AAS 18-096

FAST SPACECRAFT SOLAR RADIATION PRESSURE MODELING BY RAY-TRACING ON GRAPHICS PROCESSING UNIT

Patrick W. Kenneally*[*] and Hanspeter Schaub[†]

A description of a method for computing on the graphics processing unit the force and torque on a spacecraft due to solar radiation pressure. The method employs ray-tracing techniques, developed in the graphics rendering discipline, to resolve spacecraft self-shadowing and self-reflections at faster than real-time computation speed. The primary algorithmic components of the ray-tracing process which contribute to the method's computational efficiency are described. These components include bounding volume hierarchy acceleration data structures, fast ray to bounding box intersection testing using the slab intersection algorithm and fast triangle intersection testing using the Möller-Trumbore algorithm. The process is implemented using C++ and OpenCL and executed on a consumer grade graphics processing unit. Initial model validation is presented comparing computed values to both the analytic cannonball model and ray traced LAGEOS II spacecraft model. A performance analysis and characterization of the effect on performance of multiple ray bounces is presented using the Mars Reconnaissance Orbiter spacecraft.

INTRODUCTION

Effective orbit determination, maneuver and mission design and mission numerical simulations require tools that enable accurate modeling of the spacecraft dynamical system. Solar radiation pressure (SRP), the momentum imparted to a body by impinging solar photons, becomes a dominant non-conservative force above the Low Earth Orbit (LEO)[1] region. For example, to maintain a desired spacecraft attitude, the SRP-induced torque on a spacecraft is absorbed using reaction wheel devices. Under the influence of sustained torque in a constant direction, the reaction wheels will reach an operational maximum angular rate and require desaturation. The requirement to perform desaturation operations may be mitigated through a judicious choice of reaction wheel orientation or more typically by a momentum unloading process using spacecraft thrusters.[2] Given the importance of SRP, knowledge of the resultant forces upon a body due to SRP are a primary consideration in the modeling and analysis of spacecraft operating above the LEO region.[3,4]

The video game industry's pursuit for more vivid artificial worlds has driven the development of highly optimized vector processing software and graphics processing unit (GPU) computer hardware capable of carrying out many thousands of floating point operations in parallel.[5] In the animation and movie industry, the pursuit of photo-realistic modeling has pushed the techniques employed in ray-tracing algorithms to produce rendering results at near real-time computation speeds. Two key themes in ray-tracing research are the pursuit of algorithmic techniques and efficient hardware utilization, which increase computing efficiency, and therefore reduce the time of a photo-realistic

[*]Graduate Research Assistant, Aerospace Engineering, University of Colorado, Boulder.

[†]Alfred T. and Betty E. Look Professor of Engineering, Aerospace Engineering Sciences, University of Colorado at Boulder, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO 80309-0431, AAS Fellow

model rendering.[6] The algorithmic techniques developed in the pursuit of photo-realistic model rendering have provided the tools which are leveraged in the faster than real-time SRP ray-tracing methodology presented in this paper.

A survey of the current landscape of SRP research reveals a variety of approaches. The nature of the approaches can be characterized as analytic, semi-analytic or empirical. Whereas analytic models rely only on pre-launch engineering information, empirical models are constructed post-launch using flight data. Commonly, a semi-analytic model is used during a mission. These models are comprised of both analytic and empirical components with tunable parameters. Prior to flight, the tunable parameters are determined using an analytic model. Following launch, the parameters are incorporated into a parameter estimation process which tunes the model to more closely match flight data. Prominent examples of the three modeling approaches include the ROCK42 analytic model, the Bern semi-analytic model and the Jet Propulsion Lab (JPL) empirical model.[7,8]

The most basic analytic model employed is referred to as the cannonball model. The cannonball model, given in Eq. (1), is computed from the surface area upon which radiation is incident A, solar flux $\Phi_{\odot}$, the spacecraft mass M, speed of light c, heliocentric distance to the spacecraft r and the reflection, absorption and emission characteristics of the spacecraft surface which are grouped together within the coefficient of reflection C_r.

$$\boldsymbol{a}_{\odot} = -C_{\mathrm{r}} \frac{A\Phi_{\odot}}{Mc} \left(\frac{1AU}{r} \right)^2 \hat{\boldsymbol{s}} \tag{1}$$

Increased accuracy in analytic models is often achieved by representing the spacecraft as an approximation of various volumes. A common approximation is to model the spacecraft bus and solar panels as a box and panels respectively. Additionally, the individual reflection, absorption and emission material characteristics are kept distinct for each surface and set based on known spacecraft material properties.[9] However, common among shape approximation methods is that they are augmented and become semi-analytic models where much of the modeling uncertainty is delegated to a parameter estimation process and the model is 'tuned' post-launch to more accurately match spacecraft tracking data.

Notably, Ziebart et al., develop an analytic modeling approach based on ray-tracing techniques for the assessment of SRP force analysis of spacecraft in the GLONASS constellation.[10] Ziebart's method precomputes the body forces over all 4π steradian attitude possibilities. Ziebart's approach is also capable of modeling self-shadowing and multiple solar radiation ray reflection by ray-tracing a spacecraft model that comprises a set of volume primitives (boxes, cylinders etc.). McMahon and Scheeres extend Ziebart's approach to a semi-analytic model by aggregating the resultant SRP forces into a set of Fourier coefficients of a Fourier expansion.[9] The resulting Fourier expansion is available for both online and offline evaluation within a numerical integration process. Evaluation of the Fourier expansion in numerical simulation demonstrates successful prediction of the periodic and secular effects of SRP. Additionally, the Fourier coefficients may replace spacecraft material optical properties as parameters estimated during the orbit determination effort.

