











RESEARCH ARTICLE

SISPO: Space Imaging Simulator for Proximity Operations

Mihkel Pajusalu¹ , Iaroslav Iakubivskiy¹ *, Gabriel Jörg Schwarzkopf² , Olli Knuuttilla² , Timo Väisänen² , Maximilian Bühner^{2,3} , Mario F. Palos² , Hans Teras^{1‡} , Guillaume Le Bonhomme^{1‡} , Jaan Praks^{2‡} , Andris Slavinskis^{1‡} 

1 Space Technology Department, Tartu Observatory, University of Tartu, Tõravere, Estonia, **2** Department of Electronics and Nanoengineering, School of Electrical Engineering, Aalto University, Espoo, Finland, **3** School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, Bedfordshire, United Kingdom

 These authors contributed equally to this work.

‡ HT, GLB, JP and AS also contributed equally to this work.

* iaroslav.iakubivskiy@ut.ee



OPEN ACCESS

Citation: Pajusalu M, Iakubivskiy I, Schwarzkopf GJ, Knuuttilla O, Väisänen T, Bühner M, et al. (2022) SISPO: Space Imaging Simulator for Proximity Operations. PLoS ONE 17(3): e0263882. <https://doi.org/10.1371/journal.pone.0263882>

Editor: Antonio Agudo, Institut de Robotica i Informatica Industrial, SPAIN

Received: May 12, 2021

Accepted: January 28, 2022

Published: March 4, 2022

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: <https://doi.org/10.1371/journal.pone.0263882>

Copyright: © 2022 Pajusalu et al. This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All SISPO algorithm files are available from the GitHub (<https://github.com/SISPO-developers>).

Funding: This work was funded by the ESA Contract No. 4000131003/20/NL/IB/g with the

Abstract

This paper describes the architecture and demonstrates the capabilities of a newly developed, physically-based imaging simulator environment called SISPO, developed for small solar system body fly-by and terrestrial planet surface mission simulations. The image simulator utilises the open-source 3-D visualisation system Blender and its Cycles rendering engine, which supports physically based rendering capabilities and procedural micropolygon displacement texture generation. The simulator concentrates on realistic surface rendering and has supplementary models to produce realistic dust- and gas-environment optical models for comets and active asteroids. The framework also includes tools to simulate the most common image aberrations, such as tangential and sagittal astigmatism, internal and external comatic aberration, and simple geometric distortions. The model framework's primary objective is to support small-body space mission design by allowing better simulations for characterisation of imaging instrument performance, assisting mission planning, and developing computer-vision algorithms. SISPO allows the simulation of trajectories, light parameters and camera's intrinsic parameters.

Introduction

A versatile image-simulation environment is required in order to design advanced deep-space missions, to simulate large sets of mission scenarios in parallel, and to develop and validate algorithms for semi-autonomous operations, visual navigation, localisation and image processing. This is especially true in the case of Small Solar System Body (SSSB) mission scenarios, where the mission has to be designed with either very limited information about the target (i.e., precise size, shape, exact composition and activity) or the targets can remain a near-complete mystery before their close encounter (i.e., as in the case of interstellar objects [1, 2]). Some publicly known cosmic-synthetic-image generators for space missions are available, and they are briefly described in the next paragraph.

University of Tartu ("Comet Interceptor (EE-1): OPIC Engineering Model Development", PI is MP; the salary was paid to: MP, II, GLB, HT, and hardware acquisition), the Archimedes Foundation (<https://archimedes.ee>, UT ASTRA project 2014–2020.4.01.16-0029 KOMEET "Benefits for Estonian Society from Space Research and Application" in the form of travel supports), the Eesti Teadusagentuur (EE) (MOBTP151 and PUTJD601, awarded to MP), and the base funding of Tartu Observatory (registration number 74001073). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Abbreviations: **3-D**, three-dimensional; **BRDF**, Bidirectional Reflectance Distribution Function; **ESA**, European Space Agency; **JAXA**, Japan Aerospace Exploration Agency; **MAT**, Multi-Asteroid Touring; **OASIS**, Optical Aberrations for Still Images Simulator; **OpenGL**, Open Graphics Library; **OpenMVG**, Open Multiple View Geometry; **OpenMVS**, Open Multiple View Stereo Reconstruction; **OPIC**, Optical Periscopic Imager for Comets; **PANGU**, Planet and Asteroid Natural Scene Generation Utility; **PSF**, Point Spread Function; **RGB**, Red, Green and Blue; **SfM**, Structure from Motion; **SISPO**, Space Imaging Simulator for Proximity Operations; **SPICE**, Spacecraft Planet Instrument Camera-matrix Events; **SSSB**, Small Solar System Body.

Airbus Defence and Space has developed SurRender, which renders realistic images with a high level of representativeness for space scenes [3]. It uses ray tracing to simulate views of scenes composed of planets, satellites, asteroids and stars, taking into account the illumination conditions and the characteristics of the imaging camera through a user-defined Point Spread Function (PSF). The textures are accessed in a large virtual file, or procedural texture generation can be used. SurRender uses the different models for Bidirectional Reflectance Distribution Function (BRDF), for example Lambertian [4] or [5–7] for the Moon and asteroids, [8] for the Jovian moons. The University of Dundee, UK has developed the Planet and Asteroid Natural Scene Generation Utility (PANGU), which generates realistic, high-quality, synthetic images of planets and asteroids using a custom graphics-processing-unit-based renderer, which includes a parameterisable camera model [9]; it also has a graphical user interface, which makes it more intuitive to use. PANGU implements a Spacecraft Planet Instrument Camera-matrix Events (SPICE) interface, which provides historical and future ephemerides of the Solar System and selected spacecraft. The standard Lambertian diffuse reflectivity model is included as well as Hapke, Oren–Nayar, Blinn–Phong and Cook–Torrance BRDFs. NASA's Navigation and Ancillary Information Facility developed the internationally recognised Spacecraft Planet Instrument Camera-matrix Events (SPICE) tool, which provides the fundamental observation geometry needed to perform photogrammetry, map making and other kinds of planetary science data analysis [10]. It is a numerical tool that provides position and orientation ephemerides of spacecraft and target bodies (including their size and shape), instrument-mounting alignment and field-of-view geometry, reference frame specifications, and underlying time-system conversions; however, it does not have surface-rendering capabilities, and it is limited to shape rendering by implementing the digital shape kernel system (tessellated plate data and digital elevation models). SPICE has the three-dimensional (3-D) visualisation application Cosmographia [11], and has been recently implemented in the toolkit with rendering capabilities for spacecraft orbit visualisation and depictions of observations by probe instruments in Blender [12]. Hapke model also has been used with Blender previously [13].

The comparison between various simulators that are capable of SSSB synthetic image generation is summarised in Fig 1 (camera, orientation and exact light parameters may differ between simulator set-ups). The 25143 Itokawa asteroid model by the [14] was used.

The Space Imaging Simulator for Proximity Operations (SISPO) has been developed to provide a full pipeline from simulated imagery to final data products (e.g., 3-D models); to include photorealistic, physically based rendering; to support automatic surface generation with **procedural displacement textures**; to allow the implementation of spacecraft, instrument and environmental models (e.g., imaging distortions, gas and dust environment, attitude dynamics); and to avoid legacy software. SISPO uses Cycles rendering with the Blender software package, which allows for programming procedures in Python [15]. The preliminary results of 3-D reconstruction and localisation using SISPO, without providing details for simulated imagery and near environment (i.e., gas and dust), were published by [16]. SISPO works on large scales and could simulate a variety of objects at the Solar System scale. It could also be used to generate realistic videos from individual frames, either for visualisation purposes or public outreach.

This article demonstrates the synthetic image simulation capabilities of SISPO, and 3-D reconstruction as a case study showing how such images can be utilised. It also discusses SISPO application to actual space missions, architecture, rendering system and supplementary models.

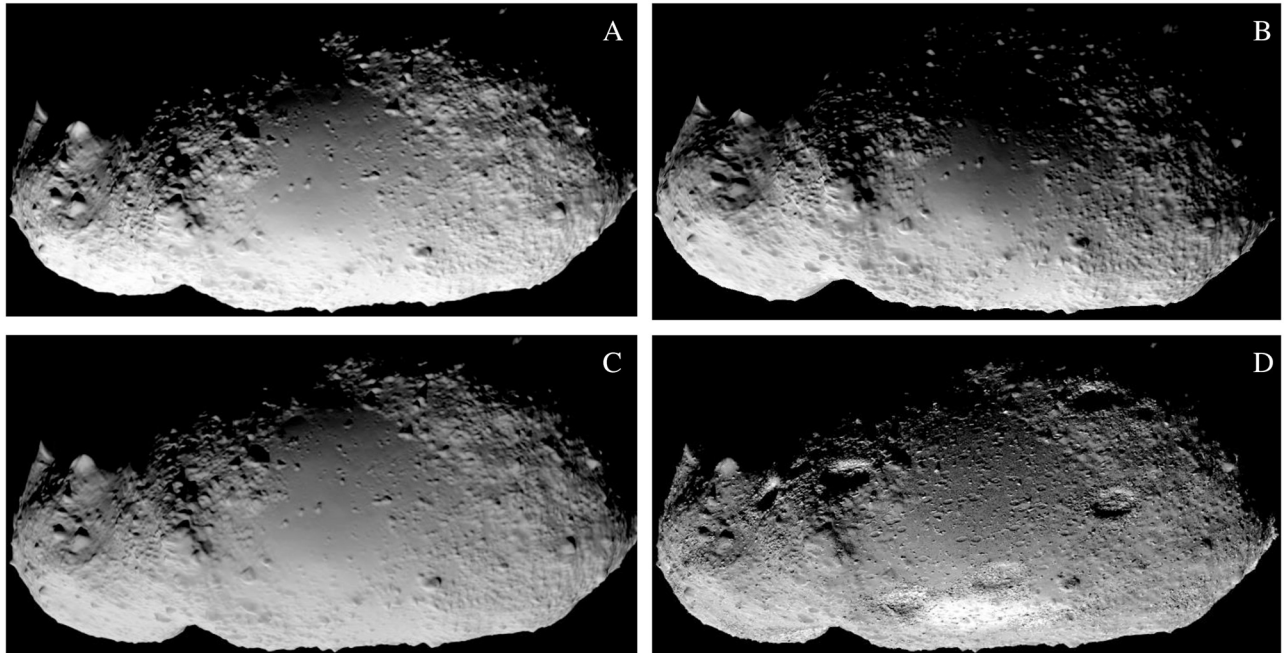


Fig 1. The comparison of available simulators for space-scene image rendering. The rendered example for each simulator is asteroid 25143 Itokawa. (A) SurRender image generator by Airbus; it uses backward ray tracing or image generation with Open Graphics Library (OpenGL). (B) PANGU image generator by University of Dundee, UK and ESA; it uses fractal terrain generation using OpenGL. (C) SISPO by the University of Tartu, Estonia and Aalto University, Finland; it uses Blender Cycles physically based path tracer (the same model as above with a simple diffuse shader); (D) SISPO with Blender Cycles physically based path tracer (with procedural displacement with new surface features and reflectance textures for extra detail).

<https://doi.org/10.1371/journal.pone.0263882.g001>

Application to space mission designs

An increasing number of missions and mission concepts for studying SSSBs require advanced and autonomous operations. For instance, Hera's Asteroid Prospection Explorer daughter-craft (previously called ASPECT) has to navigate autonomously within the proximity of Didymos and its natural satellite [17]; The M-ARGO might be the first-ever standalone nanospacecraft to rendezvous with an asteroid [18]; CASTAway is a mission concept to fly by and explore 10–20 main-belt asteroids and use optical navigation in their proximity [19]; a proposed mission, Castalia, to the main-belt comet 133P/Elst–Pizarro would reveal much that is currently unknown about it and detect water *in situ* in the main belt [20]; Caroline was proposed in 2010 to also fly by a main-belt comet [21]; Titius–Bode is a mission concept to investigate a sequence of asteroids by orbiting its targets for about six months and dispose the Bode lander on the surface [22]; The Martian moons Deimos and Phobos were also targeted by DePhine [23] and Phobos Sample Return [24]. Development and planning of these missions and similar ones require a sophisticated and realistic simulator.

