AAS 17-127

MODELING SOLAR RADIATION PRESSURE WITH SELF-SHADOWING USING GRAPHICS PROCESSING UNIT

Patrick W. Kenneally* and Hanspeter Schaub†

This paper presents a method for the fast computation of spacecraft force and torque due to solar radiation pressure (SRP). The method uses the highly parallel execution capabilities of commodity Graphics Processing Unit (GPU) and the Open Graphics Library (OpenGL) vector graphics software library and Open Compute Language (OpenCL) to render a Computer Aided Design (CAD) generated spacecraft model on the GPU. A custom-developed OpenGL render pipeline computes the per model facet SRP forces and torques which and summed on the GPU before the resultant spacecraft force and torque vectors are copied back to the CPU bound process. The evaluation accommodates spacecraft self shadowing and is capable of accounting for arbitrary spacecraft articulation. Material properties are encoded with the model to provide realistic specular, diffuse and absorption surface light interactions.

INTRODUCTION

Effective orbit determination, maneuver and mission design, and numerical mission 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. Given this 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 have driven the pursuit to create more vivid and realistic artificial worlds. This pursuit has resulted in highly optimized vector graphics software and GPU computer hardware capable of carrying out many thousands of floating point operations in parallel.[4] While these artificial worlds are visually persuasive, their implementation of electromagnetic radiation physics is understandably inaccurate. However, it is the parallel hardware and efficient vector graphics software implementations which may be used to simplify the steps of the SRP computation with great effect.

The ability to model and compute, at orders of magnitude 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.[5] However, there are many instances in which the articulation motion and the spacecraft state are dependent on the myriad

*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

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.

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 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 manuevers. 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 ΔV.[6] Additionally, the MESSENGER team was able to reduce fuel and related structural accommodations in the spacecraft design to reduce overall mission cost.[7]

A survey of the current landscape of SRP research reveals a variety of approaches. Ziebart characterizes SRP evaluation as a two step process.[8] The first step is the development of an analytic model; the second is to compute a result from the analytic model.

The most basic model with regard to the analytic development is referred to as the cannonball model. The cannonball analytic model is 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 LAEGOS missions and continues to prove useful for initial mission analysis.[9]

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

Increased modeling accuracy 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 characteristics are kept distinct for each surface and set based on known spacecraft material properties.[10] However, common among shape approximation methods is that much of the modeling uncertainty occurs in an estimation process within the second step of the SRP evaluation. It is the model's computation, the second step of the process, in which much work is being done. Notably Zeibart, details an evaluation procedure which requires precomputation of the body forces over all 4π steradian attitude possibilities.[8] Ziebart's approach is also capable of modeling self-shadowing by using ray-tracing techniques and spacecraft re-radiation via reduced spacecraft thermal model. McMahon and Scheeres extend such a model by aggregating the resultant SRP forces into a set of Fourier coefficients of a Fourier expansion.[10] 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 estimated during the orbit determination effort.

More recently methods which 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] Inspiring some of the methods presented in this paper Tichey et al. use OpenGL, a vector graphics GPU software interface common in video games, 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.[12]

This paper builds on previous work which utilized OpenGL to resolve the SRP force and torque of a CAD based spacecraft model. In the previous work spacecraft material properties are encoded with the model to provide realistic specular, diffuse and absorptive surface light interactions. This paper extends this previous work by describing two additions to the modeling procedure. The first addition is to employ a facility built into OpenGL called depth buffering.[13] The depth buffer facility in OpenGL computes the distance from the camera to the nearest facet, such that pixels with greater depth values are overwritten by pixels with smaller values in the final rendered scene. In the SRP modeling context the depth buffer is used to determine and exclude from the force and torque evaluation those areas of the spacecraft model which are within the spacecraft's own shadow.

