PROCEEDINGS OF SCIENCE

Wilson - A Framework for Creating Animated 3D Visualization in Particle Physics

Kerscher Tobias[a,*] **and Haack Christian**[a]

[a] *ECP TU Munich,*
James-Franck-Straße 1, Garching bei München, Germany

E-mail: tobias.kerscher@tum.de, christian.haack@tum.de

Visualization is a powerful tool to get a deep insight into data at a glance rather than the challenging view presented by raw numbers. In (particle-) physics, a key visualization concept is that of an event display, which typically shows 2D or 3D animations of detector readout and reconstructed quantities as function of time. For this reason a lot of different software has been designed often tailored to specific problems or experiments while showing quite a big overlap in functionality and thus a large time is spent on solving the same tasks all over again. We introduce Wilson as a framework covering these common components reducing the task of creating 3D animations to a few lines of domain specific code. Through its experiment- and even physics agnostic data model it can be easily utilized in a broad field inside and outside of physics. Furthermore, a web based viewer is included making it unnecessary to install yet another new program. This simplifies the sharing of new data, thus allowing it to spread more quickly even outside its original field.

38th International Cosmic Ray Conference (ICRC2023)
26 July - 3 August, 2023
Nagoya, Japan

*Speaker

PoS(ICRC2023)1617

1. Introduction

Modern particle physics experiments are notorious for generating data in such vast amounts it is no longer possible for humans to process it raw. They can be made consumable again by processing them via visualization producing images and animation that not only enables the experts to get an idea of the data quickly but also allows them to share them with the public. A wide range of software has been developed for this purpose often specifically tailored to a single experiment: The CMS experiment at the LHC [1], the Pierre Auger Collaboration [2] or the IceCube Experiment [3] to give a few examples.

While the experiments themselves go in very different directions, their visualizations are at least similar and share a lot in functionality: They create and often animate geometrical shapes and their color using some form of data. Much of the software's logic is spent on this, even if a rendering engine is used. We introduce Wilson to abstract away this part, leaving only the domain-specific logic on how to represent the data using higher-level building blocks. This does not only vastly improve the speed and ease of implementing new visualizations leaving more time for the actual experiment, but also invites less skilled programmers to create their own visualization.

In addition, a common framework creates a base for synergy to emerge. Improvements to it benefit all of its users, be it general performance improvements or new building blocks. Even software that is not actively developed anymore can profit.

2. Design Goals

For Wilson to not just be yet another framework, we defined design goals to make it more widely applicable:

- **Experiment Agnostic**
  To be widely applicable, Wilson cannot rely on or insist on experiment-specific properties. Its only goal is to visualize data while leaving the specifics to the user by condensing all scenarios to their most minimal intersection, data.

- **Extended Compatibility**
  The file format is designed to be easily alterable while remaining fully forward and backward-compatible, i.e. new versions can read old data and vice versa. Although older versions most likely will not be able to correctly show new features it should still at least be able to show the data itself.

- **Platform Independent**
  To be easily integrable in already existing software stacks, Wilson cannot rely on specific platforms, e.g. OS, programming language, or installed software.

- **Self Documenting**
  Since the file format is allowed to change, its documentation must not be outdated. This is ensured by making the documentation an intrinsic part of the file format.

3. Framework Architecture

In order to make it more future-proof, Wilson as a framework is split into loosely coupled components, which can be easily altered, replaced, or even removed while remaining compatible between such variations. These components will be introduced in the following.

3.1 File Format

A high-level description of the file format is used to generate code for various programming languages via Google's protocol buffers [5] making it both platform independent as well as self-documenting, but also guarantees the *documentation* to always be up-to-date. Extended compatibility is an intrinsic property of protocol buffers and thus under minor restrictions [1] automatically satisfied.

The file format itself consists of three parts: It starts with meta information including e.g. a timestamp and description, followed by data organized as tables mapping discrete time points to either scalars or points in space. They can further specify an interpolation mode to create smooth animation. It finishes with a list of *animatibles*, primitives that can be used to create the visualization by assigning their properties like size, position, or color with references to tables. Finally, a collection of such files is bundled and compressed into a zip archive.

Introducing yet another file format seems to contradict the goal of making things easier. However, its purpose is orthogonal to the ones used in experiments to save their data.

3.2 Python Library

Since Python is among the most popular programming languages in use today, a package for creating animation in the previously described format is provided. Its main design goal was the ease of use and thus provides compatibility with numpy. A valid file can be created in a few lines of Python, as shown in listing 1. The Python package also provides a local server for the web viewer with notebook integration to speed up experimentation with different visualizations.

