RenderCore – a new WebGPU-based rendering engine for ROOT-EVE

Ciril Bohak[1,2], Dmytro Kovalskyi[3], Sergey Linev[4], Alja Mrak Tadel[5], Sebastien Strban[1], *Matevz Tadel*[5], Avi Yagil[5]

1. University of Ljubljana, Faculty of Computer and Information Science
2. King Abdullah University of Science and Technology, Visual Computing Center
3. MIT
4. GSI
5. UCSD

CHEP 2023

Overview

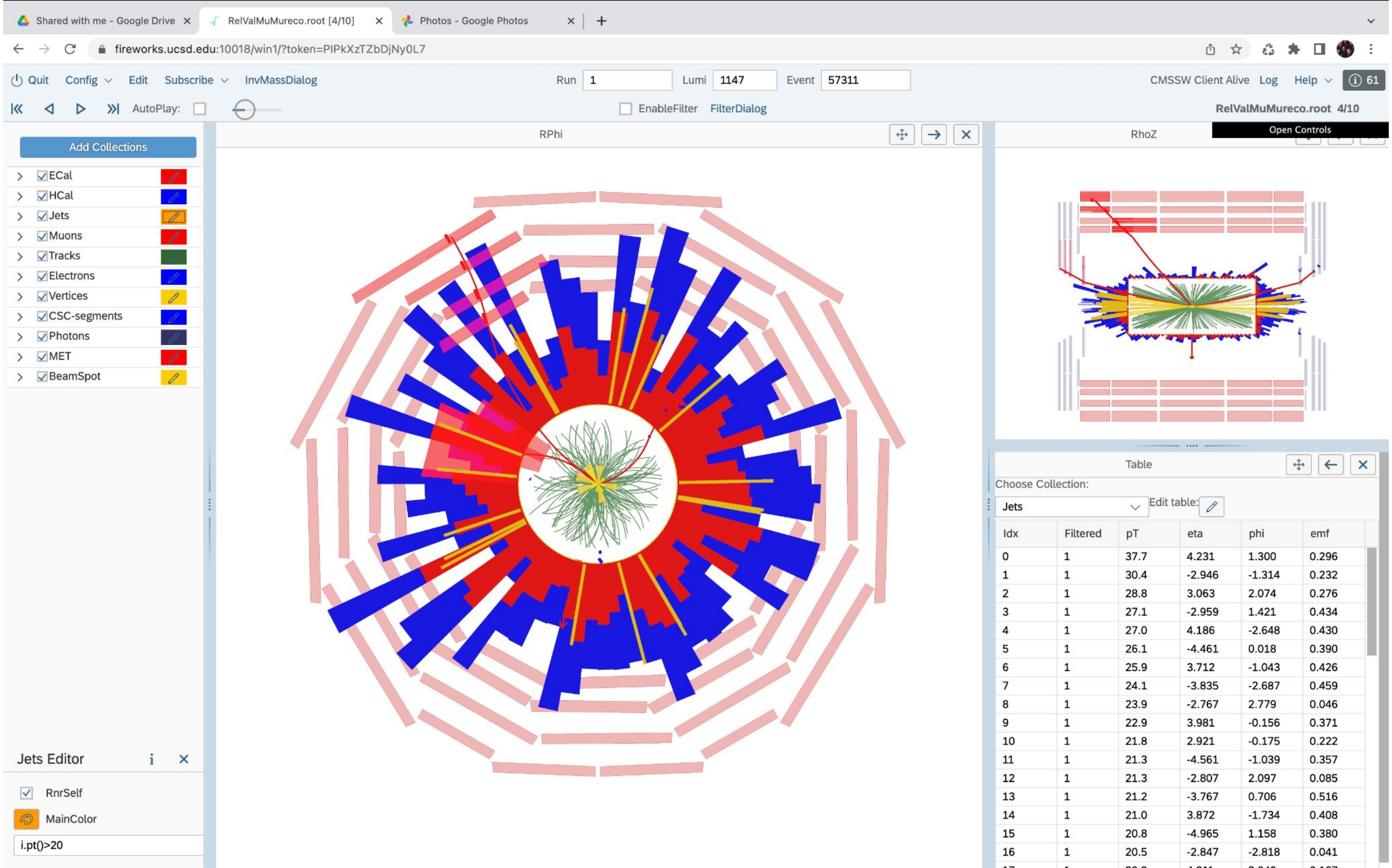
- One-slide history:
  - ROOT Event visualization: TEve → REve
  - RenderCore & REve
- REve vs. THREE.js vs. RenderCore
  - motivation (issues with THREE.js)
  - transition
  - implementation highlights
- Near future: RenderCore with WebGPU
- Conclusion

Story of REve and RenderCore I.

