Simulation-based analysis of free and forced rocking behaviour in rigid URM macro-blocks using physics engines

Yuchen Han

Department of Civil Engineering

McGill University, Montreal

December 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering.

© Yuchen Han 2023

Abstract

Physics engines used for video games are simulation tools based on Newton's second law of motion and semi-implicit Euler method as the integration method. The process of modelling and simulating within these engines resembles the simulation and analytical procedures found in more rigorous structural analysis tools, such as the Finite Element Method (FEM) and Discrete Element Method (DEM). These methods, utilised by researchers for the seismic collapse assessment of reduced-scale unreinforced masonry (URM) structures, are known for their accuracy. However, FEM and DEM methods can require long simulation times, often spanning from hours to days. On the other hand, physics engines, due to their calculation mechanisms, can dramatically reduce simulation times for simple scenarios, often to just seconds. As the demand for advanced URM collapse simulations grows—supporting controlled demolitions, seismic risk evaluations, and mitigation projects—the structural engineering profession is in need of more efficient methods for preliminary analysis. Within this landscape of evolving needs, physics engines present themselves as a promising alternative to traditional simulation tools. Their applicability and adaptability make them deserve of further examination and exploration. This research investigates deep into the comparative simulation capabilities of two distinguished physics engines: Bullet, integrated within Blender, and the native engine offered by Vortex Studio. The study quantitatively evaluates a specific scenario wherein a rigid rectangular block is subjected to free-rocking motions and sinusoidal base excitations, comparing these simulation outcomes with established analytical solutions and experimental outcomes. Preliminary results from this research not only highlight the competence of these physics engines but also emphasize their potential to deliver insightful and reliable simulation results.

Résumé

Les moteurs physiques utilisés pour les jeux vidéo sont des outils basés sur les lois du mouvement de Newton. Le processus de modélisation et de simulation au sein de ces moteurs reflète étroitement les procédures de simulation et analytiques présentes dans les outils d'analyse structurelle rigoureux, tels que la Méthode des Éléments Finis (FEM) et la Méthode des Éléments Discrets (DEM). Ces méthodes, employées par les chercheurs pour l'évaluation du risque de collapse sismique de structures en maçonnerie non renforcée (URM) à échelle réduite, sont reconnues pour leur précision. Cependant, les méthodes FEM et DEM peuvent nécessiter de longs temps de simulation, s'étendant souvent de plusieurs heures à plusieurs jours. D'un autre côté, les moteurs physiques, en raison de leurs mécanismes de calcul, peuvent réduire considérablement les temps de simulation pour des scénarios simples, souvent à quelques secondes seulement. Alors que la demande pour des simulations avancées de collapse URM augmente - soutenant les démolitions contrôlées, les évaluations des risques sismiques et les projets d'atténuation - la profession d'ingénieur en structure a besoin de méthodes plus efficaces pour une analyse préliminaire. Dans ce paysage aux besoins en évolution, les moteurs physiques se présentent comme une alternative prometteuse aux outils de simulation traditionnels. Leur applicabilité et leur adaptabilité les rendent dignes d'un examen et d'une exploration plus approfondis. Cette recherche se penche en profondeur sur les capacités de simulation comparatives de deux moteurs physiques distingués : Bullet, intégré dans Blender, et le moteur natif proposé par Vortex Studio. L'étude évalue soigneusement un scénario spécifique où un bloc rectangulaire est soumis à des mouvements de basculement libre et à des déplacements de base sinusoïdaux, contrastant ces résultats de simulation avec des solutions analytiques établies. Les résultats préliminaires de cette

recherche mettent non seulement en évidence la compétence de ces moteurs physiques, mais soulignent également leur potentiel à fournir des résultats de simulation perspicaces et fiables.

Acknowledgements

I wish to extend my profound gratitude to Prof. Daniele Malomo, whose expertise, guidance, and unwavering support have been the bedrock of my academic journey at McGill University. His insightful feedback, coupled with his commitment to academic excellence, has not only shaped my research project but has also greatly influenced my growth as a student. Prof. Daniele Malomo's ability to challenge and inspire has been a beacon throughout my graduate studies, guiding me through complex challenges and encouraging me to push the boundaries of my understanding and capabilities.

I would like to extend my deepest appreciation to Prof. Sheldon Andrews for his invaluable guidance on Vortex Studio and his extensive knowledge of physics engines. His unwavering support and readiness to assist whenever I sought guidance have been immensely beneficial to my work.

Equally important is the gratitude I owe to my friends at McGill. Navigating the rigors of graduate-level courses was made immeasurably more enjoyable and meaningful thanks to their presence. Our shared experiences, from the late-night study sessions to the invaluable discussions that broadened my perspectives, have been a source of joy and support. The sense of community we built together provided not just an academic network but also a social anchor, keeping me connected and balanced during the intense periods of study and research. These friendships have been a fundamental part of my experience at McGill, offering both camaraderie and a collaborative spirit that enriched my academic life.

Contribution of Authors

This thesis has been written following the requirements of Graduate and Postdoctoral Studies for a manuscript-based thesis. The main findings of the research undertaken by the author as part of his master’s program are presented in a single manuscript. The author carried out the numerical analysis and wrote the manuscript under the supervision of Prof. Daniele Malomo (supervisor).

Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1 Background: significance of understanding rocking behaviour in URM macro-blocks for structural safety

Many historical structures worldwide were constructed using unreinforced masonry. This construction primarily used bricks and mortar. The typical structural characteristic of masonry structures can be observed as in figure 1.1, it is a combination of motor which acts as a cohesive connection which combines the units of bricks together and forms a composite material. (Lourenço 2002)

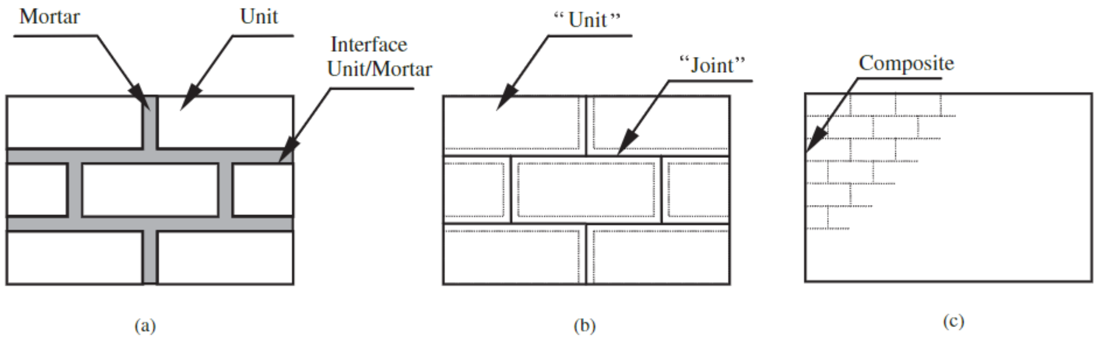


Figure 1.1. Modelling strategies for masonry structures: (a) detailed micro-modelling; (b) simplified micro-modelling; and (c) macro-modelling (Lourenço 2002)

Given their architectural and historical significance, preservation of these structures is crucial. Therefore, many numerical and analytical tools have been developed and studied to be able analyze the behaviour of masonry structures under different scenarios, such as the well known finite element method and discrete element method. In finite element method (FEM) and discrete element method (DEM), It's important to recognize the heterogeneous nature of unreinforced masonry (URM) structures. This means that the contact surfaces between the structure's elements consist of different materials, each with its distinct properties. This variance in material properties can render these structures especially susceptible to ground motions, such as those during earthquakes. (Mehrotra and DeJong 2017) However, over time, and particularly due to environmental factors, the mortar binding the bricks can deteriorate. This deterioration compromises the structural integrity of these buildings, increasing their vulnerability under various conditions. (Penna et al. 2013) This characteristic can also be observed in dry-joint masonry structures where no motor is present in between the units of bricks. A prominent behaviour exhibited by these structures during seismic events is 'rocking'. This motion resembles how a tall object might sway when nudged. (Cappelli, Di Egidio, and Vestroni 2020) Understanding this rocking behaviour is vital for engineers and researchers. By examining this motion, insights can be drawn on enhancing the resilience and safety of unreinforced masonry structures during seismic activities. (Casapulla, Giresini, and Lourenço 2017)

Housner's seminal work in 1963 paved the way for an extensive body of research into the mechanics of rocking (DeJong and Dimitrakopoulos 2014). This knowledge now underpins the design of innovative, low-damage energy dissipation devices (Kovacs and Wiebe 2019). The ability to predict rocking motion becomes critical when evaluating the seismic performance of older, sub-standard structures, especially those built of unreinforced masonry (URM). These buildings, characterized by weak or even non-existent connections between their components such as the floor-to-gable, are susceptible to severe, rocking-driven collapses (So and Spence 2013). The work by Ishiyama in 1984 has investigated deeper into the rocking motion and the jumping phenomenon of rigid blocks under sinusoidal base excitations through laboratory experimentations, and produced detailed figures to trace the motion of the blocks. (Ishiyama 1984) Within this field of study, numerous researchers have incorporated the shake table testing method into their research methodologies. For instance, in a notable study conducted by Galvez and colleagues in 2018, a two-story unreinforced masonry (URM) scaled model building was meticulously constructed. This model was then subjected to rigorous testing on a uniaxial shake table. The purpose of this experiment was to closely observe and understand the dynamic behavior of URM structures when exposed to earthquake-like conditions. (Galvez, Giaretton, et al. 2018) However, it's important to acknowledge that the nature of shake-table testing necessitates a substantial investment in terms of both time and resources. Additionally, the repeatability of this testing method is relatively limited, which can pose challenges in verifying and refining results. Furthermore, conducting shake-table tests for larger scaled structures is often impractical, if not unfeasible, due to the significant logistical, financial, and technical constraints involved. This limitation highlights the need for alternative or complementary methods that can efficiently simulate and analyze the seismic response of large structures, enabling more scalable and versatile approaches in earthquake engineering research. Therefore, computer-based analytical tools have great application potential to be implemented in this area of study.

1.2 Traditional methods: FEM and DEM approaches to rocking behaviour in URM macro-blocks

Thanks to advancements in computer technology and numerical modeling, researchers can now utilize a plethora of simulation tools, with the Finite Element Method (FEM) being a notable example. However, while there exists a wealth of successful FEM applications (Vassiliou, Mackie, and Stojadinovic 2016) (Zhong and Christopoulos 2021), conventional continuum based FEMs often struggle to simulate phenomena like separation, contact, and re-contact that define rocking responses. As a workaround, zero-thickness spring interfaces are frequently implemented to bridge FEM rocking elements with their environment, for instance, foundations or other rocking blocks. However, this can be computationally demanding.

The Discrete Element Method (DEM), which Cundall pioneered for addressing rock mechanics issues in 1971, has been adopted to model URM rocking behavior in macro rigid blocks under various conditions (Galvez et al. 2022) (El Shamy and Zamani 2012) and that of reinforced concrete structures. (Scattarreggia, Malomo, and DeJong 2022) The behavior of historical structures during major earthquake events has also been studied and models using DEM tools in previous studies. (Mehrotra and DeJong 2017) These models predominantly use rigid blocks in conjunction with spring-based contact models. In previous study, from the blind prediction contest results, the DEM was proven to be the accurate approach for simulating podium structure under seismic motion. (Malomo, Mehrotra, and DeJong 2021) While DEM and other discontinuum-centric numerical methods (Lemos 2007) appear more suitable for modeling rocking mechanisms, their often-lengthy analysis times currently discourage widespread adoption in practical applications.

Traditionally, the Finite Element Method (FEM) and the Discrete Element Method (DEM) have been critical in simulating URM structures and their associated behaviors. Inherently, FEM is a numerical strategy that decomposes a large system into smaller, more manageable components known as finite elements. As articulated by Zienkiewicz in (Zienkiewicz, Taylor, and Zhu 2005), these elements interconnect at specific junctions called nodes. Each element's behavior is defined through a suite of mathematical expressions. Systematic assembly of these elements yields global equations for the overarching system, which are subsequently solved. The out-of-plane URM collapse scenario has also been analyzed in the study done by Galvez et al. Where they have performed FEM and DEM analysis and compared the simulation results. (Galvez, Segatta, et al. 2018)

On the other hand, the Discrete Element Method (DEM) also serves as a computational technique dedicated to modeling granular materials and other discrete systems. As highlighted in (Bui et al. 2017) , DEM-based structural analysis tools, such as 3DEC, have been employed to analyze macro masonry structures, given their exceptional capability in modeling the contact surfaces of masonry blocks. In the study carried out by Dimitri and colleagues, DEM analysis for multi-drum columns and arches on buttresses was done, and they have investigated the failure modes and did collapse analysis. (Dimitri, De Lorenzis, and Zavarise 2011) In DEM, each particle or discrete element is methodically tracked spatially and temporally, with a focus on their contact dynamics and interactive forces. (Cundall 1971) As these particles engage, forces, and torques are computed based on their relative movements and established contact laws. This methodology proves invaluable for scenarios where the interactions between particles are pivotal, as observed in granular flow, soil mechanics, and powder processing. (Xiao et al. 2023)

While the aforementioned numerical techniques offer precise, reliable simulation outcomes adhering to the tenets of structural engineering, they are not without drawbacks. The significant computational resources and extended time frames they demand are notable limitations. As such, there's an imperative for a method that is both efficient and expedient yet retains a commendable level of accuracy. This becomes especially relevant given the increasing demand for rapid preliminary evaluations of masonry structure degradation, such as debris distribution, and immediate simulations aiming for safety training and risk evaluation.

1.3 The potential of videogame physics engines in analyzing rocking motion of rigid blocks

In pursuing more efficient analytical methods for URM structures, the potential of videogame physics engines emerges as a noteworthy alternative. Videogame engines are optimized for real-time performance, enabling rapid simulations and instant feedback. Existing videogame physics engines are constructed to facilitate authentic, instantaneous interactions within simulated worlds.

While traditionally created to supplement the gaming experience, the effectiveness and promptness of these engines are revealing promising directions in academic research, especially within structural dynamics and mechanics. Prior studies have utilized these physics engines in assessing building collapse and the complexities of ruin scenarios. (Zheng et al. 2020) Owing to their foundational calculus—predicated on Newton's laws of motion and Euler's methods—these engines exhibit remarkable efficiency in simulation duration and precision. (Zheng et al. 2020) The graphical representation of simulations offers an intuitive understanding, facilitating easier communication of complex concepts to a broader audience. In addition, leveraging existing game engines for research purposes can prove to be more affordable than developing custom simulation software from scratch. In Ma's research, simulations results for free and forced rocking motion of rigid rectangular blocks were tested and compared to analytical solution provided by the Housner's equation. (Q., S., and Montalla 2018) The comparison of the simulation to the analytical results was encouraging and provided a general guideline to how to practice these kinds of experiments in Blender. Another study by Zheng et al. also utilized physics engine to create a high-fidelity model for the purpose of seismic analysis for urban planning. (Xu et al. 2014) This study focuses on using a high-fidelity structural model and a physics engine to accurately predict seismic damage in urban areas.

1.4 Previous applications if physics engines within and beyond structural engineering.

In recent academic and applied research, there has been a growing interest in leveraging the capabilities of physics engines to simulate and analyze various structural collapse and ruin scenarios. These engines are specially designed to emulate the physical world in a virtual environment, making them apt tools for such experiments. In several previous studies, these physics engines have been utilized extensively. Notably, researchers carried out simulations to explore building collapse scenarios and then compared these simulation results with outcomes derived from the Finite Element Method (FEM) for a comprehensive comparative analysis. (Zheng et al. 2020) In a particularly insightful study commanded by Zheng et al, there was an innovative approach adopted where the team carried out a conversion of the geometric model typically used in FEM into a model suitable for a physics engine. The next step involved conducting a rigorous collapse analysis of the newly formed model. To ensure that the methodology was both robust and accurate, the researchers went a step further by validating their results. They did this by conducting a shake table test specifically on a 3-story reinforced concrete frame. At the end of the study a conclusion was drawn that FEM was accurate and should be implemented in the small deformation stage, however due to the nature of the physics engines, during the large deformation stage it makes more sense to implement physics engines. The study carried out by Ma et al. also utilized physics engine, namely Blender to simulate the free and forced rocking behaviour of rigid rectangular blocks and compared the outcome of the simulations to the analytical solution provided by Housner's equation. (Ma, Parshottam, and Montalla 2018) This study provided an encouraging result between the comparison of Blender simulation and analytical result, as well as a general guideline on how to perform physical simulation in Blender environment. In another study by Zhen et al. A structural model under seismic motion was created for the purpose of urban planning study. (Xu et al. 2014) This study focuses on using a high-fidelity structural model and a physics engine to accurately predict seismic damage in urban areas.