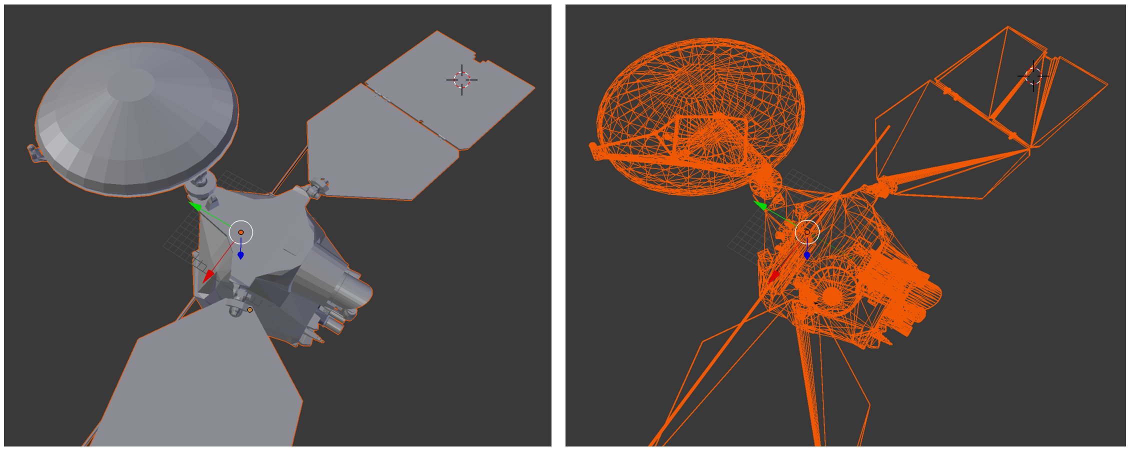
In previous work the array of per facet force and torque vectors were copied from GPU memory back to CPU bound memory where the vectors are summed in a serial order. The second addition described in this paper is an implementation which employs OpenCL in addition to OpenGL in a shared computing context to perform the final summation of the force and torque contribution of each model facet on the GPU. Utilizing a shared OpenGL and OpenCL computing context enables a parallel summation and reduces the latency introduced by copying all force and torque contributions from memory in the GPU process back to the CPU bound process.

SRP EVALUATION PIPELINE

The proposed method utilizes a computer vector graphics rendering pipeline to compute the SRP forces over the spacecraft surface. Computer vector graphics systems process sets of points defined in three dimensions. These points, referred to as vertices, are combined into groups of three to define a triangle shape primitive. A computer aided design (CAD) model is often approximated as a system of many thousands of polygon primitives, combined into a data structure referred to as a mesh. Where a flat plate surface such as a solar panel may be modeled by a single rectangle resolved as two triangle primitives, a curved surface may be approximated by many smaller triangles. Such an approximation is shown in Figures 1(a) and 1(b). In these figures it is evident that the Mars Reconnaissance Orbiter (MRO) high gain antenna shown Figure 1(a) is approximated by many thousands of primitives while the solar panels are approximated in Figure 1(b) by ten.

The entire spacecraft may be defined by one or more sub-meshes. Each mesh is assigned a material definition which describes the mesh's diffuse, specular and absorptive optical properties. Material definitions are typically and most conveniently defined within a CAD software tool, however, they also can be declared manually and assigned to each mesh in the custom-render pipleline presented in this paper. The density of primitives used to approximate the spacecraft's structure determines the geometric fidelity. Further, where relevant, an increased number of mesh material definitions will provide a more accurate SRP evaluation.

This method employs the Open Graphics Library (OpenGL) and Open Computing Language (OpenCL) to facilitate processing of the spacecraft model primitives and evaluation of the SRP forces and torques. OpenGL is a language independent application programming interface (API)



(a) Solid MRO CAD model (b) MRO triangle primitives

Figure 1. Example of Computer Aided Design (CAD) spacecraft model.

for rendering computer vector graphics.[14] The API provides tools to send, retrieve and process data on an OpenGL compliant GPU. OpenCL is a cross-platform standard for parallel programming across a range of processors from multi-core CPU devices to GPU devices.[15] OpenCL facilitates both the read and write of data on processing unit(s) and the submission of code for execution on the processing unit.

The OpenGL rendering process, referred to as the pipeline, is divided into stages where each stage in turn operates on the result of the previous stage and accepts data types defining either vertex or primitive shapes. Each stage is defined by a mini-program called a shader as shown in both pipeline Figures 3(a) and 3(b). For each pipeline stage many thousands of parallel instances of a shader program are executed, each performing processing on their specific input data type.[13] It is this highly parallel per primitive operation for which GPU devices have been specifically designed. Additionally, the pipeline may be executed multiple times before producing the final scene rendered image. Each successive pipeline execution may consist of a different set of shader programs to generate a different set of render output.