The initial SISPO package was developed to assist in the Multi-Asteroid Touring (MAT) mission, where a fleet of nanospacecraft, propelled by electric sails, flies by a large number of main-belt asteroids [25, 26]. The MAT concept cannot rely on typical deep-space-network-based solutions to operate and navigate the fleet; it requires most of the operations to be performed autonomously. The case study of MAT performing a Didymos fly-by has been evaluated using SISPO; the set of images was generated for the reconstruction and basic localisation for fly-by distances of 33–300 km [16]. The MAT mission was proposed to the European Space

Agency (ESA) announcement of opportunity for “New Science Ideas” [27] and it was not chosen despite reaching the final three [28].

Currently, the SISPO simulator is actively used for Optical Periscopic Imager for Comets (OPIC) development [29], which will be hosted on one of three spacecraft making up the Comet Interceptor mission (<https://www.cometinterceptor.space>). Comet Interceptor is ESA’s first F-class mission (in cooperation with Japan Aerospace Exploration Agency (JAXA)) to fly by either a dynamically new comet approaching the Sun for the first time from the Öpik—Oort cloud, an interstellar object, or a long-period comet as a backup target [30]. OPIC will use automatic image capturing—algorithms will be developed and tested using photorealistic 3-D renderings and a reconstruction pipeline of SISPO using a set of possible encounter velocities and geometries, as well as cometary and camera properties. From a scientific point of view, the simulator will also be used to develop image-prioritisation algorithms, which are required due to the limited data budget, short fly-by timeline and the possibility of the probe being damaged by high-velocity dust impact. The EnVisS coma mapper [31] of the Comet Interceptor mission also uses SISPO for algorithm development.

General architecture

The general structure of SISPO comprises the following parts:

1. Core features:
 - a. Image rendering using Blender Cycles (see Subsection: Rendering in Blender Cycles) or OpenGL (see Subsection: Lightweight OpenGL-based rendering);
 - b. Simulation of on-board image processing. Currently the image processing is primarily related to compression; however, inclusion of both cropping and image prioritisation is planned;
 - c. 3-D reconstruction (see Section: Usability of images produced for 3D reconstruction).
2. Additional models:
 - a. Gas and dust environment (see Subsection: Gas and dust);
 - b. Camera distortions (see Subsection: Camera);
 - c. Attitude dynamics in the initial stage.

The core functionalities are split into three subpackages. The first subpackage uses Keplerian orbit data for an SSB and a simplified definition of the encounter geometry so the spacecraft can propagate realistic trajectories using the Orekit library [32] and render an image series of the encounter. The second subpackage provides various algorithms for image compression and decompression. The third subpackage uses images to reconstruct a textured 3-D model using the Structure from Motion (SfM) technique. The three subpackages combined provide a processing pipeline from an initial 3-D model to a reconstructed 3-D model via rendered and compressed images. SISPO is a Python software package that is hosted on a public *GitHub* repository under a *GPL v3.0* licence and is maintained by the authors [33] among other contributors (e.g., authors of this paper). A description of the core functionality and the additional models is provided in Fig 2.

Rendering system

The most crucial part of SISPO is image synthesis. Two separate rendering modes are implemented: Blender Cycles and OpenGL.

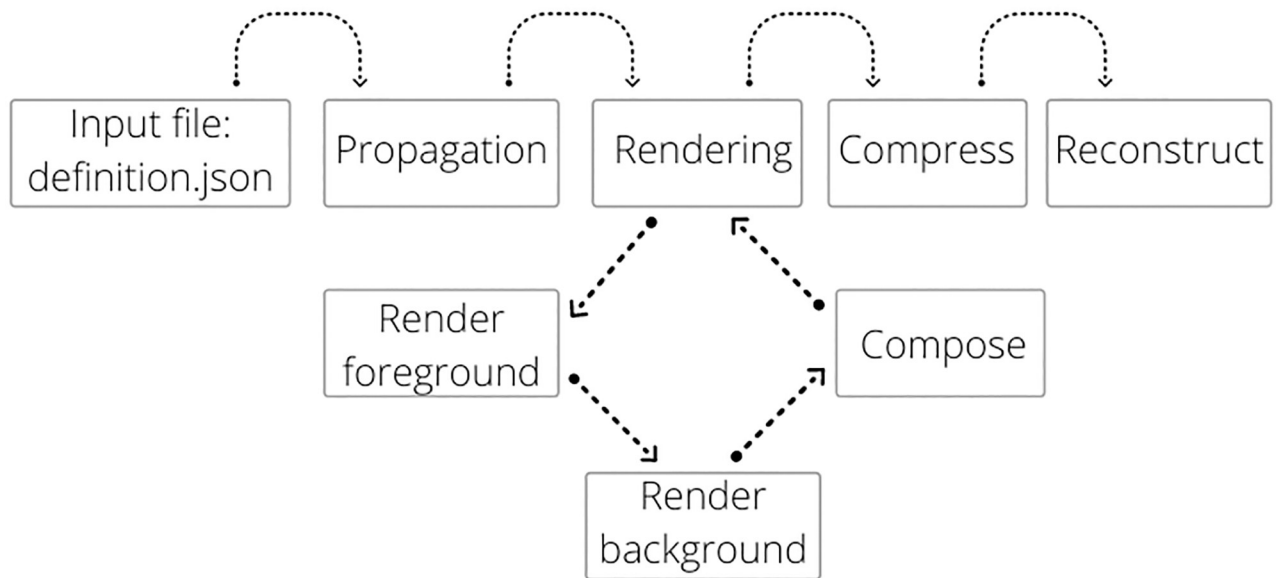


Fig 2. Program flow of SISPO.

<https://doi.org/10.1371/journal.pone.0263882.g002>

Rendering in Blender Cycles

Blender Cycles uses path tracing, which is a type of ray tracing. In classic ray tracing [34], each camera's pixel shoots one or multiple rays, which interact with a surface and then encounters light sources directly or interacts with other surfaces before reaching the light. For realistic reproduction, various optimisations can be used. The ray-surface and ray-volume interactions can be modelled by shaders, which model or approximate the interacting objects' properties.

The Cycles rendering engine supports various shaders, the most relevant of those for this paper being: (i) diffuse bidirectional scattering distribution function, which provides access to Lambertian and Oren–Nayar shaders based on surface roughness; (ii) emission shader that allows surfaces or volumes to emit light; (iii) subsurface scattering that supports cubic, Gaussian and Christensen–Burley models; (iii) glossy shader that supports Sharp, Beckmann, GGX, Asikhmin–Shirley and multi-scatter GGX models; (iv) volume scattering shader that allows simulating light scattering in volumes [15]. These shaders can be combined with both procedurally generated or pre-existing texture maps to change their parameters and mix various shaders on and within the scene models. Cycles also supports Open Shading Language, which allows the use of arbitrarily defined shaders.

Path tracing is, in some way, an improvement on ray tracing; it produces multiple rays from the same pixel in random directions, then each ray keeps bouncing (without producing new rays) until it reaches the light source or user-defined bounce limit. In Blender Cycles, one can limit the following bounce limits (ideally it should be infinite; however, typically, a limited amount is sufficient): total, diffuse, glossy, transparency, transmission and volume scattering. Then, the amount of light per pixel is calculated for each ray along with surface colour properties; afterwards, the value is averaged and assigned to that specific pixel. Moreover, Blender has a branched path tracer, which can be used for volumetric scattering (see Subsection: Case 2: volumetric particle effects). The branched path tracer is similar to path tracing, but at the first ray interaction it will split the path for different surface components, and for shading, it will take all lighting parameters into account instead of just one [15]. The branched path tracer can also be useful for solving light-related problems such as caustics.

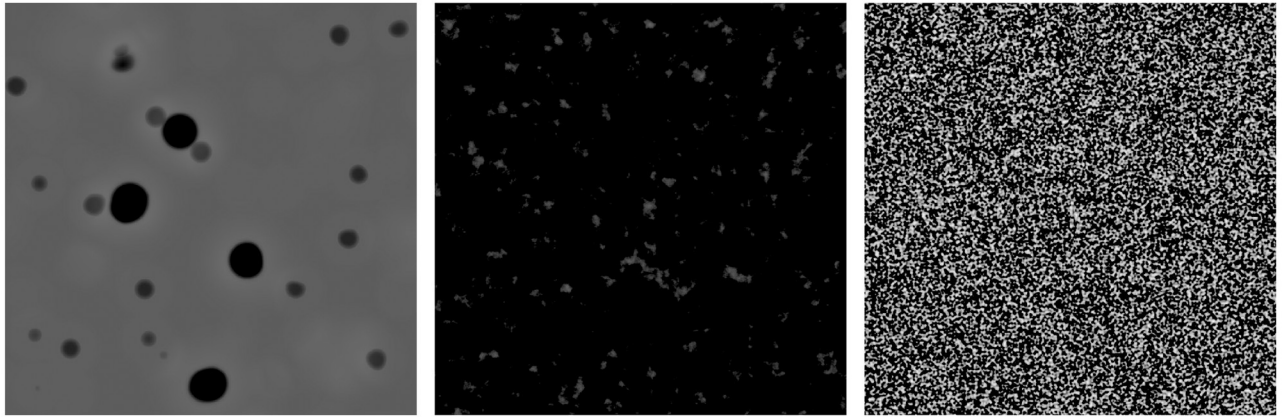


Fig 3. Procedurally generated height maps. For craters (left), rocks (centre) and sand (right).

<https://doi.org/10.1371/journal.pone.0263882.g003>

All the surface features can be generated procedurally (e.g., by applying mathematical equations instead of externally-captured image texture) inside the Blender software [35]. The SISPO surface model includes sand flats, rock formations and craters. The size of these features, their number and their distribution can be adjusted by modifying the corresponding parameters.

The elevation details are simulated by using height maps for craters, rocks and sand inside the rendering engine (Fig 3). These consist of textures converted into surface displacement distances and are used to displace parts of the model mesh and provide surface normal changes.

Height maps can be mixed together to simulate different types of terrestrial bodies (see Section: Simulations of realistic asteroid imagery). Each map's weight in the mix can be modified, or even obliterated, and they can be made to affect only specific areas of the mesh by using masks. The shader adds a texture whose albedo corresponds to the albedo measured in real asteroids in addition to the procedural elevation. The final procedural shader example is shown in Fig 4.

Lightweight OpenGL-based rendering

There is an option to use a custom OpenGL-based renderer instead of Cycles and Blender. It was added because it is desirable to generate images fast in certain situations even though some fidelity (e.g., softer shadows or procedurally generated details) might be lost. For instance, during the development of image-processing algorithms, it is useful to generate large datasets for training data in a relatively short time. Validation data can be generated with Cycles to give better confidence in the characterised algorithm performance. Fast image generation also enables a closed-loop simulation of a guidance, navigation and control system that depends on the navigation camera input.

Three different BRDFs have been implemented as OpenGL shaders: Lambertian [4], Lunar-Lambertian [36] and Hapke [37]. Textures are supported and correspond to the single scattering albedo of the corresponding shape model region.

Efficient shadowing is achieved by shadow mapping [38]. Shadow mapping works by first rendering the scene orthographically from the direction of the Sun. The resulting depth buffer contents are imported to the actual rendering pass as a texture. The object vertices are again projected orthographically towards the Sun; the corresponding fragment is considered to be in the shadow if the depth of that projection is greater than what is in the shadow texture.

