GSoC 2023 Project Proposal for

Improving the sense of scale and navigation in high energy physics event visualization

(Duration: 350 hours)

SOMYA BANSAL

INDIAN INSTITUTE OF TECHNOLOGY (BHU), VARANASI

+91 9413321059

somya.bansal.cse20@itbhu.ac.in

somya-bansal-73b041205

Somya-Bansal159

Table of Contents

1. Basic Information

1.1. Personal Details

Name: Somya Bansal
Major: Computer Science and Engineering
Degree: Integrated Dual Degree (B. Tech + M. Tech)
Year: 3rd
Institute: Indian Institute of Technology BHU, Varanasi, Uttar Pradesh, India
Resume:
https://drive.google.com/file/d/1xKwPCaVIXsvcdzliYJvS4WkHmOjJ1vbd/view?usp=sharing
Time zone: Indian Standard Time (UTC + 5.30)

1.2. About Me

I am Somya Bansal, a 3rd-year undergraduate at the Indian Institute of Technology BHU, Varanasi, pursuing an Integrated Dual Degree (B. Tech + M. Tech) program in Computer Science and Engineering. I have experimented with various domains like machine learning, computer vision, robotics, web development, etc. I love writing reusable and maintainable code for others to use.

I have previously worked with the Django framework for the backend and developed a few websites. For the front end, I am familiar with Angular JS.

- I have developed a Django-based application for the online registration of vaccination slots for Covid-19. This portal allows a user to book slots for his/her vaccination as per the available slots in various centers of his/her city. He/she can choose the type of vaccine as well.
- Another application is to write a daily review to boost productivity. Users can write a review of their whole day and view previous reviews as well as compare their reviews with that of their friends. Users can change the visibility of their review such that it is visible to everyone, their friends, or no one.

2. Project

2.1. Abstract

Phoenix is an open-source Angular-based web application written using the ThreeJS library to visualize the experiments conducted in various detectors of CERN LHC in 3D. Various tools can be found on the page to customize the view.

This project focuses on two features along with documenting them as well, i.e., navigation and scale. Navigation refers to developing a feature to easily browse through the geometry of the detector. The aim is to display a list of various parts of the detector, on which, if the user clicks, that part in the 3D scene is automatically highlighted so as to provide the user with an idea of its location. Scale refers to developing a feature to get an idea of the positions, dimensions, and scale of the objects while visualizing an event. This can be implemented using 3D rulers with grids or an interactive tool to show 3D coordinates of an object.

2.2. Motivation

The 3D visualization of the detector experiments by the means of Phoenix is an intelligent initiative to study collisions. Right now, the geometry of the detector and the event data has been rendered as ThreeJS meshes, but a researcher also needs an estimate of the relative locations of the particles and dimensions of the detector parts. The 3D coordinates, as well as gridlines, should be introduced so as to extract the position coordinates and do the necessary calculations. The Phoenix application greatly lacks this functionality, the implementation of which can highly boost the research work.

2.3. Why HEP Software Foundation?

I am specifically applying for HSF only for GSoC' 2023 and do not intend to apply anywhere else, the reason being I have always been intrigued by the quest going on in the Large Hadron Collider, especially since the discovery of the Higgs Boson particle. I also take an interest in the standard model and the latest discoveries done by

LHC. As a software developer who would love to contribute to open source and is equipped with ThreeJS, and also as a keen physics enthusiast, I could find no other place to learn from other than contributing to this project. I am looking forward to working with CERN-HSF in the future. Submitting this proposal is my very first initiative for the same.

