ANIMATING PROTON-PROTON COLLISIONS IN VRSPY FOR CMS

By

Youssef LECHGUAR

This report is submitted as part of the summer student program
in CMS experiment at CERN

22 August 2024

Supervisors: Muhammad Ansar Iqbal and David Barney

ABSTRACT

In high-energy physics, virtual reality has emerged as a powerful tool for bridging scientific complexity and human perception. This project, part of the CERN Summer Student Programme 2025, contributes to VRSPY, a VR visualization tool developed for the CMS experiment, by animating the aftermath of proton-proton collisions.

Focusing on the post-collision phase, the objective was to develop a realistic and educational animation based on actual CMS data. Implemented using Unity, the animation enhances the immersive experience by accurately depicting how particles emerge and interact with the detector environment. This work not only improves outreach and engagement but also lays the groundwork for future physics-based extensions of the VRSPY platform.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Muhammad Ansar Iqbal, for his continuous guidance, support, and encouragement throughout this project. His expertise and feedback have been invaluable in shaping both the direction and outcome of my work.

I am also thankful to CERN for offering me the opportunity to take part in the Summer Student Programme 2025. This experience has been incredibly enriching, both technically and personally, and has deepened my interest in the field of high-energy physics and scientific visualization.

Finally, I would like to thank my family and friends for their unwavering support and motivation during this journey.

Contents

1 Introduction

Scientific visualization plays a critical role in helping researchers and the general public understand the complex phenomena observed in high-energy physics experiments. With the growing availability of immersive technologies, virtual reality (VR) has emerged as a powerful medium to explore and communicate the inner workings of particle detectors and collision events in an intuitive and engaging way.

The Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) is one of the world's most sophisticated detectors, designed to observe a wide range of particles resulting from proton-proton collisions. However, the sheer complexity and scale of the data produced make it challenging to present it in a human-understandable format. To address this, UCLA has developed a VR platform called **VRSPY**, aimed at visualizing real CMS collision events inside a virtual reality environment.

This report summarizes the work carried out as part of the CERN Summer Student Programme 2025, contributing to the VRSPY project by designing and implementing animations that depict the aftermath of proton-proton collisions. The objective was to enhance the immersive experience by providing accurate and educational representations of the collision dynamics using Unity, with a particular focus on the post-collision phase where various particles are created and propagate through the detector.

Through this work, the project seeks not only to improve scientific outreach but also to open the door for future extensions of the VRSPY platform, potentially incorporating real-time data streaming, user interaction, or augmented analysis capabilities.

2 VRSPY Framework and Tools

VRSPY is a virtual reality application developed by the University of California, Los Angeles (UCLA), in collaboration with the CMS experiment at CERN. It is designed to bring real proton-proton collision events from the CMS detector into an immersive 3D environment, enabling users to explore particle interactions from a first-person perspective using a VR headset.

The system consists of two main components: a **desktop application** and a **headset application**. The desktop app is responsible for parsing CMS event data files, selecting events of interest, and transferring them to the standalone VR headset (e.g., Meta Quest) via USB debugging. The headset app, built using the Unity game engine, interprets this data and renders the corresponding detector geometry and particle tracks inside a fully navigable VR scene.

2.1 VRSPY Architecture

The overall architecture of VRSPY supports both outreach and scientific purposes. Users can load selected CMS events, typically in '.obj' or simplified formats, into the desktop interface, preview the reconstructed objects, and then stream the event to the VR headset.

Once received, the Unity-based VR application decodes the particle tracks, assigns them visual representations (e.g., colored trails or lines), and integrates them into a 3D model of the CMS detector. The user can then walk around the scene, look inside the interaction point, and observe the aftermath of a real high-energy collision from different perspectives.

2.2 Tools and Technologies

The development of VRSPY involves a variety of tools and technologies:

- **Unity 3D:** The game engine used to build the immersive VR application.
- **C#:** The programming language used within Unity for scripting logic, animations, and interaction.

- **Meta Quest** / **Oculus:** The hardware platform targeted for standalone VR experience.
- **Event Loader and Parser:** Custom tools developed to convert CMS event data into a simplified format suitable for real-time VR rendering.

This modular setup allows developers to expand or update the experience without needing to recompile the entire system, making VRSPY an ideal platform for both educational and research use cases.

3 System Engineering Diagrams

This section provides an overview of the system design and development methodology used in the VRSPY project. Using system engineering diagrams, we describe the life cycle of the project, its core functional requirements, and the user interactions with the system.

3.1 System Life Cycle Diagram

The life cycle diagram illustrates the major phases of the VRSPY project's development process, from requirement analysis to deployment. It captures how the system evolved from early design to final implementation during the internship. This top-down view helps visualize the flow of activities and their order of execution.