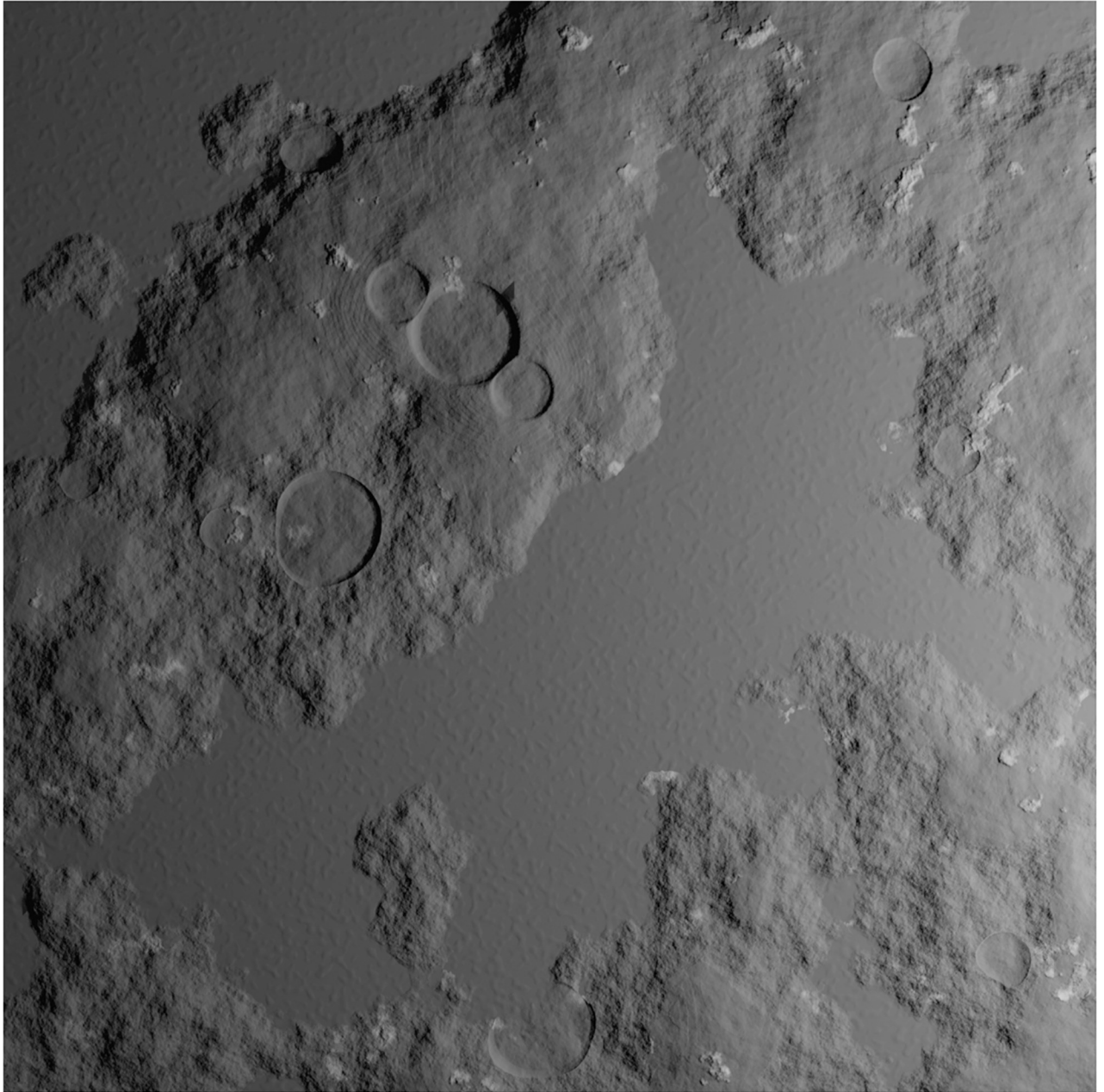


Fig 4. Final procedural shader applied to a subdivided plane.

<https://doi.org/10.1371/journal.pone.0263882.g004>

The output of the OpenGL render is a floating-point value giving the irradiance [$\text{W} \cdot \text{m}^{-2}$] incident on each theoretical pixel. This “irradiance image” can be further passed to the default camera model used by SISPO, which then applies appropriate imaging effects to get the final synthetic image. There is currently no procedural shape or texture generation due to the simplified nature of the OpenGL renderer. Dust and gas coma rendering is not fully integrated with the OpenGL renderer. Instead, emission-based rendering of a previously generated coma is done separately in Python after OpenGL rendering (see Subsection: Case 2: volumetric

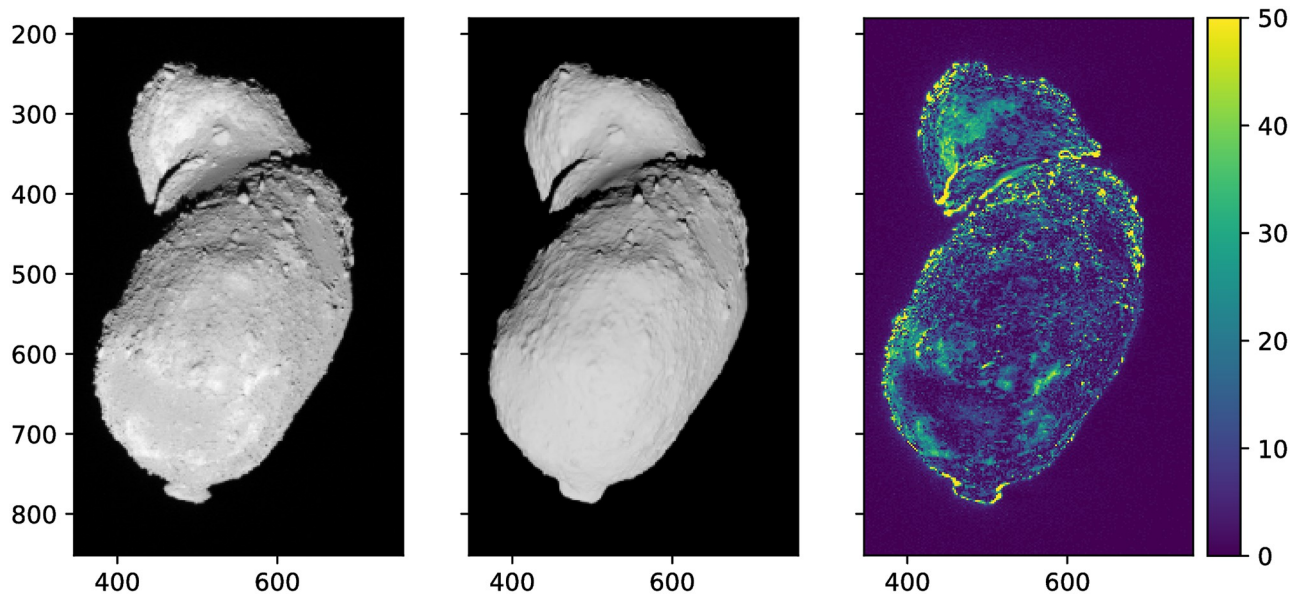


Fig 5. The pixel-by-pixel comparison of the original AMICA image on the left and generated OpenGL image in the middle. The scale indicates percent error. The AMICA image is published under Public Domain 1.0.

<https://doi.org/10.1371/journal.pone.0263882.g005>

particle effects), and it does not take into account shadows cast by objects in the scene. One fundamental limitation to OpenGL-based rendering is that it does not support extended light sources. Thus, for instance, earthshine in low Earth orbit—or in the case of a binary asteroid, the reflected light from the primary body to the secondary body—cannot be reasonably modelled. As the light from reflecting surfaces is not considered, the shadow of rocks, cliffs and other geological formations will be darker than in reality.

The OpenGL rendering engine was used for a preliminary guidance, navigation and control analysis done for the preliminary design review of the now-defunct Asteroid Prospection Explorer project. For this purpose, the rendering engine was interfaced from a Simulink model designed to handle the system dynamics. The same model then forwarded the generated images to the visual navigation algorithms for position estimates. The generation of one image took around 0.7–1.0 seconds. The simulation time for one mission week was around 12 hours on a laptop with an Intel i7–7700HQ (2.8 GHz) processor, with most of the processing time spent on executing the visual navigation algorithms.

Fig 5 demonstrates the comparison between the image taken by the AMICA instrument of the Hayabusa spacecraft and rendered image using OpenGL. The main rendering discrepancies are induced by the inaccuracy of the 3D model [14] and local variations of albedo and roughness.

Supplementary SISPO models

Gas and dust

In the inner Solar System, the solar radiation heats nuclei of comets, which causes them to release gas together with dust. This gas and dust surround a comet forming a coma, which is blown by the solar wind resulting in ion and dust tails that can extend over 100 million kilometres and might be visible from Earth. Sometimes the comet releases strong outbursts of these gasses and dust to produce distinctive coma-derived features called jets [39]. The gas and

dust environment is visible to the camera instrumentation, and hence, in order to provide realistic space-scene simulation, the visible effects should be included in modelling.

The dust and gas environment causes noise that can affect the 3-D shape reconstruction. SISPO offers models for comae, jets and tails with various details. The problem can be divided into two parts: modelling the environment and rendering it. The modelling parts can contain, for instance, the description of the environment, such as jets represented as geometric cones from the surface of the comet, mathematical models from [40], or gas and dust simulations such as those from [41, 42].

The rendering problem comes down to the level of desired details and accuracy, but on the other hand, to reasonable computing time. In reality, the volume is a mixture of gas and dust, which are composed of different particle sizes with different densities, and this produces various scattering characteristics [43]. Realistic and accurate representation of the effects mentioned above in the rendering pipeline, which is not specialised in volume scattering, is a challenge; a visually realistic approximation should be sought. In the current version of SISPO, volume-scattering effects from the coma and jets are computed using a simple volume-scattering shader from Cycle that uses the voxel presentation of the comet's surroundings as an input for the density.

Camera

The Optical Aberrations for Still Images Simulator (OASIS) provides simulation tools for optical aberrations that are usually not implemented in popular 3-D software (e.g., Blender). Since it uses two-dimensional images as input, motion blur effects, which require spatial awareness of a scene, are not modelled. Optical Aberrations for Still Images Simulator (OASIS) is used in a complementary manner with SISPO to further enhance rendered two-dimensional output images.

Tangential and sagittal astigmatism, as well as an internal and external comatic aberration, are modelled with distinct PSFs, which vary with the field height and orientation of the sensor centre—image point vector. By default, aberration intensity increases with the field height. However, a custom lens file can be provided to model any desired lens array.

Lateral chromatic aberration is modelled by rescaling individual colour channels and simulating wavelength-dependent refraction of light rays. While wavelengths are continuous in the real world, the digital Red, Green and Blue (RGB) triplet only distinguishes between three discrete primitive colours, which introduces sharp edges at the boundaries of colour separation. These can be avoided by blending chromatic aberration with tangential astigmatism (more information with an example can be found in Section 5.7 in [44]).

Dark-current noise is currently drawn from a folded normal distribution with a zero mean and user-defined standard deviation. Readout noise is modelled by the addition and subtraction of random values at the subpixel level, governed by a Gaussian custom standard deviation process. The projection of light rays with random origin positions generates realistic shot noise and follows the user-defined average sample size per pixel.

Lens distortion is simulated with a sophisticated Brown–Conrady model [45], that corrects for both radial and tangential distortion. It is implemented in the computer-vision library OpenCV [46] and supports up to six radial distortion coefficients— k_1 to k_6 —and two tangential distortion coefficients— p_1 and p_2 .