- ROOT-EVE – REve (aka Eve-Web)
  - Rewrite of TEve for the web & ROOT-7
    - Uses OpenUI5 & JSRoot
    - Driven by CMS FireworksWeb development
    - Several high-level Fireworks features moved into REve:
      - physics collections,
      - item filtering, and
      - table-views
- REve History:
  - CHEP 2018 (Sofia) – proof of concept: server-client core, data exchange, remote method execution.
  - CHEP 2019 (Adelaide) – functional prototype of CMS FireworksWeb
  - **End of 2021:** Deployed Event-display service for CMS; servers at CERN and at UCSD
    - Access any CMS data through AAA (through XCache), from CERN EOS and CERNBox
- **REve is the core technology used for CMS visualization**
  - TEve application still supported for exotic use-cases: P5 online, geometry browser

Story of REve and RenderCore II.

- JSRoot uses THREE.js for 3D plots & geometry
    - "inherited" into REve → a bunch of issues, to be discussed …
- RenderCore – lightweight deferred rendering WebGL 2.0 framework in JS
    - Research-oriented render engine developed at the U. of Ljubljana, Department for Computer Graphics and Multimedia
    - Also used for collaborative visualization of medical data → Med3D
- RenderCore ⇒ REve timeline
    - 2018 – predecessor Med3D presented at HSF Workshop in Naples
    - 2019 – expression of interest from our side, some in-depth explorations through 2020
    - Mid 2021 – extraction of Med3D rendering engine + cleanup → RenderCore!
    - **End of 2022** – RenderCore is the default render engine for REve (post root-6.28)
    - 2023/24, in progress – RenderCore uses WebGPU


Shared with me - Google Drive
RelValMuMureco.root [4/10]
Photos - Google Photos
fireworks.ucsd.edu:10018/win1/?token=PIPkXzTZbDjNy0L7
Quit
Config
Edit
Subscribe
InvMassDialog
Run 1
Lumi 1147
Event 57311
CMSSW Client Alive
Log
Help
61
AutoPlay:
EnableFilter
FilterDialog
RelValMuMureco.root 4/10
RPhi
RhoZ
Open Controls
Add Collections
ECal
HCal
Jets
Muons
Tracks
Electrons
Vertices
CSC-segments
Photons
MET
BeamSpot
Jets Editor
RnrSelf
MainColor
i.pt()>20
Table
Choose Collection:
Jets
Edit table:
Idx Filtered pT eta phi emf
0 1 37.7 4.231 1.300 0.296
1 1 30.4 -2.946 -1.314 0.232
2 1 28.8 3.063 2.074 0.276
3 1 27.1 -2.959 1.421 0.434
4 1 27.0 4.186 -2.648 0.430
5 1 26.1 -4.461 0.018 0.390
6 1 25.9 3.712 -1.043 0.426
7 1 24.1 -3.835 -2.687 0.459
8 1 23.9 -2.767 2.779 0.046
9 1 22.9 3.981 -0.156 0.371
10 1 21.8 2.921 -0.175 0.222
11 1 21.3 -4.561 -1.039 0.357
12 1 21.3 -2.807 2.097 0.085
13 1 21.2 -3.767 0.706 0.516
14 1 21.0 3.872 -1.734 0.408
15 1 20.8 -4.965 1.158 0.380
16 1 20.5 -2.847 -2.818 0.041

