UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Physics

Karin Kruuse

Implementing a Terramechanics-based Wheel-terrain Contact Model into a Multi-body Simulation Environment

Master's thesis (30 ECTS)
Physics

Supervisors:
Quazi Saimoon Islam, MSc
Hans Teras, MSc

Tartu 2023

Implementing a Terramechanics-based Wheel-terrain Contact Model into a Multi-body Simulation Environment

Abstract:
The efficacy of planetary rover mission planning and analysis relies heavily on robust simulations. This research is centered on the refinement of ULYSSES, a planetary surface sandbox simulation framework. The integration of a terramechanics-based model stands as a focal point, enriching the precision of rover mobility simulations, particularly for navigating soft and challenging soil conditions. The study involves a comprehensive reassessment of ULYSSES' physics engine, followed by a methodical model implementation and a benchmarking process against preexisting experimental data. The findings highlight a stride towards heightened simulation accuracy within complex terrains, thus facilitating more precise mission planning strategies. This holds particular relevance in light of the upcoming lunar expeditions that have garnered substantial attention within the space exploration domain.

Keywords: Simulation, Terramechanics

CERCS: T125 — Automation, robotics, control engineering, T320 — Space Technology

Terramehaanikal-põhineva mudeli rakendamine mitme-keha simulatsioonikeskkonnas

Abstrakt:
Planetaarkulgurite missioonide planeerimine ja analüüsimine toetub suuresti simulatsioonidele. Käesolev töö keskendub Kuu missioonide simuleerimiseks loodud keskkonna ULYSSESe täiustamisele. Selleks tutvutakse terramehaanikal põhineva ratta ja pinnase kontakti mudeliga. Lisaks hõlmab töö ka ULYSSESis kasutusel oleva jäiga keha füüsikamootori uurimist, mille abil täpsem kontakti mudel rakendatakse. Töös antakse ülevaade terramehaanika põhimudeli, Bekkeri valemi ja selle täienduste rakendamisest ULYSSESe keskkonda. Töö tulemused valideeritakse võrdluses eksperimendi tulemustega.

Võtmesõnad: Simulatsioon, Terramehaanika

CERCS: T125 — Automatiseerimine, robootika, control engineering, T320 — Kosmosetehnoloogia

Contents

Introduction

Simulations have always played a crucial role when it comes to the intangible – be that something too big, small, temporally challenging, or just far away. But it can also be a valuable tool to aid new technologies' development and investigation processes. So it is no surprise that as the number of planned planetary exploration missions is rising, so is the interest in simulation tools that assist mission planning and the analysis of mobility for robots.

The surge in upcoming lunar missions has become a significant catalyst for advancing planetary rover mobility research [1]. These systems are now confronted with challenges concerning mission planning, obstacle avoidance, and soil adaptability. Testing under actual environmental conditions can be expensive, time-consuming, or infeasible. Moreover, the operational sites often entail uncertain and insufficiently known environmental conditions. Hence, experimental tests are supplemented with numerical simulation models [2, 3]. This approach allows for cost-effective and time-efficient exploration of various scenarios.

Such capabilities have become a topic of interest also at Tartu Observatory. More specifically, the researchers are working on ULYSSES – a sandbox simulation framework for planetary surfaces [4]. It is developed in Unreal Engine 5 [5]. ULYSSES' initial purpose was to validate a lunar mission planning tool built in collaboration with Milrem Robotics and ESA (European Space Agency) ESOC (European Space Operations Centre). One of the validation runs is shown in Figure 1. This collaboration project was successfully finished in 2022 [6], but the development of ULYSSES is continued by Tartu Observatory.

When it comes to simulating rover traversals, the first step to improving the fidelity of ULYSSES is better wheel-terrain contact modeling. Currently, the framework relies on a simplistic rigid body contact representation. Despite the growing body of research in this domain, the availability of resources remains wanting. Custom-made environments are often employed when such simulations are required, excluding wider accessibility.

This thesis aims to implement a terramechanics-based model into ULYSSES, which encompasses a broader array of effects encountered during traversals through soft soil beyond the capabilities of the current version. As a preliminary step, an in-depth reassessment of the overarching rigid body physics engine within ULYSSES is undertaken. The final implementation of said model is then compared to experimental results from other sources. Analysis of the results in combination

with some recommendations is also presented.

The first two chapters are meant to provide an overview of the fundamentals of physics engines and terramechanics. The former is used as a tool in this work, while the latter can be considered a subject. In the third chapter, a description of the implementation is given. The fourth chapter serves as the crucible where the model's efficacy is put to the test.

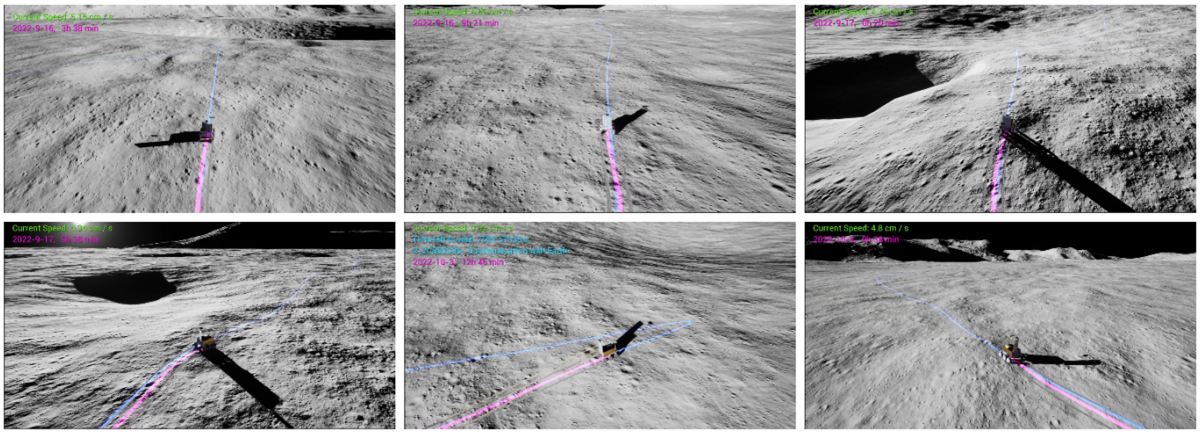

Figure 1. *Screen captures from a mission planner validation run in ULYSSES to provide an understanding of the application. The rover was made to follow the blueish line using a PID controller. The pink line indicates the traversed trajectory. The planned mission lasted for several hours.*

1. Multi Body Simulations

In this chapter, a broad overview of multi-body simulations will be given. More specifically, the mathematical background and some nuances regarding implementation will be discussed.

1.1 Rigid Body Motion

In this section, the formulations of various mechanical quantities will be covered to firstly familiarise the reader with the notation used in this thesis and, secondly, to create a better context for the following chapters. However, the intent here is not to give an extensive overview of the subject. For this, the reader might refer to [7].

1.1.1 Kinematics

It is useful to start by taking a look at the kinematics of a particle. Let its position be given by $\mathbf{x}$[1]. Then the velocity $\mathbf{v}$ is given by

$$\mathbf{v} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \dot{\mathbf{x}}. \tag{1.1}$$

This thesis will use the latter dot notation for time derivatives. The second time derivative of position gives the acceleration $\mathbf{a}$

$$\mathbf{a} = \ddot{\mathbf{x}}. \tag{1.2}$$

With these quantities, one can already describe the movement of a particle. Of course, position, velocity, and acceleration are likely quite familiar to the reader and don't require much commentary for context.

The discussion becomes more intriguing when the focus shifts from particles to rigid bodies. A rigid body is generally defined as a system of particles where the distance between each pair of particles stays constant. Considering Newton's laws, it can be shown that the translational motion of a rigid body can also be described using the three quantities already mentioned as calculated for the body's center of mass. However, a rigid body can also rotate around axes going through itself, changing its orientation. In three-dimensional space, this concludes with six degrees of freedom for a rigid body: three translational and three rotational.

It is worth looking at the various ways orientation is commonly represented. In physics engines,

[1] Vectors of any dimension will be denoted by bold letters in this thesis as is usual in the relevant literature.

Euler angles, rotation matrices, and quaternions can generally all be used. Yet, in the relevant literature, there seems to be a preference for the latter two.

The first representation, Euler angles, usually denoted as (α, β, γ), is a set of three angles that define three successive rotations in a specific order around a set of axes. On the one hand, this seems like the intuitive system, as translational coordinates are used quite similarly. But on the other hand, many different conventions exist dictating which axes and in which order are used as the axis of rotation. This can often create confusion. One widely used convention is the zxz order, where rotations are defined by a rotation around the z-axis, followed by the x-axis, and another rotation around the z-axis. Another popular convention is the zyx form. In this case, the angles are called yaw, pitch, and roll. However, It can be noted that these terms are sometimes used loosely in literature – the definition an author operates with can be hazy.

In regards to the preference for other representations, while Euler angles seem more intuitive, they come with drawbacks that might initially go unnoticed when not pointed out. One of these drawbacks is that some orientations can be described by several sets of Euler angles, making the problem of solving for them non-trivial. Also, rotations around different axes don't commute. This, combined with the fact several conventions for applying rotations defined by Euler angles to a body exist, makes mistakes in calculations rather simple to occur. A phenomenon known as gimbal lock can also pose a challenge when setting up a sequence of rotations, leading to the loss of one degree of freedom [8]. Hence, when performing calculations, rotation matrices and quaternions are used preferentially, while Euler angles, specifically Tait–Bryan angles, are used to communicate angles to the user.

Quaternions were discovered as a mathematical tool more than 150 years ago by an Irish mathematician and physicist William Rowan Hamilton, who worked on popularising quaternions by writing difficult-to-read books for hundreds if not thousands of pages [9]. Luckily for students, vector analysis took over around the end of the 19th century. In 1985 Shoemake [10] introduced unit length quaternions to the computer graphics community as a means of representing rotation where, as said, they prevail today.

Quaternions are an extension of complex numbers and provide an elegant mathematical framework for expressing three-dimensional orientations. A quaternion $\mathbf{Q}$ consists of four parameters v_1, v_2, v_3 and s such that,

$$\mathbf{Q} = v_1\mathbf{i} + v_2\mathbf{j} + v_3\mathbf{k} + s, \tag{1.3}$$

where 1, $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are basis vectors. Here the use of the letter v for the imaginary part of the quaternion is adapted from Shomaker's work and has no relation to velocity. In vector format,

they are written as

$$\mathbf{Q} = \begin{pmatrix} \mathbf{v} \\ s \end{pmatrix}. \tag{1.4}$$

Rotations are represented by unit quaternions, meaning

$$v_1^2 + v_2^2 + v_3^2 + s^2 = \mathbf{v} \cdot \mathbf{v} + s^2 = 1 \tag{1.5}$$

and hence still have three degrees of freedom. Probably the most useful operation in this context is the multiplication of quaternions $\mathbf{QQ}'$:

$$\mathbf{QQ}' = \begin{pmatrix} s'\mathbf{v} + s\mathbf{v}' + \mathbf{v} \times \mathbf{v}' \\ ss' - \mathbf{v} \cdot \mathbf{v}' \end{pmatrix}, \tag{1.6}$$

which results in the combined rotation of both quaternions. Note that due to the cross-product, this operation is non-commutative, as it should be considering what it is representing.

The last common way of representing orientations mentioned is the rotation matrix $\mathbf{R}$, which can be used to rotate a vector by a simple matrix multiplication

$$\mathbf{x}' = \mathbf{Rx}. \tag{1.7}$$

A 3×3 matrix has nine elements, representing nine degrees of freedom. However, to describe a valid rotation, these nine degrees of freedom must be constrained to the three – the matrix must be orthogonal and have a determinant of 1: $|\mathbf{R}| = 1$.

Due to numerical drift, the matrix can lose its orthogonal property. As a consequence, a non-orthogonal matrix may cause shearing and scaling effects. To prevent these issues, regular re-orthogonalization of the rotation matrix is required. The standard approach for achieving this is the Gram-Schmidt orthonormalization, which is discussed in various linear algebra textbooks. To construct a rotation matrix for an arbitrary rotation, Foley et al. [11] provide guidance. While quaternions suffer from the same effect, it is to a much lesser degree as quaternions use fewer elements to represent a rotation.

As rotations are introduced to the system, discussing the frame of reference becomes important. In rigid body simulations, one usually starts with a global or inertial reference frame Σ. This can also be referred to as world space. For each rigid body $\mathcal{B}$ considered in the simulation, a local reference frame $\Sigma_{\mathcal{B}}$ is defined, also referred to as body space. The transform of the body describes a transformation from the global frame to the local frame. Given a point $\mathbf{x}^{\mathcal{B}}$ in the

reference frame $\Sigma_{\mathcal{B}}$, its coordinates in the global frame are given by the transformation

$$\mathbf{x} = \mathbf{x}_{\text{CM},\mathcal{B}} + \mathbf{R}_{\mathcal{B}}\mathbf{x}^{\mathcal{B}}. \tag{1.8}$$

Here and later, the subscript CM denotes a vector designated for the center of mass. Also, a superscript was used to differentiate between coordinate systems. The rotation matrix $\mathbf{R}_{\mathcal{B}}$ represents the orientation of the body with respect to Σ. By default, the origin of $\Sigma_{\mathcal{B}}$ is located at the center of mass of the rigid body in every well-known physics engine.