Monochrome sensors are modelled by averaging an RGB colour triplet of a virtual light ray. Additionally, a weighted-average model can be selected that reflects the indeed perceived luminosity. The generation of dark current and readout noise is adjusted accordingly to maintain a specified standard deviation on monochromatic detectors. The simulation of sagittal

astigmatism, coma, chromatic aberration, shot noise and readout noise is demonstrated in more detail in [44] (Section 5.7).

Currently, OASIS is limited to simulating one type of PSF at a time, as a realistic convolution of multiple PSFs is not yet provided, with the exception of small aberrations. Also, the generation of dark noise, which should follow a Poisson distribution, is subject to change once physical units are implemented for photon flux and for the conversion between light-ray energy and digital-sensor value.

Orbital simulation

The trajectory simulation within SISPO is handled by the Orekit library [32]. A simple Keplerian orbit propagator is used to propagate both the SSSB and the spacecraft. Additionally, it is possible to rotate the SSSB around a single axis at a constant spin rate.

The implemented Orekit Python bindings run a virtual machine to execute the underlying Java code. During propagation, Orekit determines state information of the SSSB and the spacecraft for each sample along the trajectory. The state information includes the date, position and the rotation angles of the SSSB.

In SISPO, the spacecraft trajectory is normally defined by its Keplerian elements, but the user does not explicitly enter these elements. Instead, the elements are calculated from the target body's Keplerian elements and the expected encounter geometry relative to the SSSB at the closest approach. The parameters presented in Table 1 are used to calculate the state vector of the spacecraft at the encounter, which defines the spacecraft trajectory. The *sssb_state* is calculated based on the SSSB input data and the encounter data.

The propagation process is defined by the duration of a fly-by, the number of frames to be rendered, *timesampler* mode and a *slowmotion* factor as presented in Table 2. The *timesampler* mode determines whether the steps are distributed linearly in time (mode 1, default) or whether an exponential model (mode 2) is used, which increases the number of frames around the encounter. The number of additional samples can be controlled with the *slowmotion* factor.

Table 1. Parameters defining the encounter state of the spacecraft in SISPO. The first five parameters are required as input.

Parameter	Unit	Type	Description
encounter_distance	m	float	Minimum distance between SSSB and spacecraft.
with_terminator	–	bool	Determines whether the terminator is visible at the closest approach.
with_sunnyside	–	bool	Determines whether the spacecraft passes the SSSB on the Sun-facing side or the side facing away from the Sun.
relative_velocity	m s ⁻¹	float	Relative velocity of the spacecraft to the SSSB at the encounter.
encounter_date	–	dict	Date of the closest approach of the spacecraft and SSSB. The type is a Python dictionary with an <i>int</i> for year, month, day, hour, minute, and <i>float</i> for second.
sssb_state	m, m s ⁻¹	tuple	SSSB state vector containing three position and three velocity components at the encounter. The spacecraft encounter state is calculated relative to this state vector. The SSSB state is not required as input since it is calculated based on the SSSB trajectory and encounter date.

<https://doi.org/10.1371/journal.pone.0263882.t001>

Table 2. Input parameters that define the propagation step in SISPO.

Parameter	Unit	Type	Description
duration	s	float	Total length of the simulation. The encounter date is reached after half of the duration.
frame	–	int	Number of frames (samples) taken during the encounter.
timesampler_mode	–	int	Mode 1 is linear and mode 2 is exponential sampling.
slowmotion_factor	–	float	Determines how many more state samples are taken around the encounter. Only applies if <i>timesampler_mode</i> is exponential.

<https://doi.org/10.1371/journal.pone.0263882.t002>

Mode 2 is especially helpful when simulating a long fly-by, since, when the spacecraft is far from the SSSB nucleus, only minor changes are visible in rendered images.

Attitude dynamics

Along with the orbital simulation, a simplified attitude dynamics portion is already built into the Orekit framework, also accessible via the SISPO library. Attitude information in Orekit is handled essentially as another frame transformation. It contains the rotation from the reference frame to the satellite frame, and then the angular velocity (i.e., spin) and angular acceleration (i.e., rotation acceleration) of the spacecraft in its frame. This makes Orekit's *spacecraft_state* a great construct to hold attitude-related and orbital data, which can then successfully be propagated in time. The Orekit library also allows SISPO to build user-defined "attitude law". This attitude law can be selected from several pre-existing and common attitude modes (e.g., pointing, spin stabilised) or recompiled entirely in a way necessary for the mission simulation.

It is, however, essential to note that the Orekit library mainly focuses on orbital mechanics and propagation. The attitude model, and any attitude laws it follows, presumes the existence of a "perfect attitude control system" without necessarily considering the physical limitations or perturbations of the satellite attitude by external forces. The Orekit library lacks the necessary constructs to convey and utilise the moments of inertia of the satellite. Since this information is crucial for the inclusion of external forces and their effect on the spacecraft's attitude or other celestial bodies, this will be implemented as a secondary layer into the current framework; this is explained in Section: Discussion and future work.

Simulations of realistic asteroid imagery

This section demonstrates five different use cases of the SISPO software.

Case 1: Asteroid 25143 Itokawa

[Fig 6](#) demonstrates the comparison between the image taken by the AMICA instrument of the Hayabusa spacecraft and rendered image using Cycles on the plain model using a simple diffuse shader. The main discrepancies in the rendering are due to the inaccuracy of the 3D model [14]. The higher error spots are caused mainly by the local variations of albedo and roughness. The biggest advantage of the procedural texturing, which introduces new surface features, is demonstrated in [Fig 7](#). The fragmented zoomed view in [Fig 8](#) demonstrates how surface features are added procedurally and the level of details that can be achieved in comparison with the plain mesh.

[Fig 7](#) shows a comparison between a plain mesh of the asteroid 25143 Itokawa and a mesh recreation using the procedural shader in Blender. The initial mesh is a 3-D reconstruction of the actual asteroid as described in [47]. The shader then adds procedural surface features on top.

A small area of the model mesh was rendered using OpenGL and Cycles to show the ultimate advantage of procedural texturing. Both renders are shown in [Fig 8](#), and the one with procedural texturing indicates the capability to generate almost arbitrary level of detail. This feature would be required, for example, to train probes that can land on the surface.

Case 2: Volumetric particle effects

Coma and dust jets are produced by particles emanating from the surface, and these are used to produce a volumetric density distribution. For the preliminary proof of concept, the jets

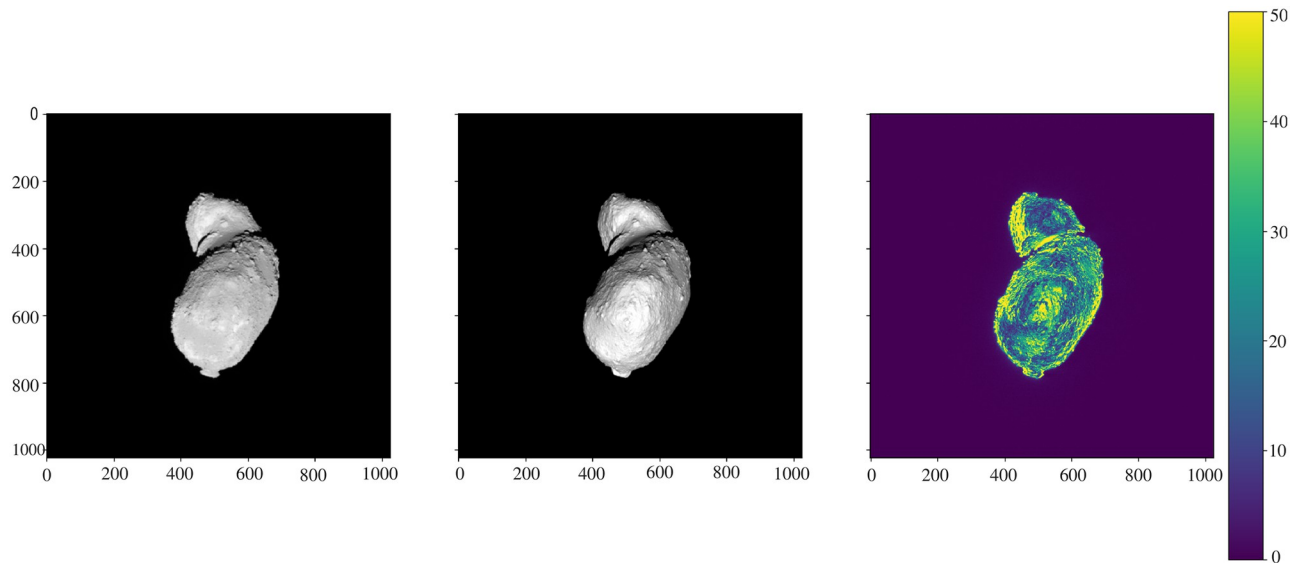


Fig 6. The pixel-by-pixel comparison of the original AMICA image on the left and generated Cycles image in the middle. The scale indicates percent error. The AMICA image is published under Public Domain 1.0.

<https://doi.org/10.1371/journal.pone.0263882.g006>

were modelled using simplified versions of the gas and dust models from [41, 42]. First, the gas source strengths and gas velocities are generated for each face of the mesh from the noise function. The gas source data are then used as input for the gas model [41], which computes the gas field densities and velocities around the comet. This data is then used in a particle simulation following [42] from which the particle position data is further processed into a three-dimensional number-density map around the comet, which is encoded to a single OpenEXR file [48]. Before the render, the number-density map is smoothed with tricubic interpolation [49]. It can then be loaded by Cycles and rendered using either volumetric scattering or emission (volumetric emission, although less accurate, is orders of magnitude faster).

Volumetric particle effects on the comet 67P/C-G. In Fig 9 (the coma might appear differently to the article's viewer depending on the monitor or the print quality), the dust environment capabilities of SISPO are presented by simulating and rendering images of the comet 67P/C-G. This image is then compared to the actual image, and with an OpenGL rendered

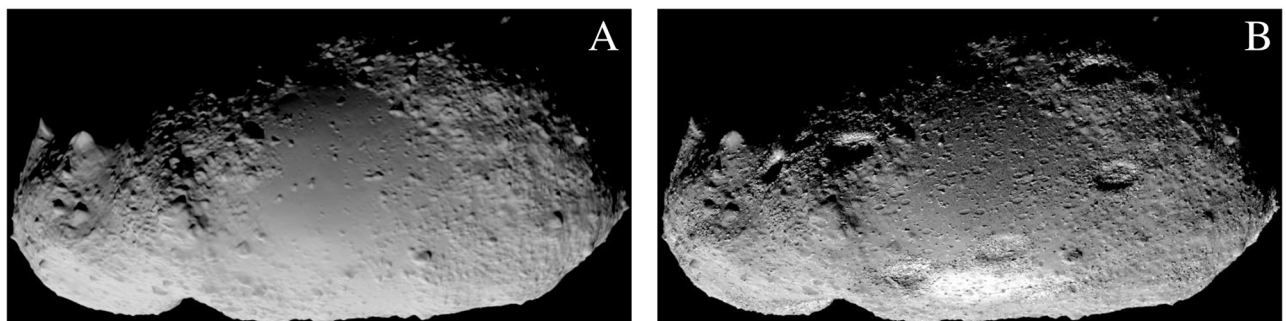


Fig 7. Surface mesh of asteroid 25143 Itokawa. Created by the AMICA imaging team [14] and rendered by authors in Blender Cycles. (A) plain mesh and (B) smooth mesh with procedural shader.