THREE.js grievances: high-level

- TEve used custom, low-level OpenGL-1.x engine / scene graph (RootGL)
  - Developed in sync with TEve as needed to support CAD-like features of EVE
    - One gets spoilt by this flexibility. Several advanced features not available in standard rendering engines.
  - Migration to a "modern" OpenGL was never even considered → would require a major rewrite
  - *REve ⇒ migration to both a modern GL and to a server-client architecture (geometry serialization)*
- Global / sociological issues with THREE.js
  - Large project with a lot of users and a lot of functionality & features REve does not need
    - Tight integration from API classes → rendering pipeline → shaders
  - Very hard to introduce custom changes that spawn across the whole framework …
    - Changes need to go into several places and coexist with other advanced/exotic features we do not need.
    - They are hard to implement as one needs full chain understanding.
  - … and impossible to get them included in the main distribution / repo
    - What we need is rarely used (specific to CAD-like nature of Eve)
    - Would make things harder to support (Who are you, anyway?)

    } I actually sympathize with this attitude.
  - Release and low-level change cadency is rather large
    - Guaranteed (to some extent) API class interface and functionality – but back-end can change significantly.
    - For REve we really prefer complete stability (backport things if needed). E.g., RootGL stable since ~2008.

THREE.js grievances: technical / functional

1. Support for morphable memory-optimized instantiated objects
   - E.g. polygons or polyhedra with *some* varying properties: position, angle(s), scale(s), color
     - High-granularity calorimeter hits
   - Requires support from API classes, render-driver and shaders (including custom shader input)
2. Picking / selection / highlight
   - THREE uses ray–mesh intersection; this does not work for:
     - points (sprites) / lines, esp. at close-up views
     - instanced objects; does not work at all when geometry morphing / transformation is done in vertex shader
3. Multiple subsets of highlighted objects and/or sub-objects
   - Requires low-level renderer and render-buffer control (juggling) for optimal implementation
4. 3D lines of arbitrary thickness
   - Complete pain in (Web)GL; GL-1.x-style thick lines only supported on few architectures
   - Especially when you want 2. and 3. above to work

We had workarounds for some of those … to some extent …

RenderCore – motivation for adoption

- Most of the stuff on previous two slides … but:
  - It doesn't mean things were easy … they were for sure easier and, in fact, possible.
  - Will discuss some of them in more detail
- Collaboration!
  - It is of great help to be able to discuss details of APIs and implementations
    - RCore folks do the low-level core changes as needed
    - ROOT folks get to learn the art of low-level WebGL and are able to provide back meaningful extensions
  - Further avenues: PR, VR, advanced rendering techniques; interesting for other parts of ROOT
- Beautification: lighting models, post-processing / filtering
- Release and versioning at our own pace
- Follow state of the art in web graphics
  - Early transition towards WebGPU