1.5 Research objectives and methodological overview.

The focus of this research delves deep into the exploration and examination of the preliminary methodologies used in the area of structural engineering, specifically looking into the feasibility of employing physics engines for practical applications. The emphasis is placed on understanding the behavior of Unreinforced Masonry (URM) structures when subjected to specific ground motion conditions. The research intended to reconstruct and perform the analysis that was carried out by previous studies utilizing different physics engines and analysis software and compare the results. Starting with the most fundamental free rocking motion for single rigid blocks, the two chosen physics engines will be tuned and tested for their accuracy and efficiency. Once the free rocking motion for single rigid block has been analysed, the study scope was then leveraged to exploring the motion of macro rigid block under sinusoidal based motion. Further enriching the depth of this study, a comparative analysis is embarked upon. The objective here is to accurately match the simulated results obtained from the physics engines with actual results from tangible laboratory experiments. An example of such an experiment can be seen in Figure 1.2 below. In this experiment, conducted by researchers Fernando Peña, Paulo B. Lourenço, and Alfredo Campos-Costa, marble macro blocks of diverse dimensions and aspect ratios were exposed to a spectrum of loading scenarios, providing invaluable data and insights for this research. They have performed laboratory testing for both single and multi-block in their studies. (Peña et al. 2007) (Peña, Lourenço, and Campos-Costa 2008b)

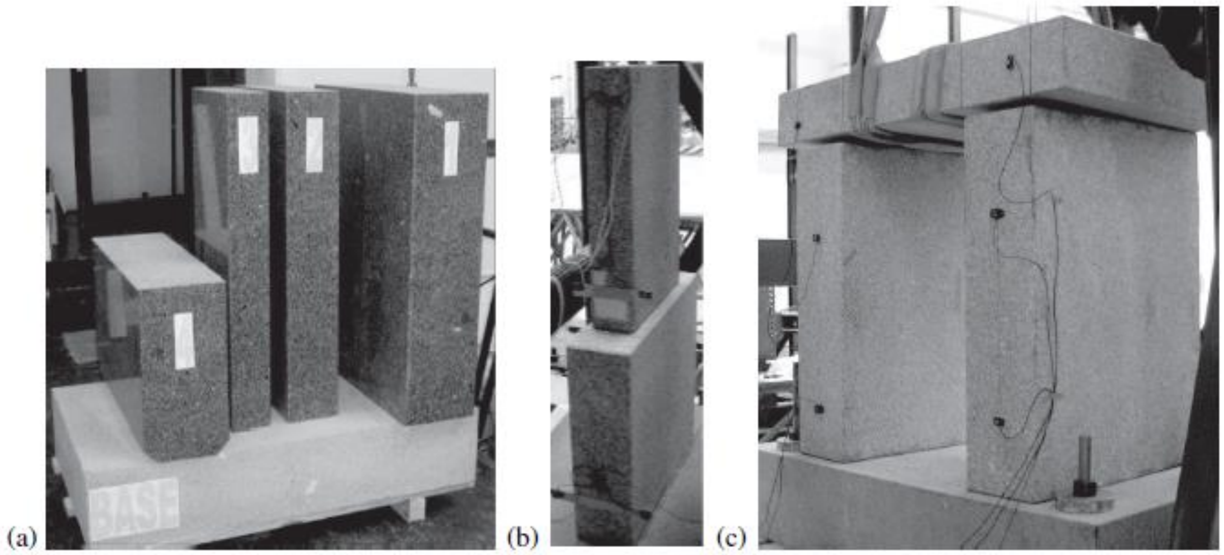


Figure 1.2. Test specimens: (a) single blocks; (b) stacked blocks; (c) trilith. (Peña, Lourenço, and Campos-Costa 2008a)

Upon reaching the culmination of this in-depth study, a grand-scale simulation was carried out. The chosen subject for this exercise was the iconic bank buildings located in Belfast, UK. This particular building was selected for its masonry façade which was a ruin resulted from a fire accident, a key feature that has both aesthetic and structural importance. (Malomo and DeJong 2022) One of the pivotal components of this simulation was to conduct a debris distribution analysis. This was deemed necessary to not only evaluate the structural integrity and potential risks posed by the masonry façade but also to forecast possible debris patterns in the event of demolishment. Such an analysis presents invaluable insights, offering critical safety measures and ensuring the preservation of a building's heritage. It plays a pivotal role in mitigating potential hazards not only for its occupants but also for the general public.

1.6 Research outline

This study is organized into four subsequent sections given as follows:

Chapter 2 gives the exploration of the contact model in physics engines. Starting with the traditional FEM and DEM contact models and move on to discussing the difference between the numerical models. The detailed contact model interpretations in Blender and Vortex studio are explained in terms of the contact geometry and position and force updating calculation mechanism.

Chapter 3 explores the free rocking motion of rigid macro blocks modelled in Blender and Vortex studio and compares their rocking angles against theoretical values derived from the Housner's equation. Following a similar approach proposed by Ma et al. (Q., S., and Montalla 2018) The study starting with the construction of the model in both software and followed by a methodology to extract and export the angle of rotation data at each time frame. The establishment of the model consists of defining the geometry mesh in the portal, and setting the contact geometry, simulation frame rates, simulation sub steps, and solver iterations for the rigid body simulation, also, the bounciness and friction coefficient were defined. The blocks were subjected to an initial angle of rotation and then allowed to go into free rocking motion once the simulation starts.

Chapter 4 evaluates the scenario where imposed sinusoidal ground acceleration was applied to the single rigid block and induced rocking behaviour. The blocks were modelled in Blender and Vortex and placed on a base plate. Sinusoidal acceleration will be imposed onto this base plate in Blender and Vortex. The rocking behaviour of the blocks were studied and compared to theoretical values. The input ground motion was imported into Blender using python scripts and for Vortex studio, the base plate was connected to a virtual motor that controlled by python script that imposes a sinusoidal ground motion onto the motor. Then, a comparative study was carried out where models created in physics engines that resembled the experimental marble specimens that Fernando et al. (Peña, Lourenço, and Campos-Costa 2008a) where specimens with trimmed edges were subjected to sinusoidal ground motion and behaviour of stacked blocks were studied. At the end of this section, a study in progress for a building scale experiment of debris distribution analysis for a façade was performed. The façade is a real-world case study from a five-story heritage construction located in the historic center of Belfast (UK). The model of the façade was reconstructed in Blender and in Vortex. A pushover analysis will be carried out to assess the debris distribution and the results were compared to the analysis outcome from the DEM model in 3DEC.

Chapter 5 serves as the conclusion section, which succinctly summarizes every aspect that has been discussed throughout this research. It extracts the extensive investigations, key findings, and theoretical implications into a coherent overview. This chapter not only encapsulates the essence of the research but also reflects on the methodologies employed, evaluates the results in the context of the existing literature, and provides a critical assessment of the study's limitations and potential avenues for future research.

Chapter 2 – Contact model Overview

2.1 Contact models in typical FEM approach

In the continuum methodology, the fundamental characteristics of granular materials are depicted through constitutive principles. (Zienkiewicz, Taylor, and Zhu 2005) These are typically articulated as differential equations connecting mechanical variables, such as stress and strain. When using this methodology to model a material, it is presumed that the substance is unbroken and occupies its space entirely. Consequently, the actions of individual particles are overlooked. The derived constitutive equations are then solved using numerical techniques, like the Finite Element Method (FEM).

Within the context of this research, contact models are conceptualized as representations delineating the interactive behaviors exhibited by two or more entities upon coming into contact. Such models facilitate the simulation of diverse intra-system interactions under a variety of conditions. In the domain of the FEM, structures are formulated as a compilation of interconnected elements, each characterized by distinct stiffness matrices and force vectors. (Zhong and Christopoulos 2021) The nodal displacement within these structures is ascertained through the following foundational relationship:

$$q=Ku+f$$

In this equation:

- q denotes the nodal force vector,
- K signifies the stiffness matrix pertinent to the element,
- u represents the nodal displacement vector, and

- *f* stands for the force vector which counterbalances the nodal forces. (Zienkiewicz, Taylor, and Zhu 2005)

FEM employs this relationship to predict an element's subsequent position or its corresponding force vector, operating on the assumption of a state of equilibrium. Given FEM's inherent methodology in addressing such problems, it may not exhibit optimal efficiency when applied to structures characterized by the presence of discontinuous elements.

2.2 Contact models in typical DEM approach

Rather than using the continuum method, the discrete technique considers each particle as its own individual unit within a granular material, conceptualizing it as an ideal cluster of particles. It's the interactions between these singular particles that shape the system's overarching (macroscopic) behavior. According to this approach, one can closely study occurrences specific to the size of an individual particle and better simulate the aggregate behavior of these particles.

The contact mechanics implemented in physics engines is surprisingly similar to that of most DEM-based micro-models used in structural and earthquake engineering analyses, see Figure 2.1 below.

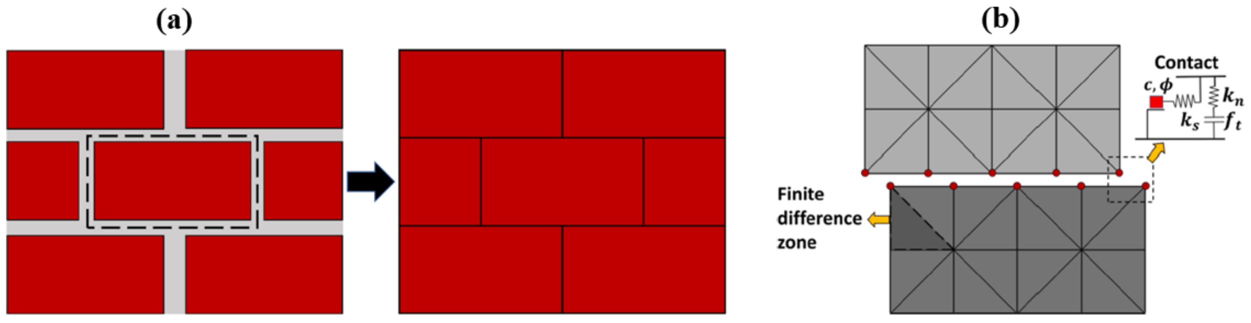


Figure 2.1 (a) simplified DEM micro-modelling of URM, (b) DEM contact mechanics (Pulatsu et al.

2020)

In the Discrete Element Method (DEM), when two rigid blocks, each possessing six degrees of freedom, come into contact, their faces undergo triangulation, forming sub-contacts at the element vertices— a common practice in many physics engines. For blocks that can deform, a Finite Element (FE) mesh is necessary to account for unit deformations. Here, each internal element serves as a domain with uniform strain, boasting three autonomous degrees of freedom. In such scenarios, sub-contacts are established at every node of the FE mesh. Zero-thickness springs, defined by specific constitutive contact models and bolstered by both normal (k_n) and shear (k_s) stiffnesses, are designated at these sub-contacts between solid elements, as illustrated in Figure 2.1.

Within the elastic range, the behavior of the joint is steered by both k_n and k_s. Concurrently, plastic parameters such as cohesion (c), friction angle (ϕ), and tensile strength (ft) ensure that the springs can snap if stresses surpass set thresholds, often determined using Coulomb criteria. For dry joints between two surfaces, the rocking movement is predominantly influenced by kn, ks, and ϕ. Addressing this particular aspect is our primary objective in the ensuing sections. It's noteworthy that physics engines adhere quite meticulously to the tenets of DEM.

DEM contact model can be generalized of consisting three major parts: particles, walls, and contacts. (Chen, Zhou, and Yin 2023) Similar to physics engines, the Newton's law of motion will be the general guideline behind mechanism governing the contact between the particles, and between the particles and the walls in DEM. In the meantime, the force-displacement method is controlling the motion updating mechanism for each contact that occurred during the simulation. For translational motion, the newton's second law of motion is utilized as shown in equation 2.1.

$$F = m * (\ddot{x}_i - g)$$

Where F stands for the resultant force and m stands for the mass of the particle. $\ddot{x}_i$ is the acceleration of the particle, and g is the gravitational acceleration. In a similar fashion, the rotational motion for a spherical particle can be then defined as follows in equation 2.2.

$$M = I\dot{\omega}_j = \left(\frac{2}{5}mR_i^2\right) * \dot{\omega}_j$$

Where M is the resultant moment, and I is the rotational inertia of the particle, for a spherical object, I is given as $\frac{2}{5}mR_i^2$, and R_i is the radius of particle i, $\dot{\omega}_j$ is the rotational velocity of particle i.

After the forces and moment have been obtained, the positions of the particle and walls will be updated using the force-displacement law, which takes the unit normal vector from the center of the particle to the wall given as vector n, and the position of the particle x_i and calculates the position of the contact $x_i^{[c]}$. The length from the center of the particle to the wall is noted as l. As shown in the following equation:

$$x_i^{[c]} = x_i + \left(R_i - \frac{R_i - l}{2}\right) * n$$

In this case, the force can be broken down into two main components: the normal contact force and the tangential contact force. These forces are derived according to the relative displacements in both normal and tangential directions. The calculation for the normal contact force uses the normal contact stiffness and the thickness of the contact overlap. (Chen, Zhou, and Yin 2023) On the other hand, the tangential contact force is tied to the elastic tangential contact force, which is influenced by the relative contact velocity at the interface given as Vi, this velocity is determined by the difference in velocities between the wall and the particle at their point of contact. To increase the contact displacement, the following relationship is utilized:

$$\Delta x_i^{[cn]} = V_i * \Delta t$$

Where Δt denotes the simulation timestep, and $\Delta x_i^{[cn]}$ is the contact normal displacement incrementation. Furthermore, the incremental changes in both normal and tangential displacements of the contact are presented, with the latter's force update also being dependent on the tangential contact stiffness.

In general, DEM is a method that perfectly compatible in simulating bulk granular particle behaviour. (Burman, Cundall, and Strack 1980) Due to the fact it considers each element as discrete entity unlike FEM.

2.3 Contact models in physics engines: Blender and Vortex studio

In Blender, the built-in physics engine is the Bullet physics engine. Bullet is a physics engine designed to emulate collision recognition and the dynamics of both soft and rigid bodies. It's been employed in video gaming and for cinematic visual effects. Erwin Coumans as the primary creator of Bullet, received a Scientific and Technical Academy Award for his contributions to the engine. The manuscript he composed offers a comprehensive explanation of the theories related to the physics of contact models and rigid body dynamics. (Coumans and Rights 2009) This knowledge has become indispensable in real-time physics simulation applications, largely owing to the impressive speed at which Bullet can simulate these events.

Bullet and Vortex are adept at simulating rigid-body physics, utilizing generalized coordinates for each body. These coordinates encompass six degrees of freedom: three that dictate linear velocity and another three for angular velocity. When objects collide, their interactions and the subsequent linkage are captured through a constrained rendition of the dynamical equations. This is then addressed using the Lagrange multipliers method. Such a methodological approach inevitably paves the way for the widely accepted complementarity problem formulation in contact dynamics. Here, it's imperative to address non-interpenetration and frictional forces during every frame, all while ensuring the constraints of feasibility and complementarity are maintained.

Each contact is represented by a collision point as shown in Figure 2.2, which gives the position where two objects collide, as well as a normal direction that determines the orientation of the contact plane. Non-interpenetration forces are aligned with the normal direction, whereas the friction forces are generated in the space tangent to the contact plane and follow a Coulomb friction model.

