N-Lateration

Problem Statement

GPS-based positioning inside buildings lacks precision, making it difficult to accurately determine the location of a device (e.g., a smartphone).

Proposed Solution: N-Lateration

N-Lateration is a technique that estimates the position of a device by calculating its distances from multiple known emitters, such as Wi-Fi access points or beacons.

N-Lateration Principle

N-Lateration follows an **N+1 principle**, meaning:

- **In 2D space**: At least **3 emitters** are required.
- **In 3D space**: At least **4 emitters** are needed.

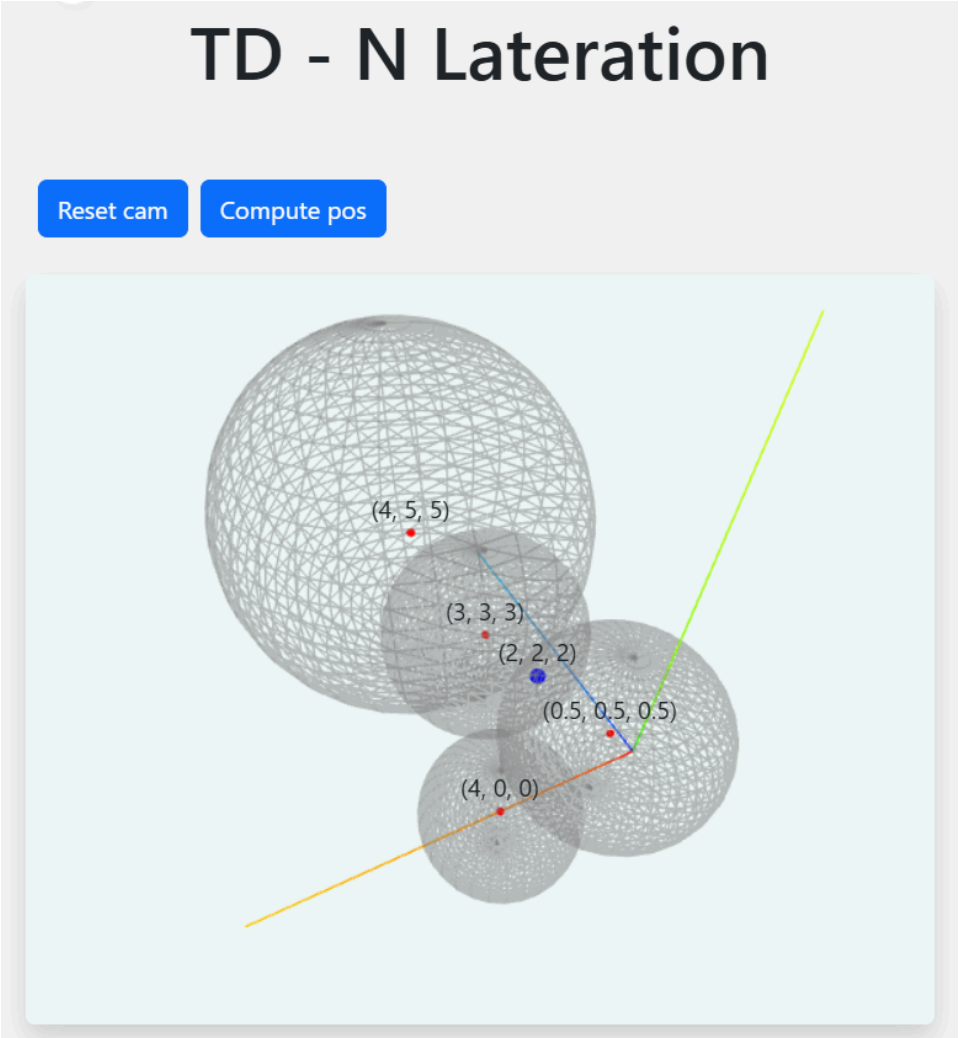
Potential Issues

- **Obstacles** can interfere with accurate distance calculations.
- **Environmental factors** may affect measurement precision.
- **Large areas** require a high number of emitters to maintain accuracy.