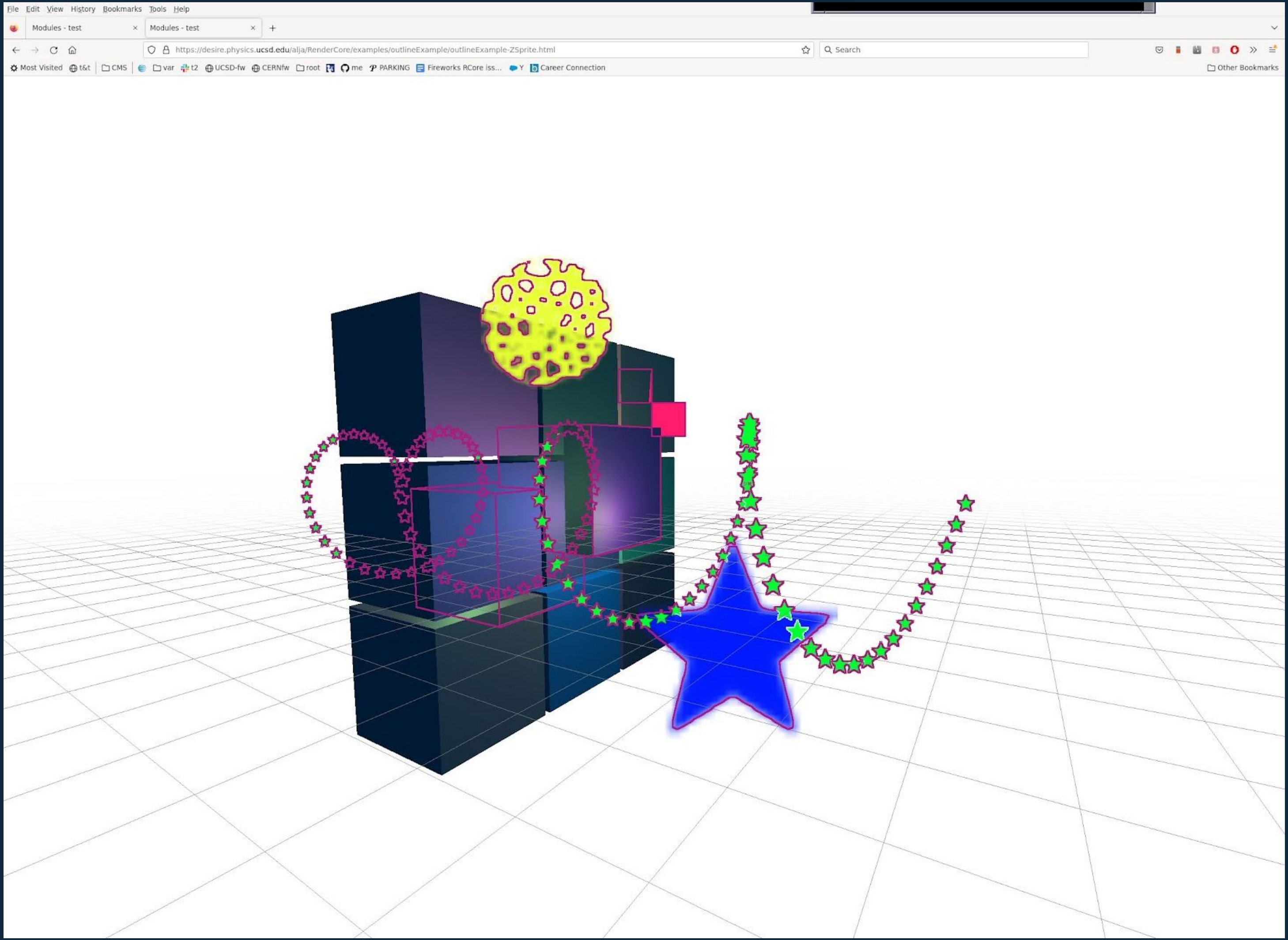
REve@RCore implementation highlights I.

