 0, 12000], 'right-cube')
      ],
      defaultView: [4000, 4000, 4000],
      // Set the phoenix menu to be used (defined above)
      phoenixMenuRoot: this.phoenixMenuRoot,
      // Default event data to fallback to if none given in URL
      // Do not set if there should be no event loaded by default
      defaultEventFile: defaultEvent
```

DOCUMENTATION

- How do you learn more?
  - Phoenix has detailed developer and user guides, as well as API docs



The best way to start contributing

If you have already tried the applica

Include a brief description and cont
`question` ... to give extra informati

2. Start coding

Once you are decided to start cont
be found here.

3. Commit messages

For commit messages, we follow a

Namely, every message should cor

```
<header>
<body>
```

The `header` is mandatory and mus

The `body` is encouraged, and shou

Commit message header

```
<type>(<scope>): <short summ
  |       |             |
  |       |             └─⫸ Summary in present tense. Not capitalized. No period at the end.
  |       |
  |       └─⫸ Commit Scope: app | event-display
  |
  └─⫸ Commit Type: feat | fix | docs | style | build
```

Here is an example of a documentation improvement for the `phoenix-app` package:

CONTRIBUTING.md

The demo grid

When you first open the Phoenix demo (see the developer instructions for how to check it out and run locally) you will see a grid of Phoenix demos:

- **Playground** : a blank canvas where you can load 3D objects, move them around and generally experiment with Phoenix
- **Geometry display** : a simple demo of generating geometry procedurally/programmatically with Phoenix
- **ATLAS** : the ATLAS experiment demo. Here you can load `Phoenix JSON` or `JiveXML` event data files, and visualise physics objects such as Jets, Tracks, Calo cells etc within the ATLAS geometry.
- **LHCb** : the LHCb experiment demo shows a detailed view of the LHCb geometry, as well as tracks passing through it.
- **CMS** : the CMS experiment demo. Here you select from various event data files, and visualise physics objects such as Jets, Tracks, Calo cells etc within the CMS geometry. One special feature of the CMS demo is the visualisation of Muon Chambers.
- **TrackML** : this shows the imaginary detector created for the TrackML challenges.

The phoenix standard UI

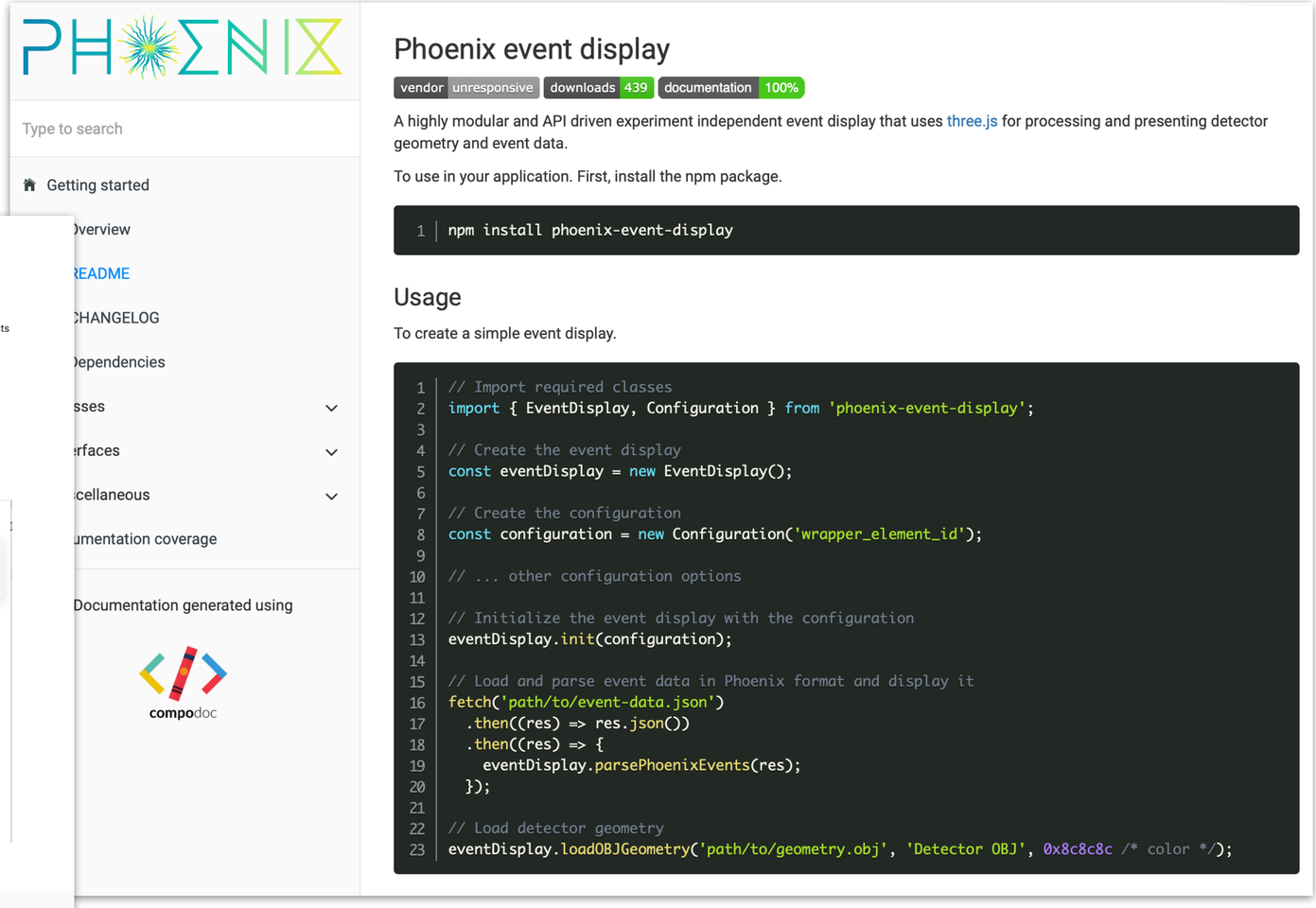
Since Phoenix is configurable, it is not guaranteed that all demos/implementations will look the same, but a typical Phoenix view is shown below:

In the centre, you see the 3D view of the experiment and event data.

Around it, you have:

Open "https://github.com/HSF/phoenix/blob/master/guides/images/phoenix-main-view.png" in a new tab

users.md



PHOENIX

Type to search

Getting started

Overview

README

CHANGELOG

Dependencies

sses

rfaces

cellaneous

umentation coverage

Documentation generated using

compodoc

Phoenix event display

vendor unresponsive | downloads 439 | documentation 100%

A highly modular and API driven experiment independent event display that uses three.js for processing and presenting detector geometry and event data.

To use in your application. First, install the npm package.