Advantages

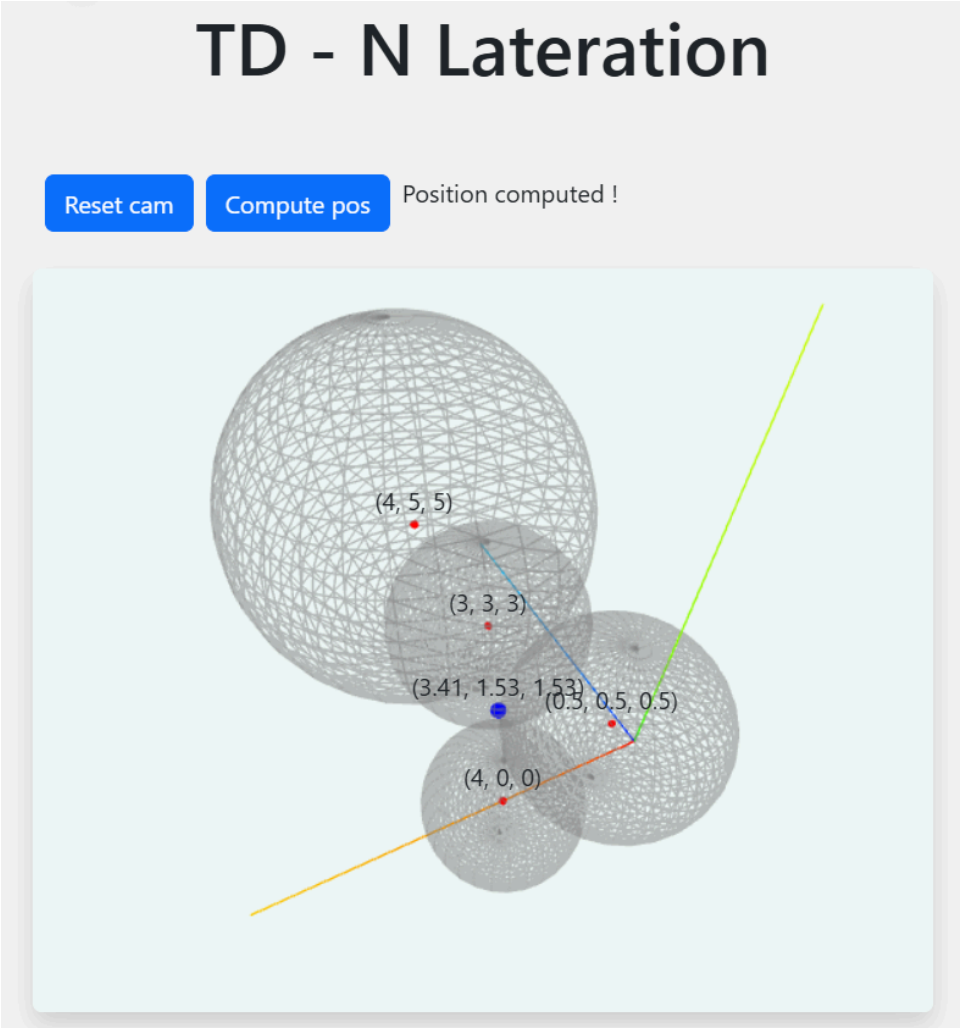
✔ Can be deployed anywhere, with no need for initial data.
✔ Easily scalable by adding new emitters.
✔ Works well in open environments.

N-Lateration relies on **N+1 emitters** in a given area, using the measured distances between the phone and each emitter to determine its exact coordinates.



Since the required number of emitters depends on the **dimension** we are working in, we use **4 emitters in a 3D space** (representing four spheres made of wireframes).

The **blue point** represents the phone's initial random position. When we press **"Compute Position,"** the phone is repositioned to its correct location based on the calculated distances from the four emitters.



In this second screen we can see that the phone moved to be correctly placed in between all the different emitters

Explanation of the Technologies Used

Our application was developed using Vue.js, a progressive JavaScript framework that enables the creation of interactive and modular user interfaces. Vue.js facilitates component management and communication, allowing us to structure our application in a clear and efficient way.

For 3D rendering, we used Three.js, a JavaScript library that simplifies the creation and manipulation of 3D scenes in the browser. Three.js is built on WebGL (Web Graphics Library), a low-level graphics API that leverages GPU acceleration for rendering 2D and 3D graphics in web browsers. WebGL provides optimized performance by using the underlying OpenGL ES, enabling the rendering of complex objects with advanced lighting and texturing computations. Thanks to Three.js, we were able to create and manipulate 3D objects, manage the camera, and add dynamic interactions.