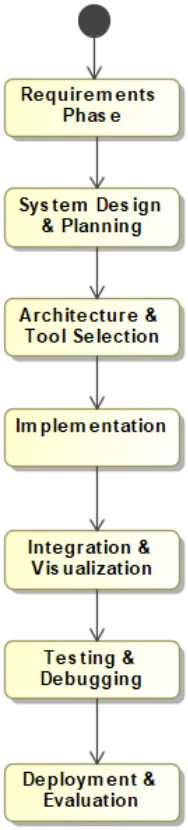

Figure 1: System Life Cycle Diagram for VRSPY

3.2 Requirements Diagram

This diagram breaks down the core functional and non-functional requirements of the VRSPY system. It connects each requirement to its purpose and relevance to different system components, such as animation handling, event loading, and VR rendering. The goal is to clarify what the system must do to fulfill user and technical expectations.

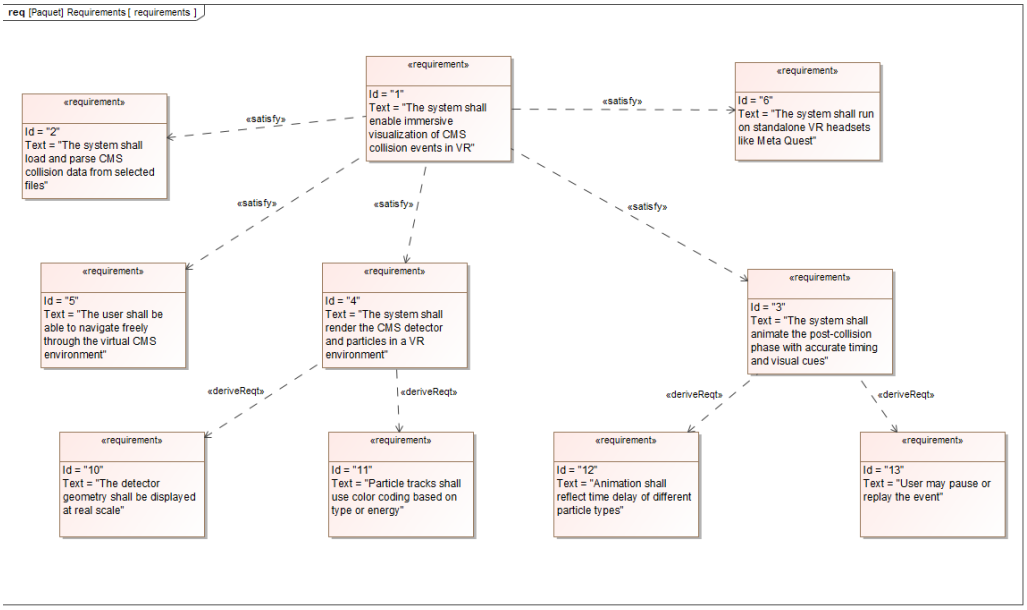

Figure 2: Requirements Breakdown Diagram

3.3 Use Case Diagram

The use case diagram identifies the different types of users (actors) who interact with the VRSPY system and the key functions they perform. It includes public users, physicists, and developers, each of whom engages with different system features such as starting the experience, visualizing events, or uploading content.