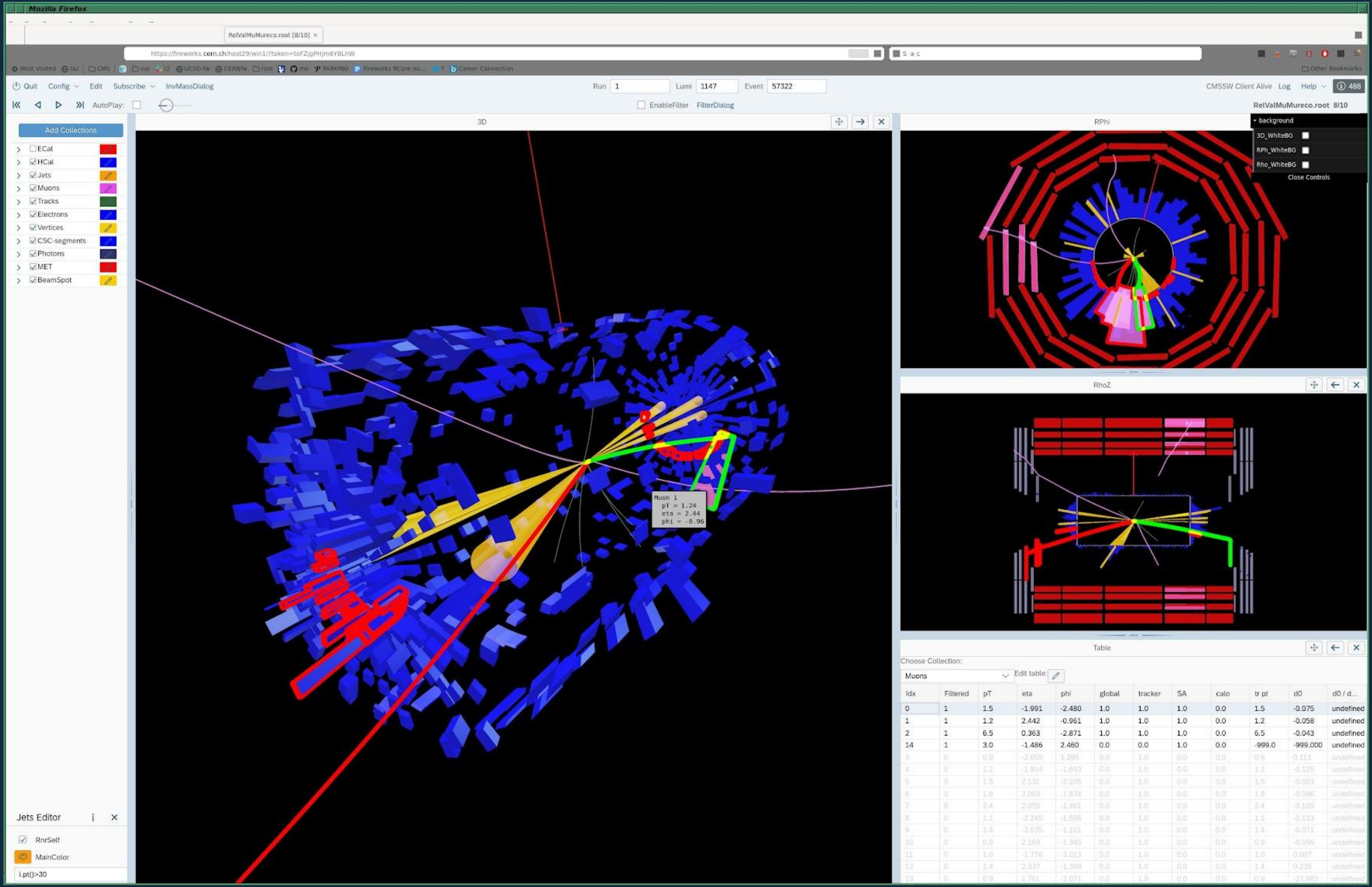
Object & sub-object picking with rendering to off-screen buffers

- Gives pixel-perfect results: if it is under mouse ⇒ it gets picked
- Implemented via dedicated shaders (or code paths) that "render" object ids (uint)
    - automatically assign object ids during pre-render traversal
    - use limited viewport, 32x32 around pick (mouse) position
    - z- depth is also extracted (R32F) – for placement of annotations, camera center placement
    - sub-object / instance picking implemented in a similar way:
        - single object is rendered, using instanceID or vertexID for output color, as required in shader
- Requires support in MeshRenderer and render-driver (custom REve component)

Highlighting / outlines based on a subset of G-buffer components

- Detects selected object edges and sudden changes in normal direction
    - Outlines objects and also edges to aid in shape recognition
- Requires: z-depth + view-space normals + view-direction vectors


File Edit View History Bookmarks Tools Help
Modules - test
Modules - test
https://desire.physics.ucsd.edu/alja/RenderCore/examples/outlineExample/outlineExample-ZSprite.html
Search
Most Visited t&t CMS var t2 UCSD-fw CERNfw root me PARKING Fireworks RCore iss... Y Career Connection
Other Bookmarks