More recently, methods that make use of the parallel processing nature of GPUs have been developed. Tanygin and Beatty employ modern GPU parallel processing techniques to provide a significant reduction in time-to-solution of Ziebart's "pixel array" method.[11] In previous work presented by the authors the GPU computation environment OpenGL, a vector graphics software interface common in video games, is used to dynamically evaluate the force of the incident solar radiation across a spacecraft structure approximated by many thousands of facets.[12]

The method presented here leverages advances in ray-tracing and the OpenCL application programming interface (API) to produce a ray-tracing SRP modeling approach at faster than real-time computation speeds. OpenCL is an API and C based programming language which facilitates the execution of massively-parallel computations on heterogeneous computation devices. OpenCL is a cross-platform standard for parallel programming across a range of devices including multicore CPUs, GPUs and other computation accelerators.

This ray-tracing method is a departure from previous approaches presented by the authors. Previous approaches have focused on employing the OpenGL vector graphics render pipeline to compute the per facet force and torque of a triangulated mesh model.[12] The OpenGL method provides a high-geometric fidelity SRP computation of the spacecraft mesh, however, it is unable to capture self-reflections. The approach presented addresses the shortcomings of the OpenGL method by using proven ray-tracing algorithms implemented with OpenCL on the GPU and provides the following features:

- Faster than real-time SRP induced force and torque.
- Capturing spacecraft self-shadowing and multiple light ray bounces.
- Employing a wide variety of material optical properties.

In the remainder of this paper the fundamental components and initial results are outlined for the OpenCL ray-tracing methodology. In section two, primary considerations are given to porting the serial and recursive CPU ray-tracing execution to the parallel and iterative GPU execution environment. Additionally, an overview of the ray-tracing methodology implemented is provided. In section three, key algorithm components are described and their importance in a GPU ray-tracing implementation is discussed. Finally, in sections four and five the effect of multiple ray bounces in resolving the SRP force and the faster than real-time computational performance is discussed.

PARALLEL RAY-TRACING

The goal of ray-tracing is to compute the color in a pixel within a view port. The light arriving at the pixel is traced backwards through the scene where its scene interactions are modeled providing the final color of the view port pixel. In a serial execution environment ray reflections are computed using a recursive algorithm. The recursive algorithm tests for a ray intersection in the scene, and in the case where it finds an intersection, the same intersection search algorithm is initiated again to trace the ray in the new reflected direction. A common recursion termination condition is a maximum number of ray reflections.

The parallel GPU computing environment requires two primary changes to the serial ray-tracing algorithm. The first is required because recursive function execution is not available in current GPU execution environments. As a result, the recursive computation of ray reflections must be achieved through iteration. The second change is that rather than making the algorithm parallel by pixel as is suggested by the serial implementation, the algorithm should be parallel by ray. The Single Instruction Multiple Device (SIMD) GPU execution environment is most efficient when developers ensure that each compute unit on the GPU is actively working. In the case that the algorithm is parallel by pixels, rays from certain pixels will terminate sooner than others. This leaves compute units inactive resulting in poor utilization of the GPU's computing resources. Rather, an algorithm which is parallel by rays cast may discard terminated rays at each iteration. The reflected rays are then repacked for a second iteration ensuring all compute units marshaled are active.

To initialize the process, a spacecraft CAD model is input as a triangulated mesh with material definitions for the absorption, diffusion and specular optical characteristics. The mesh is then processed to generate the bounding volume hierarchy (BVH) data structure to accelerate the processes of intersection testing. With initialization now complete, the parallel ray-tracing algorithm can be executed on the GPU. The ray plane definition is copied to the GPU along with the BVH traversal structure and the spacecraft mesh material definitions. The algorithm then iterates through ray generation, BVH traversal, intersection testing and SRP computation until all rays have reached the set termination condition. In this work, the conditions for ray termination are that a ray either exits the scene or completes three ray reflections. The aggregated force and torque values are then returned to the CPU bound process where the values can be integrated into the dynamics propagation component of a spacecraft numerical simulation.

ALGORITHM COMPONENTS

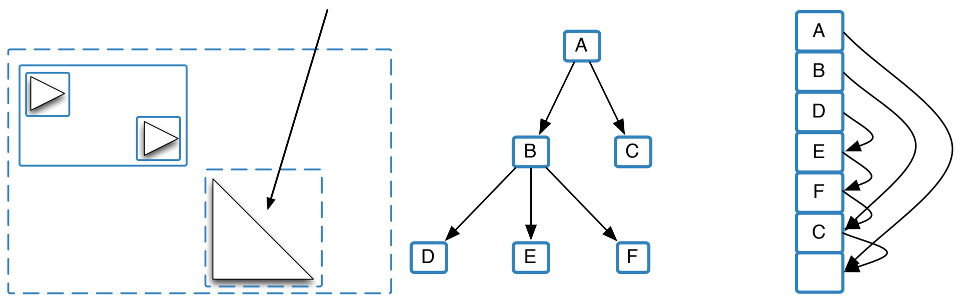
The presented approach employs a number of key algorithms to minimize the otherwise high computational load of a naive ray-tracing algorithm. These techniques include:

- Generation of an acceleration data structure in particular that of a BVH.
- Bounding box intersection testing algorithm using clipping planes.
- The computationally fast Möller-Trumbore ray and triangle facet intersection algorithm to test for intersections with a spacecraft facet and ray.