<https://doi.org/10.1371/journal.pone.0263882.g007>

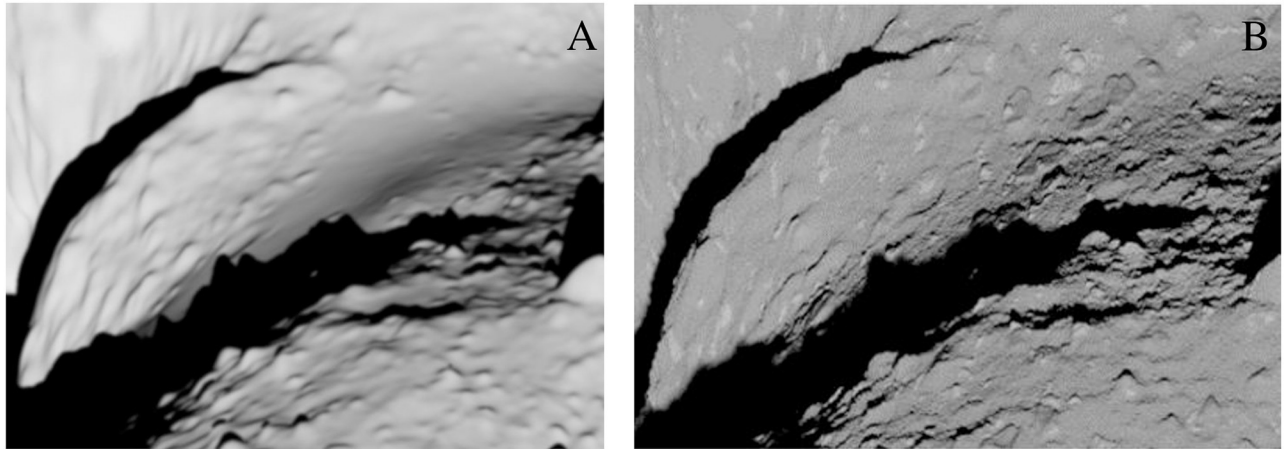


Fig 8. Close view of the 25143 Itokawa surface. Zoomed-in region. (A) OpenGL render and (B) Cycles render.

<https://doi.org/10.1371/journal.pone.0263882.g008>

image. All the shader features (rocks, craters and sand flats) can be used simultaneously for complex SSSBs if needed. The masks that separate the “sand flats” from the rest of the surface features are created with procedural noise, but they could be painted by hand if necessary.

Case 3: Larger bodies

The reproduction of larger terrestrial bodies in SISPO is demonstrated in Fig 10, which is based on the narrow-angle-camera digital terrain model of the Apollo 15 landing site and operations area [50], obtained by Lunar Reconnaissance Orbiter Narrow Angle Camera (http://wms.lroc.asu.edu/lroc/view_rdr_product/NAC_DTM_APOLLO15_M111571816_50CM, accessed 28.01.2021). The original terrain model has a 2 m resolution for elevation maps and 0.5 m resolution for orthographic photos, which is not enough to produce images from the surface that would be useful for navigation testing. This case demonstrates the image scalability produced in SISPO. The lunar rendering was made by implementing a digital

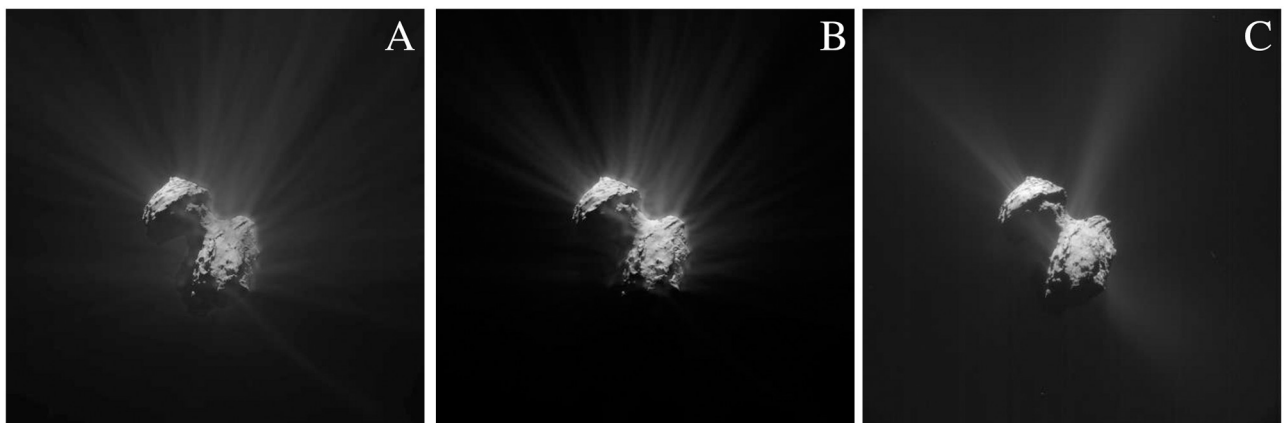


Fig 9. Comet 67P/C-G with coma. (A) OpenGL, (B) Cycles and (C) Real image. The brightness was enhanced at the post-processing stage in order to better visualise coma.

<https://doi.org/10.1371/journal.pone.0263882.g009>

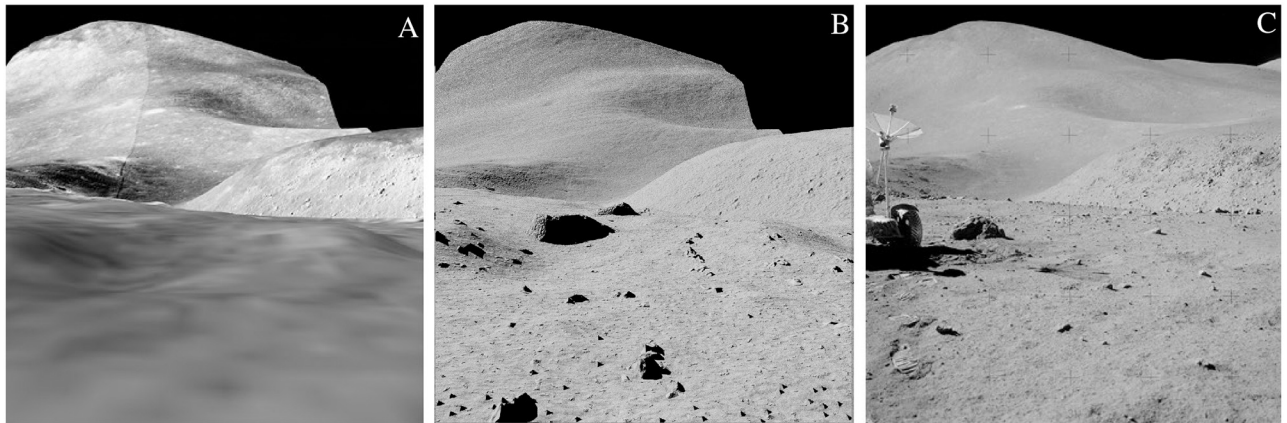


Fig 10. Application of procedural texturing on the scale of a larger airless body. (A) using a digital terrain model and applying orthographic photos as a texture. (B) using a digital terrain model and procedural terrain generation with Blender Cycles on the same digital terrain model. (C) image taken during the Apollo 15 mission from roughly the same location.

<https://doi.org/10.1371/journal.pone.0263882.g010>

terrain model and procedural texturing over the whole $5.1 \times 28.6 \text{ km}^2$ area; the farthest point visible on the render is roughly 11 km away from the camera.

Case 4: Subsurface exploration

Lava caves, the isolated underground environments, exist on Moon [51] and Mars [52]. These caves have been studied by remote sensing and have not been explored by dedicated missions, although some were proposed and developed, such as Moon Diver [53] and rock climber Lemur [54]. Lava tubes have been morphologically related to ones formed on Earth in volcanic rock by a volcanic eruption [55]. SISPO could be utilised for synthesising versatile sets of lava-caves images for (i) developing navigation algorithms and sampling spots detection by image processing (e.g., biological mats or their preserved remains on the geological substrate) or (ii) mapping the caves by photogrammetry for potential human settlements. An example of SISPO cave rendering is demonstrated in Fig 11.

Case 5: Fly-by of a spacecraft

The SISPO environment is suitable for spacecraft fly-by rendering. This can be useful for multi-spacecraft missions, in-orbit servicing and fleets. The Cycles rendering of the spacecraft fly-by is demonstrated in Fig 12. The model uses a *principled bidirectional scattering distribution function* shader including multiple layers to create spacecraft materials. The set of produced images can be used for developing and training formation-flying proximity algorithms demanded by attitude control and orbit determination subsystem.

Usability of images produced for 3D reconstruction

A set of synthetically generated images can be used for photogrammetry-based 3-D surface reconstruction within the SISPO environment. The steps executed in the reconstruction pipeline are described in [16]. The reconstruction uses two libraries:

1. Open Multiple View Geometry (OpenMVG) C++ library [56], which creates a sparse point cloud based on two different SfM techniques:



Fig 11. An example of a cave rendering. Colour, specularity and normal maps were applied. Colour maps are used from the walls of the Hillingdur and Blámi lava caves in Iceland obtained by Iakubivskiy. White spots indicate possible biological mats or secondary mineral precipitation; the red areas simulate iron oxide.

<https://doi.org/10.1371/journal.pone.0263882.g011>

- a. Global SfM [57].
 - b. Incremental SfM [58], which is much more suitable for SISPO applications [16].
2. Open Multiple View Stereo Reconstruction (OpenMVS), which uses input from OpenMVG, creates a dense point cloud, a faceted surface (mesh) or a set of planes [59].

Case 1: SSSB fly-by

A set of 25 images was generated in SISPO and then used to reconstruct the 3-D model using the pipeline. The images were generated using a pinhole camera (i.e., no optical aberrations,



Fig 12. Rendering of a spacecraft fly-by.

<https://doi.org/10.1371/journal.pone.0263882.g012>

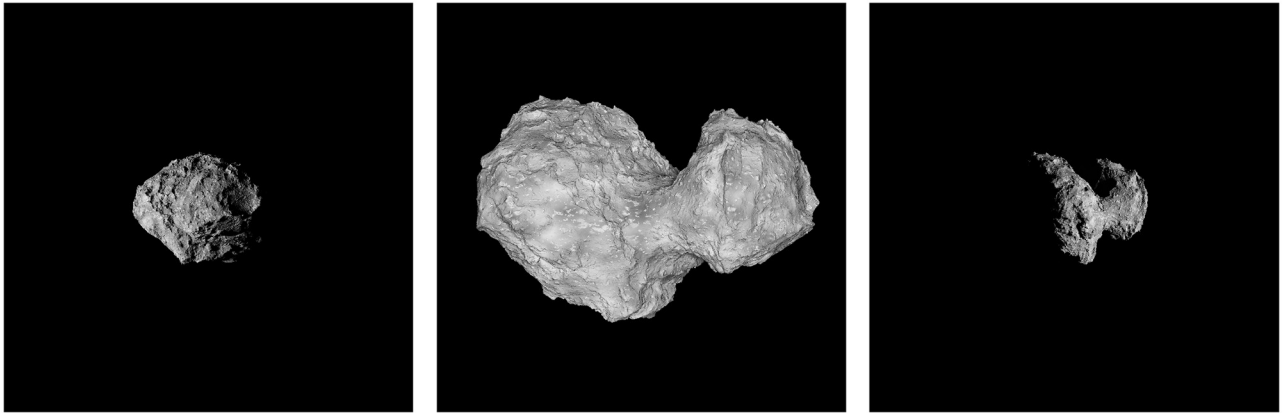


Fig 13. Three out of 25 frames used for the reconstruction. Images rendered in SISPO. Furthest approaching frame, closest approach and furthest leaving frame.

<https://doi.org/10.1371/journal.pone.0263882.g013>

which would decrease the reconstructed accuracy; however, they can be implemented as discussed in Subsection: Camera). The two furthest points and the closest approach are shown in Fig 13.

