# Satellite Collision Detection using Spatial Data Structures

Christian Hellwig*, Fabian Czappa*, Martin Michel†, Reinhold Bertrand†‡, Felix Wolf*
* Technical University of Darmstadt, Department of Computer Science, Germany
† Technical University of Darmstadt, Institute of Flight Systems and Automatic Control, Germany
‡ European Space Agency ESA/ESOC
hellwig.christian@gmx.de, {fabian.czappa, felix.wolf}@tu-darmstadt.de
michel@fsr.tu-darmstadt.de, reinhold.bertrand@esa.int

***Abstract*—In recent years, the number of artificial objects in Earth orbit has increased rapidly due to lower launch costs and new applications for satellites. More and more governments and private companies are discovering space for their own purposes. Private companies are using space as a new business field, launching thousands of satellites into orbit to offer services like worldwide Internet access. Consequently, the probability of collisions and, thus, the degradation of the orbital environment is rapidly increasing. To avoid devastating collisions at an early stage, efficient algorithms are required to identify satellites approaching each other. Traditional deterministic filter-based conjunction detection algorithms compare each satellite to every other satellite and pass them through a chain of orbital filters. Unfortunately, this leads to a runtime complexity of $O(n^2)$. In this paper, we propose two alternative approaches that rely on spatial data structures and thus allow us to exploit modern hardware's parallelism efficiently. Firstly, we introduce a purely grid-based variant that relies on non-blocking atomic hash maps to identify conjunctions. Secondly, we present a hybrid method that combines this approach with traditional filter chains. Both implementations make it possible to identify conjunctions in a large population with millions of satellites with high precision in a comparatively short time. While the grid-based variant is characterized by lower memory consumption, the hybrid variant is faster if enough memory is available.**

***Index Terms*—parallelization, spatial data structures, conjunction detection, orbit, space safety, space debris**

## I. Introduction

In recent years, the number of satellites in Earth's orbit has increased rapidly. While the past decades have been characterized by the use of space mainly by government agencies or military and single commercial operators, lower launch costs and new business applications have led to an increased number of new satellite operators. The largest private operator to date is *SpaceX*[1] deploying so-called mega-constellations to enable worldwide satellite-based internet. SpaceX has already sent several thousand *Starlink* satellites into orbit and has submitted applications for at least 30,000 more [1]. Other operators like *Planet Labs*[2] use a large fleet of nearly unmaneuverable *CubeSats*—small satellites with standardized sizes and interfaces to generate commercial Earth observation data with a near-global coverage and minimum revisit-time. These two examples show the rapidly changing satellite population, especially in low Earth orbit (LEO), which creates new challenges for satellite operators to ensure safe and efficient operations. Figure 1 shows this evolution in terms of artificial objects sent into low Earth orbit compared to the beginning of the space age in the 1960s.

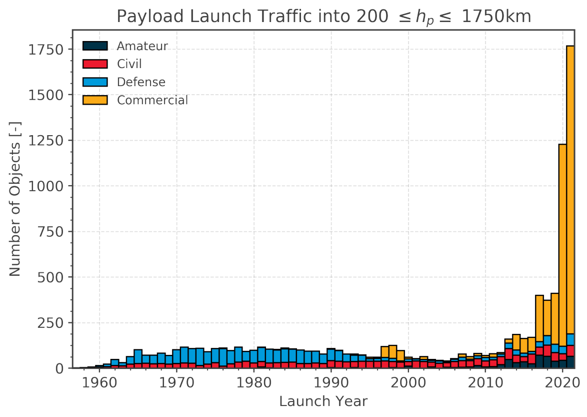


Fig. 1: Evolution of the number of payloads launched into the low Earth orbit (height of perigee $h_p$ between 200 km and 1750 km) by mission funding [2].

With every satellite launched, the number of objects in space increases. Besides the satellite itself, this includes, for example, parts of the launcher and mission-related objects, like clamps, shell structures, or bolts, which are released into the orbital environment. Moreover, satellites at their end of life or in case of malfunctions significantly contribute to the growing space debris problem if a correct de-orbiting could not be achieved.

To date, space surveillance sensors can track more than 30,000 objects larger than approximately 10 cm in Earth's orbit [2]. About 5000 of these objects are operational satellites, of which around 1800 have been launched just in 2021 [3]. With new sensors that will be put into operation in the following years, the extension of the catalog to more than 1,000,000 traceable objects is expected [4]. Based on simulations and in situ measurements, the number of untrackable objects (too small) is expected to be around 130 Million [5].

[1]https://www.spacex.com
[2]https://www.planet.com

A collision of these objects with other satellites or debris could lead to a chain reaction. Even with the smaller objects, each collision creates more debris, increasing the likelihood of further collisions [6]. Such a development would have enormous negative consequences for space operations' safety and economy. Previous incidents, like the collision of the Chinese satellite Yunhai 1-02 with the remains of a Russian rocket in March 2021, show the severity of this risk [7].

Therefore, it is essential to detect *conjunctions*, i.e., satellites coming close to each other or existing debris objects, at an early stage and initiate suitable collision avoidance maneuvers. Due to the increased number of objects in Earth's orbit, current deterministic algorithms for conjunction detection are reaching their limits. As these algorithms typically perform a pairwise comparison of the orbits of all objects in space ("all-on-all"), the process results in the consideration of a quadratic number of satellite pairs. The resulting pairs of satellites are then successively excluded through a series of filters, which improves the actual runtime but does not solve the fundamental problem of quadratic complexity.

In this paper, we propose two approaches that rely on spatial data structures to overcome the quadratic number of comparisons and exploit parallelization methods. We first introduce a purely grid-based variant relying on non-blocking atomic hash maps to identify conjunctions. Additionally, we present a hybrid method that combines this idea with classical orbital filter chains.

Both implementations allow for identifying conjunctions in a large population with millions of objects (satellites or debris particles—the implementation can deal with both) in a comparably short amount of time with high precision. While the grid-based variant is characterized by lower memory consumption, the hybrid variant is faster if enough memory is available. Overall, we see our contribution as the following:

- Implementation and validation of an orbital conjunction detection algorithm using spatial data structures.
- Implementation that achieves a maximum speed up of 19x for the grid based variant and a maximum speed up of 14x for the hybrid variant with respect to a single thread.
- Implementation of a GPU Kepler Solver.
- While the complexity does not improve in the worst case, we see an improvement from quadratic to linear time complexity in the best case. For the average case, we remain in the same general complexity class, however, we see a significantly better scaling behavior.

## II. Related Work

Contrary to classical N-body simulations that calculate the gravitational forces between the objects, e.g., sub-atomic particles or celestial bodies, we can neglect the forces between the simulated objects due to the high speed of the objects in comparison to their masses. This simplification, however, also disqualifies us from using well-established auxiliary methods such as the Barnes–Hut algorithm [8] for the Fast Multipole Method [9].

Software tools for all-on-all conjunction detection applications are typically based on topological methods, spatial partitioning, or a combination of both [10] [11]. Topological methods usually encompass a series of sequential filters that compare the orbits and positions of the individual objects and successively exclude object pairs that cannot generate a conjunction [12]. The *apogee/perigee filter* takes the farthest (apogee) and nearest point (perigee) of an orbit and compares the range between with the respective range of all other objects, excluding those as potential collision pairs that do not overlap [13]. The *orbit path filter* further reduces the number of object pairs by calculating the minimal distance between the two orbits. The pairs are excluded if this distance is larger than a predefined threshold, which considers several orbit and position uncertainties [13]. Several other geometric filters have been defined to further reduce the number of relevant pairs [14]. By calculating the *true anomaly window* around the intersection line of the two orbits, it is possible to apply a *time filter* that takes the actual position of the two objects into account. It excludes all object pairs that are not in these windows simultaneously and can, therefore, not generate a conjunction [15]. The *sieve method* [16] and *smart sieve method* [17] include an additional set of filters that compares the propagated Cartesian coordinates of two objects at two different points in time and derives if the trajectories overlap between these two points.

