MIRI: Computer Graphics & Virtual Reality

Polytechnic University of Catalonia - BarcelonaTech

Facultat d'Informàtica de Barcelona

- October, 22th 2019 -

Master Thesis

Developing an efficient algorithm for computing Solar Radiation Pressure

Student: Leandro Iván Zardaín Rodríguez

Advisors: Dr. Anna Puig

Dr. Ariadna Farrés

Tutor: Dr. Isabel Navazo

Abstract

The aim of this Master Thesis is to develop an efficient way of computing the solar radiation pressure acceleration that acts on a spacecraft. This acceleration is produced by the impact of the photons emitted by the Sun onto a spacecraft 's surface. This collision generates an acceleration which affects its motion and has a relevant effect on a long-term propagation.

This proposal studies which models can be used to approximate the computation of this acceleration. Some of the models are more efficient and some others are more accurate. The objective was to find a model that is both efficient and accurate. In order to do so, some of the most accurate models were implemented using the GPU to parallelize part of the computation. The chosen model that fulfils both conditions is based on the Raytrace technique. It considers secondary rays (bounces) in order to obtain more precision when computing the acceleration.

The main contribution of this thesis is the creation of a project that allows the user to compute and compare between different approximations of the SRP acceleration, and decide which approximation should be used. It also includes the possibility of visualizing the spacecraft model and the accelerations for a better understanding of them.

Acknowledgments

I would like to express my sincere gratitude to my advisors Anna Puig and Ariadna Farrés for the continuous support of this Master Thesis.

My sincere thanks also goes to David Folta for letting me visit one of the installations of the NASA.

I would like to thank my family: my parents and to my brothers for supporting me when I was writing this thesis.

Contents

1 Introduction

1.1 Motivation

The decade of 2020 is going to be one of the most important ones in the recent history of the universe exploration. Especially, for the Solar system. It is true that, in general, the time of big missions is gone and now, the space agencies prefer to do other projects with less cost, such as sending satellites, using telescopes to study the evolution and the origin of the universe and other objects that are in the Solar System.

In addition, this is going to be a crucial decade for the programs and missions of the agencies such as NASA or ESA.

In this context, it is especially important how to measure or predict the trajectory of a spacecraft. Indeed, the trajectory that a spacecraft will follow is difficult to approximate because of the many complex forces that are involved. There are a lot of parameters to consider and many forces to model to obtain something close to its real motion. The main forces that will affect the trajectory are the gravity of different planets and moons (which are relevant when you are in a close proximity of these objects) have been studied for a long time ago. Nowadays, it is known which models approximate better these forces.

However, a small error in the approximation can mislead its motion and make it follow a different trajectory. Even so, there are other forces that are also important to consider because depending on the situation, they can influence significantly the spacecraft's trajectory. One of them is the Solar Radiation Pressure (SRP) force (see [17]). In fact, they are more interested to approximate the acceleration of this force.

The SRP acceleration is produced by the collision between the sunlight photons and the surface of the spacecraft. This is relatively small if it is compared to some other forces such as the gravitation force of the Earth, but it becomes important to model when a long-term propagation trajectory is considered.

In order to appreciate the effect of the produced acceleration in a spacecraft, the aerospace engineers are interested in analysing this acceleration, considering all the possible orientations of the vehicle in relation to the Sun. This will allow them to orientate, later, the vehicle during the trajectory. For example, they could use this acceleration and a chosen orientation to modify its trajectory and then, reducing the consumption of fuel.

The SRP acceleration can be modelled in the basis on light transport theory in a similar way that the computation of shading in rendering algorithms. Light transportation models tend to be intensive in computation and different GPU-based methods have been proposed in the literature [12]. This usually leads to a lower performance in rendering time. Additionally, visual analytic could be helpful in the aim of exploring the computed forces globally. In line with this, in this thesis we propose to apply rendering strategies goodness to the SRP acceleration problems.

1.2 Aims of the thesis

The main goal of this Master Thesis is two-folded. First, we will explore different approximations of the SRP acceleration in terms of accuracy and performance, using CPU and GPU based approaches. Second, we will adapt data visualizations to easily explore local and global data.

A common objective for all of the proposed techniques is developing robust and accurate algorithms which are able to deal with simple and complex spacecrafts. Following we will give a brief description of particular objectives for every subject.

For the setting and computation of the SRP acceleration for a specific orientation of the spacecraft in relation to the Sun, our objectives are:

- Initially, it is important to study which models can be used to approximate better the local computation.
- Then, how the models can be computed efficiently analysing CPU and GPU-based approaches.
- And finally, the design and development of a graphical interface to define the properties of each approach, and the main direction of the Sun in a usable way.

So, apart from giving the approximated computation, it is important how they can be shown to the user in order to improve their understanding of these accelerations. Then, in terms of visualization, our goals are:

- Visualization of global SRP accelerations. For the global computation, the SRP acceleration would be computed considering the vector of light $\vec{r}_s$ coming from different points in the Space. In particular, the acceleration is computed considering $\vec{r}_s$ as a point from a discrete representation of a sphere, defined

by two angles (azimuth, elevation) where $-180 \leq azimuth \leq 180$ and $-90 \leq elevation \leq 90$.

- Visualization of a particular subset of SRP accelerations. In this case, the acceleration would be computed for individual $\vec{r_s}$ directions on a 3D viewer with the representation of the Sun and the spacecraft.

In order to validate our approaches, we will validate our algorithms in terms of accuracy and performance. Moreover, to evaluate how the application is easy to use for expert users, we will perform a usability test to the target users or expert users (i.e. engineers who work for a space agency or an organization and who need to compute precisely the SRP accelerations, such as the NASA).

Finally, our project is oriented to be multiplatform, so, the code could be compiled in several operative systems.

1.3 Reading structure

The problem that this thesis wants to solve is described in the next section. In particular, it introduces how the computation of the SRP acceleration is being done nowadays and also which are the objectives that this project wants to fulfil.

Then, there is the related work which will compare this project with the literature of computing the SRP acceleration.

After that, our proposal is presented: which new models have been studied, which techniques were implemented and how.

The next section is the validation and results section. There, we explain which method produces better accuracy and performance and also, it will be explained the usability test that was performed to some expert users from the NASA.

Finally, there is the conclusion and the future work sections of this project.

2 SPR models

Before stating the problem to solve, it is important to understand how the approximation of the SRP acceleration is being done nowadays (see [4]). Essentially, there are three basic models that approximate this acceleration and two of them are the most common ones to perform this approximation: the Cannon-Ball and the N-Plate model. They are easy to compute and fast, although they are not very accurate (specially the Cannon-Ball model).

2.1 Overview

In the next sections, several models will be explained, and later on, which (SRP) methods were implemented considering those models. Each method tries to approximate the computation of the SRP acceleration in its own way and for most of them, they will require some common parameters, as it is shown in Fig.1

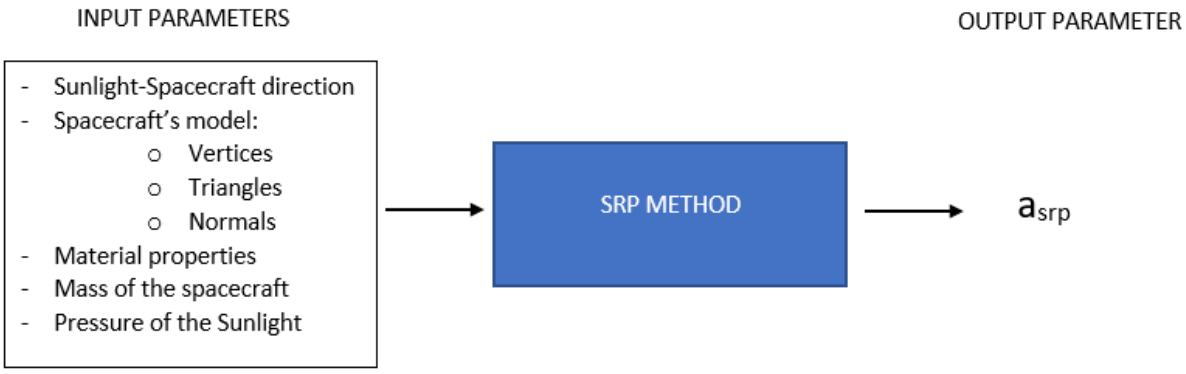

Figure 1: General representation of the parameters needed to compute a SRP method.

2.2 Current models

2.2.1 Cannon-Ball model

This is the simplest modeling of the SRP acceleration. The shape of the spacecraft is approximated by a sphere (see Fig 2). The acceleration is always constant and it does not depend on the orientation of the spacecraft respect to the sunlight direction.

$$\vec{a}_{srp} = \frac{P_{srp} A C_r}{m_{sat}} r_s \tag{5}$$

Where:

- $C_r = (1 - \rho_s)$ which is called the reflectivity coefficient.
- A is the area of the sphere.
- m_{sat} is the mass of the spacecraft.
- P_{srp} is the solar radiation pressure at a distance R from the Sun.

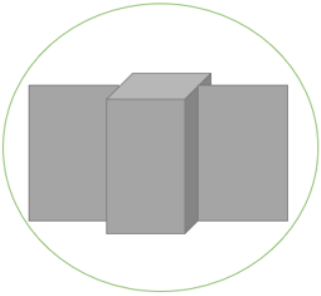

Figure 2: Representation of the spacecraft as a sphere.

2.2.2 Flat surface model

This model is the key to understand more complex models that are based on this one.

The SRP force ($\vec{F}_{srp}$) on a flat surface is a linear combination of the forces produced by the specular ($\vec{F}_s$), absorption ($\vec{F}_a$) and diffuse ($\vec{F}_d$) reflections (as it can be seen in Fig. 3)

$$\vec{F}_{srp} = \rho_a \vec{F}_a + \rho_s \vec{F}_s + \rho_d \vec{F}_d \tag{1}$$

Each one of these forces is defined as:

$$\begin{aligned} \vec{F}_a &= P_{srp} A < \vec{n}, \vec{r}_s > \vec{r}_s \\ \vec{F}_s &= 2 P_{srp} A < \vec{n}, \vec{r}_s >^2 \vec{n} \\ \vec{F}_d &= P_{srp} A < \vec{n}, \vec{r}_s > (\vec{r}_s + \frac{2}{3}\vec{n}) \end{aligned} \tag{2}$$

Where:

- ρ_a, ρ_s, ρ_d are the material reflectivity properties of the flat surface. $0 \leq \rho_a \leq 1$, $0 \leq \rho_s \leq 1$, $0 \leq \rho_d \leq 1$ and $\rho_a + \rho_s + \rho_d = 1$.
- A is the area of the flat surface.
- $\vec{n}$ is the normal vector to the surface.
- $\vec{r_s}$ is the Sun-to-spacecraft direction.
- P_{srp} is the solar radiation pressure at a distance R from the Sun.

Joining the expressions at (1) and (2), the SRP force can be defined as:

$$\vec{F}_{srp} = \rho_a(P_{srp}A < \vec{n}, \vec{r_s} > \vec{r_s}) + \rho_s(2P_{srp}A < \vec{n}, \vec{r_s} >^2 \vec{n}) + \rho_d(P_{srp}A < \vec{n}, \vec{r_s} > (\vec{r_s} + \frac{2}{3}\vec{n})) \tag{3}$$

$$= P_{srp}A < \vec{n}, \vec{r_s} > [(1 - \rho_s)\vec{r_s} + 2(\rho_s < \vec{n}, \vec{r_s} > + \frac{\rho_d}{3})\vec{n}]$$

The SRP acceleration, then, is computed by dividing the force by the mass of the spacecraft (m_{sat}):

$$\vec{a}_{srp} = \frac{F_{srp}}{m_{sat}} = \frac{P_{srp}A}{m_{sat}} < \vec{n}, \vec{r_s} > [(1 - \rho_s)\vec{r_s} + 2(\rho_s < \vec{n}, \vec{r_s} > + \frac{\rho_d}{3})\vec{n}] \tag{4}$$

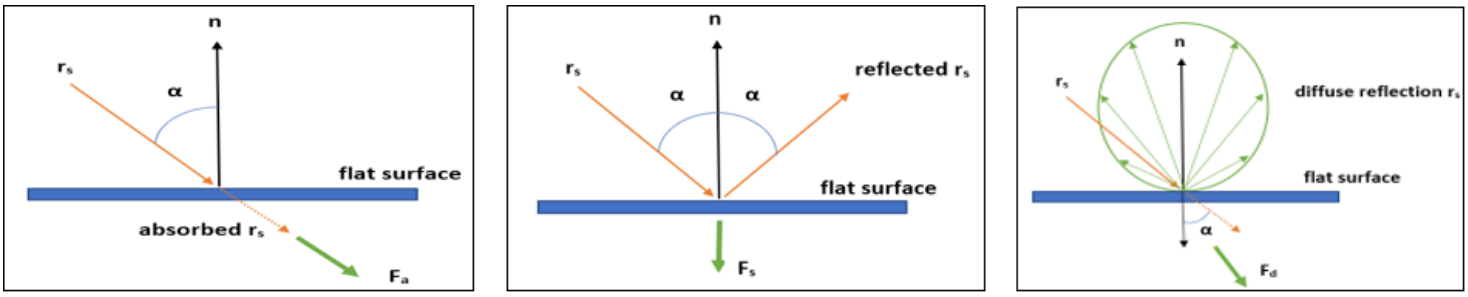

Figure 3: Absorption (left), specular (middle) and diffuse (right) reflections.

2.2.3 N-Plate model

The spacecraft's shape is approximated by a set of N flat plates (see Fig. 4).