In the OpenGL modeling approach the render pipeline is executed twice to compute the SRP force and torque. Each execution uses a different set of shader programs. This approach is often referred to as deferred rendering.[13] In deferred rendering a first render pass (execution of the pipeline) generates the effects of fundamental render products such as multiple light sources and shadow generation. The second pass then uses the data produced in the first pass to compute and composite the final scene. The deferred rendering approach used to model SRP force and torque leverages an existing vector graphics technique called shadow mapping. As shown in Figure 3(a), shadow mapping employs a minimal pipeline consists of a vertex and fragment shader stage to perform a first render pass. The first render pass computes a two-dimensional depth map of the model facets projected into a plane perpendicular to the light source to model direction vector. As demonstrated in Figure 2 each entry in the depth map records the distance from the plane to the nearest model facet. In a graphics context the second render pass then uses the saved depth map to generate regions of dark and light to define the model's shadow in the rendered scene.[13] The technique is used similarly in the SRP computation where the light source is the sun and rather than using the depth map to rendering an on screen scene, the depth map is used in the second render pass to identify and exclude from further computation spacecraft model facets which are shadowed by other spacecraft model facets positioned closer to the sun.

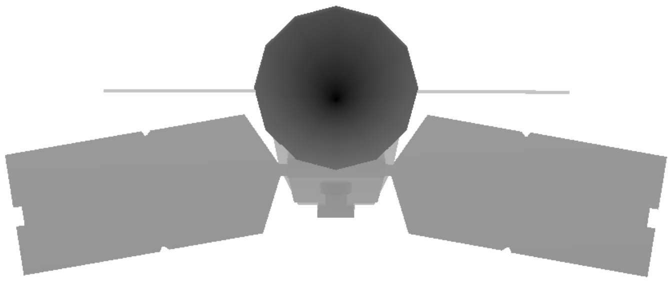


Figure 2. The generated depth map texture of the MRO spacecraft viewed as an image. The darker the region the region closer that region is to the projection plane.

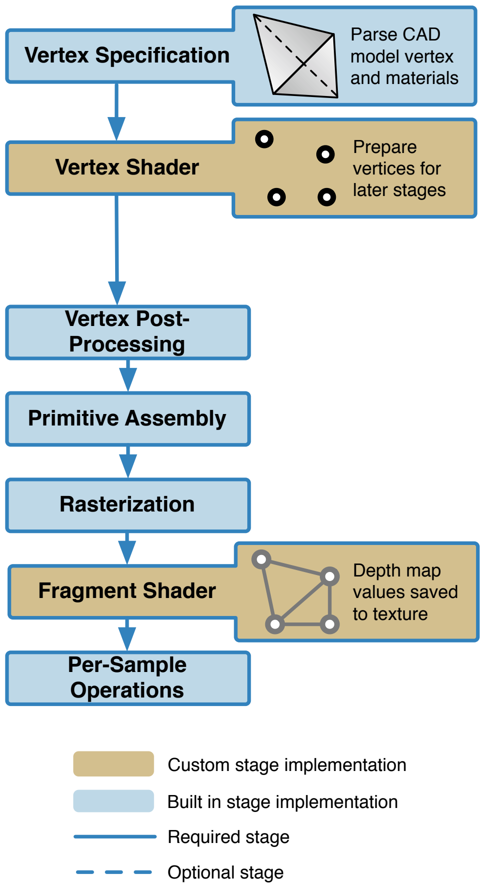
The depth map is computed in the first render pass and saved as a texture object. A texture object is an OpenGL image data format (or matrix of data) for which each pixel element is encoded as the distance from a plane to the nearest model facet. The final depth map texture contains the depth values of the 2D projection of the spacecraft model into the defined plane. The production of the depth values is a highly optimized built in OpenGL feature. In this application the first execution of the OpenGL pipeline is instructed to perform only operations needed to produce the built in depth values and to have these values output to the texture depth map. To facilitate the generation of the depth map the spacecraft vertices must be mapped to a projection plane. The mapping requires three sequential orthogonal reference frame mappings as demonstrated in Figure 4. The three mappings are between each of the following reference frames.

1. The orthogonal model reference frame (M) is the frame in which vertices are defined.
2. The orthogonal inertial reference frame (N) is the frame in which the location of the sun and spacecraft are defined.
3. The orthogonal view frame (V) is a reference frame defined such that its $\hat{\boldsymbol{z}}$ axis points along the sun-spacecraft unit direction vector.