The result of reconstruction is shown in Fig 14, which visualises vertices and the triangular mesh via the inclusion of normals and applied texture in the reconstructed model.

As a result, two 3-D models are obtained: the input model for SISPO and the reconstructed model from images. Only the visible part of the comet was obtained because of the nature of

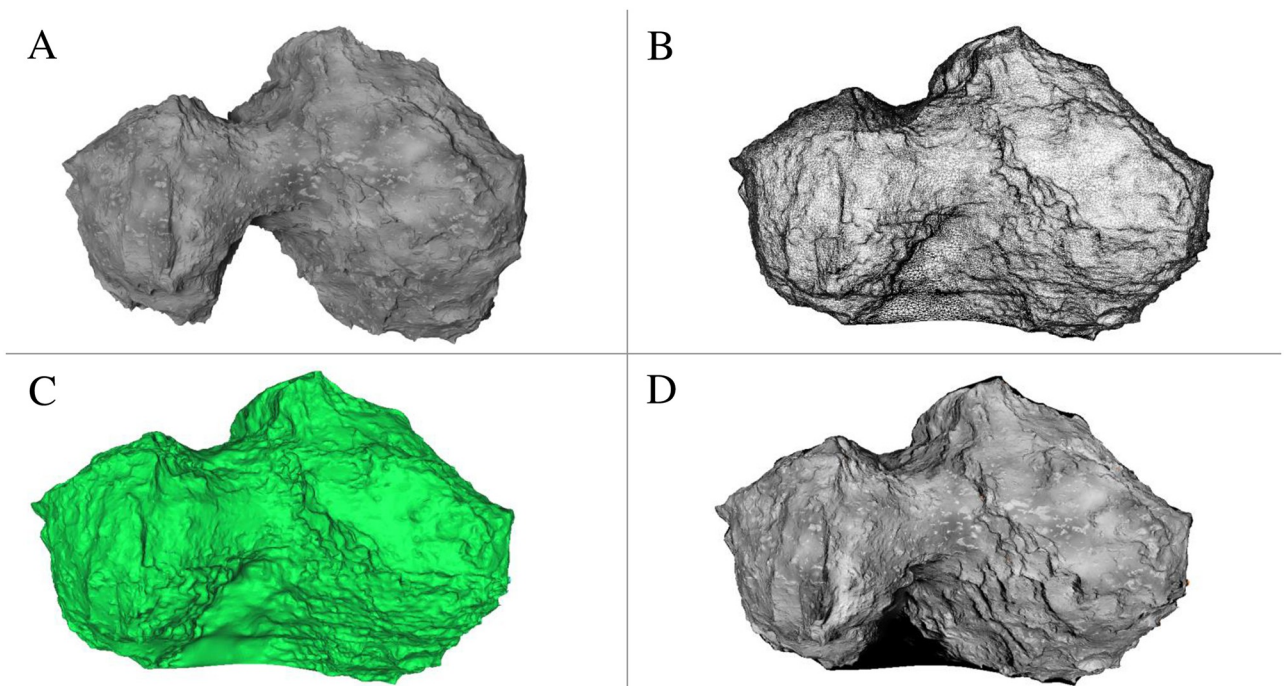


Fig 14. Final reconstruction of the visible part of the target without post-production. (A) Synthetically generated image (input). (B) Vertices of reconstruction. (C) Mesh with normals. (D) Triangular mesh with texture.

<https://doi.org/10.1371/journal.pone.0263882.g014>

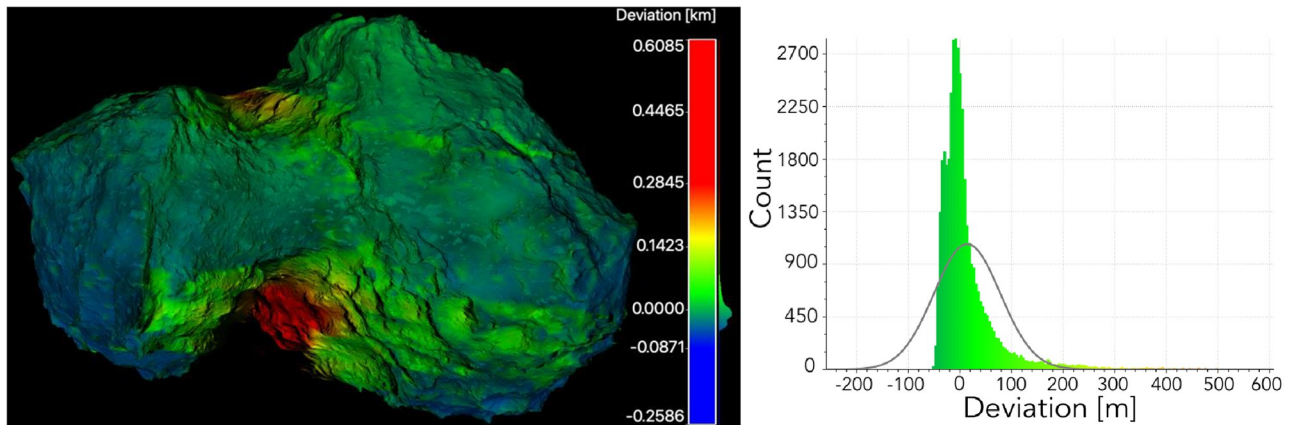


Fig 15. Deviation analysis of the reconstructed partial mesh and input 3-D model for image synthesis. Visualisation of deviation (left); errors and Gaussian distribution (right).

<https://doi.org/10.1371/journal.pone.0263882.g015>

the fly-by (i.e., during the fly-by only one illuminated part of the target is visible). The reconstructed model was compared with the input 3-D model externally using the *CloudCompare* software [60]. Generally, the reconstructed model is close to the input model, except for the edges (e.g., the black area in Fig 15), where the error deviation was very high (up to 600 m). These highly deviated parts could be removed in the point cloud or mesh post-processing, but have not been done in this analysis. The Gaussian distribution mean error is 14.7 m and the standard deviation is 63.9 m. The visualisation of deviation and the error analysis are shown in Fig 15.

Case 2: SSSB orbiting

During SSSB orbiting, it is possible to observe every side of the target because as the target spins, each part will be illuminated at a certain point. Three different input-image views (out of 53 used in total) and a fully reconstructed 3-D model are shown in Fig 16. Fifty-three images were generated from different angles using a pinhole camera.

The input and reconstructed 3-D model were compared using the same method as in the fly-by case. Because of the larger set of observational angles, the deviation is decreased in the orbiting case and reaches 60–89.7 m at some crater spots, but most of the surface is aligned with the input model as shown in Fig 17. The Gaussian mean error distribution is 1.2 m and the standard deviation is 6.6 m. The accuracy can be increased by using a more significant set of images and higher resolution.

Discussion and future work

SISPO is a sophisticated tool for terrestrial-cosmic-scenery rendering. Its most significant advantage is the physically based, high-quality, realistic renderings, which can reproduce sub-millimeter surface resolution and beyond for SSSBs and other terrestrial bodies thanks to micropolygon procedural texturing and Blender's Cycles path-tracing rendering engine. The produced renderings are mature for deep-space mission design and algorithm development for semi-autonomous operations, visual navigation, localisation and image processing. There are a few more functions that are considered for future implementation:

- Attitude dynamics auxiliary package (see below);

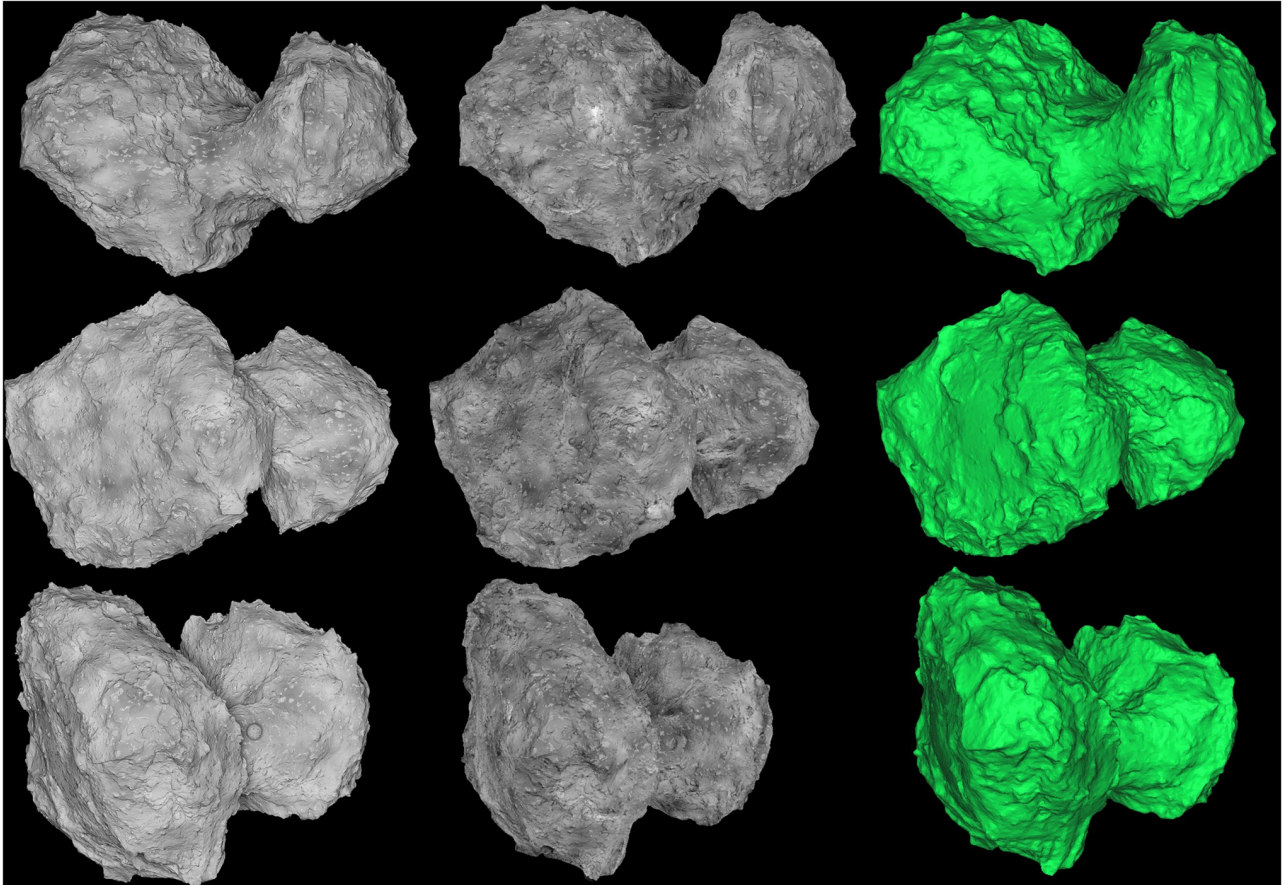


Fig 16. Full reconstruction of the comet during orbiting. The first column shows actual synthetic images, the second column is the textured view of the 3-D reconstruction, and the third column shows the meshed view of the reconstruction.