Flat plate $i = \{\vec{n}_i, A_i, \rho_s^i, \rho_a^i, \rho_d^i\}$, $i \in \{1, \ldots, N\}$

Where:

- $\rho_a^i, \rho_s^i, \rho_d^i$ are the material reflectivity properties of the flat plate. $0 \leq \rho_a^i \leq 1$, $0 \leq \rho_s^i \leq 1$, $0 \leq \rho_d^i \leq 1$ and $\rho_a^i + \rho_s^i + \rho_d^i = 1$.
- A_i is the area of the surface of the flat plate.
- $\vec{n}_i$ is the normal vector to the surface of the flat plate.

The acceleration is modelled as:

$$\vec{a}_{srp} = \frac{P_{srp}}{m_{sat}} \sum_{i=1}^{N} (A_i \cos\theta_i [(1 - \rho_s^i)\vec{r}_s + 2(\rho_s^i \cos\theta_i + \frac{\rho_d^i}{3})\vec{n}] H(\theta_i)) \tag{6}$$

Where $\cos\theta_i =< \vec{n}_i, \vec{r}_s >$ and $H(\theta_i)$ is the visibility function. It is defined as $H : [-1, 1] \longrightarrow \{0, 1\}$. This function indicates whether the plate is used for the computation of the acceleration or not.

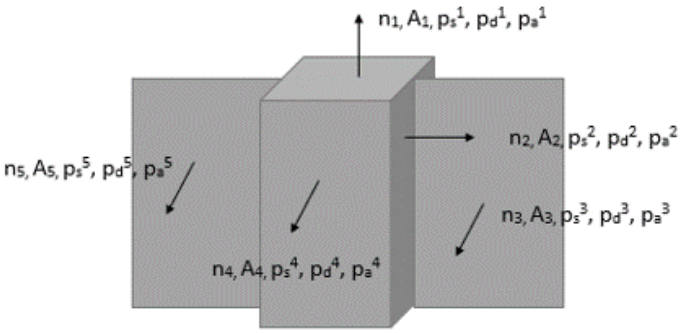

Figure 4: Representation of the spacecraft as a set of flat plates.

3 Related work

As mentioned before, there are two main lines that describe this project. One is centered on the computation of SRP accelerations (accuracy and performance). And the other one is about how they can be represented. For this reason, the literature related to this project was studied considering both aspects.

3.1 Computing SRP acceleration

When the sunlight photons impact on the spacecraft, a resultant acceleration is generated. This problem is a particular case of a more generic one that consists of computing the interaction of the light with a certain model that is described in the *rendering equation* [7].

As we presented in the previous chapter, many models have been developed to approximate the SRP acceleration, such as the ones mentioned in the previous section. However, they present some drawbacks.

Firstly, the main problem with the previous models is that they are too generic and simple in two aspects: (i) they approximate the shape of the spacecraft to a sphere or a set of plates and also, (ii) they do not consider multiple bounces of the light over the spacecraft.

Therefore, there is a need to search for new models (high fidelity models) that better represent this resultant acceleration while considering the performance, as well.

3.1.1 Raytrace model

Solving the rendering equation for any given spacecraft is the primary challenge in the SRP acceleration computation, that is a problem similar to perform realistic renderings. Among other approaches to solving the equation, such as finite element methods, several Monte Carlo approaches has been proposed (path tracing, photon mapping, and Metropolis light transport, among others).

Thus, we considered a Raytrace model to approximate the SRP acceleration, based on the combination of Whitted Raytrace [18] and Path Tracing [3] techniques. Then, we adapted the RayTracing technique as a model of the SRP acceleration, considering specular and diffuse reflections.

The main difference between these approaches is how they compute reflections and refractions (see [8]).

We considered the specular reflections as 'perfect' reflections. It means that the computation of the specular reflection is deterministic: for the same input (the incident ray and the surface normal), the procedural will always give the same reflected direction. See [18] to get more details of this computation.

While for the diffuse reflections, the Path Tracing technique was considered which uses the Monte Carlo integration for the evaluation of integrals (which is not deterministic). It casts only several random points from a hemisphere following the normal vector of a point of the surface. A sample of a diffuse reflection is the vector of the difference between a random point and the surface point. Then, the contribution for the diffuse part is the mean of the contribution obtained by each diffuse reflection sample, as it is explained here [8]. Adding more samples, the contribution will converge to the correct one.

This Raytrace model could be more accurate than the proposed ones, but also it could be the costliest in time and in memory. For this reason, we also propose a Raycast model that only casts primary rays and not the secondary ones, losing some accuracy in the final computation but improving the performance. Additionally, we will explore possible GPU optimizations to compute the SRP acceleration more efficiently.

3.1.2 GPU optimization

As we mentioned above, the Raytrace model is the slowest one in terms of computational time because of its high complexity. For this reason, it was important to find a way to optimize the computation and our approach uses the GPU parallelization mechanisms.

For the Raytrace model, it was considered to take the computation to the GPU, so it could be parallelized [2].

Then, the parameters needed for the computation of the SRP acceleration must be transfered to the GPU.

Apart from the Raytrace model, there are others that can improve the interaction of the light with an object such as the Ambient Occlusion technique [20]. It renders the scene twice. Firstly, considering the camera following the direction of the sunlight and then, considering the camera to where the observer is and using the information obtained in the previous rendering.

However, this technique does not add relevant information to the computation of SRP accelerations because the observer is considered to be watching in the direction of the sunlight vector. So, for example in this case, it would be applying the same rendering twice.

Another step to improve the performance of the Raytracing technique is to apply a voxelization over the spacecraft model. The implementation of the voxelization for our proposal is based on [14] and [13], which is computed in the CPU. However, there are some other options such as [16] that does the computation of the voxelization in the GPU (the computation can be done efficiently by using the rasterization tool built on the GPU), so it would be more efficient, but the problem is that it does not store the list of triangles that each one of the voxels intersects with. And then, they compute the contribution of the light in a discretized shape of the objects, producing some aliasing in the boundary of the objects.

A similar option was developed in [6]. It consists of using the voxelization grid and linked-list A-buffers. It uses several linked-lists to store the triangles that each voxel intersects with.

This approach is similar to our proposal, we apply a voxelization of the spacecraft to be later sent to the GPU. But one of the differences is the additional textures that are sent to the GPU. In our proposal, we sent four 1-D textures with the values of the vertices, normal vectors, materials and triangles. Although, the information sent for the triangles correspond to the indices of the vertices, materials and normal vectors from the previous textures.

In [16], it sends A-buffers with the information of the vertices, brdfs, normal vectors and depth values. The depths values are used to test if a point is inside the rendered object.

The main difference is that in our proposal we have developed a structure similar to a stack that is saving the intermediate computations when it is doing the Raytracing. Thanks to the use of this structure, it does not require to use a large part of the memory of the shader and it scales well with the number of rays that are tracing.

However, the other approach has problem dealing with a long number of rays (the author says that his approach does not support tracing more than 64 rays).

A possible improvement could be done by using kd-tree structures, as it is explained in [19], [9] & [5] for the construction, ray traversal and the use of advance structures, respectively. It would enhance the efficiency on the computation of SRP

accelerations, but the computational time on doing the recursive voxelization for each layer of the kd-tree could be expensive.

In summary, we propose different accelerations based on a single pass GPU strategy:

- For the Raycasting model of the SRP computation, we use a GPU based Z-Buffer technique [15] which produces the same output as the Raycasting technique but faster, although the computation is done in a different way, as it is explained in further sections.

- In relation to the Raytracing model of the SRP computation, we propose two GPU based Raytracing approaches: one based on a structure of triangles and edges, and another one based on a voxelization of the spacecraft. Both approaches consider single scattering and multiple scattering.

- In our voxelization approach, each voxel contains the information of all the triangles that forms the spacecraft model and intersects with the voxel.

  Then, the voxels are sent to the GPU as a texture to improve the performance of checking the intersection of a ray with the triangles of the spacecraft.

  Once we have the voxelization of a spacecraft, it is important how the iteration over the voxels is done. In [1], it is described a way of iterating over the voxels in an efficient way.

  However, it was tested that the accuracy obtained was not good enough and some modifications were needed to add such as checking the boundary of the bounding box that includes the spacecraft and reducing the step used to jump from a voxel to the next one.

  One important thing was the implementation of a structure (which is based on a stack) that optimize the space required to do the Raytracing.

3.2 Data Visualization

Nowadays, the way that the data is visualized has become very important these last years. More and more people are realizing the importance of having a good or a bad visualization of large amount of data. Not only because it can help you explain what you have obtained, but also it will require less effort for the audience to understand it.

The literature about visualizing SRP acceleration is not very wide. In most cases ([11], for example), it consists of a chart with two axes: one is the magnitude of the SRP acceleration and the other one is the time.

This is a bit similar on how we represent the SRP accelerations although we do not consider time. But, let us imagine we have a static satellite or device, because the earth is always in motion, from the perspective of the satellite, it will receive the SRP acceleration from different directions (according to the time).

In our approach, we want to compute the SRP acceleration, given the Sun-spacecraft direction, for each possible rotation (in a discrete way) of the vehicle. And to do so, it was easier to think that the spacecraft is fixed and what is changing is the sunlight direction (which it is the same, considering the local coordinates of the spacecraft and not global coordinates).

At the end, both approaches consider the object fix and the light coming from different directions. However, the chart used by [11] only shows the magnitudes of the accelerations obtained in one day (see Fig.5). From this chart, it cannot be known from which direction the acceleration is coming. In addition, because the earth does not follow a circular trajectory around the Sun, depending on the day, the data shown in the chart could not be true.

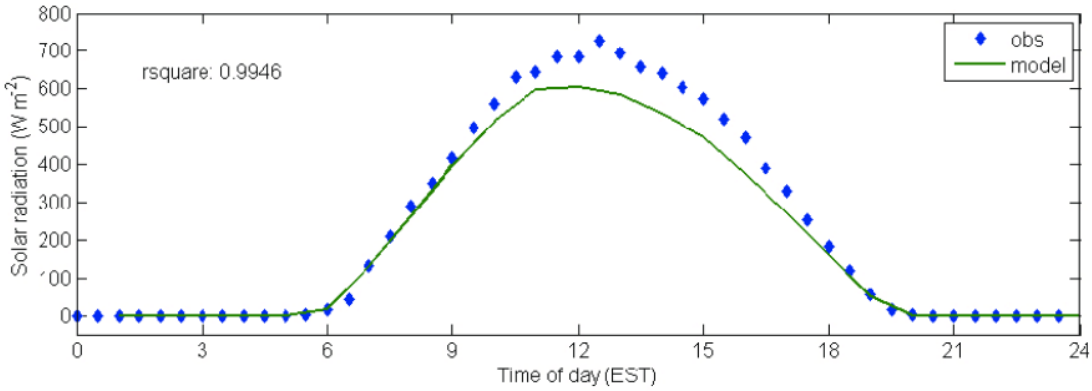

Figure 5: Visualization of the solar radiation magnitude during one day, obtained from [11].

Another approach, in terms of visualization, was done by [10]. In this case, the visualizations that were performed give more information to the user. In Fig. 6, the author represents the accelerations in a 2D chart, but it uses also two elements as new dimensions to represent the data: the **size** of the elements and the **colours** which are based on the heat map.

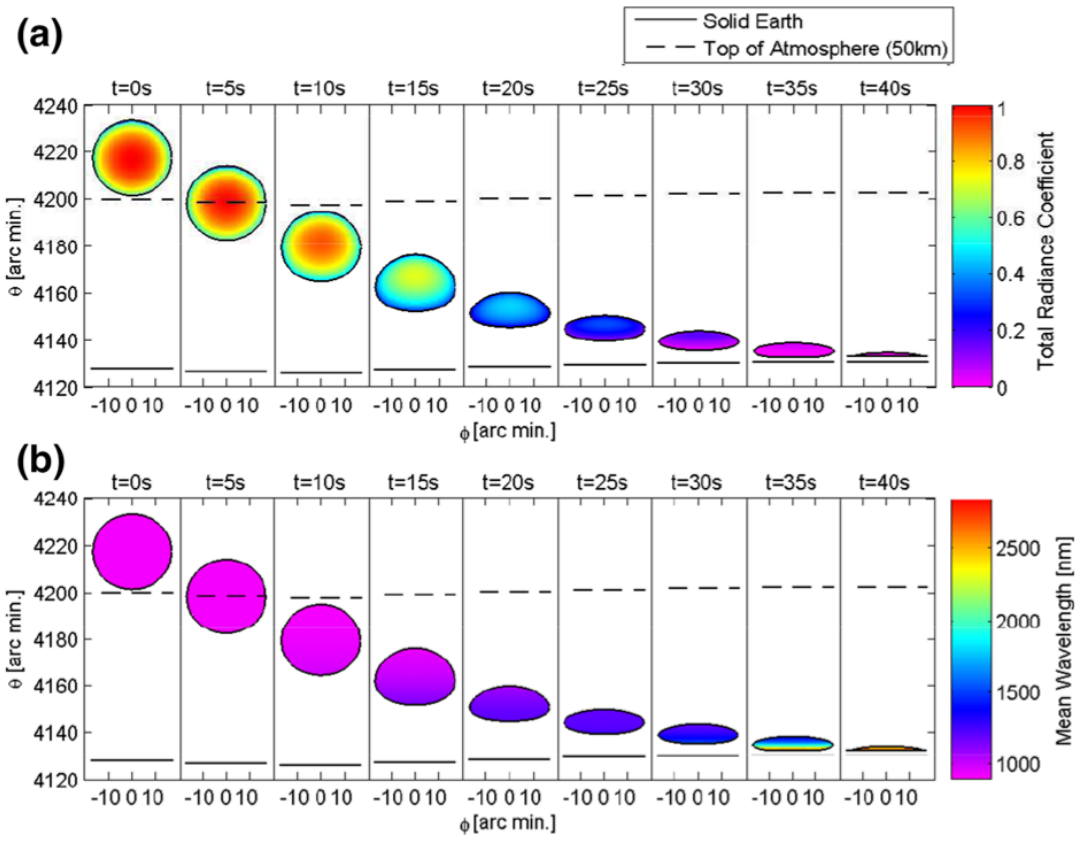

Figure 6: Visualization of the solar radiation pressure, obtained from [10].