Acceleration Structures

Many acceleration structures are presented in ray and path tracing literature. Each of these structures offer advantages and disadvantages which are typically dependent on the model to be rendered.[13] This method employs a simple BVH to efficiently reduce the ray intersection search space and therefore the required ray intersection computations performed. Figure 1(a) portrays a notional BVH, demonstrating that the example ray need only test two volumes before testing for a triangle intersection. To begin building the BVH, a bounding volume is computed for each triangular facet in the spacecraft mesh model. In this implementation, the bounding volume is computed as a bounding box aligned to the spacecraft model body frame. The list of bounding volumes is sorted along the first spacecraft body frame axis, then divided in half and a new bounding volume is computed around each half of the list. This process is carried out recursively while, at each new division, the sort axis selected is sequentially the next axis in the body frame triad. This results in a BVH that groups successive bounding volumes as containing facets spatially near to each other.

An efficient method of traversing the BVH is a key aspect of the development of real-time SRP ray-tracing.[13] This implementation uses as the BVH traversal method a depth-first search array as described by Smits.[13] An example BVH comprising 6 nodes is shown in Figure 1(b) first as a recursive depth-first search-tree and second as a depth-first search array with precomputed skip pointers. In the recursive tree structure, if bounding volume node A is intersected, the search recursively descends to test for an intersection against node B. If no intersection is found at node B the recursion meets a termination condition and the search moves back up the tree and proceeds down the next branch to test node C.



(a) Notional BVH intersection testing (b) Two BVH traversal structures

Figure 1. The BVH traversal structures are constructed to provide efficient traversal. The left structure demonstrates a simple recursive BVH traversal. The right demonstrates the same BVH as shown on the left yet organized as a depth-first search array with precomputed node skip pointers.

For the depth-first search array, if bounding volume A is intersected, then the next node to be tested is the next node sequentially in the array, node B. If the bounding volume at node B is not intersected, the next node is found by following the precomputed skip pointer to the next sibling in the array, which for node B is node C.

The depth-first search array avoids the function call overhead inherent in a recursive search-tree traversal and takes advantage of the fact that the next node in the search-tree can be precomputed and stored with the left-most sibling as a skip-pointer to the next node. An additional benefit to the array BVH traversal, particularly for large meshes, is the greater memory coherency which yields more efficient contiguous memory accesses on the GPU.[13]

Bounding Volume Intersection

Bounding volume intersection uses the algorithm originally presented by Kay and Kajiya.[14] The algorithm models the bounding box as 3 sets of parallel planes. The algorithm employs each set of parallel planes as clipping planes. As demonstrated in Figure 2, once the ray is clipped by each set of planes, any remaining portion of ray inside the bounding volume indicates an intersection. The algorithm is particularly suited to implementation in the GPU environment because it does not require code branching (the execution of divergent code paths based on conditional code statements). Modern floating-point instruction sets are capable of computing the minimum or maximum, between two values, without code branching. This parallel plane algorithm employs the non-branching minimum and maximum functions and results in an intersection test with no code branching or division operations.

Triangle Facet Intersection

The spacecraft model mesh is comprised of many thousands of triangular facets. To compute a ray to triangle intersection, the Möller-Trumbore algorithm is used. This algorithm is a fast and memory-efficient ray to triangle intersection algorithm making it ideal for use in the memory constrained GPU computation environment. The basis of the algorithm is the knowledge that the point of intersection of a line through a triangle in barycentric coordinates (u, v) must lie within coordinate bounds which are easily testable as boolean values. The bounds defined by the barycentric coordinate system require $u \geq 0$, $v \geq 0$ and $u + v \leq 1$.[15]

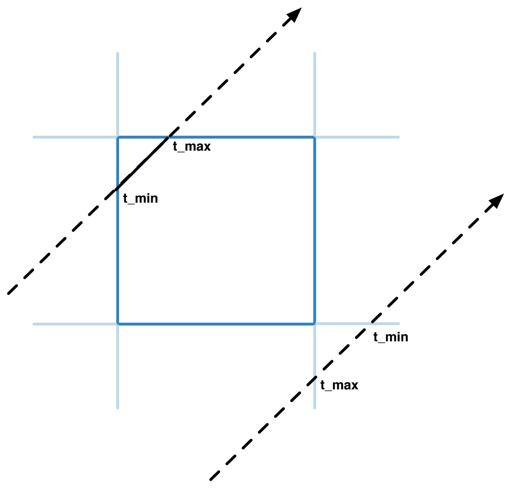


Figure 2. Example results of the parallel plane bounding box intersection algorithm. For the top left ray intersection, the algorithm returns t_max as greater than or equal to t_min. For the bottom right ray miss, the algorithm returns t_max as less than t_min.