Volumetric approaches are typically used in long-term simulations [11] [18] [19], employing spatial partitioning for the statistical analysis of conjunctions. For the *Flux*-based approach [20], the space is divided into several "bins", and the intersections of each orbit with these volumes are calculated. Therefore, each object can be assigned to multiple volumes with a specific probability based on the residence period. The spatial object density in each volume can be derived for statistical analysis. The *Cube*-method [21] divides the space into quadratic volumes and uses randomized object positions on their orbits to fill the volumes. Unlike the previously presented filters, the volumetric approaches have a runtime complexity linear in the number of objects. However, they can not be used to generate deterministic conjunctions due to the stochastic approaches and are not suited for the simulation of large satellite constellations [22].

For better performance of the conjunction detection process, several concepts for parallelization have been examined, which are based on slicing the time span of the simulation [23], dividing the object population [24], or the parallel application of independent filter steps [25]. GPU-based approaches have been mainly used to accelerate the propagation of the satellite position [26] [27] or in the context of conjunction detection to adapt the classic filter chains [28].

Two alternative approaches for conjunction detection are presented in [29] and are based on Kd-trees and spatial hashing. For the first approach, the minimum and maximum positions of the satellites in their orbit are calculated and inserted into the Kd-tree so that all overlapping satellites can be determined. Alternatively, spatial hashes are derived from the extreme values so that satellites likely to overlap can be identified. However, both methods have their drawbacks. Building the Kd-tree for every step is tedious, so they use relatively few time steps and can produce a lot of false positives, requiring more filters afterward and thus cannot be used alone. Their spatial hashing method also uses large time steps and needs a correcting bounding box containing all possible grid cells, which they try to trim down. These operations involve expensive computations, which slows their approach down significantly.

Spatial data structures are used to speed up simulations in nearly every discipline. Primarily used in computer graphics [30] [31], it also sees uses in areas such as surgery planning [32], particle packing [33], land surveillance [34] and other geosciences [35], and many more.

When implementing spatial data structures such as grids, concrete memory representation often matters. For sparsely-filled grids, hash maps are a good choice as they require a memory footprint that is only a constant factor larger than the actual filling. Combining hash maps with parallelism is a broad subject, for example, with CUDA [36] [37], MPI [38], or general multi-threading of the CPU [39].

## III. Approach

In orbital conjunction detection, we are interested in the so-called *Point of Closest Approach* (PCA) and the *Time of Closest Approach* (TCA). A PCA is a local minimum of the distance between two satellites in orbit, and the TCA is the time when the PCA occurs. On the one hand, a satellite pair can have multiple PCAs and TCAs depending on their orbits and the simulated period. On the other hand, the PCAs of some pairs might be so large that they are not relevant, as there is no actual risk of collision.

Each pair of satellites—regardless of how far apart the satellites are—has at least one point where they are closest on their orbits. As the objects are subject to constant perturbing forces, and, therefore, their orbits are constantly changing, the exact positions of both satellites are unknown to the operators and *Space Situational Awareness* (SSA) data providers. The magnitude of the uncertainty depends on the time and the accuracy of the individual objects' last position measurements by SSA-sensors or on-board systems and could be estimated during the orbit determination process. The goal of a typical conjunction screening scenario is to identify all potential collisions between the objects, which will then be further analyzed by the operator as part of a more detailed subsequent conjunction assessment process. During the screening phase we use a uniform *screening threshold*, which size should include the largest typical uncertainties. All encounters with a minimal distance below this threshold are considered for further assessment, while those above are discarded (see Figure 2), as a risk of collision could be ruled out due to the large distance.

Our approach for the screening phase uses a spatial data structure—a *grid* that subdivides the simulation space into subspaces (cells) of equal height, width, and length, depending on the screening threshold—to speed up the conjunction detection. We insert the satellites into the grid in parallel and thus know when to check whether a conjunction between two satellites occurs. However, because we do not know the satellite's exact position inside the cell, we also check all $3^3 - 1 = 26$ neighboring cells. This covers the cases where two satellites are close to each other but in different cells. In this paper, we present two variants, one is based exclusively on the grid and a hybrid variant that uses the grid as a filter before examining the satellites with classical orbital filters, e.g., checking if orbits are coplanar. Both sample the total simulation period in equidistant steps, insert the satellites into the grid, and check for conjunctions. If the grid-based variant identifies a pair of satellites that might cause a conjunction, they are examined in detail to find their PCA and TCA. This variant requires comparably small grid cells to avoid having to check too many pairs. In turn, this requires propagating the position on the orbit in small steps so that no satellite skips boxes during the simulation. The hybrid variant passes all identified pairs to the classical orbital filtering methods. The additional checks reduce the number of pairs we have to examine for their PCAs and TCAs, so we sample less frequently. Consequently, this allows us to use larger cells. However, it has to check more pairs per sample step, effectively trading time for space.

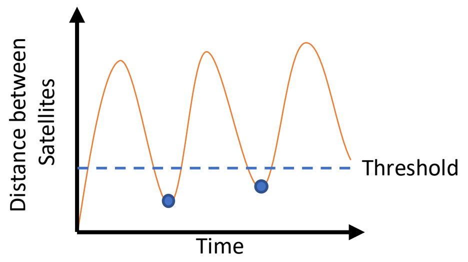


Fig. 2: Distance between two satellites over a period of time. The blue dots symbolize the local minima at which the satellites come closest to each other. The blue dashed line corresponds to a screening threshold under which a conjunction should be detected. The time of a local minimum is a TCA. The distance at this point is the respective PCA.

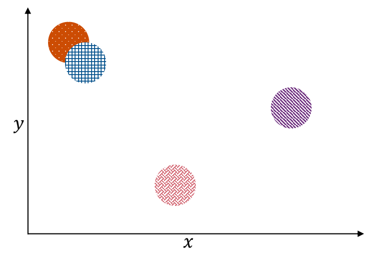
Both conjunction detection implementations share three major steps. However, the hybrid variant requires an additional step for the orbital filters. The structure of our approach is as follows:

1) Memory allocation to store the initial satellite position data, the grid structures, and results (once at the start).
2) Parallel propagation of the satellite positions, parallel insertion into the grid, and parallel identification of potential colliding satellite pairs.
3) (Hybrid variant only) Application of the orbital filters to reduce the number of potentially colliding pairs.
4) Determination of the TCAs and PCAs.

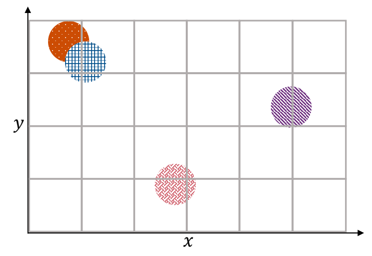
### A. Grid

A *grid* is a simple spatial data structure that divides a geometric space into uniform cells (see Figures 3a and 3b). It suits the problem of detecting conjunctions in space well because all objects have the same size (approximately; compared to the size of a cell). We calculate the cell size by considering the worst-case scenario for our method. It occurs when two satellites are at the edge of their cell, but the two cells are not neighbors, such that they are separated slightly more than the screening threshold (see Figure 4a). In the next sampling step (see Figure 4b), the actual undercut of the threshold that would occur is skipped (see Figure 4c). To circumvent this, the cell size $g_c$ (in km) is based on the screening threshold $d$, the typical speed of a satellite in LEO (7.8 km/s), and the seconds between the samples $s_{ps}$:

$$g_c = d + 7.8 \cdot s_{ps} \tag{1}$$



(a) For $n = 4$ objects we have to perform $4 \cdot (4 - 1)/2 = 6$ collision tests to check all (unordered) pairs of objects.