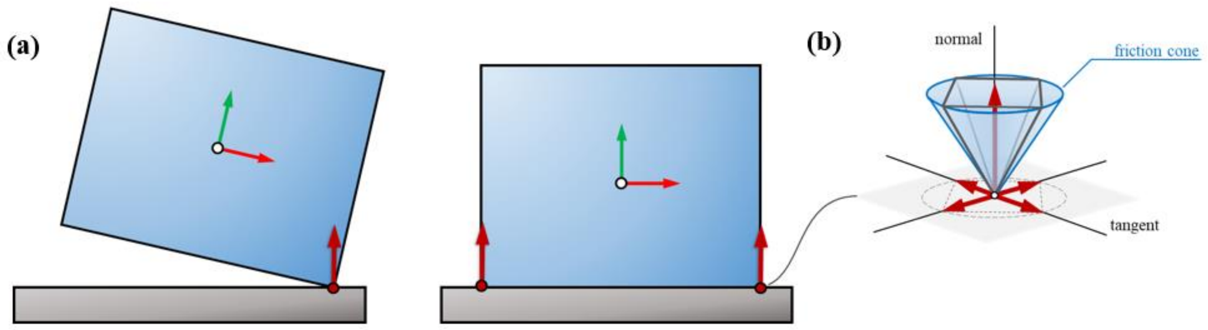


Figure 2.2 (a) point-based contacts generated for two different configurations of the box, (b) contact frame is composed of the contact normal, giving the direction of non-interpenetration forces, and friction forces. (CM Labs Simulations 2016)

In the Bullet physics engine, before one can apply principles of soft or rigid body dynamics, each mesh object (which represents the geometric details of the model to be analyzed) is assigned a specific contact shape. The Bullet engine is equipped with collision detection mechanisms that feature algorithms and acceleration structures. These are specifically designed to locate the closest points, incorporating both distance and penetration metrics, and also facilitate ray and convex sweep examinations. To assist users, Bullet has introduced a variety of primitive shapes as collision geometries. This suite of shapes allows users to make an initial selection and subsequently adapt using two other options: mesh and convex hull, which cater to more intricately shaped objects. (Coumans and Rights 2009) The rationale behind choosing a particular contact geometry is illustrated in figure 2.3.

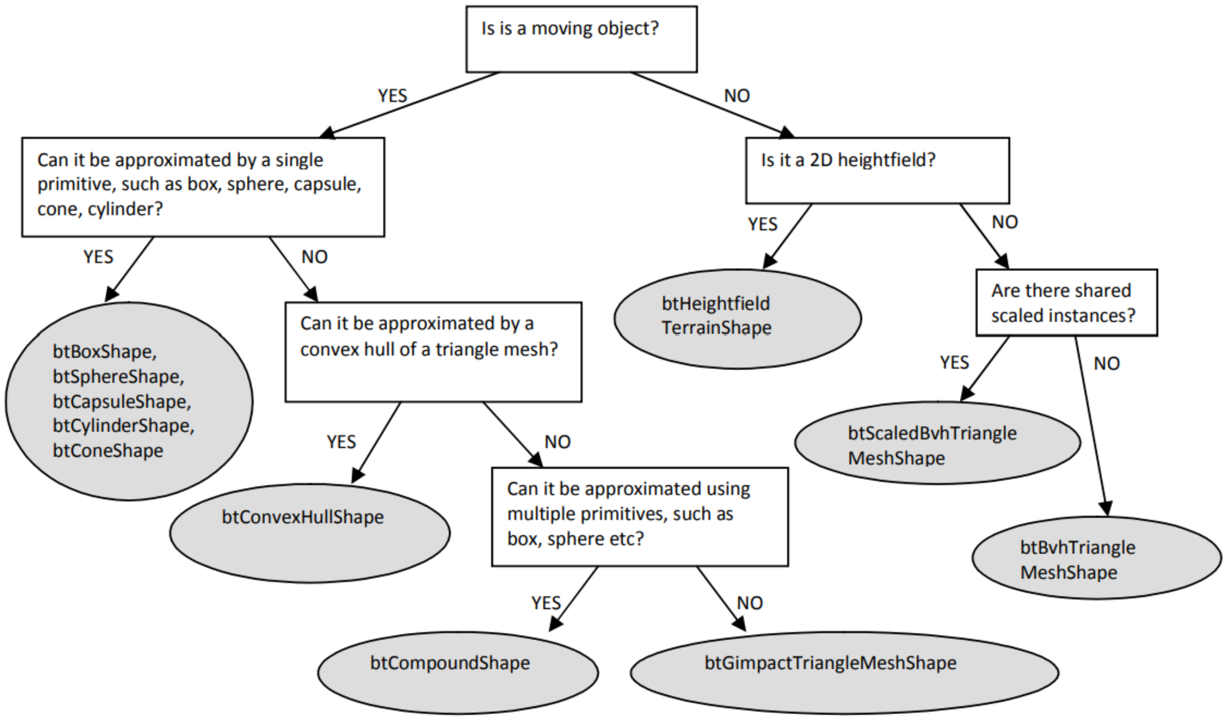

Is is a moving object?
YES
NO
Can it be approximated by a single primitive, such as box, sphere, capsule, cone, cylinder?
Is it a 2D heightfield?
YES
NO
YES
NO
btBoxShape, btSphereShape, btCapsuleShape, btCylinderShape, btConeShape
Can it be approximated by a convex hull of a triangle mesh?
btHeightfield TerrainShape
Are there shared scaled instances?
YES
NO
YES
NO
btConvexHullShape
Can it be approximated using multiple primitives, such as box, sphere etc?
btScaledBvhTriangle MeshShape
btBvhTriangle MeshShape
YES
NO
btCompoundShape
btGimpactTriangleMeshShape

Figure 2.3 collision shape determination logic map (Coumans and Rights 2009)

Collision detection forms contacts by identifying interactions between pairs of geometries. The count and positioning of these contact points are contingent on the specific geometry representing each object, such as a box and the ground in our instance. To optimize performance and attain real-time frame rates, these systems frequently employ approximate representations of the geometry. This is because these approximations streamline and hasten collision tests. Both engines we incorporated in our studies can handle collisions involving analytical primitives, such as boxes, spheres, planes, and cylinders. Additionally, they can also manage triangle meshes and their associated convex hulls. They essentially envelope the object being analyzed, establishing its volume and surface. This encapsulation not only makes it feasible to detect contacts between various objects but also serves as the foundation for implementing rigid body dynamics interactions between them. Also, a collision margin is introduced to increase the stability and performance of the simulation in bullet physics engine, by default this collision margin is set to 0.04 m in blender. For some collision shapes the collision margin is embedded in the geometry, implying a reduction in the actual shape, which results in a slight shrinking of the object. However, in many other cases, this margin introduces a slender external layer, effectively enlarging the object's dimensions. For the specific requirements of this study, it was necessary to limit the impact of the collision margin. Hence, the margins were particularly set to an ultra-minimal value of 0.000001 m. It's worth noting that an absolute value of 0 m could inadvertently introduce discrepancies or errors during the simulation. (Blender Documentation Team 2023) Overall, the collision between objects in bullet physics engine will be detected once the collision shapes of the two objects come in to contact and penetrates the collision margin layer, and then the calculation for updating the position of the objects in each time frame will start following the Newton’s law of motion and utilizing a simplistic Euler algorithm to update the motion of a rigid body. (Zheng et al. 2020)

Chapter 3 – Free rocking motion analysis of undamped rigid URM macro-blocks

3.1 Free rocking motion overview and Housner's equation

Housner formulated simple block equations that adeptly predict the motion of a rectangular block engaging in free rocking. This is based on three foundational assumptions as delineated by Housner. (Housner 1963) Initially, there's no bounciness between the block and the ground. Next, the friction between the block's corners and the ground is ample, preventing any sliding. Lastly, the rocking motion transpires exclusively in-plane. As illustrated in Figure 3.1, the distance from the base to the center of gravity is denoted by h, and b represents the distance from the block's side.

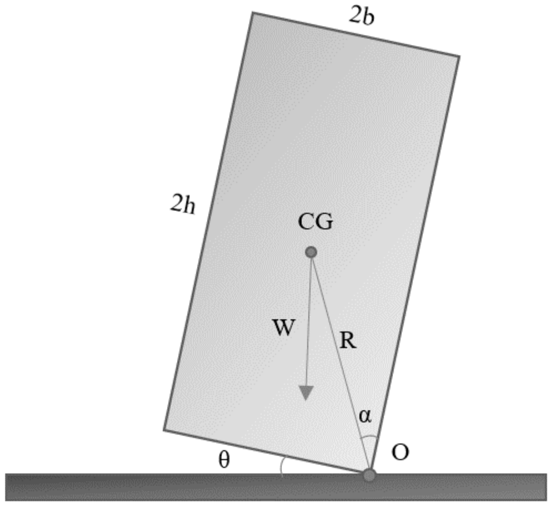


Figure 3.1 Housner's rocking block main parameters

The main parameters in Housner's equation of motion are radial distance from the center of rotation *R*, the critical (or slenderness) angle *α*, and the moment of inertia I_o. To obtain the radial distance from the center of rotation, equation (1) is used. The critical angle is calculated as equation (2).

$$R = \sqrt{h^2 + b^2} \qquad (1) \qquad \alpha = \arctan\left(\frac{b}{h}\right) \qquad (2)$$

Then, the equation of motion of the rocking block is given by (3).

$$I_o \frac{d^2\theta}{dt^2} = WR * sin(\alpha - \theta) \qquad (3)$$

By setting $\frac{WR}{Io} = p^2$, and assume $\alpha < 20^{\circ}$, the above equation can be written in the form of $\ddot{\theta} = -p^2\alpha + p^2\theta$. For a rectangular block, $p = \sqrt{3g/4R}$. (Vlachakis et al. 2021) With initial conditions $\theta = \theta_o$and $t = 0$, the displacement of the block can be expresses as:

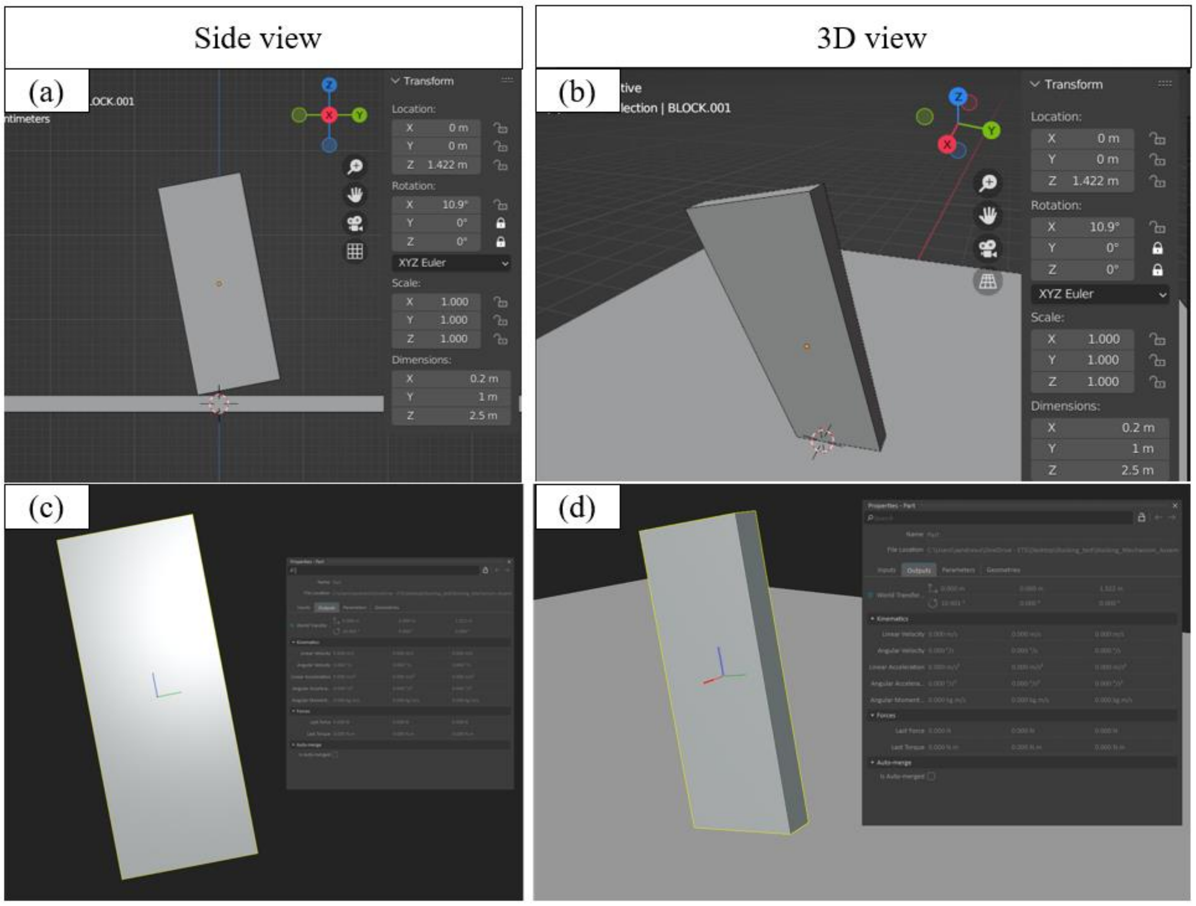
$$\theta = \alpha - (\alpha - \theta_0) \qquad (4)$$

Where θ_o is the initial tilted angle of the block. Ordinary differential equation (ODE) solvers were used to obtain the angle of rotation of the block versus time from the Housner's equations for rocking blocks. (Lachanas and Vamvatsikos 2021)

3.2 Modelling process in Blender

In this study, URM rocking members were treated as rigid macro-blocks, i.e. assuming that their behaviour is fully governed by rocking motion. Also, crushing damage and other types of failures within or outside the macro-blocks are not accounted for numerically at this stage of the research. The study renowned by Augusti, G. and Sinopoli, A. has provided great insight into the modelling of macro block structures. (Augusti and Sinopoli 1992) The interface between macro-blocks and fixed foundation is a dry-joint contact, with a friction angle of 45°. In Blender and Vortex, multiple rocking macro-block models were created that comprise a large variability of aspect ratios and initial rotation α. A list of aspect ratios *h/b* and the initial angle of rotation θ_o (normalized with respect to the critical angle α) is reported in Table 3.1.

In the Blender software, when working with the simple cube mesh, its dimensions can be finely tuned via the transform bar. After making these dimensional adjustments, it was crucial to reset the scales to one, by using Ctl+A and then select ‘scale’ to reset the scale of the mesh. This step ensures that any potential errors in the physics simulations are negated. Within the physics properties section in Blender, the block's status was designated as 'active', while the base plate, sometimes referred to as the foundation, was labeled 'passive'. The chosen geometry for collision detection was the ‘box’ geometry, and the collision margin was set at 0.000001 m. In terms of friction settings, the parameter was locked at 1, for a friction angle of 45°. Simultaneously, the coefficient of restitution was defined as zero. It's also worth noting that both the translational and rotational damping parameters were precisely calibrated to zero, which effectively eradicates any unforeseen damping effects. These parameter settings ensure the simulation behaviors would most closely replicate the condition assumed by the Housner’s equation, where maximum friction that would not produce any sliding, and perfect rigid body behavior of the block were expected. The simulation's frame rate was picked to run at 50 frames per second. When transitioning to post-processing the results in MATLAB, the time step assigned for numerical integration was 0.02 seconds, ensuring the calculation's time intervals matched up seamlessly. In a bid to maintain the simulation's precision, the sub-steps per frame were determined to be 200, and the solver iteration was cranked up to its maximum permissible value, which is 1000. Across all different aspect ratios and initial angles, these simulations were uniformly "baked", a term in Blender's vocabulary, spanning a total of 500 frames.



Transform

Location:
X 0 m
Y 0 m
Z 1.422 m

Rotation:
X 10.9°
Y 0°
Z 0°
XYZ Euler

Scale:
X 1.000
Y 1.000
Z 1.000

Dimensions:
X 0.2 m
Y 1 m
Z 2.5 m

Figure 3.2 Example of models for aspect ratio 2.5 with initial angle 0.5α: (a) Blender side-view, (b) Blender 3D-view, (c) Vortex side-view, (4) Vortex 3D-view

In this experimental setting, the initial angle of rotation was set to be around the x-axis. Before implementing the initial angle of rotation, the block's orientation needs to be set with in the following manner: the height aligning the z-axis, and width aligning the y-axis, with the thickness of the block aligning the x-axis. After the orientation was fixed, the x rotation in the 'transform' panel would be modified to the desired angle of rotation. After the angle of rotation was set, create a base plate (foundation) by adding another cube mesh, and modify the dimension and orientation of the added mesh in the same manner. To precisely locate the rocking block on the base plate, the 'snap' function in blender needs to be utilized. Similar to snap function in other CAD software, the 'snap' function in blender can precisely align the object's edge, face, or endpoint with another object. To use this function in this scenario, the snap button needs to be activated by single clicking it after it turns blue. The mode of the snap function can be adjusted by clicking the drop bar right to the snap button, and since the goal was to place the block onto the top surface of the base plate, mode 'Face Project' was selected, as shown in figure 3.3.