To begin, a point, $T(u, v)$, on a triangle described by vertices $\boldsymbol{V}_0, \boldsymbol{V}_1$ and $\boldsymbol{V}_2$ and mapped to barycentric coordinates is described as given in Eq. (2).

$$\boldsymbol{T}(u, v) = (1 - u - v)\boldsymbol{V}_0 + u\boldsymbol{V}_1 + v\boldsymbol{V}_2 \tag{2}$$

The ray equation is given in Eq. (3) where $\boldsymbol{O}$ is the ray origin, t the distance from the ray origin to the intersection point and $\boldsymbol{D}$ the ray direction.

$$\boldsymbol{R}(t) = \boldsymbol{O} + t\boldsymbol{D} \tag{3}$$

It is then evident that for a ray to intersect the barycentric description of the triangle, the ray equation must be equal to a point on the triangle, $\boldsymbol{R}(t) = \boldsymbol{T}(u, v)$, and results in the expression at Eq. (4).

$$\boldsymbol{O} + t\boldsymbol{D} = (1 - u - v)\boldsymbol{V}_0 + u\boldsymbol{V}_1 + v\boldsymbol{V}_2 \tag{4}$$

Rearranging the equation into a matrix form yields Eq. (5) where it is evident that the terms $\boldsymbol{V}_1 - \boldsymbol{V}_0$ and $\boldsymbol{V}_2 - \boldsymbol{V}_0$ are the edges of the triangle and are substituted for $\boldsymbol{E}_1 = \boldsymbol{V}_1 - \boldsymbol{V}_0$ and $\boldsymbol{E}_2 = \boldsymbol{V}_2 - \boldsymbol{V}_0$. Additionally, the substitution $\boldsymbol{T} = \boldsymbol{O} - \boldsymbol{V}_0$ can be made and is interpreted as a translation of the ray origin to the barycentric coordinate frame origin.

$$[-\boldsymbol{D}, \boldsymbol{V}_1 - \boldsymbol{V}_0, \boldsymbol{V}_2 - \boldsymbol{V}_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \boldsymbol{O} - \boldsymbol{V}_0 \tag{5}$$

Using Cramer's rule, a solution can be found for u, v and t as shown in Eq. (6). The solution for u, v and t will provide the values with which to test against the barycentric coordinate bounds conditions.

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-\boldsymbol{D}, \boldsymbol{E}_1, \boldsymbol{E}_2|} \begin{bmatrix} |\boldsymbol{T}, \boldsymbol{E}_1, \boldsymbol{E}_2| \\ |-\boldsymbol{D}, \boldsymbol{T}, \boldsymbol{E}_2| \\ |-\boldsymbol{D}, \boldsymbol{E}_1, \boldsymbol{T}| \end{bmatrix} \tag{6}$$

A final solution is computed with slightly more efficiency by recognizing that each determinant of the form $|\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}| = -(\boldsymbol{A} \times \boldsymbol{B}) \cdot \boldsymbol{C}$, is shown in Eq. (7). This is further simplified by computing the cross products $\boldsymbol{P} = (\boldsymbol{D} \times \boldsymbol{E}_2)$ and $\boldsymbol{Q} = (\boldsymbol{T} \times \boldsymbol{E}_1)$ once and then substituting, yielding Eq. (8).

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(\boldsymbol{D} \times \boldsymbol{E}_2) \cdot \boldsymbol{E}_1} \begin{bmatrix} (\boldsymbol{T} \times \boldsymbol{E}_1) \cdot \boldsymbol{E}_2 \\ (\boldsymbol{D} \times \boldsymbol{E}_2) \cdot \boldsymbol{T} \\ (\boldsymbol{T} \times \boldsymbol{E}_1) \cdot \boldsymbol{D} \end{bmatrix} \tag{7}$$

$$= \frac{1}{\boldsymbol{P} \cdot \boldsymbol{E}_1} \begin{bmatrix} (\boldsymbol{Q} \cdot \boldsymbol{E}_2) \\ (\boldsymbol{P} \cdot \boldsymbol{T}) \\ (\boldsymbol{Q} \cdot \boldsymbol{D}) \end{bmatrix} \tag{8}$$