(b) The space is divided into cells. We can easily determine that two objects overlap, while the other two are far apart. Thus, we reduce the number of collision tests from 6 to 1.

Fig. 3: Collision detection example with and without spatial partitioning.

*1) Insertion of Satellites:* Depending on the memory available, we can calculate several sample steps simultaneously. Each step is considered separately and requires its own grid, however, the insertion of the elements into this grid is easily parallelizable. For doing so, a thread propagates a satellite along its orbit based on the Kepler elements. We have divided the simulation space into Euclidean space rather than by respective Kepler elements, so the thread needs to convert the position into the usual three-dimensional Cartesian coordinates. Lastly, the thread calculates the grid cell based on the Cartesian coordinates and inserts the satellite therein. Consequently, the grid cells must be able to handle more than one satellite.

*2) Conjunction Detection:* After all elements have been inserted into the grid, we perform the actual conjunction detection, where we can check each cell in parallel. Whenever a cell is not empty, each satellite within the cell makes up a pair for further conjunction detection with every other satellite in that cell or the $3^3 - 1 = 26$ neighbor cells. We save all pairs employing the satellites' ids and the sampling step. This helps to prevent considering possible conjunctions twice (from the point of view of both satellites), however, it allows multiple conjunctions at different sampling steps. The purely grid-based variant calculates the PCA and TCA for each such pair. The hybrid variant passes these pairs to the classical orbital filters for processing and only then calculates the PCAs and TCAs.

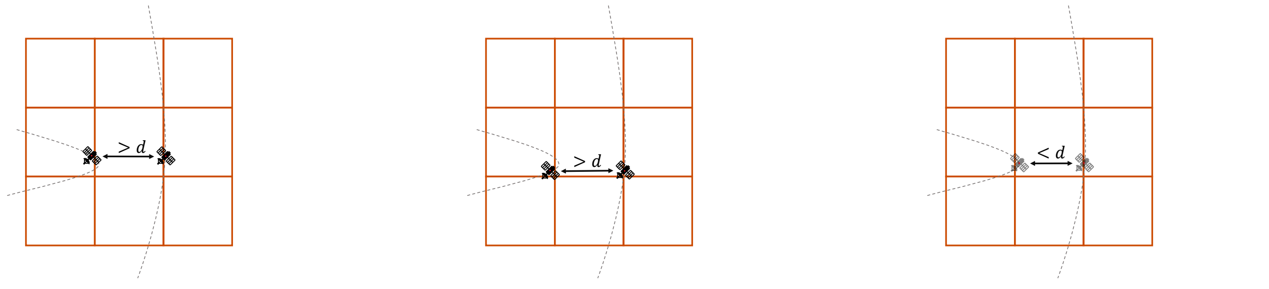
## B. Complexity

For the complexity analysis of our approach, we look at the number of times two satellites have to be checked for their PCA and TCA. As is often the case with auxiliary methods, our approach shares the same worst-case complexity as the previous algorithm. If all satellites were simultaneously at the same point in space, both the grid-based and hybrid variants would have to compare every satellite with every other one, resulting in quadratic complexity. This, however, does not occur in practice: Even though a few orbits are populated by a larger number of satellites (geostationary orbit, Sun-synchronous orbit), the positions of the individual satellites are distributed across this orbit, such that a conjunction is highly unlikely. In the extreme case of a catastrophic fragmentation event, the generated space debris objects will start at one point in space. But even then they will immediately spread across the orbit due to different initial velocities and be distributed over the whole orbital shell due to perturbing effects on their orbit.

However, in the best case, all satellites are far away from each other. This way, every grid cell is occupied by at most one satellite, resulting in zero calculations of PCA and TCA. Thus, our approach has linear complexity (each satellite still has to be inserted), whereas a naive examination of all pairs would have quadratic complexity.

For the average case analysis, we use some approximations. Firstly, we assume that all satellites have near-circular orbits, i.e., their eccentricity is close to zero. Secondly, we assume that satellites with close orbits traverse the same number of cells per orbital period. Figure 5 shows a 2D visualization of our following argument. The satellite orbits have a relative high eccentricity to showcase the maximum number of orbit intersections.

We pick a sequence of increasing radiuses: $r_0, r_1, \ldots, r_k$, along which we divide the simulation space. That is, we consider the hollow spheres $S_1, \ldots, S_k$, where $S_i$ has inner radius $r_{i-1}$ and outer radius $r_i$. The total number of satellite pairs we have to check is the sum of the satellite pairs over all hollow spheres, where we ignore inter-hollow sphere pairs (using the first approximation). Now, we assign all satellites to a particular hollow sphere based on the height of their orbit, splitting the total number of satellites $n$ into $n = n_1 + n_2 + \cdots + n_k$. Within a hollow sphere, two satellites can produce at most two conjunctions per orbital period (we ignore co-orbiting objects; for orbits, the Earth is always at



(a) At time $t_0$, the satellites are located at the outer edge of the middle cell. They are slightly further apart than the screening threshold $d$.

(b) At time $t_1$, the satellites are again at the outer edge of the middle cell. They are again slightly further apart than the screening threshold $d$.

(c) Between the time points $t_0$ and $t_1$, the distance between the satellites was below the screening threshold $d$. However, due to a too large time step this point in time was skipped.

Fig. 4: Worst case analysis within a grid over two time steps. Both satellites are at the edge of a cell and the cells are not neighboring cells.

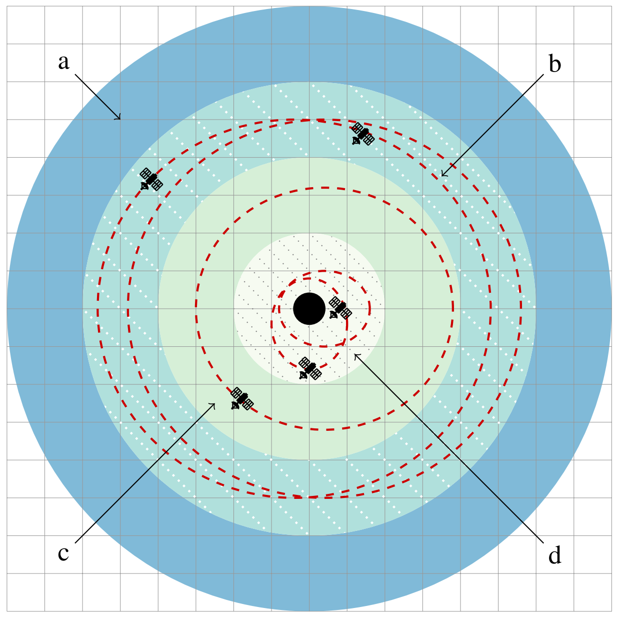


Fig. 5: This illustrates the hollow spheres from the arguments in Section III-B as 2D cuts. For better visibility, the satellites' orbits are red dashed ellipses. Within a hollow sphere, two nearly circular orbits can intersect at most twice because the earth lies in one of the focal points. Sorting the orbits into their respective hollow spheres shows that there are two intersections in **b**, two in **d**, and none in **a** and **c**.

a focal point [40]). As the satellites are in only one grid cell at a time, we can give a bound on the number of a satellite's conjunction per orbital period as $2 \cdot n_i \cdot b_i^{-1}$. Here $b_i$ is the number of grid cells on a satellite's orbit in $S_i$ (if the radiuses are close enough, this can be considered a constant using the second approximation).