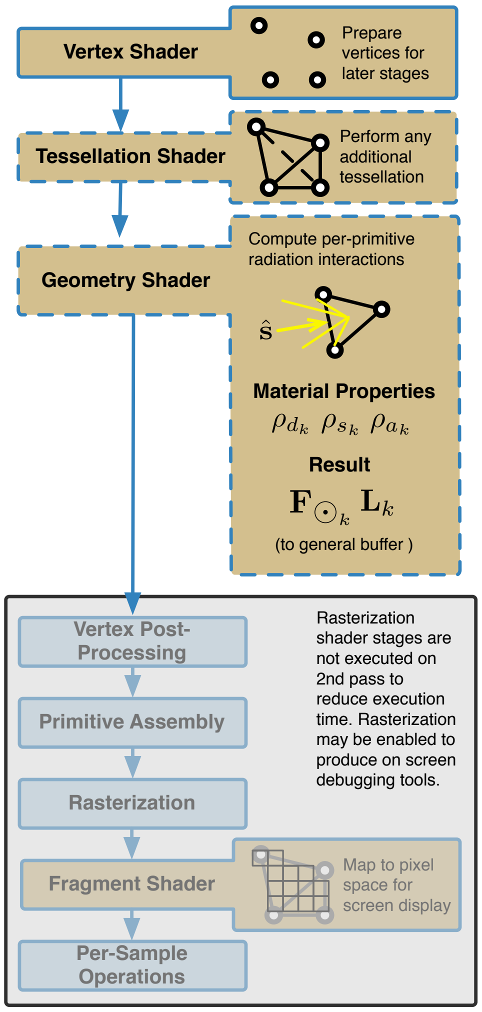
The mapping from the model frame M to the inertial frame N is given by the direction cosine matrix (DCM) denoted $[NM]$. The second mappng from the inertial frame to the sun-spacecraft view frame is $[VN]$ as shown in the result given by Eq. (2). The final orthogonal projection into the plane P is given simply in Eq. (3). The built in OpenGL computation of the depth map uses this final projected geometry which to produce the depth map texture within the first pipeline execution. An example of the generated depth map texture as an image in Figure 2.

$$^{\mathcal{V}}\boldsymbol{v} = [VN][NM]^{\mathcal{M}}\boldsymbol{v} \tag{2}$$

$$\begin{pmatrix} ^{\mathcal{P}}\boldsymbol{v}_x \\ ^{\mathcal{P}}\boldsymbol{v}_y \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} {}^{\mathcal{V}}\boldsymbol{v} \tag{3}$$

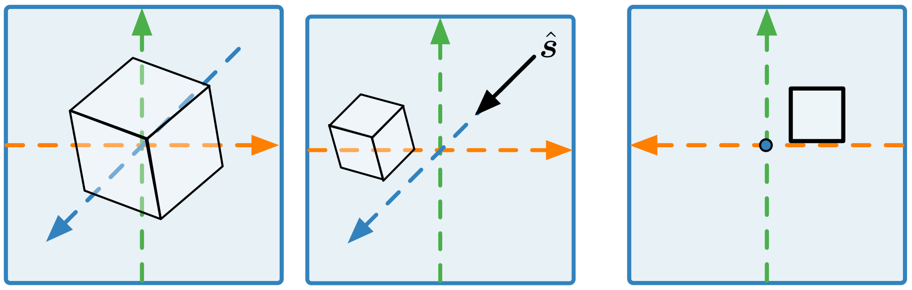

Vertex Specification
Parse CAD model vertex and materials
Vertex Shader
Prepare vertices for later stages
Vertex Post-Processing
Primitive Assembly
Rasterization
Fragment Shader
Depth map values saved to texture
Per-Sample Operations
Custom stage implementation
Built in stage implementation
Required stage
Optional stage

(a) Custom pipeline for 1st pass.


Vertex Shader
Prepare vertices for later stages
Tessellation Shader
Perform any additional tessellation
Geometry Shader
Compute per-primitive radiation interactions
$\hat{s}$
Material Properties
ρ_{d_k} ρ_{s_k} ρ_{a_k}
Result
$\mathbf{F}_{\odot_k}$ $\mathbf{L}_k$
(to general buffer)
Vertex Post-Processing
Primitive Assembly
Rasterization
Fragment Shader
Map to pixel space for screen display
Per-Sample Operations
Rasterization shader stages are not executed on 2nd pass to reduce execution time. Rasterization may be enabled to produce on screen debugging tools.

(b) Custom pipeline for the 2nd pass.

Figure 3. The minimal custom OpenGL render pipeline for the 1st pass in the deferred render generates the texture depth map in the fragment shader stage. The second pass produces the per facet force and torque values to be summed by the OpenCL program. Gold fill stages are custom shader stage implementations. Blue fill stages are OpenGL built-in shader stages. A Solid border stage indicates a required pipeline stage, while a broken border indicates optional stages.