The angular velocity of the body $\mathcal{B}$ is usually denoted by $\boldsymbol{\omega}_{\mathcal{B}}$. Its direction defines the rotational axis and its magnitude, the rate of rotation. As every particle in this rigid body has the same angular velocity, the velocity $\mathbf{v}$ of a point or particle on the body $\mathcal{B}$ can be calculated as

$$\mathbf{v} = \mathbf{v}_{\text{CM},\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{B}} \times \mathbf{x}^{\mathcal{B}}. \tag{1.9}$$

From here, one can define a generalized coordinate vector $\mathbf{q}$ containing its position and rotation for some body and similarly a generalized velocity vector $\mathbf{u}$:

$$\mathbf{q} = (\mathbf{x}_{\text{CM}}, \mathbf{Q}), \qquad \mathbf{u} = (\mathbf{v}_{\text{CM}}, \boldsymbol{\omega}). \tag{1.10}$$

It can be seen from the dimensions of these two vectors that $\dot{\mathbf{q}}$ is not equal to the generalized velocity vector. Instead

$$\dot{\mathbf{q}} = \mathbf{H}\mathbf{u} = \begin{pmatrix} \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{G} \end{pmatrix} \mathbf{u}, \tag{1.11}$$

where the 3×4 matrix $\mathbf{G}$ comes from the relation

$$\dot{\mathbf{Q}} = \mathbf{G}\boldsymbol{\omega}. \tag{1.12}$$

1.1.2 Dynamics

Physics engines usually assume that the center of mass lies in the geometrical center of a rigid body, i.e., the body's density is homogenous over its volume. This assumption generally is not far from the truth. As was done with position and velocity, a generalized mass matrix $\mathbf{M}$ can be defined as it will be helpful when formulating the laws of motion.

$$\mathbf{M} = \begin{pmatrix} m\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}. \tag{1.13}$$

The matrix $\mathbf{I}_3$ is a 3×3 identity matrix, and the matrix $\mathbf{I}$ is the inertia matrix. The latter deserves more attention, mainly when it comes to considering different frames of reference. For this, the reader might want to read [7]. The Newton-Euler equations describing the combined translational

and rotational dynamics of a rigid body in local space with the origin at the center of mass can be written:

$$\mathbf{M}\dot{\mathbf{u}} = \mathbf{g}. \tag{1.14}$$

A new vector, the load vector $\mathbf{g}$, is introduced here. This vector contains the forces $\mathbf{f}$ and torques $\boldsymbol{\tau}$ acting on the body.

$$\mathbf{g} = \begin{pmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} \end{pmatrix}. \tag{1.15}$$

The force vector $\mathbf{f}$ encapsulates all forces acting on the center of mass of a body, and so does the vector $\boldsymbol{\tau}$ encapsulate all torques acting about the center of mass. If some contact force $\mathbf{f}_c$ is applied to any other point $\mathbf{x}_c$ on the body, it adds to $\mathbf{f}$ but also adds to torque

$$\boldsymbol{\tau}_c = \mathbf{x}_c \times \mathbf{f}_c. \tag{1.16}$$

1.2 Constraints

The knowledge from the previous section provides one with enough to describe the motion of unconstrained rigid bodies: they do not interact with the environment or each other via contact. Constraints are used to add some character, so to say, to a simulation. They describe how bodies are allowed to move relative to one another. For example, a door is constrained to a rotation about one axis. Similarly, temporary collisions between rigid bodies are also resolved using constraints. Generally, when discussing constraints as used in physics engines, they are just kinematic restrictions. Hence they do not explicitly define the forces needed to be enforced. Mathematically the simplest constraints are formulated as equations

$$C(\mathbf{q_1}, \mathbf{q_2}, ..., t) = 0. \tag{1.17}$$

As the function C only depends on the position of bodies under consideration[2], the corresponding constraint equation (1.17) is said to be a holonomic constraint. Such constraints reduce the degrees of freedom of the given system. Non-holonomic constraints take the general form

$$C(\mathbf{q_1}, \mathbf{u_1}, \mathbf{q_2}, \mathbf{u_2}, ..., t) \geq 0. \tag{1.18}$$

They can both be dependent on velocity and be described by an inequality. Constraint (1.17) is also called an equality or a bilateral constraint, and (1.18) an inequality or a unilateral constraint. Not to say that the latter is synonymous with non-holonomic, as a non-holonomic constraint can still be a bilateral constraint. The distinction is that a non-holonomic constraint can not be integrated to give an equivalent holonomic constraint. One should note that non-holonomic

[2]The dependence on time t is more of a formality and often left out. However, it can be noted that holonomic constraints not dependent on time are called scleronomic and the ones dependent on time are called rheonomic constraints.

equality constraints remove only instantaneous degrees of freedom from the system.

A common example used to illustrate constraints is of a particle restricted to move along a circular trajectory with a radius of r. In this case, the constraint function can be written as

$$C = \frac{1}{2}\left(\mathbf{x}\cdot\mathbf{x} + r^2\right). \tag{1.19}$$

Such a formulation makes the differentiation simple:

$$\dot{C} = \mathbf{x}\cdot\dot{\mathbf{x}}, \tag{1.20}$$

$$\ddot{C} = \ddot{\mathbf{x}}\cdot\dot{\mathbf{x}} + \dot{\mathbf{x}}\cdot\dot{\mathbf{x}}. \tag{1.21}$$

One might be left to think here of what use are the derivatives of the constraint function to us. Consider having the constraint equation $C = 0$ satisfied. If, at the same time, the constraint $\dot{C} = 0$ is not satisfied, then during the next time step, the position constraint will be broken. Similarly, if both equations $C = 0$ and $\dot{C} = 0$ are satisfied, then having $\ddot{C} \neq 0$ will lead to them being also broken. With these considerations, different strategies can be formed to deal with equality constraints:

1. one keeps the equation $C = 0$ satisfied by changing the position of the body under consideration,
2. one assumes $C = 0$ and keeps the equation $\dot{C} = 0$ satisfied by calculating corrective impulses to be added to the body under consideration,
3. one assumes both $C = 0$ and $\dot{C} = 0$ are satisfied and calculates corrective forces, or constraint forces, to also assert $\ddot{C} = 0$.

These are the central ideas in position-based, impulse-based, and force-based methods of constraint solving, respectively. Currently, velocity-based or, rather, impulse-based methods are most widely used in physics engines. While force-based methods were introduced first, due to their tendency to produce higher and higher corrective forces, they have been largely disregarded today [12]. However, the numerical methods introduced then have been partially adapted for impulse-based constraint solving. This is the most prevalent way of dealing with constraints now. A new field of position-based dynamics is also growing. Previously this approach wasn't considered seriously, as when implemented incorrectly, noticeable discontinuities in velocity-space can be created, which is rather noticeable to the eye. It also has a reputation of being non-physical [13].

It serves to discuss the penalty method for contact resolution briefly. In this paradigm, contact forces are based on spring equations, usually consisting of two terms: one associated with the elastic component (the spring) and another related to energy dissipation (the damper). Two

formulas more common in the literature relating to physics engines are the Kelvin-Voigt [14] and Hunt-Crossley formulas [15], as many works build upon them. Currently, none of the well-known open-source rigid body engines uses this method. However, it was popular before the popularisation of the impulse-based method [12]. This is likely partially due to the discovery of the linear complementarity problem as a perfect hammer for the nail of collision solving.

1.2.1 Linear Complementarity Problem

The author would like to take a moment to delve into the topic of the linear complementarity problem (LCP). It was initially thoroughly explored by Dantzig and Cottle [16]. In 1994 Baraff suggested using an adaptation of Dantzig's algorithm for computing contact forces during a two-body collision, as the LCP formulation encapsulates the problem of non-penetrating contact quite elegantly [17]. For n contact points, an LCP is formulated as:

$$\mathbf{A}\mathbf{a} + \mathbf{b} = \mathbf{f}, \tag{1.22}$$

$$a_i \geq 0, \tag{1.23}$$

$$f_i \geq 0, \tag{1.24}$$

$$f_i a_i = 0. \tag{1.25}$$

The $n \times n$ matrix $\mathbf{A}$ represents the masses and contact geometries of the bodies. This matrix can create coupling between the contact points, which makes the problem of solving for the forces difficult. The vector $\mathbf{b}$ reflects the external and inertial forces acting on the body. Vectors $\mathbf{a}$ and $\mathbf{f}$ hold the acceleration and force magnitudes a_i and f_i for each contact – once again, the notation deviated a bit from what was shown in section 1.1. The directions of the accelerations and forces are determined by the contact normal and are not directly considered in the LCP. Here, a rigid body's acceleration is positive when it separates from the other body involved in the collision. The equation (1.22) is essentially a formulation of Newton's second law. Inequality (1.23) forces the bodies to separate; as negative accelerations are not allowed, penetration is not allowed. The inequality (1.24) doesn't allow pulling forces, and the complementarity condition (1.25) states that a separating contact force can not act on bodies that are already separating.

Various pivoting algorithms – meaning algorithms that find and refine the solution by iterating over points under consideration – have been developed for LCPs. Lemke's algorithm [18] is the favorite amongst the real-time physics engine community. However, Dantzig's algorithm is also widely used.

1.2.2 Friction Constraints

Kinetic constraints are relatively easy to include in an LCP as similar constraints to (1.23)-(1.24); the number of such constraints is not limited for an LCP to be solvable. When also

including bilateral constraints, the problem becomes an MLCP – a mixed LCP. However, when encountering friction, this formulation loses some of its elegance. The central problem with including a frictional force is its direction. The magnitude of Coulomb friction force $\mathbf{f_f}$, which is the friction model used in physics engines, is constrained to

$$|\mathbf{f_f}| \leq \mu |\mathbf{f_z}|, \tag{1.26}$$

while its direction is perpendicular to the normal force $\mathbf{f_z}$. Here, μ is the friction coefficient. This means that the contact force is now bound to a cone instead of just being in the direction of the contact normal. To still make the constraint linear, the cone is approximated by a 4-sided pyramid [19].

1.3 Physics Engines

Broadly speaking, a physics engine is software that deals with various physics-related calculations: rigid body dynamics, soft body dynamics, fluid dynamics, and such. In this thesis, a rigid body engine is used. However, the main contribution of this work deviates from the usual framework of rigid body simulations. The implementation of a more complex wheel-ground interaction model generally lies in the field of terramechanics.

The section below will explore the anatomy of a real-time rigid-body physics engine. After an overview of the generic pipeline, some phases will be covered more thoroughly – mainly collision detection, as this hasn't been discussed yet.

Rigid body simulators typically operate through a simulation loop, as illustrated in Figure 1.1. The loop begins by applying gravity to each body. If applicable, other non-contact forces, for example, due to external electric fields or air resistance, would be applied here as well. Based on these forces, the positions of all bodies are calculated for a discrete timestep. This can easily be done based on the previously introduced equations of motion.

After the initial propagation of bodies, collision detection has to be performed to identify contact points between different bodies. This is done in two phases to prevent collision detection from becoming a computational bottleneck. The first, termed broadphase, sorts out likely pairs of bodies to be in contact with higher speed and lower accuracy than the following phase. The increase in speed is achieved by using simplified geometry to test for collisions. The most common method makes use of the axis-aligned bounding box (AABB). This is just a box around an object with its edges aligned with the world frame axes. As bodies move and rotate, the bounding boxes have to be recalculated in the world space at every simulation step. But this is by far outweighed by the efficiency of testing collisions between AABBs compared to

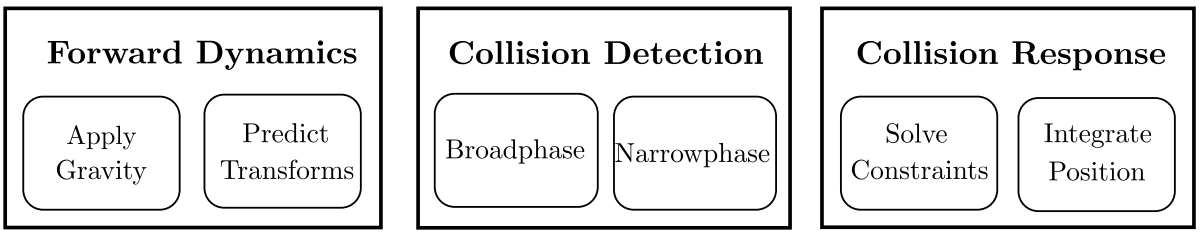

Figure 1.1. *Overview of main computational phases of a modern physics engine. These tasks are executed during every simulation step. The first block of forward dynamics performs calculations based on the mechanics discussed in section 1.1. Collision detection is usually done in two phases to save computation time, as the algorithms used in the narrowphase are rather expensive.*

more complex geometries. In addition to using simplified geometry, spacial partitioning can be used to avoid comparing bodies that are certainly too far from each other to collide. By splitting the simulation volume into cells the two bodies can only collide if they lie in the same cell(s).

The potentially colliding pairs of bodies are then passed to the narrowphase collision detection, where the detailed geometries of the bodies are used to precisely identify the body features in contact and determine the location of contact points. This process is sometimes called contact generation. Usually, a surface area and pressure come to mind when one thinks of a contact. However, in physics engines, as seen before in section 1.2, contacts are resolved discretely by applying either forces or impulses to only some points determined to be the best to describe the contact. The exact way of determining these specific points for a detected contact varies from engine to engine [20].