In total, the average number of pairs we have to check per simulation step is in $O(\sum_{i=1..k} n_i \cdot (2 \cdot n_i \cdot b_i^{-1}))$. This shows that the number of possible conjunctions still grows quadratically within a hollow sphere with the number of satellites. However, we can ignore all pairs of satellites that reside in different hollow spheres, and we utilize the spatial location of the satellites within each hollow sphere to further reduce the number of conjunctions.

## IV. Implementation

In this section, we highlight different aspects of our implementation. We give details for the implementation of the grid and how we propagate the satellites, and, for all determined pairs, how we calculate their PCA and TCA. Furthermore, we provide the associated code as open source.

### A. Grid

Several factors influence the underlying data structure we use to represent the grid. The two most significant factors are the geometric space we need to simulate and the number of objects within the grid. The simulation space must be at least $(85{,}000\,\mathrm{km})^3$ to represent the entire space up to the geostationary orbit. At the same time, there are only a few artificial objects that we need to keep track of, compared to the extremely large size of this space. Therefore, simple data structures like a three-dimensional array where each item corresponds to a grid cell are not practical. In principle, such a grid representation would fit into memory, however, it is important to note that we have to consider multiple sampling steps. As a single grid reflects only one point in time, it is desirable to calculate as many grids as possible in parallel. For this reason, such memory-intensive representations are unsuitable. Furthermore, if we used three-dimensional arrays, we had to erase the content for every iteration.

Memory sparse representations like *compressed sparse matrices* or *hash maps* are more suitable because they only store non-empty cells. Thus, no storage place is wasted by the abundance of empty cells. However, they come at the downside of a higher computational cost when inserting elements. Nevertheless, grids (e.g., in the form of hash maps) are superior to data structures such as octrees or Kd-tree. These must be recreated each time an object moves, requiring higher computational cost at each iteration.

We use a fixed-size hash map as the underlying data structure because the assembly of a grid with sparse matrices can only be done iteratively. Thus, we can quickly calculate the memory location using a hashing procedure. A remaining disadvantage is that memory coalescing techniques no longer function optimally due to the hashing. These techniques bundle memory accesses of closely located memory cells to ensure maximum bandwidth. However, hashing makes it unlikely that two neighboring cells will be adjacent in memory. Thus, the memory throughput is limited. Fortunately, the bandwidth of modern hardware is very high, so we prefer the hash map implementation over compressed sparse matrices.

*1) Construction:* We use the fast *MurMur3*[3] hash for calculating the position of a grid cell. To ensure that the value range of the slot does not exceed the size of the hash map, apply the modulo operation to the hash value as well.

The simplest method to resolve a collision in the hash map is linear probing. If a collision occurs, a new index is created by incrementing the last calculated index $s_i(x)$ of the key $x$ by one and applying the modulo operation with the size of the hash map $M$ to prevent that indices are larger than the hash map's capacity:

$$s_{i+1}(x) = s_i(x) + 1 \mod M \qquad (2)$$

Linear probing generally fills every memory location of the hash map, however, by doing so it dents so form *cluster*—long chains of occupied slots. Such a cluster increases the insertion time because the recalculation of the hash map position has to be done frequently.

We rely on atomic operations to ensure thread-safe access to the hash map. Each slot of the hash map consists of a pair of the key from which the slot was calculated and corresponding additional fields representing the data. As a memory location can never be truly empty, we use the maximum of a 64-bit value as a unique value that indicates an empty slot. The entire memory area of the hash map is initialized with this value at the beginning to threat the whole hash map as empty.

*2) Insertion:* When inserting a satellite into the grid, we use the atomic compare-and-swap (CAS) operation to check whether the slot is empty and to replace it with a new value if this is the case. After indicating that this memory location is now occupied, we set the value within the slot to the satellite in question. If, however, the slot is not empty, the check will fail and return the stored key. If this key indicates another grid cell, we encountered a hash collision and calculate a new slot with linear probing. Otherwise, we have found another satellite within the same cell. We build a singly-linked list of satellites within one cell, that is, an entry of a satellite contains, among other things, the satellite's identifier, the satellite's position, and a pointer to the next entry (possibly *null*). See Figure 6 for a depiction of the hash map and its entries. Each satellite produces exactly one of these entries, so we can allocate them in advance and just set the pointers to the next entry dynamically.

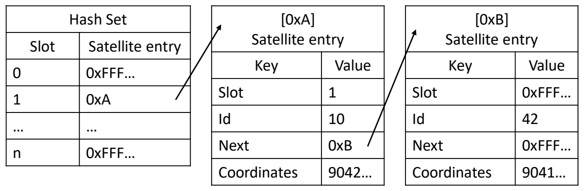


Fig. 6: Partially filled hash set. The entry at slot 1 points to a satellite entry that represents the content of a grid cell by means of a singly linked list. To build up the linked list, each satellite entry contains a next field that points to further satellite entries.

*3) Conjunction Detection:* We examine all non-empty slots of the hash map in parallel for the conjunction detection. We calculate the individual cell's position for each such slot, look at all satellites from the neighboring cells and extracting their Euclidean positions. Afterward, we insert all pairs of satellites with at least one satellite in the original cell combined with the sampling step in one conjunction hash map. In the hybrid variant, we pass these pairs to the orbital filters to further reduce the possible conjunctions. Then, we calculate the PCAs and TCAs of each (remaining) pair.

## B. Propagation

The satellite propagation describes the determination of the future position of a satellite, which uses the *Kepler elements*. These six values (*semi-major-axis, eccentricity, inclination, longitude of the ascending node, argument of perigee, true anomaly*) describe the shape and the position of an orbit, as well as the position of a satellite (see Figure 7 and Figure 8). The future position of the satellite is determined by the computationally intensive recalculation of the true anomaly as a function of time.

In our implementation, we use a modified version of the high-performance Contour Kepler solver [43]. The provided CPU C++ code[4] in the associated publication of the Contour Kepler solver is designed to compute several true anomalies for a single satellite. Due to the computation of many true anomalies, the reference implementation reuses partial computations. On a graphics card, it makes sense that each individual thread can calculate a true anomaly on its own. By splitting the reference implementation into independent parts, these partial calculations are no longer available. We can compensate for this by either recalculating these values at each calculation of the true anomaly or by precalculating the reusable parts independently once and then storing them in the global graphics memory. As we have to perform this calculation for each satellite at each time step, we store the data in memory.

[3]https://github.com/aappleby/smhasher

[4]https://github.com/oliverphilcox/Keplers-Goat-Herd

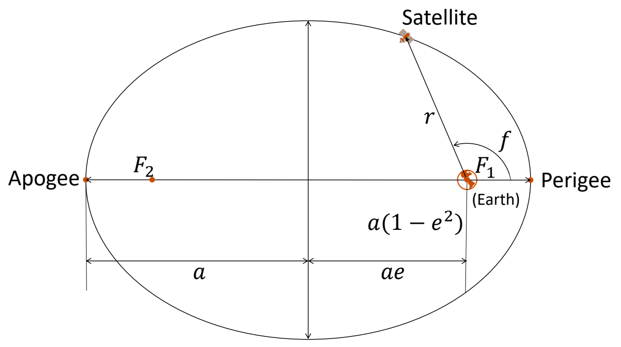


Fig. 7: Orbital elements in an elliptical orbit of a satellite. The semi-major axis $a$ is the largest diameter of the ellipse. $F_1$ and $F_2$ are the focal points. In the focal point $F_1$ lies the earth as the central body. The position of the satellite is determined by the angle of the true anomaly $f$ and the distance between satellite and focal point $F_1$. Apogee and perigee describe the farthest and closest point to the central body. The ratio between the semi-major axis and the distance between the focal points is called eccentricity $e$ (adapted from [41]).

