IAC-17,C1,4,3,x40634

# Parallel Spacecraft Solar Radiation Pressure Modeling Using Ray-Tracing On Graphic Processing Unit

**Patrick Kenneally[a,*] and Hanspeter Schaub[b]**

[a] *Graduate Student, Aerospace Engineering Sciences, University of Colorado Boulder.*, patrick.kenneally@colorado.edu
[b] *Glenn L. Murphy Endowed Chair, Professor of Engineering, Department of Aerospace Engineering Sciences, University of Colorado, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO 80309-0431.*, hanspeter.schaub@colorado.edu
[*] Corresponding Author

**Abstract**

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, self-reflections and arbitrary spacecraft articulation at faster than real-time computation speed. The primary algorithmic components of the ray-tracing process which contribute to the methods 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 in the parallel execution environment of a consumer grade graphic processing unit. Initial model validation is presented comparing computed values to both the analytic cannonball model and post-launch data of the Mars Reconnaissance Orbiter spacecraft.

## 1. 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 Low Earth Orbit (LEO)[1] regime. 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 a torque in a constant direction the reaction wheels will reach an operational maximum angular rate and require desaturation. The requirment to perform desaturation operations may be mitigated through a judicious choice of reaction orientation or more typically by a momentum unloading process using spacecraft thrusters.[7] 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.[2,3]

The video game and animation industries are driving the pursuit to create more vivid and realistic artificial worlds. In the video game industry this pursuit has resulted in highly optimized vector processing software and graphic processing unit (GPU) computer hardware capable of carrying out many thousands of floating point operations in parallel.[4] 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 model rendering.[5] 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 high geometric fidelity SRP ray-tracing methodology presented in this paper.

The ability to model and compute faster than real-time, the SRP forces and torques on flexible and time varying spacecraft structures, presents compelling opportunities. Current SRP evaluation approaches are capable of modeling the resultant force of an articulated spacecraft where the articulation motion is known prior to evaluation as computed forces are saved to a lookup table in an off-line process.[6] However, there are many instances in which the articulation motion and the spacecraft state are dependent on the myriad spacecraft control inputs and constraints. Accounting for all possible permutations of the spacecraft dynamical state is further challenged by the inclusion of flexing in large spacecraft structures or time varying surface properties.

It is evident then that a method of SRP evaluation characterized by an ability to include time varying information of the spacecraft state has potential for a wide range of applications. Effective modeling of the SRP induced perturbation of a spacecraft enables mission designers to consider SRP as a valuable actuator rather than a disturbance. Such a novel use of the SRP force in maneuver and mission design is exemplified by the MErcury Surface, Space ENvironment, GEochemistry and Ranging (MESSENGER) mission. The MESSENGER mission designers employed a solar sailing technique to perform each trajectory change maneuver (TCM) and accurately target each of the mission's six planetary flyby maneuvers. Typically TCM's are performed using onboard thrusters. However, using SRP as the TCM actuator allowed the MESSENGER team to perform TCMs with more accuracy and finer control due to the smaller magnitude of the SRP induced acceleration.[7] Additionally, the MESSENGER team was able to reduce fuel and related structural accommodations in the spacecraft design to reduce overall mission cost.[8]

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. The majority of models used during flight guidance and navigation efforts are semi-analytic models. These models are comprised of both analytic and empirical components. Often an analytic model will be employed pre-flight and then post-launch the particular parameters in the model may be incorporated into a parameter estimation process. 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.[9, 10] The ROCK42 model is a pre-launch analytic model of the GPS Block II/IIA satellites which models the spacecraft structure and materials at component level detail. The Bern semi-analytic model extends the ROCK42 model by agumenting the final acceleration due to SRP computed by the ROCK42 with a set of 9 parameters which are derived from a post-launch orbit estimation effort. Finally, the JPL model is an empirical model that exclusively uses operational orbit estimation data to estimate the coefficients of a Fourier Series which when evaluated yields the SRP force on the spacecraft. The OpenCL ray-tracing modeling method presented may be classified as an analytic model and seeks to make extensive use of the pre-launch engineering data available to spacecraft engineering and operations teams.

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 $C_r$. It is often the case that the $C_r$ parameter is continually estimated and updated by an orbit determination effort. This model was most notably used during the Laser Geodynamics Satellites (LAGEOS) missions and continues to prove useful for initial mission analysis.[11]

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

Increased accuracy in analytic models is often achieved by defining the spacecraft as an approximations 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.[12] However, common among shape approximation methods is that they are augmented as semi-analytic models where much of the modeling uncertainty is lumped in 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.[13] Ziebart's method precomputes the body forces over all $4\pi$ 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.[12] 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.[14] In previous work presented by the authors the GPU computation environment OpenGL, a vector graphics GPU software interface common in video games, is used to dynamically render the spacecraft model and evaluate the force of the incident solar radiation across a spacecraft structure approximated by many thousands of facets.[?]