2.4. Proposed Deliverables

- When the user clicks anywhere, the first object under the mouse pointer that is **visible to the user** will be noted, and its coordinates will be displayed near the mouse pointer for a second.
- An option will be provided that can be toggled to view a customizable 3D cartesian grid at the origin of the detector. The grid will have parallel XY planes, parallel YZ planes, and parallel ZX planes. Some of the customizations will be:
    - View any one, two, or all of the XY, YZ, and ZX planes.
    - Change the spacing between parallel planes.
    - Select the distance from the origin up to which gridlines are to be shown.
    - Show labels and coordinates on the X, Y, and Z axes.
    - (Optional) Change the origin of the gridlines to a specific subdetector, i.e., to translate the gridlines and thereby translate the position coordinates of the points by a certain value.
- User can click on a checkbox to enable finding distances between any two points. User can click any two points to find the Euclidean distance between them.
- Various parts of the detector will be listed upon toggling an option from the UI menu, and some information will be there beside every part. There will be two options in front of each list element. One of them can be clicked to zoom into the bounding box of that particular part. The second one will highlight/outline that particular part.
- When the user clicks on any detector part, its dimensions will be displayed for a second. In order to develop this representation, I

will investigate solutions in industry standard 3D applications, such as Blender.

- (Optional) **Fisheye effect:** Center of the scene (heart of the detector) will be enlarged and boundaries will be shrunk. The scene will be displayed on a logarithmic scale. It will allow user to view whole of the event simultaneously without zooming in/out.
- Documentation and Testing: I will write unit tests and E2E tests for the features to ensure that the implemented features will work as intended in various scenarios. I will document the updates made in various helper files. I will practice best practices to write optimized code so that lesser computations are to be done to smoothly render the display.

2.5. Plan of Action

I will begin by reviewing the structure of the codebase and analyzing where all the components displayed in the UI are defined. Also, I will take time to read the documentation to understand the helper functions in various manager files like `scene-manager.ts`, `selection-manager.ts`, `controls-manager.ts`, etc.

I have already started to work on some easier items. As of now, I have explored how to generate gridlines and have written its code. I have also generated a UI component to list detector parts. I will first finalize my ideas in discussion with Phoenix users and developers. Gridlines will not require much effort, but showing dimensions, distances, making it easier to navigate the geometry etc., will require a lot of research and will consume major portion of the timeline.

2.5.1. Improve the Grid Functionality

Expected due date: June 11
Objective: Add a customizable cartesian grid to the detector to better get an estimate of the position of the objects.
Deliverables:

- A toggle button for a 3D cartesian grid.

- A trackbar to change the number of planes. For example, say XY plane (z=0), so we can have z=±100, z=±200 etc. as well.
- A trackbar to change the sparsity of gridlines. For example, if the XY planes are z = 0, 100, 200, 300… its sparsity can be increased to z = 0, 200, 400, 600…
- Three toggle buttons to separately view XY, YZ and ZX planes. So, at a time, the user can choose to view any one, two or all three of the XY, YZ or ZX planes.

I will begin by modifying the `view-options` component in the UI menu to add another checkbox to show the cartesian grid.

The `view-options.component.ts` file will have the following extra functions:

- **`setCartesianGrid`** will initialize the grid, if none is already present, and toggle its display.
- **`addXYPlanes`** will display more XY planes even farther from the origin (using a trackbar).
- **`showXYPlanes`** will toggle the display of XY planes.
- **`changeSparsity`** will be used to change the spacing between the gridlines (using a trackbar).
- **`callSetShowCartesianGrid`** will call the main function from UI Manager.

```
callSetShowCartesianGrid() {
  this.eventDisplay
    .getUIManager()
    .setShowCartesianGrid(
      this.showCartesianGrid,
      this.showXY,
      this.xDistance,
      this.sparsity
    );
}
```