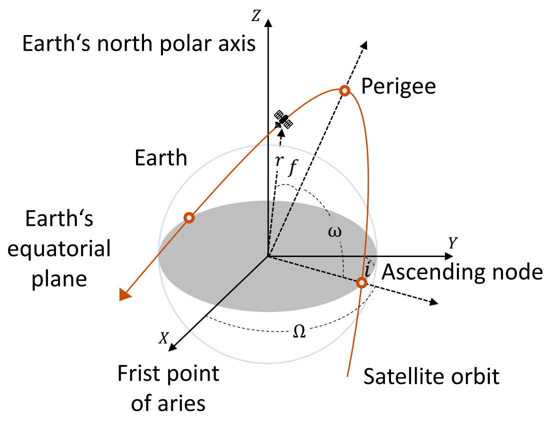


Fig. 8: Representation of the orbital elements in the three-dimensional ($X$-, $Y$-, $Z$-axis) geocentric equatorial system. The angle $f$ corresponds to the true anomaly. The orientation of the orbital plane to the Earth is defined by three angles $\Omega$, $\omega$, and $i$. $\Omega$ is the angle between first point of aries and ascending node. $\omega$ is the angle between ascending node and the perigee. $i$ is the vertical tilt of the ellipse (adapted from [42]).

### C. PCA and TCA Calculation

The conjunction detection based on the grid (and possibly the orbital filters) produces pairs of satellites together with a time stamp. We calculate the PCA and TCA for each such pair to detect if an actual conjunction occurs. For this, we rely on the *Brent* optimization algorithm [44] that combines a golden-section search's reliability with an interpolation method's performance. We utilize the reference implementation provided by the Boost[5] library. The different pairs are independent, so we can check them all in parallel.

[5]https://www.boost.org

For the grid-based variant, we know the sample time at which both satellites were close. We use this time step as the center ($c$) of an interval $I$, having radius $t$, i.e., $I = [c-t, c+t]$. Here, $t$ is the time it takes the slower of both satellites to cross two cells, which we can calculate simply by using the velocity vector at that time step. Accordingly, we use Brent's search on the interval $I$ to determine the minimum distance of the two satellites' positions. However, if this search finds a minimum at the end of the interval $I$, this might not actually be the local minimum. In fact, the local minimum might be just beyond the border of the interval, so we check a little further, and if this is indeed the case, we discard the pair (the minimum will be found when considering the neighboring interval).

For the hybrid variant, we distinguish between coplanar and non-coplanar pairs. The orbital filters determine the interval to search in for non-coplanar pairs. For the coplanar ones, the procedure is the same as for the grid-based variant.

## V. Evaluation

To demonstrate the superiority of our approach over traditional orbital filter chains, we performed several benchmarks with a realistic synthetically-generated satellite population and a screening threshold of 2 km, which is a typical for a rough screening process. We do this to examine the scalability of our approach, both in terms of time and memory consumption.

We performed most of the benchmarks on a Windows 10 system. For better comparability, we test both a CPU implementation accelerated with OpenMP[6] and a GPU implementation with CUDA[7]. We use a numba[8] JIT accelerated single-threaded Python implementation that uses traditional filter chains as a baseline consideration, referring to this variant as *legacy* [45]. We also test our implementation on a RedHat 8.6 system with two high-end CPUs. This way, we hope to provide a better comparison on a hardware level. Table I shows the benchmark system configuration.

### A. Data Generation

We used synthetically-generated orbit data to perform the measurements for a large set of satellites. This data was partially derived from the database of real operational satellites in early 2021 [46]. We employed a bivariate kernel density estimate to model the distribution and relationship between the semi-major axis and the eccentricity (see Figure 9). The other orbital parameters are distributed uniformly at random (see Table II). To mimic typical use-cases for orbit simulations with conjunction detection (see Chapter I), we have generated several satellite populations with sizes between 2000 and 1,024,000.

### B. Parameterization

Fixed-size hash sets/maps require a prior size estimation—so, we need to consider the memory consumption. In particular, within the spatial data structures, we have to calculate how

[6]https://www.openmp.org/
[7]https://developer.nvidia.com/cuda-toolkit
[8]https://numba.pydata.org/

| System Property | Values |
|---|---|
| Operating System | Windows 10 |
| GPU name | NVIDIA RTX 3090 |
| GPU CUDA cores | 10496 |
| GPU base clock | 1.4 GHz |
| GPU memory | 24 GB GDDR6X |
| CPU name | AMD Ryzen 9 5950X |
| CPU cores | 16 |
| CPU threads | 32 |
| CPU base clock | 3.4 GHz |
| System memory | 64 GB DDR4 |
| Operating System | RedHat 8.6 |
| CPU name | 2x Intel Xeon Platinum 9242 |
| CPU cores | 2x 48 |
| CPU threads | 2x 48 |
| CPU base clock | 2.3 GHz |
| System memory | 384 GB DDR4 |

TABLE I: We use two different systems for benchmarking to allow better comparability between a high-end CPU and a high-end GPU.

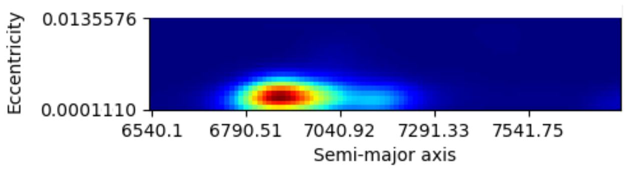


Fig. 9: Bivariate distribution function generated from the real satellite data between the semi-major axis and the eccentricity. A high satellite concentration is shown in red at a semi-major axis of about 7000 km and an eccentricity of 0.0025. The concentration becomes weaker with increasing deviation of the concentration point outward, which is shown in yellow, green and finally blue.

| Kepler Element | Value Range |
|---|---|
| Semi-major axis | From distribution |
| Eccentricity | From distribution |
| Inclination | $0 - \pi$ |
| Right-ascension of ascending node | $0 - 2\pi$ |
| Argument of perigee | $0 - 2\pi$ |
| (Mean anomaly) | $0 - 2\pi$ |
| True anomaly | From mean anomaly |

TABLE II: Value ranges of the Kepler elements used to generate the artificial satellite population.

many samples and grids we can process in parallel before running out of memory.

We can calculate the maximum number of samples $p$ if we subtract the fixed allocations consisting of the satellite information $a_s$, the Kepler solver data $a_k$, and the conjunction hash map $a_{ch}$ from the free memory $m$ and divide the result by the resources needed per grid. For each grid instance, we need a hash map $a_{gh}$ and the elements to represent the singly linked list $a_l$:

$$p = \frac{m - a_s - a_k - a_{ch}}{a_{gh} + a_l}$$

In addition, we can calculate the total number of samples $o$ that we need to process for each satellite. It is equal to the quotient of the *simulated time span* $t$ in seconds and the *seconds per sample* $s_{ps}$, which corresponds to the step size that is made in the time domain.

$$o = \frac{t}{s_{ps}}$$

By dividing the total number of samples that we need to process by the number of calculation steps that fit into memory at once, we can calculate the number of computational rounds $r_c$ that are required to process all samples:

$$r_c = \frac{o}{p}$$

while the satellite information $a_s$, the Kepler solver data $a_k$, and the number of elements of the singly linked list $a_l$ obviously correspond to the number of satellites $n$ multiplied with the corresponding data structure size.

The size of the grid hash set can also be defined easily. Each satellite must be inserted exactly once, so the hash set must have at least as many slots. However, the hash function can produce collisions (that have to be resolved by linear probing). We use twice the number of satellites as slots to mitigate the number of hash collisions and break up long clusters.

Compared to the grid hash set, the growth of the conjunction hash map behaves differently. The number of elements stored in it is not known in advance. Instead, the size depends on several factors like the number of satellites, the grid cell size or the seconds per sample, and the simulation period.