<https://doi.org/10.1371/journal.pone.0263882.g016>

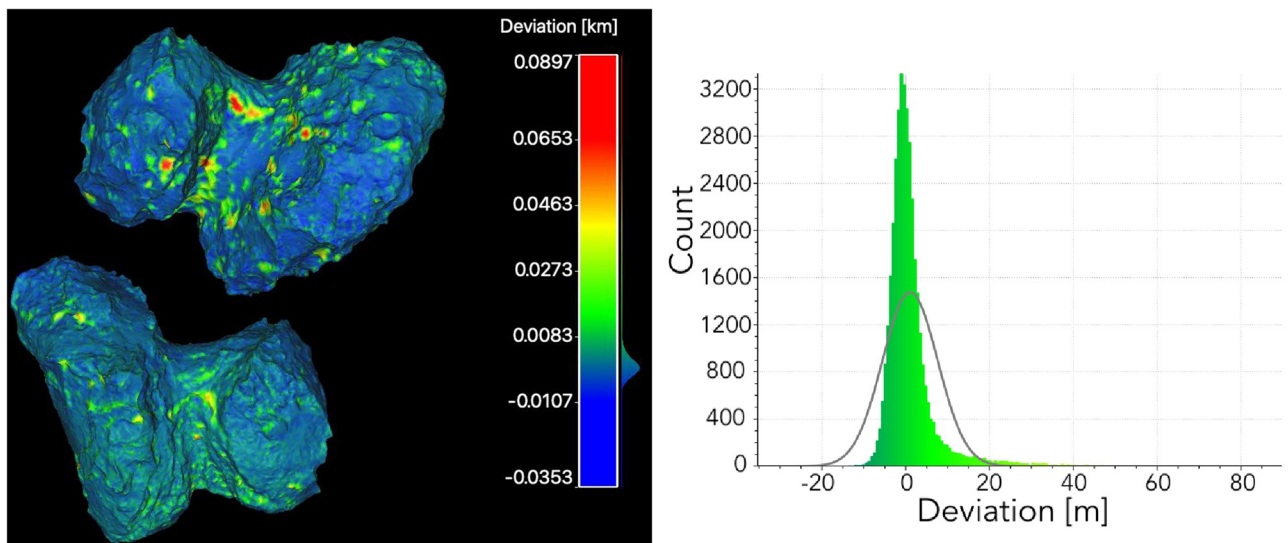


Fig 17. Deviation analysis of the entire reconstructed mesh and input 3-D model for image synthesis. Visualisation of deviation (left); errors and Gaussian distribution (right).

<https://doi.org/10.1371/journal.pone.0263882.g017>

- Image compression algorithms for efficient data storage and transmission evaluation, which was preliminarily assessed by [61];
- Reconstruction of various scenery;
- Algorithms for spacecraft photogrammetry-based localisation;
- Capability to simulate measurements of the target spectral reflectance in certain wavelength channels;
- Improved shader and user-defined parameterisation of Cycles in order to minimise the approximation;
- Solar System ephemeris integration for historical and upcoming events. This can include both terrestrial bodies and spacecraft via the possible adaptation of Spacecraft Planet Instrument Camera-matrix Events (SPICE) data.

Attitude dynamics auxiliary package

The SISPO simulation environment will include optional packages for calculating the effects of external forces such as dust particle impacts, atmospheric drag, gravity gradient and various other effects necessary to account for. Each of these functionalities will be built as an add-on to the current Orekit-based attitude framework. This is done by adding an optional extra calculation step between the propagations of the spacecraft state through time by Orekit to account for the perturbations. With the addition of the second layer on top of Orekit orbital simulations, for accurately simulating the attitude dynamics of the spacecraft or a celestial body, the SISPO environment will have the necessary groundwork for later expansions into different active and passive attitude control systems present in any future mission simulations.

GitHub algorithm repository

The open-source algorithm is stored in the public GitHub repository (<https://github.com/SISPO-developers>), and everyone is welcome to use and modify it. The algorithm is updated, and new functionalities are managed and added by authors. The instruction and comments can also be found in the repository. It contains following subrepositories: main SISPO (<https://github.com/SISPO-developers/sispo>), dust and gas environment generator for SSSBs (<https://github.com/SISPO-developers/ComaCreator>), comatic aberration and astigmatism simulator (<https://github.com/SISPO-developers/OASIS>) and docker image (https://github.com/SISPO-developers/sispo_docker) for fast SISPO deployment. Additional information about the algorithm can be found in [44, 61].

Conclusions

In this paper, the Space Imaging Simulator for Proximity Operations (SISPO) architecture and capabilities were described and several features demonstrated by case simulations. The tool generates physically based, photorealistic images from an input 3-D model using Blender's Cycles path-tracing rendering engine. For regolith surface reconstruction it uses procedural texturing. SISPO has supplementary models for optical aberrations as well as for gas and dust of small bodies. The description and usage of these models were discussed in this paper. Other use cases have demonstrated renderings of the Moon and a spacecraft. The set of produced images can be implemented for 3-D surface reconstruction. The tool is open access and currently requires basic programming skills to be used [33]. The level of detail currently produced

by SISPO is suitable for the design of advanced deep-space missions, the simulation of large sets of scenarios, and the development and validation of algorithms for (semi-)autonomous operations, vision-based navigation, localisation and image processing. SISPO has already supported the development of deep-space mission concepts and is currently being used for the ESA–JAXA Comet Interceptor mission (more details in Section: Application to space mission designs).

Some further improvements have been considered and are listed in Section: Discussion and future work. Other teams are welcome to use SISPO, implement new functions and contact the authors of this paper.

Supporting information

S1 Script. Image comparison. The file contains python algorithm used to compare images in this manuscript.
(ZIP)

Acknowledgments

We thank Mattias Malmer for allowing us to use his shape model of the comet 67P/Churyumov–Gerasimenko. Thanks to Hayabusa’s AMICA imaging team for publishing the shape model of 25143 Itokawa. We would like to express our gratitude to Airbus Defence and Space for developing and providing the SurRender software; to J r my Lebreton and Christine Pelsener-Scamaroni in particular for providing and accommodating changes to the licence agreement. Thanks to the University of Dundee and Martin Dunstan for developing and providing PANGU software and helping to set it up. A special mention goes to the Blender community, thanks to whom the open-source software exists and improves. Thanks to Tomas Kohout (University of Helsinki) for helping to arrange Timo V is nen’s civilian service (a.k.a. semi postdoc) at Aalto University.

Author Contributions

Conceptualization: Mihkel Pajusalu.

Data curation: Mihkel Pajusalu, Iaroslav Iakubivskyi, Olli Knuuttila.

Formal analysis: Gabriel J rg Schwarzkopf.

Funding acquisition: Mihkel Pajusalu.

Investigation: Iaroslav Iakubivskyi, Gabriel J rg Schwarzkopf, Olli Knuuttila, Maximilian B hrer, Mario F. Palos, Hans Teras, Andris Slavinskis.

Methodology: Mihkel Pajusalu, Iaroslav Iakubivskyi, Gabriel J rg Schwarzkopf, Timo V is nen, Maximilian B hrer, Hans Teras.

Project administration: Andris Slavinskis.

Software: Mihkel Pajusalu, Iaroslav Iakubivskyi, Gabriel J rg Schwarzkopf, Olli Knuuttila, Timo V is nen, Maximilian B hrer, Mario F. Palos, Hans Teras.

Supervision: Mihkel Pajusalu, Jaan Praks, Andris Slavinskis.

Validation: Iaroslav Iakubivskyi, Olli Knuuttila, Guillaume Le Bonhomme, Jaan Praks, Andris Slavinskis.

Visualization: Iaroslav Iakubivskiy, Olli Knuuttila, Timo Väisänen, Maximilian Bühner, Mario F. Palos, Guillaume Le Bonhomme.

Writing – original draft: Iaroslav Iakubivskiy.

Writing – review & editing: Mihkel Pajusalu, Iaroslav Iakubivskiy, Olli Knuuttila.

References

1. Fitzsimmons A, Snodgrass C, Rozitis B, Yang B, Hyland M, Seccull T, et al. Spectroscopy and thermal modelling of the first interstellar object 1I/2017 U1 'Oumuamua. *Nature Astronomy*. 2018; 2(2):133–137. <https://doi.org/10.1038/s41550-017-0361-4>
2. Fitzsimmons A, Hainaut O, Meech KJ, Jehin E, Moulane Y, Opitom C, et al. Detection of CN Gas in Interstellar Object 2I/Borisov. *The Astrophysical Journal*. 2019; 885(1):L9. <https://doi.org/10.3847/2041-8213/ab49fc>
3. Brochard R, Lebreton J, Robin C, Kanani K, Jonniaux G, Masson A, et al. Scientific image rendering for space scenes with the SurRender software. In: 69th International Astronautical Congress (IAC). Bremen, Germany: IAF; 2018. p. 1–11. Available from: <https://arxiv.org/abs/1810.01423>.
4. Lambert JH. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Klett; 1760. Available from: https://archive.org/details/bub_gb_zmpJAAAAYAAJ.
5. Hapke B. Bidirectional reflectance spectroscopy: 1. Theory. *Journal of Geophysical Research: Solid Earth*. 1981; 86(B4):3039–3054. <https://doi.org/10.1029/JB086iB04p03055>
6. Hapke B. Bidirectional reflectance spectroscopy: 3. Correction for macroscopic roughness. *Icarus*. 1984; 59(1):41–59. [https://doi.org/10.1016/0019-1035\(84\)90054-X](https://doi.org/10.1016/0019-1035(84)90054-X)
7. Hapke B. Bidirectional Reflectance Spectroscopy: 5. The Coherent Backscatter Opposition Effect and Anisotropic Scattering. *Icarus*. 2002; 157(2):523–534. <https://doi.org/10.1006/icar.2002.6853>
8. Oren M, Nayar SK. Generalization of the Lambertian model and implications for machine vision. *International Journal of Computer Vision*. 1995; 14(3):227–251. <https://doi.org/10.1007/BF01679684>
9. Martin I, Dunstan M, Gestido MS. Planetary Surface Image Generation for Testing Future Space Missions with PANGU. In: 2nd RPI Space Imaging Workshop. Saratoga Springs, NY, USA; 2019. p. 1–13. Available from: https://pangu.software/wp-content/pangu_uploads/pdfs/SpaceImagingWorkshop_2019_paper_pangu_final.pdf.
10. Acton C, Bachman N, Semenov B, Wright E. SPICE TOOLS SUPPORTING PLANETARY REMOTE SENSING. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2016; XLI-B4:357–359. <https://doi.org/10.5194/isprs-archives-XLI-B4-357-2016>
11. Semenov B. WebGeocalc and Cosmographia: Modern Tools to Access OPS SPICE data. In: 2018 SpaceOps Conference. Marseille, France; 2018. p. 2366. Available from: <https://doi.org/10.2514/6.2018-2366>.
12. Garreta Piñol B. Study: Visualization of spacecraft trajectories with NASA SPICE and Blender [B.S. thesis]. Universitat Politècnica de Catalunya; 2020. Available from: <https://upcommons.upc.edu/handle/2117/330122>.
13. Aiazzi C, Quadrelli MB, Gaut A, Jain A. Physics-based rendering of irregular planetary bodies. In: The International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS). Virtual: LPI contribution No.2358; 2020. Available from: <https://www.hou.usra.edu/meetings/isairas2020fullpapers/pdf/4009.pdf>.
14. AMICA imaging team. Shape models of 25143 Itokawa, the target of the HAYABUSA mission; 2007. Available from: <https://darts.isas.jaxa.jp/planet/project/hayabusa/shape.pl>.
15. Blender Online Community. Blender—a 3D modelling and rendering package; 2020. Available from: <http://www.blender.org>.
16. Pajusalu M, Slavinskis A. Characterization of Asteroids Using Nanospacecraft Flybys and Simultaneous Localization and Mapping. In: 2019 IEEE Aerospace Conference; 2019. p. 1–9. Available from: <https://doi.org/10.1109/AERO.2019.8741921>.
17. Kohout T, Näsilä A, Tikka T, Granvik M, Kestilä A, Penttilä A, et al. Feasibility of asteroid exploration using CubeSats—ASPECT case study. *Advances in Space Research*. 2018; 62(8):2239–2244. <https://doi.org/10.1016/j.asr.2017.07.036>
18. Walker R, Binns D, Bramanti C, Casasco M, Concari P, Izzo D, et al. Deep-space CubeSats: thinking inside the box. *Astronomy & Geophysics*. 2018; 59:24–30.