Another chart (see Fig. 7) proposed previously by [10] consists of showing the solar radiation according to the wavelength which is a different way of visualizing these accelerations.

In our approach, we used the heat map as well, to identify the lower values of the accelerations, with blueish colours and higher values with reddish colours. Moreover, in our case, we compute a set of SRP accelerations (considering the sunlight direction coming from a discrete representation of points in a sphere, according to two angles: azimuth and elevation) and visualize them as four 3D viewers, where the viewers show: the magnitude and the 3 axis components of the accelerations.

The user can interact with each one of the viewers (rotate the scene, zoom in and out and get details on demand when one of the accelerations in the chart is selected). In the other approaches, the user cannot interact with the charts.

Furthermore, another option was added in our proposal, where the user can visualize particular forces (up to the user) together with a visualization of the spacecraft and the Sun in a 3D viewer.

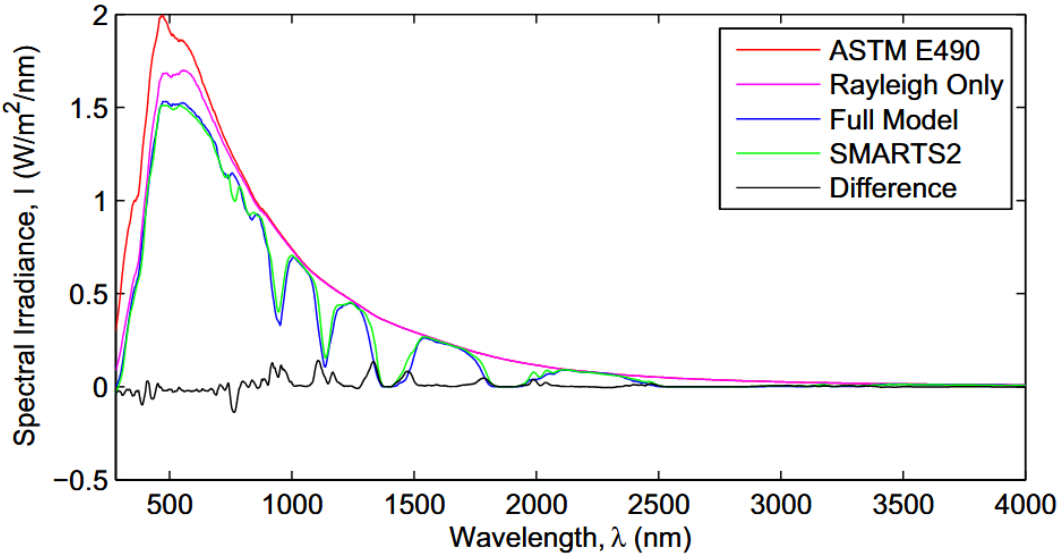

Figure 7: Visualization of the solar irradiance respect to its wavelength, obtained from [10].

Finally, we also included the possibility of downloading the information of the charts into a txt file, following the format produced by the SPAD program (Solar Pressure and Aerodynamic Drag) which was validated by members of the NASA and it can be imported to other programs.

4 Computation of SRP accelerations: our proposal

In previous sections, it was presented the current models that are used nowadays to approximate SRP accelerations. These models, even though they do not approximate faithfully the shape of the spacecraft, they are good enough to detect the possible perturbations in the spacecraft's trajectory.

In this chapter, we are proposing more precise models based on the tracing of rays and as they are high time consumers, we propose different optimizations based on GPU implementations.

4.1 High fidelity models

This is a family of models that are based on the Raycast and Raytrace techniques.

For these models, the shape of the spacecraft is approximated by a CAD model, composed by a mesh of triangles. In particular, each triangle contains the information of three vertices, a normal vector and its material properties (such as the reflectivity properties seen in the flat surface model).

4.1.1 Raycast model

The SRP acceleration is approximated using Raycast technique, where only the primary ray is considered. We use the Raytracing algorithm proposed by Whitted [18] to obtain realistic renderings. And also, the Monte Carlo integration [8] for the computation of diffuse reflection vectors.

This technique consists of creating a grid of $N_x \times N_y$ cells over a plane perpendicular to the sunlight direction, and enough big so the grid contains the projection of the spacecraft over it (as it is represented in Fig. 8).

And in our case, we use the same strategy to compute the SRP acceleration where the observer is in the Sun, and we compute the values of this acceleration at each cell of the viewport (window).

Then, for each cell of this grid, a ray is casted to check whether it intersects with any triangle of the spacecraft's surface and then, for the triangle with the closest intersection, the local SRP acceleration is computed as:

$$\vec{a}_{srp\_cell} = \frac{P_{srp}A_{cell}}{m_{sat}} < \vec{n}, \vec{r}_s > [(1-\rho_s)\vec{r}_s + 2(\rho_s < \vec{n}, \vec{r}_s > + \frac{\rho_d}{3})\vec{n}]$$

Considering the cells being really small, the addition of the contribution of each cell approximates an integration.

The formula of the SRP acceleration can be expressed as:

$$\vec{a}_{srp} = \frac{P_{srp}A}{m_{sat}} \int_{\delta\Omega} (< \vec{n}, \vec{r}_s > [(1-\rho_s)\vec{r}_s + 2(\rho_s < \vec{n}, \vec{r}_s > + \frac{\rho_d}{3})\vec{n}]) \tag{7}$$

Where A is the addition of the cell's areas which is constant. As mentioned previously, N_x, N_y are the size of the grid and $\delta\Omega$ corresponds to the surface of the spacecraft.

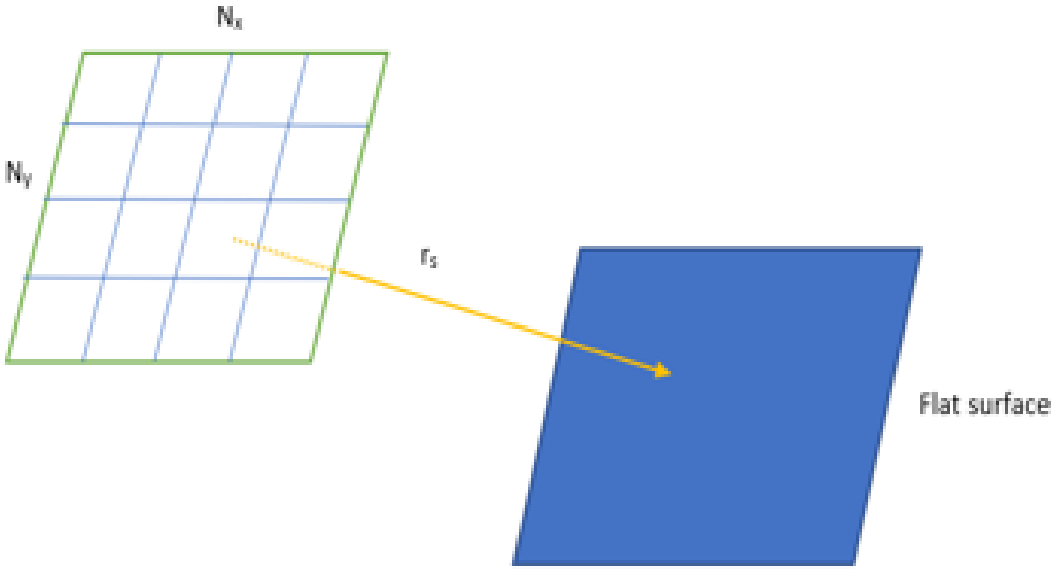

Figure 8: For each cell of the grid, a cast of a ray is performed.

Find triangles method.

It is based on the Raycast model and it is an approximation of the Raycast method.

This method consists of a grid defined accordingly to the size of the spacecraft, so, for each cell of the grid, a ray is casted. If the ray intersects a triangle of the spacecraft mesh, this triangle is added to a list.

So, at the end, for each triangle of the list of intersected triangles, the SRP acceleration is computed as it was defined in the formula 7.

Algorithm 1: Find triangles method

Input: r_s, N_x, N_y

Output: a_{srp}

findTriangles$(\vec{r}_s, N_x, N_y)$

$T = \#$ (flat surfaces)

X(t) $= 0$, $\forall t \in \{1, \cdots, T\}$

for $cell \in grid(N_x, N_y)$ **do**

  r = ray(cell, r_s)

  t = getClosestCollidedFlatSurface(r)

  if $t \in \{1, \ldots, T\}$ **then**

    X(t) $= 1$

$\vec{a}_{srp} = \frac{P_{srp}A}{m_{sat}} \sum_{i=1}^{T} (X(i) < \vec{n}_i, \vec{r}_s > [(1 - \rho_s^i)\vec{r}_s + 2(\rho_s^i < \vec{n}_i, \vec{r}_s > + \frac{\rho_d^i}{3})\vec{n}_i])$

return $\vec{a}_{srp}$

Raycast method.

Initially, the implementation of the Raycast model is similar to the Find Triangles method. It considers a grid where a ray is casted for each cell of the grid. If the ray intersects with a triangle of the spacecraft mesh, the SRP acceleration is computed as defined in the formula 7.

The difference with the Find Triangles method is that, in this case, the computation is done for each cell and for the other method, it is done only once for each hit triangle.

Algorithm 2: Raycast method

Input: r_s, N_x, N_y

Output: a_{srp}

Raycast$(\vec{r}_s, N_x, N_y)$

$T = \#$ (flat surfaces)

$\vec{a}_{srp} = 0$

for $cell \in grid(N_x, N_y)$ **do**

    r = ray(cell, r_s)

    t = getClosestCollidedFlatSurface(r)

    if $t \in \{1, \ldots, T\}$ **then**

        $\vec{a}_{srp} = \vec{a}_{srp} + \frac{P_{srp}A}{m_{sat}}(< \vec{n}_t, \vec{r}_s > [(1 - \rho_s^t)\vec{r}_s + 2(\rho_s^t < \vec{n}_t, \vec{r}_s > + \frac{\rho_d^t}{3})\vec{n}_t])$

return $\vec{a}_{srp}$

4.1.2 Raytrace model

The SRP acceleration is approximated using the Raytrace technique, taking into account primary and secondary rays.

For our project, spacecrafts are considered to be opaque i.e. they are not transparent because in the practise these vehicles or devices do not have any transparent material on their surfaces, although absorption reflections were implemented but not used at the end.

As in the Raycast model, we consider a grid where a ray through the sunlight direction $\vec{r}_s$ is casted for each cell of the grid (see Fig. 9). If a ray intersects with any triangle of the mesh, then secondary rays are casted but following a reflected direction according to the incident light.

In particular from the incident ray, we compute:

- A specular ray, computed as the reflected direction depending on the incident ray direction and the surface normal.
- A set of diffuse rays casted into random directions.

Each ray contribution will be weighted with the corresponding to the specular and diffuse properties of the spacecraft.

So, the SRP acceleration of the Raytrace model can be defined as:

$$\vec{a}_{srp} = \frac{P_{srp}A}{m_{sat}} \int_{\delta\Omega} (< \vec{n}, \vec{r}_s > [(1-\rho_s)\vec{r}_s + 2(\rho_s < \vec{n}, \vec{r}_s > + \frac{\rho_d}{3})\vec{n}]) \tag{7}$$
$$+\rho_s \vec{a}_{srp}(reflect_{specular}(\vec{r}_s))$$
$$+\frac{1}{k}\rho_d \sum_{i=1}^{k} \vec{a}_{srp}(reflect_{diffuse}(\vec{r}_s))$$

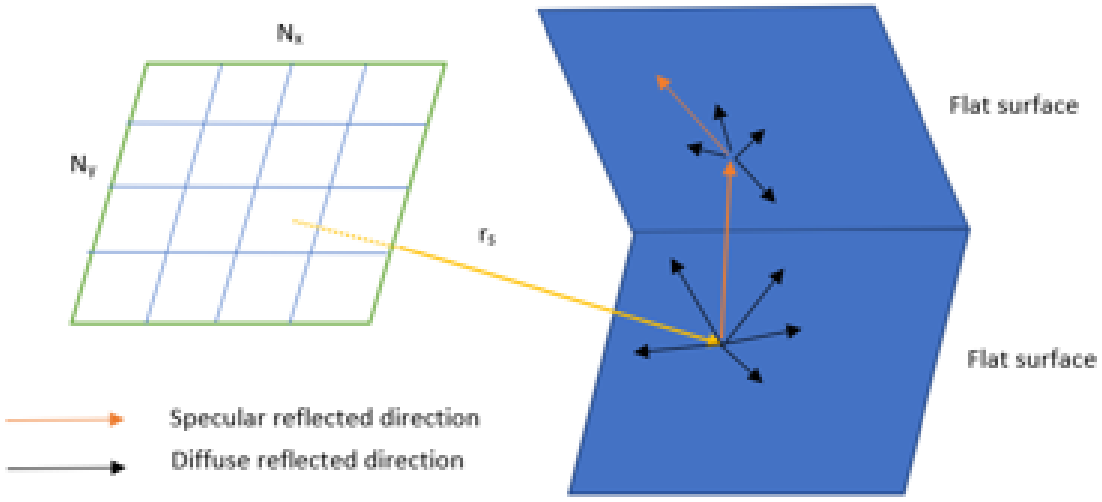

Figure 9: For each cell of the grid, a cast of a ray is performed. And for each one of those, other rays are casted.