To determine the optimal size, we created an empirical model for the grid-based and the hybrid variant using Extra-P[9] [47] and embedded it in the application. The estimated models for the grid-based variant (see Equation 3) and the hybrid variant (see Equation 4) show that the estimated number of conjunctions $c'$ is mainly composed of the product of the number of satellites $n$, the simulated time span $t$ and the screening threshold $d$:

$$c' = 2.32 \cdot 10^{-9} \cdot n^2 \cdot s^{\frac{4}{3}} \cdot t \cdot d^{\frac{7}{4}} \tag{3}$$

$$c' = 2.14 \cdot 10^{-9} \cdot n^2 \cdot s^{\frac{5}{3}} \cdot t \cdot d \tag{4}$$

To ensure that the size is not too small, we ensure that at least 10,000 elements fit into the conjunction hash map. Like the grid hash map, the conjunction hash map needs additional space to allow fast insertion, so we double the number of slots. In addition, the number of conjunctions varies strongly as it depends on the properties of the satellite population. Thus, we treat the Extra-P–model more as a base size assumption, and accordingly, we double the hash map size again.

$$c = \max(c'; 10,000) \cdot 2 \cdot 2$$

The actual memory consumption of the hash map $g_{ch}$ can now be calculated from $c$ and the size of the data structure:

$$g_{ch} = c \cdot 16\,\mathrm{B}$$

[9]https://github.com/extra-p/extrap

We can decrease the number of elements in the conjunction hash map, and thus the required memory, by either adjusting the simulation period or adjusting seconds per sample parameter.

We automatically reduce the second per sample for the hybrid variant until a parallelization factor $p$ is obtained, which corresponds approximately to 512. This equals the number of CUDA threads that can be used in a single block of the grid conjunction detection kernel function.

## *C. Benchmarks*

Figure 10 shows the runtime of the two new conjunction detection methods on CPU and GPU for different-sized satellite populations compared to the legacy variant. The GPU variants' timings include all necessary calculation steps, including memory allocation and CPU/GPU data transfer operations. These take up about 3 % of the total time on average.

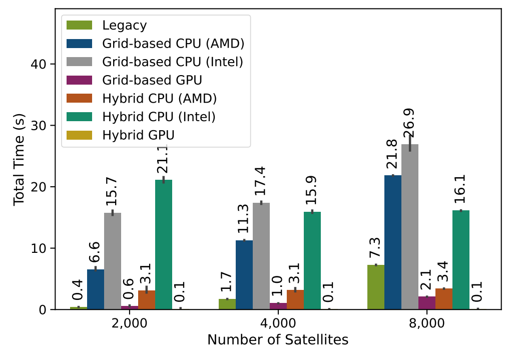
The results show that the hybrid GPU variant is the fastest among all tested variants. At a small number of 2000 satellites, it is about eight times faster than the legacy variant (see Figure 10a). As the number of satellites increases, the hybrid method can extend the performance advantage compared to the legacy variant even further. The test with 64,000 satellites is 675 times faster (see Figure 10b). Thus, the collision detection for 64,000 elements takes, on average, 1.15 s.

Compared to the grid-based GPU variant, the legacy method slightly outperforms it at 2000 satellites (see Figure 10a). However, at 4000 satellites, the grid-based GPU method is already approximately 30 % faster. Like with the hybrid GPU variant, the advantage increases significantly with the number of satellites. At 64,000 elements, the grid-based variant is, on average, about 43 times faster than the legacy variant but 15 times slower than the hybrid GPU implementation. From 8000 satellites on, the performance of the CPU variant of the hybrid method surpasses the legacy method. It halves the time that the legacy variant needs. The grid-based CPU variant can only outperform the legacy variant late at 32,000 satellites.

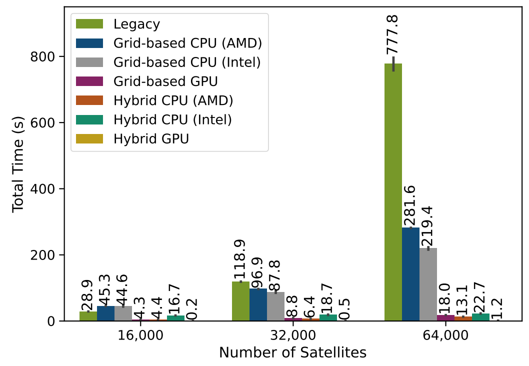
It is also worth mentioning that the CPU variant of the hybrid method is faster than the grid-based GPU variant for 16,000–64,000 satellites. Therefore, it is well suited to process a medium number of satellites if no graphics card is available.

For a higher number of satellites, the measurements show that both GPU variants are superior to the CPU implementations. As with the small population, the hybrid GPU implementation performs best. On average, 1,024,000 objects are processed in 3 min. The grid-based GPU variant beats the hybrid CPU variant at 128,000 satellites and can further expand its lead with an increasing number of satellites.

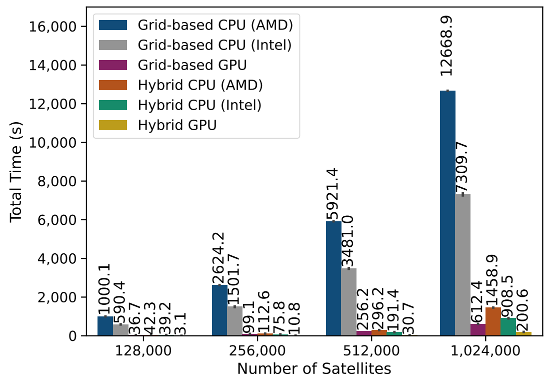
However, the difference in speed between the hybrid GPU and grid-based GPU implementation becomes increasingly smaller as the hybrid variant requires significantly more memory. For 512,000 and 1,024,000, the algorithm automatically adjusts the seconds per sample parameter. Thus, for 512,000 satellites, the parameter is set from nine to four, and for 1,024,000, it is set from nine to one to fit the conjunction hash map into the graphics memory. While at 256,000 elements



(a) For a small number of objects, the legacy variant (green, left) already shows its super-linear scaling. Our proposed methods, both on the GPU and CPU, show linear speed up or less due to the comparably large overhead.



(b) For populations between 16,000 and 64,000, the legacy variant continues its super-linear scaling, while our proposed variants start to show their scalings as well. The grid-based CPU variant (blue/grey, second/third to left) is notably slower than the hybrid CPU variant and both their GPU counterparts.



(c) For a number of satellites which would exceed the capabilities of the legacy variant, both GPU variants outperform the CPU variants. Most notable is the performance degradation of the hybrid GPU variant (yellow, right) from 512,000 to 1,024,000 satellites. At this point, the available memory limits the number of steps we can process in parallel.

Fig. 10: Runtime of the grid-based CPU, hybrid CPU, grid-based GPU, hybrid CPU, and legacy variant for different sized satellite populations.

the hybrid variant was still nine times faster than the grid-based implementation, the advantage decreased. At 512,000 elements, the factor is only eight. At 1,024,000 satellites, the hybrid implementation is only three times faster.

The hybrid implementation performs much better than the grid-based one between the two CPU variants. The higher number of calculations executed in the grid-based variant slows down the process significantly. Additionally, the hybrid CPU variant benefits from the large system memory. As a result, the hybrid CPU variant can process 1,024,000 Satellites 8 times faster than the grid-based CPU variant. We also see that the high-end Intel-CPU system outperforms the AMD-CPU from 16,000 satellites onward. This does not surprise: For a relative small number of objects, the faster clock speed of the AMD CPU has a higher impact than the additional cores of the Intel CPU. However, with a growing number of satellites, the additional cores outweigh their slower clock speed.