The method presented here leverages advances in ray-tracing and the OpenCL set of programming tools to produce a ray-tracing SRP modeling approach at faster than real-time computation speeds. OpenCL is an application programming interface (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. OpenCL facilitates both the read and write of data on processing unit(s) and the submission of code for execution on the processing unit.

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.[?] The OpenGL method provides a high-geometric fidelity SRP computation of the spacecraft mesh, however, it is unable to capture self-reflections. The OpenGL method employed a small modification to the OpenGL render pipeline to perform SRP calculations. While efficient to implement, utilizing the OpenGL render pipeline has two significant shortcomings. The first is the computational cost of initializing and executing the entire pipeline, even in the case that portions of the pipeline are redundant to the task of computing SRP forces. The second is the associated software development challenges, in particular debugging code executing on the GPU. The approach presented in the following, addresses the shortcomings of the OpenGL method by using well 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.
- Model arbitrary changes in spacecraft configuration e.g. solar panel rotations.
- Employing a wide variety of material optical properties.

In the rest 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 discussed. The resulting OpenCL modeling methodology is described in section four with initial validation and the faster than real-time computational performance presented in sections five and six respectively.

## 2. 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 a single or set of ray reflections are computed using a recursive algorithm for each individual pixel. The recursive algorithm tests for a ray intersection in the scene , and in the case it finds an intersection, the same intersection search algorithm is called 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, as the scene is traced the rays from certain pixels will terminate sooner than others. This leaves compute units inactive resulting in poor utilization of the computing resources GPU. Rather by producing an algorithm which is parallel by rays cast, after each iteration terminated rays my be discarded and the reflected rays repacked for a second iteration ensure all compute units marshaled are active.

An overview of the method presented in this paper is shown in Figure 1. To initialize the the process a spacecraft CAD model is provided as a triangulated mesh with model materials which define 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 allowing with the BVH traversal structure, the spacecraft mesh material definitions. The parallel ray-tracing 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 termination condition is either the ray leaves 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.

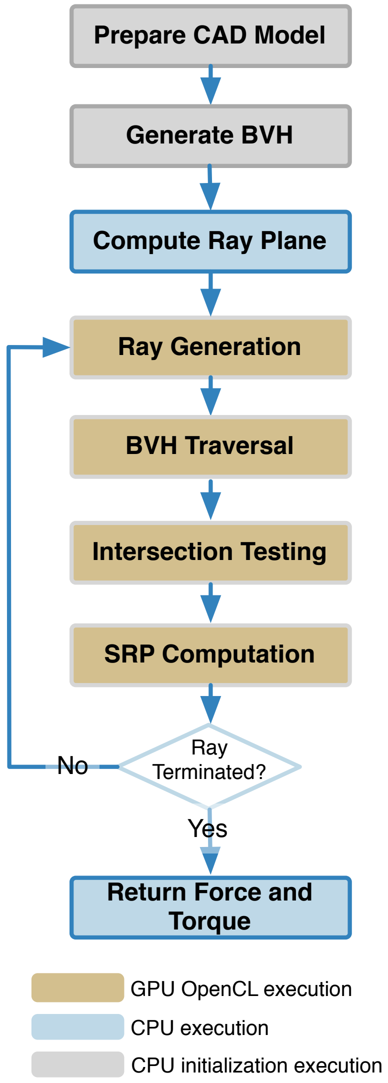

**Fig. 1:** Illustration of a set of five bounding boxes and a test ray. Intersections are recorded for the boxes with the dash outlines.

## 3. Algorithm Components

The presented approach employs a number of key techniques and 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 bounding volume hierarchy (BVH)
- bounding box intersection testing algorithm using clipping planes

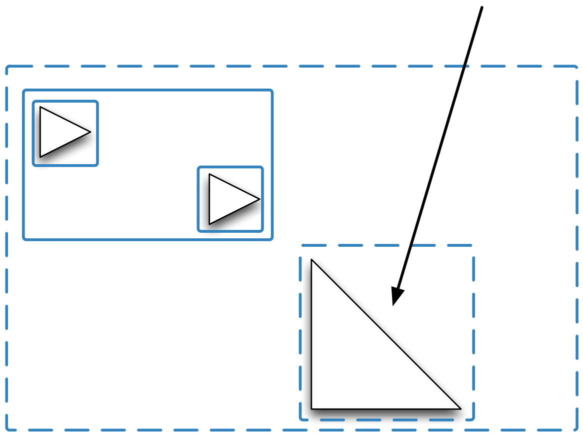

**Fig. 2:** Illustration of a set of five bounding boxes and a test ray. Intersections are recorded for the boxes with the dash outlines.

