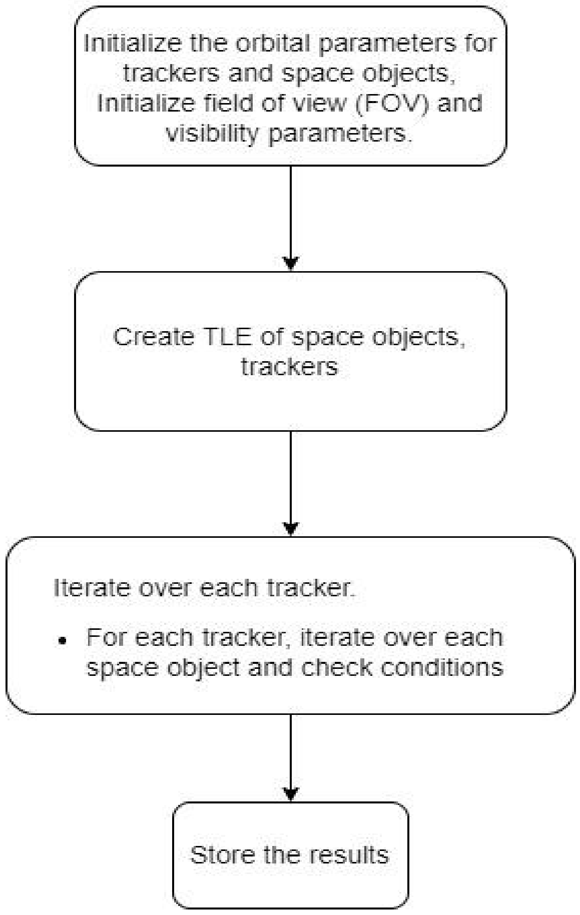
Flowchart


Initialize the orbital parameters for trackers and space objects, Initialize field of view (FOV) and visibility parameters.
Create TLE of space objects, trackers
Iterate over each tracker.
• For each tracker, iterate over each space object and check conditions
Store the results

<u>Algorithm for performing the computation</u>

1. **Initialization**:

- Define the orbital parameters for trackers and space objects.
- Define field of view (FOV) and visibility parameters.

2. **Data Preparation**:

- Create a list of TLEs for space objects.
- Create a list of TLEs for trackers.
- Create a list of observation times for the analysis period.

3. **Analysis**:

- Iterate over each tracker
    - For each tracker, iterate over each space object
        - For each space object, calculate positions at each time step.
        - Check if the space object is within the FOV and distance limit.
        - Check if the space object is sunlit.
        - Record crossing and visible events.

4. **Output**:

- Store the results and visualize if necessary

There are different ways to increase the efficiency of computations. Some of them are Parallel computing where it distributes the workload across multiple CPU cores. For example one of the tracker will track each of the objects, at the same time another tracker will also track objects simultaneously in a parallel manner. All of the trackers could be used in this parallel manner according to the number of CPU cores available and thereby parallel results could be obtained in a faster time rate. This can significantly speed up the computations. Libraries like *multiprocessing* or *joblib* in Python can help distribute the workload across multiple CPU cores.

Other ways are to use vectorized computations for vectorized operations to speed up the mathematical computations involving large arrays for position and distance calculations. In such cases NumPy could be used to perform it. The number of time steps could be reduced by increasing the interval wherever possible and only use finer time steps during close approaches or within critical time windows. By doing so we could eliminate the redundant calculations and precise calculations could be implemented only at critical cases.