Raytrace methods.

It is based on the Raytrace model and it is an extension of the Raycast method. For each cell of the grid, a ray is casted and if it hits a triangle then the computation of SRP acceleration is done. However, in this case, other secondary rays are casted following the direction given by the reflection (specular or diffuse) of the sunlight direction. Then, if any of these new rays hit a triangle, its contribution is computed and new secondary rays are casted considering the new reflected direction.

Although this method improves the accuracy, depending on the model it could spend a lot of time to do the computation because its complexity became exponential.

For this reason, it was considered the case of only casting one new secondary ray each time a ray intersects with a triangle of the mesh. Because the complexity becomes linear and it is useful when the material of the spacecraft's surface is completely lambertian or completely metallic.

This method that only considers one secondary ray is called Single Scattering and the other one that casts several rays is called Multiple Scattering.

Single Scattering: Here, it is described the algorithm that represents this method.

Algorithm 3: Raytrace Single Scattering method

Input: r_s, N_x, N_y

Output: a_{srp}

Raytrace($\vec{r}_s, N_x, N_y$)
  $\vec{a}_{srp} = 0$
  for $cell \in grid(N_x, N_y)$ **do**
    $\vec{a}_{srp} = \vec{a}_{srp}$ + **Raytrace_SS(cell, $\vec{r}_s$, 0)**
  return $\vec{a}_{srp}$
Raytrace_SS(cell, $\vec{r}_s$, numReflections)
  T = # (flat surfaces)
  if numReflections == MAX_REFLECTIONS **then**
    return (0,0,0)
  force = 0
  r = ray(cell, r_s)
  t, hitpoint = getClosestCollidedFlatSurface(r)
  if $t \in \{1, \ldots, T\}$ **then**
    $force = \frac{P_{srp}A}{m_{sat}}(< \vec{n}_t, \vec{r}_s > [(1 - \rho_s^t)\vec{r}_s + 2(\rho_s^t < \vec{n}_t, \vec{r}_s > + \frac{\rho_d^t}{3})\vec{n}_t]) + \rho_s^t \times$
    $Raytrace\_SS(hitpoint, reflect_{specular}(\vec{r_s}), numReflections + 1)$
  return force

Multiple Scattering: In this case, both specular and diffuse reflectivity properties are being considered in order to do the reflection for the secondary rays, as it is described in the following algorithm.

Algorithm 4: Raytrace Multiple Scattering method

Input: r_s, N_x, N_y

Output: a_{srp}

Raytrace($\vec{r}_s, N_x, N_y$)
  $\vec{a}_{srp} = 0$
  for $cell \in grid(N_x, N_y)$ **do**
    $\vec{a}_{srp} = \vec{a}_{srp}$ + **Raytrace_MS(cell, $\vec{r}_s$, 0)**
  return $\vec{a}_{srp}$
Raytrace_MS(cell, $\vec{r}_s$, numReflections)
  $T = \#$ (flat surfaces)
  if numReflections == MAX_REFLECTIONS **then**
    return (0,0,0)
  force = 0
  r = ray(cell, r_s)
  t, hitpoint = getClosestCollidedFlatSurface(r)
  if $t \in \{1, \ldots, T\}$ **then**

$$force = \frac{P_{srp}A}{m_{sat}}(< \vec{n}_t, \vec{r}_s > [(1-\rho_s^t)\vec{r}_s + 2(\rho_s^t < \vec{n}_t, \vec{r}_s > + \frac{\rho_d^t}{3})\vec{n}_t]) +$$
$$+ \rho_s^t \; Raytrace\_MS(hitpoint, reflect_{specular}(\vec{r_s}), numReflections + 1) +$$
$$+ \tfrac{1}{k}\rho_d^t \; Raytrace\_MS(hitpoint, reflect_{diffuse}(\vec{r_s}), numReflections + 1)$$

  return force

4.2 Optimization

Now, for the methods that use both the CPU and the GPU, they cannot be called directly as it happens with the only CPU methods, because the computation is done in different threads, so there was a need to synchronize them.

Here it is explained briefly the process of computing SRP accelerations when one of the methods that uses the GPU is invoked.

1. A triangular mesh is loaded from the information of the spacecraft given by the user. Then, its vertices, normal vectors and triangles are sent to the GPU.

2. In case the method needs extra information, it is sent via textures.

3. Each time a method that uses the GPU is invoked, its state (attributes) is added to the queue of computation.

4. In another thread, the one responsible to refresh what is being visualized in the GLWidget with the paintGL method, it is also in charge of checking whether there is data in the queue to be computed. If it so, then the GPU pipeline is called in order to compute the method with the data retrieve from the queue. Finally, the output of the GPU is obtained and processed to get the final computation of the SRP acceleration,

5. In the meantime, in the thread that initially called the method to compute the SRP acceleration, is waiting for the response of the thread that called the GPU pipeline.

The main difference between these methods is about the extra information that it is send to the GPU and how the computation is performed in the fragment shader. In the following section, we will explain the main proposed optimizations.

4.2.1 Optimization of SRP based on Raycasting

At the end of the computation of the Raycast method, it is obtained the pixels where the faces of the spacecraft were projected into the viewport (the window with the grid of cells where the rays were casted). In addition, we are considering the spacecraft CAD model to be opaque, so we can apply the ZBuffer technique and we will get the same result as the Raycast method.

In this approach, the mesh of the spacecraft (its vertices, normal vectors, triangles and reflectivity material properties) are sent to the GPU (some of them via VAO object such as the vertices, normal vectors and triangles, and via uniform the rest of parameters such as the sunlight direction).

Finally, in the fragment shader, for each pixel (or fragment), the formula seen in 4 is computed.

4.2.2 Optimization of SRP based on Raytracing

In general, the main difference between the only CPU versions and the ones that use the GPU is *where* the computation is done and *how* the parameters needed are obtained.

We used the GPU-based Raytracing technique proposed by [2], where we send in a one single pass the faces of the bounding box of the spacecraft to active the corresponding fragment shaders and the rest of parameters are sent via 1-D textures.

On the GPU, we have the fragments that belong to the visible faces of the bounding box of the spacecraft. There, each fragment performs a cast of a ray (primary ray) and checks if the ray intersects with any face of the model and if so, it gets the triangle with the closest intersection.

The closest intersection obtained belongs to the surface of the spacecraft and from this point, secondary rays can be casted (considering specular or diffuse reflections).

In relation to the data sent to the GPU, the vertices, the triangles, the normal vectors and the materials are sent as 1-D textures, while a cube is sent via VAO object.

In the Fig. 10 there is a representation of these textures.

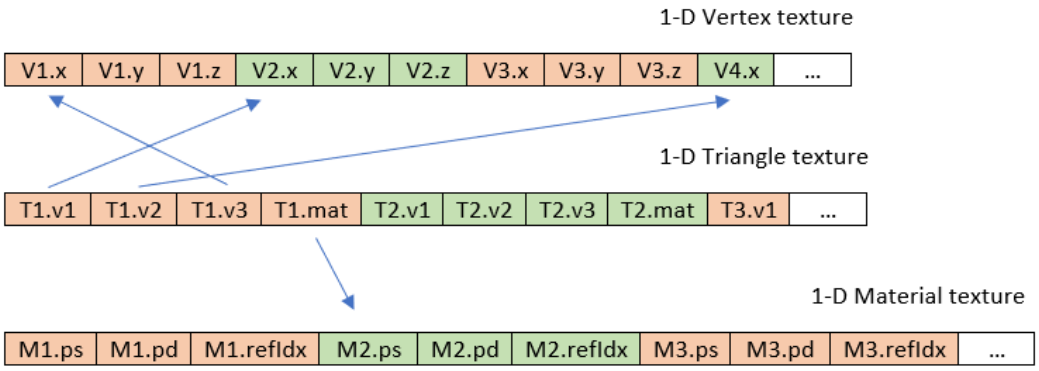

Figure 10: It is showing the main textures sent to the GPU.

Basic Raytrace method

Raytrace methods are more accurate than the other methods, although their computational time are also higher. The main reason of this is due to the test of intersection between a ray and the set of triangles. For each ray that is casted, it needs to test whether it intersects with any triangle of the mesh and if so, retrieve the one with closest intersection.

One problem with the GPU-based version is about decoding the data to be sent to the GPU as textures. One restriction that these textures has is that their values must be between 0 and 1. So, a conversion factor is applied to translate (and

possibly, scale) the data of the mesh before to be sent. And then, on the fragment shader the inverse of the conversion factor is applied to retrieve the original values.

For this reason, it was decided to translate the spacecraft model to the center of origin and scale it by dividing each vertex with the biggest axis length. Therefore, the values of the vertices are between -1 to 1.

The conversion factor applied was: $convert(vertex) = \frac{vertex+(1,1,1)}{2}$ in order to obtain vertices with values between 0 to 1. This same conversion factor is applied for the normal vectors due to the values of their components are also between -1 to 1.

In the case of the triangles, each triangle contains the information of the index position (in the vertices list) of 3 vertices, the normal index position from the normal list and also which material the triangle has. Because all the values of the triangle table are indices (positions from a table), we took for each index, the division between itself and its biggest value in its corresponding table. Therefore, the values of the indices would be between 0 and 1.

For the diffuse reflections, it was required to obtain a way of computing random values. So, a texture was generated with random values so the fragment shader can use to obtain samples of the diffuse reflection.

However, the problem with these conversion factors is that an error of precision is being added. For example, let us imagine we have 1000 vertices, then their indices would be values such as $\{0, 0.001, 0.002, 0.003, ..., 1\}$ and the small error in obtaining the values from the texture at the GPU would make the program access to vertices that the triangle does not contain. In relation to this problem, we studied how to increase the precision of the values in the textures and also, an epsilon was introduced in some cases at the fragment shader in order to avoid the problem of reading values that do not correspond.

Raytrace Voxelization methods.

A new method was developed to solve the problem of checking the intersections between a ray and all the triangles in the mesh. It is based on the Raytrace model, and it consists of voxelizing the spacecraft once the spacecraft information is loaded. This method is based on the articles [14] [13] and the code provided[1].

[1]The implementation of this method is based on the code provided by the author of those two articles [14] [13]. https://github.com/davidstutz/mesh-voxelization

It means that a new data structure was considered to represent the spacecraft without losing the information of the surface of the model as it happens in the Cannon-Ball and the N-Plate. This data structure is a 3D grid where each voxel will have the set of triangles that the voxel intersects with. And later on, it will be sent to the GPU as a 3D texture. In particular, a new 1-D texture is created and it contains the indices of the triangles contained for each one of the voxels.

Then, each voxel on the 3D texture stores two values: one is the index that points to a position from the previous 1-D texture (the one containing the indices of the triangles for each voxel) and also, the number of triangles that this voxel contains. So, the index for the next voxel in the 3D texture can be easily deduced as the addition of the index and the number of triangles of the previous voxel. One example is shown in the Fig.11.

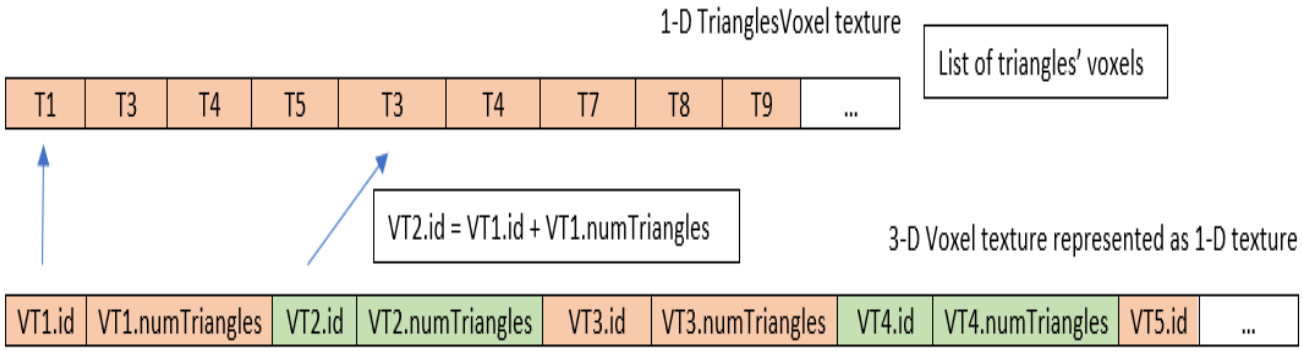

Figure 11: Additional textures to be sent to the GPU.

Then, instead of checking if a ray intersects with every single triangle that belongs to the spacecraft, it only needs to check the triangles that belongs to the voxels that intersects with the sunlight direction, as it can be seen in the Fig. 12. Therefore, the intersection of the ray with a subset of the whole set of triangles from a voxel is being checked, instead.

Even though this method enhances the performance on computing the SRP acceleration, the precision of this method is slightly worse than the original Raytrace methods. This is due to the problem that if the algorithm that does the ray marching avoids a voxel then the triangles of this voxel would not be considered and probably the computation of the SRP acceleration for the pixel with this problem would not be done. Or maybe, the triangle obtained is not the one with the closest intersection because the voxel of that one has been skipped.