- the computational fast Möller-Trumbore ray and triangle facet intersection algorithm is used to test for intersections with a spacecraft facet and ray

and are presented in the subsequent subsections.

### *3.1 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.[16] This method employs a simple bounding volume hierarchy to efficiently reduce the ray intersection search space and therefore the required ray intersection computations performed. To build the bounding volume hierarchy 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. To begin, the list of bounding volumes is sorted along the first frame spacecraft body frame axis. The sorted list is 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 split, sequentially selecting the sort axis as the next axis in the body frame triad. This results in a bounding volume hierarchy that groups successive bounding volumes as containing facets spatially near to each other.

An efficient method of traversing the bounding volume hierarchy is a key aspect in the development of real-time SRP ray-tracing.[16] This implementation uses as the BVH traversal method a depth first search array as described by Smits.[16] An example BVH hierarchy comprising 6 nodes is shown in Figure 3 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

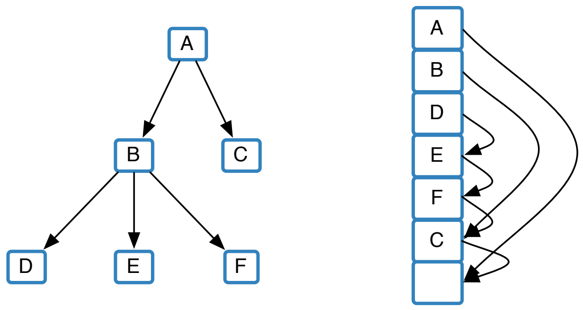

**Fig. 3:** Two BVH traversal structures. The left structure demonstrates a simple recursive BVH traversal. The right demonstrates the same BVH as shown on the left yet organized as a depth first search array with precomputed node skip pointers.

```
Data: idx is the index of the current node in the
      BVH depth first sorted traversal array
while idx is in range do
    node = fetch next node at idx;
    if intersectBbox then
        if node is leaf then
            intersectTriangle;
        else
            idx = idx + 1 (move to first child node);
            continue;
        end
    end
    idx = skip pointer at node (follow skip pointer
      to next node)
end
```

**Algorithm 1:** Algorithm to traverse the depth first sorted BVH array using skip pointers.

a termination condition and the search moves back up the tree and proceeds down the next search branch to test node C. For the array traversal structure, if bounding volume A is intersected, the next node to try is the next node in the array which is 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 array traversal algorithm is shown as pseudo code in Listing 1.

The depth first array search structure 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 structure is that the structure results in greater memory coherency for large meshes and therefore more efficient contiguous memory accesses on the GPU given its sequential nature.[16]

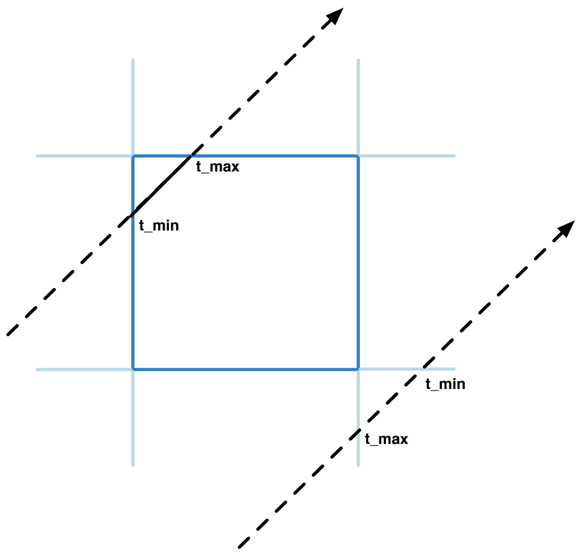

**Fig. 4:** Example results of the parallel plane bounding box intersection algorithm. For the top left ray intersection the algorithm returns t_max as greater than or equal to t_min. For the bottom right ray miss the algorithm returns t_max as less than t_min.

### *3.2 Bounding Volume Intersection*

Bounding volume intersection uses the algorithm originally presented by Kay and Kajiya.[17] Here 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 4, once the ray is clipped by each set of planes any remaining portion of ray inside the bounding volume indicates an intersection. The algorithm given in Listing 2, is particularly suited to implementation in the GPU environment because it does not require any testing of conditional code statements referred to as code branching. Modern floating-point instruction sets are capable of computing minimum and maximum operations without branches and this parallel plane algorithm results in a ray to bounding box intersection test with no code branches or divisions operations.

