Computing and Visualizing Reachable Volumes for Maneuvering Satellites

Ming Jiang, Willem H. de Vries, Alexander J. Pertica, Scot S. Olivier
Lawrence Livermore National Laboratory

ABSTRACT

Detecting and predicting maneuvering satellites is an important problem for Space Situational Awareness. The spatial envelope of all possible locations within reach of such a maneuvering satellite is known as the Reachable Volume (RV). As soon as custody of a satellite is lost, calculating the RV and its subsequent time evolution is a critical component in the rapid recovery of the satellite. In this paper, we present a Monte Carlo approach to computing the RV for a given object. Essentially, our approach samples all possible trajectories by randomizing thrust-vectors, thrust magnitudes and time of burn. At any given instance, the distribution of the "point-cloud" of the virtual particles defines the RV. For short orbital time-scales, the temporal evolution of the point-cloud can result in complex, multi-reentrant manifolds. Visualization plays an important role in gaining insight and understanding into this complex and evolving manifold. In the second part of this paper, we focus on how to effectively visualize the large number of virtual trajectories and the computed RV. We present a real-time out-of-core rendering technique for visualizing the large number of virtual trajectories. We also examine different techniques for visualizing the computed volume of probability density distribution, including volume slicing, convex hull and isosurfacing. We compare and contrast these techniques in terms of computational cost and visualization effectiveness, and describe the main implementation issues encountered during our development process. Finally, we will present some of the results from our end-to-end system for computing and visualizing RVs using examples of maneuvering satellites.

1. INTRODUCTION

Detecting and predicting maneuvering satellites is an important problem for Space Situational Awareness (SSA). In the absence of a maneuver, it is a straightforward exercise to predict where a satellite will go once its orbit has been determined. If a satellite does maneuver, though, it quickly moves away from its initial trajectory, especially in the Low Earth Orbit (LEO) regime. The spatial envelope of all possible locations within reach of such a maneuvering satellite is known as the Reachable Volume (RV). The determination of an RV serves a few distinct purposes. Since an RV spans all possible locations a particular object can maneuver to given a certain burn-budget and elapsed time, it is an excellent representation of the probability density distribution of the target, i.e., the RV tells us how likely we are to find the object at any given location. This information is very useful in chain-of-custody applications. As soon as custody of a satellite is lost, calculating the RV and its subsequent time evolution is a critical component in the rapid recovery of the satellite, because searching and asset-tasking based on the RV can result in more efficient recovery methods.

An additional application for the RV is assessing whether certain targets are within reach of the maneuvering object. This is an extension of the conjunction analysis module we reported on last year [16], in the sense that it will do a probabilistic conjunction against the RV instead of against another orbiting object. Finally, since the RVs tend to grow rapidly with time, depending on the allowed maximum delta-v, it is useful to have additional constraints to limit the growth. Especially in the LEO regime, it does not take very long for the RV to wrap around the Earth (a few hours typically). Once this has happened, the usefulness of the RV diminishes rapidly as it starts filling more and more of the available space. By inserting additional knowledge about the maneuver, e.g., specific knowledge on thrust limits (the satellite is using an ion-thruster), or information about the intent of the maneuver (it is on its way to Geostationary Earth Orbit (GEO)), this allows us to extend the usefulness of the RV to many days. This restricted RV, which we are calling Trajectory Volume (TV), is especially useful for LEO to GEO transfer maneuvers.

At Lawrence Livermore National Laboratory (LLNL), we have developed a comprehensive modeling, simulation and visualization environment for SSA [10]. One of the tools that we have developed performs debris-cloud threat assessments against other orbiting objects in order to determine how a threatened target would have to maneuver to reduce the risk of a collision. This tool generates virtual debris-clouds based on just a few key parameters (e.g., mass, overlap, and relative velocity), all calibrated against hydro-code simulations [13]. A virtual cloud of debris

particles is not so much different than a set of virtual trajectories of a maneuvering object. Both cases concern new trajectories that are different from the unperturbed orbit. If there are no constraints on the maneuver other than a certain upper limit to the delta-V (e.g., burn in any direction, with any speed up to the maximum) the virtual trajectory "cloud" is indistinguishable from a virtual debris cloud with a similarly small velocity range. Instead of calculating collision risks, we are now using the trajectory cloud for chain-of-custody purposes. After a particular object has been lost due to a maneuver, all possible locations to where it could have moved, assuming a limit to the burn-budget (and time period over which the maneuver was executed), constitute a volume of space that is reachable from the maneuvering object. Another parallel between the two cases is that in the former it is debris posing a risk to a certain asset, while in the latter it is a maneuvering satellite that could potentially interfere with the asset.

In this paper, we present our Monte Carlo approach to computing RVs for a given object. Essentially, our approach samples all possible trajectories by randomizing thrust-vectors, thrust magnitudes and the time of burn. As an initial condition, it typically uses the state-vector of the last successful observation. Simple two-body propagation of the virtual trajectories captures the evolution of the RV over time. At any given instance, the distribution of the "point-cloud" of the virtual particles defines the RV. To computationally represent the volume, we utilize a structured grid in Earth-centered coordinate system. At each time step, the underlying probability density distribution is estimated by first accumulating the number of trajectory points that are contained within the grid cells and then normalizing the accumulation by the total number of trajectory points. Next, we present techniques for visualizing both the virtual trajectories and the computed RVs. We present a real-time out-of-core rendering technique for visualizing the large number of virtual trajectories. We also examine different techniques for visualizing the computed volume of probability density distribution, including volume slicing, convex hull and isosurfacing. We compare and contrast these techniques in terms of computational cost and visualization effectiveness, and describe the main implementation issues encountered during our development process. Finally, we will present some of the results from our end-to-end system for computing and visualizing RVs using examples of maneuvering satellites.