In the narrowphase, the pairs of bodies are sorted based on their geometry, again for optimization. When two bodies with primitive shapes – these are simple collision shapes like a sphere or a box that can be, in certain cases, used to represent the collision shape of a body – are marked as potentially colliding, it is rather easy to analytically check whether they intersect or not. For other convex bodies, the Gilbert–Johnson–Keerthi distance algorithm first presented in [21] is used most often. For concave bodies, the same applies after convex decomposition is performed. It is worth mentioning that certain narrowphase algorithms may not supply all the essential contact information needed to establish the dynamic model. In such cases, a separate contact point generation algorithm becomes necessary. However, this thesis does not specifically explore collision detection methods. For further discussion on this topic, the author recommends referring to [22].

The newly generated contact points are then formulated similarly to (1.22)-(1.25). In addition, other kinematic constraints alike (1.17) and (1.18) are added to the LCP formulation. The part of the engine responsible for finding the solution of the LCP is often referred to as the

solver. As the last step, the whole simulation is incremented by a time step by integrating the Euler-Newton equations (1.14) and also the velocity equation (1.11) to update the velocities and positions. A variation of the Euler integration method is often used – especially in engines intended for games and animations – and might not even be accessible to the user without editing the source code.

2. Wheel-Ground Modeling in Planetary Exploration

The field of terramechanics[1], as it has been called since the Polish scientist and engineer Mieczysław G. Bekker published the book "Off-the-road Locomotion" in 1960 [23], deals with soil properties and its interaction with wheeled or tracked vehicles. Bekker's work was the basis for the Lunar Roving Vehicle [24] built for the Apollo missions.

Several computational approaches are taken for wheel-ground interaction modeling. Here, the focus is on semi-empirical models due to their real-time feasibility and accurate prediction of experimental outcomes [25]. While the discrete element method is also commonly used, it generally serves other purposes giving more thorough feedback for a specific wheel design [3].

2.1 Normal Force

The magnitude of the normal force f_z is the foundation of almost every contact mechanics formulation. From a simple Coulomb friction model to complex terramechanics models, they all use the magnitude of the normal force to determine the available traction force in tangent directions. So while often the slip behavior of the wheel is of interest, the normal force has to be modeled accurately.

2.1.1 Visco-Elastic Model

A dampened spring model is occasionally used to simulate the interaction between a wheel and soft soil. This is mostly used in older works. The choice of this mathematical model is primarily driven by the consideration that it appears accurate when using suitable stiffness parameters. Selecting the right parameters seems more of a black art than a methodical process. Nevertheless, the aspect of "looking right" still holds value in some cases.

To represent the wheel-soil contact normal force, a so-called Kelvin-Voigt element is introduced. The name comes from separate works by Kelvin [26] and Voigt [27], who investigated

[1] It may seem like the name borrows from the Latin word *Terra*, meaning Earth. Instead, the word *terramechanics* is compounded from the words *terrain* and *mechanics*, and so the same word is used in researching extraterrestrial soils as well.

contact forces generated in metals. So generally, the following formulas aren't necessarily considered under the terramechanics umbrella as they weren't intended to replicate any of the physical soil phenomena. However, these formulas have been an essential tool in planetary rover modeling.

The normal force acting on the wheel induced by the soil is calculated based on the wheel sinkage h as

$$f_z = kh^n + \gamma \dot{h}. \tag{2.1}$$

The stiffness parameter k, spring exponent n and dampening γ are often set by fine-tuning based on physical reasoning and are rarely discussed in depth as they're not based on specific physical quantities but rather groups of them. For example, in [28] authors state: "k is large". This model was used in the ARTEMIS simulation environment as a comparison for a more complex model [29] and also for a similar reason in [30]. Another formulation used in the ROAMS environment developed in JPL for a wide range of rover systems is the Hunt-Crossley contact force:

$$f_z = kh^n + \gamma \dot{h} h^n. \tag{2.2}$$

To calculate the sinkage h, the lowest point on the wheel along the gravity axis is compared to the terrain plane. This is reasonable for contact models, which only relate the normal force to sinkage without considering other dynamic variables. It can also be noted that in most literature, sinkage is marked by z. The author decided to forgo this to avoid further overlaps in the notation with the coordinate z.

2.1.2 Bekker-Wong Model

The semi-empiric pressure-sinkage relation proposed by Bekker in the '60s for ground vehicles with a wheel width of b

$$\sigma(h) = \left(\frac{k_c}{b} + k_\phi \right) h^n, \tag{2.3}$$

is now largely adapted for planetary rovers. Bekker aimed to characterize the terrain's bearing capacity by parameterizing the stresses observed under a plate. The pressure-sinkage modules k_c and k_ϕ are to be derived from bevameter test on the soil as done by Bekker. For a comprehensive overview of modifications proposed to the equation (2.3), the author recommends [31].

The equation (2.3) was reformulated using radial coordinates by Bekker as a normal stress distribution along the cylindrical wheel area in contact with the terrain by considering

$$h = r(\cos\theta - \cos\theta_1), \qquad 0 \leq \theta \leq \theta_1, \tag{2.4}$$

where r is the wheel radius and θ_1 is the so-called entrance angle – the angle between the direction of gravity and the line connecting the wheel center and the forward contact point. The

stress calculated at any h by these formulas is considered to act along the radial direction of the wheel. These are depicted in Figure 2.1. According to these two equations, on level terrain, the stress is maximal at the very bottom of the wheel, where it then abruptly turns to zero, as contact between the wheel and soil is considered broken with $\theta < 0$.

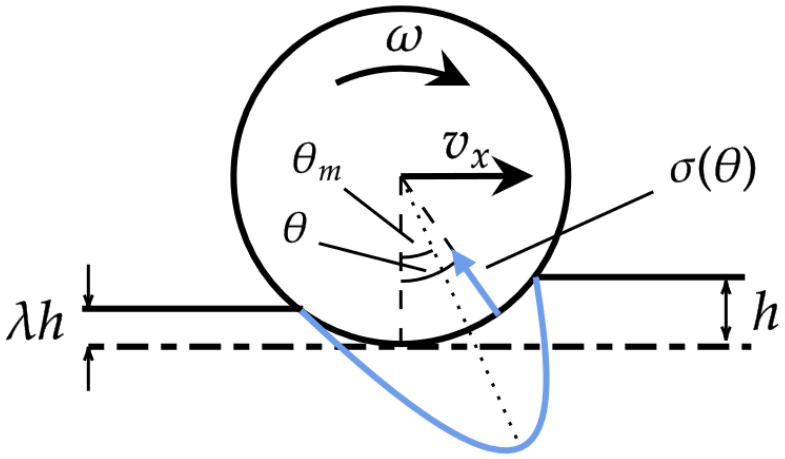

Figure 2.1. *Illustration of some of the important quantities in the Bekker wheel stress model as modified by Wong. The arrow noted v_x is here to show the direction of the x-component of the wheel's linear velocity, as other components aren't necessarily zero.*

In 1967 Wong and Reece demonstrated that the maximum normal stress occurs at the point where the two soil failure zones beneath the wheel join each other [32]. An illustration of this is given in Figure 2.1 with a dotted line. After their work, the equation (2.4) is instead given as:

$$\begin{cases} h = r(\cos\theta - \cos\theta_1), & \theta_m \leq \theta \leq \theta_1, \\ h = r(\cos\left(\theta_1 - \frac{\theta-\theta_2}{\theta_m-\theta_2}(\theta_1 - \theta_m)\right) - \cos\theta_1), & \theta_2 \leq \theta \leq \theta_m. \end{cases} \tag{2.5}$$

Here the notion of h representing sinkage loses some of its meaning. Wong and Reece also proposed an equation for estimating the maximum stress angle θ_m based on their observation of experimental results:

$$\theta_m = (a_1 + a_2 s)\theta_1. \tag{2.6}$$

Here the slip ratio s was introduced. Its definition is given below in equation (2.13). Along with this equation, some suggestions for the values of the tuning parameters a_1 and a_2 were also given in [32]. In the equation (2.5), the exit angle θ_2 was also introduced. In a static case, when the wheel is not moving, $\theta_2 = \theta_1$. In his work, Bekker set $\theta_2 = 0$. This was also later done in [33, 34], for example. They argued the soil has no rebound, and thus no pressure is generated beyond the point directly underneath the wheel center. However, Ishigami et al. instead used the formula [35]

$$\theta_2 = \arccos\left(1 - \lambda\frac{h}{r}\right), \tag{2.7}$$

where λ is yet another parameter that could be derived from experiments. The effect of λ is shown in Figure 2.1. In their work, setting the parameter $\lambda \in [0.9, 1.1]$ provided a satisfactory match between theory and experiment. This range for the parameter was determined via visual estimation. While this usually isn't a topic of much discussion, the discrepancy in the exit angle estimation here is worth noting, especially as [35] is one of the most cited models in recent terramechanics literature. Although it seems the use of λ was not always picked up in later works.

As was observed in experiments in [36], a wheel in motion produces a deeper sinkage than a static wheel. This is called slip sinkage. When taken into account, it is modeled by modifying the sinkage exponent:

$$n = n_0 + |s| n_1. \tag{2.8}$$

This relation is largely attributed to the authors of [37], who worked on predicting the slip behavior of a lunar rover. While they didn't offer any analytical reasoning in this nor the following paper [38], their overall model predicted single-wheel testbed experimental results well. Others soon adapted this, for example, the ARTEMIS environment as described in [29].

2.2 Shearing Forces

The shear stress-displacement relationship in the longitudinal direction is most often expressed as:

$$\tau_x(\theta) = (c + \sigma(\theta) \tan \phi) \left(1 - \exp\left(-\frac{j_x}{K_x}\right)\right) \tag{2.9}$$

where τ is the shear stress, j_x is the shear displacement, with units of length, c and ϕ are the cohesion stress and the angle of internal friction of the terrain, respectively, and K_x is referred to as the shear deformation modulus. Equation (2.9), proposed by Janosi and Hanamoto in [39], is a simplified form of a relation suggested by Bekker [40]. The shear displacement can be calculated as follows:

$$j_x(\theta) = \int_0^t v_r \mathrm{d}t = r\left(\theta_1 - \theta - (1 - s)(\sin \theta_1 - \sin \theta)\right). \tag{2.10}$$

The value v_r is used to denote the relative linear velocity of the lowest point of the wheel with respect to the ground. The combination of equations (2.3) and (2.5) for the normal stress and (2.9) for the longitudinal stress is often referred to as the Bekker or the Bekker-Wong model. Similarly to τ_x, a stress-shear relation is also used for the lateral direction

$$\tau_y(\theta) = (c + \sigma(\theta) \tan \phi) \left(1 - \exp\left(-\frac{j_y}{K_y}\right)\right) \tag{2.11}$$

and

$$j_y(\theta) = r(1 - s)(\theta_1 - \theta) \tan \beta. \tag{2.12}$$

In Figure 2.2, these stresses are indicated. The figure also includes the forces induced by said stresses. The forces as used in the simulation, however, are discussed in section 3.2.2

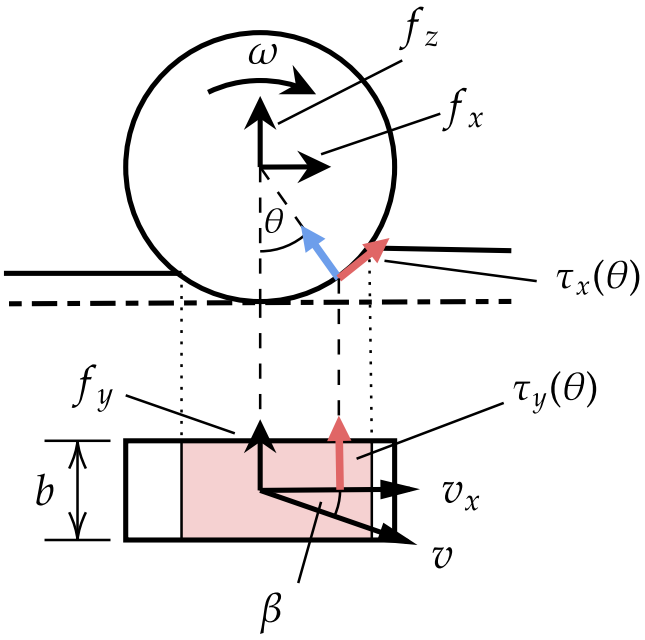

Figure 2.2. *Illustration of forces produced by the stresses σ and τ.*

The aforementioned slip ratio is calculated as

$$s = \begin{cases} \frac{r\omega - v_x}{r\omega}, & \text{when } |r\omega| > v_x \\ \frac{r\omega - v_x}{v_x}, & \text{when } |r\omega| < v_x \end{cases} \tag{2.13}$$

while the slip angle β is calculated as

$$\beta = \arctan \frac{v_y}{v_x}. \tag{2.14}$$

Values v_x and v_y refer to the velocity components of the wheel center. The slip angle isn't included in terramechanics based analysis often. It was first used in [35], where the steering behavior was simulated. It is common in similar papers to only investigate longitudinal motion, and the lateral soil shearing forces are often neglected. Consequently, no widely established model exists for such forces. The lateral stress distribution shown in equation (2.11) was also first used and validated in [35]. Although, as is apparent, both stresses τ_x and τ_y were formulated together; just the latter does not get used as often. While they conclude the model is validated, the theoretical predictions for the forward pulling force on the wheel are not affected by the slip angle nearly as much as the experiment shows. This is to say, when it comes to modeling steering, there is still room for improvement.