### *3.3 Triangle Facet Intersection*

The spacecraft model mesh is comprised of many thousands of triangular facets. To compute a triangle-ray intersection the Möller-Trumbore algorithm is used. This algorithm is a fast and memory efficient triangle-ray intersection algorithm making it ideal for use in the memory constrained GPU computation environment. The algorithm turns on the knowledge that the point of intersection of a line through a triangle in barycentric coordinates $(u, v)$ must adhere to easily boolean testable coordinate bounds. The bounds are defined by the barycentric coordinate system which requires $u \geqslant 0$, $v \geqslant 0$ and $u + v \leqslant 1$.[18]

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 converted to barycen-

```
Input  : The ray data structure containing an
         origin, direction and inverse direction
         vectors and the bounding box extents.
Output: True for intersection, False otherwise
tx1 ← (box.min.x - r.o.x)*r.dirinv.x;
tx2 ← (box.max.x - r.o.x)*r.dirinv.x;
t_min ← Min (tx1, tx2);
t_max ← Max (tx1, tx2);

ty1 ← (box.min.y - r.o.y)*r.dirinv.y;
ty2 ← (box.max.y - r.o.y)*r.dirinv.y;
t_min ← Max (t_min, Min (ty1, ty2));
t_max ← Min (t_max, Max (ty1, ty2));

return t_max ⩾ t_min;
```

**Algorithm 2:** Example of fast bounding box intersection computation for a box in a plane.

tric 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 equations 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 can be substituted with $\boldsymbol{E}_1 = \boldsymbol{V}_1 - \boldsymbol{V}_0$ 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.

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

Given in equation Eq. (6) is solution for $u, v$ and $t$ found using Cramer's rule, which yields the new matrix formulation. The solution for $u, v$ and $t$ will provide the values with which to test against the barycentric coordinate bounds conditions.

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

```
   Data: $\boldsymbol{o}$ is the origin of the ray
   Data: $\hat{\boldsymbol{d}}$ is the direction of the ray
   Data: $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ triangle vertices
   Result: Return triangle intersection
 1 $\boldsymbol{e}_1 = \boldsymbol{v}_2 - \boldsymbol{v}_1$;
 2 $\boldsymbol{e}_2 = \boldsymbol{v}_3 - \boldsymbol{v}_1$;
 3 $\boldsymbol{p} = \hat{\boldsymbol{d}} \times \boldsymbol{e}_2$;
 4 $det = \boldsymbol{e}_1 \cdot \boldsymbol{p}$;
 5 if $det < eps$ then
 6     return false;
 7 end
 8 $\boldsymbol{t} = \boldsymbol{o} - \boldsymbol{v}_1$;
 9 $u = (\boldsymbol{t} \cdot \boldsymbol{p})det^{-1}$;
10 if $u < 0$ or $u > 1$ then
11     return false;
12 end
13 $\boldsymbol{q} = \boldsymbol{t} \times \boldsymbol{e}_1$ ;
14 $v = (\hat{\boldsymbol{d}} \cdot \boldsymbol{q})det^{-1}$;
15 if $v < 0$ or $u + v > 1$ then
16     return false;
17 end
18 $t = (\boldsymbol{e}_2 \cdot \boldsymbol{q})det^{-1}$;
```

**Algorithm 3:** Möller-Trumbore algorithm used for fast and memory efficient triangle intersection testing.

A final solution is computed with slightly more efficiency by recognizing that each determinant $|\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}| = -(\boldsymbol{A} \times \boldsymbol{B}) \cdot \boldsymbol{C}$, is shown in Eq. (7). This can be 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 each and substituting as seen in the final matrix equation of Eq. (8).

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

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

As shown in Listing 3 the mapped ray and triangle can be tested against the barycentric coordinate frame's bounds. These checks occur at lines 10 and 15 in the algorithm's pseudo code representation. An additional check is performed early in the execution at line 5 to determine if the facet is facing the ray (facet normal vector opposite in direction to the ray unit direction vector). If the facet is not facing the ray the algorithm returns early as no intersection is possible.

## 4. Computing Solar Radiation Pressure

At each time update a new wave of ray vectors are generated. The current 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 sun frame is constructed and used to map the eight vertices, which define the extents of the spacecraft model bounding box, to the new temporary sun frame. The eight mapped vertices are used to compute a new $S$ frame axis aligned box of the mesh model. Given that the spacecraft sun distance is much greater than the spacecraft extents, it is assumed that the radiation emitted by the Sun can be modeled as a plane wave front. As shown in Figure 5, the sun facing side of the rotated bounding box is used as a finite plane perpendicular to $\hat{\boldsymbol{s}}_B$ from which the origins of all ray vectors shall be defined. The ray origin plane and the direction cosine matrix $[BS]$ are copied to the GPU memory space. The wave of rays are efficiently computed in parallel using a dedicated OpenCL ray generation kernel.