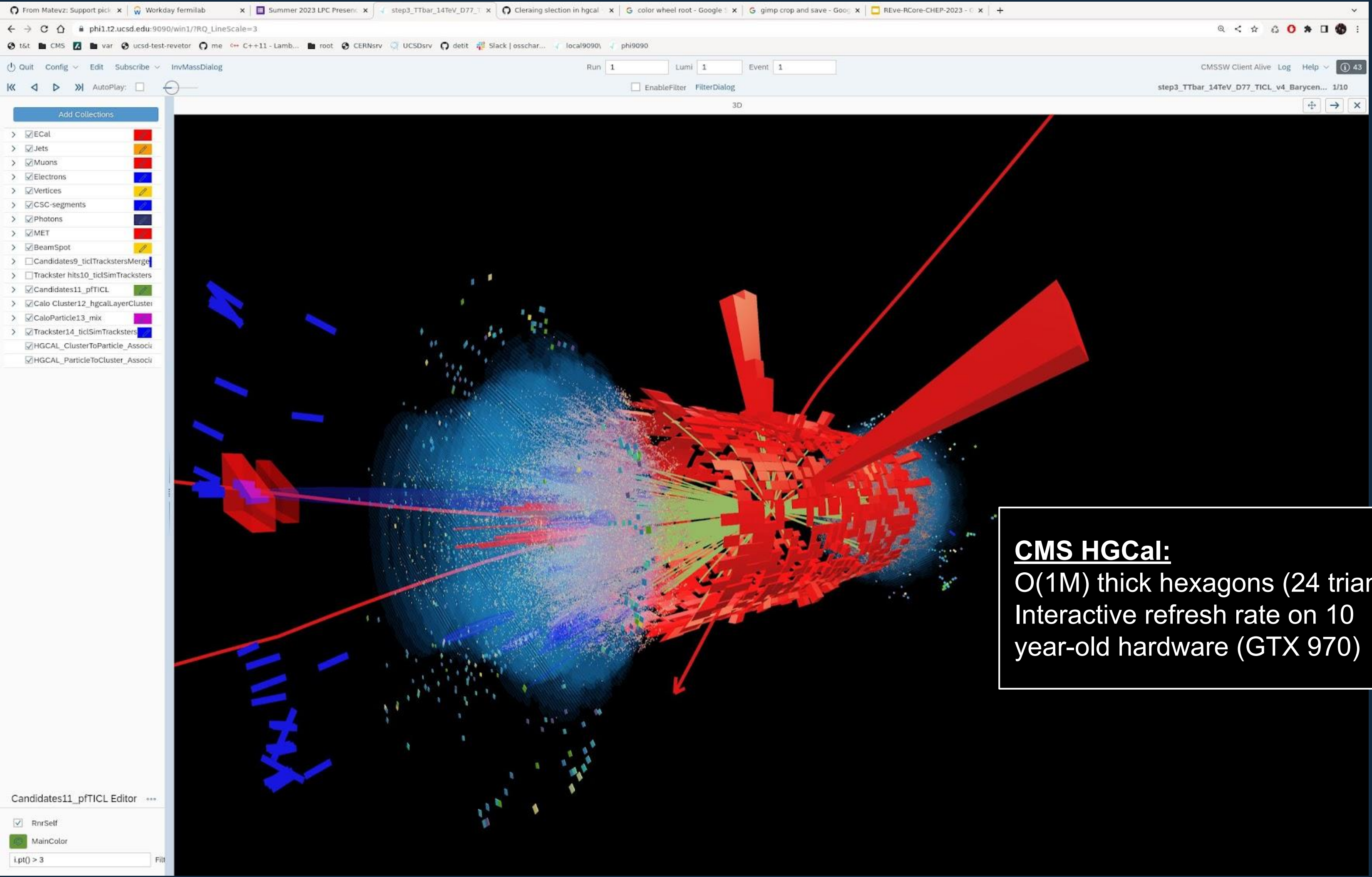


Mozilla Firefox
RelValMuMureco.root [8/10]
https://fireworks.cern.ch/host29/win1/?token=toFZjgPHjm6YBLhW
Quit
Config
Edit
Subscribe
InvMassDialog
Run 1
Lumi 1147
Event 57322
CMSSW Client Alive
Log
Help
488
AutoPlay:
EnableFilter
FilterDialog
RelValMuMureco.root 8/10
Add Collections
ECal
HCal
Jets
Muons
Tracks
Electrons
Vertices
CSC-segments
Photons
MET
BeamSpot
3D
RPhi
background
3D_WhiteBG
RPh_WhiteBG
Rho_WhiteBG
Close Controls
RhoZ
Muon 1
pT = 1.24
eta = 2.44
phi = -0.96
Table
Choose Collection:
Muons
Edit table:
Idx Filtered pT eta phi global tracker SA calo tr pt d0 d0 / d...
0 1 1.5 -1.991 -2.480 1.0 1.0 1.0 0.0 1.5 -0.075 undefined
1 1 1.2 2.442 -0.961 1.0 1.0 1.0 0.0 1.2 -0.058 undefined
2 1 6.5 0.363 -2.871 1.0 1.0 1.0 0.0 6.5 -0.043 undefined
14 1 3.0 -1.486 2.460 0.0 0.0 1.0 0.0 -999.0 -999.000 undefined
Jets Editor
RnrSelf
MainColor
i.pt()>30

REve@RCore implementation highlights II.