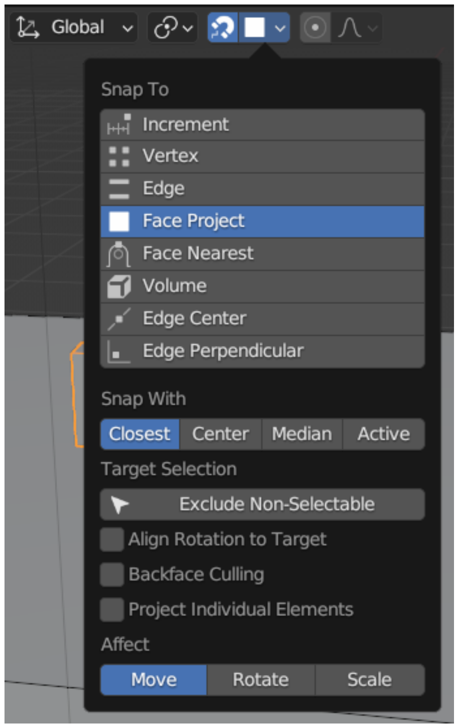


Figure 3.3 Utilizing snap function in Blender

The block can be conveniently moved by using the shortcut G → Z, G button is the shortcut for 'Move', and Z confines the direction of the movement to be along the z-axis. Once the block started moving in the z direction and the snap function was properly tuned and turned on, the block could then be placed onto the base plate.

3.2.1 Bullet physics engine mechanism overview

Bullet engine is a comprehensive physics SDK widely used in videogame and animation physics simulations. It offers an approach to model physical interactions, while employing advanced collision detection, rigid body dynamics, and constraints mechanisms. (Coumans and Rights 2009) One of its standout features is its complex collision detection system, structured in two distinct phases to optimize efficiency. Initially, during the broad phase, the system employed the Dynamic AABB Tree algorithm, which was a spatial partitioning technique. (Xing, Liu, and Xu 2010) This algorithm segmented the simulation space and quickly dismissed object pairs that were spatially separated and unlikely to collide, ensuring rapid preliminary filtering of the colliding objects. On the other hand, for pairs that potentially intersect, the system progressed to the narrow phase, implementing the Gilbert-Johnson-Keerthi (GJK) algorithm. The GJK is renowned for its precision, determining the closest points between convex shapes and confirming collisions. (Lindemann 2009) After these intersections have been detected, Bullet would generate a 'contact manifold', which was a representation that captured the collection of contact points between colliding objects. This manifold was essential for the resulting resolution of collisions, controlling the reaction forces and ensuring realistic simulation responses. In addition to the collision mechanics, Bullet's approach to rigid body dynamics is deeply rooted in classical Newtonian physics. It simulates motion by integrating equations of motion over discrete time steps. This integration accounts for a numerous of factors: external forces like gravity, torques influencing rotational motion, and the inherent inertia of objects, which is determined by their mass distribution. (Coumans and Rights 2009) To further improve the realism of the simulations, Bullet incorporates an extensive constraints system. These constraints, which can be visualized as invisible bonds or joints, restrict, and guide the motion of objects. They range from simple configurations, such as point-to-point attachments that mimic a ball-and-socket joint, to more intricate setups like the 6-DOF (Degrees of Freedom) constraints.

3.2.2 Implementation of python scripts in Blender

Within the Blender software environment, the incorporation of Python scripting presents a variety of choices for advanced functionalities. These capabilities extend well beyond what is readily accessible via the standard graphical user interface provided to users. One of the notable utilities of such scripts is the ability to import data seamlessly from external sources, including spreadsheets such as Excel. Once imported, this data can serve numerous purposes, one of the most significant being to actuate objects within the Blender workspace. In addition to this, Python scripts in Blender facilitate the extraction of specific attributes or parameters of simulation objects. For instance, a researcher or animator may wish to accurately track the angle of rotation, displacement vectors, or even the velocity attributes of objects over time. Through Python scripting, such specialized data extraction becomes feasible, paving the way for more rigorous academic analyses and enhanced visualization techniques. In this specific segment of the research undertaking, the Python scripting infrastructure within Blender was employed with a principal objective: to extract the angular rotation of the block for each discrete simulation frame. To realize this objective, the Python script was architected to perform a series of sequential tasks.

Initially, the code was mandated to establish a CSV (Comma Separated Values) file in a predefined directory, which would serve as the repository for the extracted simulation metrics. Upon successful creation, the file would then be programmatically accessed for data storage purposes. Utilizing the csv_writer.writerow command, a series of header entries were inscribed to the file, namely: "frame", "object", "locX", "locY", "locZ", "rotX", "rotY", and "rotZ". These entries were not merely textual descriptors, but they delineated the structured columns under which corresponding simulation data would be cataloged.

After this initial task, the Python script was designed to identify and target the specific object of interest within the Blender environment – for the purposes of this research, it was the block. This was accomplished by pinpointing the unique object name associated with the block. With the block duly identified, a systematic iteration mechanism was stated. This was expressed in a 'for' loop, the construct of which is detailed in the succeeding section. Through this loop, the script methodically perused the simulation, capturing the required data attributes frame by frame, and inscribing them to the previously established CSV file:

```
for frame in range(scene.frame_start, scene.frame_end + 1):
        scene.frame_set(frame)
        for ob in scene.objects:
            if "BLOCK" in ob.name:
              mat = ob.matrix_world
              loc = mat.translation.xzy
              rot = mat.to_euler()
              scale = mat.to_scale()
              csv_writer.writerow(tuple(chain((frame, ob.name), loc, rot)))
    scene.frame_set(frame_current)
```

In the above Python scripting approach, detailed attention was given to capturing specific spatial and rotational attributes of the designated object(s). Objects that contained the identifier “BLOCK” within their object name were pinpointed for analysis. Specifically, the location coordinates in the x, y, and z directions, as well as the rotational values across these axes, were systematically extracted.

3.3 Modelling process in Vortex Studio

Vortex Studio organizes its files in a distinctive way, segregating the scene, mechanism, and assembly files. Before initiating a physics simulation, the model needs to be structured within the assembly file. This is where both the geometry and the associated contact shape information are conserved. In the Vortex Studio environment, there's an option to construct object geometry from the ground up by incorporating collision geometries directly into the assembly file. It's pertinent to mention that the permitted geometries for addition encompass the following: box, cylinder, sphere, capsule, plane, and composite. For the purposes of streamlining this study, existing geometries were imported using the obj or fbx formats. These imports supplied a foundational geometry, which was subsequently enhanced with the appropriate collision geometry. The geometry of the block was first imported into the assembly file, at this stage, the geometry was just an mesh without any specification on its collision properties just as meshes in Blender before enabling rigid-body physics. To convert the geometry to be a physical object with applied contact model, a 'part' was added to the geometry, and a best-fit collision geometry was selected for the block (box shape). After the geometry was selected, and the control mode for the bottom block was set to 'static', the inertia tensor of the top block needed to be recalculated by pressing 'reset inertia' and then 'compute inertia and COM'. In the mechanism file, the python script that help read the angle of rotation of the block at each time frame was implemented, and in the 'connection' panel, link the global transform data of the block to the out put portal crated by the python script to export data. Before running the simulation, it was crucial to ensure that the frame rate and sub-stepping were set to the proper values by opening the editor file located in the installation folder of Vortex, and change the frame rate to 50 fps, and sub-stepping to 200. After all the parameters were adjusted, the simulation could then be run and data of the angles of rotation could be exported. The friction coefficient could be adjusted through the material table in Vortex Studio, for the purpose of this specific experiment, a friction coefficient of 10 was implemented to ensure no unexpected sliding would occur during the free-rocking phase of the block.

3.3.1 Vortex physics engine mechanism overview

The physics engine built-in in Vortex works similar to that of Bullet. However, in Vortex Studio, the calculation process consists of 4 stages: collision detection, partitioner, constraint solver, integrator. In stage 1, the collision detection between rigid bodies was conducted when contact happen between colliding rigid bodies. In which case the contact constraints were generated consequently and added to the simulation. (CM Labs Simulations 2016) In stage 2, before all the user-defined and autogenerated constraints were processed by the constraint solver, a partition stage was introduced which organized all the rigid bodies and divided them into different subsets. This process was introduced to make sure only objects that are colliding or has contact between each other are grouped together during the simulation process, and therefore ensuring that independent collision problems will be processed in parallel during the next stage of the analysis. The third stage of the simulation was the constraint solver, which takes the user-defined and autogenerated constraints from stage 1 and the grouped subsets from stage 2 and start actually solving the constraints. In this case, stage 2 partitions have significantly reduced the time consumed at stage 3 by allowing problems to be tackled in a parallel manner. In the methodology employed here, the interconnected partitions are first addressed independently and in tandem, resulting in the calculation of interaction forces. These forces are then exerted on the rigid bodies at the junctures of interaction, followed by a subsequent calculation phase. This sequence is carried out iteratively for a predetermined set of repetitions. After determining the constraint forces, the rigid bodies were transitioned to their updated positions by the conclusion of the current simulation phase. This transition is coordinated during the fourth stage for each individual partition concurrently. (CM Labs Simulations 2016)

3.3.2 Implementation of python scripts in Vortex

In the Vortex Studio software platform, users are provided with the flexibility to manipulate simulations and models either via a user-friendly graphical interface or by implementing Python scripts. This dual approach offers both ease-of-use for general tasks and the advanced computational capabilities for more complex operations. Utilizing Python scripting within Vortex allows users to automate repetitive tasks, run simulations with varying parameters, and export specific datasets for further analysis. For instance, a researcher might be interested in exporting motion data of a simulated object for post-simulation statistical analysis or graphical representation. To facilitate this scripting capability, Vortex provides a set of API functionalities available through the VxSim module. Here's a simple demonstration of how one might initiate such a script:

*from VxSim import **

*from math import **

To compute and store the angle of rotation data of the block during the simulation, a post step function was defined, and its main task is to compute the angle (ang) between the z-axis and the third column of the orientation matrix (ori). The result is stored in the theta output.

- It extracts the orientation matrix tm of an object and calculates its rotation.
- Using the z-axis of this rotation (ori[2]), it computes the dot product with the global z-axis (straight up).
- Based on the y-component of ori[2], it determines whether the angle should be positive or negative.
- The computed angle ang is set as the output theta.

Connection panel in Vortex also needed to be utilized to set the input of the matrix ‘tm’ from the python script and connect it to the output of the 'world transform' matrix for the rocking block, as shown in figure 3.4. After these setups, the python script was applied, and the rocking angle data would be obtained for further analysis.

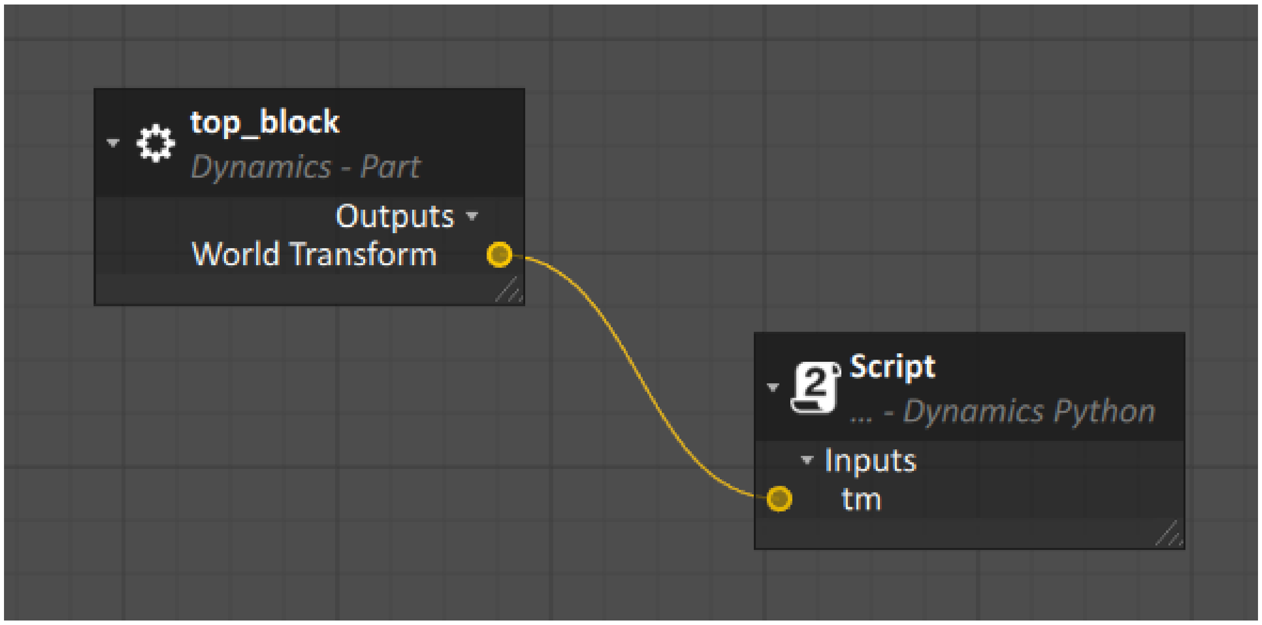


Figure 3.4 Connection panel in Vortex Studio and setup for rocking angle analysis

3.4 Overview of proposed numerical parametric study

The parametric study of the free rocking macro blocks was conducted according to the methodology proposed by Ma and colleagues. (Ma, Parshottam, and Montalla 2018) In their study, 102 cases of free rocking block were tested, and 96 sinusoidal base acceleration simulation cases were carried out. In the scope of this study, a set of a relatively different geometries and aspect ratios were chosen, as listed in table 1.

Table 1 Rocking macro-block combinations tested in this study

Aspect ratio (*h*/*b*)	Fixed base, *b* (m)	Fixed height, *h* (m)	Initial angle of rotation (θ_o) (degrees)		
			0.2α	0.5α	0.99 α
0.5	0.5, 1.0, 1.5	2.75	12.68	31.72	62.80

1.0	0.5, 1.0, 1.5	2.75	9.00	22.50	44.55
1.5	0.5, 1.0, 1.5	2.75	6.74	16.85	3.35
2.0	0.5, 1.0, 1.5	2.75	5.31	13.28	26.30
2.5	0.5, 1.0, 1.5	2.75	4.36	10.90	21.58

For the testing specimens, the dimensions of the blocks were selected based on two parameters separately, namely the width and the height. Instead of implementing macro block with various height, a fixed height of 2.75 was chosen to resemble a typical height of the wall for a masonry structure. For another set of specimens, the width of the blocks was chosen to be 0.5, 1.0, and 1.5 meters which are also reasonable sizes for a unit in a masonry structure. For each designated width and height, the other unspecified dimension was determined according to the aspect ratio, for example, for a fixed base of 1.0m with an aspect ratio of 0.5, the height of the block would be 0.5m. For a fixed height of 2.75m block with a 1.0 aspect ratio, the base of the block would be 2.75m. After the establishment of the above dimensions of blocks, each block would be subjected to an initial angle of rotation with respect to their critical angle α, namely 0.2 α, 0.5 α, and 0.99 α. As mentioned before, critical angle stands for the angle the maximum angle of rotation that the block can sustain before overturn. (Housner 1963) To ensure the block does not accidently overturn, the largest initial angle of rotation was chosen to be 0.99 α instead of 1 α. After the initial setup was done, the blocks were then allowed to go through free-rocking motion for 5 seconds, at a frame rate of 50 fps. To ensure idealistic performance, the rotating motion of the block was constrained around the x-axis only. The angle of ration at the centre of the block was retrieved at each time frame and plotted against the theoretical values derived from the Housner's equation. The resulting plots are shown below in figure 3.5. The errors were calculated and compared for vortex and blender data at the end.