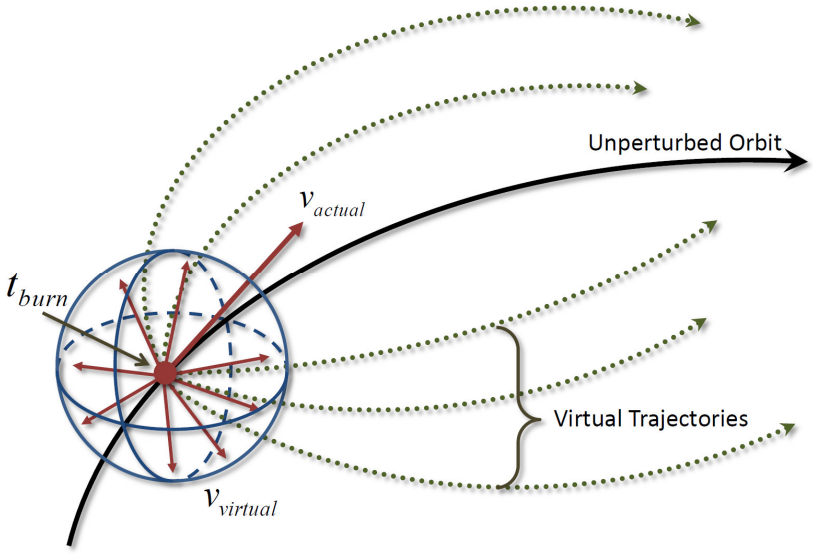

Fig. 1. This diagram illustrates our Monte Carlo approach to computing the virtual trajectories for the RV of a given object and its unperturbed orbit. At time t_{burn}, we compute $v_{virtual}$ by adding an offset vector to v_{actual}.

2. COMPUTING REACHABLE VOLUME

A maneuver can be defined as a deviation from an unperturbed orbit. External forces like drag acting on a drag-plate, or thrust from an engine, will change the trajectory and therefore can be considered a maneuver. The former, given its small magnitude and inherently long time-scales are more appropriate for station-keeping. We are, for the purposes of our RV/TV calculations, solely concerned with the large and immediate thrust cases powered by internal engines. From an orbital mechanics standpoint, the thrust acts as a modification of the velocity vector. An instantaneous impulse would be a single addition to the velocity component of the state vector in the appropriate

direction, whereas a more continuous thrusting maneuver would continue to modify the state vector for the duration of the burn. Since we are focused in this paper on more conventional propulsion mechanisms as opposed to, for instance, continuous ion-thrust propulsion, we make the assumption that our simulated burns are instantaneous.

All object propagation is done using a force-model propagator [14]. In its simplest incarnation, this propagator is just a strictly Newtonian two-body case with a Runge-Kutta method for solving the second order differential equation of motion. We have also implemented a version that includes, among other things, the higher gravitational moments, atmospheric drag and solar radiation pressure. The main difference between the two propagators, other than accuracy, is a factor of about 100 in execution speed. Both are well suited for our purpose though: a maneuver is nothing more than a modification of the velocity component at a certain time.

There are only four free parameters in our Monte Carlo framework: the magnitude of the burn, two directional angles (θ and φ), and the time of the burn t. Each one can be varied in a few ways. The burn magnitude can be kept fixed (in cases where it is known), it can be varied uniformly (e.g., between 0.0 and 2.0 km/s), or it can be varied about a mean value by a Gaussian σ. Both directional angles are varied such that the resulting vector is uniformly distributed on a sphere. This is done by varying θ uniformly between 0 and 2π, while at the same time setting $\varphi = \text{acos}(\alpha)$ with α varied uniformly between -1 and 1. Finally, the burn-time t can be kept fixed, varied uniformly over a particular period, or varied about a mean with a certain Gaussian σ.

Fig. 1 provides a diagram that illustrates our Monte Carlo approach to computing the virtual trajectories of the RV of a given object and its unperturbed orbit. At time t_{burn}, we compute $v_{virtual}$ by adding an offset vector to v_{actual}. Our example RV calculation is done with a fixed t_{burn}, set at the time of last good observation on the object[1], with both thrust angles varied uniformly (i.e., we do not know where it is going), and a burn magnitude of anywhere between 0.0 and 2.0 km/s (see Section 5 for more details). To computationally represent the volume, we utilize a structured grid in either Earth-centered inertial (ECI) or Earth-center, Earth-fixed (ECEF) coordinate systems. Our approach can operate on both rectilinear and curvilinear (spherical) grids. At each time step, the underlying probability density distribution is estimated by first accumulating the number of trajectory points that are contained within the grid cells and then normalizing the accumulation by the total number of trajectory points. The resulting volume is sampled with 1E5 to 1E7 virtual trajectories depending on desired grid resolution (see Section 4 for more details).

As the volume expands over time, certain checks are performed to ensure that virtual trajectories are properly removed in case of collision with the Earth or the Moon. Then, at each time of interest, the positions of all the remaining virtual particles are used to construct the volume. Relative densities are directly related to the relative likelihood that the maneuvering object is in that location. An additional filtering step is implemented that removes all trajectories that do not coincide with a given target trajectory/orbit. We call this subset of an RV a Trajectory Volume. It involves stepping through time and at each step generating a large Monte Carlo cloud of virtual trajectories. Only those trajectories that arrive at the target over the period of interest are kept. As such it can be considered a maneuver "with purpose". A typical example is a LEO to GEO maneuver where it is not known when and how the maneuver is executed other than that it has a known destination (e.g., a GEO longitude slot).