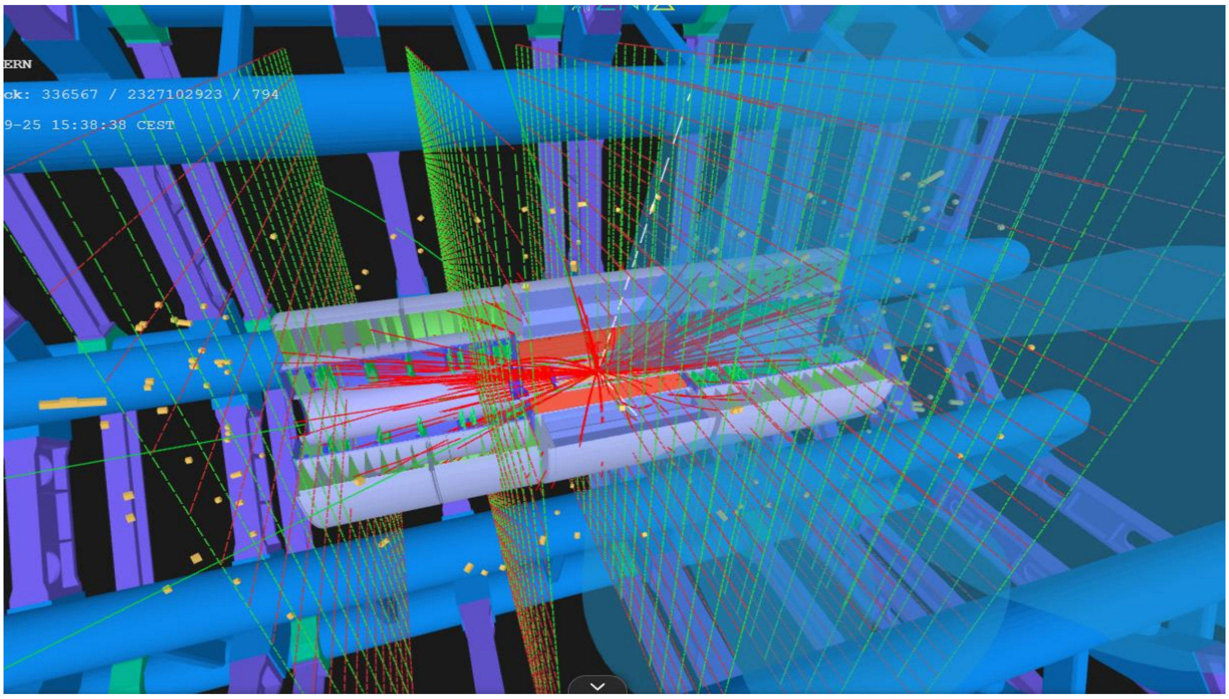
The respective changes will be incorporated in the corresponding html file as well.

The UI Manager will call the facility from the Scene Manager where it will be actually implemented. The main function inside the scene manager will look something like:

```
public setCartesianGrid(
    visible: boolean,
    zDistance: number,
    sparsity: number = 1,
    scale: number = 3000 // the maximum extent of the
                          // gridline
) {
    // initialise the grid for the first time
    if (this.cartesianGrid == null):
        this.cartesianGrid = new Group();

        for z = -scale to +scale:
            // define every consecutive xyPlane
            for y = -scale to +scale:
                // generate geometry of horizonatal lines
            for x = -scale to +scale:
                // generate geometry of vertical lines

            // add these lines to the scene

    this.cartesianGrid.children.visible = false;
    if(visible):
        for all the gridlines that the user wants:
        // based on sparsity, zDistance from origin upto
        // which gridlines are to be shown
            this.cartesianGrid.children.visible = true;
}
```

I will then extend this feature to YZ and ZX planes and add ruler (coordinates) on X, Y, and Z axes.

Here is an image of gridlines, similar to which can be expected:



To change the origin of the gridlines, I will introduce an offset to the coordinates of the geometry and translate it to the offset. The offset will be selected by the user. Accordingly, the labels will also be modified.

2.5.2. Add Measurement Capability

Expected due date: June 25
Objective: To enable the user to easily find the positions and distances of various objects.
Deliverables:

- A toggle button for the 3D coordinates.
- User can click on the window and get the scene 3D coordinates of the point clicked.
- The coordinates will be visible near the mouse pointer for a second.
- Another toggle button for the distance between two points.
- User can click on two points and get the distance between them displayed for a second.

I can proceed by adding another UI Menu component. On toggling it, the 3D coordinates will be displayed.

So, I am planning to get the coordinates using ThreeJS raycaster. It will give a set of intersects. Firstly, I will filter out those intersects that are not visible in the scene. This is because raycaster intersects don't consider clipping and return those intersects as well that have been clipped.