3.5 Simulation outcomes and comparison with the analytical solutions

In this section, the simulation results from Blender and Vortex were plotted against analytical results produced by the Housner's equation, considered the same cases summarized in Table 1.

Overall, an excellent agreement between numerical and analytical results was found as it can be gathered from Figure 3.5, where the vertical axis is the angle of rotation θ normalized with respect to the critical angle α, and the horizontal axis is the simulated pseudo-time in seconds. The difference between the three curves is negligible, as an almost perfect match was obtained for all cases (only a small selection is depicted in Figure 3.5 due to space constraints). This represents a considerable improvement with respect to previous works on the topic (Ma, Parshottam, and Montalla 2018), were errors up to 15% were reported for squat blocks. Increasing the level of zoom, a very small discrepancy between the simulation and analytical curves can still be found, albeit that is believed to be related to small error accumulation in the MATLAB post-processing and ODE calculation.

The mean absolute error between the numerical and analytical solutions was calculated by taking the absolute difference in amplitude for corresponding time values, and then averaging the results. This difference in θ values through time for both motions was normalized by the critical angle α, to reflect how the aspect ratio of the block affects error. To better reflect the inaccuracies for the actual rocking of the block, a "data cutoff" was applied. This simplified strategy ensured that the error was calculated only for the instants in which either the analytical or numerical simulation were still undergoing significant rocking, see Figure 3.6.

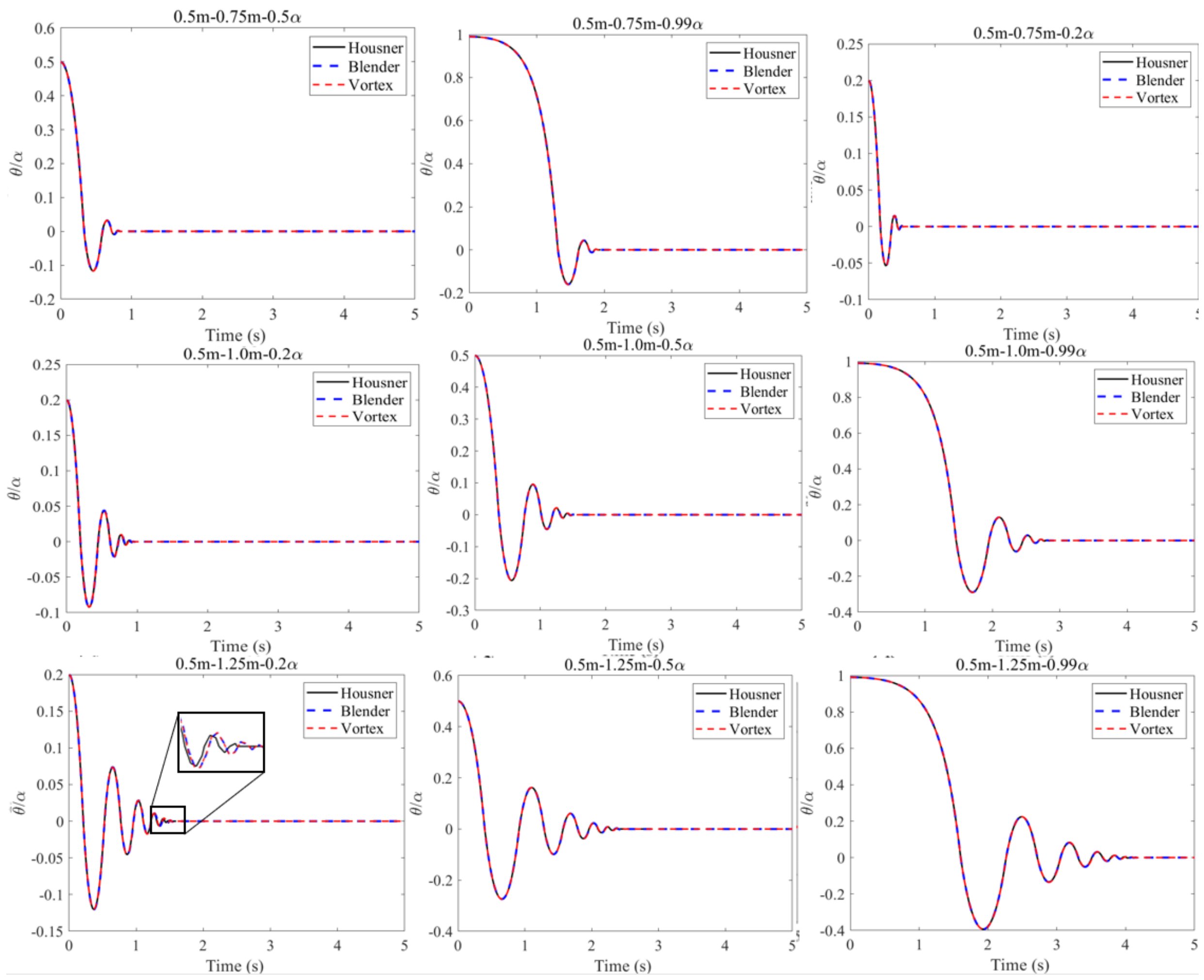


Figure 3.5 Simulation results from blender and vortex with analytical solution; (a) 0.5m x 0.75m 0.2, 0.5, 0.99alpha, (b) 0.5m x 1.0m 0.2, 0.5, 0.99alpha, (c) 0.5m x 1.25m 0.2, 0.5, 0.99alpha

Once both the analytical and numerical modelling predictions detected negligible rocking, the points are no longer considered for the mean absolute error.

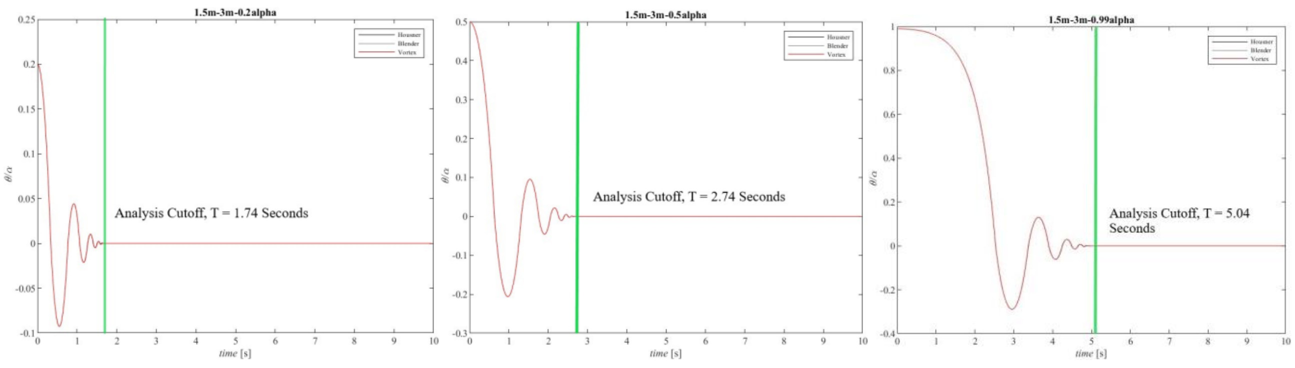


Figure 3.6 Example of "data cutoff" points (see green line) for the absolute error analysis

In order to assess the precision of the simulations during the initial phases prior to any impact, a metric was employed - the mean absolute error, particularly focusing on the freefall period, delineated as T/4. A visual representation of these analytical outcomes is encapsulated in Figure 3.7. Upon examination, it becomes evident that the absolute errors across all aspect ratios are notably miniscule. Drawing a comparative analysis between the simulations from Blender and Vortex, preliminary observations suggest that Blender's simulations manifest a slightly superior accuracy quotient. However, it is imperative for readers and scholars to exercise caution while interpreting this observation. Given the exceptionally small magnitude of the errors, any perceived superiority in one simulation platform over the other might not signify a consistent trend or inherent advantage in real-world applications.

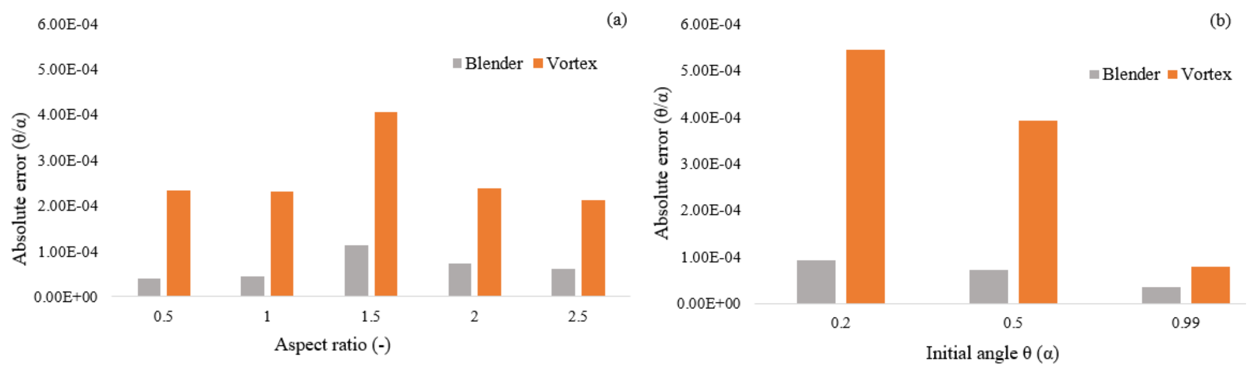


Figure 3.7 (a) The mean error in *T/4* time range for all aspect ratios, (b) mean error for blocks with different initial angle of rotations during the *T/4* time range

For a more granular understanding and to yield a broader perspective on the distinctions observed between the chosen simulation engines, Figure 3.8 digs into the discrepancies relative to the analytical solutions. These differences have been normalized against the maximum observed value to provide a standardized metric for comparison. This visualization aids in discerning any nuanced variations between the engines and facilitates an in-depth exploration of their performance metrics.

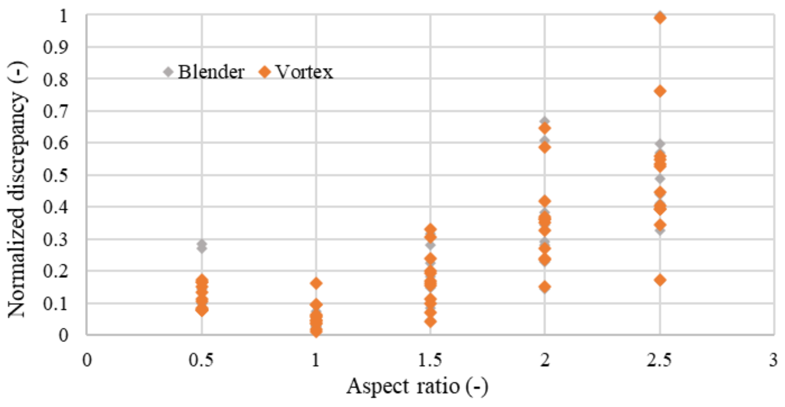


Figure 3.8 Normalized discrepancy by the maximum error for each aspect ratio

It should be noted that the computational time required for simulations in both Blender and Vortex was significantly shorter in comparison to the conventional Finite Element Method (FEM) and Discrete Element Method (DEM) numerical models. On a typical benchmark, executing a simulation lasting 10 seconds was accomplished in under 20 seconds when utilizing a standard office-grade laptop. This computational efficiency renders our forthcoming research especially pertinent for structural and earthquake engineers. The relevance is further heightened when one considers the extensive parametric analyses and evaluations related to risk and reliability factors associated with Unreinforced Masonry (URM) components governed by rocking mechanisms.

To visualize the time efficiency of Blender on simple simulations, the following figure 3.9 gives a comparison of the simulation time for a typical set-up.

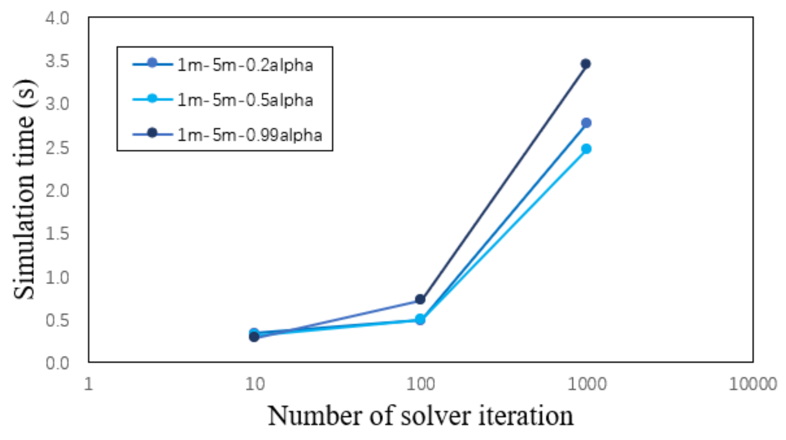


Figure 3.9 1m-5m simulation time comparison for using different solver iterations

The figure presented above provides a comprehensive illustration detailing the influence of the number of solver iterations on the overall simulation time. Initial observations indicate a minimal discrepancy in simulation times with an iteration count as low as 10. However, as one progresses to higher iteration values, the divergence becomes increasingly evident. This increasing discrepancy could be because, for larger initial angle of rotation, the initial falling period before the first contact was relatively larger, which might have resulted in longer and more complicated calculation process undergone in the physics engine calculation, and therefore resulting in longer overall simulation time. An important observation to emphasize is that, while there was a marked increase in simulation time as the solver iteration approached 1000, culminating in a simulation duration of 3.5s, this duration remains impressively swift for a 5-second simulation when compared to classical DEM approach.

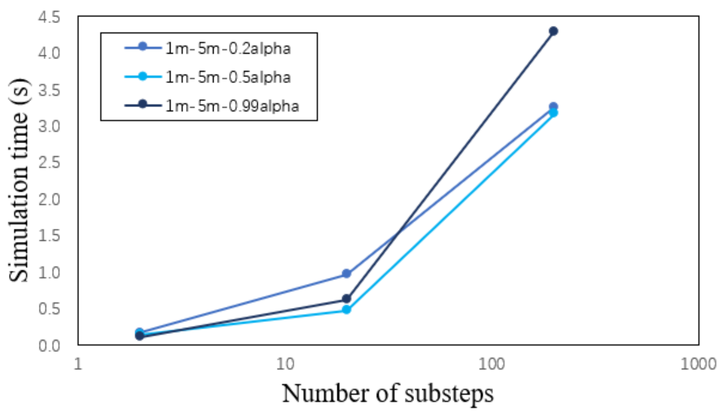


Figure 3.10

Figure 3.10. 1m-5m simulation time comparison for using different number of sub-steps

In figure 3.10, the similar simulation time comparison is made whilst this time with differing number of sub-steps. The trend of time increasing with increasing initial angle does not seem as prominent as the previous results. However overall speaking, the simulation time increases as the number of sub-step increases and the initial angle of rotation rises.

3.4.1 Initial angle at 0.2 alpha

In the examination of blocks subjected to an initial inclination of 0.2α, both Blender and Vortex simulations produced largely consistent results. As illustrated in Figure 3.7.b, upon analyzing the simulation outcomes with varied starting angles – normalized in terms of α – a discernible trend appears: errors tend to be more evident for blocks with smaller initial angle of rotations. Conducting a side-by-side comparison of the two physics simulation engines shows that Vortex tends to demonstrate more significant discrepancies in its results compared to Blender. Nonetheless, it's worth noting that the magnitude of the error, regardless of the engine, remained relatively minor. A potential explanation for these deviations observed in Vortex could be attributed to its utilization of fewer sub-steps in the default simulation configurations.

3.4.2 Initial angle at 0.5 alpha

A close examination of Figure 3.7.b reveals that the absolute error in the rocking angle derived from simulations, when benchmarked against Housner's predictions, indicates a higher level of precision. The degree of accuracy is notably refined, with a minimal absolute error ratio of 0.0004 theta/alpha, especially when compared to the initial angle case of 0.2 alpha. The trend of errors observed from both Blender and Vortex simulations demonstrates a convergence, particularly in the 0.2 alpha instance. However, Blender consistently yields results with marginally lower absolute errors, suggesting a more precise alignment with the theoretical model. This distinction in accuracy between Blender and Vortex becomes more apparent when assessing the deviations from the expected rocking behavior, as Blender exhibits a closer adherence to Housner's theoretical framework.