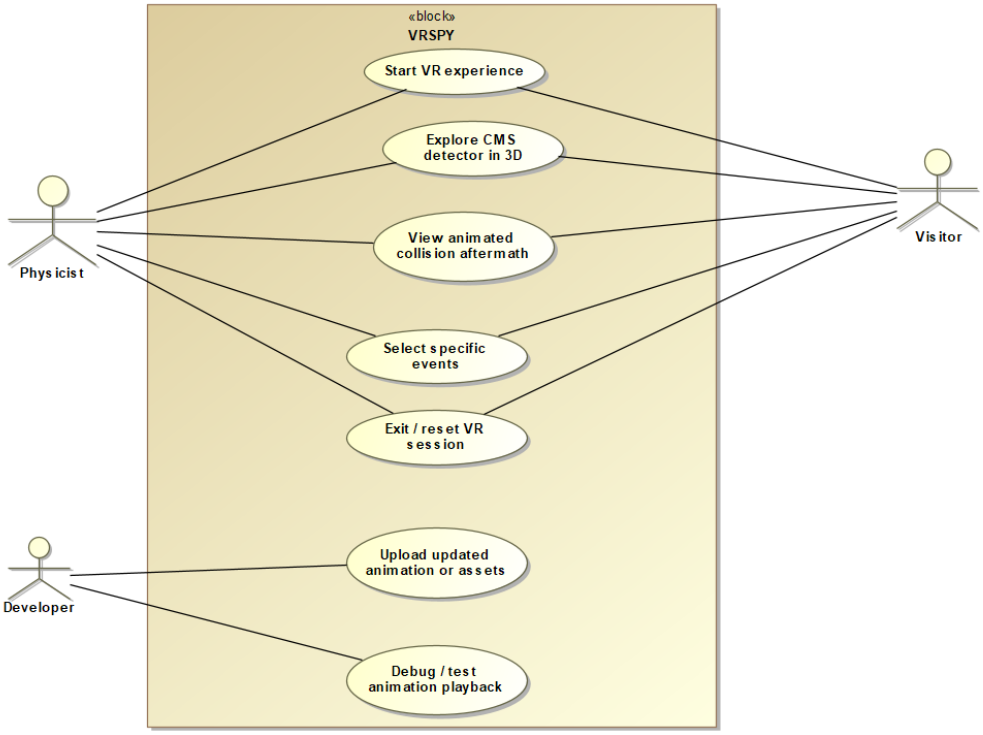

Figure 3: Use Case Diagram for VRSPY Visualization

4 Implementation and Development

This chapter outlines the technical contributions made during the CERN Summer Student Programme, focusing on the implementation and integration of an animated visualization in the VRSPY project. The following sections detail the shader-based animation system, Unity scene configuration, data integration, and performance optimizations carried out over the course of the internship.

4.1 Shader-Based Animation Using Alpha Clip

A core component of the animated effect in VRSPY was made possible through the use of a custom shader developed in Unity Shader Graph. This shader was designed to control the visibility of mesh surfaces dynamically based on their distance from a defined center, using the **Alpha Clip Threshold**.

The technique involves calculating the distance between the object's world position and a central origin point. This distance is then passed through a `Step` function to create a hard threshold, objects within the threshold remain visible while others are clipped out. By animating this threshold over time, the material reveals itself gradually in a spherical manner, creating the illusion of an expanding shockwave or particle burst following the proton-proton collision.

The final value is multiplied with other gradients and connected to the Alpha Clip input of the fragment shader, allowing efficient per-pixel masking. This method is not only performant but also visually compelling, especially in the VR environment where clarity and smoothness are critical.

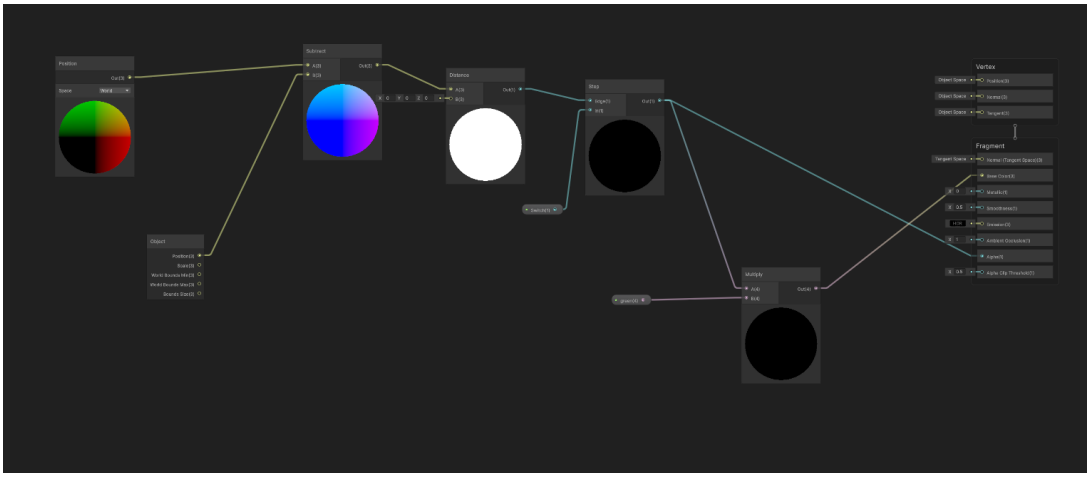

Figure 4: Custom Shader Graph used to control Alpha Clip for animated material visibility

4.2 Unity Scripting for Animation Control

To automate material assignment and control the Alpha Clip effect during the animation, I developed two custom scripts in Unity. These scripts simplify the material setup process and allow precise control over the shader's visibility logic during playback.