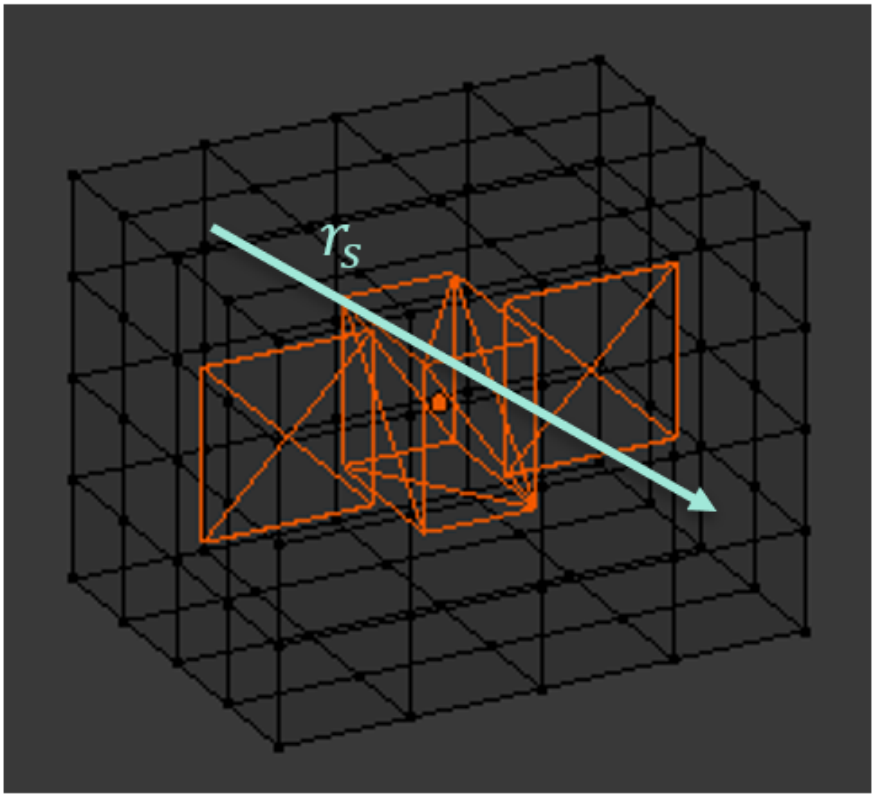

Figure 12: Representation of the voxelization of a spacecraft. It also indicates the sunlight direction for which cells are going to be checked.

This problem occurs, above all, when the intersection of the ray with the surface of the model is in the boundary of two voxels. Because, the error of precision at reading the values from the texture could make the program choose wrongly the voxel that the ray does not intersect with or at least, it is not the closest voxel to check its triangle intersections.

This technique of doing ray marching (iterating over the closest voxels to check the intersection with their triangles) is based on the article [1]. However, some modifications have been done, such as using smaller steps on the ray marching algorithm and also some epsilons were added (small values, 10^{-6}) to reduce this error.

5 Visualization

5.1 The problem

In the previous section, several methods were implemented to compute SRP accelerations. However, it is also important how the user can interact with them.

Firstly, the user should be able to select which SRP method they would like to use.

Then, the user should have the option of visualizing individual accelerations together with the spacecraft.

On the other hand, the user should be able to select how many samples would be taken in order to compute the global accelerations. It means that the SRP acceleration would be computed several times but considering the sunlight direction coming from a discretized set of points of a sphere.

Then, a set of plots should be displayed to the user and showing them the components (and also the magnitude) of the acceleration obtained for each sample.

And finally, another feature that must be added is the possibility of downloading the global accelerations into a txt file and it must follow the same format as the SPAD program does.

5.2 GUI design

For our proposal, we have designed a GUI that solves the requirements defined in the previous section. And, essentially, this GUI was designed to offer two types of visualizations to the user. One that facilitates the study of some selected accelerations and another one to do a more generic study. In Fig.13, there is the main window of this project. The first step (in the top part) is to load the spacecraft information and set its basic properties such as its mass and its material properties.

Then, the second step is to choose which method they would like to apply. Once a method is selected, it would appear a short explanation about the method.

In addition, it will display the parameters that the user can tune for the computation. For each one of these parameters that the user can modify, there is an icon with the letter 'i' that if you hover it, a message will appear explaining what this parameter would do in the computation.

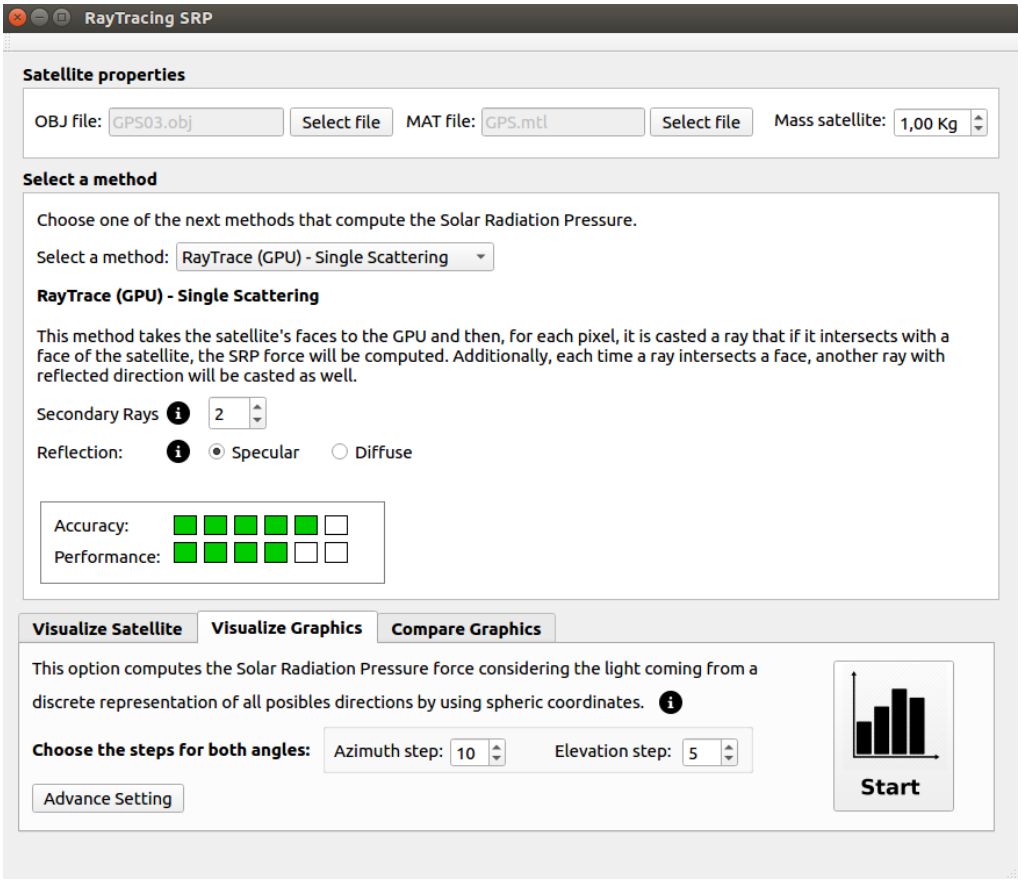

Figure 13: Main window of the application.

Parameters that the user can modify:

- Cannon-Ball: the user can set the area of the sphere (A) and the reflectivity property (Cr).
- N-Plate: the user must set the path to the file where the set of plates are defined.
- Find Triangles & Raycast: the user can set the number of cells (N_x and N_y) of the viewport (window) where the rays are casted.
- Raytrace Single Scattering: the user can set the number of cells (N_x and N_y) of the viewport (window) where the rays are casted. Also, they can set the number of secondary rays.
- Raytrace Multiple Scattering: the user can set the number of cells (N_x and N_y) of the viewport (window) where the rays are casted. Also, they can set the number of secondary rays and the number of diffuse rays that will be

sampled. (for the GPU methods, the size of the viewport is fixed: $N_x = 512$ and $N_y = 512$)

- ZBuffer: it does not have parameters to be set by the user.
- Raytrace Single Scattering: the user can set the number of secondary rays.
- Raytrace Multiple Scattering: the user can set the number of secondary rays and the number of diffuse rays that will be sampled.

Furthermore, two properties of the selected method will be shown: the accuracy and the performance of this method in comparison to the other ones. It is represented as a set of little green boxes in order to make it easier to understand to the users.

Finally, the third step corresponds to the three actions that the user can apply.

5.2.1 User actions

. Three actions were implemented for this project that users can perform to study SRP accelerations considering the user has picked up one of the possible methods.

Visualize Spacecraft. When the user has pressed the start button in the Visualize Spacecraft tab, it will show a 3D viewer of the spacecraft with its 3 axes and the sunlight direction as it appears in Fig. 14.

Then, the user can set the initial rotation of the spacecraft by interacting with the three sliders. Each one of them corresponds to one of the local axes of the spacecraft (an example is shown in Fig. 15).

For example, in the first slider, it appears 'X' in red and this indicates that the red line in the 3D viewer correspond to the x axis.

The user can also rotate the scene by pressing the right button of the mouse. It will not affect the computation of the SRP accelerations because it is modifying the orientation and position of the observer (camera) and nor the model neither the sunlight direction. And having pressed the left button, the user can zoom in and zoom out.

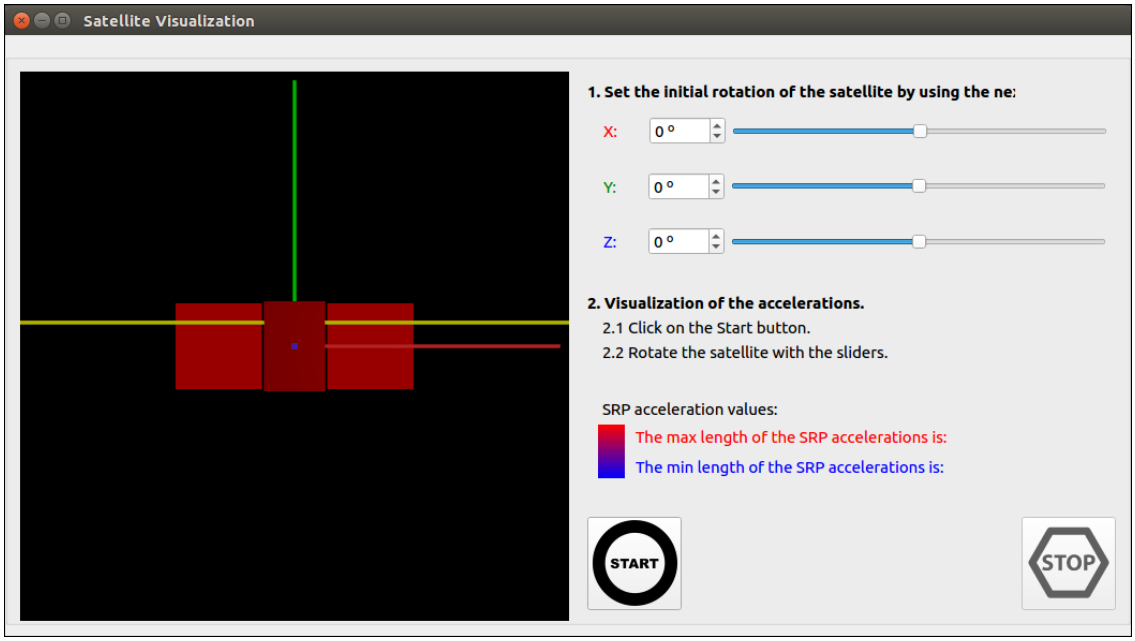

Figure 14: The window for visualizing individual forces.

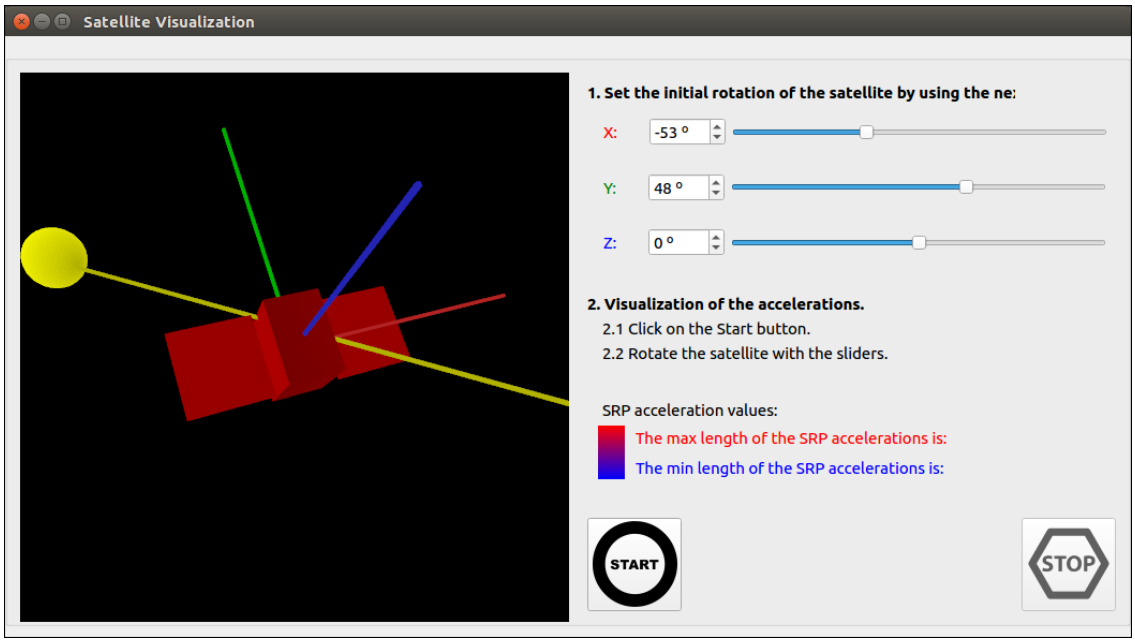

Figure 15: It was applied a rotation on the spacecraft and also in the scene.