3.4.3 Initial angle at 0.99 alpha

In graph 3.7.b, the error bars correspond to the initial angle theta normalized by alpha, where a 0.99 alpha represents an initial angle very close to the critical angle at which the block is expected to topple. The Blender simulation shows significantly lower errors across all aspect ratios compared to Vortex, indicating a higher precision in capturing the dynamics of rocking motion close to the tipping point. At the 0.99 alpha initial angle, the Blender engine’s error remains considerably lower than that of the Vortex engine. This suggests that Blender may have a more refined calculation mechanism for handling near-critical rocking motions where the nuances of the physics are particularly sensitive. It is also likely that Blender's handling of contact dynamics and inertia effects at high angles is more accurate, which is crucial for predicting the behavior of blocks nearing the point of overturning. In practical terms, the lower errors at high initial angles are significant for safety evaluations, especially in seismic analyses where understanding the behavior of structures at near-collapse conditions is critical. The smaller the error, the more reliable the simulation, and thus the more confidence engineers and researchers can have in the predictive models used for assessing structural responses to seismic events.

3.6 Effects of tuneable parameters in Blender

In Blender, there are four major tunable parameters that could be adjusted in the physical simulation process, namely the collision geometry, the collision margins, the solver iterations, and the sub-steps. Each one of these parameters could affect the simulation accuracy and simulation speed to a different extent. It should be noted that all the below parametric studies were made with only one considered parameter change, all the other tunable parameters were remained unchanged to ensure validity of the comparison. As a reference, the chosen parameter values that have been employed for the purposes of this specific study are listed below:

- Solver iteration = 1000
- Simulation steps = 1000/s or 10000/s (use 1000 for 0.99a)
- Collision detection mechanism: Box collision
- Collision margin = 1 μm (0.000001 m)
- Simulation rate = 50 fps
- Velocity damping = 0
- Bounciness = 0
- Friction coefficient = 1

For the following sub parts of this section, all the above values are going to be unchanged throughout the tests except for the specified parameter chosen in each subsection.

3.5.1 Collision geometry

As mentioned previously, the most appropriate collision geometry should be selected to best-fit the object before conducting rigid-body simulations. In this specific case, since the object is a perfect rectangular block, the 'box' geometry would be the best-fit geometry, and it would provide a simplified contact model that avoid inaccuracies. As the simulation results shown in figure 3.11 below, 'box' geometry yields the closest simulation result when compared to the analytical solution (Housner's equation), while convex-hull and mesh geometry, although also fit the geometry of the block, since they consist of more complex contact surfaces and collision detection mechanisms, the simulation results would deviate from the analytical solution.

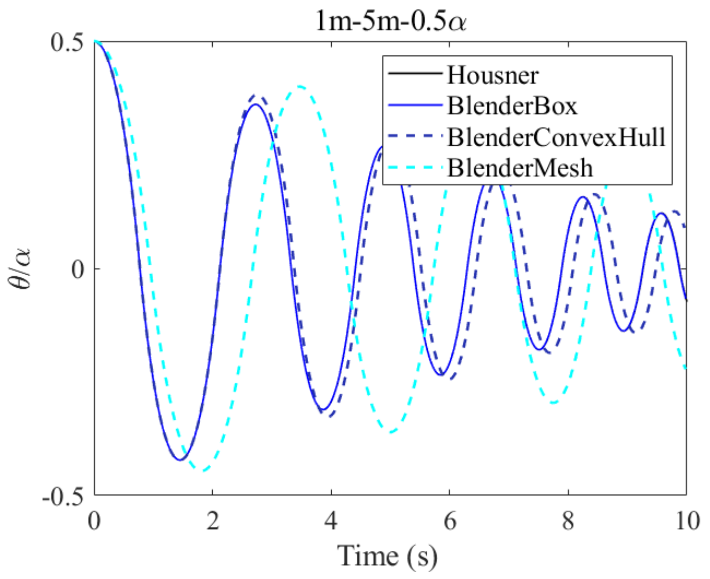


Figure 3.11 1m-5m simulation results using different contact geometries (box, convex-hull, mesh)

The 'mesh' geometry is useful for object with irregular shapes and smooth surfaces, since this contact geometry consists of a matrix of triangle shaped surfaces that would envelop the entire object's surface. Therefore, it would be difficult for 'mesh' geometry to envelop objects with hard edges and acute corners since it might create errors when the mesh trying to model these rigid shapes and ultimately results in inaccurate contact calculations when collision occurs.

3.5.2 Collision margins

The collision margin in Blender is defined as the threshold of distance near the surface of the object where collisions are still considered. (Blender Documentation Team 2023) The collision margin was implemented to improve the performance and stability of the rigid body dynamic responses. As aforementioned, the implementation method varies depending on the collision geometry selected, while some geometries embed it, and some geometries have a visible gap between the margin surface and the object surface.

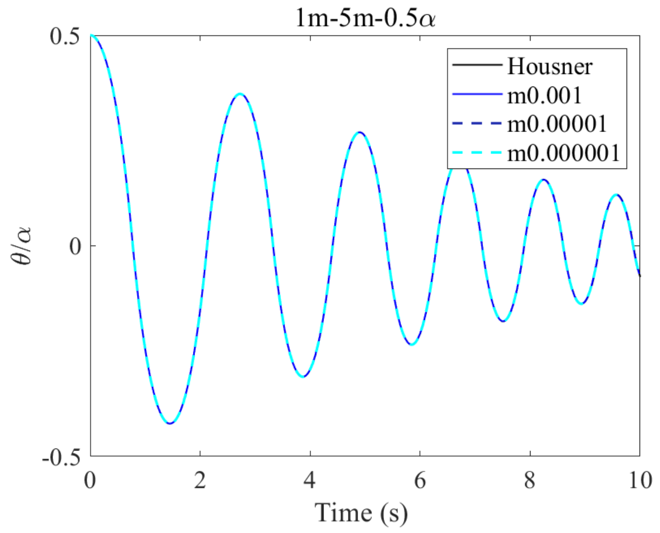


Figure 3.12 1m-5m simulation results using different number of solver iterations (10, 100, 1000)

In a comprehensive assessment, with all other parameters held constant, variations in the collision margin appear to exert a minimal impact on the accuracy of the simulation outcomes, as shown in figure 3.12.

3.5.3 Solver iterations

The solver iteration in blender is the number of numbers of constraint solver iterations made per simulation step, and higher values are more accurate but could result in slower simulation speed. Increasing this parameter makes constraints and object stacking more stable. (Blender

Documentation Team 2023) However, although in simpler simulations, solver iterations do not seem to contribute much to the accuracy of the simulations, it could play a more important role in more complex simulations when collision calculations became more complex.

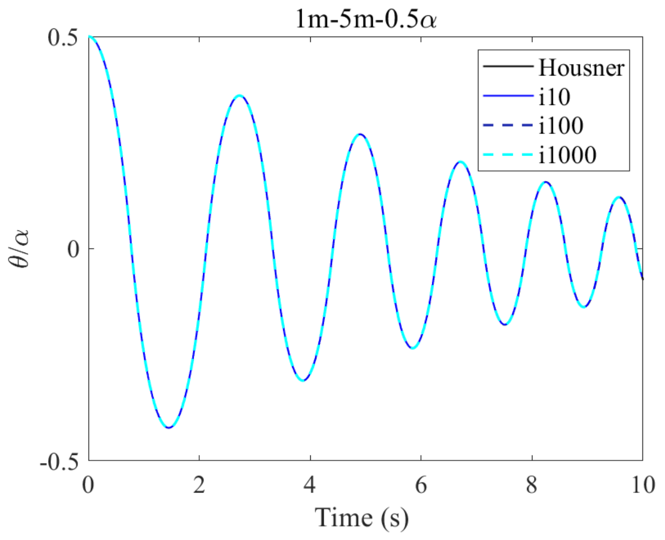


Figure 3.13 1m-5m simulation results using different number of solver iterations (10, 100, 1000)

From simulation results in blender, the difference between simulation results with different solver iterations were not obvious. As shown in figure 3.12, the difference between simulation results with solver iterations 10, 100, and 1000 are almost unidentifiable.

3.5.4 Sub-steps

The sub-steps in Blender rigid body simulation stands for the number of simulation steps taken per frame. (Blender Documentation Team 2023) Normally, higher sub-steps provide better accuracy while increasing the simulation time.

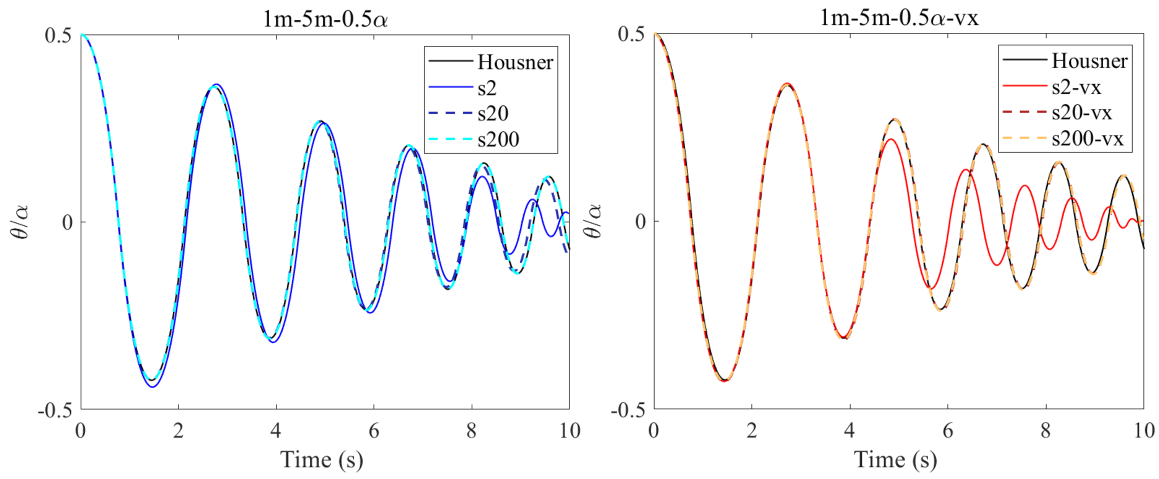


Figure 3.14 1m-5m simulation results using different number of sub-steps (2, 20, 200)

From the above figure, simulation sub-steps affect the accuracy of the simulation to a great extent. For both Blender and Vortex, the accuracy of the simulation significantly decreases as the sub-step was 2, but for sub-steps higher than 20, the simulation became considerably more stable and accurate. For Vortex, this difference was even more prominent as it presented a lower accuracy compared to Blender when the sub-step was 2, and a better accuracy when the sub-steps are higher than 20.

3.7 Effect of tuneable parameters in Vortex Studio

In Vortex Studio, the tuneable parameters exposed to the users are relatively less then that of Blender. The main tunable parameter that could potentially affect the accuracy of the physics simulations was the simulation sub-steps. The simulation sub-steps parameter, which is available in many simulation platforms including both Vortex Studio and Blender, plays a crucial role in determining the accuracy and stability of a simulation. Increasing the number of simulation sub-steps can enhance the accuracy of the simulation as it captures more details within each timestep. However, this also demands more computational resources, potentially slowing down the simulation. Therefore, it's a balance between accuracy and computational performance.

3.7.1 Sub-steps in Vortex

Sub-stepping in vortex could be adjusted just as in Blender, as shown in figure 3.14, simulation results in Vortex with different sub-steps were plotted against the analytical solution. Due to difference in the collision calculation and integration methods, the simulated results in Vortex were different from the Blender results. While when the sub-step = 2 Vortex appear to be less reliable, once the number of sub-steps was larger than 20, Vortex appears to be relatively stable and accurate.

Chapter 4 – Forced rocking motion of rigid URM macro-blocks under sinusoidal base motion

4.1 Overview of proposed numerical parametric study

The first portion of the forced rocking test was to compare the rocking behavior of perfect rectangular blocks under sinusoidal ground acceleration with the predicted results from the Housner's equation. The models of the blocks would be constructed in Blender and Vortex environments, with the proposed aspect ratios as stated in Table 4.1.

Table 2 Forced rocking sinusoidal ground motion combinations.

Aspect ratio (*h/b*)	Fixed base, b (m)	Amplitude (activation accel)	Frequency （Hz）
1.0	0.5, 1.0, 1.5	2, 4, 6	0.5, 1.5, 3
1.5	0.5, 1.0, 1.5	2, 4, 6	0.5, 1.5, 3
2.0	0.5, 1.0, 1.5	2, 4, 6	0.5, 1.5, 3
2.5	0.5, 1.0, 1.5	2, 4, 6	0.5, 1.5, 3

108 analyses in total

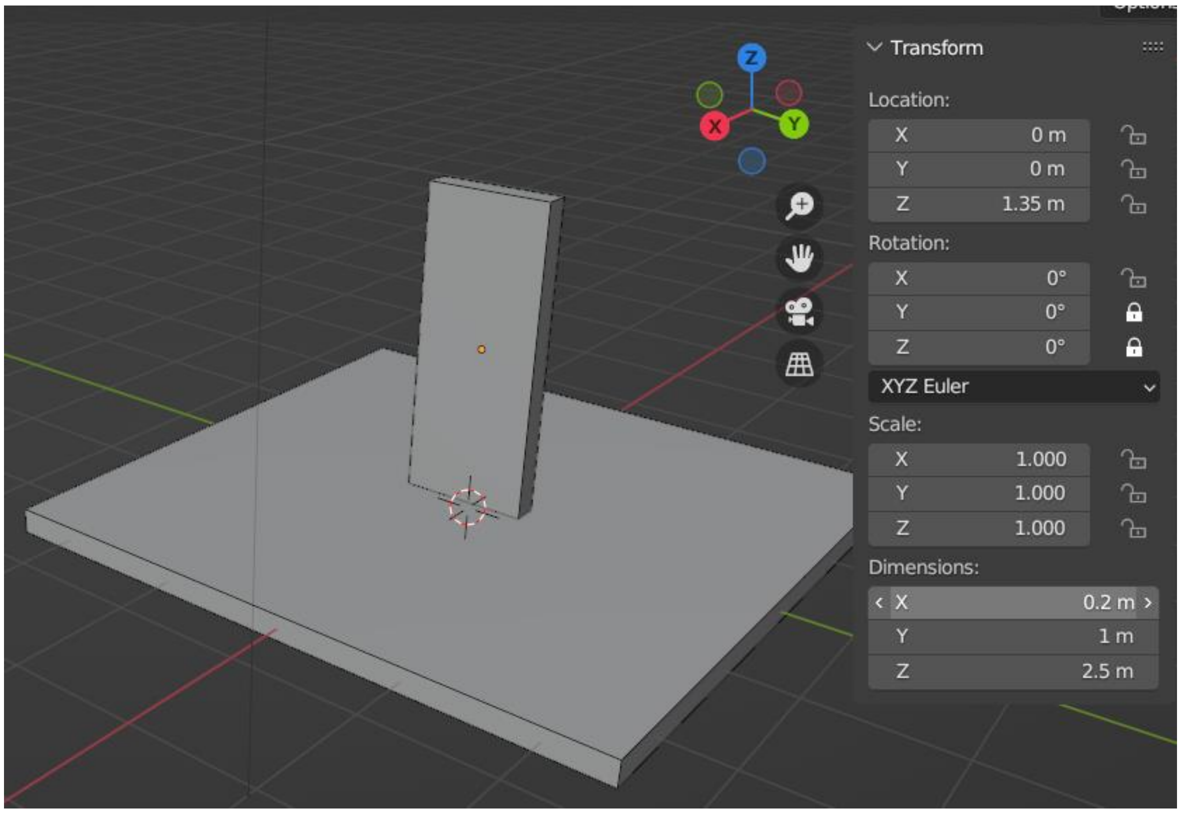
The above table listed out all aspect ratios, dimensions, amplitudes and frequencies of the ground acceleration tested in this part of the research phase. The aspect ratios chosen were characteristic values of common URM elements, and the dimensions of the blocks were also chosen based on this rational. The amplitudes of the sinusoidal acceleration were normalized by the activation acceleration of the respective aspect ratio. The activation acceleration was the acceleration that would initiate the rocking behavior of the macro block. This value was calculated by the following formula.

$$A_{min} = 9.81 \times \tan\left(\tan^{-1}\left(\frac{b}{h}\right)\right) \qquad (15)$$

The activation acceleration normalization method would ensure that the block will go into rocking motion with all the applied ground motion cases. The amplitude for the ground motion cases were chosen to be 2, 4, and 6 times the activation acceleration, and the frequency tested were 0.5, 1.5 and 3 Hz. The frequency were some typical ground motion frequencies captured in earthquake events. From the study by Nishida and colleagues, the dominant frequency for earthquake events in low and medium frequency band were from 0.1 to 10 Hz. (Nishida 2017) The numbers 0.5, 1.5 and 3.0 Hz were chosen since rocking motion for macro blocks were the most obvious when subjected to the aforementioned ground motion frequencies.

