VIRTUE: Virtual Interactive Reality Toolkit for Understanding the EIC

User Guide V 3.0.0

Sean Preins
University of California, Riverside

Introduction

VIRTUE is a standalone customizable event display for collider experiments for desktop or virtual reality (VR). The application is available for free on Steam, Google Play, and Zenodo, and features a simplified and engineering model of the ePIC detector for the future Electron-Ion Collider, as well as simulated electron-proton collisions. New collision data and detector geometries can be uploaded into the program as JSON or FBX files within the application folder.

`https://store.steampowered.com/app/2728380/VIRTUE`
`https://play.google.com/store/apps/details?id=com.Quantanaut.VIRTUE&hl=en_US`
`https://doi.org/10.5281/zenodo.10372110`

System Requirements

The desktop version of the application is available for Windows, macOS, and Linux platforms. System requirements are determined by the Unity 6 runtime requirements:

`https://docs.unity3d.com/6000.2/Documentation/Manual/system-requirements.html`

The VR version of the application is compatible with Oculus Quest 2/3 headsets through Meta Horizon Link and with OpenXR-compatible VR systems, including SteamVR-supported headsets. The host Windows PC must satisfy the Unity 2022.3 system requirements:

`https://docs.unity3d.com/2022.3/Documentation/Manual/system-requirements.html`

When using Meta Horizon Link with a Quest headset, the additional Meta requirements are:

`https://www.meta.com/help/quest/140991407990979/`

For other OpenXR-compatible headsets, users should consult the requirements of the corresponding OpenXR runtime and VR platform.

The mobile version of the application is available for Android devices. System requirements are determined by the Unity 6 runtime requirements listed above.

Displaying Event Data

The program comes pre-loaded with a JSON file containing simulated electron-proton data in the ePIC detector at the EIC, as well as an example event data file. To display your own data, create a JSON file within the folder located at:

- **Windows:** `"VIRTUE [Platform]_Data/StreamingAssets/Events/"`
- **macOS:** Right-click on the application, select "Show Package Contents," and navigate to `"Contents/Resources/Data/StreamingAssets/Events/"`
- **Linux:** `"VIRTUE [Platform]_Data/StreamingAssets/Events/"`

The JSON files in this folder contain data for particle collision events within the virtual environment. For best performance, it is recommended to store only a small (< 100) number of events within each file, depending on how populated the events are.

Particle collision data can be generated directly from EDM4eic ROOT files using the Python conversion tools available in the RootForVIRTUE repository:

`https://github.com/SeanBP/RootForVIRTUE`

The conversion scripts read EDM4eic data products from CERN ROOT files and produce VIRTUE-compatible event JSON files containing tracks, hits, clusters, and other detector information.

The format of the JSON data file is as follows:

```
"header": {  // Header contains metadata for the event data
  "version": "3.0.0",   // Version of the data format
  "experiment": "Example Data",  // Optional: Name of the experiment
  "energy_unit": "GeV",          // Optional: Unit for energy (default: GeV)
  "color_bar": "Lin",            // Optional: Type of color bar (Lin or Log)
  "scale": 2.0,                  // Optional: Scales the size of the event
  "length_unit": "mm",           // Unit of length (e.g., m, cm, mm)
  "particles": [                 // Optional: Array of particle objects to be
      animated before the event
    {
      "size": 150.0,        // Size of the particle
      "color_rgba": [1.0, 0.0, 0.0, 1.0], // RGBA color values
      "ip": [0.0, 0.0, 0.0], // Optional: Interaction point at t=0 (default: origin)
      "angle_rad": [0.0, 3.14159] // Direction in radians
    }
  ],
  "tracker_settings": {
    "B_field_T": 2,         // Optional: Magnetic field in Tesla (default: 0 T)
    "tracker_boundary": [10000.0, -20000.0, 20000.0]
    // Cylindrical boundary for the tracker as an array of radius, left boundary,
        right boundary. All tracks will terminate outside of this boundary.
  }
},
// Array of events containing event-specific data
"events": [
  {
    // Contains metadata for the event
    "event_data": {
      "info_text": "Example Event", // Optional: Event description
      "energy_scale": [0, 10]       // Optional: Energy range for the color bar
    },
    "hits": [                     // Optional: Array of hit objects
      {
        "position": [1000, 0, 0], // Position of the hit
        "time_ns": 10,            // Optional: Time of the hit in nanoseconds
        "size": 200.0,            // Size of the hit
        "color_rgba": [0.5, 0.0, 0.5, 0.8]
        // Optional: RGBA color values for the hit
      }
    ],
    "clusters": [                    // Optional: Array of cluster objects
      {
        "position": [0, 1000, 1000], // Position of the cluster
        "granularity": 300,          // Granularity of the cluster
        "length": 3000,              // Length of the cluster
        "time_ns": 20,               // Optional: Time in nanoseconds
        "color_rgba": [0.0, 0.0, 1.0, 1.0]
        // Optional: RGBA color values for the cluster
      }
    ],
    "jets": [                       // Optional: Array of jet objects
      {
        "length": 3000,             // Length of the jet
        "R_rad": 0.1,               // Opening angle in radians
        "angle_rad": [1.57, 1.57],  // Direction in radians
        "vertex": [0, 1000, 0],     // Optional: Vertex of the jet (default: origin)
        "time_ns": 20,              // Optional: Time in nanoseconds
```

```
        "color_rgba": [0.0, 0.0, 1.0, 1.0]
        // Optional: RGBA color values for the jet
      }
    ],
    "tracks": [                    // Optional: Array of track objects
      {
        "qOverP": -3,              // Optional: Charge over momentum ratio (default
            : 0)
        "angle_rad": [2.3, -1.57],  // Direction of the track in radians
        "vertex": [0.0, 500.0, 0.0],// Optional: Starting position (default: origin)
        "duration_ns": [5.0, 30.0], // Optional: Start and end times
        "color_rgba": [0.0, 0.0, 1.0, 1.0]
        // Optional: RGBA color values for the track
      }
    ],
    "blocks": [                        // Optional: Array of block objects
      {
        "position": [1000, 1000, 1000], // Position of the block
        "time_ns": 20,                 // Optional: Time of the block in
            nanoseconds
        "size": [200, 400, 600],       // Size along x, y, z axes
        "euler_angles_deg": [100, 200, 300],
        // Optional: Rotation in degrees along x, y, z axes
        "color_rgba": [1, 0, 0, 1]     // Optional: RGBA color values for the block
      }
    ]
  }
]
```