COMPUTING SOLAR RADIATION PRESSURE

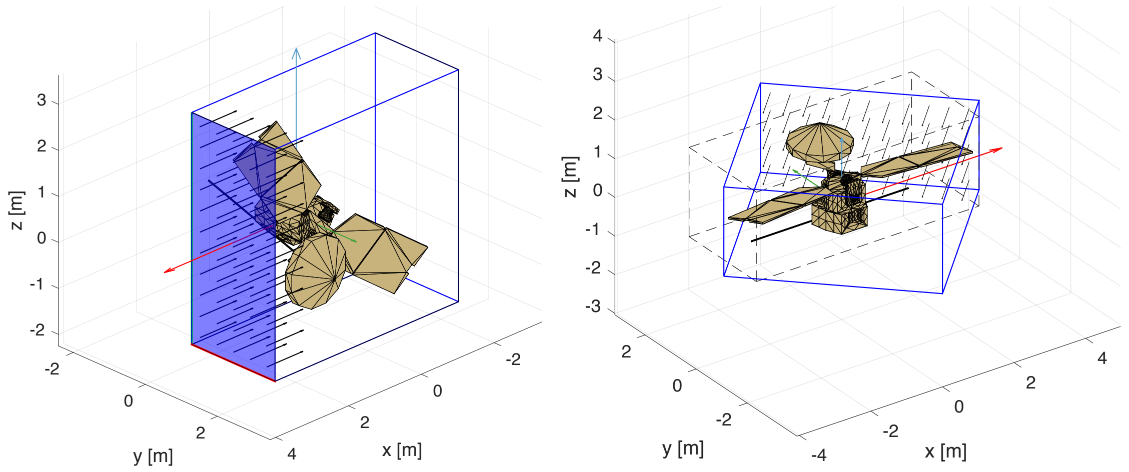
At each time update, a new wave of ray vectors is generated. The spacecraft-to-sun unit direction vector $\hat{\boldsymbol{s}}_B$ is computed and used as the first axis in an orthogonal Sun S frame. The direction cosine matrix $[SB]$ which defines the rotation from the body frame B to the S frame is constructed and used to generate an S frame axis-aligned bounding box of the mesh model. As shown in Figure 3(a), the side of the bounding box nearest the sun is used as a finite plane from which the origins of all ray vectors is defined. The wave of rays are computed in parallel using a dedicated OpenCL ray generation kernel.

The ray plane is divided into unit areas determined by the resolution units chosen by the user. For example, a 2 m x 1 m plane can be divided into 10 mm sized squares giving a plane of 200 x 100 squares, producing 20000 rays. The discretization of the incident radiation wave front has the potential to introduce errors into the computation. The error source is due to the discretization of the surface area over the spacecraft which is intersected by the unit area of an individual ray. Ziebart shows in a study of this discretization error that for representative test geometries, the error, for a maximum ray cross section of 10 mm^2, is 2% and decreases to less than a percent for ray cross sections of less than 5 mm.[10] Paying heed to Ziebart's study, the maximum ray resolution used in this method is 10 mm. The origin for a ray is taken as the corresponding center of a unit area and the direction for all rays is taken as $-\hat{\boldsymbol{s}}_B$. The ray intersection testing must occur in the same coordinate frame in which the spacecraft vertices are defined. As a result, the ray vectors are mapped from Sun frame S to the body frame B using the $[BS]$ rotation matrix.

Each compute unit on the GPU launches an instance of the coded OpenCL kernel program. Each kernel instance accesses the ray wave front data in the GPU global memory space copying a specific ray and the BVH traversal array to local memory in the compute unit. A wave of rays is then tested for intersections until all rays have been tested. If a BVH intersection is found and the intersection is a terminal node, then the triangle intersection is computed and the index of the facet and the intersection location are recorded and placed in an intersection array. If no intersection is computed, the miss is similarly recorded in the intersection array.

Currently, this method accommodates only the modeling of specular ray reflections. The angle of incidence is equal to the angle of reflection for a specularly reflected ray.[10] The reflected unit direction vector for the i^{th} ray intersection is computed as given in Eq. 9, where $\hat{\boldsymbol{r}}_{i_{\text{ref}}}$ is the reflected unit direction vector, $\hat{\boldsymbol{r}}_i$ the impinging ray direction and $\boldsymbol{n}_k$ is the k^{th} triangle facet unit normal vector.

$$\hat{\boldsymbol{r}}_{i_{\text{ref}}} = \hat{\boldsymbol{r}}_i - 2(\hat{\boldsymbol{r}}_i \cdot \hat{\boldsymbol{n}}_k)\hat{\boldsymbol{n}}_k \tag{9}$$



(a) S frame generated rays (b) S frame rays mapped to the B frame

Figure 3. The MRO mesh model surrounded by a B frame oriented bounding box (dashed black) and an S frame bounding box (blue solid). The red, green and blue vectors are the B frame axes and the black vectors indicates rays originating from the blue ray-plane.

As shown in Eq. 10, this work uses Ziebart's formulation to compute the radiance, where the irradiance is scaled per unit area by the reflectivity ν and specularity μ of the intersected triangular facet's material.

$$\boldsymbol{I}_{\text{ref}} = \mu\nu\boldsymbol{I}_i \tag{10}$$

Force and Torque Computation

Where an intersection is found, the force on the spacecraft due to the incident radiation flux of the ray is evaluated in the spacecraft body frame using the expression for the normal and shear components of the force as given in Eq. (11) and Eq. (12) respectively. In these two expressions $P(|\boldsymbol{r}_\odot|)$ is the solar radiation pressure scaled by the heliocentric distance to the spacecraft, k is the ray index, i is the index of the intersected facet, A_k is the cross sectional area of the ray, θ_k defines the sun angle of incidence relative to the intersected facet normal vector and $\hat{\boldsymbol{s'}}_k$ is the shear direction vector as defined by Ziebart.[10]

$$\boldsymbol{F}_{normal_k} = -P(|\boldsymbol{r}_\odot|)A_k\cos(\theta_k)\left[(1+\mu_i\nu_i)\cos(\theta_k) + \frac{2}{3}\nu_i(1-\mu_i)\right]\hat{\boldsymbol{n}}_k \tag{11}$$

$$\boldsymbol{F}_{shear_k} = -P(|\boldsymbol{r}_\odot|)A_k\cos(\theta_k)\sin(\theta_k)(1-\mu_i\nu_i)\hat{\boldsymbol{s'}}_k \tag{12}$$