Also, it allows the user to compute and visualize particular accelerations by rotating the spacecraft after having pressed the 'Start' button and consequently, interacted with the sliders as it is shown in Fig. 16.

These accelerations that will appear in the 3D scene would have a different colour depending on their magnitudes. It was chosen to use the heat map colours

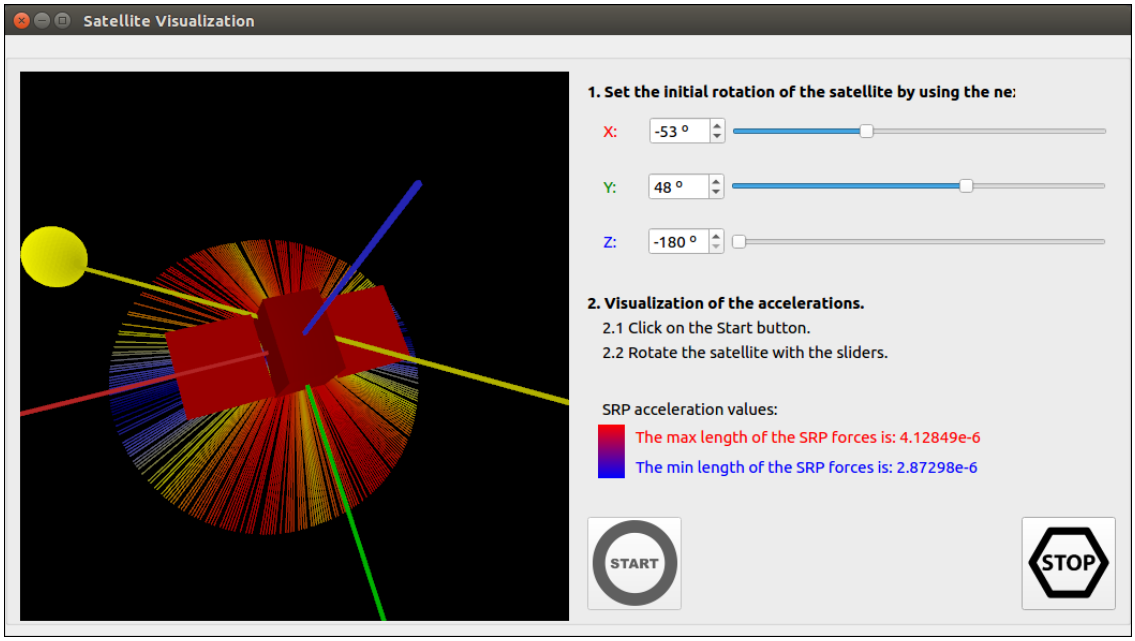

Figure 16: Several accelerations were computed.

to represent in blue the forces with lower magnitudes and in red the ones with higher magnitudes. Also, in this window it indicates which is the lowest and the highest magnitude among the accelerations that were computed.

Visualize Graphics. The tab of Visualize Graphics in the middle tab of the main window of this program allows the user to set some parameters before the global computation is done.

In particular, it lets the user to select the azimuth and the elevation steps (they indicate how many points are used to discretize sample points from a sphere).

These sample points would be the ones used for the sunlight direction, even though the sunlight direction can be considered fixed. Instead of applying a rotation to the spacecraft model, it is easier to consider the opposite case: the light is rotating and the spacecraft is considered to remain in the initial orientation set by the user.

Then, if a GPU-based method was selected, another option would be added to this tab and it allows the user to visualize the results obtained from SRP accelerations while the computation of the global accelerations is being done.

If the user pressed the 'Start' button, these accelerations would be represented in a window with four 3D viewers showing each one of the components of the accelerations (x, y and z) and also their magnitudes (see Fig. 17 and 18). In addition, users can download the result as a txt file.

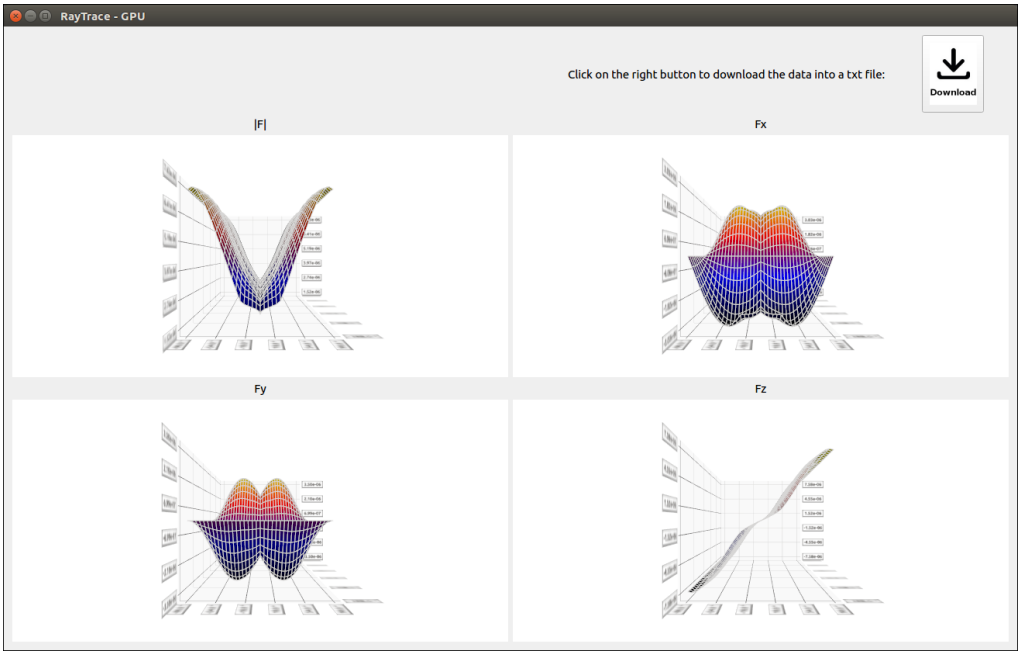

Figure 17: Window in charge of visualizing charts.

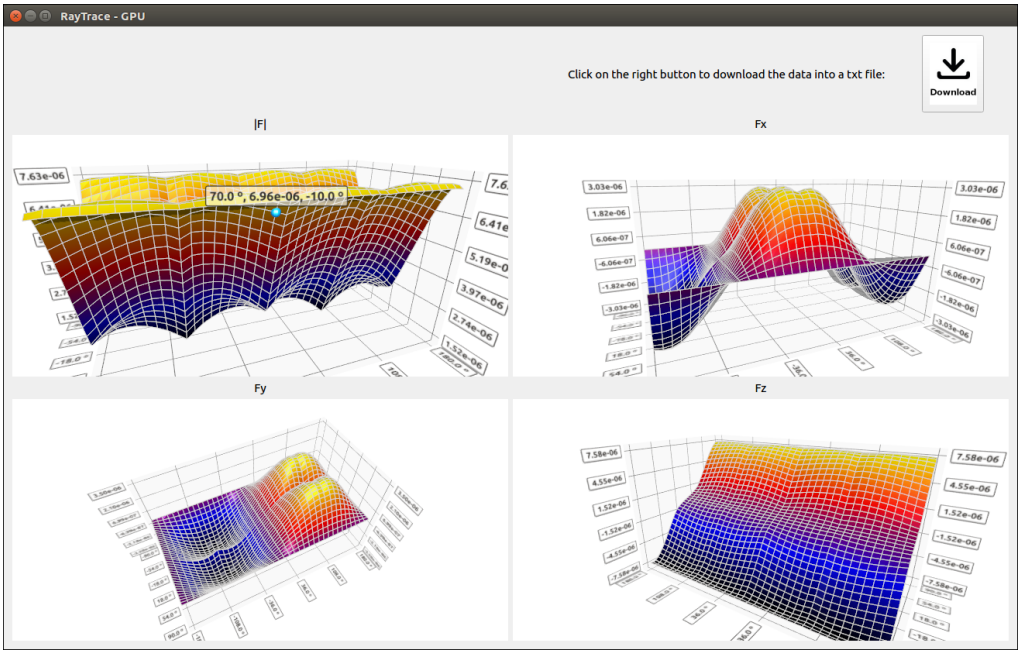

Figure 18: Window in charge of visualizing charts.

Compare Graphics. This option allows the user to compare the result of two graphics that were previously generated. It is important to have this tool of comparison because it lets the user to compute the difference between two already computed graphics. Also, it shows the mean square error (MSE) and the maximum difference between the points on the charts. (See Fig. 19, 20)

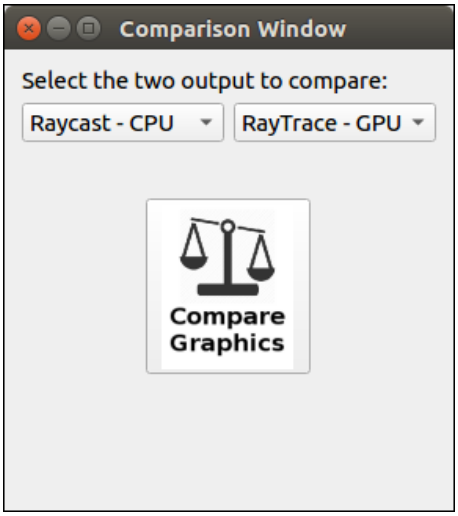

Figure 19: Comparison menu.

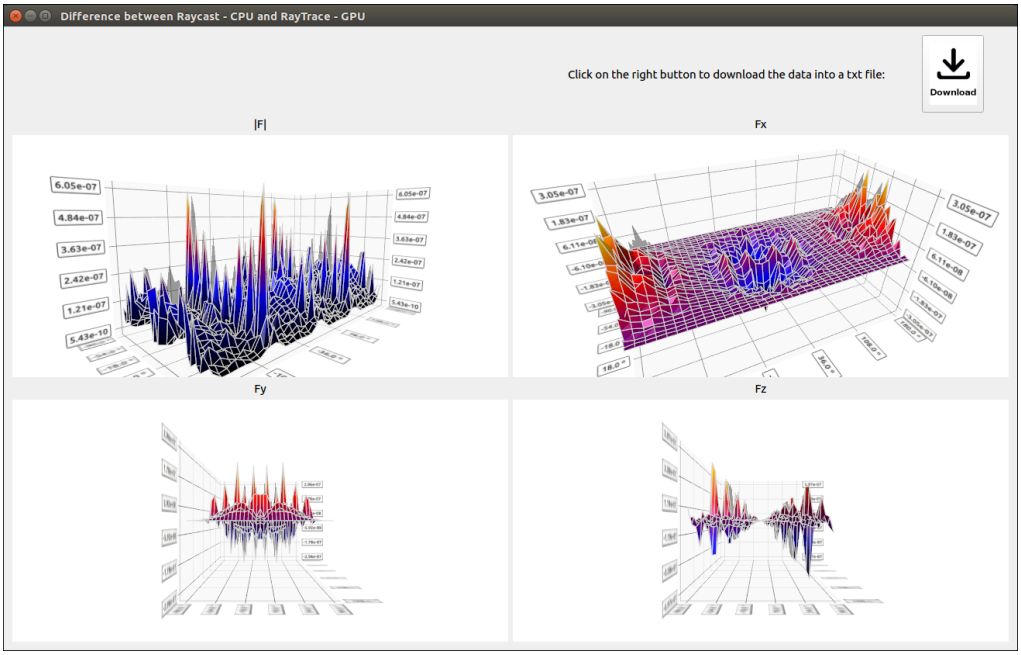

Figure 20: Difference between the charts.

6 Simulation and results

In this part of the thesis, we describe the software architecture of the project, with the main classes. Next, we describe the methodology used to validate the correctness of the implemented methods. Moreover, we analyse which methods are more accurate and which ones are more efficient. And finally, we present the usability test that was performed in order to improve the interaction of the user with the application.

6.1 Project software architecture

The solution that is proposed in this thesis consists of a C++ application, using the Qt framework. It was coded in C++ for the CPU part and GLSL (OpenGL Shading Language) for the GPU part. It includes the libraries of Eigen and glm, so the user does not need to install them on their computer.

First of all, we will detail the used methodology to analyse the correctness of the implemented methods. We implemented the CPU based solutions corresponding to the GPU based optimizations. Fig. 21 details all the implemented approaches.

Model		CPU	GPU
Cannon-Ball		YES	NO
N-Plates		YES	NO
Finite Element (based on RayCasting)	Find Triangles	YES	NO
	RayCasting / ZBuffer	YES	YES
Finite Element (based on RayTracing)	RayTracing (single scattering)	YES	YES
	RayTracing (multiple scattering)	YES	YES
Finite Element (based on RayTracing with Voxelization)	RayTracing (single scattering)	NO	YES
	RayTracing (multiple scattering)	NO	YES

Figure 21: Summary of the implemented methods and whether they use or not the GPU.

We implemented the Cannon-Ball and the N-Plate methods in the CPU to fix the baseline of the accuracy of SRP acceleration values.

The computation of **Cannon-Ball** method is a direct implementation of the formula 5, described in the Cannon-Ball model. It is simple to implement and fast

in terms of computational time. Although, it does not consider the shape of the spacecraft. The magnitude and the direction of the SRP acceleration is constant.

In the case of **N-Plate method**, it is similar to the Cannon-Ball method. It consists of applying the formula 6, described in the N-Plate model. It is also simple to implement and fast in terms of computational time. However, it does not consider occlusions between different plates.

In relation to the **Raycast** and **Raytracing** methods we implemented their CPU-based version in order to control the accuracy of the computation of GPU-based methods.

The main relationships between the different classes of the project are shown in Fig. 22. It starts with the main file (main.cpp) which generates a window (MainWindow) that the user can interact with, as it is shown in Fig. 13.