4.2 Blender modelling approach for single block simulation

In the single block rocking test with sinusoidal base acceleration, the primary characteristic from the free rocking test is the introduction of a periodic external motion applied onto the base plate, simulating conditions like seismic activities or other oscillatory forces. The modeling method for both the block and the base remains consistent with the free rocking test, ensuring uniformity and comparability between the two tests. While in the free rocking test, the block was initiated with a tilt, introducing potential for spontaneous rocking; in the sinusoidal base acceleration test, the block starts in a neutral, upright position. This ensures that any rocking motion observed is primarily a consequence of the imposed sinusoidal acceleration, rather than any initial conditions. Positioning the block with its height along the global z-axis, width along the global y-axis, and thickness along the global x-axis ensures a standardized orientation. This standardized setup aids in deriving meaningful conclusions from the test, as the block's response to the base acceleration can be analyzed in a consistent frame of reference.



Transform

Location:
X 0 m
Y 0 m
Z 1.35 m

Rotation:
X 0°
Y 0°
Z 0°

XYZ Euler

Scale:
X 1.000
Y 1.000
Z 1.000

Dimensions:
X 0.2 m
Y 1 m
Z 2.5 m

Figure 4.1 1m-2.5m block model in Blender environment with corrected scaling

In this phase of the research, attention was paid to ensuring the block's placement and scale were consistently set up, which is crucial for obtaining reliable and reproducible results. Initially, the block's bottom surface was precisely aligned or 'snapped' to the top surface of the base plate. This alignment ensures that the block maintains a uniform starting position across multiple tests, eliminating variability due to the block's initial position. After properly set up the position of the block, it was equally important to ensure the block's scale was consistent, especially if any adjustments to its dimensions were made during the setup phase. While modifying dimensions might be necessary to accommodate to specific test requirements, it was critical that the block returns to a standard scale post-adjustment. This normalization step was essential to ensure that any observations or results from the test could be attributed to the test conditions, rather than variations in the block's scale. In Blender, resetting the block's scale was achieved through a specific command sequence. By pressing Ctrl+A, a command menu or action list was called. From this list, the 'scale' option was selected, effectively resetting the block's scale to a value of 1. This action ensures that the block maintains its intended dimensions and any results derived from subsequent tests would be consistent and valid.

4.2.1 Modelling and rigid body physics setting

the settings for rigid body physics largely remained mostly consistent as the free rocking test phase, ensuring consistency between different phases of the study. The first step involved designating the block's rigid body dynamics to an 'active' state. An active state implies that the block is free to move and respond to forces and constraints within the simulation environment. To ensure that the block's motion closely mimics real-world physics, its mass was determined. This was accomplished through the software interface by navigating to the ‘Object’ menu, then selecting ‘rigid body’, and finally choosing the ‘calculate mass’ option. This function uses the block's dimensions and a specified material density to compute its mass, ensuring that the block's inertial properties in the simulation are representative of a real-world counterpart. The base plate serves as the foundation upon which the block rests. Its rigid body type was set to 'passive', a setting that ensures the base plate remains stationary and unaffected by other forces or objects within the simulation. This is crucial for the initial setup, as the block should only experience motion due to the imposed sinusoidal acceleration and not any other unrelated factors. While the base plate started in a fixed state, it would eventually be subjected to sinusoidal motion. To accommodate this, the animation function had to be activated within the settings menu. This step prepares the simulation environment to apply time-varying displacements to the base plate, thus inducing the intended sinusoidal motion during the test phase.

4.2.2 Defining the sinusoidal base motion

Sinusoidal motion offers a straightforward and consistent pattern. Its repetitive and predictable nature makes it an ideal choice for creating controlled experimental conditions. This simplicity is advantageous in both laboratory setups, where physical models might be tested, and in computer simulations where mathematical representations are examined. Given its deterministic nature, sinusoidal motion allows researchers to conduct a meticulous post-analysis. The results derived from sinusoidal motions can be analyzed with greater depth, allowing for the extraction of detailed insights. This ensures that the findings are not just surface-level observations but are backed by comprehensive data analysis.

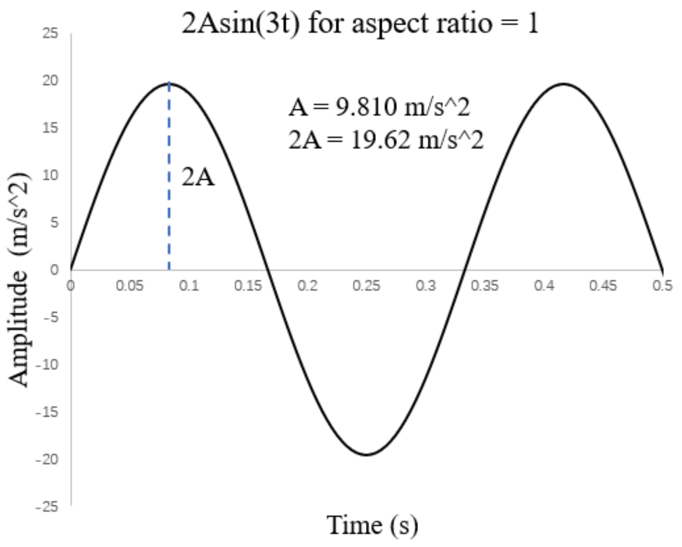


Figure 4.2 Graphical representation of the inputting sinusoidal ground acceleration case 2Asin(3t) for aspect ratio = 1

The results from sinusoidal motion tests provide a consistent dataset that can be subjected to various statistical methodologies. This helps in understanding the range, variance, and other statistical parameters of the data, which are crucial for validating the findings and drawing robust conclusions. The sinusoidal base acceleration in this part of the research was defined in a specific way as follows:

$$\text{Acceleration} = a(t) = 2\text{Asin}(3\text{t}) \qquad (10)$$

In the expression 2Asin(3t), the term 2A signifies the amplitude of the sinusoidal ground acceleration being input. Here, A delineates the critical acceleration threshold required to trigger the onset of a rocking motion within a rectangular block of a designated aspect ratio. Specifically, the magnitude of A corresponds to the activating acceleration. The determination of A is underpinned by the subsequent formula:

$$A = 9.81 * \tan\left(tan^{-1}\left(\frac{b}{h}\right)\right) \qquad (11)$$

In the given context, the parameters b and h represent the width and height of the block, respectively. By characterizing the amplitudes of the sinusoidal motion in terms of multiples of the activation acceleration A, it guarantees the initiation of rocking across all examined aspect ratios of blocks. This methodical approach ensures consistent and comparative behavior is observed, irrespective of the block's specific dimensional characteristics. Furthermore, the coefficient "3" in the term 3t symbolizes the frequency of the sinusoidal motion. Specifically, a value of "3" indicates that the sine wave oscillates with a frequency of 3Hz. The choice of this frequency has implications on the dynamic response of the system, making it a pertinent parameter in seismic studies. The formula in consideration is integral to describing the character of the sinusoidal motion. Utilizing a structured computational spreadsheet, the displacement of the base plate was deduced for each discrete frame. This deduction is premised upon the acceleration formula, as illustrated in equation (10). Upon further examination of the data set, for every individual frame, the acceleration pertaining to the y-axis undergoes a transformation process. It's subject to integration—a methodological accumulation—which yields its velocity. This velocity, in turn, is subjected to an additional layer of integration, concluding in the determination of the displacement value. The significance of this displacement is underscored in the context of the Blender python script. In this computational environment, where graphical objects are manipulated, it was exclusively the displacement data that can be operationalized. Thus, for the desired articulation of object motion within the script, it's imperative to transmute acceleration and velocity datasets into displacement data. In the ground motion input file, the first column would be the time column, which each successive frame time would be the time at previous frame time plus the frame rate. For example, at frame 1, the time would be 0, and at frame 2, for a 240-fps simulation, the time would be 0 + 1/240 seconds, and so on so forth. The second column in the file would be the acceleration in the y direction calculated based on equation 10, with the calculated values of amplitude based on the activation acceleration, and the frequency of the motion converted to the unit of 2pi/period from hertz, and finally the time calculated from the frame rate of the simulation. Integration on the acceleration equation was then performed according to definition, and as shown in equation (12) and (13), the displacement was derived at the end, and later imported into Blender using python script.

$$\frac{d}{dt}v(t) = a(t)$$

$$\frac{d}{dt}x(t) = v(t)$$

$$v(t) = \int a\,dt + C_1 = at + C_1$$

When initial velocity v(0) = v_0,

$$v_0 = 0 + C_1$$

$$v(t) = v_0 + at \qquad (12)$$

Then,

$$x(t) = \int (v_0 + at)dt + C_2$$

When we have x(0) = x_0,

$$x(t) = v_0 t + \frac{1}{2}at^2 + C_2, then\, x_0 = 0 + 0 + C_2$$

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2 \qquad (13)$$

Drawing from the previously discussed formulas, it was obvious that the displacement for each specific time frame, delineated by the simulation time t, was systematically derived. This process was depending upon the acceleration values known for the current step, while simultaneously factoring in the displacement and velocity data from the preceding steps. It's worth noting that both the initial displacement and velocity started from a null value, signifying that the block's simulation began from a state of absolute rest.

In the latter phase of the forced rocking tests, the trimmed block models were evaluated against experimental data. The approach for defining the sinusoidal base motion diverged from the initial methodology. Rather than scaling the amplitude of the sinusoidal motion by the activation acceleration, the amplitude was specified in terms of the peak displacement achieved during the motion. For instance, a sinusoidal base motion described as 2Asin(3t) would represent an oscillation with a peak-to-peak amplitude of 4 mm at a frequency of 3 Hz. This alteration in defining the motion's parameters allows for a solid understanding of the block's response under variable displacement conditions, thus providing a more robust simulation that closely mirrors practical seismic events. In this case, the ground motion case would resemble the form as shown in figure 4.3 as below.

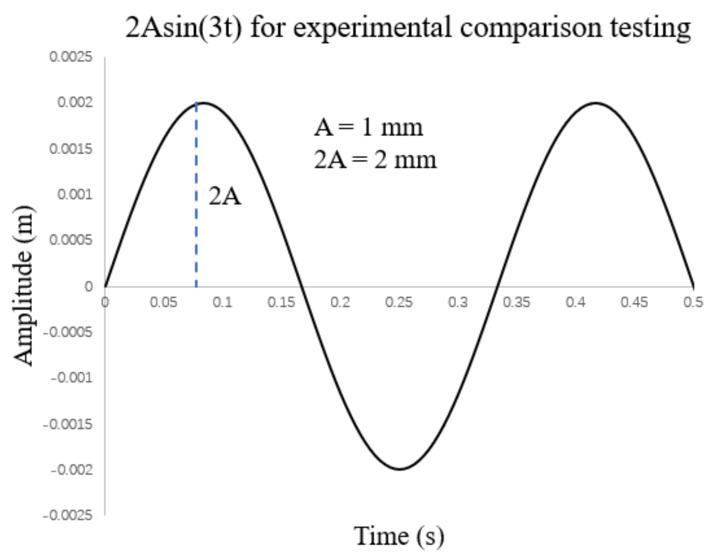


Figure 4.3 Graphical representation of the inputting sinusoidal ground acceleration case 2Asin(3t) implemented in the experimental comparison testing scenarios

In this research scenario, the amplitude of the sinusoidal ground motion is established independently of the block's activation acceleration, thus cancel out the influence of the block's aspect ratio on the motion's application. Consequently, this uniform approach allows the sinusoidal ground motion to be consistently applied across all macro blocks under analysis, regardless of their individual dimensions or aspect ratios. This methodology enables a standardized assessment of seismic responses, permitting a more direct comparison of outcomes across different block configurations. However, it should be noted that due to the nature of the way of defining the ground motion, it does not guarantee rocking initiation during the simulations. Hence, there would be cases where the block does not go into rocking motion, and that specific testing trial would be trivial.

4.2.3 Utilizing python script to implement base motion in Blender

In Blender, python scripts could be utilized to import motion data through spreadsheets and therefore control the motion of selected objects. For this section of the research, a python script was used to assign the displacement data onto the base moving plate. First of all, the script started with importing the necessary libraries that would be needed for later operations. The command import pby was used to import the Blender Python API that allows for scripting and automation in the Blender environment, and import csv was to import python's built-in library for reading and writing CSV (Comma-Separated Values) files. The next step was the variable initiation, where the frame counter was set to 1 by using the command f = 1, also defining the file path of the csv by using csv_path. Next step would be importing the csv file data and assigning the displacement information to the designated object, this was done using the following loop:

```
ob = bpy.context.object
with open(csv_path) as file:
    groundmotioncsv = csv.reader(file, delimiter=",")
    for row in groundmotioncsv
        ob.location = [float(v) for v in row[:3]]
        ob.keyframe_insert("location", frame=f)
        f += 1
        if f == 3000:
            break
```

In the above script, '*ob = bpy.context.object*' was used to select the object and assigning the object to 'ob'. The phrase *'[float(v) for v in row[:3]]'* Sets the location of the active object (ob) in Blender to the first three values of the current row (assumed to be x, y, and z coordinates). It converts the values in the row to floats. The phrase *'ob.keyframe_insert("location", frame=f)'* Inserts a keyframe for the object's location at the current frame f. This means that at frame f, the object will be at the specified location. The *'if'* condition checks if the frame counter has reached 3000. If it has, the loop breaks, and the code stops processing further rows. This would be a protection to ensure that not too many frames were being processed.

4.3 Vortex modelling approach for single block simulation

In the modeling process using Vortex Studio, the initial step involved generating distinct files, each designated for specific functions. The mechanism file integrates the script, while the connection panel organizes the input and output operations directed by the script. The assembly file retains the geometric data of the objects, along with their transformational details.

To facilitate rigid body simulations, 'parts' were established for each object, which is a critical step in ensuring accurate physical behavior in the simulation environment. For each part, appropriate collision shapes were carefully selected to match the mesh's contours, ensuring that the physics engine accurately calculates interactions during simulations. This detailed setup allows for the precise application of forces and constraints, which is vital for the realism and reliability of the simulated dynamics.

4.3.1 Model generation in Vortex

In this phase of the study, the rectangular blocks utilized for sinusoidal base acceleration testing were crafted following the same procedure as the initial phase, which focused on examining free rocking motion. In this iteration, the blocks were designed to start from an initial position where their bottom surfaces were entirely in contact with the base plate. The sinusoidal ground acceleration was then applied to the base plate, facilitated by a motor part and guided by instructions from a Python script. This methodical approach ensured consistency in the setup, allowing for a controlled environment in which to observe and measure the blocks' responses to the induced motion.