3. VISUALIZING REACHABLE VOLUME

Visualization plays an important role in gaining insight and understanding into the complex and evolving manifolds of the RV of an object. In the following section, we describe the different visualization techniques that we utilize for both the virtual trajectories and the computed RV. We describe a real-time out-of-core rendering technique for visualizing the large number of virtual trajectories. We also examine various techniques for visualizing the computed volume of probability density distribution, including volume slicing, convex hull and isosurfacing.

4.1 Out-of-Core Rendering

Rendering the set of virtual trajectories is a challenging problem due to the sheer number of trajectories involved. Naively trying to render all the trajectories at once may not even be feasible using a standard workstation due to memory constraints. Even if memory was not a bottleneck, the amount of time it takes to render all that data, even if

[1] In other words, we assume the object is initiating a maneuver immediately after we lose custody of it.

the rendering is just simple line drawing, would result in a non-interactive visualization system. One way to address this issue is to access the data in an out-of-core fashion [12, 15], so that at any time only part of the data is resident in memory for the rendering. A similar approach was presented in [6] for building a scalable visualization system for SSA. In an out-of-core rendering approach, the idea is to decouple the data access from the rendering and execute them on separate threads. The entire process becomes asynchronous. As the data access thread reads in data, it signals the rendering thread when a sufficient amount of data is ready for rendering. Likewise, the data access thread can be interrupted by the rendering thread when a change in view occurs and a new scene needs to be rendered. Care must be taken in order to maintain the previously rendered data within the same view. A simple and effective way to accomplish this is to maintain the OpenGL depth buffer (i.e., do not clear it) throughout all the renderings within the same view. In this fashion, the incremental rendering converges in an iterative fashion towards the final rendering with accuracy and correctness ensured. Hence, the entire rendering process remains interactive and scalable for large-scale data sets.

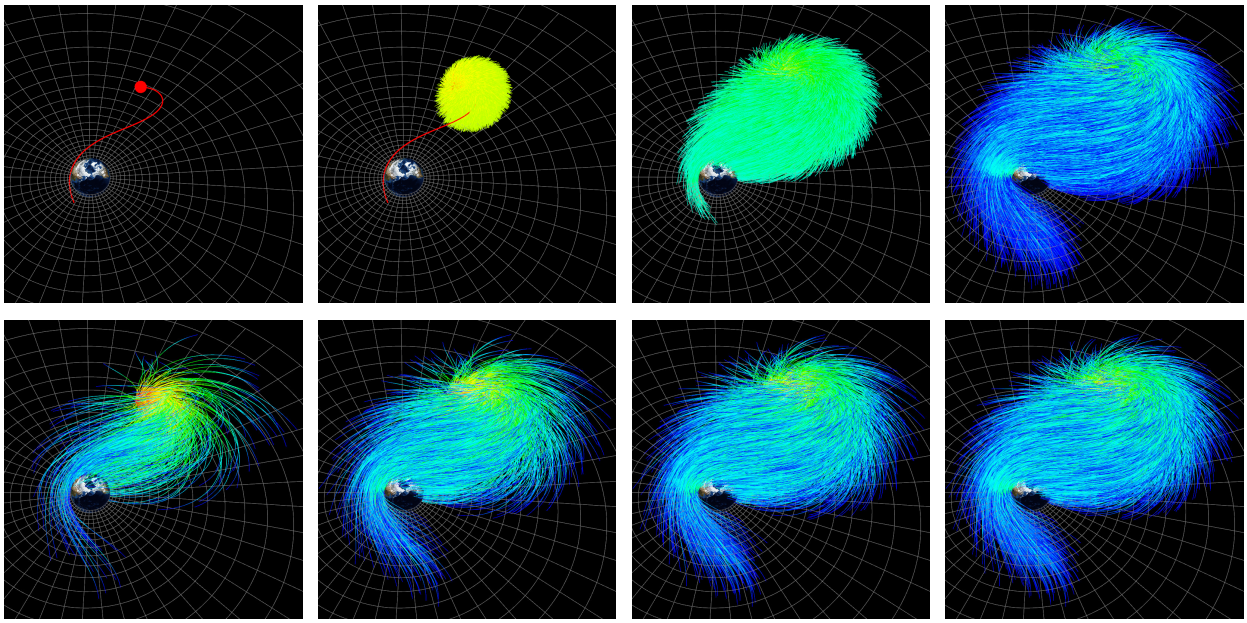

Fig. 2. Top row shows the results from the out-of-core rendering of time steps from the beginning of the propagation (red) to the end (blue) in the ECEF coordinate system. Bottom row shows out-of-core rendering of subsets of the virtual trajectories.

We present two approaches for iteratively rendering the virtual trajectories in an out-of-core fashion. The first approach utilizes the positional data from the orbit propagator at each time step and iteratively renders the time steps in order to visualize the temporal evolution of the virtual trajectories. Points within each time step are connected to the corresponding points from the previous time step to visualize the individual trajectories. The second approach constructs the trajectories from the positional data across all time steps and iteratively renders a subset of the trajectories until the desired number of trajectories has been rendered. Fig. 2 shows the results for both approaches. The top row shows the out-of-core rendering of time steps in the ECEF coordinate frame. The bottom row shows out-of-core rendering of subsets of the virtual trajectories. All line segments representing trajectories are color mapped according to the time step from the beginning of the propagation (red) to the end (blue).