*1) Relative Time Consumption:* Regarding the relative time consumption of the different variants, all variants spend most of their time with the actual conjunction detection (CD; see Section IV-A3). The second most time is spent in the insertion into the grid (INS; see Section IV-A2). The hybrid GPU variant spends 68 % in CD, 21 % in INS, and 9 % determining if orbits are coplanar. For the hybrid CPU variant, the numbers are 87 % in CP, 9 % in INS, and 3 % determining if orbits are coplanar. The grid-based variants do not perform the coplanarity check, so the GPU variant spends 72 % of the time in CD and 26 % in INS. The CPU variant spends 92 % in CD and 7 % in INS.

The time for insertion into the hash map in the GPU variants varies strongly with the number of collisions and thus the hash map fill level. The median is about 20 % for INS and about 6 % for CD.

*2) CPU Thread Impact:* An evaluation of the grid-based CPU variant and the hybrid CPU variant using the AMD CPU shows that the grid-based variant benefits more from an increasing number of threads than the hybrid variant. The grid-based variant achieves a maximum speedup of 19, while the hybrid variant achieves a maximum speedup of 14 with 32 threads. Accordingly, the maximum efficiency with 32 threads is 59 % for the grid-based variant and 44 % for the hybrid variant.

*3) CPU–GPU Comparability:* We turn to comparing the thermal design power (TDP) of the hardware components for evaluating which is the most efficient. The AMD CPU has a TDP of 105 W, the Intel CPUs have a TDP of 350 W each, and the NVIDIA GPU has a TDP of 350 W. This shows that the GPU is more efficient than both test systems with CPUs—the Intel CPUs have a higher energy consumption while still taking more time and the AMD CPU takes consistently more than 7x as long as the GPU (comparing the same variants respectively).

## *D. Accuracy*

After assessing the performance of our approach, we now turn to validating the proposed methods in terms of accuracy, i.e., comparing the number of found conjunctions between the legacy variant and our implementations. For this, it is crucial to note that we have to differentiate between the conjunctions detected (whenever two satellites come too close to one another) and possibly colliding pairs because our variants sometimes produce multiple conjunctions for the same pair due to the smaller time steps. Our approaches all perform excellently, with the CPU and GPU implementations producing the same number (for the grid-based and the hybrid variant, respectively). For example, looking at the population of 64,000 satellites, the legacy variant identifies 17,184 conjunctions, the grid-based variant produces 17,264, and the hybrid variant 17,242. The hybrid variant finds all the colliding pairs of the legacy variant (and 30 more), while the grid-based variant misses 5 pairs and produces 35 more. The 5 missing conjunctions in the grid-based variant are exclusively edge cases where the Brent optimization algorithm stops searching for the minimum too early. In these cases, the conjunctions are at most 50 m above the 2 km threshold and therefore do not inherent any increased risk of collision. Adjusting the search algorithm would make it possible to find these conjunctions. However, as described, such minor deviations are acceptable in a fundamentally inaccurate system and, therefore, not of further relevance. Looking at the population of 1,024,000 satellites, the grid-based variant produces 4,418,979 conjunctions, while the hybrid variant produces 4,413,087 ones; the results for the other population sizes are all comparable.

## *E. Parallelism Trade-Off*

In our implementation, data parallelism is preferred over functional parallelism because today's GPU architectures are particularly well-designed for this type of parallelism. That is, in our case, the data is a tuple consisting of satellite information and future times.

On the GPU, we use one thread per tuple to propagate the satellite position and insert it into the grid as there are no dependencies between different ones. Furthermore, we calculate as many points in time in parallel as fit into the memory (see Section V-B). Thus, we maximize the data parallelism until no further positions can be computed temporarily due to memory limitation. We use the same procedure on the CPU, except that a thread is responsible for propagating and grid-inserting multiple tuples.

## VI. Conclusion

In this paper, we have shown that it is possible to significantly speed up the process of orbital conjunction detection using spatial data structures. We developed two conjunction detection variants based on spatial data structures to circumvent the quadratic number of (satellite) orbit comparisons. The grid has proven to be a suitable data structure since the objects do not show significant differences in size concerning the cell size. To map a geometric space up to the geostationary orbit, we used hash sets and hash maps as a compromise between memory and performance.

The benchmarks for both presented variants show that the conjunction detection using the grid-based and the hybrid variant is superior to the conventional conjunction detection in terms of performance and memory consumption. While it is not possible with the legacy variant to calculate more than 64,000 satellites on our system with 64 GB RAM, one million objects are not yet the limit for the grid-based and the hybrid variant on the graphics card.

Our presented variants make it possible to examine all objects in space that can be tracked in the near future for possible conjunctions and thus increase space safety. As for the possible future extensions, we have noted that memory usage is the current limiting factor—using multiple GPUs would solve this problem to some degree. Furthermore, improving the portability of the code—for example, using AMD HIP or OpenMP target offloading—would make our approach applicable in more use cases. Lastly, exchanging parts of the algorithm, like GPU-specific faster/more fine-tuned hash methods or other propagators instead of the Kepler Contour solver, might increase the performance and precision.

## Acknowledgments

This work was funded by the Hessian LOEWE initiative within the Software-Factory 4.0 project, by the European Space Agency within the project Rules4CREAM, ESA Contract No. 4000131551/20/NL/BJ/va, as well as by the Federal Ministry of Education and Research (BMBF), funding no. NHR2021HE, and the state of Hesse (HMWK), funding no. Kapitel 1502, Förderprodukt 19 NHR4CES as part of the NHR Program. We acknowledge the support of the European Commission and the German Federal Ministry of Education and Research (BMBF) under the EuroHPC Programme DEEP-SEA (GA 955606, BMBF funding no. 16HPC015), which receives support from the European Union's Horizon 2020 programme and DE, FR, ES, GR, BE, SE, GB, CH.

The authors gratefully acknowledge having conducted a part of this study on the Lichtenberg high-performance computer of TU Darmstadt. Furthermore the authors would like to thank the team of the ESA Space Debris Office for the useful discussions and their support.

## References

[1] J. Foust. (2021) Spacex launches starlink satellites and expands international service. [Online]. Available: https://spacenews.com/spacex-launches-starlink-satellites-and-expands-international-service/

[2] ESA, "Esa's annual space environment report," ESA/ESOC, Darmstadt, Tech. Rep., 2021.

[3] J. McDowell, "Space activities in 2021," 2022. [Online]. Available: https://planet4589.org/space/papers/space21.pdf

[4] M. A. Stevenson, M. Nicolls, I. Park, and C. Rosner, "Measurement precision and orbit tracking performance of the kiwi space radar," in *Proc. of 2020 Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, 2022.

[5] ESA, "Space environment statistics," 2022. [Online]. Available: https://sdup.esoc.esa.int/discosweb/statistics/

[6] D. J. Kessler and B. G. Cour-Palais, "Collision frequency of artificial satellites: The creation of a debris belt," *Journal of Geophysical Research: Solid Earth*, vol. 83, no. A6, pp. 2637–2646, 1978.

[7] Mike Wall. (2021) Space collision: Chinese satellite got whacked by hunk of russian rocket in march. [Online]. Available: https://www.space.com/space-junk-collision-chinese-satellite-yunhai-1-02

[8] J. Barnes and P. Hut, "A hierarchical o (n log n) force-calculation algorithm," *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[9] L. Greengard and J. Strain, "The fast gauss transform," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[10] E. Kerr and N. Sánchez Ortiz, "State of the Art and Future Needs in Conjunction Analysis Methods, Processes and Software," in *Proc. of 8th European Conference on Space Debris*, 2021.

[11] J. Radtke, S. Mueller, V. Schaus, and E. Stoll, "LUCA2 - An enhanced long-term utility for collision analysis," in *Proc. of 7th European Conference on Space Debris*, 2017.

[12] J. Woodburn, V. T. Coppola, and F. Stoner, "A description of filters for minimizing the time required for orbital conjunction computations," *Advances in the Astronautical Sciences*, vol. 135, 2010.