```
npm install phoenix-event-display
```

Usage

To create a simple event display.

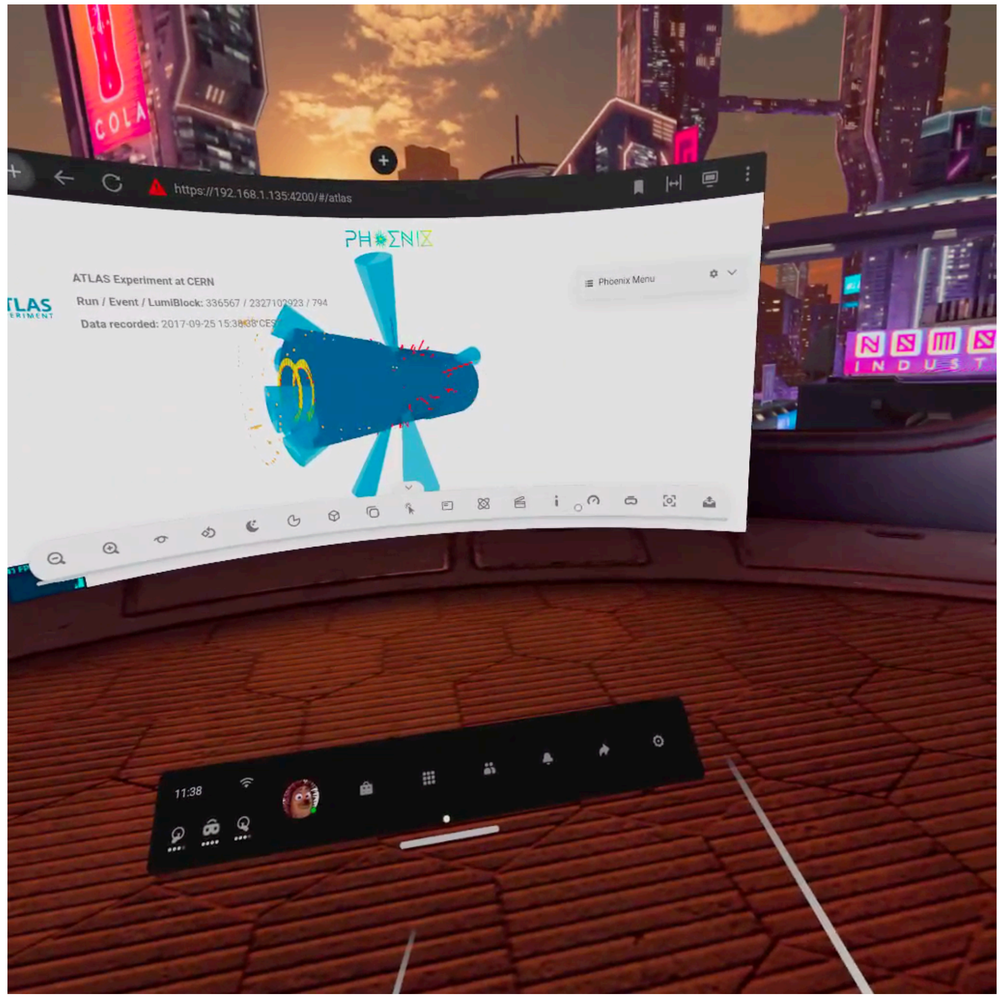
```
// Import required classes
import { EventDisplay, Configuration } from 'phoenix-event-display';

// Create the event display
const eventDisplay = new EventDisplay();

// Create the configuration
const configuration = new Configuration('wrapper_element_id');

// ... other configuration options

// Initialize the event display with the configuration
eventDisplay.init(configuration);

// Load and parse event data in Phoenix format and display it
fetch('path/to/event-data.json')
  .then((res) => res.json())
  .then((res) => {
    eventDisplay.parsePhoenixEvents(res);
  });

// Load detector geometry
eventDisplay.loadOBJGeometry('path/to/geometry.obj', 'Detector OBJ', 0x8c8c8c /* color */);
```

https://hepsoftwarefoundation.org/phoenix/api-docs/

FINAL THOUGHTS

- Very brief overview of Phoenix
  - Thanks to everyone who has contributed to the Project (especially our GSOC students, Fawad and Emilio)
- If you are interested in using Phoenix, or contributing, please contact us:
  - On our **Gitter** channel: [link]
  - Or via **github** issues: [link]
- Latest release
  - v1.1.0 release notes

AND A TEASER…



https://github.com/HSF/phoenix/blob/master/guides/users.md#vr-mode