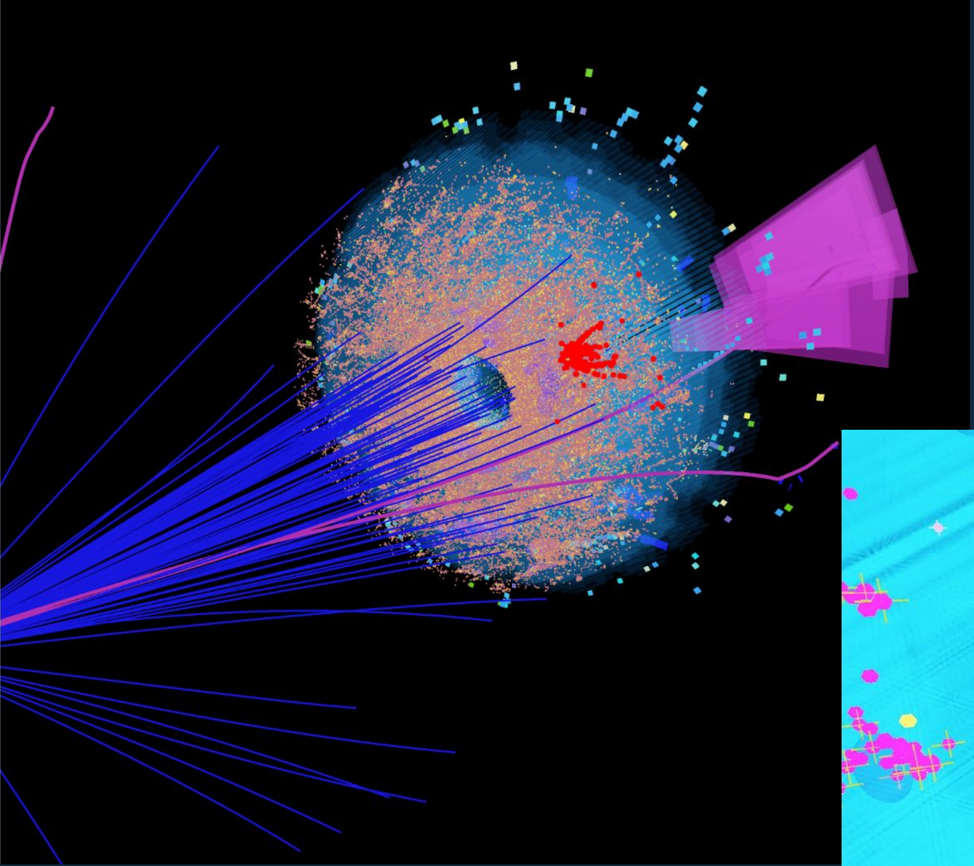
Instanced objects

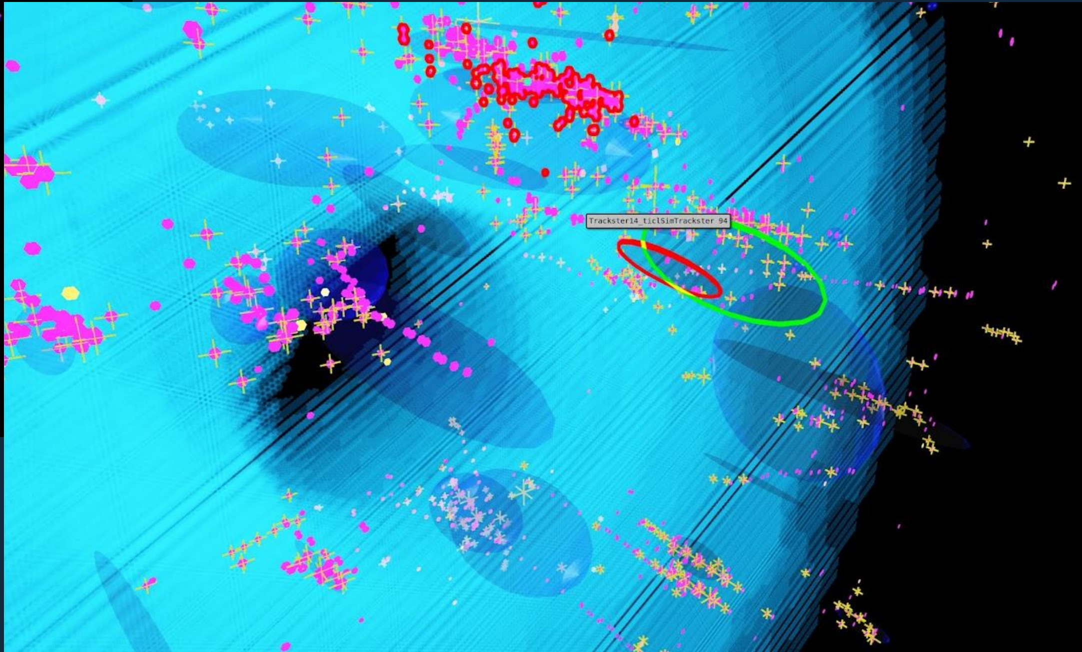
- Needed for display of digits / hits / towers / cylinders / cones
  - We need: position, scaling, color; limited ability to rotate things, if at all
- "Standard" instancing uses the same mesh and full 4x4 matrix per instance
  - Can save significant amount of memory and transfer less data by reducing the per-instance data.
  - Hard to implement instance picking and instance outlining without duplicating objects.
- For WebGL instance-data needs to be packed into "data" textures
  - Transferred from the server to GPU as a blob. However, one has to coordinate three things:
    - data packing on the server side (hard to mix floats and ints / shorts / bytes),
    - texture format / size interpretation in JS/WebGL, and
    - instance stride and component interpretation in the shader code.
- Can potentially also be used for geometry rendering
  - … to be explored



<u>CMS HGCal:</u>

O(1M) thick hexagons (24 triangles)
Interactive refresh rate on 10 year-old hardware (GTX 970)




Trackster14_ticlSimTrackster 94

Near future: WebGPU & WebAssembly

- WebGPU – RenderCore implementation in progress
  - Motivation:
    - Performance
    - Ability to carry memory blocks and data-structures from server to GPU without repacking or reinterpretation
    - Compute shaders / programs
  - RCore transition happening now – in summer 2022 it seemed everything will happen faster
    - WebGPU implementation in most browser/OS combos getting out of beta (e.g. Chrome-113, May 2, 2023).
    - Will follow up and transition when browser support is solid, especially on linux.