(a) Model frame (b) Inertial frame with sun-spacecraft direction vector (c) Sun-spacecraft aligned reference frame

Figure 4. Indicative reference frame transformations from model frame to the final sun-spacecraft aligned view frame.

The second pipeline execution employs a geometry shader stage which operates on entire primitives, and the generated depth map to compute the per facet SRP force and torque result. The second pipeline begins by projecting each facet onto the same depth mapping plane as defined in the first pipeline. The three, two dimensional vertices resulting from the projection are used as (x, y) lookup coordinates in the texture depth map. The depth map values for the facet's projected vertices are compared to the original depth values of the three dimensional vertices. If all three vertex depth values mach those contained in the depth map then the facet is sunlit and the facet's SRP force and torque contribution is computed in the remainder of the geometry shader. As indicated in Figure 3(b), the geometry shader receives the vertex data for a single facet in addition to values for solar flux $\Phi_{\odot}$ and the sun unit direction vector defined in the body frame $\hat{\boldsymbol{s}}_B$. The force for each k^{th} primitive, $\boldsymbol{F}_{\odot_k}$, is evaluated in the spacecraft body frame using the expression shown at Eq. (4), where $P(|\boldsymbol{r}_{\odot}|)$ is the solar radiation pressure scaled by the heliocentric distance to the spacecraft.[16]

$$\boldsymbol{F}_{\odot_k} = -P(|\boldsymbol{r}_{\odot}|)A_k \cos(\theta_k) \left\{ (1-\rho_{s_k})\hat{\boldsymbol{s}} + \left[\frac{2}{3}\rho_{d_k} + 2\rho_{s_k}\cos(\theta_k) \right] \hat{\boldsymbol{n}}_k \right\} \tag{4}$$

The primitive area A_k is computed in Eq. (5), where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are edges of the primitive defined by the vertices.

$$\boldsymbol{e}_1 = \boldsymbol{v}_2 - \boldsymbol{v}_1 \tag{5a}$$

$$\boldsymbol{e}_2 = \boldsymbol{v}_3 - \boldsymbol{v}_1 \tag{5b}$$

$$A_k = \frac{1}{2}\|\boldsymbol{e}_1 \times \boldsymbol{e}_2\| \tag{5c}$$

The sun angle of incidence θ_k as given by Eq. (6), is simply the dot product of the primitive surface normal $\hat{\boldsymbol{n}}_k$ and the sun unit vector $\hat{\boldsymbol{s}}$.

$$\hat{\boldsymbol{n}}_k = \frac{\boldsymbol{e}_1 \times \boldsymbol{e}_2}{\|\boldsymbol{e}_1 \times \boldsymbol{e}_2\|} \tag{6a}$$

$$\theta_k = \hat{\boldsymbol{n}}_k \cdot \hat{\boldsymbol{s}} \tag{6b}$$

The parameters ρ_{s_k} and ρ_{d_k} in Eq. (4) are respectively the specular and diffuse reflection coefficients of the material definition associated with the primitive. Additional radiation pressure sources such as albedo from planetary bodies may be accounted for through an additional contribution in the $P(|r_\odot|)$ term. Following the force computation, the torque $\boldsymbol{L}_k$ contribution of a single primitive, as given in Eq. (7), 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 per primitive force $\boldsymbol{F}_{\odot_k}$.

$$\boldsymbol{L}_k = \boldsymbol{c}_k \times \boldsymbol{F}_{\odot_k} \tag{7}$$

The GPU executes many thousands of geometry shader instances in parallel resulting in the force and torque contributions of many thousands of primitive being evaluated simultaneously. The Geometry shader outputs the force and torque computations for each primitive to a general buffer to be consumed by an OpenCL program. The simple role of the OpenCL program id to sum all force and torque vectors to produce the resultant spacecraft force and torque vector and pass the two resultant vectors back to the CPU bound process. The OpenCL program is initialized on the GPU using a shared OpenGL and OpenCL computing context. The shared computing context allows OpenCL to read data buffers defined for input and output of the OpenGL computing context. This allows a developer to implement in OpenCL arbitrary parallel computations on the GPU using the OpenGL data which is resident on the GPU. Using the already populated OpenGL data buffers saves 'wall clock' execution time by not incurring the latency produced by copying data from the CPU bound memory to GPU resident memory.