The ray plane is divided into unit squares determined by the resolution units chosen by the user. For example a 2m x 1m plane can be divided into 10mm sized squares giving a plane of 200 x 100 squares and a resolution of 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 square 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 10mm$^2$, is 2% and decreases to less than a percent for ray cross sections of less than 5mm.[13] Paying heed to Ziebart's study the maximum ray resolution used in this method is 10mm. The origin for a ray is taken as the corresponding center of a square 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 are 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.[13] Computing the reflected ray unit direction vector is given in Eq. 9, where $\hat{\boldsymbol{r}}_{i_{\text{ref}}}$ is the reflected unit direction

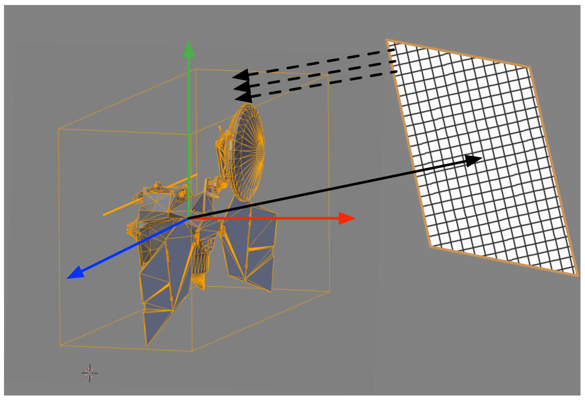
**Fig. 5:** The MRO mesh model surrounded by its bounding box. The red, green and blue vectors indicate the spacecraft body frame and the solid black vector indicates the spacecraft-sun vector, which is oriented perpendicular to the ray casting plane to the right. The dashed black vectors exemplify rays cast from the plane.

vector given the impinging ray direction $\hat{\boldsymbol{r}}_i$ and the $k^{th}$ triangle facet unit normal vector $\boldsymbol{n}_k$.

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

The irradiance of the reflected ray is used as given by Ziebart in Eq. 10 scaled per unit area by the combined scale factor $\mu v$, where $v$is the reflectivity and $\mu$ the specularity of the intersected triangular facet.

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

### *4.1 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 shown at Eq. (11), where $P(|\boldsymbol{r}_\odot|)$ is the solar radiation pressure scaled by the heliocentric distance to the spacecraft and $A_k$ is the cross sectional area of the ray.[19]

$$\begin{aligned}\boldsymbol{F}_{\odot_k} = -P(|\boldsymbol{r}_\odot|)A_k\cos(\theta_k)\Big\{(1-\rho_{s_k})\hat{\boldsymbol{s}} + \\ \left[\frac{2}{3}\rho_{d_k} + 2\rho_{s_k}\cos(\theta_k)\right]\hat{\boldsymbol{n}}_k\Big\}\end{aligned} \tag{11}$$

The sun angle of incidence $\theta_k$ as given by Eq. (12), is simply the dot product of the primitive surface normal $\hat{\boldsymbol{n}}_k$ and the sun unit vector $\hat{\boldsymbol{s}}$.

$$\hat{\boldsymbol{n}}_k = \frac{\boldsymbol{e}_1 \times \boldsymbol{e}_2}{\|\boldsymbol{e}_1 \times \boldsymbol{e}_2\|} \tag{12a}$$

$$\theta_k = \hat{\boldsymbol{n}}_k \cdot \hat{\boldsymbol{s}} \tag{12b}$$

The parameters $\rho_{s_k}$ and $\rho_{d_k}$ in Eq. (11) are respectively the specular and diffuse reflection coefficients of the

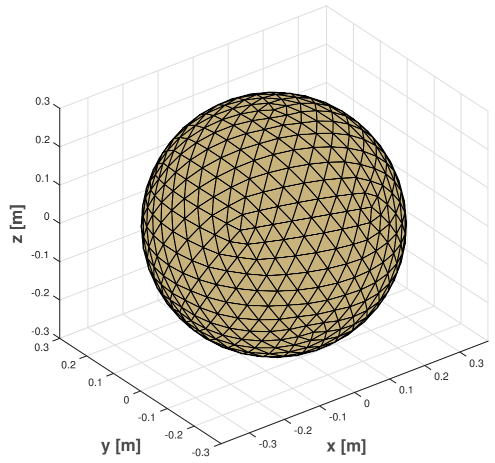

**Fig. 6:** The 1280 primitive, LAGEOS II size satellite mesh used in intial method validation.

material definition associated with the primitive. Following the force computation, the torque $\boldsymbol{L}_k$ contribution of a single intersection, as given in Eq. (13), is computed as the cross product of the vector defined from the body frame origin to the primitive's centroid $\boldsymbol{c}_k$ and the resolved ray force $\boldsymbol{F}_{\odot_k}$.

$$\boldsymbol{L}_k = \boldsymbol{c}_k \times \boldsymbol{F}_{\odot_k} \tag{13}$$

## 5. 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, given in Table 1, are used in both evaluations. A spherical model spacecraft is generated to replicate the LAGEOS II spacecraft parameters. Figure 6 illustrates the spherical spacecraft model being tested and Figure 7 shows the ray-tracing rendered version of the sphere colored according to facet normal vectors. The perturbative acceleration due to SRP as given by the cannonball model and the OpenCL model are given in Table 2. The OpenCL model yields a resultant torque of $2.3\times10^{-16}$ Nm. However, it is expected that a perfectly spherical object yields zero torque. The non-zero torque value returned by the OpenCL model is due to the faceted nature of the model not being perfectly spherical. It is observed that by increasing the number of vertices of the model to better approximate a sphere and increasing the resolution of the projected rays, the resulting torque value approaches machine precision. The agreement between the two simple evaluations provides initial confidence of the method's correctness.