Once again, this does not mean Wilson is limited to Python. It's rather an aid to ease its implementation into existing projects.

3.3 Web Viewer

Wilson provides a web app allowing one to view the data visualization as easily as clicking on a link. This not only makes it possible to run on a wide range of platforms with no need to install special software but also further increases the potential reach of research in the public.

The 3D visualization is built on top of the Babylon.js rendering engine [6], which utilizes the local GPU making it possible to draw large experimental setups. Since it's a full-fledged engine it can also render arbitrary 3D models, e.g. the geometry of the setup to make the visualization more expressive.

The app is static, i.e. runs no code on the server side, and can thus easily be installed by copying its files onto a server.

[1] IDs assigned to data fields must be unique and cannot be reused

```python
import numpy as np
import wilson

t = np.linspace(0, 2*np.pi)
x = 5.0 * np.cos(t)
y = 5.0 * np.sin(t)
z = np.zeros_like(t)

path = np.column_stack((t, x, y, z))
orbit = wilson.Path(path, name='Moon Orbit')

earth = wilson.Sphere('Earth', radius=1.0, color='blue')
moon = wilson.Sphere('Moon', radius=0.27, color='grey')
moon.position = orbit

project = wilson.Project('Earth Moon',
    author='wilson',
    description='The Moon orbiting around the Earth.',
    animationSpeed=2*np.pi / 5.0,
    animatables=[earth, moon])

wilson.saveProject(project, 'earth_moon.wlsn')
```

Listing 1: Example for creating an animation in Python

4. Show Case and Comparison

In the following, we give a brief overview of the available building blocks for visualizing events as well as the user interface of the viewer app. Finally, a small comparison to an already existing software is shown.

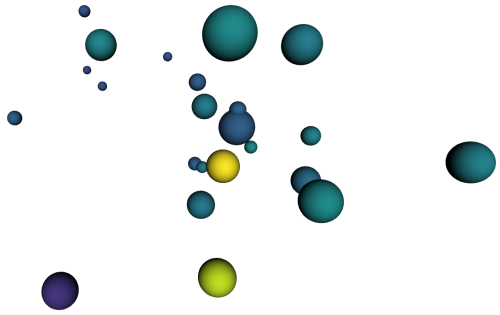
4.1 Animiatables

Figure 1 shows the currently available building blocks for visualizing events using Wilson. All of them are fully animatable including the text in overlays and description boxes by referencing graphs in template strings. The file format is open for the addition of new animatables at any time while preserving backward and forward compatibility.

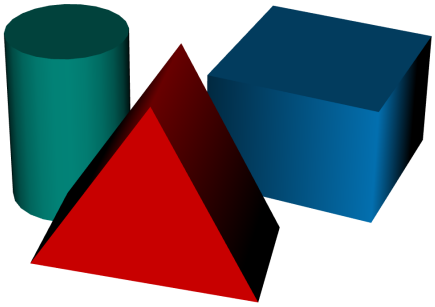
4.2 Event Viewer

The event viewer as shown in Figure 2 consists of two main parts, namely the 3D View and the Graphs View.

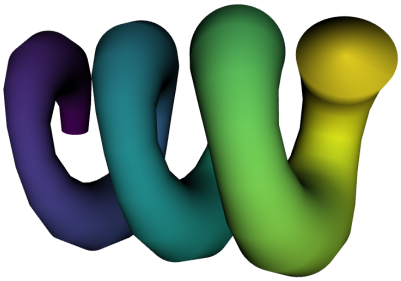
In the 3D view, one can not only inspect but also interact with the event, e.g. freely moving the camera or getting more information by clicking on objects. The latter are grouped and accessible from the right side, allowing one to search for groups and individual objects as well as toggle their visibility. If desired, the colors can be dynamically fetched from a color map whose range is adjustable from within the viewer. Below the scene is the control for animation: It can be paused and played at various speeds as well as skipping to specific time marks.



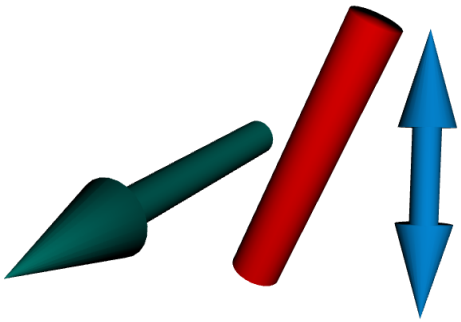
(a) Spheres with independent radius and color



(b) Prisms with independent shape, size and color



(c) Tube with independent path, radius and color



(d) Lines with independent color, width, start- and endpoint

Figure 1: List of animatables currently available in Wilson.