Now, I will take the closest intersect and render its coordinates.

To achieve this, there will be two functions defined, first is to check if an intersect is an event data or not, second is to check if it is inside the clipped region or not. So, if an intersect is inside the clipped region, but is an event data, then I will not discard it. Because clipping is only for detector geometry and not for the event data.

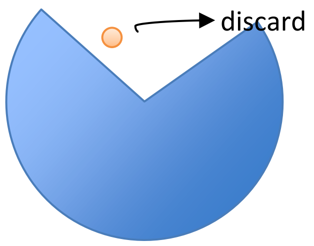
The main implementation of the function will be written in the `three-manager/index.ts`:

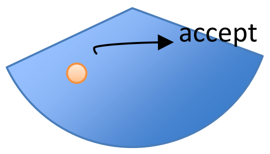
```
public show3DMousePoints(show: boolean)  {
    const camera = this.controlsManager.getMainCamera();
    const scene = this.sceneManager.getScene();
    const raycaster = new Raycaster();
```

A function to check if the clicked object is an event data or not:

```
this.isEventData = (elem) => {
  elem.object.traverseAncestors((elem2) => {
    if (elem2.name == 'EventData') {
      // elem is an Event Data
    }
  });
};
```

A function, `isVisible` will be added to check if the clicked point is a clipped point or not. If the clipped angle is less than 180 deg, then discard the intersect if it is between the clipped region. If the clipped region is greater than 180 deg, then consider the intersect if it is between the visible region.



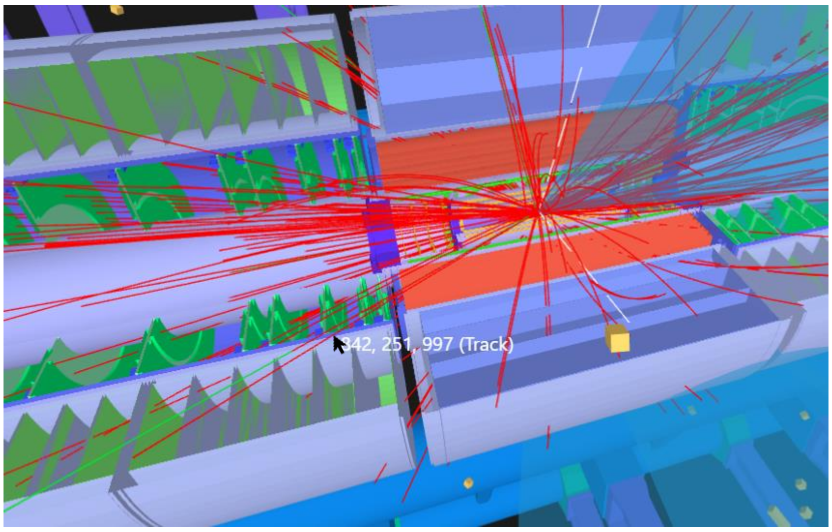


I will then add an event listener to detect any mouse clicks. On each mouse click, it will find raycaster intersects under the mouse pointer and return the closest visible intersect.

The 3D coordinates of the intersected point will be displayed on the screen (at the mouse pointer position) for a second:

```
const coordinates = mainIntersect.point;
```

This is how clicking on a point can display its coordinates:



In a similar fashion, the points clicked can be grouped into two. For example, if points A, B, C, D, E, and F are clicked (in order), then three groups, (A, B), (C, D), and (E, F), can be formed, and the distance between both the points of a group can be calculated and displayed using the distance formula. So, I know the 3D coordinates of A and B, the distance AB will be:

$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

2.5.3. Navigate to Detector Parts

Expected due date: July 10
Objective: Let the user discover various parts of the detector geometry.
Deliverables