The total force is computed by summing the force components for each ray as shown in Eq. (13), where n equals the total number of rays.

$$\boldsymbol{F}_\odot = \sum_{k=1}^{n}(\boldsymbol{F}_{shear_k} + \boldsymbol{F}_{normal_k}) \tag{13}$$

Following the force computation, the torque $\boldsymbol{L}_k$ contribution of a single intersection, as given in Eq. (14), is computed as the cross product of the vector defined from the body frame origin to the ray intersection point $\boldsymbol{c}_k$ and the total ray force $\boldsymbol{F}_{\odot_k}$.

$$\boldsymbol{L}_k = \boldsymbol{c}_k \times \boldsymbol{F}_{\odot_k} \tag{14}$$

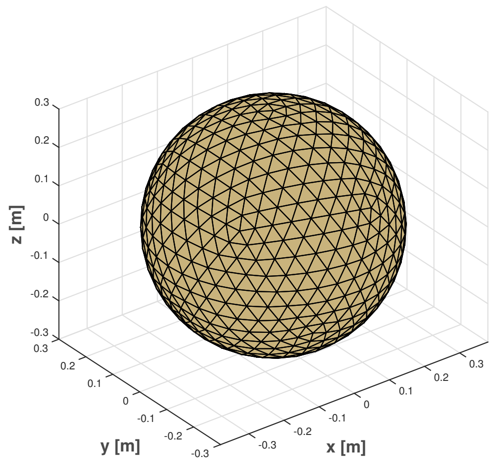


Figure 4. LAEGOS mesh model.

Table 1. SRP acceleration for each method.

Model	SRP Acceleration
Cannonball	3.58×10^{-9} [m/s^2]
Ray-Traced Cannonball	3.56×10^{-9} [m/s^2]

INITIAL MODEL VALIDATION

The initial validation is performed by comparing results from an analytic cannonball model and a sphere-shaped spacecraft model evaluated by the OpenCL ray-tracing method. The values for the LAGEOS II spacecraft are as follows; a mass of 405.38 kg, area of 0.2817 m^2, Φ (at 1 AU) is 1.38×10^3 W/m^2 and a coefficient of reflection C_r of 1.12. These parameters are used in both the cannonball and ray-traced evaluations. A spherical model spacecraft is generated to replicate the LAGEOS II spacecraft parameters. Figure 4 illustrates the spherical spacecraft model being tested. The perturbative acceleration due to SRP as given by the cannonball model and the OpenCL model are given in Table 1. The agreement between the cannonball model and the ray-traced method provides initial confidence of the method's correctness.

FORCE RESOLUTION

A key feature of the ray-tracing approach is its ability to capture and resolve the impact of spacecraft self-reflection on the resultant SRP force. An initial analysis is performed to demonstrate the impact that the number of live reflected rays has on the resultant force magnitude and direction. For this analysis, a 1424 primitive mesh of the MRO spacecraft is processed at a heliocentric distance of 1AU where the sun vector defined in the body frame as $\hat{\boldsymbol{s}}_B = [-0.44, 0.90, 0.06]^T$. The ray resolution used is 10 mm which, results in a ray plane width of 497 and height of 286, generating 142,142 rays in the first wave.

The resulting force direction, magnitude and number of live rays is recorded for increasing numbers of ray bounces. Figure 5 shows the difference in each of the x, y and z body frame components between bounce one and two, bounce two and three, and so on. It is evident, for this spacecraft at this attitude, that after three or more bounces the difference in the direction of the resultant SRP force becomes significantly less than the difference prior to bounce three. Figure 6 presents the magnitude of the resultant force and the number of live rays at successive ray bounces. It is similarly evident that after three ray bounces, the force magnitude becomes stable with values between 3.60×10^{-5} N and 3.62×10^{-5} N. The number of live rays significantly decreases from 142,142 to 7551 between bounces one and three.

A visual demonstration of the above analysis is shown in Figures 7(a) to 7(c). Each figure shows

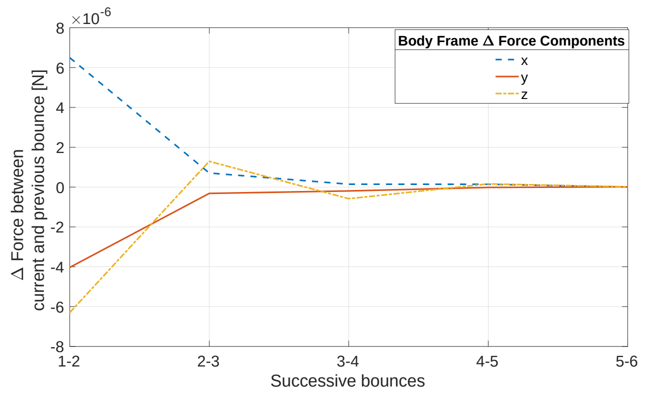


Figure 5. The delta in the direction of the resultant force between each successive ray bounce.