The OpenCL program, referred to as a kernel, performs a parallel reduction over the array of per facet force and torque results.[15] A parallel reduction algorithm operates to sum all vectors in a binary tree order. As demonstrated in Figure 5 the binary tree order is highly parallel and allows for many associative ($(ab)c = a(bc)$) operations to be carried out across the multitude of compute units on the GPU.[17] For example a commodity laptop GPU (Intel Iris 6100 1536 MB) possesses 64 compute units upon which may be executed 64 instance of the OpenCL kernel. This provides the opportunity for 4096 parallel summations to reduce 8192 force or torque vector entries to 4096 entries. The binary tree order of parallel reduction continues until only a single resultant SRP force and torque vector remains.

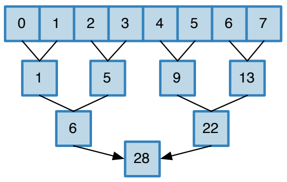


Figure 5. A trivial example of the binary tree ordering of parallel reduction.

The OpenGL method provides an ease of setup and configuration, which the authors believe, is not present in other techniques. The easy model import process allows the user to select the level of model vertex detail and materials in familiar CAD software tools. The only user required input to the modeling is the spacecraft CAD model. The remainder of the inputs such as solar flux, ephemeris and spacecraft dynamics states are supplied by numerical simulation software. This modeling approach is implemented as a modular tool which can be incorporated into any trajectory numerical simulation. At each integration step the model is provided with updated vehicle parameters. The two pass pipeline is executed returning the force and torque results for the spacecraft model.

MODEL VALIDATION

The initial validation is performed by comparing results from an analytic cannonball model and a sphere shaped spacecraft model within the OpenGL 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 evaluated using the OpenGL method. The perturbative acceleration due to SRP as given by the cannonball model and the OpenGL model are given in Table 2. The OpenGL model yields a resultant torque of 1.51×10^{-12} Nm. However, it is expected that a perfectly spherical object yields zero torque. The non-zero torque value returned by the OpenGL model is due to the faceted nature of the model not being perfectly spherical. It is observed that by increasing the number of vertices and therefore primitives in the model (better approximating a sphere), the resulting torque value approaches zero. The agreement between the two simple evaluations provides confidence of the methods correctness.

Table 1. LAGEOS II spacecraft parameters used for computation of SRP by cannonball model and OpenGL model.

LAGEOS II Attribute	Value
mass	405.38 [kg]
area	0.2817 [m^2]
Φ (at 1 AU)	1.38×10^3 [W/m^2]
C_r	1.12

Table 2. LAGEOS II spacecraft SRP induced acceleration computed by cannonball and OpenGL models.

Model	SRP Acceleration $a_\odot$
Cannonball	3.56×10^{-9} [m/s^2]
OpenGL	3.60×10^{-9} [m/s^2]

Additional validation is carried out using an equivalent implementation and data structures in MATLAB. The MATLAB validation tool has been used to validate single time step evaluations of more complex spacecraft model geometries. An example evaluation of a complex spacecraft geometry is shown in Figure 7. In this evaluation a 1424 primitive model of the Mars Reconnaissance Orbiter (MRO) is processed at a heliocentric distance of 1AU where the sun vector in 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 emmissivity and diffusivity, as quoted by You et. al. are set at 0.12 and 0.05 respectively. The SRP force value as determined by the MRO navigation team post launch of the spacecraft is $a_\odot \approx 9 \times 10^{-11}$ km/s^2.[18] The OpenGL method computed SRP acceleration is $a_\odot = 8.306 \times 10^{-11}$ km/s^2. The small difference between these two results is a promising indication that the OpenGL modeling method can offer a pre-launch SRP force accuracy close to that achieved after on orbit small force calibration exercises. More promising is to acknowledge that this OpenGL method result does not yet include modeling of thermal re-radiation and secondary photon impacts. The MRO navigation team found that thermal re-radiation in particular was a significant contributor to the total observed small forces due to radiation pressure.[18] The authors are confident that future