[13] F. R. Hoots, L. L. Crawford, and R. L. Roehrich, "An analytic method to determine future close approaches between satellites," *Celestial Mechanics and Dynamical Astronomy*, vol. 33, no. 2, 1984.

[14] J. Radtke, S. Flegel, J. Gelhaus, M. Möckel, V. Braun, C. Kebschull, C. Wiedemann, H. Krag, K. Merz, and P. Vörsmann, "Revision of Statistical Collision Analysis for Objects Inside of Satellite Constellations," in *Proc. of 64th International Astronautical Congress (IAC 2013)*, 2013.

[15] J. Woodburn and D. Dichmann, "Determination of Close Approaches for Constellations of Satellites," in *Mission Design & Implementation of Satellite Constellations*, ser. Space Technology Proc., J. C. Ha, Ed. Dordrecht: Springer Netherlands, 1998, vol. 1, pp. 337–345.

[16] L. M. Healy, "Close conjunction detection on parallel computer," *Journal of Guidance, Control, and Dynamics*, vol. 18, no. 4, pp. 824–829, 1995.

[17] J. R. A. Rodríguez, F. M. M. Fadrique, and H. Klinkrad, "Collision Risk Assessment with a 'Smart Sieve' Method," in *Proc. of the Joint ESA-NASA Space-Flight Safety Conference*, ser. ESA SP, B. Battrick, Ed. Noordwijk: ESA Publications Division, 2002.

[18] B. Bastida Virgili, "DELTA (Debris Environment Long-Term Analysis)," in *Proc. of 6th International Conference on Astrodynamics Tools and Techniques (ICATT)*, 2016.

[19] H. G. Lewis, "DAMAGE: A dedicated geo debris model framework," in *Proc. of 3rd European Conference on Space Debris 2001*, 2001.

[20] H. Klinkrad, "Collision risk analysis for low Earth orbits," *Advances in Space Research*, vol. 13, no. 8, pp. 177–186, 1993.

[21] J. C. Liou, D. J. Kessler, M. Matney, and G. Stansbery, "A New Approach to Evaluate Collision Probabilities Among Asteroids, Comets,and Kuiper Belt Objects," in *Lunar and Planetary Science Conference*, ser. Lunar and Planetary Science Conference, S. Mackwell and E. Stansbery, Eds., Mar. 2003, p. 1828.

[22] H. G. Lewis, S. Diserens, T. Maclay, and J. P. Sheehan, "Limitations of the cube method for assessing large constellations," in *Proc. of 1st International Orbital Debris Conference (IOC 2019)*, 2019.

[23] E. R. George, "A High Performance Conjunction Analysis Technique for Cluster and Multi-Core Computers," in *Proc. Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS 2011)*, 2011.

[24] V. T. Coppola, S. Dupont, K. Ring, and F. Stoner, "Assessing satellite conjunctions for the entire space catalog using cots multi-core processor hardware," *Advances in the Astronautical Sciences*, vol. 135, 2010.

[25] B. Abernathy, D. Surka, S. Harvey, and M. O'Connor, "The CAOS-D Architecture for Conjunction Analysis," in *Proc. of Infotech@Aerospace 2011*, 2011, p. 1434.

[26] F. Schrammel, F. Renk, A. Mazaheri, and F. Wolf, *Efficient Ephemeris Models for Spacecraft Trajectory Simulations on GPUs*. Springer International Publishing, 08 2020, pp. 561–577.

[27] M. Möckel, *High performance propagation of large object populations in earth orbits*. Berlin: Logos Verlag Berlin GmbH, 2015.

[28] M. Fehr, V. Navarro, L. Martin, and E. Fletcher, "The Process of Parallelizing the Conjunction Prediction Algorithm of ESA's SSA Conjunction Prediction Service using GPGPU," in *Proc. of 6th European Conference on Space Debris 2013*, 2013.

[29] I. A. Budianto-Ho, C. Alberty, R. Scarberry, S. Johnson, and R. Sivilli, "Scalable Conjunction Processing using Spatiotemporally Indexed Ephemeris Data," in *Proc. of 15th Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS 2014)*, 2014.

[30] C. Ericson, *Real-time collision detection*. Crc Press, 2004.

[31] Y. Gu, Y. He, K. Fatahalian, and G. Blelloch, "Efficient bvh construction via approximate agglomerative clustering," in *Proceedings of the 5th High-Performance Graphics Conference*, ser. HPG '13. ACM, 2013, pp. 81—-88. [Online]. Available: https://doi.org/10.1145/2492045.2492054

[32] M. Rincon-Nigro, N. Navkar, and N. Tsekos, "Gpu-accelerated interactive visualization and planning of neurosurgical interventions," *Computer Graphics and Applications, IEEE*, vol. 34, pp. 22–31, 05 2014.

[33] S. Raschdorf and M. K. Clausthal, "Loose octree : a data structure for the simulation of polydisperse particle packings," 2009.

[34] R. Jaljolie, P. Van Oosterom, and S. Dalyot, "Spatial data structure and functionalities for 3d land management system implementation: Israel case study," *ISPRS International Journal of Geo-Information*, vol. 7, no. 1, 2018. [Online]. Available: https://www.mdpi.com/2220-9964/7/1/10

[35] "Chapter 3 - spatial data structures," in *Geographic Information Systems for Geoscientists*, ser. Computer Methods in the Geosciences, G. F. Bonham-Carter, Ed. Pergamon, 1994, vol. 13, pp. 51–82. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080424200500080

[36] D. Jünger, C. Hundt, and B. Schmidt, "Warpdrive: Massively parallel hashing on multi-gpu nodes," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 441–450.

[37] S. Ashkiani, M. Farach-Colton, and J. D. Owens, "A dynamic hash table for the gpu," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 419–429.

[38] D. Tsukamoto and T. Nakashima, "Implementation and evaluation of distributed hash table using mpi," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 684–688.

[39] T. Maier, P. Sanders, and R. Dementiev, "Concurrent hash tables: Fast and general(?)!" *ACM Trans. Parallel Comput.*, vol. 5, no. 4, feb 2019. [Online]. Available: https://doi.org/10.1145/3309206

[40] D. Xue and J. Li, "Collision criterion for two satellites on Keplerian orbits," *Celestial Mechanics and Dynamical Astronomy*, vol. 108, no. 3, pp. 233–244, 2010.

[41] D. A. Vallado and W. D. McClain, *Fundamentals of astrodynamics and applications*, 4th ed., ser. Space technology library. Hawthorne: Microcosm Press, 2013.

[42] H. D. Curtis, *Orbital mechanics for engineering students*, 1st ed., ser. Elsevier Aerospace engineering series. Amsterdam: Elsevier/Butterworth-Heinemann, 2008.

[43] O. H. E. Philcox, J. Goodman, and Z. Slepian, "Kepler's goat herd: An exact solution to kepler's equation for elliptical orbits," *Monthly Notices of the Royal Astronomical Society*, 2021.

[44] R. P. Brent, "An algorithm with guaranteed convergence for finding a zero of a function," *The Computer Journal*, vol. 14, no. 4, pp. 422–425, 01 1971. [Online]. Available: https://doi.org/10.1093/comjnl/14.4.422

[45] S. Burgis, L. Rohrmüller, M. Michel, and R. Bertrand, "Simulation of satellites and constellations for the assessment of collision avoidance operations," *CEAS Space Journal*, 2022. [Online]. Available: https://doi.org/10.1007/s12567-022-00471-y

[46] Celestrak.com, "Tle list of active satellites as of 2021 apr 08 22:06:03 utc," 2021. [Online]. Available: https://celestrak.com/NORAD/elements/active.txt

[47] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA*. ACM, November 2013, pp. 1–12.