19. Bowles NE, Snodgrass C, Gibbings A, Sanchez JP, Arnold JA, Eccleston P, et al. CASTAway: An asteroid main belt tour and survey. *Advances in Space Research*. 2018; 62(8):1998–2025. <https://doi.org/10.1016/j.asr.2017.10.021>
20. Snodgrass C, Jones GH, Boehnhardt H, Gibbings A, Homeister M, Andre N, et al. The Castalia mission to Main Belt Comet 133P/Elst-Pizarro. *Advances in Space Research*. 2018; 62(8):1947–1976. <https://doi.org/10.1016/j.asr.2017.09.011>
21. Jones GH, Agarwal J, Bowles N, Burchell M, Coates AJ, Fitzsimmons A, et al. The proposed Caroline ESA M3 mission to a Main Belt Comet. *Advances in Space Research*. 2018; 62(8):1921–1946. <https://doi.org/10.1016/j.asr.2018.02.032>
22. Probst A, Förstner R. Spacecraft design of a multiple asteroid orbiter with re-docking lander. *Advances in Space Research*. 2018; 62(8):2125–2140. <https://doi.org/10.1016/j.asr.2017.07.041>
23. Oberst J, Wickhusen K, Willner K, Gwinner K, Spiridonova S, Kahle R, et al. DePhine—The Deimos and Phobos Interior Explorer. *Advances in Space Research*. 2018; 62(8):2220–2238. <https://doi.org/10.1016/j.asr.2017.12.028>
24. Ferri A, Pelle S, Belluco M, Voirin T, Gelmi R. The exploration of PHOBOS: Design of a Sample Return mission. *Advances in Space Research*. 2018; 62(8):2163–2173. <https://doi.org/10.1016/j.asr.2018.06.014>
25. Slavinskis A, Janhunen P, Toivanen P, Muinonen K, Penttilä A, Granvik M, et al. Nanospacecraft fleet for multi-asteroid touring with electric solar wind sails. In: 2018 IEEE Aerospace Conference. IEEE; 2018. p. 1–20. Available from: <https://doi.org/10.1109/AERO.2018.8396670>.
26. Iakubivskiy I, Mačiulis L, Janhunen P, Dalbins J, Noorma M, Slavinskis A. Aspects of Nanospacecraft Design for Main-Belt Sailing Voyage. *Advances in Space Research* (in press). 2020;.
27. ESA. Announcement of opportunity for new science ideas in ESA's science programme, viewed 08 April 2020; open call 2016. <https://www.cosmos.esa.int/web/new-scientific-ideas>.
28. ESA. CDF-178(C) study report: Small Planetary Platforms (SPP) in NEO and MAB (study manager: Bayon, S.), pp. 81–100, viewed 08 April 2020. European Space Agency; technical report 2018. Available from: <https://sci.esa.int/web/future-missions-department/-/60411-cdf-study-report-small-planetary-platforms-spp>.
29. Pajusalu M, Kivastik J, Iakubivskiy I, Slavinskis A. Developing autonomous image capturing systems for maximum science yield for high fly-by velocity small solar system body exploration. In: 71st International Astronautical Congress, IAC-20-A3.4B.4. Cyber Space: IAF; 2020. p. 1–8. Available from: <https://dl.iafastro.directory/event/IAC-2020/paper/61048/>.
30. Snodgrass C, Jones GH. The European Space Agency's Comet Interceptor lies in wait. *Nature communications*. 2019; 10(1):1–4. <https://doi.org/10.1038/s41467-019-13470-1> PMID: 31780664
31. Pernechele C, Deppo VD, Brydon G, Jones GH, Lara L, Michaelis H. Comet interceptor's EnVisS camera sky mapping function. In: Ellis SC, d'Orgeville C, editors. *Advances in Optical Astronomical Instrumentation 2019*. vol. 11203. International Society for Optics and Photonics. SPIE; 2020. p. 115–118. Available from: <https://doi.org/10.1117/12.2539239>.
32. Maisonobe L, Pommier V, Parraud P. Orekit: An open source library for operational flight dynamics applications. In: 4th International Conference on Astrodynamics Tools and Techniques. ESAC, Madrid, Spain; 2010.
33. Schwarzkopf GJ, Pajusalu M. Space Imaging Simulator for Proximity Operations; 2020. Available from: <https://doi.org/10.5281/zenodo.3661054>.
34. Whitted T. An Improved Illumination Model for Shaded Display. *Communications of the ACM*. 1980; 23(6):343–349. <https://doi.org/10.1145/358876.358882>
35. Flavell L. Chapter 4: Lighting and Procedural Textures. In: *Beginning Blender: Open Source 3D Modeling, Animation, and Game Design*. New York: Apress; 2011. p. 69–96.
36. McEwen AS. Photometric functions for photoclinometry and other applications. *Icarus*. 1991; 92(2):298–311. [https://doi.org/10.1016/0019-1035\(91\)90053-V](https://doi.org/10.1016/0019-1035(91)90053-V)
37. Hapke B. *Theory of Reflectance and Emittance Spectroscopy*. 2nd ed. Cambridge: Cambridge University Press; 2012. Available from: <https://doi.org/10.1017/CBO9781139025683>.
38. Williams L. Casting Curved Shadows on Curved Surfaces. In: *Seminal Graphics: Pioneering Efforts That Shaped the Field*. New York, NY, USA: Association for Computing Machinery; 1998. p. 51–55. Available from: <https://doi.org/10.1145/280811.280975>.
39. Schmitt MI, Tubiana C, Güttler C, Sierks H, Vincent JB, El-Maarry MR, et al. Long-term monitoring of comet 67P/Churyumov-Gerasimenko's jets with OSIRIS onboard Rosetta. *Monthly Notices of the Royal Astronomical Society*. 2017; 469:S380–S385. <https://doi.org/10.1093/mnras/stx1780>
40. Finson M, Probst R. A theory of dust comets. I. Model and equations. *The Astrophysical Journal*. 1968; 154:327–352. <https://doi.org/10.1086/149762>

41. Kramer T, Läuter M, Rubin M, Altwegg K. Seasonal changes of the volatile density in the coma and on the surface of comet 67P/Churyumov–Gerasimenko. *Monthly Notices of the Royal Astronomical Society*. 2017; 469(Suppl_2):S20–S28. <https://doi.org/10.1093/mnras/stx866>
42. Kramer T, Noack M, Baum D, Hege HC, Heller EJ. Dust and gas emission from cometary nuclei: the case of comet 67P/Churyumov–Gerasimenko. *Advances in Physics: X*. 2018; 3(1):1404436.
43. Lévassieur-Regourd AC, Renard JB, Hadamcik E, Lasue J, Bertini I, Fulle M. Interpretation through experimental simulations of phase functions revealed by Rosetta in 67P/Churyumov–Gerasimenko dust coma. *Astronomy and Astrophysics*. 2019; 630:A20. <https://doi.org/10.1051/0004-6361/201834894>
44. Bühner M. Simulation of Optical Aberrations for Comet Interceptor's OPIC Instrument [M.Sc. thesis]. Luleå University of Technology and Cranfield University; 2020. Available from: <http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-81638>.
45. Brown DC. Close-range camera calibration. *Photogrammetric Engineering*. 1971; 37(8):855–866.
46. Culjak I, Abram D, Pribanic T, Dzapo H, Cifrek M. A brief introduction to OpenCV. In: 2012 Proceedings of the 35th International Convention MIPRO. Opatija, Croatia; 2012. p. 1725–1730. Available from: <https://ieeexplore.ieee.org/document/6240859>.
47. Gaskell R, Barnouin-Jha O, Scheeres DJ, Konopliv A, Mukai T, Abe S, et al. Characterizing and navigating small bodies with imaging data. *Meteoritics & Planetary Science*. 2008; 43(6):1049–1061. <https://doi.org/10.1111/j.1945-5100.2008.tb00692.x>
48. Kainz F, Bogart R, Hess D. The OpenEXR image file format. In: ACM SIGGRAPH Technical Sketches; 2003. Available from: <https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-26-openexr-image-file-format>.
49. Lekien F, Marsden J. Tricubic interpolation in three dimensions. *International Journal for Numerical Methods in Engineering*. 2005; 63(3):455–471. <https://doi.org/10.1002/nme.1296>
50. Spudis PD, Ryder G. Geology and petrology of the Apollo 15 landing site: Past, present, and future understanding. *Eos, Transactions American Geophysical Union*. 1985; 66(43):721–726. <https://doi.org/10.1029/EO066i043p00721>
51. Robinson MS, Ashley JW, Boyd AK, Wagner RV, Speyerer EJ, Ray Hawke B, et al. Confirmation of sublunarean voids and thin layering in mare deposits. *Planetary and Space Science*. 2012; 69(1):18–27. <https://doi.org/10.1016/j.pss.2012.05.008>
52. Cushing GE, Titus TN, Wynne JJ, Christensen PR. THEMIS observes possible cave skylights on Mars. *Geophysical Research Letters*. 2007; 34(17):L17201. <https://doi.org/10.1029/2007GL030709>
53. Nesnas IA, Kerber L, Parness A, Kornfeld R, Sellar G, McGarey P, et al. Moon Diver: A Discovery Mission Concept for Understanding the History of Secondary Crusts through the Exploration of a Lunar Mare Pit. In: 2019 IEEE Aerospace Conference; 2019. p. 1–23. Available from: <https://doi.org/10.1109/AERO.2019.8741788>.
54. Uckert K, Parness A, Chanover N, Eshelman EJ, Abcouwer N, Nash J, et al. Investigating Habitability with an Integrated Rock-Climbing Robot and Astrobiology Instrument Suite. *Astrobiology*. 2020; 20(12):1427–1449. <https://doi.org/10.1089/ast.2019.2177> PMID: 33052709
55. Sauro F, Pozzobon R, Massironi M, De Berardinis P, Santagata T, De Waele J. Lava tubes on Earth, Moon and Mars: A review on their size and morphology revealed by comparative planetology. *Earth-Science Reviews*. 2020; 209:103288. <https://doi.org/10.1016/j.earscirev.2020.103288>
56. Moulon P, Monasse P, Perrot R, Marlet R. OpenMVG: Open Multiple View Geometry. In: Kerautret B, Colom M, Monasse P, editors. *Reproducible Research in Pattern Recognition*. Cham: Springer International Publishing; 2017. p. 60–74. Available from: https://link.springer.com/chapter/10.1007/978-3-319-56414-2_5.
57. Moulon P, Duisit B, Monasse P. Global Multiple-View Color Consistency. In: *Proceedings of CVMP 2013*. Londres, United Kingdom; 2013. Available from: <https://hal-enpc.archives-ouvertes.fr/hal-00873517/>.
58. Liu Z, Marlet R. Virtual line descriptor and semi-local matching method for reliable feature correspondence. In: *Proceedings of British Machine Vision Conference 2012*. Surrey, United Kingdom; 2012. p. 16.1–16.11. Available from: <https://hal.archives-ouvertes.fr/hal-00743323>.
59. Cernea D. OpenMVS: Multi-View Stereo Reconstruction Library; 2021. Available from: <https://cdcseacave.github.io/openMVS>.
60. Girardeau-Montaut D. Cloudcompare—open source project; 2011. Available from: <https://www.cloudcompare.org>.
61. Schwarzkopf GJ. 3D Reconstruction of Small Solar System Bodies using Rendered and Compressed Images [M.Sc. thesis]. Luleå University of Technology, Aalto University; 2020. Available from: <http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-77846>.