Whereas the first approach is ideal for visualizing the temporal evolution of the cloud of trajectory points, the second approach is better suited for navigating the trajectory data because the overall structure of the trajectories is visible even with a few subsets of the rendered trajectories. In the current implementation, each approach has its own out-of-core representation. In order to utilize both approaches, one would have to double up the amount of stored data. This increase in data storage requirement may not be necessary if one employees an effective spatial-temporal data representation scheme [17]. In general, one disadvantage with out-of-core rendering is what is known as the *popping effect*, which occurs when one switches from a final rendering to an initial rendering after a change in view. One way to reduce this effect is to keep a set of data resident in main memory so that it can always be rendered as part of the initial rendering. Another disadvantage of this approach is that as the data size increases, so

does the number of iterations to converge to a final rendering. One way to overcome this issue is to introduce a notion of multi-resolution representation: coarse levels can be used to render an overview and finer levels can be used for specific detailed views.

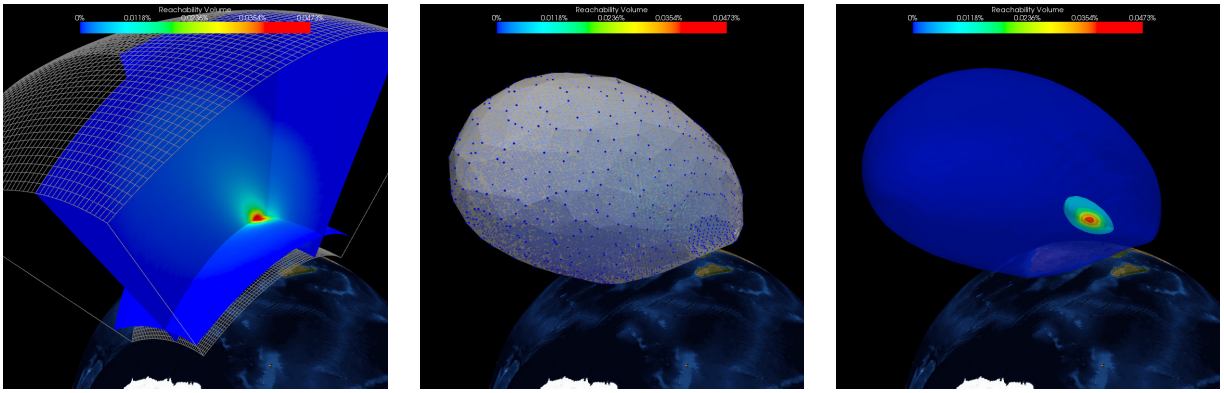

Fig. 3. Three visualization techniques for RV: (left) volume slicing within each dimension, (middle) convex hull of the grid points, and (right) isosurfacing of the extracted curvilinear grid at various contour values.

4.2 Volume Visualization

Once the structured grid representing the RV is computed, we utilize three visualization techniques: volume slicing, convex hull and isosurfacing, to explore the scalar volume data (probability density distribution). The first technique is volume slicing [5], which extracts a 2D plane from the 3D volume and maps colors to the scalar value on the plane. Color mapping is one of the most basic forms of scalar visualization. In order to get a better sense of the 3D volume, one can extract planes along each dimension as shown in the left image of Fig. 3. In this example, the slice planes were pre-selected to illustrate the high probability density core region. This is an efficient technique that works well for simple volumes. Unfortunately, this technique cannot adequately capture the shape of volumes with complex structures. This technique also requires more user interaction in the form of moving the slice planes back and forth in order to see different parts of the volume.

One way to better capture the shape of the underlying volume is to construct a convex hull to represent the outer envelope of the volume. The convex hull of a set of points is the smallest convex polygon for which each point in the set is either on the boundary of the polygon or in its interior [1]. Although there are several algorithms for computing the convex hull, we utilize the approach of computing the 3D Delaunay triangulation on the grid points. Note that the 3D Delaunay triangulation is computationally expensive, which is the main reason why we do not use the original trajectory points for this computation. In a sense, one can think of the structured grid as a way to prune spatially proximate points in order to reduce the run-time of the algorithm. For more details on convex hull and Delaunay triangulation, please see [1].

The middle image of Fig. 3 shows the convex hull extracted for the grid points of the RV. We rendered a semi-transparent convex hull as well as the grid points color mapped to their probability density, in order to better visualize the internal probability distributions enveloped by the convex hull. One of the issues with using the convex hull to visualize the RV is that one can only visualize the outer envelope of the RV, and not any of the probability density envelopes within. Given that the outer envelope contains very low probability density, the visualization may not be useful especially when large amounts of time have elapsed since the maneuver. Although rendering the convex hull surface semi-transparent can help better visualize the internal structure of the RV, the issue remains for how to effectively visualize the internal structures.

One way to better capture the internal structures of the RV is to extract isosurfaces at various isovalues, which correspond to contours of the probability density distribution. An isosurface is a surface that represents points of constant value (contours) within a volume; in other words, it is a level set of a continuous function in 3D space. Marching Cubes [8] is a standard technique that is based on the divide-and-conquer approach. The basic assumption is that a contour can pass through a grid cell in only a finite number of ways. A case table is constructed that enumerates all possible topological states of a grid cell, given combinations of scalar values at the cell points. The

number of topological states depends on the number of cell vertices and the number of inside/outside relationships a vertex can have with respect to the contour value. A vertex is considered inside/outside a contour if its scalar value is larger/smaller than the contour value. Acceleration techniques for isosurface extraction, such as using interval trees [4] and octrees [17], have been proposed, as well as isosurfacing in higher dimensions (e.g., time-varying) [2].

The right image of Fig. 3 shows the extracted isosurfaces, rendered semi-transparently, corresponding to a set of isovalues that was automatically computed based on the range of the probability density distribution. For visualizing the RV, isosurfacing provides an effective way to not only visualize the outer envelope of the RV, but also the internal structures as well, by specifying higher isovalues that correspond to higher probability densities for the extraction algorithm. More importantly, by keeping the isovalues constant across all time steps, one can visualize the temporal evolution of the RV by visualizing the evolution of these nested isosurfaces. Since extracting isosurfaces can be much less computationally expensive than extracting the convex hull of the grid points, then this approach is a better option for visualizing RVs. (See [3] for a clustering based approach to visualizing spatial probability density function data.)