Displaying 3D Models

VIRTUE comes pre-loaded with a JSON file and FBX files containing the geometry of the ePIC detector at the EIC, as well as an example JSON 3D model file. To display a new 3D model, create a JSON or FBX file in the appropriate directory:

- **Windows:** `VIRTUE [Platform]_Data/StreamingAssets/Models/`
- **macOS:** Right-click on the application, select `"Show Package Contents"`, and navigate to: `Contents/Resources/Data/StreamingAssets/Models/`
- **Linux:** `VIRTUE [Platform]_Data/StreamingAssets/Models/`

For Steam installations, locate the application folder by opening Steam, selecting VIRTUE in the library, clicking the gear icon, then selecting `"Manage"` → `"Browse local files"`.

FBX Model Support

Many 3D computer-aided design (CAD) programs allow models to be exported in FBX (Filmbox) format. Alternatively, many CAD file formats can be converted to FBX using third-party software, including online file conversion tools. When exporting, ensure that textures, materials, and colors are embedded into the file.

FBX files in VIRTUE are loaded using the TriLib2 model loading package. To verify compatibility, upload your FBX file to the TriLib2 Model Viewer Web Demo: `https://ricardoreis.net/trilib/trilib2demo/modelviewer/`.

SketchUp FBX Export Procedure

The native FBX exporter in SketchUp often results in improperly rendered models. To ensure proper rendering, it is recommended to convert SketchUp files to FBX using the Unity Editor. The Unity Editor can be downloaded via the Unity Hub at `https://unity.com/download`. To correctly export a SketchUp model to FBX, follow these steps:

1. In SketchUp, **explode** all groups within the model and save.
2. In the Unity Editor, navigate to `Assets` → `Import New Asset...` and select the SketchUp (`.skp`) file.
3. In the `Assets` folder, create a new folder named `Color`.
4. Select the imported SketchUp file in the `Assets` folder. In the `Inspector` under the `Materials` tab, click `"Extract Textures..."` and `"Extract Materials..."`, saving them in the `Color` folder.
5. Drag and drop the SketchUp asset into the Unity scene. Adjust its orientation and position as needed.
6. In the `Hierarchy`, right-click the model and select `"Export To FBX..."`.
7. In the export options:
   - Set `"Include"` to `"Model(s) Only"`.
   - Set `"Object(s) Position"` to `"World Absolute"`.
   - Ensure `"Embed Textures"` is checked.
8. The exported FBX file is now ready for use in VIRTUE.

JSON-Based 3D Model Geometry

In addition to supporting FBX models, VIRTUE implements a custom JSON format for defining simplified detector geometries. This format enhances event visibility and allows for individual component labeling. It supports three component types: **toroid**, **block**, and **spheroid** components.