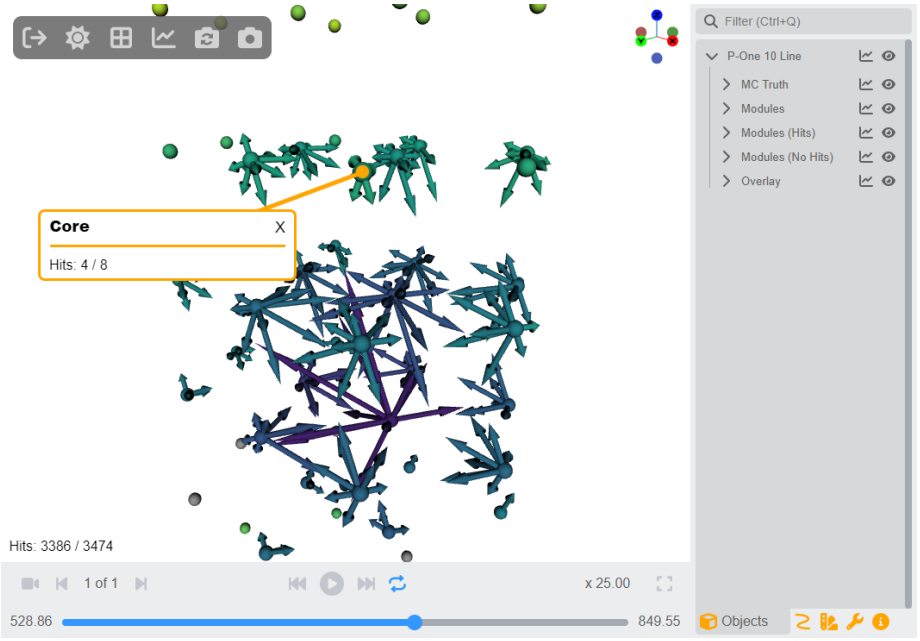
By either clicking on the Graphs button from the toolbox or object explorer, the Graphs view can be opened in a new browser tab. In the latter case, the graphs are prefiltered to match the name of the object. It is built on top of Plotly [4] and inherits its capability to draw a virtual arbitrary amount of graphs as well as toggle between linear and logarithmic scales.

4.3 Comparison with Existing Software

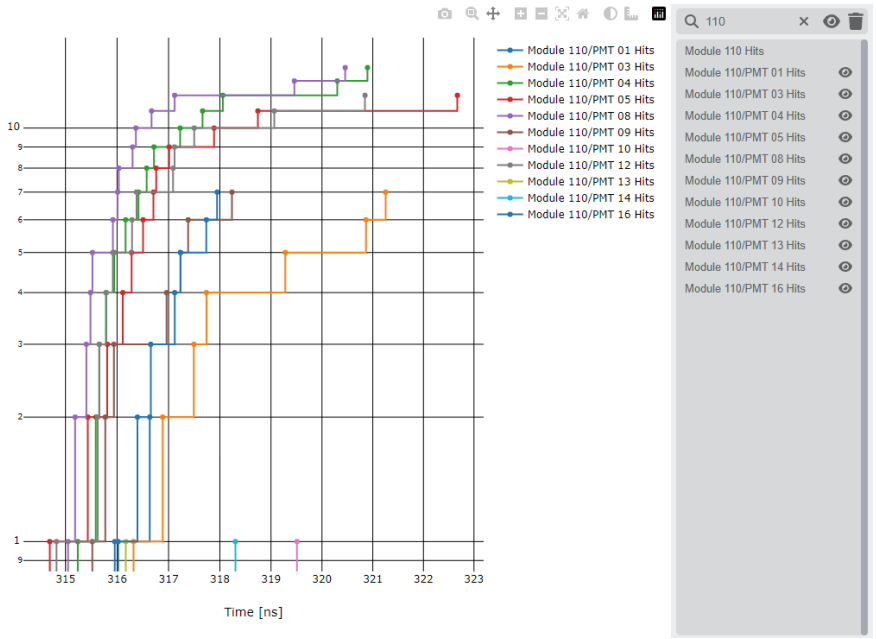
For a comparison we chose the Open Data from the Pierre Auger Collaboration [2], as it both provides an easy-to-process file format as well as its own publicly available event viewer. To recreate a presentation of a collection of events less than 100 lines of code in Python were necessary. Figure 3 shows the same event using the public viewer of the Pierre Auger Collaboration as well as Wilson.