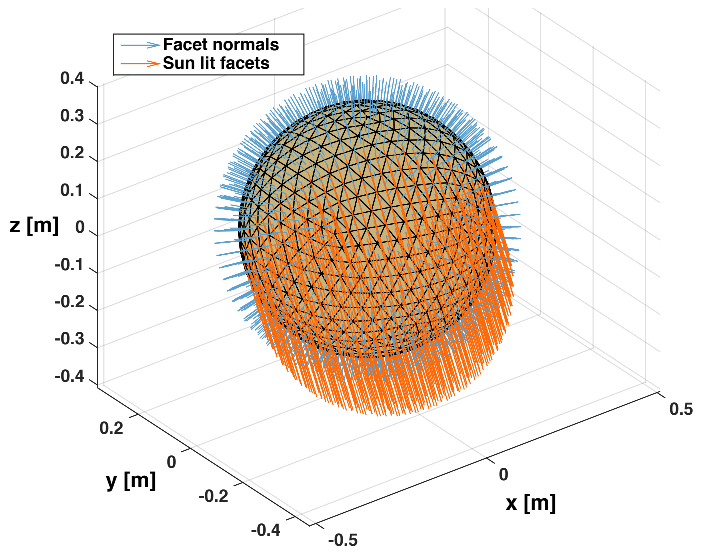


Figure 6. A visualized single evaluation of a 1280 primitive, LAGEOS size satellite. Orange vectors indicate $\hat{s}_B$ incident on primitives. Blue vectors indicate primitive face unit normal vector $\hat{\boldsymbol{n}}_B$

near term model process additions such as thermal re-radiation and secondary photon impacts will allow for a fidelity of the spacecraft physical processes.

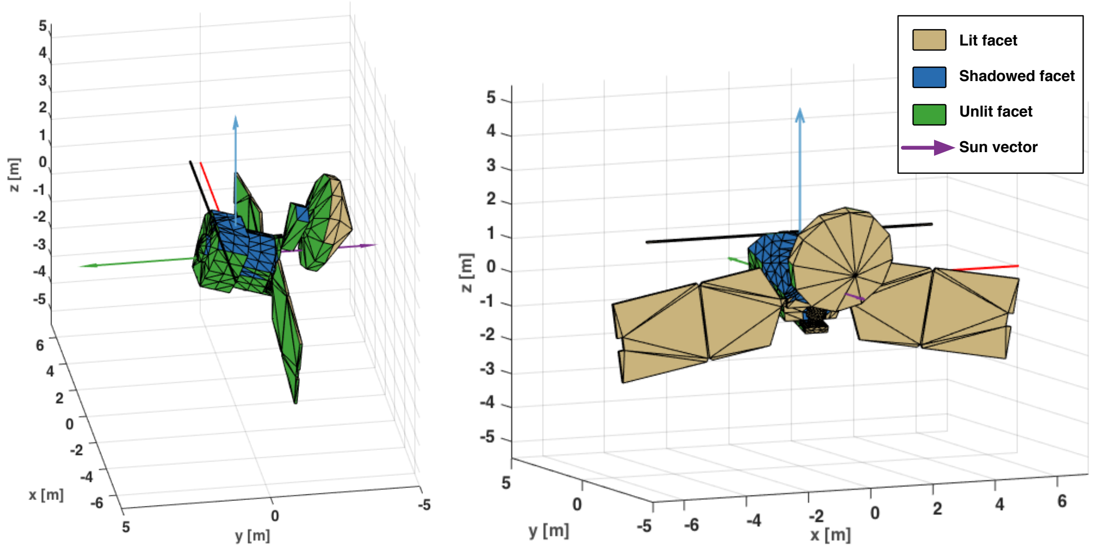
COMPUTATIONAL PERFORMANCE

Computational performance is a goal of the OpenGL modeling method. To provide a crude demonstration of the increase in evaluation speed afforded by the OpenGL method a single evaluation of the MRO model shown in Figure 6 is carried out using the MATLAB verification tool and the OpenGL method. This comparison is termed 'crude' as the same serial evaluation implemented in MATLAB would undoubtedly execute in less time if implemented using a non-interpreted programming language such as C++. However, in a C++ implementation a difference in execution time would remain making the crude demonstration instructive. The result of the two evaluations are given in Table 3.

Table 3. Execution time for single evaluation of MRO model using serial MATLAB verification tool and the OpenGL method. (OpenGL 4.1 on 2015 MacBook Pro 3.1 GHz Intel Core i7, 8GB Ram, Intel Iris 6100 1536 MB).