Additional validation is carried out by comparing a more complex spacecraft mesh evaluation with existing post-launch SRP data. An example evaluation of the complex MRO spacecraft geometry is shown in Figure 8. In this evaluation a 14750 primitive model of the MRO spacecraft is processed at a heliocentric distance of 1AU where the sun vector is defined in the body frame is $\hat{\boldsymbol{s}}_B = [0, -1, 0]^T$. Approximate material optical properties are assigned to the spacecraft bus whereas the solar array emissivity and diffusivity, as quoted by You et. al. are set at 0.12 and 0.05 respectively.[20]

The post-launch SRP force magnitude value determined by the MRO navigation and the OpenCL ray-tracing force magntude value are given in Table 2. The small difference between these two results is a promising indication that the this method can offer a pre-launch SRP force accuracy close to that achieved after on-orbit small force calibration exercises. Additional visual validation is provided by an on screen rendering of the ray-

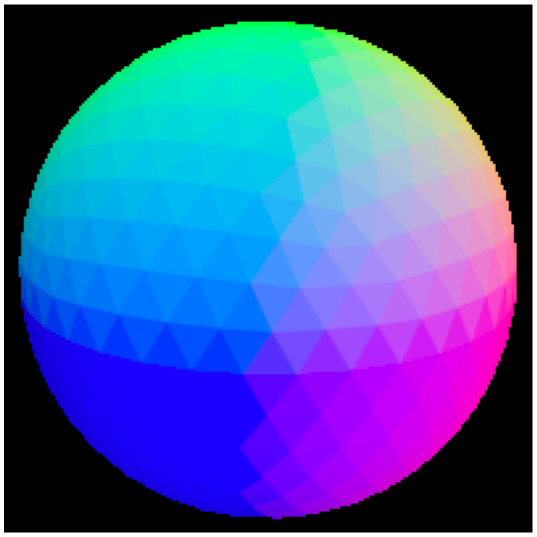

**Fig. 7:** A ray-traced evaluation of a 1280 primitive, LAGEOS II size satellite. Facet colors are purely cosmetic to allow for visual validation of accurate model rendering.

**Table 1:** LAGEOS II spacecraft parameters used for computation of SRP by cannonball model and OpenCL ray-tracing model.

| LAGEOS II Attribute | Value |
|---|---|
| mass | 405.38 [kg] |
| area | 0.2817 [m$^2$] |
| $\Phi$ (at 1 AU) | $1.38\times10^3$ [W/m$^2$] |
| $C_r$ | 1.12 |

**Table 2:** LAGEOS II spacecraft SRP induced acceleration computed by Cannonball and OpenCL models, where the OpenCL ray resolution is 10mm.

| Model | SRP Acceleration $a_\odot$ |
|---|---|
| Cannonball | $3.58\times10^{-9}$ [m/s$^2$] |
| Ray-Traced Cannonball | $3.60\times10^{-9}$ [m/s$^2$] |
| MRO Nav Team[20] | $\approx 9\times10^{-11}$ km/s$^2$ |
| Ray-Traced MRO | $8.552\times10^{-11}$ km/s$^2$ |

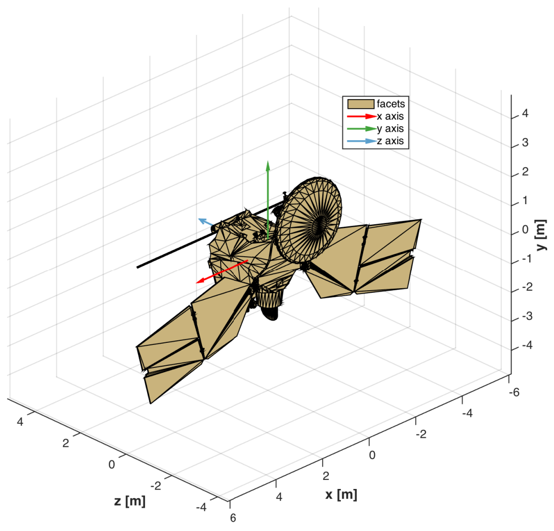

**Fig. 8:** The 14750 facet MRO mesh model used as the MRO ray-tracing test mesh.

traced spacecraft as shown in Figure 9. The rendering of the MRO spacecraft in Figure 9 demonstrates the correct geometric realization of the spacecraft mesh by the ray-tracing process. More promising is to acknowledge that this OpenCL method result does not yet include modeling of thermal re-radiation. The MRO navigation team found that thermal re-radiation was a significant contributor to the total observed small forces due to general radiation pressure.[20] The authors are confident that future near term model additions such as thermal re-radiation will allow for greater fidelity of the spacecraft physical processes.