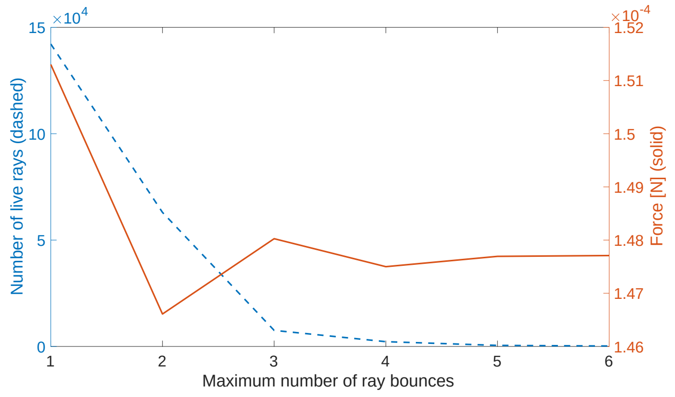
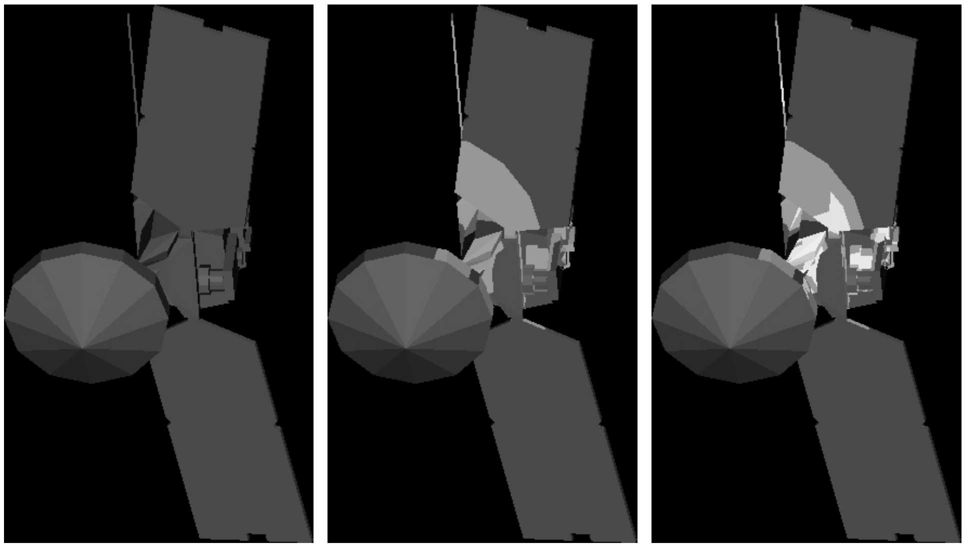


Figure 6. The number of live rays and resultant force magnitude at successive bounces.

the areas on the spacecraft mesh where initial, secondary and tertiary self-reflections occur. Increased grey scale brightness has been used to emphasize the areas impacted by successive bounces. It is expected that different spacecraft geometries and $\hat{s}$ orientations will yield variations in the number of ray bounces required to approach a solution with a minimal delta in the resultant force between successive ray bounces. This analysis reaffirms that resolving spacecraft self-reflections is a significant component of computing a more accurate SRP force.

COMPUTATIONAL PERFORMANCE

A primary goal of the OpenCL ray-tracing method is efficient computational performance resulting in faster than real-time evaluation. An analysis is performed to characterize the execution time of the method. The execution time is computed as the time point when ray generation begins to the time point when the CPU-bound process receives the final force values. Intuitively, the execution duration is critically dependent on the number of rays cast during model evaluation. Increased resolution increases the required number of rays to be traced and therefore the execution time. Additionally, there is a fixed base duration due to latency introduced by the data transfer between the CPU and GPU memory spaces. This latency can be larger or smaller depending on the CPU/GPU architecture being employed. Generally, the latency is lower if the GPU resides on the same chip as the CPU, as opposed to being a separate hardware device for which data is transfered across a common transfer bridge such as PCI Express.[16]



(a) One bounce (b) Two bounces (c) Three bounces

Figure 7. Increasing areas of brightness (left to right) indicate mesh areas where self-reflected rays have impacted the spacecraft mesh.

To demonstrate the dependency of computation speed on ray resolution, the MRO model is evaluated for ray resolutions of 2.5 mm, 5 mm and 10 mm. The execution time is taken as an average over 30 seconds whereby the computer is configured similarly for each test. The computer used to produce the below results is a MacBook Pro with a 3.1 GHz Intel Core i7 CPU. The GPU employed is an AMD Radeon Pro 560 4GB. The results of the evaluations are show in Figure 8. As expected the, 10 mm ray resolution result in the fastest execution time of 1.15 ms, 1.83 ms and 2.4 ms for the one, two and three bounce evaluations respectively. The 5 mm case executes slightly slower 1.27 ms, 2.06 ms and 2.41 ms for each number of bounce evaluations. The significantly slower execution time of the 2.5 mm case can be explained by the fact that for particularly high ray resolutions the GPU can no longer process all rays in one execution. The C++ code which supports the interface between a numerical simulation and OpenCL on the GPU, is required to break up the ray plane into 'tiles' sufficient to fill the GPU. Each tile is submitted to the GPU for evaluation and recombined with all other tiles to provide the final ray-traced force evaluation.