2.3 Other Effects

The lug pattern on a wheel can play an important role in the overall tractive performance. Often the added traction of the lugs is modeled by just increasing the wheel radius by the lug height. As discussed by Ding in [33], this is reasonable for lugs with smaller heights and larger cross-sectional areas. This, however, often isn't true for planetary rovers, which have generally opted for the opposite [41]. So a more complex formulation of the force acting on each grouser has been adapted. As it is also heavily parameterized and won't be discussed in the thesis much further, it is not explicitly written out. Its use in ARTEMIS is briefly but concisely given in [42]. Another consideration is that the wheel lugs enter and leave the soil periodically. In the measurement data – including the wheel sinkage, normal force, drawbar pull, and driving torque – sinusoidal fluctuations can be seen [43]. However, this phenomenon hasn't really gotten attention from other authors, likely because artifacts of the defect don't seem to persist after averaging.

The bulldozing force is also sometimes considered, mainly when lateral forces are being investigated, as was done in [35]. These are forces exerted on the side walls of the wheel by the soil. While it was initially formulated as a part of the resistive forces in the wheels' driving direction, this was never validated nor widely used. In [42], authors ignored this as the bulldozing of soil was not seen in the imagery of the Spirit, Opportunity, or Curiosity. However, in the same work, they did use it for forces in the lateral direction. In these two articles, [35] and [42], slightly different equations were used for the bulldozing forces in the lateral direction. Both include several new parameters. It was decided not to consider this force in ULYSSES for the time being, as the correctness of these models is unclear.

3. Simulation Environment

3.1 Selection of a Physics Engine

3.1.1 Problems with Unreal Engine 5

The aim of this thesis is to advance the physical behavior of a rover in the simulation environment ULYSSES, described more in Appendix A and in [4]. It is being developed using Unreal Engine 5 (UE5). The game development software, or a game engine, has incorporated a sizeable and new physics system named Chaos Physics. However, it suffers from comparatively low accuracy and numerical instability. This isn't surprising, considering it was released in the spring of 2021. For comparison, other well-known open-source engines like Bullet Physics engine [44], Open Dynamics Engine [45], and PhysX[1] [46] have been refined over decades.

In the early development of ULYSSES, some of these problems started to affect the output. For example, the rover's behavior on slopes was even on visual inspection unrealistic. The wheels would sometimes slide down a crater wall. Also, when running a long traversal, sometimes the rover would 'tumble' at certain locations, while sometimes it would not. It was very hard to predict when this might happen. There were also questions from the wider game development community regarding the trustworthiness of simulations supported by Chaos.

Although no academic data yet exists comparing Chaos to other physics engines, the decision was made to opt for a more widely-known physics framework for ULYSSES. There is a substantial amount of data from individuals pointing out issues with the new physics system in UE5, particularly when compared to the engine used in previous UE versions. The primary concern is its speed, and there are indications that contact resolution may fail under stress.

Another factor influencing the decision is that other open-source physics engines are more easily modifiable. While UE5 itself is open-source and freely available on GitHub, its physics system is closely integrated with the rest of the engine and lacks comprehensive documentation. Consequently, making changes or understanding its inner workings becomes a time-consuming process.

However, the visualization is still performed using Unreal Engine 5. This vast framework

[1] This physics engine was used in previous Unreal Engine versions.

incorporates numerous other highly advanced and convenient tools and systems ranging from ray tracing to mesh handling and texture manipulation. These other components of the software make it a superb tool for real-time visual output, and ULYSSES has already demonstrated its potential as an effective tool for generating synthetic data for developing and testing visual odometry methods [47]. As such, the simplest way to overcome the difficulties with the novel Chaos physics solver is to simply investigate swapping out the physics engine for more time-tested alternatives.

3.1.2 Discussion on the Selection Process

Most of the simulation environments designed for engineering purposes, such as Vortex and MCS-Adams are not open-source. This would significantly complicate their integration into ULYSSES. On the other hand, there are several open-source rigid body dynamics libraries tailored more toward game-making and animation. However, gaming engines are known for prioritizing stability and speed over physical accuracy, often resorting to artificial damping or even disregarding Coriolis forces, as do Havok and PhysX [48]. It is generally acknowledged that no single engine excels in all aspects, like collision detection, contact resolution, and physical accuracy. Therefore, some consideration is required to select an appropriate engine for ULYSSES.

For this thesis, the Bullet Physics engine was selected as the framework for rigid body physics. Before settling on a specific physics engine, various options were compared using existing literature. An essential criterion was that the chosen engine should be open-source. Additionally, the pick needed to have a reasonable amount of documentation and support. Three options were seriously considered: Bullet Physics engine, Open Dynamics Engine, and PhysX. The latter was disregarded early due to the example as mentioned above of preference for speed. All of these, like most physics engines, are written in C++, which eases their integration into an Unreal Engine project.

In 2007 Boeing et al. [49] performed several tests to probe the performance of various physics engines. They concluded that while no physics engine is the best at everything, the Bullet engine "provided the best results overall, outperforming even some of the commercial engines". From the results presented, it seems Bullet is the most accurate at constraint and contact resolving. Even though, at the time, Bullet was a relatively new package. Bullet was also shown to rather accurately predict the motion of tangram pieces on a surface when pushed [50]. Although the authors noted this choice of examining the Bullet engine was mainly due to the Boeing article and should be taken as a representation of most available physics engines. In 2012 Hummel et al. [51] evaluated open-source engines similarly to Boeing. They concluded that their current (or at the time) choice of Bullet for an assembly simulation could, in theory, be replaced by PhysX or

Newton Dynamics as they performed similarly. Considering they all use similar paradigms, it is not surprising the results are also similar. For example, the Dantzig algorithm implementation in Bullet for solving the LCP is taken from ODE source code [52]. While Bullet also includes an implementation of Lemkes's algorithm, the former seemed more capable when tested and hence was used in this work.

The simulation of a lowly rover doesn't demand much from a rigid body physics engine – there is a relatively low number of contacts, and usually, the motion state is slow-changing. Of course, the rover articulation can be made very complex, but still, all of the aforementioned engines should be capable of simulating it to a reasonable accuracy. However, these engines are for rigid body dynamics meaning the rigid body model itself isn't accurate enough for certain scenarios. This is to say, the choice between the engines in this specific case isn't likely a big factor in the physical accuracy of the simulation, as the inaccuracies will arise from aspects none of them are addressing anyway.

Due to this consideration, the main decisive factor in the choice of Bullet Physics Engine was that it is deterministic: given the same set of input parameters, the simulation output will always be the same. In [53], this topic was analyzed in the context of driving simulators, including possible sources of non-determinism.

One might argue that adding some variance to the system might be physically more accurate at this level of abstraction. The problem under consideration is, to a certain extent, non-deterministic. The exact amount of slip during a rover traversal, for example, can vary due to a vast amount of reasons that just can't be taken into account. However, this variation should still be controlled for better statistical analysis. Hence it is preferable to use an underlying framework that behaves deterministically, and the nondeterministic behavior can be added on top to better asses the effects if deemed necessary.

This creates somewhat of an annoyance. To guarantee determinism, the simulation time step has to be fixed. Unreal Engine 5, which takes care of the rendering during run-time, can not keep the time step constant. This isn't just a characteristic of UE but of real-time systems in general. As no good solution to this exists, the visual simulation speed will depend on the frame rate. This will not affect the numerical output, nor will it be very noticeable to the eye, as UE can keep the framerate somewhat steady. The variation seems to be around 1% of the time step with a framerate of 60 Hz.

3.1.3 Integrating Bullet Physics Engine into a UE5 Project

Integrating the Bullet Physics engine into an Unreal Engine project has been done before. For example, the game Rocket League did it with UE3, meaning they switched from PhysX. They cited a need for deterministic behavior as the reason. As this thesis is more focused on the semi-empirical modeling of a wheel, the description of combining Bullet and Unreal Engine is described in Appendix B.

3.2 Semi-empirical Models in a Rigid Body Dynamics Environment

In this section, the calculations of the forces and torques acting on the wheel are discussed by applying the formulas from the previous chapter. The relations presented in the previous section were developed for predicting wheel dynamics for a given static input. When applying these same formulas in a fully dynamic environment, some consideration has to be taken.

The Bekker model includes some underlying assumptions which complicate its usage in a dynamic environment. These assumptions include:

1. uniformity of the normal pressure $\sigma(\theta)$ and shear stress $\tau_x(\theta)$ between the tire and terrain along the wheel's width,
2. the velocity and slip of the wheel are constant,
3. the wheel is moving parallel to flat ground.

These assumptions impose limitations when applying this method to non-steady-state vehicle dynamic mobility simulations. Assumption 1. is also incorporated into simulations – firstly to simplify, but also as the random variations these distributions could take on have really not been analyzed on the macro-scale neither via experiment nor higher fidelity simulations [25]. Assumption 2., on the other hand, can gravely affect the stability of the simulation. When any of the velocities, ω or v_x, approaches zero, the slip becomes ill-defined. In this work, undefined values are largely avoided by monitoring the state of a wheel, like whether has it settled into the soil before any calculations involving slip are taken on. This is also why the dynamic use of these models should be validated, even though the models have been validated in a static variant with non-changing input parameters. The third assumption can be tackled in various ways. The approach taken in this work is described in the next subsection.

3.2.1 Contact Generation

Before getting to the forces, the contact area between the wheel and the ground should be determined. In this work, a wheel is modeled as a cylinder. This is the usual approach, as the

Bekker normal stress distribution (2.3) was derived for such a case. The other approach is to represent the wheel as a mesh. For example, the German Aerospace Center (DLR) has been developing a soil contact model (SCM), first presented in [54], which models a wheel as a mesh. However, this is largely done to calculate the deformation of the soil. The forces acting on the wheel in the SCM are not strongly affected by this difference – the force models are developed and validated for cylindrical wheels.

The contact estimation used in this work is inspired by the recent publication [55]. Here it is useful to define the wheel reference frame Σ_w. It is set so that the wheel's axis of rotation, the y-axis, is along its axle, z-axis is perpendicular to the wheel-ground contact plane, i.e., along its normal and x completes a right-hand coordinate system. The origin is in the center of the cylinder. Figure 3.2 shows the x, y and z coordinate axes as red, green and blue, respectively.

The ground is represented as a digital elevation map (DEM) with a constant difference between neighboring vertexes in both x and y axes. This knowledge was exploited when calculating the contact points. To estimate the contact plane, three points of contact $\mathbf{P_i}$ are picked: two at the edges of the front side of the wheel and one on the backside on the center line of the wheel circumference. The exact location, or rather a good initial guess, of these points is estimated by using the entrance angle θ_1 from the previous timestep. In the wheel coordinate system, these points are expressed as

$$\mathbf{P_1}^w = \begin{pmatrix} r\sin\theta_1 & \frac{b}{2} & -r\cos\theta_1 \end{pmatrix}, \tag{3.1}$$

$$\mathbf{P_2}^w = \begin{pmatrix} r\sin\theta_1 & -\frac{b}{2} & -r\cos\theta_1 \end{pmatrix}, \tag{3.2}$$

$$\mathbf{P_3}^w = \begin{pmatrix} -r\sin\theta_1 & 0 & -r\cos\theta_1 \end{pmatrix}. \tag{3.3}$$

These points are then rotated into the global coordinate system: $\Sigma_w \rightarrow \Sigma$. For each point, bilinear interpolation can then be performed to calculate the DEM height value based on the four neighboring vertices. When the underlying DEM is dense, using the closest of the four to estimate the new height value for the point $\mathbf{P_i}$ is easier. Both were implemented in this work. The contact plane unit normal $\mathbf{v}$ is calculated based on these three points. Also, the sinkage h can be calculated as shown in Figure 3.2.

It should be noted that the transformation $\Sigma_w \rightarrow \Sigma$ is based on the wheel reference frame established during the previous timestep. Defining Σ_w through the contact plane while also using this reference frame to calculate the contact plane creates a bit of a chicken and an egg problem. In the article [55], the wheel z axis was set to be "up and perpendicular to the wheel velocity" instead. This was initially tested in ULYSSES, but the wheel's velocity component in the normal direction became, at times, too prominent when total velocity was low. It is also unclear what

"up" exactly means. However, the chicken and egg situation converges nicely to the appropriate plane, as the timesteps and velocity are small.