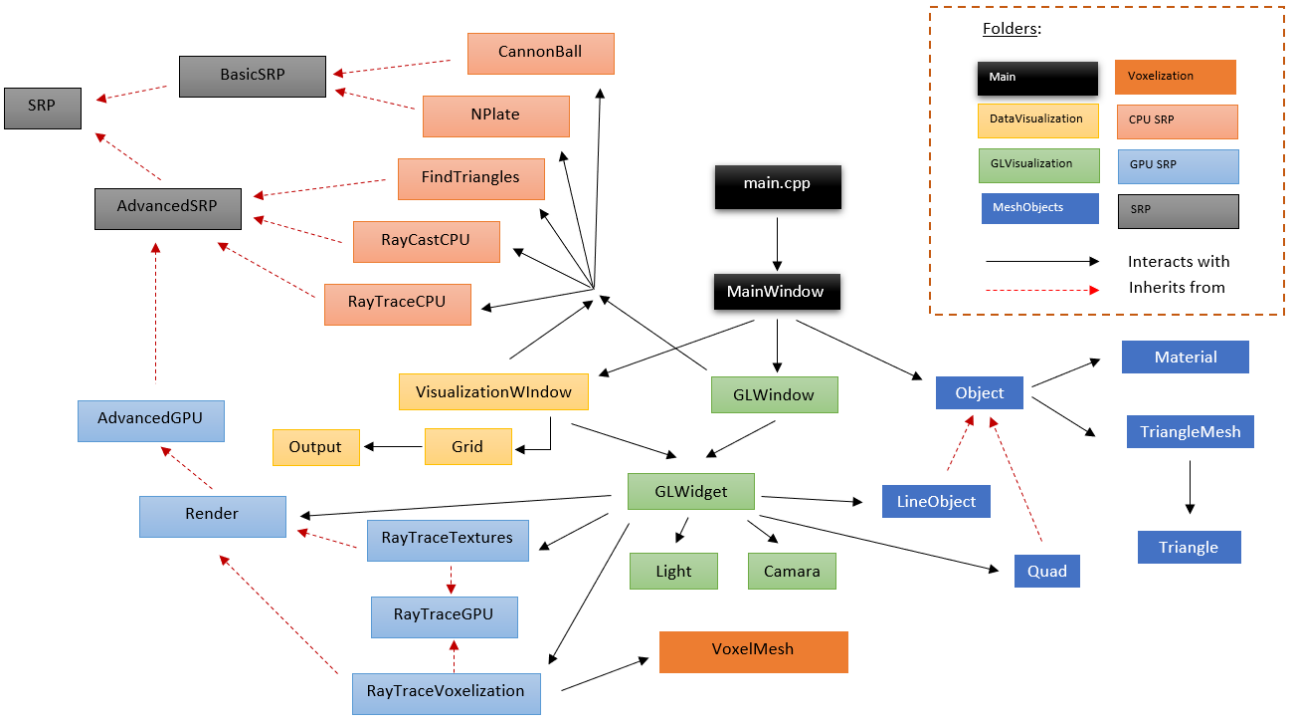

Figure 22: Represents the state of this project: its main classes, which folders contains and what classes are inherited.

6.2 Validation

It is important to validate that the output obtained from the developed methods is correct. For this reason, several tests were done in order to prove that.

6.2.1 Cannon-Ball test

The Cannon-Ball model considers the shape of the spacecraft to be a sphere. So loading a triangle mesh of a sphere and using another method (like one of the Raytrace techniques) produces the same result as applying the Cannon-Ball method.

In particular, the largest difference between the accelerations of both (Cannon-Ball and Raycast method, for example) was 4.7×10^{-7}.

6.2.2 N-Plate test

In this case, the N-Plate model considers the shape of the spacecraft to be represented as a set of flat plates (each one of them contains the information of the normal vector, area and reflectivity properties).

However, if there are two plates, one occluding the other one, the program would consider the occluded plate as well for the computation (which is a mistake). So, loading a triangle mesh of a cube or any simple object that does not have in any case occluded plates, and using another method (like one of the Raytrace techniques) it gives the same result as applying the N-Plate method.

In particular, the largest difference between the accelerations of both (N-Plate and Raycast method, for example) was 5.37×10^{-7}.

6.2.3 Raytrace test

One way to validate the Raycast technique is to compare the results given by two different implementations: the Raycast CPU method and the ZBuffer which uses the GPU and the algorithms of both of them are different in order to do the computation. Comparing one with the other, it proves that both of them give the same result.

Also, for the Raytrace single scattering using specular reflection, it can be shown graphically that it is computing the reflection of the spacecraft with other parts of it. Then, comparing with its CPU versions, the obtained error is very small.

But this is not a rigorous test. For this reason, it was requested to the NASA which objects (triangle meshes) and which material reflectivity properties they use, as well of the output of the best approximation that they have for the SRP acceleration, in order to compare if this can be replicated using our program and comparing both results (the one from the NASA and the other one computed by this project).

However, because of lack of level access to the real data from the NASA agency, it was not possible to obtain neither a spacecraft object and its material, nor the output.

6.3 Accuracy and performance

In this section, it is described which methods are more accurate and which ones are more efficient. In order to do so, a speed test was performed using the next input data:

1. The Box Wing model (it consists of a cube and two quads).
2. The size of the grid is (512, 512) cells (or pixels at the GPU).
3. The action of visualization graphics was used with azimuth step and elevation step of 20.
4. For the Raytrace methods, it was considered 4 secondary and diffuse rays.

All the results were obtained using an MSI laptop with an intel core i7 processor with a Nvidia GeForce 1050 GTX graphics card.

Fig. 23 shows a table with the time comparison of the different methods that have been implemented in this project. For the accuracy comparison, the CPU Raytrace Multiple Scattering method was chosen among the other ones as the ground truth (although it is not fully validated yet) because it performs multiple secondary rays with both specular and diffuse reflections. The CPU version was selected in front of the GPU one, not because of the performance but, for the accuracy because in the GPU version there is a small loss of precision with the transfer of data in the communication between the CPU and the GPU.

In the chart (Fig. 24), it is showing which methods are more efficient and which ones are more accurate.

It is important to take into account that the accuracy presented in the table (Fig. 23) shows the mean square error between the methods and the ground truth and for example, some of them are really small (10^{-9}) however, the values of the acceleration usually are $O(10^{-6})$. So, in fact the 'error' between the methods is in the order of $O(10^{-3})$ for this example.

In the next table (see Fig. 25), it is presented a version of the MSE fixing this problem (by multiplying the previous MSE's with 10^{-6}).

Method	Computational Time	MSE*
Cannon-Ball (CPU)	< 1 ms	$1{,}79 \times 10^{-6}$
N-Plate (CPU)	< 1 ms	$6{,}90 \times 10^{-7}$
Find Triangles (CPU)	20,26 min	$2{,}90 \times 10^{-7}$
Raycast (CPU)	46,00 min	$5{,}05 \times 10^{-8}$
Raytrace (CPU) – Single Scattering	50, 50 min	$4{,}52 \times 10^{-9}$
Raytrace (CPU) – Multiple Scattering	1,08 h	0
ZBuffer (GPU)	2,02 s	$5{,}04 \times 10^{-8}$
Raytrace (GPU) – Single Scattering	5,14 s	$5{,}33 \times 10^{-9}$
Raytrace (GPU) – Multiple Scattering	38,76 s	$4{,}07 \times 10^{-9}$
Raytrace (GPU) Voxelization – Single Scattering	2,82 s	$6{,}08 \times 10^{-9}$
Raytrace (GPU) Voxelization – Multiple Scattering	32,78 s	$4{,}82 \times 10^{-9}$

Figure 23: Comparison table of SRP methods.

The main difference between the CPU and the GPU methods is that the CPU methods have less error of precision and on the other hand, they do not parallelize the computation, so they are significantly slower in computational time than the GPU versions.

In the case of the Cannon-Ball and N-Plate, they are simple to implement and fast in terms of computational time. However, the Cannon-Ball method does not consider the shape of the satellite and also, the magnitude and the direction of the SRP acceleration is always constant.

The N-Plate method does not have this problem, but it does not consider occlusions between the different plates.

For this reason, a new model was studied (high fidelity model) where in contrast to the previous methods, it considers a CAD model for the representation of the surface of the spacecraft. Although, it is slower than the previous ones in terms of computational time.

From this model, we have implemented three techniques: Raycast (CPU), Find Triangles (CPU) and ZBuffer (GPU). As mentioned before, the Raycast and the

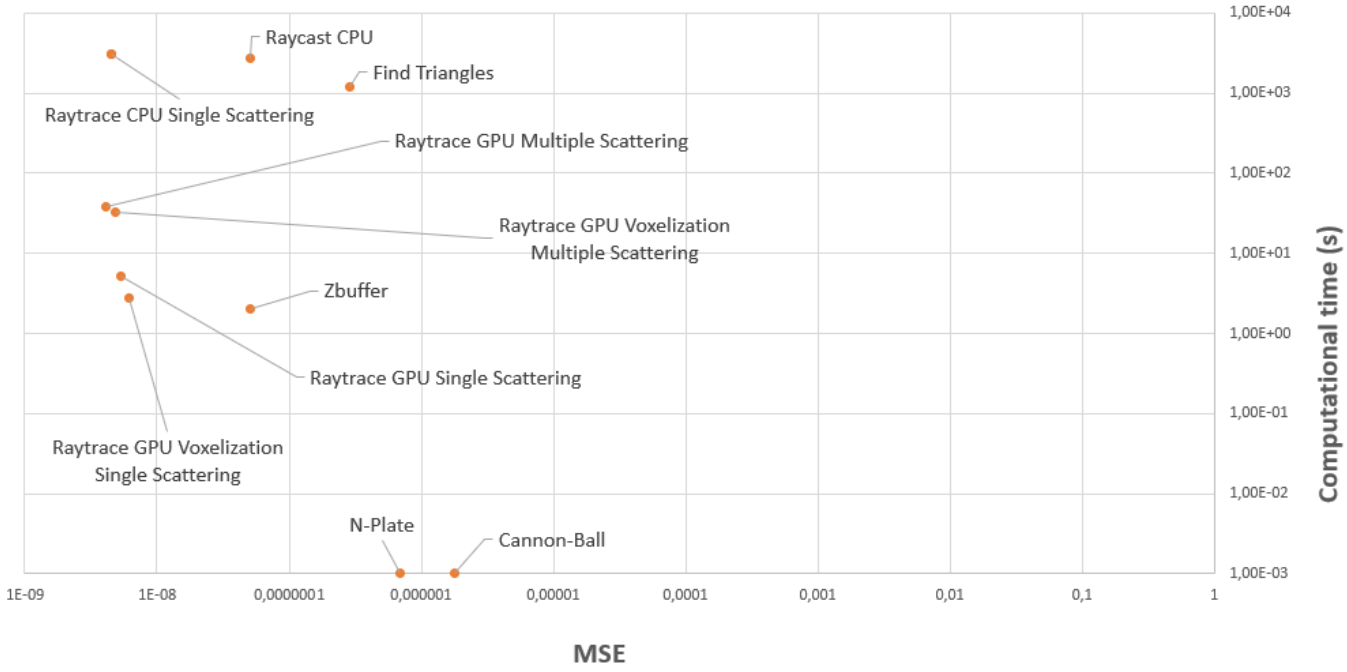

Figure 24: Chart representing the computational time and the MSE of the methods.

ZBuffer produce the same approximation of the SRP acceleration. Although, the Raycast method is slower in terms of computational time because it is not parallelizing anything and the ZBuffer has a higher precision error. The Find Triangle method, as it was said before, it is an approximation of the Raycast method, so its accuracy is a bit worst but faster than the Raycast but slower in comparison to the ZBuffer technique.

In relation to Raytrace methods, these ones are the only methods that consider secondary rays. The single scattering approach is faster than the multiple scattering because the first one has linear complexity (in terms of rays casted) and the multiple scattering has exponential complexity (for each casted ray, several reflected rays can be casted).

Finally, it should be noted that the voxelization of the mesh is faster although its accuracy is a bit worse. It could be due to the loss of some voxels during the sampling of the ray.

6.4 Usability test

The Research Question of this usability test is "How easy the user can interact with the different methods and visualizations".

Method	Computational Time	MSE*	Fixed MSE
Cannon-Ball (CPU)	< 1 ms	$1{,}79 \times 10^{-6}$	1,79
N-Plate (CPU)	< 1 ms	$6{,}90 \times 10^{-7}$	0,69
Find Triangles (CPU)	20,26 min	$2{,}90 \times 10^{-7}$	0,29
Raycast (CPU)	46,00 min	$5{,}05 \times 10^{-8}$	0,0505
Raytrace (CPU) – Single Scattering	50, 50 min	$4{,}52 \times 10^{-9}$	0,00452
Raytrace (CPU) – Multiple Scattering	1,08 h	0	0
ZBuffer (GPU)	2,02 s	$5{,}04 \times 10^{-8}$	0,0504
Raytrace (GPU) – Single Scattering	5,14 s	$5{,}33 \times 10^{-9}$	0,00533
Raytrace (GPU) – Multiple Scattering	38,76 s	$4{,}07 \times 10^{-9}$	0,00407
Raytrace (GPU) Voxelization – Single Scattering	2,82 s	$6{,}08 \times 10^{-9}$	0,00608
Raytrace (GPU) Voxelization – Multiple Scattering	32,78 s	$4{,}82 \times 10^{-9}$	0,00482

Figure 25: In this table, it is considered the correct comparison between the methods.

A usability test was performed at one of the centres of the NASA in Washington D.C. in order to improve the interaction of the users with the application. Because, at the end, they are the users that this application was created for.