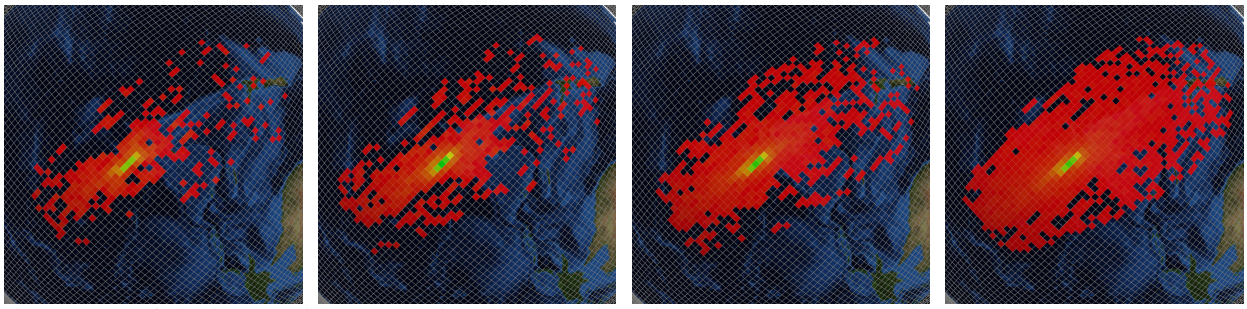

Fig. 4. One of the issues with constructing the RV using virtual trajectories is undersampling, which results in a sparsely populated volume. The images show a slice of the spherical grid and only the non-zero grid cells are rendered. The different numbers of virtual trajectories are: 40K, 80K, 160K and 320K. (Note the colormap has been inverted to provide better visual contrast with the background imagery.)

4. IMPLEMENTATION DETAILS

One of the issues with constructing the structured grid to represent the RV using is undersampling. The resolution of a structured grid is typically selected based on the desired level of detail one wishes to capture from the RV. At finer grid resolutions, the number of virtual trajectories that is needed to fully capture the RV must increase in order to avoid generating sparsely populated volumes. This problem is illustrated in Fig. 4, for which we populated the same structured grid, with a resolution of (360,180,500), using different numbers of virtual trajectories: 40K, 80K, 160K and 320K. To avoid the problem of occlusion, we rendered a slice of the structured grid by only displaying the populated grid cells. Note the colormap was inverted (from red to blue) to provide better visual contrast with the Blue Marble Earth imagery. Using Fig. 3 as a reference, one can clearly see the sparseness of the volume and the convergence towards the fully filled volume as the number of virtual trajectories increases. (The number of virtual trajectories used in Fig. 3 is 1E7.)

Clearly, generating more virtual trajectories requires not only longer running time, but also more data storage. This impacts both the data generation phase as well as the data analysis and visualization phase. If a coarser resolution grid is not desirable, then one way to address this issue is to apply a smoothing filter to the structured grid in order to fill-in the missing grid cells by spreading the accumulated probability density around to neighbors. Fig. 5 provides an example of smoothing filter which we implemented as a 3D Gaussian filter on the structured grid. For this RV, we used 1E5 virtual trajectories to populate the same structured grid as above. The left image of Fig. 5 shows the sparse volume slice and the right image shows smoothing after 5 iterations of a 3x3x3 Gaussian kernel. Note the colormap in the image was re-normalized to show the full dynamic range of the smoothed probability density. One of the side effects of Gaussian smoothing is that it can reduce the dynamic range of the data by reducing the extrema within the data. Note that for curvilinear grids, the more accurate method to applying Gaussian smoothing is to solve an equivalent heat equation on the hexahedral grid cells by using the finite element method. The implementation for this method is non-trivial and computationally more expensive.

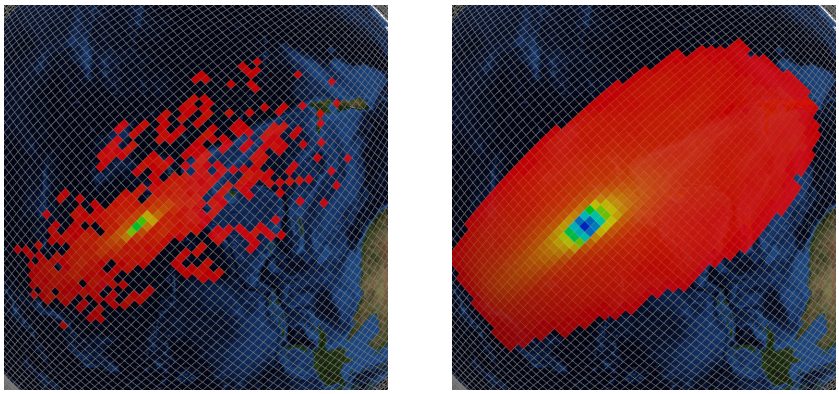

Fig. 5. One way to address the sparse RV issue is to apply a smoothing filter to the grid in order to fill in the empty grid cells. Left image show the slice of the RV constructed using 1E5 trajectories. Right image shows the same volume smoothed using a Gaussian kernel. (Note the colormap was re-normalized to show the full dynamic range.)

Another issue that we encountered during the implementation was using the spherical grid, which has a singularity at each pole that causes the hexahedral grid cells to shrink and degenerate into wedges. The large variations in the grid cell sizes can cause problems for the visualization algorithm, especially near the pole region. The left image in Fig. 6 shows this problem for the BURNSAT simulation in the outer isosurface (blue) near the North Pole region. Notice the deformation in the otherwise smooth isosurface that occurs only near the pole region. A more subtle problem is that the probability density accumulation does not take into account of the grid cell size when normalizing. For the spherical grid, the accumulation is carried out in the computational space (right ascension, declination, radius), where the grid cell sizes are constant, as opposed to the physical space (ECI or ECEF), where they are not.