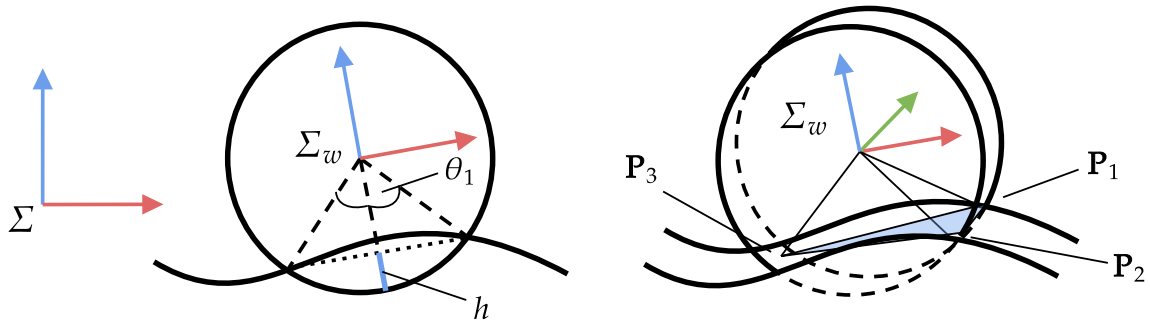

Figure 3.1. *Illustration of wheel-terrain contact and establishment of reference frames. An indication is given of how the entrance angle θ_1 is used to estimate the three contact points used to calculate the contact plane normal and how the sinkage h is calculated.*

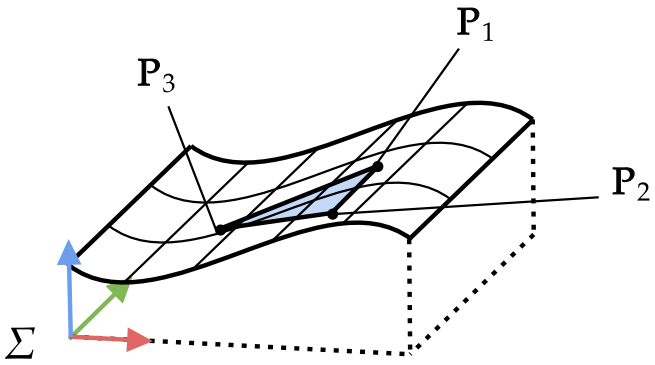

Figure 3.2. *Illustration of a DEM and an estimation for the wheel-terrain contact plane. The gird of the DEM depicted here is far more sparse compared to the distance between the points $\mathbf{P}_\mathrm{i}$ than the DEMs used in this work.*

3.2.2 Numerical Force Calculations

The Bekker model-based wheel-terrain contact model is implemented in the Bullet Physics engine by deactivating its internal collision detection between the wheel and the terrain. Instead, external forces are applied to the wheel's center of mass, as is the resisting torque. The magnitudes of the normal force f_z, drawbar pull (consisting of motion resistance and thrust) f_x, lateral shear force

f_y and resistive torque T can be calculated as

$$f_z = br \int_{\theta_2}^{\theta_1} (\sigma(\theta)\cos\theta + \tau_x(\theta)\sin\theta)\,\mathrm{d}\theta + \gamma\dot{h}, \tag{3.4}$$

$$f_x = br \int_{\theta_2}^{\theta_1} (\tau_x(\theta)\cos\theta - \sigma(\theta)\sin\theta)\,\mathrm{d}\theta, \tag{3.5}$$

$$f_y = br \int_{\theta_2}^{\theta_1} \tau_y(\theta)\mathrm{d}\theta, \tag{3.6}$$

$$T = br^2 \int_{\theta_2}^{\theta_1} \tau_x(\theta)\mathrm{d}\theta. \tag{3.7}$$

For calculating the normal force f_z, a dampening term $\gamma\dot{h}$ was added. This term helps the wheel to settle into the ground as in ULYSSES; it might not initially be located such that the forces pulling it down, or the load W, are equal to the normal force. This would result in oscillations in the normal directions, as the normal stress σ and hence the normal force f_z describes a non-linear spring. In [56], the author explains this as a shortcoming of the Bekker model when applied to a dynamical environment: the energy loss in the normal direction doesn't get accounted for.

As integrals (3.4)-(3.7) do not have analytic solutions, numerical integration is performed at every simulation step. In various implementations, this is done over the angle θ. When using an uneven terrain model, this integration can be done in slices effectively also integrating over the wheel width so that the entry angle θ_1 may vary over the wheel front. As this integration has to be performed at least once every time step to reduce computation time, it can be substituted with a simplified closed-form formula derived by Ding in [57]. But in this work, the author decided to include the integration as with newer computers, such calculations aren't very cumbersome.

To avoid forces blowing up, for lack of a better word, every simulation starts with a settling period. The wheel doesn't start moving forward before the oscillations in the normal direction have been attenuated.

3.3 Validation of the Dynamic Calculations

The terramechanics models implemented in ULYSSES have been previously validated by experiment. Direct qualitative analysis usually compares the theoretical predictions for constant slip, equating f_z to the wheel's load to measurements gathered from a single-wheel testbed. What remains to be validated is the use of the model in a dynamic simulation environment. More specifically, its use in ULYSSES as a similar model has been combined with ODE [35]. This is done by studying a single wheel with an imposed slip in the simulation. As this thesis presents preliminary work on this topic in Tartu Observatory, such experimental facilities are not accessible. However, this is a hot research topic, so a lack of data isn't necessarily a problem.

Implementing the discussed models in ULYSSES is compared to experimental data in [35, 36]. The paper [35] by Ishigami et al. was chosen as it provides a comparatively thorough description of both the experiments and calculations they performed and also as it is one of the foundational papers in the field of terramechanics. It is also one of the only, if not the only, paper that gathers experimental data to validate lateral force calculations. The paper [36] by Ding et al. was also chosen due to its thoroughness. They reported experiments performed with wheels of varying sizes and lug numbers.

4. Results and Discussion

In the field of terramechanics, single-wheel tests are common. This test set usually involves a system for controlling the wheel's angular and linear velocity independently of each other. It will also include sensors to measure the forces and driving torque the contact induces. As a result, it is common to see plots of force versus slip ratio. Such plots are used below to validate the implemented wheel-terrain contact modeling in ULYSSES qualitatively. However, a wheeled vehicle system will also be simulated and analyzed as it will eventually be used for simulating rover traversals.

In the simulations, the general forward dynamics, most importantly for the vehicle model, were solved by the Bullet physics engine. The time step was set to one 600th of a second, and the integration step for the angle θ when calculating wheel stresses was set to 0.01 rad.

Parameter	Value from [35]	Value from [36]	Unit
k_c	1.37	15.6	kPa/m^{n-1}
k_ϕ	814	2407.4	kPa/m^n
n_0	1	1.1	
n_1	-	-	
a_1	0.4	-	
a_2	0.15	-	
c	0.8	0.25	kPa
ϕ	37.2	31.9	deg
K_x	43β + 36	9.7-13.1	mm
K_y	20β + 13	-	mm
λ	0.9-1.1	0	

Table 4.1. Lunar soil simulant parameters.

4.1 Single-wheel Travelling

Firstly, single-wheel forced slip traveling was analyzed in ULYSSES as an analog to the single-wheel testbed experiments. These results were then plotted for comparison with the data from [35] and [36]. The soil parameters from each article are listed in Table 4.1. The estimates given by Ding et al. were derived in a separate article [57] by the same authors by fitting theoretical models to experiment data. As the article didn't report any values for a_1 and a_2, the values from Ishigami's work were used. The wheel dimensions were $r = 9$ cm, $b = 11$ cm, and $r = 15.735$

cm, $b = 16.5$ cm, in experiments presented in either article, respectively.

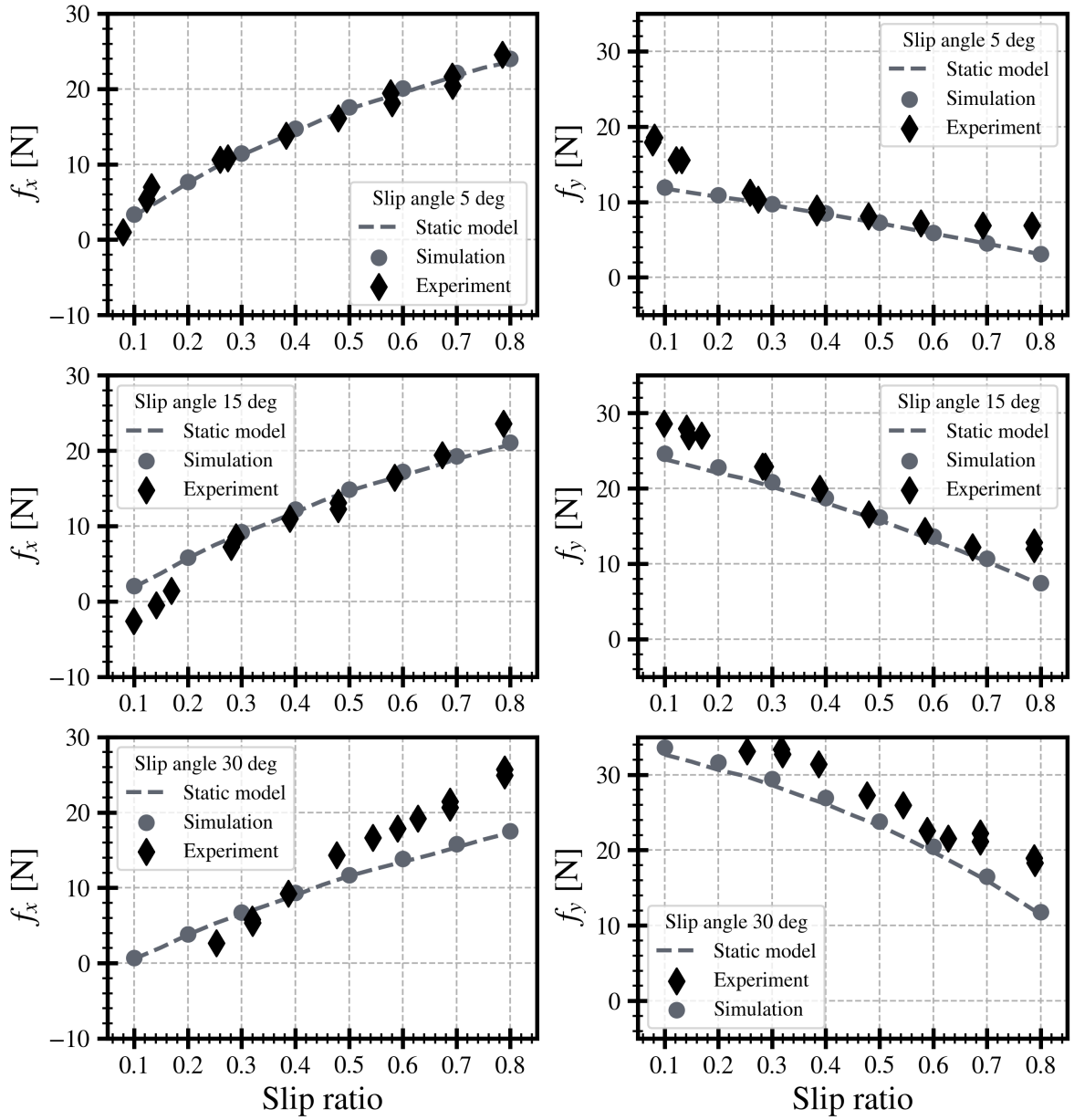

Figure 4.1. *Comparison of calculated forces for a forced slip wheel and forces observed by Ishigami et al. in a single-wheel testbed. The forces in the longitudinal direction are in the left column, and the forces in the lateral direction of the wheel are in the right column. The slip angle increases row-by-row.*

Figure 4.1 compares the calculated forces f_x and f_y to experimental data from Ishigami's work. The graphs also include a line labeled "Static model". This line was calculated separately with a Python script. The normal force, without the dampening term, was set to equal the wheel's load for a given slip ratio. From this equation, the entrance angle is estimated and then used to calculate the other forces. This line is used as an indication that the model implementation in ULYSSES does not deviate too much from the predictions of the original formulas due to invalid

entrance angle estimations or the dampening, for example.

Generally, the simulation data seems to correlate with the experiment rather well. Although, as the slip angle increases, the slope of the simulation data seems to fit the experimental f_x force less and less. The theoretical line in the last row of the left column is not nearly steep enough. Although the same can be seen in [35], the authors didn't discuss it. From this, one could hypothesize that the slip angle should be accounted for more in calculating stresses. It seems research on this specific observation is lacking.

As for the lateral force, f_y, its slope seems to be in rather good accordance with the experiment data. However, now there seems to be a constant offset: the predicted forces are slightly lower than what was observed. In the original article, the authors explained this by adding the bulldozing force. While the effect isn't big, it might be helpful to consider this addition to ULYSSES. Although, this will also depend on what kind of wheels are to be considered, as the bulldozing force acts on the side walls of the wheel.

A similar comparison with the data from Ding et al. [36] is shown in Figure 4.2. In addition to f_x, they also gathered data on the driving torque T and the sinkage h while not considering the slip angle and the lateral force f_y. Curiously, when looking at the graph for f_x on the left side, the model does not fit the experimental data nearly as well. The experimental data plateaus after the slip ratio of 0.1, while the predicted values, given in gray, seem to do the same at around 0.5 to 0.6, with the plateau value being almost 10 N higher.

Another obvious discrepancy is visible in the plot of slip ratio versus sinkage in the left column. The gray line, corresponding to setting $n_1 = 0$, stays constant, while in the experimental data, the sinkage is roughly linearly dependent on the slip ratio. This is exactly why the parameter n_1 was later proposed. To calculate the data for the blue line on the graphs in Figure 4.2, this parameter was set to 0.5 in combination with lowering n_0 from 1.1 to 1. The sinkage-slip ratio relation is explained much better by this parametrization. However, the effect for the force f_x and torque T is the opposite, albeit less noticeable. Such coupling between the predicted values makes the process of finding the correct parameterization difficult [57].

Coming back to the discrepancy in predicting f_x correctly. In the article by Ishigami et al., it is not stated explicitly whether the height of the lugs was also included in the stated wheel radius. However, as in the past, using a larger radius for calculations has been a method of accounting for lugs, it is likely they did the same. From the images that were included, it seems the test rover definitely had lugs. So it is possible the model describes a lugged wheel slightly better, or in other words, it overestimates the tractive force of a wheel with no lugs and the same radius.