5. Conclusion

We have shown that using Wilson and a few lines of Python Code one can (re)create a visualization of similar quality to already existing solutions. This approach can save a substantial amount of time used on developing new visualization. Plus, through its web-based viewer, it might allow experiments to generate more public outreach.

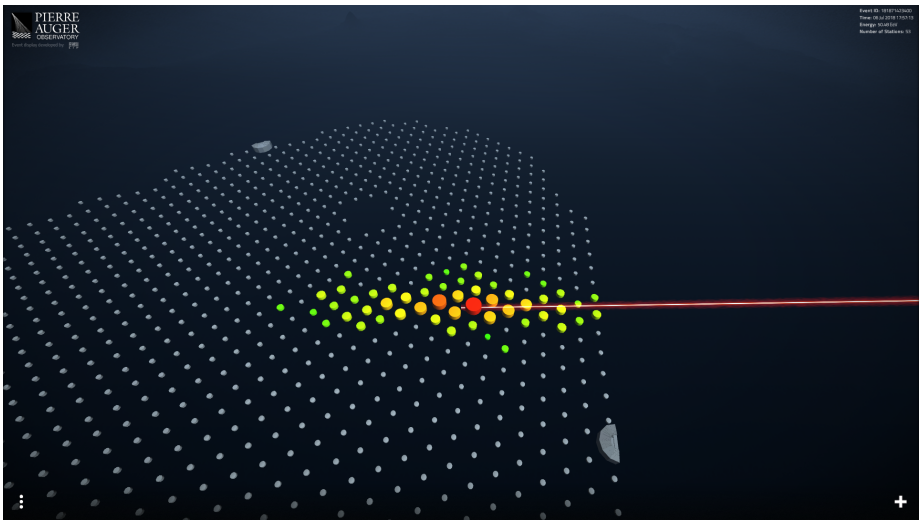


(a) 3D View

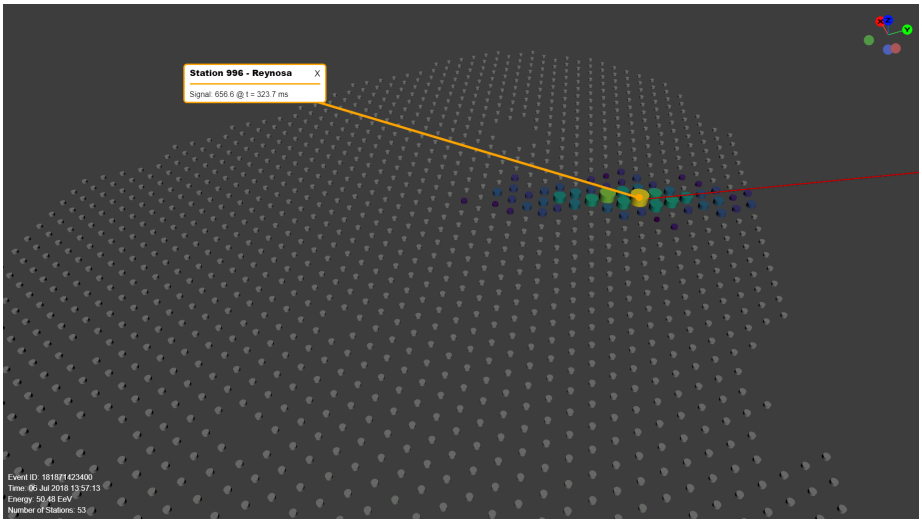


(b) Graphs View

Figure 2: Showcase of the event viewer app



(a) Pierre Auger Collaboration



(b) Wilson

Figure 3: Comparison of the visual presentation of an event of the Pierre Auger Open Data using their own viewer and Wilson.

References

[1] T. McCauly, *A browser-based event display for the CMS Experiment at the LHC using WebGL*, *J. Phys.: Conf. Ser.* **898** 072030

[2] V. Scherini on behalf of the Pierre Auger Collaboration, *The 2021 Open-Data release by the Pierre Auger Collaboration*, *PoS: ICRC2021*; p. 1386

[3] R. Tredinnick and J. Madsen, *CAVE Visualization of the IceCube Neutrino Detector*, *Proceedings - IEEE Virtual Reality*

[4] Plotly Technologies Inc., Collaborative data science. Montréal, QC, 2015. https://plot.ly.

[5] Google LLC, Protocol Buffers, 2023. https://protobuf.dev/.

[6] D. Catuhe, Babylon.js, 2023. https://www.babylonjs.com/.