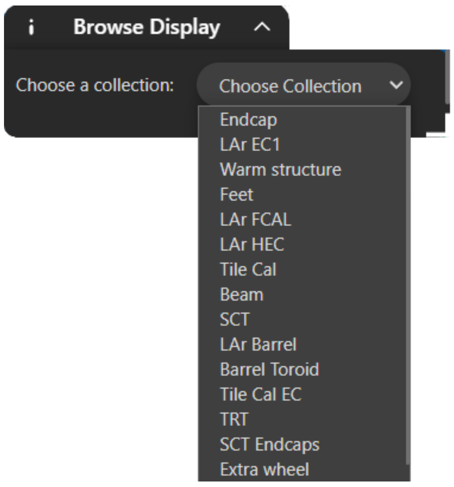
- User can toggle a button to view the collection of geometry parts, such as warm structure, feet, barrel toroid etc.
- On selecting a collection, its elements will be listed.
- There will be two buttons in front of each element.
- Selecting first button will zoom into its bounding box.
- Selecting second button will highlight the detector object.

I will add another component in the UI menu, say `geometry-browser`. It will have a sub-component `geometry-browser-overlay`. Toggling this button will display an overlay containing information about the various parts of the detector. The detector parts are grouped into various collections, for example, SCT endcaps, feet, warm structure, etc. So, each group will have a separate collection, and `getGeometries` function of SceneManager will help to get all the collections. The overlay will give the option to select one of the collections to view its children. The `changeCollection` function will be called after selecting any collection. It will simply extract all the children of the selected collection.

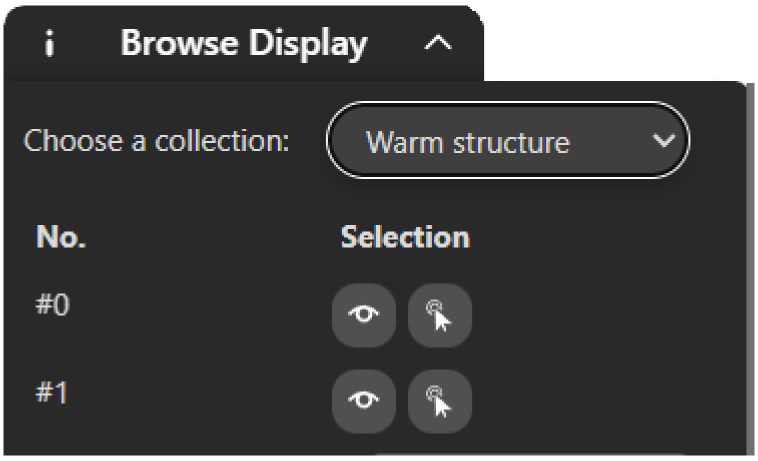
Another function, `highlightObject` will be called when the first button of any detector part will be clicked. This will outline that particular part. It will make use of the `selectionManager` to outline the objects.

`lookAtObject` will be called on clicking the second button whose implementation will be in the `controlsManager`. It will zoom into that particular part.

Collection can be selected from a drop-down list.



All elements of the collection will be listed down.



Information about each component can be displayed further.

2.5.4. Visualise Geometry Dimensions

Expected due date: August 20
Objective: To get an estimate of the size of detector parts.
Deliverables

- User can toggle a button to view the collection of geometry parts, such as warm structure, feet, barrel toroid etc.
- On selecting a collection, its elements will be listed.
- Selecting any element will display its dimensions for a few seconds.

I will extract the dimensions of each detector part and try to render it upon clicking. So, here again, I will list all the components of the detector. On clicking any component, I am planning to display the dimensions of the bounding cuboid. The dimensions will be visible for a few seconds, or the user can toggle to hide them.

2.5.5. Testing and Documentation

Expected due date: September 4
Objective: To test if the new features work well with the existing ones.
Deliverables

- Unit tests for each feature and E2E tests.
- Documentation of the additional features.

I will write unit tests after completing each feature. I will then write E2E tests to check the overall working before merging the updates to the main application. I will follow the documentation along with this and study the test.ts files. This will help me write tests for the new features as well. I will make sure that introducing my changes will not affect any of the existing functions. Finally, I will document the features.