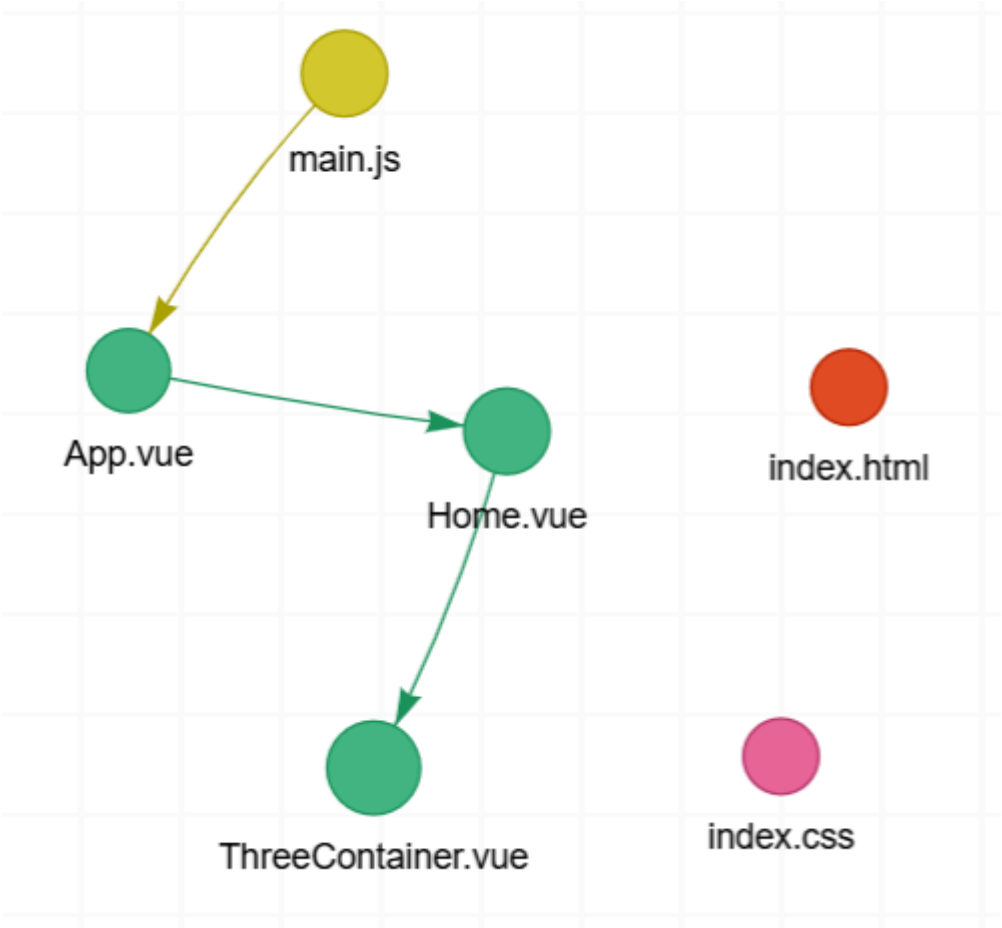


Diagram of the app files

App diagram explanation

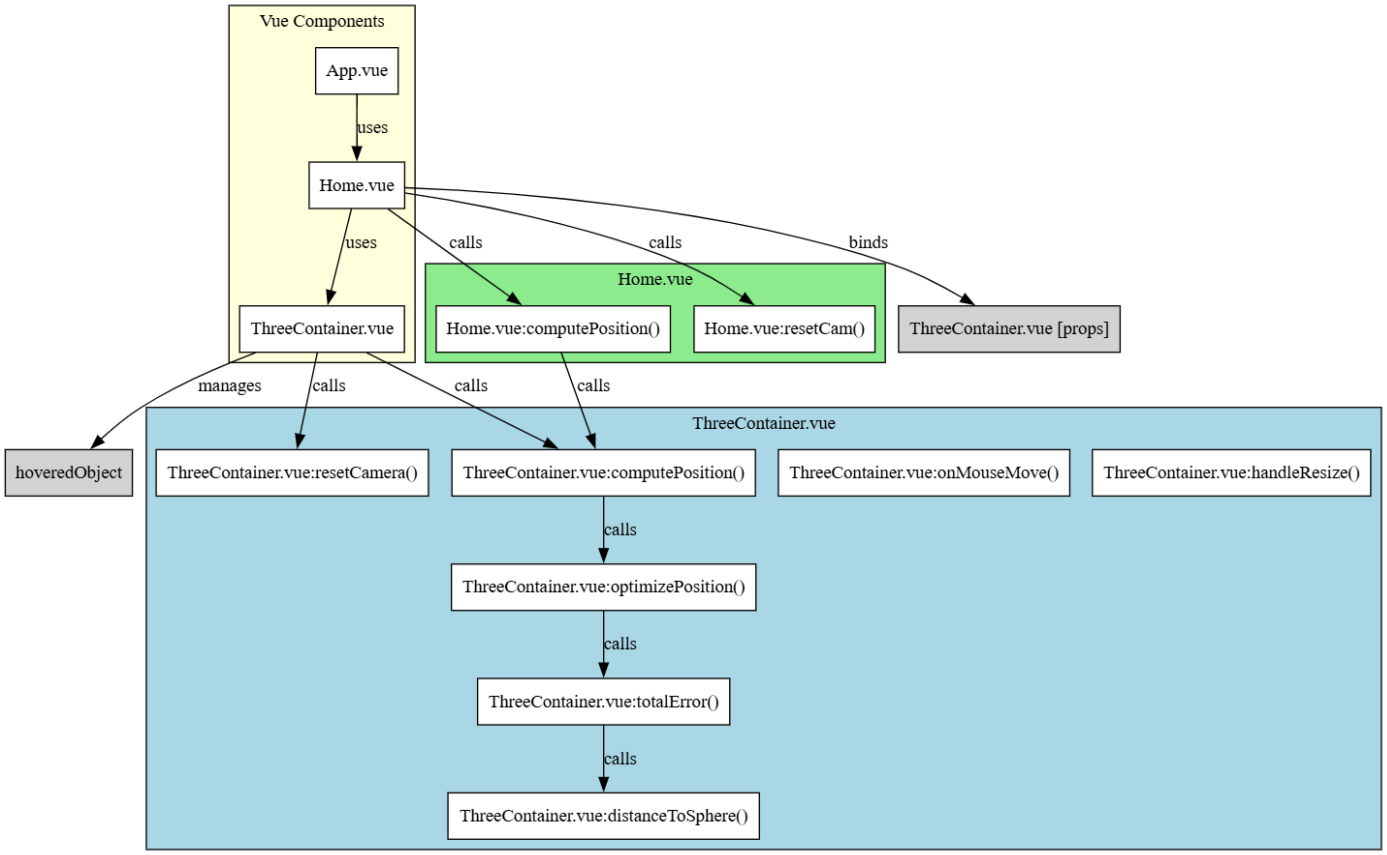


Diagram of the web app

1. Vue.js Components

- `App.vue` is the entry point of the application and uses `Home.vue`.
- `Home.vue` is the main component that integrates `ThreeContainer.vue` for managing 3D visualization.

2. Functions in `ThreeContainer.vue`

- `resetCamera()`: Resets the camera position.
- `computePosition()`: Computes the estimated position of an object.
- `onMouseMove()`: Handles mouse movements to detect interactions.
- `handleResize()`: Adjusts rendering when the window is resized.
- `distanceToSphere()`: Calculates the distance between a given point and a reference sphere.
- `totalError()`: Computes the total error based on distances to reference spheres.
- `optimizePosition()`: Optimizes the estimated position by minimizing total error.

3. Methods in `Home.vue` and Their Interactions

- `resetCam()`: Calls `resetCamera()` from `ThreeContainer.vue` to reposition the camera.

- `computePosition()`: Triggers `computePosition()` from `ThreeContainer.vue` to estimate the position.

4. Relationships Between Components and Properties

- `Home.vue` is linked to the props of `ThreeContainer.vue`, allowing it to access and modify them.
- `ThreeContainer.vue` manages `hoveredObject`, a variable used to track interactive objects.