CONCLUSIONS

This paper demonstrates how SRP forces and torques can be resolved for complex spacecraft structures more accurately and at high speed using an OpenCL GPU-focused ray-tracing methodology. Spacecraft self-shadowing and self-reflection are implicitly captured by a ray-tracing methodology resulting in faster than real-time spacecraft model evaluation. An initial analysis has shown that three or more ray bounces resolves a large majority of SRP force magnitude and direction variations caused by spacecraft self-reflection. This method presents mission analysts with a tool that requires minimal set up and makes use of the wealth of pre-launch spacecraft engineering data. Further validation and characterization of the method is currently being conducted where more accurate surface optical characteristics and their radiation interactions are modeled.

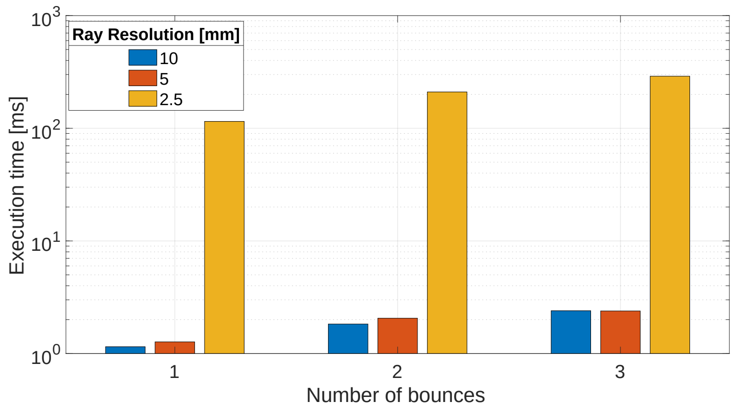


Figure 8. Execution times for 10 mm, 5 mm and 2.5 mm ray resolutions on the MRO spacecraft mesh.

REFERENCES

[1] D. Vallado, *Fundamentals of astrodynamics and applications*. New York: Springer, 2007.

[2] D. J. O'Shaughnessy, J. V. McAdams, P. D. Bedini, A. B. Calloway, K. E. Williams, and B. R. Page, "Messenger's use of solar sailing for cost and risk reduction," *Acta Astronautica*, Vol. 93, January 2014, pp. 483–489, 10.1016/j.actaastro.2012.10.009.

[3] H. F. Fliegel and T. E. Gallini, "Solar force modeling of block IIR Global Positioning System satellites," *Journal of Spacecraft and Rockets*, Vol. 33, No. 6, 1996, pp. 863–866, 10.2514/3.26851.

[4] J. A. Marshall, S. B. Luthcke, P. G. Antreasian, and G. W. Rosborough, "Modeling Radiation Forces Acting on TOPEX/Poseidon for Precision Orbit Determination," tech. rep., 1992.

[5] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, Vol. 96, No. 5, 2008, 10.1109/JPROC.2008.917757.

[6] I. Wald and P. Slusallek, "State of the art in interactive ray tracing," *State of the Art Reports, EUROGRAPHICS*, 2001, pp. 21–42, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.6266.

[7] T. A. Springer, G. Beutler, and M. Rothacher, "A New Solar Radiation Pressure Model for GPS Satellites," *GPS Solutions*, Vol. 2, No. 3, 1999, pp. 50–62, 10.1007/PL00012757.

[8] Y. E. Bar-Sever, "New and Improved Solar Radiation Models for GPS Satellites Based on Flight Data Final Report," tech. rep., Jet Propulsion Laboratory, 1997.

[9] J. W. McMahon and D. J. Scheeres, "New Solar Radiation Pressure Force Model for Navigation," *Journal of Guidance, Control, and Dynamics*, 2010, 10.2514/1.48434.

[10] M. Ziebart, *High Precision Analytical Solar Radiation Pressure Modelling for GNSS Spacecraft*. PhD thesis, University of East London, 2001.

[11] S. Tanygin and G. M. Beatty, "GPU-Accelerated Computation of SRP Forces with Graphical Encoding of Surface Normals," *AAS/AIAA Astrodynamics Specialist Conference*, Vail, CO, Aug. 9-13 2015.

[12] P. W. Kenneally and H. Schaub, "High Geometric Fidelity Modeling of Solar Radiation Pressure Using Graphics Processing Unit," *AAS/AIAA Spaceflight Mechanics Meeting*, Napa Valley, California, Feb. 14–18 2016. Paper No. AAS-16-500.

[13] B. Smits, "Efficiency Issues for Ray Tracing," *Journal of Graphics Tools*, Vol. 3, No. 2, 1999, pp. 1–14, 10.1080/10867651.1998.10487488.

[14] T. L. Kay and J. T. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, 1986, pp. 169–278.

[15] T. Moller and B. Trumbore, "Fast , Minimum Storage Ray / Triangle Intersection," *Journal of Graphics Tools*, Vol. 2, No. 1, 1997, pp. 21–28, 10.1145/1198555.1198746.

[16] C. Gregg and K. M. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer," *ISPASS 2011 - IEEE International Symposium on Performance Analysis of Systems and Software*, 04 2011, pp. 134–144.