2.5.6. (Optional) Adding Fisheye effect

Expected due date: October 13
Objective: To allow the user to view whole of the event data simultaneously.
Deliverables

- Center of the scene (heart of the detector) will be enlarged and boundaries will be shrunk.
- It will allow user to view whole of the event simultaneously without zooming in/out.

If I fix the camera position, and try to render the whole event with zoomed in center, it is not feasible. Because, if the camera is able to view the boundaries, it implies that it is at a sufficient distance from the heart of the detector. So, even if I try to render the center in an enlarged manner, it will be very blurred.

So, my approach will be to render different portions of the scene using different camera positions. The boundaries will be rendered by keeping the camera at a farther distance. The center of the detector will be rendered by keeping the camera closer.

Although it might seem unfeasible and computationally inefficient, I will explore ways to implement it. Hence, it is kept optional.

2.6. Timeline

Days	Plan
Community Bonding Period	
May 4 – May 28	• Read the documentation thoroughly • Read the details of the codebase • Join Phoenix weekly meetings with mentors

	• Participate in group discussions • Finalize the features
May 29 – June 11	• Add cartesian gridlines to the scene • Make the grid customizable • Make the grid interactive • Make the grid movable (translation) • Label the grid • Write unit tests • Document the features • **Deliverable:** A user-friendly interactive cartesian 3D gridlines to estimate the relative locations of objects.
June 12 – June 25	• Display the 3D coordinates on clicking a point • Display the distance between any two points • Ensure that the gridlines do not have unwanted interference on the 3D coordinates • Write unit tests • Document the features • **Deliverable**: A fully functioning interface which allows clients to see the location of an object in space, and measure distances between object.
June 25 – July 10	• Display a navigation bar to list detector components • Outline and zoom into a component after selecting • Display additional information about each component • Write unit tests • Document the features • **Deliverable:** A navigation service to discover and focus on the various detector components.
Midterm Evaluation	
July 14 – August 20	• Display the dimensions of detector components • Write unit tests • Document the features

	• **Deliverable:** A handy feature to find dimensions of detector components.
August 21 – September 4	• Write E2E tests • Remove any extra code • Try to reduce the size of the codebase by using DRY principle • Optimize the code to result smoother displays • **Deliverable:** End to end testing + optimize and shorten the code.
Initial Results Announced	
September 4 – October 13 (Excluding Mid semester examinations)	• If possible, explore ways to add a fisheye effect to the scene so that the heart of the detector can be clearly seen along with the boundaries • Write unit tests • Document the feature • **Deliverable:** Add a fisheye effect to the scene
October 14 – November 6	• All unit tests and E2E tests will be run and passed • Finally receive confirmation from mentors and document the functions • Clean up the documentation written previously
Final Evaluation	

3. Commitments

I am flexible with my working hours and can work during the night time as well. This summer, I have no commitments as such. My summer vacation will start from May till July, and I can devote 5-6 hours per day during that time. From July onwards, my college will resume, but I will manage it along with my project. I will be having my mid-semester examinations in September. Also, I check my emails at least twice a day. So, communication can be done through mail and my response can be expected within less than 24 hours.

4. Contributions

4.1. Evaluation tasks:

I loaded a GLTF file in ThreeJS. It had two parts. Clicking each part will outline it and display its color on the bottom left.

Link: Basic ThreeJS evaluation

Next task was to add cartesian gridlines to the center of the detector, apart from the already existing eta-phi gridlines. I added the cartesian in 3D.

Link: Added cartesian gridlines to the scene

4.2. Merged PR:

When the “view overlay” option in the UI Menu is toggled, the orthographic view is displayed in an overlay window. But on resizing the window, some extra space was left. So, I fixed this by keeping the aspect ratio of the window fixed while resizing.

Link: Fixed overlay resizing issue

5. Post GSoC

Post the GSoC period, I would love to continue contributing to this project and implement more features that may not have been covered in GSoC in the short time interval. I will also try to provide consistent patches and bug fixes to the project. I will also try to contribute to the community by taking part in development activities other than the scope of this project also.