For optimal visualization, components should be listed from the innermost to the outermost structures. Circular components should be approximated with no more than 50 sides for improved performance. The expected JSON format follows this structure:

```
"header": { // Header contains metadata about the model configuration
  "version": "3.0.0",        // Version of the model format
  "detector": "Example Model", // Optional: Name of the model
  "length_unit": "m",          // Unit of length (e.g., m, cm, mm)
  "scale": 1.0                 // Optional: Scale of the model
},
"components": [
  {
    "index": 0,              // Index of the component to set the render order.
    "type": "toroid",        // Type of the component (toroid)
    "name": "Example Toroid",// Optional: Name of the toroidal component
    "position": [0, 0, 0],   // Position in space (x, y, z coordinates)
    "sides": 10,             // Number of sides for the toroid
    "radii": {               // Radii for the left and right sides
      "left": [0, 2],        // Inner and outer radii for the left side
      "right": [1, 2]        // Inner and outer radii for the right side
    },
    "length": {              // Lengths of the toroid
      "outer": 1,            // Outer length
      "inner": 2             // Inner length
    },
    "inner_offset": -1       // Optional: Offset of the inner barrel
    "euler_angles_deg": [10, 20, 45],
    // Optional: Rotation in degrees along x, y, z axes
    "color_rgba": [1.0, 1.0, 0.5, 0.3],  // RGBA color values
  },
  {
    "index": 1,
    "type": "block",         // Type of the component (block)
    "name": "Example Block", // Optional: Name of the block component
    "position": [4, 0, 0],   // Position in space (x, y, z coordinates)
    "size": [1, 2, 3],       // Size along x, y, z axes
    "euler_angles_deg": [0, 45, 0],
    // Optional: Rotation in degrees along x, y, z axes
    "color_rgba": [0.5, 1.0, 1.0, 0.3]  // RGBA color values
  },
  {
    "index": 2,
    "type": "sphereoid",         // Type of the component (sphereoid)
    "name": "Example Sphereoid", // Optional: Name of the sphereoid component
    "position": [8, 0, 0],       // Position in space (x, y, z coordinates)
    "size": [1, 2, 3],           // Size along x, y, z axes
    "euler_angles_deg": [0, 45, 0],
    // Optional: Rotation in degrees along x, y, z axes
    "color_rgba": [1.0, 0.5, 1.0, 0.3]  // RGBA color values
  }
]
```

Displaying Tours

The program comes pre-loaded with a JSON file containing "tour" information that will guide the user through different layers of the ePIC detector and collision events. To display your own tour, create a JSON file within the folder located at:

- **Windows:** `"VIRTUE [Platform]_Data/StreamingAssets/Tours/"`
- **macOS:** Right-click on the application, select "Show Package Contents," and navigate to `"Contents/Resources/Data/StreamingAssets/Tours/"`
- **Linux:** `"VIRTUE [Platform]_Data/StreamingAssets/Tours/"`

The JSON files in this folder contain data for scenes in a tour, which references a model and event file. The format of the JSON data file is as follows:

```
"header": {
        "version": "3.0.0",                     // Version of the data format
        "model_file": "Example_Model.json",    //Name of the model file
        "events_file": "Example_Events.json", //Name of the events file
  }
  "scenes": [                         // Array of scenes in the tour
    {
      "model_settings":
            "components": [0, 1, 2], // Indexes of the components to show
            "lines_active": [0, 1],  // Indexes of the components to show the
                wireframe of
      }
      "text":{
            "title": "Scene 1",                           // Title of the scene
            "body": "This shows a scene with no event."  // Body of text to display
                in the scene
      },
      "camera_settings":{
            "position": [ 0.0, 2.0, -10.0 ],// Position of the camera
            "focus": [ 0.0, 0.0, 0.0 ]      // Position of the focus of the camera
      }
    },
    {
      "model_settings": {
            "all_components": true// Toggle to display all components in the model
      }
      "text":{
            "title": "Scene 2",
            "body": "This shows a scene with an event."
      },
      "event_settings": {
            "index": 3,      // Index of the event to animate during the scene
            "time_before": 20,      // Nanoseconds to animate before the event
            "speed": 1              // Speed of the animation
       },
      "camera_settings":{
            "position": [ 0.0, 2.0, -10.0 ],
            "focus": [ 0.0, 0.0, 0.0 ]
      }
    }
```

Unity Projects

The Unity project files for both the desktop and VR versions of the application are available on Zenodo, to allow further customization if desired. `https://doi.org/10.5281/zenodo.10372110`

To edit the projects, download and unzip the project files and open them through the Unity Hub application: `https://unity.com/download`. The way the application renders and animates data from the JSON files can also be modified by editing the respective C# scripts in the assets folder of the project.

Once the desired edits are made to the project, build the project for your desired platform and run it from the executable file this generates. For further details, refer to the Unity documentation: `https://docs.unity.com/`.

Acknowledgments

Developer: Sean Preins
Supervisor: Prof. Kenneth Barish
https://arratialab.ucr.edu/
Electron-Ion Collider: www.bnl.gov/eic/
Simulations and detector design by the ePIC Collaboration:
www.bnl.gov/eic/epic.php
https://eic.jlab.org/Menagerie/
Music: Josh Puddington
Logo: Justyna Babinska "JustaSuta"
TriLib 2: Ricardo Reis
https://ricardoreis.net/trilib-2/
This work was supported by the University of California, Office of the President.