Another issue we encountered with using the spherical grid is dealing with the boundary condition when filtering or visualizing the volume data. One must handle the periodic boundaries appropriately; otherwise, misalignments and discontinuities can occur in the visualization result. The right image in Fig. 6 shows the isosurfacing results for the TDRS_BURN simulation. Note that there are discontinuities and misalignments in all the isosurfaces near where right ascension jumps from 360 back to 0. One approach to dealing with these issues is to avoid the use of spherical grids and use a rectilinear grid in ECI or ECEF instead, which is what we used for the TEST_TV simulation.

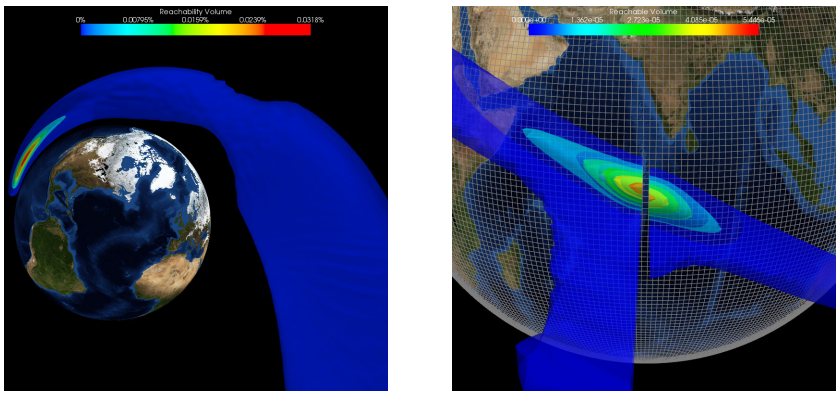

Fig. 6. Left image shows the pole singularity problem for the spherical grid in the BURNSAT simulation. The outer isosurface (blue) near the North Pole region is clearly deformed. Right image shows the boundary problem for the spherical grid in the TDRS_BURN simulation. There is a discontinuity and misalignments of the isosurfaces.

5. RESULTS

We conducted three simulations to test the performance and effectiveness of our RV computation and visualization algorithms. For the first simulation, BURNSAT, we kept the burn time fixed, the thrust direction uniform, and only

varied the burn magnitude between 0.0 and 2.0 km/s. The propagation time spans from 55458.065 to 55458.165 in Modified Julian day (MJD). We used a spherical grid to accumulate the probability density distribution, with a grid resolution of (360,180,500), where positions along the radial direction are given by: 10^(i*0.0017886+6.8181), for 0 $\leq i \leq$ 499. For this simulation, we generated 1E7 virtual trajectories for 100 time steps, with a step size of 0.001 day. The Two-Line Element (TLE) for this simulation is given in the top row of Fig. 7. The volume visualization images in Fig. 3 were generated using this simulation. The top row of Fig. 8 shows the orbit in red and the isosurfacing results at time steps: 46, 62, 78 and 94. The contour values used for the isosurface extraction were kept constant across all time steps: 1E-6, 8.5E-5, 1.7E-4, 2.55E-4, 3.4E-4 and 4.25E-4.

```
BURNSAT
1 50001U 10001A   10284.00000000  .00000000  00000-0  00000-0 0    01
2 50001 078.0000   0.0000 0050000   0.0000   0.0000 14.42576614    02
TDRS_BURN
1 27566U 00000000 10265.70740741  .00000000  00000-0  00000-0 0 00003
2 27566  27.0277 179.7786 7261736 179.8588 359.4731  2.27457895    07
TEST_TV_SRC
1 50001U 10001A   10284.00000000  .00000000  00000-0  00000-0 0    01
2 50001 033.5000 100.0000 4000000   0.0000   0.0000  2.59338460    02
TEST_TV_TGT
1 50002U 10001A   10284.00000000  .00000000  00000-0  00000-0 0    01
2 50002 000.0000   0.0000 0001000   0.0000  90.0000  0.99164129    02
```

Fig. 7. The TLEs used for the three simulations: BURNSAT, TDRS_BURN and TEST_TV.

For the second simulation, TDRS_BURN, we also kept the burn time fixed, kept the thrust direction uniform, and only varied the burn magnitude between 0.0 and 2.0 km/s. The propagation time spans from 55464.6 to 55464.85 in MJD. We used the same spherical grid with a resolution of (360,180,500), where positions along the radial direction are given by: 10^(i*0.0023906+6.80615), for 0 $\leq i \leq$ 499. For this simulation, we only generated 1E6 virtual trajectories for 180 time steps, with a step size of 120 seconds. The TLE for this simulation is given in the middle row of Fig. 7. The out-of-core rendering images in Fig. 2 were generated using this simulation. The bottom row of Fig. 8 shows the orbit in red and the isosurfacing results at time steps: 45, 80, 115 and 150. The contour values used for the isosurface extraction were kept constant across all time steps: 1E-6, 2E-5, 4E-5, 8E-5, 1.6E-4, 3.2E-4, 6.4E-4 and 1.28E-3.