The data gathered by Ding et al. also includes measurements for the same wheel with added lugs. So the simulation was repeated with a wheel with $r = 16.735$ cm to also account for 10 mm lug height and the dynamic sinkage parameter set to $n_1 = 0.5$. The experimental data given for comparison is of the same wheel with 10 mm lugs attached to it. These results are in the right column of the figure.

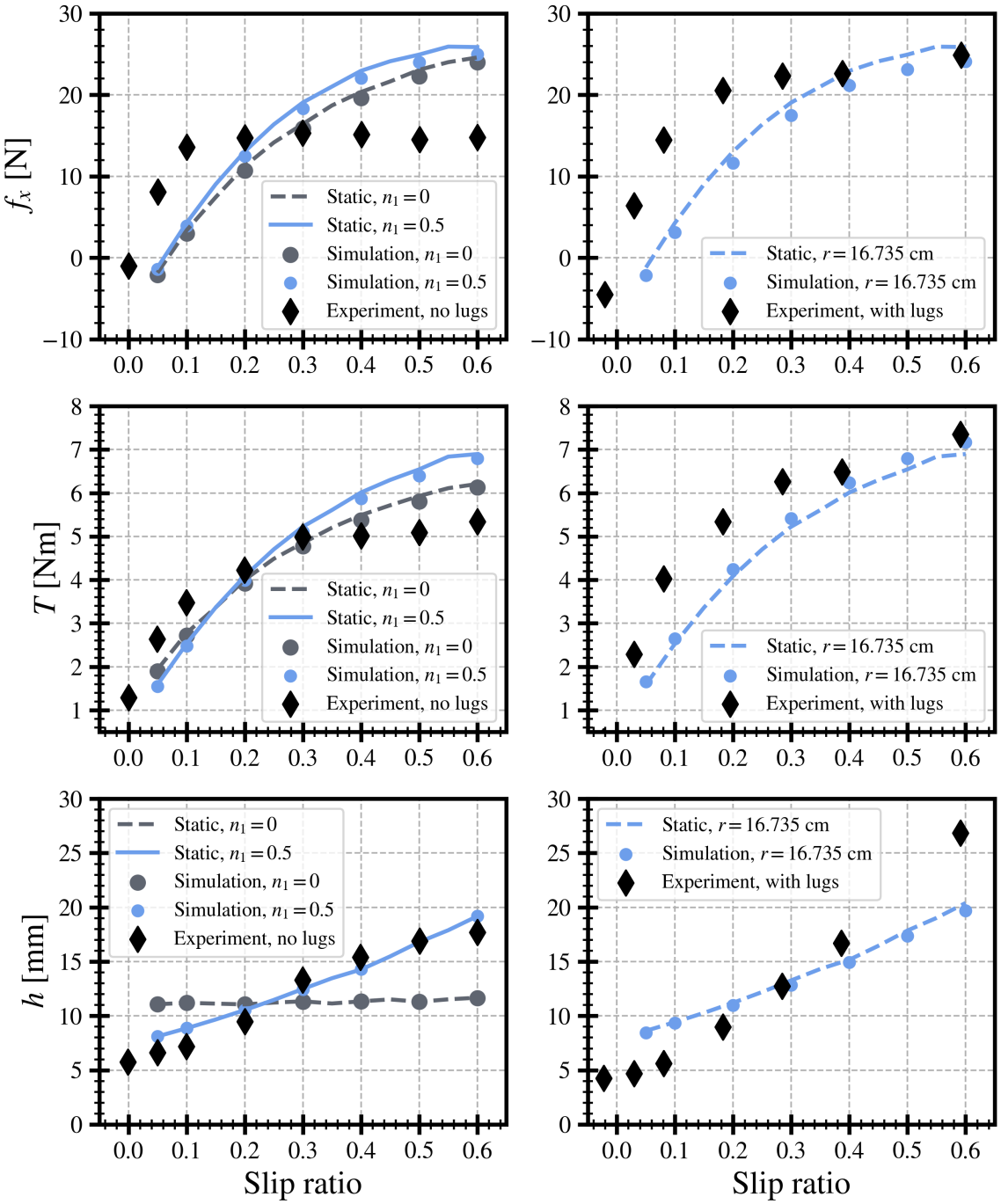

Figure 4.2. *In the left column, results from a wheel with no lugs are compared to simulation results with two different parameterizations. On the right, data from the same wheel with 10 mm lugs is compared to simulation results with an increased wheel radius.*

Qualitatively, both the drawbar pull f_x and resistive torque T seem to fit the experimental data slightly better, at least for higher slip ratios. Overall, it is hard to say with certainty what is the correct approach here. More analysis is needed into parameter sensitivity and estimation.

4.2 Wheeled-vehicle Manuvering

To see how this implementation of contact forces might affect a rover, again, data from Ishigami was used. They gave a thorough description of the model rover: the location and type of joints and also the masses of each component included in the dynamics model. There seemed to be some inconsistency with the stated moments of inertia, though. However, this likely does not affect the results much.

Figure 4.3 shows the recreated rover in ULYSSES. The visuals are rendered by Unreal Engine 5, but the dynamics is handled by the Bullet physics engine. All joints were modeled as hinge-type joints. So a wheel is only allowed one degree of freedom with respect to the steering block it is attached to with the point of comparison in the center of the wheel center.

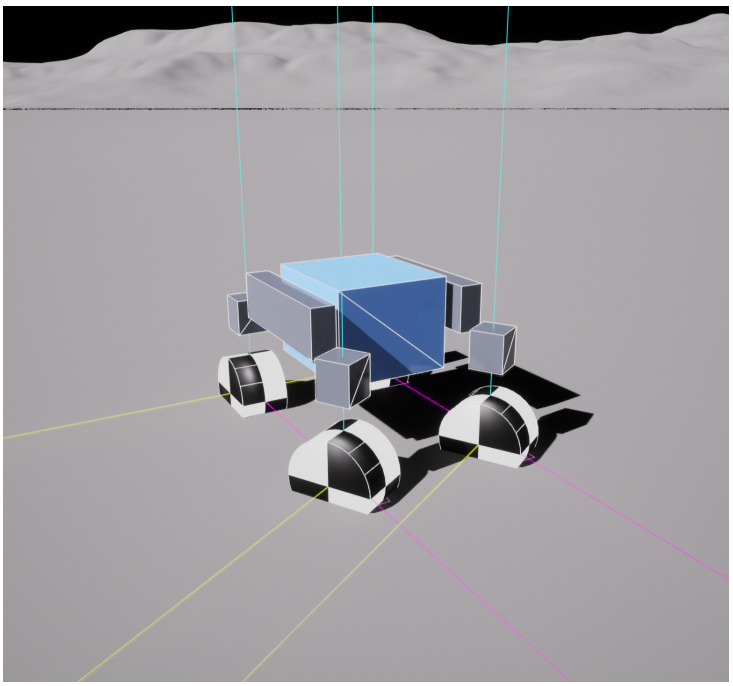

Figure 4.3. *The rover used for the multi-body test visualized in Unreal Engine. The steering angle of the two front wheels has been set to 30 degrees. The faint cyan, yellow and magenta lines originating from the wheel centers correspond to the wheel reference frame as set in Bullet.*

In the experiment, each of the rover wheels was controlled to travel with a constant angular velocity. Two different steering angles of 15 and 30 degrees were tested. During the simulation, each rover wheel was also controlled with a PID controller, and the steering joints of the two

front wheels were fixed to 15 and 30 degrees, respectively.

The trajectory of the rover body is plotted as a gray line against the experiment data in Figure 4.4. The simulation was also repeated with a rigid body contact model. This is plotted with a light blue line. Again the difference observed between the simulation using the Bekker model and the experiment might, at least partially, be explained by the missing bulldozing forces. Still, the Bekker model seems to be a step in the right direction when also compared to the rigid body contact model.

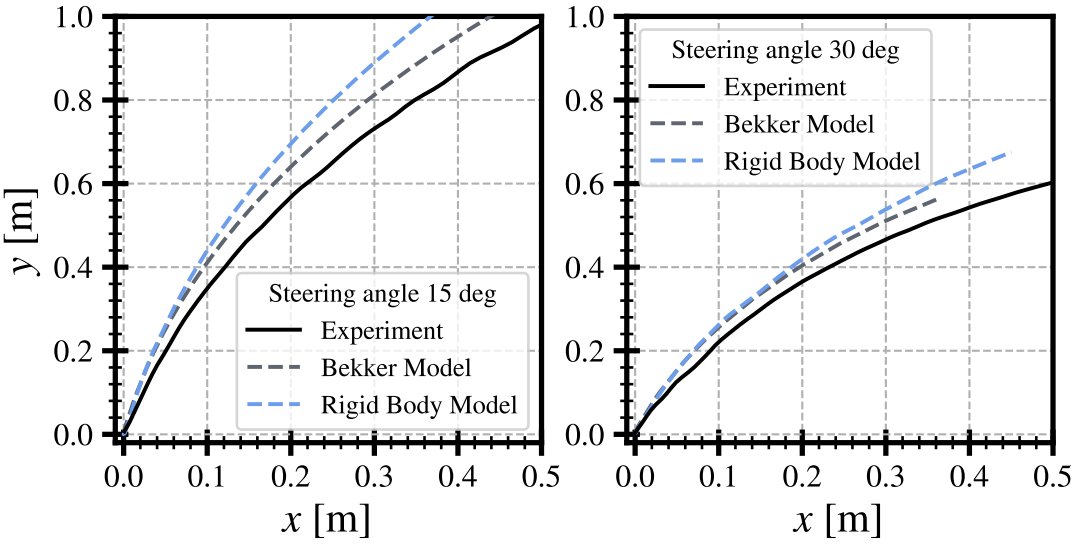

Figure 4.4. *Trajectory of the rover from Ishigami et al. and simulated trajectory of the same experiment with two different wheel-terrain contact models.*

Conclusions

The goal of this thesis was to improve the physical accuracy of rover simulations in ULYSSES. This was achieved by supplementing the rigid body contact model with one adapted from terramechanics. To better understand the work done and decisions that were made, an overview was given of both rigid body dynamics and simulation and the basics of terramechanics.

The first chapter covered rigid body dynamics and constraints. Subsequently, the discussion expanded to encompass the anatomy of physics engines, serving as the backbone of simulations. Later the decision to integrate the Bullet Physics engine into ULYSSES was discussed: Bullet is open-source and reasonably accurate. It is also deterministic, which sets it apart from other engines. Some of the technical intricacies related to the integration process were elaborated upon in Appendix B.

In the second chapter, an overview was given of semi-empiric methods used for modeling and analyzing wheel-terrain contact. In the next chapter, this knowledge was used to calculate realistic contact forces acting on a rover's wheel. While this adaptation was generally quite straightforward, some consideration had to go into making these models stable in a dynamic environment.

In the last chapter, the work was validated by comparing forces calculated during a simulation to experimental data from previously published sources. this comparison was done both with single-wheel experiments and a 4-wheeled rover. The results from the single wheel tests, when compared to Ishigami et al. [35] seemed excellent. However, a similar comparison to data from Ding et al. [36] revealed flaws in the model. In further works, parameter estimation should be investigated. The wheeled-vehicle maneuvering indicated the superiority of the implemented model over the simplistic rigid body contact model.

Bibliography

[1] A. Thoesen and H. Marvi, "Planetary surface mobility and exploration: A review," *Current Robotics Reports*, vol. 2, no. 3, pp. 239–249, 2021.

[2] M. Allan, U. Wong, P. M. Furlong, A. Rogg, S. McMichael, T. Welsh, I. Chen, S. Peters, B. Gerkey, M. Quigley, *et al.*, "Planetary rover simulation for lunar exploration missions," in *2019 IEEE Aerospace Conference*, pp. 1–19, IEEE, 2019.

[3] J. B. Johnson, A. V. Kulchitsky, P. Duvoy, K. Iagnemma, C. Senatore, R. E. Arvidson, and J. Moore, "Discrete element method simulations of mars exploration rover wheel performance," *Journal of Terramechanics*, vol. 62, pp. 31–40, 2015.

[4] H. Teras, Q. S. Islam, K. Kruuse, and M. Pajusalu, "Ulysses - a state of the art sandbox simulator for planetary surfaces," in *73rd International Astronautical Congress (IAC)*, IAF, 2022.

[5] Epic Games, "Unreal engine."

[6] M. Pajusalu, Q. S. Islam, H. Teras, K. Kruuse, R. Avarmaa, A. Olesk, K. Mikkor, S. Latt, J. Press, and M. Noorma, "Large scale mobility on the moon by transferring terrestrial autonomy capabilities," in *73rd International Astronautical Congress (IAC)*, IAF, 2022.

[7] H. Goldstein, C. Poole, and J. Safko, "Classical mechanics," 2002.

[8] W. Alan and W. Mark, *Advanced animation and rendering techniques*. 1992.

[9] W. R. Hamilton, *Elements of quaternions*. Longmans, Green, & Company, 1866.

[10] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pp. 245–254, 1985.

[11] J. D. Foley, *Computer graphics: principles and practice*, vol. 12110. Addison-Wesley Professional, 1996.

[12] B. V. Mirtich, *Impulse-based dynamic simulation of rigid body systems*. University of California, Berkeley, 1996.

[13] M. Müller, M. Macklin, N. Chentanez, S. Jeschke, and T.-Y. Kim, "Detailed rigid body simulation with extended position based dynamics," in *Computer graphics forum*, vol. 39, pp. 101–112, Wiley Online Library, 2020.