Model Implementation	Execution Time [sec] 1424 Primitives
MATLAB	3.7
OpenGL Method	0.002

The shared OpenGL and OpenCL computing context demonstrates a clear reduction in computation time. At low model facet counts the parallel reduction algorithm provides approximately 2 times reduction in execution time for the summation where the time savings increase with model facet count. A comparison of the new parallel reduction with the formerly used serial summation is shown in Figure 8. It is notable that the characterization presented in Figure 8 is carried out on a consumer grade GPU. More advanced dedicated GPU hardware will understandably demonstrate



(a) MRO unlit and shadowed facets (b) MRO sunlit facets

Figure 7. A single time step evaluation of the MRO spacecraft showing sunlit facets in gold, shadowed facets in blue and facets with a normal vector directed away from the sun direction in green.

an even greater reduction in computation time due to dedicated GPU hardware's increased number of compute units.

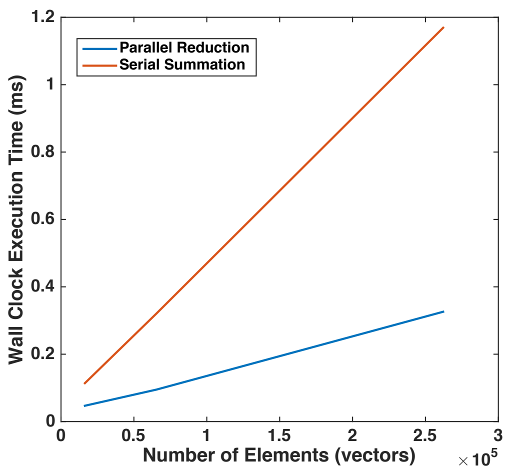


Figure 8. As the model facet count increases so to do the computational savings of the parallel reduction implementation.

ONGOING MODEL DEVELOPMENT

In the SRP analysis of particularly complex spacecraft structures, secondary photon impacts cause significant variation of the final force and torque results.[5] Further work is being undertaken to implement the ability for the OpenGL method to compute secondary photon impacts. Additionally the implementation of arbitrary spacecraft articulation is underway and once again employs well worn techniques from the 3D animation fields.

CONCLUSION

The ability to resolve the SRP forces on a dynamically articulated spacecraft composed of heterogeneous surface materials presents opportunities in the areas of attitude maneuver design and spacecraft control. This paper demonstrates how complex SRP forces can be resolved more accurately and at high speed using a GPU focused methodology. The greater geometric fidelity provided by a vertex based model allows easy initialization and subsequent evaluation of spacecraft dynamics given time varying spacecraft parameters.

ACKNOWLEDGMENT

The authors would like to thank Dr. Jay McMahon of the University of Colorado at Boulder for his guidance on the current state of SRP modeling and evaluation methods.

REFERENCES

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

[2] 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.

[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," tech. rep., 1992.

[4] 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.

[5] 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*, Vol. 36, No. 3, 2005, pp. 424–430, 10.1016/j.asr.2005.01.014.

[6] 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.

[7] D. J. O'Shaughnessy, J. V. McAdams, K. E. Williams, and B. R. Page, "FIRE SAIL: MESSENGER'S USE OF SOLAR RADIATION PRESSURE FOR ACCURATE MERCURY FLYBYS," 2011, pp. 1–16.

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

[9] D. 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*, Vol. 50, No. 10-11, 2002, pp. 1067–1100, 10.1016/S0032-0633(02)00052-1.

[10] 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.

[11] AAS/AIAA Astrodynamics Specialist Conference, At Vail, CO, *GPU-Accelerated Computation of SRP Forces with Graphical Encoding of Surface Normals*, No. AUGUST, 2015.

[12] J. Tichy, A. Brown, M. Demoret, B. Schilling, and D. Raleigh, "Fast Finite Solar Radiation Pressure Model Integration Using OpenGL," 2014.

[13] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL Programming Guide: The Official Guide To Learning OpenGL, Version 4.3*. Upper Saddle River, NJ: Addison-Wesley, 2013.

[14] Khronos Group, *OpenGL Documentation*, 10 2015.

[15] OpenCL Working Group Khronos, *The OpenCL Specification Version: 2.2*, 06 ed., March 2016.

[16] B. Wie, *Space Vehicle Dynamics and Control*. Reston, VA: American Institute of Aeronautics and Astronautics, 2008.

[17] M. McCool, A. Robison, and J. Reinders, *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann, Morgan Kaufmann, 2012.

[18] T.-H. You, E. Graat, A. Halsell, D. Highsmith, S. Long, R. Bhat, S. Demcak, E. Higa, N. Mottinger, and M. Jah, "Mars Reconnaissance Orbiter Interplanetary Cruise Navigation," *20th International Symposium on Space Flight Dynamics*, 2007.