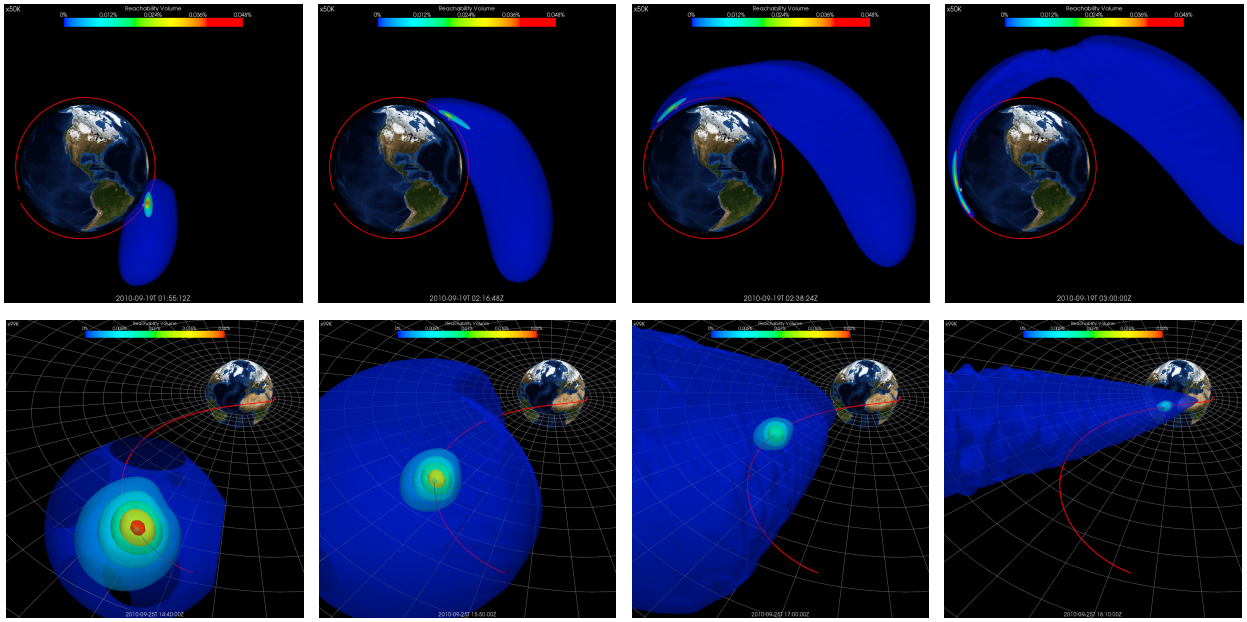

Fig. 8. Top row shows the BURNSAT orbit in red and the isosurfaces of the RV as they evolve across time. Bottom row shows the TDRS_BURN orbit and the corresponding isosurfaces for its RV across time.

In the third simulation, TEST_TV, we experimented with how to compute and visualize a restricted RV (or TV) by specifying a source object in LEO and a target object in GEO. For the restriction step, we kept only those virtual trajectories that arrive at the target over the period of interest. For this simulation, we varied the burn time uniformly over 3 days, kept the thrust direction uniform, and varied the burn magnitude between 0.0 and 3.5 km/s. The propagation time spans from 55480.0 to 55483.0 in MJD. Rather than using a spherical curvilinear grid as in the previous two simulations, we used a rectilinear grid instead, with a resolution of (400,400,400) and bounds between -200E6 m to 200E6 m in ECEF coordinate system. For this simulation, we initially generated 692E6 virtual trajectories, which after filtering were reduced to 2.5E5. The TLE for this simulation is given in the bottom row of Fig. 7. The top row of Fig. 9 shows the results from the out-of-core rendering of the virtual trajectories at time steps: 104, 536, 968 and 1400. The bottom row of Fig. 9 shows the corresponding results from the isosurfacing of the TV. The contour values used for the isosurface extraction were kept constant across all time steps: 1E-6, 4E-4, 8E-4, 1.6E-3, 3.2E-3 and 4.8E-3. (Note the grid shown in the images of Fig. 9 corresponds to the equatorial plane, and it was included to provide a visual context – it was not part of the rectilinear grid used to compute the TV.)