## 6. Computational Performance

A primary goal of the OpenCL ray-tracing method is efficient computational performance resulting in faster than real-time evaluation. The execution time computed as the time point when ray generation begins to the time point when the CPU bound process receives the final force and torque values. Intuitively, the execution duration is critically dependent on the number of rays cast during model evaluation. An increased resolution increases the required number of rays to be traced and therefore the execution time. Additionally, there is a fixed base time 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 speaking, the latency is lower if the GPU is on chip as opposed to the GPU being a separate hardware device for which data transfer is over a common transfer bridge such as PCI Express.[21]

To demonstration the dependency of computation speed on resolution both the cannonball and MRO models were evaluated for the ray resolutions of 1mm, 5mm and 10mm. The execution time is taken as an average over 30 seconds whereby the computer under load is configured similarly for each benchmark. 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 Intel HD Graphics 630 1536 MB. The result of the evaluations are given in Table 3 for the cannonball model and Table 4 for the MRO model .

**Table 3:** Execution time for various ray resolutions of the cannonball spacecraft model.

| Ray Plane (w × h) | Ray Resolution | Execution Time [ms] |
|---|---|---|
| 60 × 60 | 10mm | 2 |
| 120 × 120 | 5mm | 8 |
| 600 × 600 | 1mm | 35 |

**Table 4:** Execution time for various ray resolutions of the cannonball and MRO spacecraft models.

| Ray Plane (w × h) | Ray Resolution | Execution Time [ms] |
|---|---|---|
| 1200 × 600 | 10mm | 39 |
| 2400 × 1200 | 5mm | 52 |
| 12000 × 6000 | 1mm | NA |

It is clear that tracing an increased number of rays demands further computational time. It is noteworthy that the test for 1mm resolution on the MRO spacecraft produced too great a volume of data to copy and hold in the GPU memory concurrently. This is identified as a limitation of the memory size for the utilized Intel HD Graphics 630 1536 MB and is being remedied by employing a GPU with a greater on-board memory size.

In contrast to a previously presented approach, which used the OpenGL graphics pipeline to compute SRP forces and torques, the resolution of this method is not bound by the complexity of the spacecraft mesh.[?] The previous OpenGL method computed per facet force and torque values and summed each contribution to yield a final value. To increase the resolution of the OpenGL method a model mesh with an increased number of facets is required and therefore an increased time to compute the force contribution of each facet. With the OpenCL ray-tracing method the computation time is coupled to the number of rays that are traced through the scene. This allows for increasingly complex spacecraft meshes to be evaluated for similar computation times.

## 7. 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, self-reflection and arbitrary spacecraft articulation are implicitly captured by a ray-tracing method resulting in a faster than real time modeling evaluation. This method presents mission analysts with a tool that requires min-