In Fig. 29 it is shown a little introduction that was given to the users at the beginning of the usability test. Then, in the same figure, a full of tasks were given to the users, explaining what they need to do.

This test was performed by 8 expert users (aeronautic engineers, physics) that are working for the NASA. Nowadays, they use several programs to do the computation, the visualization of the graphics of the SRP acceleration. The program they use for the computation is called SPAD which computes a simple approximation of this acceleration and uses only the CPU. Therefore, these are the users that this application was intended for.

This usability test consists of 3 tasks, the first and the second one with individual objectives and the third one is optional. The objectives of these tasks were:

TASK1: The objective of this task is focused more on watching the interaction of the user with the application of this project.

1. Load the spacecraft information quickly.
2. Find the desired method without any problem.
3. Find the advance options and select the right option to display while the computation is being done. Also, find the button to visualize the graphics.
4. Find the button to download the data into a txt file.

TASK2: In this case, the objective is to study if the designed visualizations improves the comprehension of the data.

1. Load the spacecraft information quickly.
2. Find the method quickly.
3. Understand the parameters of the method.
4. Find the button to visualize the spacecraft.
5. Set the initial rotation of the spacecraft. How do they perform it? Via typing the number or with the slider?
6. Understand how the visualization works. Pressing the start button and then moving the slider (in this case, the Z axis slider).

TASK3: With this task, the user shows the main difficulties of using this application.

1. Explore the different methods and possible actions.

Afterwards, a satisfaction questionnaire was given to the users so they could explain if they had any problem or if there is something that could be improved (see Fig. 30, 31).

Evaluation of the results.

In overall, they were amazed about the performance of this application when using a GPU method in comparison of the CPU methods.

Also, they get excited about this application allowing them to download the data from the charts into a txt file with the same format as the SPAD program. Because they can use this output with other programs.

And they were surprised about the visualization of the spacecraft with particular accelerations. Until now, they have not used any program to do that and they liked that with this visualization, they can explain visually the effects of these accelerations.

Although, there were some part of the program at that moment that they would have liked to be improved such as:

1. The buttons to start the visualization of the spacecraft and the one for the graphics were not really helpful and instead of an image on the button they would prefer a text saying "Play" or "Start".

2. Another thing they mentioned is to add a progress bar in order to inform the user about the progression of the computation, because in some cases it could spend a lot of time (even so, if an only CPU method was selected).

3. Also, the part of visualizing the spacecraft, the representation of the colours that the accelerations took (based on the heat map colours) in order to show which ones are smaller or bigger was not very understandable.

All of these new features were added to the project. As it can be seen in Fig. 26 and Fig. 27 that the progression bar and the stop button where added both for CPU and GPU methods.

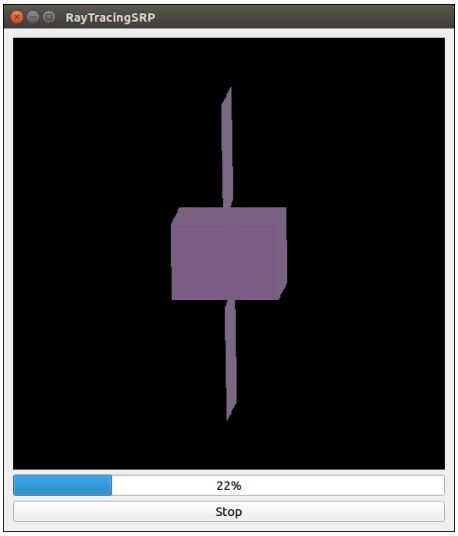

Figure 26: Progression Bar during the computation of (GPU) SRP methods.

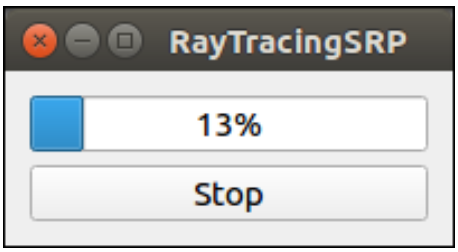

Figure 27: Progression Bar during the computation of (CPU) SRP methods.

7 Conclusions and Future Work

7.1 Conclusions

Several techniques were implemented to compute the SRP acceleration by considering different models to approximate it. The Raytrace methods are more accurate than the other ones and they are useful for missions considering the motion of the spacecraft in long term propagation.

The GPU-based methods are significantly faster than the CPU-based versions. The CPU methods could spend one day to do the computation with a complex model while the ones that use the GPU only spend a few hours.

In addition, apart from the computation, this project allows the user to save the information of the SRP accelerations into a txt file which follows the same format as the one given by the SPAD program (a validated program used by the NASA).

Also, it allows the user to visualize the forces in different ways: using charts to give a general view of the forces and also, a 3D viewer where the user can see particular forces represented together with the spacecraft representation.

Finally, we validated all the proposed approaches in terms of accuracy and time consumption, taking especial care to guarantee the minimum error in the GPU based methods. We had the great opportunity to test our proposed application with real experts of NASA agency. Results of these evaluations are very encouraging and open new extensions of our work.

7.2 Future Work

There are some things that can be done in order to improve this project:

1. Study about the voxelization of the spacecraft. Maybe there are some other data structures that can improve the performance or the communication between CPU and GPU, such as octrees for example.

2. After doing the usability test at the NASA, some of the things they would like this program to do are:

   - One part of the mesh to be static and another one to be constant in motion (one part fix, the other part not).
   - Use a similar formula but instead of computing SRP acceleration, aproximate the acceleration given by the wind (Drag effect).
   - Validate the results with real data (it is required access to the objects and reflectivity properties that they used).

3. Connect the visualization of particular accelerations with the charts of the computation of the forces considering the light coming from different points of a sphere (see Fig. 28).

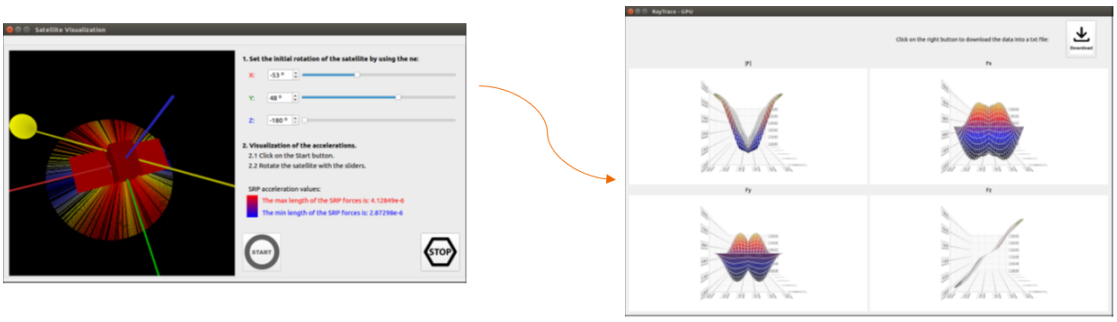

Figure 28: As one of the possible things to add to this project is the connection of both visualizations.

References

[1] Amanatides, J.; Woo, A.: A Fast Voxel Traversal Algorithm for Ray Tracing, *in Proceedings of EuroGraphics, volume 87* , 1987.

[2] Christen, M.: Ray tracing on GPU, *Master's thesis, Univ. of Applied Sciences Basel (FHBB), Jan*, volume 19, 2005.

[3] Estgren, M.; Hedin, R.; Jansson, E. S. V.: Monte Carlo Raytracing from Scratch, `https://caffeineviking.net/papers/mcrt.pdf`, 2017.

[4] Farres, A.; Folta, D.; Webster, C.: USING SPHERICAL HARMONICS TO MODEL SOLAR RADIATION PRESSURE ACCELERATIONS, *in AAS/AIAA Astrodynamics Specialist Conference, Columbia River Gorge, Stevenson, WA, Volume: (Preprint) AAS 17-780*, 2017.

[5] Foley, T.; Sugerman, J.: KD-Tree Acceleration Structures for a GPU Raytracer, *in HWWS '05 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware Pages 15-22* , `https://doi.org/10.1145/1071866.1071869` 2005.

[6] Hu, W.; Huang, Y.; Zhang, F.; et al: Ray tracing via GPU rasterization , *in Vis Comput (2014) 30: 697.*, `https://doi.org/10.1007/s00371-014-0968-8` 2014.

[7] Kajiya, J. T.: The rendering equation, *in ACM Siggraph Computer Graphics, volume 20, pages 143–150. ACM*, 1986.

[8] Keller, A.: Quasi-Monte Carlo Radiosity, *Rendering Techniques '96 (Proc. 7th Eurographics Workshop on Rendering), pages 101–110. Springer*, 1996.

[9] Reiter Horn, D.; Sugerman, J.; Houston, M.; Hanrahan, P.: Interactive k-d tree GPU raytracing , *in I3D '07 Proceedings of the 2007 symposium on Interactive 3D graphics and games Pages 167-174*, `https://doi.org/10.1145/1230100.1230129` 2007.

[10] Robertson, R.; Flury, J.; Bandikova, T.; et al.: Highly physical penumbra solar radiation pressure modeling with atmospheric effects, *in Springer Netherlands*, `https://doi.org/10.1007/s10569-015-9637-0`, 2015.

[11] Seok, B.; Helmig, D.; Ganzeveld, L.; Williams, M.; Vogel, C.: Dynamics of nitrogen oxides and ozone above and within a mixed hardwood forest in Northern Michigan, *Atmospheric Chemistry and Physics Discussions. 12*, 2012.

[12] Shumskiy, V.: GPU Ray Tracing - Comparative Study on Ray-Triangle Intersection Algorithms, *in Trans. Computational Science, volume 19, pages 78-91*, 2013.

[13] Stutz, D.: Learning Shape Completion from Bounding Boxes with CAD Shape Priors, *a Master Thesis at the RWTH Aachen University*, 2017.

[14] Stutz, D. and Geiger, A.: Learning 3D Shape Completion from Laser Scan Data with Weak Supervision , *in IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[15] Theoharis, T.; Papaioannou, G.; Karabassi, E.: The magic of the Z-Buffer: A Survey, *Published in WSCG*, 2001.

[16] Tripiana, C.:GPU Voxelization, *a Master Thesis at the Polytechnic University of Catalonia*, 2009.

[17] Vallado, D. A.: Fundamentals of Astrodynamics and Applications, 4th ed., 2013.

[18] Whitted, T.: An improved illumination model for shaded display, *in Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, 1979.

[19] Zhou, K.; Qiming, H.; Wang, R.; Guo, B.: Real-Time KD-Tree Construction on Graphics Hardware , *in (journal) ACM Transactions on Graphics (TOG) Volume 27 Issue 5, Article No. 126*, `https://doi.org/10.1145/1409060.1409079` 2008.

[20] Zhukov S.; Iones A.; Kronin G.: An ambient light illumination model, *in: Drettakis G., Max N. (eds) Rendering Techniques '98. Eurographics. Springer, Vienna*, 1998.

Appendix A: Usability Questionnaires

Usability Test

The aim of this project is to compute the Solar Radiation Pressure acceleration in an efficient way. In order to do so, we have developed an application where we can apply several models and techniques to compute this acceleration.

The models that can be used in this application are the classical CannonBall model, the N-Plate model and a Finite Element approach using RayCast and RayTracing techniques.

Once you have selected one of the methods, you can choose whether to visualize the satellite and see the directions of the SRP forces, or to visualize the graphics of the SRP force as a function of the satellites azimuth and elevation (you can also download this data into a txt file).

TASK1

1. Load the information of the satellite. In particular, load the files: **cube0.obj** and **cube0.mtl** from the folder: MasterThesis/TFM/RayTracingSRP/resources/models/.

2. Select the method that uses the GPU and is the least accurate.

3. Go to **visualize graphics** and in the advanced settings choose to visualize the **SRP forces**. Now visualize the graphics.

4. Download the information of the graphics in a txt file.

TASK2

1. Load the information of the satellite. In particular, load the files: **GPS03.obj** and **GPS.mtl** from the folder: MasterThesis/TFM/RayTracingSRP/resources/models/.

2. Select the method "**Find Triangles (CPU)".**

3. Change its parameters, so the size of the grid is **64x64**.

4. Visualize the **satellite.**

5. Set the initial rotation of the satellite to the angles: **X: 20º, Y:50º, Z: -180º**.

6. Visualize the forces by moving the **slider** of the **Z** axis.

(Optional) TASK3: Explore the methods and compare some of them.

Figure 29: Introduction and description of the tasks of the usability test.

Questionnaire (after usability test)

1. What is your overall impression of the application?

2. Is there any method where the **explanation** provided is not enough? Rate the methods:

	Very poor	Poor	OK	Good	Very good
Cannon-Ball (CPU)	☐	☐	☐	☐	☐
NPlate (CPU)	☐	☐	☐	☐	☐
Find Triangles(CPU)	☐	☐	☐	☐	☐
RayCast (CPU)	☐	☐	☐	☐	☐
RayTrace (CPU) -Single Scattering	☐	☐	☐	☐	☐
RayTrace (CPU) -Multiple Scattering	☐	☐	☐	☐	☐
RayCast (GPU)	☐	☐	☐	☐	☐
RayTrace (GPU) -Single Scattering	☐	☐	☐	☐	☐
RayTrace (GPU) -Multiple Scattering	☐	☐		☐	☐

Figure 30: First page of the questionnaire given to the users after the completion of the tasks.

3. Is there any parameter whose purpose was difficult to understand or why it is needed?

4. Was it easy to find how to download the data from the graphics into a txt file?

5. Did you understand how to rotate the satellite?

6. What about the visualization of the directions of the SRP forces? Could you identify from the colours which SRP forces have smaller or bigger magnitudes?

Figure 31: Second and final page of the questionnaire.