  - When to Release REve @ WebGPU – when to drop WebGL
    - We were hoping to converge on a single backend – and implement some advanced features only once
- WebAssembly offers additional options for optimization
  - Currently, processing is done at the server (complex part) or in shaders (lightweight things)
    - balance between memory / data-transfer volume / CPU / GPU usage
  - WASM can be useful for bulk calculations within the scene graph processing
    - Lightweight: calculating normal / model-view matrices, view / clip plane culling → wasm supports SIMD!
    - Full-scale: C++ renderer that is used both on the web and natively.
  - We have spent some time trying it out – requires rather elaborate memory mapping / management

Conclusion

- RenderCore was selected as the rendering engine for ROOT-Eve
  - This allowed us to reclaim advanced functionality and low-level control over object-data representation and rendering pipeline.
  - Learning new things and implementing advanced functionality was reasonably easy with the help of graphics professionals.
- REve and RenderCore are used as the core visualization technologies for CMS
- Transition of RenderCore to WebGPU is ongoing
  - Proceeding in sync with finalization of the API and implementation in browsers.
  - This will allow us to perform final optimizations of REve …
  - .. and provide ROOT event visualization with a state of the art 3D graphics.
- We are welcoming further collaboration
  - Both in the context of REve framework …
  - … as well as in RenderCore and, potentially, other graphics endeavours.