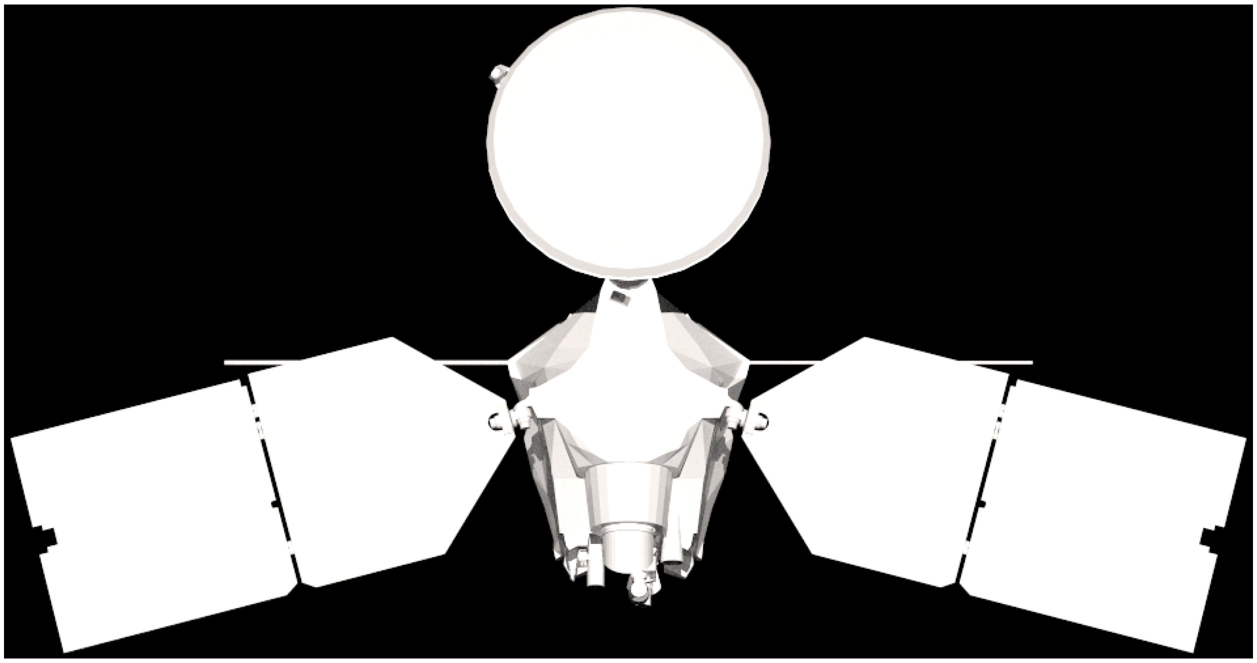

**Fig. 9:** A visual validation of the MRO model mesh rendering. Light and dark areas exemplify the various shadowing effects of the method.

imal 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 interactions are to be modeled.

## REFERENCES

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

[2] Henry F. Fliegel and Thomas E. Gallini. Solar force modeling of block IIR Global Positioning System satellites. *Journal of Spacecraft and Rockets*, 33(6):863–866, 1996.

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

[4] J.D. D Owens, M. Houston, D. Luebke, S. Green, J.E. E Stone, and J.C. C Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5), 2008.

[5] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. *State of the Art Reports, EUROGRAPHICS*, pages 21–42, 2001.

[6] M Ziebart, S Adhya, a Sibthorpe, S Edwards, and P Cross. Combined radiation pressure and thermal modelling of complex satellites: Algorithms and on-orbit tests. *Advances in Space Research*, 36(3):424–430, 2005.

[7] Daniel J. O'Shaughnessy, James V. McAdams, Peter D Bedini, Andrew B Calloway, Kenneth E Williams, and Brian R Page. Messenger's use of solar sailing for cost and risk reduction. *Acta Astronautica*, 93:483–489, January 2014.

[8] Daniel J. O'Shaughnessy, James V. McAdams, Kenneth E Williams, and Brian R Page. Fire sail: Messenger's use of solar radiation pressure for accurate mercury flybys. pages 1–16, 2011.

[9] T. A. Springer, G. Beutler, and M. Rothacher. A new solar radiation pressure model for gps satellites. *GPS Solutions*, 2(3):50–62, 1999.

[10] Y E. Bar-Sever. New and improved solar radiation models for gps satellites based on flight data final report. Technical report, Jet Propulsion Laboratory, 1997.

[11] David M. Lucchesi. Reassessment of the error modelling of non-gravitational perturbations on LAGEOS II and their impact in the Lense–Thirring derivation—Part II. *Planetary and Space Science*, 50(10-11):1067–1100, 2002.

[12] Jay W. McMahon and Daniel J. Scheeres. New solar radiation pressure force model for navigation. *Journal of Guidance, Control, and Dynamics*, 2010.

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

[14] S Tanygin and G. M Beatty. Gpu-accelerated computation of srp forces with graphical encoding of surface normals. In *Astrodynamics 2015 : proceedings of the AAS/AIAA Astrodynamics Specialist Conference held August 9-13, 2015, Vail, Colorado, U.S.A*, number AUGUST. AAS/AIAA Astrodynamics Specialist Conference, At Vail, CO, 2015.

[15] Patrick W. Kenneally and Hanspeter Schaub. High geometric fidelity modeling of solar radiation pressure using graphics processing unit. Winter Conference. American Astronomical Society, AAS, 2016.

[16] Brian Smits. Efficiency Issues for Ray Tracing. *Journal of Graphics Tools*, 3(2):1–14, 1999.

[17] T L Kay and J T Kajiya. Ray Tracing Complex Scenes. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):169–278, 1986.

[18] Tomas Moller and Ben Trumbore. Fast , Minimum Storage Ray / Triangle Intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[19] Bong Wie. *Space Vehicle Dynamics and Control*. American Institute of Aeronautics and Astronautics, Reston, VA, 2008.

[20] Tung-Han You, Eric Graat, Allen Halsell, Dolan Highsmith, Stacia Long, Ram Bhat, Stuart Demcak, Earl Higa, Neil Mottinger, and Moriba Jah. Mars reconnaissance orbiter interplanetary cruise navigation. In *20th International Symposium on Space Flight Dynamics*, 2007.

[21] Chris Gregg and Kim M. Hazelwood. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *ISPASS 2011 - IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144, 04 2011.