Script 1: ApplySharedMaterialToChildren.cs *Assigns the animated material to all child objects in the scene hierarchy.*

```
using UnityEngine;

#if UNITY_EDITOR
using UnityEditor;
#endif

public class ApplySharedMaterialToChildren : MonoBehaviour
{
    public Material targetMaterial;

#if UNITY_EDITOR
    [ContextMenu("Assign Material To Children")]
    void AssignMaterial()
    {
        Renderer[] childRenderers = GetComponentsInChildren<Renderer>(true);

        Undo.RecordObjects(childRenderers, "Assign Material to Children");

        foreach (Renderer rend in childRenderers)
        {
            rend.sharedMaterial = targetMaterial;
```

```
            EditorUtility.SetDirty(rend);
        }

        Debug.Log("Material assigned to all children.");
    }
#endif
}
```

Script 2: MaterialTimelineController.cs *Updates the shader's* `_Switch` *value at runtime to control the Alpha Clip threshold.*

```
using UnityEngine;

[ExecuteAlways]
public class MaterialTimelineController : MonoBehaviour
{
    public Material sharedMaterial;
    [Range(0, 20)]
    public float switchValue;

    private void Update()
    {
        if (sharedMaterial != null)
        {
            sharedMaterial.SetFloat("_Switch", switchValue);
        }
    }
}
```

4.3 Animator Setup with Speed Multiplier

In addition to shader-based animation, I implemented object-level motion using Unity's Animator system. Each animated track was connected to a pre-recorded motion clip and managed using an Animator Controller. The animation playback speed was controlled using a parameter-driven multiplier, which allowed for dynamic adjustment of motion depending on the context or event type.

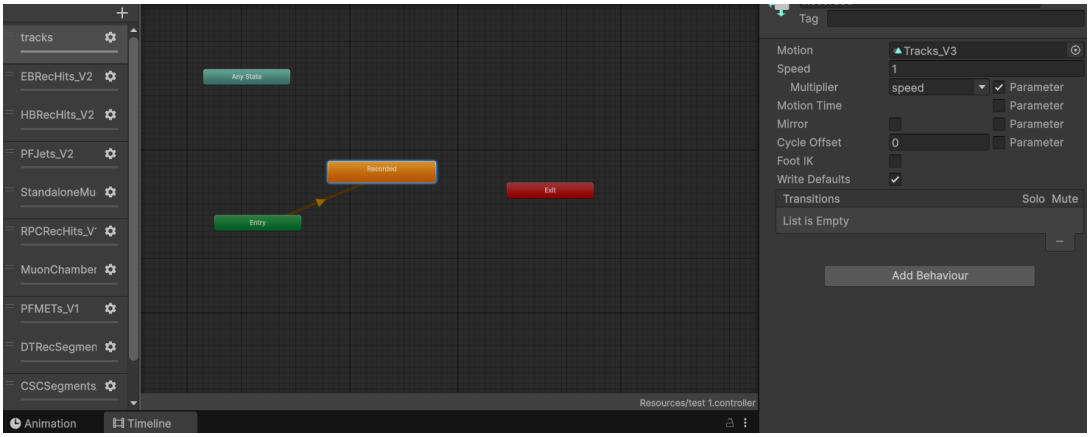

Figure 5: Animator Controller using a speed multiplier to drive motion playback

5 Conclusion

This report has presented the work carried out during my internship as part of the CERN Summer Student Programme 2025, focusing on the VRSPY visualization platform for the CMS experiment. My contribution was centered on the development of a realistic and performant animation system that enhances the immersive visualization of post-collision particle events. Through the use of custom Unity shaders, Alpha Clip thresholding, Animator controllers, and material scripting, I was able to design an experience that is both visually effective and technically efficient. These components were carefully integrated into the existing VRSPY framework, contributing to its usability in outreach and educational contexts. This project allowed me to explore the intersection between high-energy physics and real-time interactive media, while gaining hands-on experience in Unity development, data integration, and shader-based animation. I believe this work lays a foundation for future improvements such as real-time interactivity, adaptive visuals, or broader CMS data support.

References

- The CMS Collaboration. (2008). *The CMS Experiment at the CERN LHC*. Journal of Instrumentation, 3(08), S08004. https://doi.org/10.1088/1748-0221/3/08/S08004
- *VRSPY – VR Event Visualization for CMS*. UCLA Physics VR. https://vr.physics.ucla.edu/vrspy.html
- Unity Technologies. (2024). *Unity Shader Graph Manual*. https://docs.unity3d.com/Packages/com.unity.shadergraph@12.0/manual/index.html
- Unity Technologies. (2024). *Unity Manual: Animator Controllers*. https://docs.unity3d.com/Manual/AnimatorControllers.html
- CERN Summer Student Programme. https://home.cern/summer-student-programme
- OpenGL ARB & Khronos Group. (2023). *Shader Programming Best Practices*. https://www.khronos.org/opengl/wiki/Shader