[14] S. Earles, “Impact: The theory and physical behaviour of colliding solids,” 1961.

[15] K. H. Hunt and F. R. E. Crossley, “Coefficient of restitution interpreted as damping in vibroimpact,” 1975.

[16] R. W. Cottle, G. B. Dantzig, *et al.*, “Complementary pivot theory of mathematical programming,” *Linear algebra and its applications*, vol. 1, no. 1, pp. 103–125, 1968.

[17] D. Baraff, “Fast contact force computation for nonpenetrating rigid bodies,” in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 23–34, 1994.

[18] C. E. Lemke, “The dual method of solving the linear programming problem,” *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 36–47, 1954.

[19] M. Anitescu and F. A. Potra, “Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems,” *Nonlinear Dynamics*, vol. 14, pp. 231–247, 1997.

[20] J. Yoon, B. Son, and D. Lee, “Comparative study of physics engines for robot simulation with mechanical interaction,” *Applied Sciences*, vol. 13, no. 2, p. 680, 2023.

[21] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[22] C. Ericson, *Real-time collision detection*. Crc Press, 2004.

[23] M. G. Bekker, *Off-the-road locomotion: Research and development in terramechanics*. Michigan Univ. Press, 1960.

[24] V. Asnani, D. Delap, and C. Creager, “The development of wheels for the lunar roving vehicle,” *Journal of Terramechanics*, vol. 46, no. 3, pp. 89–103, 2009.

[25] T. Wasfy and P. Jayakumar, “Next-generation nato reference mobility model complex terramechanics–part 1: definition and literature review,” *Journal of Terramechanics*, vol. 96, pp. 45–57, 2021.

[26] W. Thomson, “Iv. on the elasticity and viscosity of metals,” *Proceedings of the Royal Society of Londo*, no. 14, pp. 289–297, 1865.

[27] W. Voigt, “Ueber innere reibung fester körper, insbesondere der metalle,” *Annalen der Physik*, vol. 283, no. 12, pp. 671–693, 1892.

[28] G. Sohl and A. Jain, "Wheel-terrain contact modeling in the roams planetary rover simulation," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 47438, pp. 89–97, 2005.

[29] B. Trease, R. Arvidson, R. Lindemann, K. Bennett, F. Zhou, K. Iagnemma, C. Senatore, and L. Van Dyke, "Dynamic modeling and soil mechanics for path planning of the mars exploration rovers," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 54839, pp. 755–765, 2011.

[30] R. Lichtenheldt, S. Barthelmes, F. Buse, and M. Hellerer, "Wheel-ground modeling in planetary exploration: From unified simulation frameworks towards heterogeneous, multi-tier wheel ground contact simulation," *Multibody Dynamics: Computational Methods and Applications*, pp. 165–192, 2016.

[31] R. He, C. Sandu, A. K. Khan, A. G. Guthrie, P. S. Els, and H. A. Hamersma, "Review of terramechanics models and their applicability to real-time applications," *Journal of Terramechanics*, vol. 81, pp. 3–22, 2019.

[32] J.-Y. Wong and A. Reece, "Prediction of rigid wheel performance based on the analysis of soil-wheel stresses: Part ii. performance of towed rigid wheels," *Journal of Terramechanics*, vol. 4, no. 2, pp. 7–25, 1967.

[33] L. Ding, Z. Deng, H. Gao, J. Tao, K. D. Iagnemma, and G. Liu, "Interaction mechanics model for rigid driving wheels of planetary rovers moving on sandy terrain with consideration of multiple physical effects," *Journal of Field Robotics*, vol. 32, no. 6, pp. 827–859, 2015.

[34] H. Shibly, K. Iagnemma, and S. Dubowsky, "An equivalent soil mechanics formulation for rigid wheels in deformable terrain, with application to planetary exploration rovers," *Journal of terramechanics*, vol. 42, no. 1, pp. 1–13, 2005.

[35] G. Ishigami, A. Miwa, K. Nagatani, and K. Yoshida, "Terramechanics-based model for steering maneuver of planetary exploration rovers on loose soil," *Journal of Field robotics*, vol. 24, no. 3, pp. 233–250, 2007.

[36] L. Ding, H. Gao, Z. Deng, K. Nagatani, and K. Yoshida, "Experimental study and analysis on driving wheels' performance for planetary exploration rovers moving in deformable soil," *Journal of Terramechanics*, vol. 48, no. 1, pp. 27–45, 2011.

[37] L. Ding, H.-b. Gao, Z.-q. Deng, and J.-g. Tao, "Wheel slip-sinkage and its prediction model of lunar rover," *Journal of Central South University of Technology*, vol. 17, no. 1, pp. 129–135, 2010.

[38] H. Gao, J. Guo, L. Ding, N. Li, Z. Liu, G. Liu, and Z. Deng, "Longitudinal skid model for wheels of planetary exploration rovers based on terramechanics," *Journal of Terramechanics*, vol. 50, no. 5-6, pp. 327–343, 2013.

[39] Z. J. Janosi, B. Hanamoto, and I. elettrotecnico nazionale Galileo Ferraris, "The analytical determination of drawbar pull as a function of slip for tracked vehicles in deformable soils," 1961.

[40] M. G. Bekker, *Theory of land locomotion: the mechanics of vehicle mobility*. 1956.

[41] L. Ding, Z. Deng, H. Gao, K. Nagatani, and K. Yoshida, "Planetary rovers' wheel–soil interaction mechanics: new challenges and applications for wheeled mobile robots," *Intelligent Service Robotics*, vol. 4, pp. 17–38, 2011.

[42] F. Zhou, R. E. Arvidson, K. Bennett, B. Trease, R. Lindemann, P. Bellutta, K. Iagnemma, and C. Senatore, "Simulations of mars rover traverses," *Journal of Field Robotics*, vol. 31, no. 1, pp. 141–160, 2014.

[43] L. Ding, "Wheel-soil interaction terramechanics for lunar/planetary exploration rovers: modeling and application," *Harbin Institute of technology*, 2009.

[44] "Bullet physics library." `https://pybullet.org/wordpress/`.

[45] "Open dynamics engine." `https://www.ode.org/`.

[46] "Nvidia physx." `https://developer.nvidia.com/physx-sdk`.

[47] Q. S. Islam, H. Teras, K. Kruuse, and M. Pajusalu, "Lunarpoint: Interest point detector and descriptor for lunar landscapes," in *73rd International Astronautical Congress (IAC)*, IAF, 2022.

[48] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 4397–4404, IEEE, 2015.

[49] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pp. 281–288, 2007.

[50] E. Weitnauer, R. Haschke, and H. Ritter, "Evaluating a physics engine as an ingredient for physical reasoning," in *Simulation, Modeling, and Programming for Autonomous Robots: Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings 2*, pp. 144–155, Springer, 2010.

[51] J. Hummel, R. Wolff, T. Stein, A. Gerndt, and T. Kuhlen, "An evaluation of open source physics engines for use in virtual reality assembly simulations," in *Advances in Visual Computing: 8th International Symposium, ISVC 2012, Rethymnon, Crete, Greece, July 16-18, 2012, Revised Selected Papers, Part II 8*, pp. 346–357, Springer, 2012.

[52] "Bullet physics sdk," 2023. https://github.com/bulletphysics/bullet3/blob/master/src/BulletDynamics/MLCPSolvers/btDantzigLCP.h [Accessed: (10.08.2023)].

[53] G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pipe, and K. Eder, "On determinism of game engines used for simulation-based autonomous vehicle verification," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[54] R. Krenn, A. Gibbesch, and G. Hirzinger, "Contact dynamics simulation of rover locomotion," 2008.

[55] R. Zhou, W. Feng, L. Ding, H. Yang, H. Gao, G. Liu, and Z. Deng, "Marssim: a high-fidelity physical and visual simulation for mars rovers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 59, no. 2, pp. 1879–1892, 2022.

[56] A. Azimi, *Wheel-soil interaction modelling for rover simulation and analysis*. Phd thesis, McGill University, 2014. Available at https://escholarship.mcgill.ca/concern/theses/9880vv18m.

[57] L. Ding, K. Yoshida, K. Nagatani, H. Gao, and Z. Deng, "Parameter identification for planetary soil based on a decoupled analytical wheel-soil interaction terramechanics model," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4122–4127, IEEE, 2009.

[58] C. Shearer and G. Tahu, "Lunar polar volatiles explorer (lpve) mission concept study," *Science*, 2010.

A. ULYSSES

Figure A.1 outlines the main pipeline and components of the ULYSSES framework. Components more relevant to this thesis are briefly discussed in this chapter. The simulation environment, now named ULYSSES, has been under development for around one and a half years. It was initially used as a validation method for a mission planning tool. The output of this mission planner was a list of points of interest, their coordinates, and time of arrival. Currently, such planning for planetary missions is largely done by hand. Hence ESA has been looking for ways to automate this process. The mission planning tool developed in Tartu Observatory in collaboration with Milrem Robotics was one step towards a cohesive system. As is usual with ESA projects, the result requires extensive testing and validation. The output of the mission planning tool was tested by planning a mission around the observatory building in Tõravere. The corresponding real test run was successful. However, the tool was created for Lunar missions, meaning a simulation is also required to create a relevant environment for testing. A simulation also allows for a considerably longer and hence more realistic mission plan to be validated.

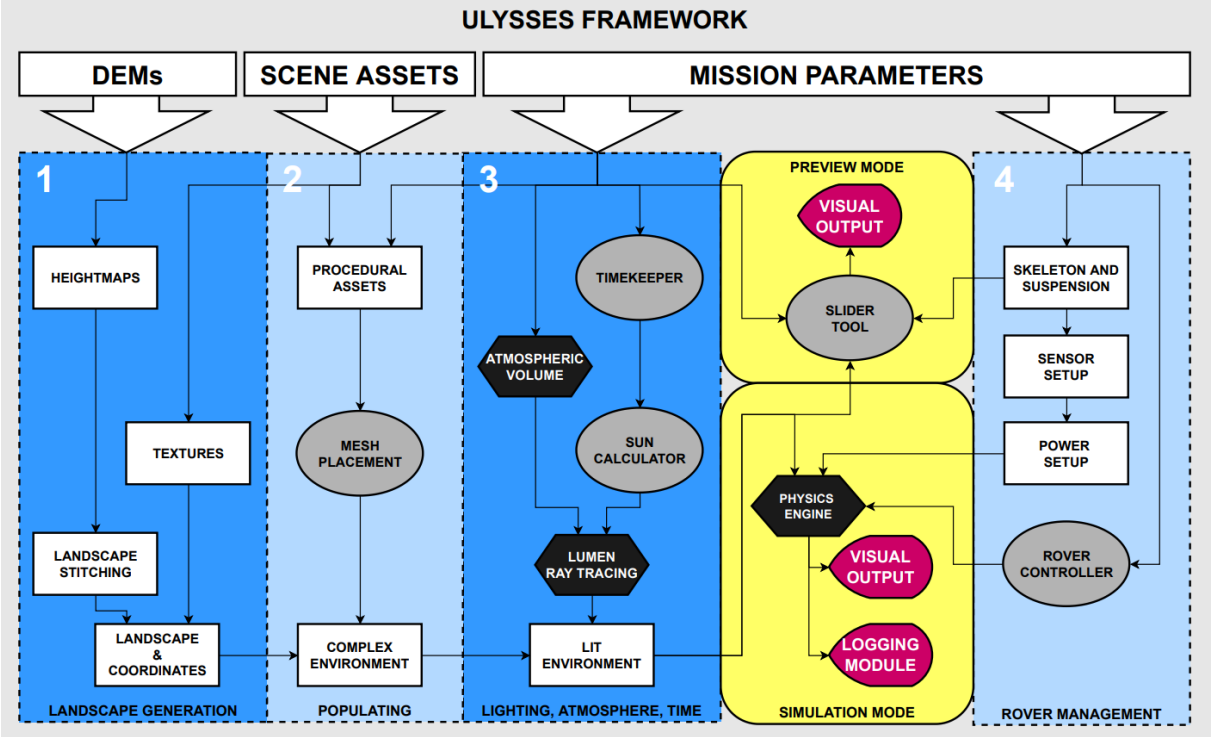

Figure A.1. *Block diagram of the ULYSSES framework, its components and modules. Grey - custom add-ons. Black - Unreal Engine components. Purple - visual output displayed to the user or saved into a file.*

In order to simulate a relevant testing environment, realistic lunar terrain, accurate sun propagation, and corresponding shadows, a rover model and both data input and logging had to be implemented. For the terrain, real lunar elevation data was used. However, its resolution is rather low, meaning the smallest level of detail is larger than an average lunar rover. To mitigate this problem, details like rocks and pebbles, even smaller craters, can be added with procedural generation. Sun propagation and dynamic shadows are also crucial factors, as avoiding shadowed areas is one of the central objectives of the mission planner tool. Last but not least, a physically based rover model was needed actually to traverse the planned path.

Terrain

To generate landscapes – a specific object type within the Unreal Engine framework for including large terrains – for ULYSSES in UE5, certain steps of dataset preparation are required. First, the area under consideration must be identified, and the relevant Digital Elevation Maps (DEM) must be obtained. Next, the DEMs should be converted into unsigned 16-bit heightmaps, i.e. grayscale images. This is the preferred data type for generating Unreal landscapes. However, this conversion process will result in the loss of important coordinate transform information regarding the x- and y-coordinates. It is also recommended to crop the existing randomly-shaped heightmaps, as seen on the left image in Figure A.2, into a predefined tile size format. This approach helps to minimize the additional buffering that Unreal would otherwise have to add to the edges of the heightmaps.