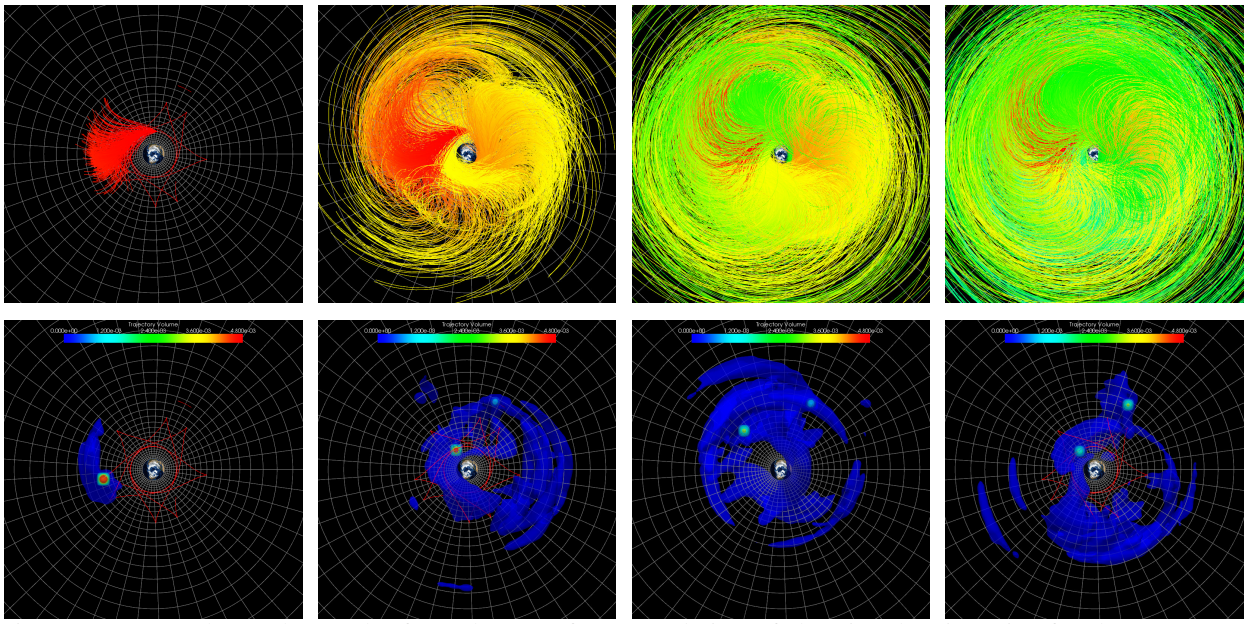

Fig. 9. Top row shows the results from the out-of-core rendering of the virtual trajectories for the TEST_TV simulation at time steps: 104, 536, 968 and 1400. Bottom row shows the corresponding results from the isosurfacing of the TV.

6. CONCLUSION

In this paper, we presented a Monte Carlo approach to computing the RV for a given object and several techniques for visualizing the RV and its virtual trajectories. Our approach samples all possible trajectories by randomizing thrust-vectors, thrust magnitudes and time of burn. At any given instance, the distribution of the "point-cloud" of the virtual particles defines the RV. For short orbital time-scales, the temporal evolution of the point-cloud can result in complex, multi-reentrant manifolds. To computationally represent the volume, we utilize a structured grid in Earth-centered coordinate system. At each time step, the underlying probability density distribution is estimated by first accumulating the number of trajectory points that are contained within the grid cells and then normalizing the accumulation by the total number of trajectory points. We also presented a real-time out-of-core rendering approach for visualizing the large number of virtual trajectories. Our out-of-core rendering approach can operate on the point cloud through time steps or subsets of virtual trajectories. We also examined different techniques for visualizing the computed volume of probability density distribution, including volume slicing, convex hull and isosurfacing. We compared and contrasted these techniques in terms of computational cost and visualization effectiveness. We also described the main implementation issues encountered during our development process. Finally, we presented results from our end-to-end system for computing and visualizing RVs using three different simulations of maneuvering satellites.

For future work, we plan to investigate adaptive techniques, such as adaptive mesh refinement, for computing the RV in order to minimize the high computational and storage requirements for large number of virtual trajectories. As an alternative to our approach of accumulation and normalization for estimating the probability density distribution, we plan to examine kernel density estimation [7]. We also plan to investigate other spatial partitioning schemes [11] based on the sphere that can overcome the pole singularity issue as well as the boundary condition issue of the spherical grid. In terms of visualization techniques, we plan to examine direct volume rendering techniques [9, 18], such as volume ray casting and splatting, and transfer function design for alternative ways to visualize the underlying probability density distribution.

7. ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-499168.

8. REFERENCES

1. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. *Springer*, 1997.
2. P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in Higher Dimensions. *IEEE Visualization Conference*, pp. 267-273, 2000.
3. U. Bordoloi, D. L. Kao, and H.-W. Shen. Visualization Techniques for Spatial Probability Density Function Data. *Data Science Journal*, 3:153-162, 2004.
4. P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158-170, 1997.
5. C. D. Hansen and C. R. Johnson. The Visualization Handbook. *Elsevier*, 2004.
6. M. Jiang, M. Andereck, A. J. Pertica, and S. S. Olivier. A Scalable Visualization System for Improving Space Situational Awareness. *Advanced Maui Optical and Space Surveillance Technologies Conference*, pp. 603-612, 2010.
7. O. D. Lampe and H. Hauser. "Interactive Visualization of Streaming Data with Kernel Density Estimation. *Pacific Visualization Symposium*, pp. 171-178, 2011.
8. W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163-169, 1987.
9. N. Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99-108, 1995.
10. S. S. Olivier, K. Cook, B. Fasenfest, D. Jefferson, M. Jiang, J. Leek, J. L. Levatin, S. Nikolaev, A. J. Pertica, D. W. Phillion, H. K. Springer, and W. H. de Vries. High-Performance Computer Modeling of the Cosmos-Iridium Collision. *Advanced Maui Optical and Space Surveillance Technologies Conference*, pp. 720-731, 2009.
11. E. Praun and H. Hoppe. Spherical Parametrization and Remeshing. *ACM SIGGRAPH Conference*, pp. 340-349, 2003.
12. C. T. Silva, Y.-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-Core Algorithms for Scientific Visualization and Computer Graphics. *IEEE Visualization Conference Course Notes*, 2002.
13. H. K. Springer, W. O. Miller, J. L. Levatin, A. J. Pertica, and S. S. Olivier. Satellite Collision Modeling with Physics-Based Hydrocodes: Debris Generation Predictions of the Iridium-Cosmos Collision Event and Other Impact Events. *Advanced Maui Optical and Space Surveillance Technologies Conference*, pp. 362-372, 2010.
14. D. A. Vallado. Fundamentals of Astrodynamics and Applications. *Microcosm Press*, 2007.
15. J. S. Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys*, 33(2), pp. 209-271, 2001.
16. W. H. de Vries and D. W. Phillion. Monte Carlo Method for Collision Probability Calculations Using 3D Satellite Models. *Advanced Maui Optical and Space Surveillance Technologies Conference*, pp. 123-132, 2010.
17. C. Wang and Y.-J. Chiang. Isosurface Extraction and View-Dependent Filtering from Time-Varying Fields Using Persistent Time-Octree (PTOT). *IEEE Transaction on Visualization and Computer Graphics*, 15(6):1367-1374, 2009.
18. L. Westover. Footprint Evaluation for Volume Rendering. *ACM SIGGRAPH Conference*, pp. 367-376, 1990.