It is also important to somehow take into account the influence of faraway mountains on the horizon and their capacity to create extended shadows during planetary sunrises and sunsets. An effective approach to address this is to use multiple layers of terrain, where the more expansive yet distant heightmaps are of a lower spatial resolution. It is important to highlight that the current version of ULYSSES does not incorporate surface curvature on a planetary scale when calculating horizon view distances and distant shadowing. Though this is a work in process.

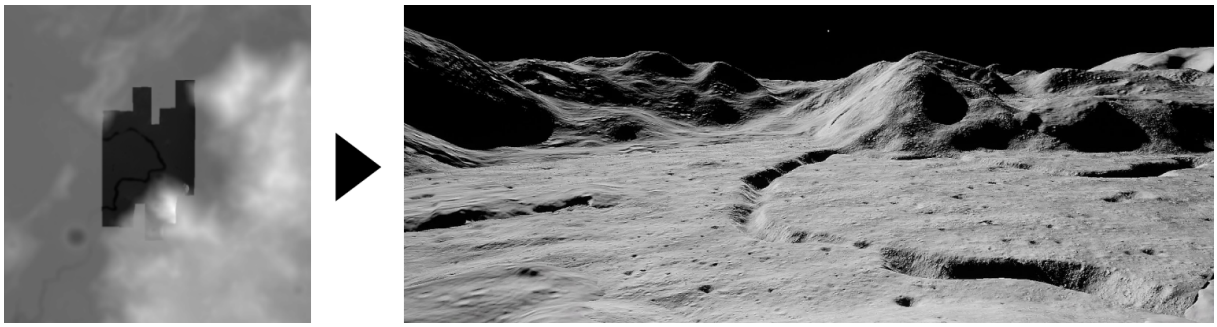

Figure A.2. [Left] *Image of combined heightmaps generated of the simulated landscape. The dark central region shows the high-resolution Apollo 15 landing site surrounded by lower-resolution LOLA data.* [Right] *Rendered landscape in ULYSSES.*

Rock Distribution

To add detail in scales lower than the resolution of the DEM data, rocks and pebbles can be added to the environment. Procedural generation of terrain clutter requires first a set of 3D assets and textures from which an automated script can select a number of random objects, scale, orient and place them randomly into the environment. The models used were acquired from an asset pack in Unreal Marketplace.

While procedural generation tools exist within UE5, a custom Python script has been created instead to better control the distribution. This script was then written that could be run in the UE5 editor through the built-in command terminal, which would generate several different levels of clutter in the area of operations. Larger boulders would be placed as static meshes, improving their collision box mesh resolution for calculating the physical interactions of a rover driving over rocks. Smaller rocks and pebbles would be generated in the vicinity as movable objects, lacking the complex collision meshes but allowing the rover to interact with them and move them around in a more realistic fashion by changing the exposed parameters, the spatial density, extent, and scale limits of the assets can be tuned, or limiting the generation to a subset of the available assets. Selecting the optimal parameters will also depend on the referenced landscape.

Rover Model

The envisioned use cases for ULYSSES rely heavily on its ability to include customizable rovers, acquire and log real-time statistics about it and get access to multiple photorealistic on-board camera feeds. The work done on ULYSSES so far can be broken down into several smaller modules and plugins, explained in more detail below, along with the development decisions and logic.

Custom rover models and suspension

The initial implementation of rovers in ULYSSES was primarily based on the existing UE5 Chaos Vehicle blueprint, including its movement, wheel components, and game modes. Other lunar rovers, such as the Apollo-era seated Lunar Roving Vehicle and the Lunar Polar Volatiles Explorer (LPVE) [58], were custom-built using Blender and freely available 3D models. These custom models were equipped with basic skeletons and plain textures and then imported into UE5 for further configuration and integration. The LPVE eventually served as the foundation for the current VIPER rover design.

While the Chaos Wheeled Vehicle systems worked well with the Apollo rover, it became evident during initial tests with the six-wheeled, bogie-suspension LPVE rover design that a separate, physics-based solution was preferable for more complex suspension types commonly found in planetary exploration rovers. The existing Chaos Vehicle configurations in UE5 pre-

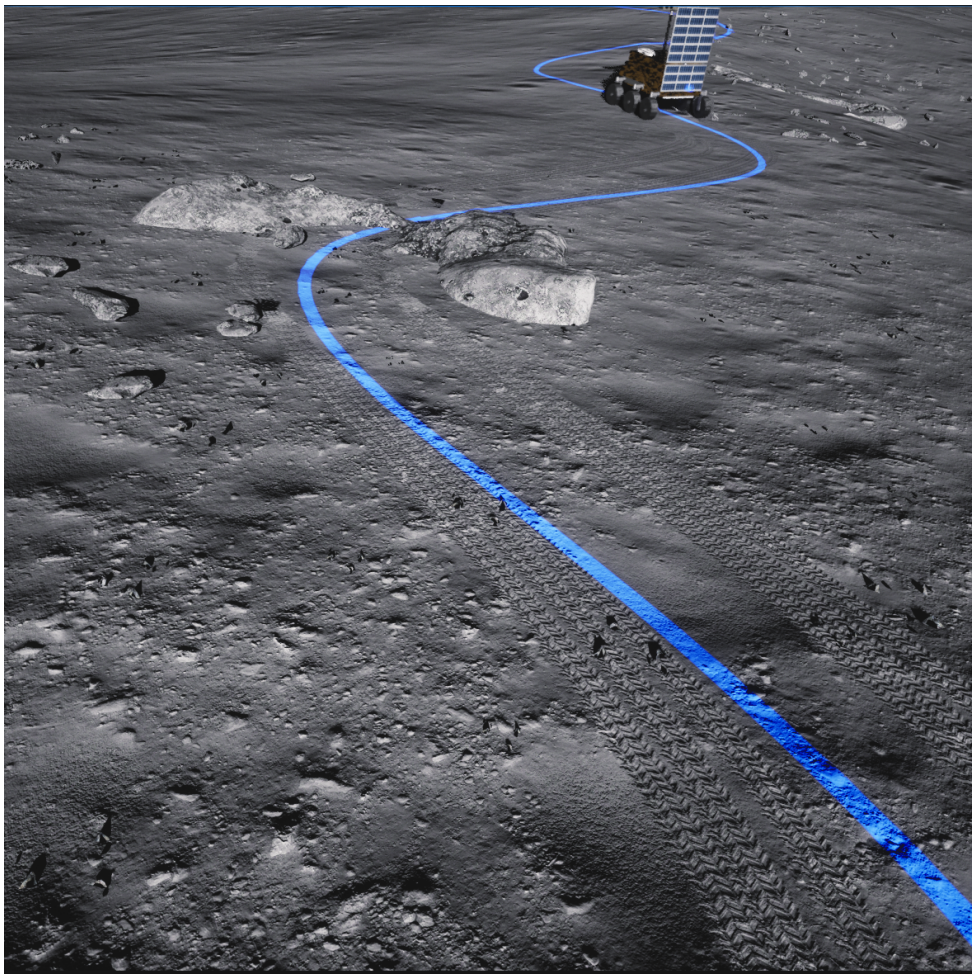

Figure A.3. *The rover follows an imported path, drawn in blue here, using a P-controller. The path depicted here is just a sinusoid on the xy-plane. Tracks left by the rover are also shown here.*

sented challenges, particularly with phantom wheels that were difficult to connect reliably within a bogie-suspension system. These phantom wheels operated independently of the actual physical body and wheels of the rover, resulting in unusual test results when navigating speed bumps. By transitioning to relying solely on the Chaos physics solver for the physical body parts, the issues were promptly resolved.

Controls

As the path to be traversed is predefined, the rover must somehow stay on track. The speed of the traversal was set at 5 cm/s, as is rather usual for planetary rovers. Considering this relatively slow speed, a simple proportional controller was implemented. While the arrow keys could also be used to play around with the rover, for the traversal, autonomy was required. An illustration is seen in Figure A.3.

Tire Tracks

To enhance the visual fidelity of the scene and provide valuable input for visual odometry algorithms, a practical approach is to depict the tire tracks left behind by the rover's wheels or instruments in the soft regolith. Figure A.3 shows an example of these tracks. In ULYSSES, this feature was implemented using deferred decals. These tools utilize smart shaders and normal maps to create the illusion of depth changes within the landscape, resulting from variations in lighting and shadows. It's important to note that this method is computationally efficient and performs well in rendering photo-realistic camera views. However, alternative methods may be necessary when simulating LiDAR data, as they must account for the additional distance traveled in the troughs of the tracks.

B. Integrating Bullet Into an Unreal Project

It is assumed here that the reader is somewhat familiar with the terminology used in Bullet and Unreal Engine. The central piece to a Bullet simulation is the world object – in the figure below, it is referred to as Bullet Simulation. This object brings all the other pieces together: the constraint solver, the collision dispatcher, and the rigid body objects. A custom C++ actor class was implemented for ULYSSES, where this object is created and held. This custom class is the Bullet-Unreal Interface class. It is not an interface as the word is generally used in C++; rather, it is the class that is exposed to the Unreal Engine editor and includes Bullet's header files. Hence through this class, the user can modify exposed simulation parameters in the Unreal editor.

New rigid bodies to be added to the simulation can be first placed in the Unreal editor and then passed to the interface class, again in the editor. Bullet Physics engine is generally meant to be combined with a rendering system. So every Bullet's rigid body object can hold a pointer to some other object. In this case, these objects are the Unreal Actors the user has passed to be included in the simulation. This makes it easy to update the locations and poses of the actors in Unreal based on Bullet calculations.

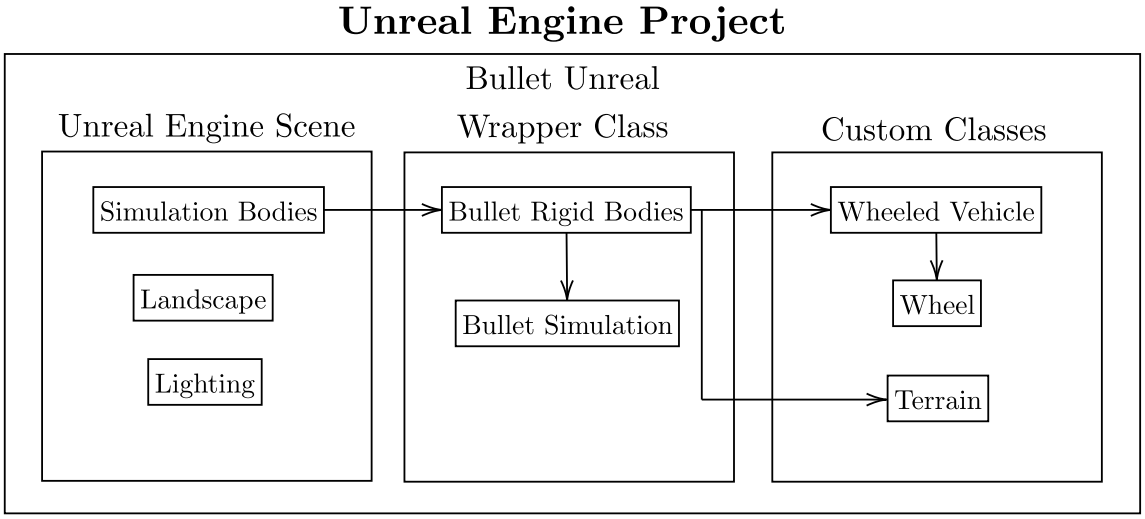

Figure B.1. *A diagram of central objects for integrating Bullet into ULYSSES.*

To accommodate the addition of a new contact model, a wheel class was written to hold extra data around the wheel's rigid bodies and also where the force calculations are implemented. A wheel object can be set to either use the usual rigid body model or the Bekker model. For the vehicle-wheel model, a wheeled vehicle class was written, which holds the wheel objects and all

the constraints used to define the dynamics model of the rover. Similarly, a terrain class holds extra data and implements contact generation when passed necessary data around the heightfield – Bullet's 2D collision object.

Like the UE equivalent landscape, the heightfield can be created based on a png image. To allow this, the C++ package libpng was used. However, it is much more convenient to do this using CSV files as these are easier to create and modify to the users liking. As seen in Figure B.1, the landscape data from UE isn't used to directly create the heightfield object, although the same png could be used. Even if this wasn't a reason, the heightfield data can be modified during a simulation, meaning the terrain could be deformed. A landscape object in UE, however, is completely static once created.

What is not shown in the figure above are some of the supporting classes not explicitly involved with keeping or updating the simulation data. Firstly, there is a class for Bullet engine debug drawing. This class implements the Bullet engine draw interface and uses UE functions to draw bodies as Bullet sees them. Secondly, a separate class for writing to files was created; although this feature is technically available in the UE architecture, it seemed easier just to implement a simple custom logging feature. A Bullet-UE unit conversion function suite is also used, mainly in the wrapper class.

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Karin Kruuse,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Implementing a Terramechanics-based Wheel-terrain Contact Model into a Multi-body Simulation Environment

   mille juhendajad on Hans Teras, MSc ja Quazi Saimoon Islam, MSc, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Karin Kruuse
19.08.2023