4.3.2 Implementing sinusoidal base motion in Vortex

In Vortex, the approach to implementing sinusoidal base motion differs from that in Blender; the motion is calculated and applied to the base plate in a singular Python script. The script is architected to compute the designated ground motion and the base plate's acceleration at each time frame, utilizing the amplitude and frequency as input parameters. Subsequently, it applies the calculated motion to the object linked with the script's output, coordinated through the information provided by the connection panel. This streamlined process enables the simulation to execute the sinusoidal motion in a coherent and unified manner, ensuring that the base plate's movement is in sync with the predetermined parameters of the simulation environment. The following code gives the function that was defined in the script which calculates the sinusoidal base acceleration. In order to test both the forced motion and free rocking following the termination of the plate motion, the script has a if statement to terminate the plate motion at 10 s after the initiation of the simulation.

```
def pre_step(self):
A = 6*9.810
f = 3
t = self.getApplicationContext().getSimulationTime()
dt = self.getApplicationContext().getSimulationTimeStep()

if t <= 10:
v = self.getInput("velocity").getValue()
v = v + dt * A * sin(f * 2 * pi * t)
self.getOutput("velocity").setValue(v)
```

In the above code, where A is the amplitude of the sinusoidal acceleration, f is the frequency, t is the current simulation time, and dt is the simulation time step. The script checks if the simulation time t is less than or equal to 10 seconds to apply this motion. A ground motion definition for a

6Asin(3t) ground acceleration case was presented, and here the amplitude of the sinusoidal wave was given by the multiples of the activation acceleration of the block. In this case the block has an aspect ratio of 1, therefore the subsequent activation acceleration would be 9.810 m/s^2. After defining the ground acceleration, the next step would be to assign the motion to the selected object and in this case the base plate. The transformational data of the block should also be extracted and allowed to be export once the simulation was carried out, which could be accomplished by the following code:

```
def post_step(self):
tm = self.getInput('tm').value
ori = tm.getRotation()
zaxis = VxVector3(0,0,1)
dp = ori[2].dot(zaxis)
if ori[2].y > 0 :
ang = -acos(dp)
else:
ang = acos(dp)

self.getOutput('theta').value = ang
```

After the simulation step is completed, the ‘post_step’ function was called. It calculated the angle ‘ang’ based on the current orientation ‘ori’ of an object and the dot product ‘dp’ with the z-axis ‘zaxis’. This angle represents the rotation of the object around the z-axis and is set as an output of the script. This time, the function defined above could extract the transformational data of the block through the variable ‘tm’.

In the latter stages of our forced rocking motion analysis, the comparative studies with laboratory data required an adjustment in the definition of sinusoidal base motion. Previously dictated by the activation acceleration, the amplitude was now characterized by the peak linear displacement of the base plate. This imposed a modification in the computational script. The revised script now calculates the sinusoidal motion using the peak displacement values, which offers a more direct correlation with the experimental setup and results. In this case, the code would be modified as follow:

```
def pre_step(self):
A_displacement = 5*0.001 # Amplitude of displacement
f = 3.3
t = self.getApplicationContext().getSimulationTime()
dt = self.getApplicationContext().getSimulationTimeStep()

displacement = A_displacement * sin(f * 2 * pi * t)
velocity = A_displacement * f * 2 * pi * cos(f * 2 * pi * t)
prev_displacement = self.getInput("displacement").getValue()
delta_displacement = displacement - prev_displacement

velocity_change = delta_displacement / dt
self.getOutput("velocity").setValue(velocity_change)
self.getOutput("displacement").setValue(displacement)
```

The above code now takes in the displacement in mm as the amplitude of the ground motion. As shown in the second and the third line of the script, it sets the amplitude of the displacement (A_displacement) to 5 millimeters (as 0.001 is a millimeter in meters) and the frequency (f) to 3.3 Hz.

4.4 Analysis of the simulation results for single block forced motion

From simulation results, both physics engines exhibit good agreement with the frequency of the rocking motion predicted by Housner's analytical model, as evidenced by the consistent periodicity of the waveforms across all simulations. In figure 4.4, the plots for 0.5m-0.5m block and 0.5m-1.25m block was presented as an example of this finding.

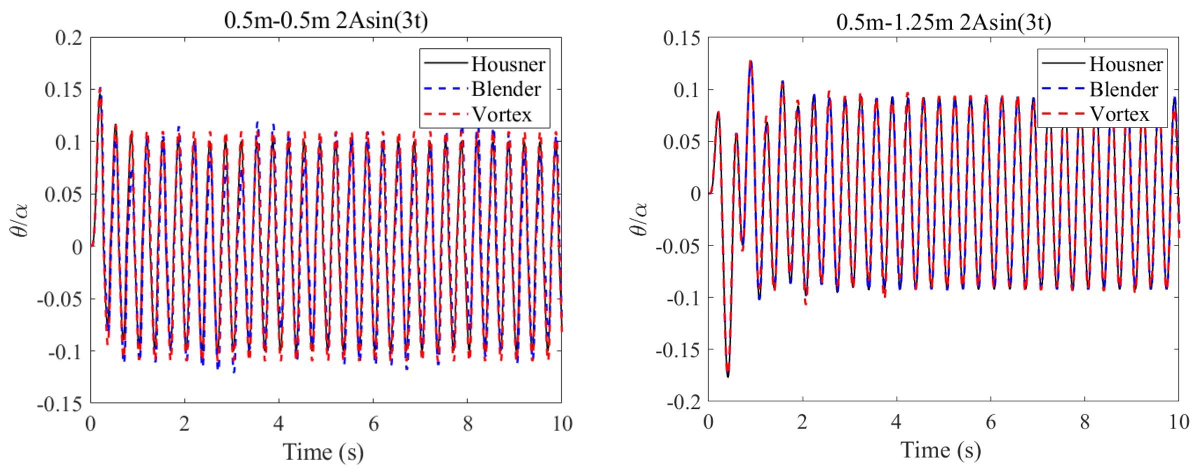


Figure 4.4 (a) simulation results for 0.5m-0.5m block under 2Asin(3t) ground acceleration case (b) simulation results for 0.5m-1.25m block under 2Asin(3t) ground acceleration case

There are slight discrepancies in the amplitude of the rocking motion between the simulations and the analytical model. This could be due to differences in how the physics engines handle the contact and friction between the block and the base, as well as differences in numerical integration methods. When comparing Blender and Vortex, Blender’s simulations are closer to Housner's model in the 0.5m-0.5m case, while Vortex shows a more pronounced amplitude in both cases. These differences highlight the sensitivity of the simulation results to the specific physics engine used and the importance of selecting the appropriate simulation parameters. The alignment of the simulated results with the analytical model across two different aspect ratios indicates that both physics engines can reasonably replicate the rocking behavior of blocks under sinusoidal base motion, although care must be taken in interpreting the results due to the noted differences. The impact of sub-steps and simulation frame rate on the precision and accuracy of physics engines in capturing the motion of objects is a significant factor in the fidelity of simulations. These parameters are critical as they determine the granularity of the simulation's temporal resolution. A higher number of sub-steps or a higher frame rate can result in more detailed and accurate calculations of the physics involved, particularly in situations where the dynamics of the system are complex and fast-changing. Conversely, fewer sub-steps may lead to a loss of detail and potential inaccuracies in the simulation, especially in cases where rapid changes in motion occur within short time spans. The subsequent sections of this study will explore this matter further, providing concrete examples that elucidate the influence of these settings on simulation outcomes. The results will guide users in configuring simulations to achieve a balance between computational efficiency and the level of detail necessary for their specific application requirements.

4.4.1 Effect of frame rate per second on the simulation result

The frame rate per second (fps) in simulations executed by physics engines plays a critical role in determining the quality of the results. Higher frame rates enable the engines to capture the intricacies of an object’s motion more accurately, as they provide a finer temporal resolution. With lower frame rates, the time interval between successive frames widens, leading to fewer computational steps for integrating the dynamic state of the object. This sparser data can result in a cumulative increase in error over the simulation period. As physics engines operate on a frame-by-frame basis, increasing the frame rate effectively increases the level of detail, akin to enhancing the discretization in the Discrete Element Method (DEM). This improvement in precision, however, often comes at the cost of extended simulation times, thus presenting a trade-off between accuracy and computational efficiency.

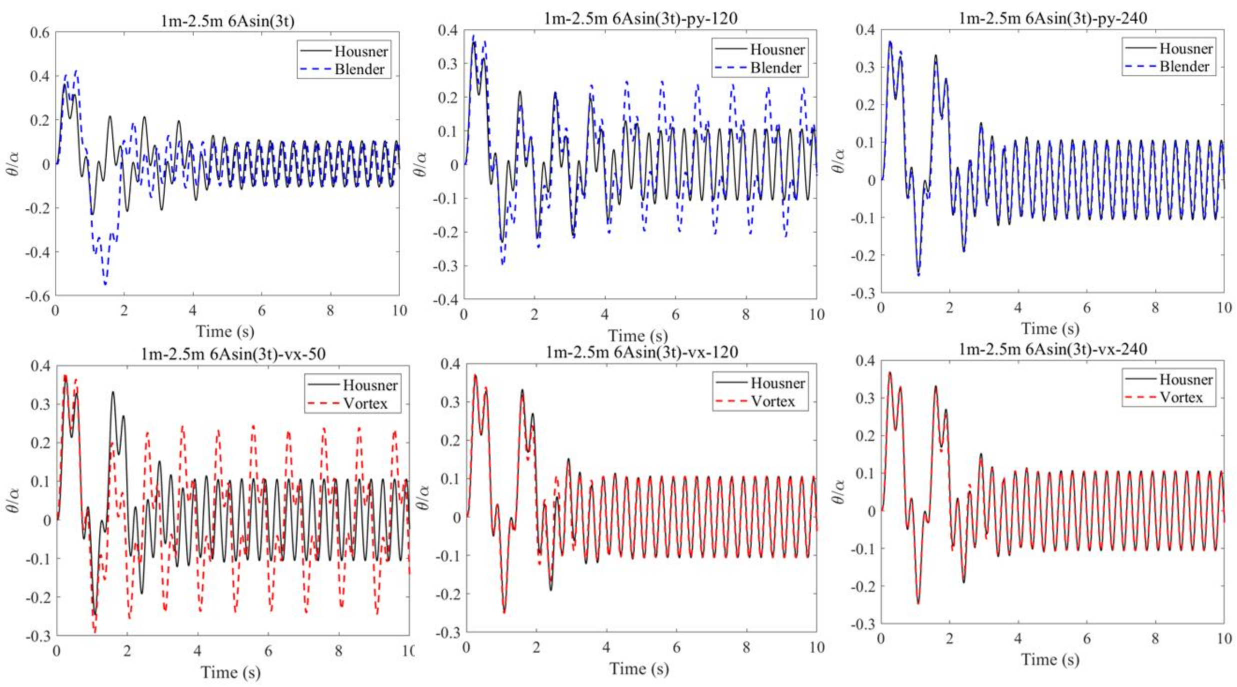


Figure 4.5 Simulation results for 1m-1.25m block under 6Asin(3t) ground acceleration case with 50, 120, and 240 fps in Blender and Vortex distinctly

Figure 4.5 illustrates the angular displacement over time for a block subjected to sinusoidal base excitation, comparing results from Housner's analytical model with simulations conducted in Blender and Vortex physics engines. The simulations are executed with varying frame rates, denoted by "vx" for Vortex and "py" for Blender (utilizing python script to import the ground motion data), followed by a numerical value indicating the frame rate used. The Blender simulation closely follows Housner's analytical model across different frame rates (50, 120, and 240 fps). This suggests that Blender's engine maintains a consistent level of accuracy across the range of frame rates, with the higher frame rate (240 fps) showing a more precise adherence to the analytical model, especially in the amplitude of oscillations. The Vortex simulations show a notable divergence from Housner's model, particularly at lower frame rates (50 fps). As the frame rate increases to 120 and 240 fps, the alignment with Housner's predictions improves, indicating a reduced error in the angular displacement with higher frame rates. (Epic Games 2023)

4.4.2 Effect of the number of sub-steps

Sub-step is a concept in physics engines, representing the subdivision of a single frame. This parameter was introduced to enhance the robustness of simulation quality by the physics engine, without altering the actual frame rate. For instance, consider a simulation running at 50 frames per second (fps). By introducing a sub-step of 2, the integration solver is instructed to perform more steps in calculating the position and state of an object. Consequently, the effective frame number that the solver computes doubles, becoming 100 (50 fps multiplied by 2 sub-steps). (Blender Documentation Team 2023) To investigate the impact of sub-steps on forced rocking simulations, various trials were conducted. These simulations encompassed three different cases: 1m-2.5m 2Asin(1.5t), 1m-1m 2Asin(1.5t), and 1m-1m 2Asin(3t). The trials were executed with 2, 20, and 200 sub-steps, respectively. The results were then compared against the analytical predictions derived from Housner's equation, as illustrated in Figure 4.6.

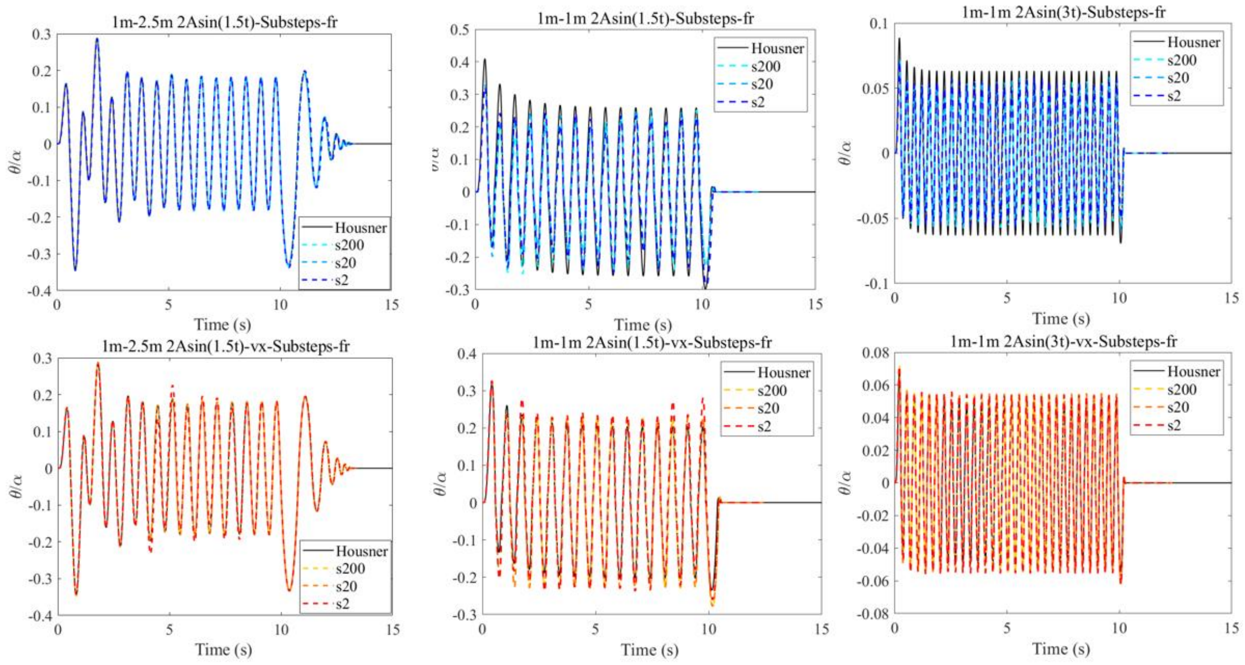


Figure 4.6 Simulation trails for rigid macro blocks in Blender (in blue) and Vortex (in orange) under

sinusoidal base acceleration ran with 2 sub-steps, 20 sub-steps and 200 sub-steps

From the top row, it's evident that the Blender simulations with a low sub-step count (s2) deviate significantly from the analytical model, particularly in terms of energy dissipation over time. When the sub-step count is increased to 20 (s20) and further to 200 (s200), the simulation results demonstrate a progressive convergence towards Housner's prediction, indicating a more accurate representation of the physical dynamics at higher sub-step levels. Turning to the Vortex simulations in the bottom row, there is a clear distinction in how each sub-step level corresponds with the analytical model. Similar to the Blender results, the s2 simulations reveal a marked discrepancy, highlighting the limitations of low sub-step counts in capturing the full complexity of the system's behavior. As the sub-step count was increased, the reliability of the simulation improves evidently, with the s200 results closely mirroring the theoretical curve. The comparison between Blender and Vortex suggests that both engines benefit from higher sub-step counts to enhance the precision of the simulation. However, the degree of deviation at lower sub-step levels indicates that the internal algorithms and numerical solvers employed by each engine process the sub-step parameter differently. This is particularly noticeable in the forced rocking simulations with higher frequencies such as in the case 1m-1m 2Asin(3t), where the temporal resolution of the simulation becomes critical.

4.4 Comparative study of simulated motion in physics engines and laboratory experimental results

In the next section of the research, a comparative study was carried out for forced rocking behavior of URM macro-block under sinusoidal ground motion using physics engines and laboratory experimentations. From the experiments carried out by Pena et al (Peña, Lourenço, and Campos-Costa 2008a) Five granite specimens were prepared, and the edges of the specimens were trimmed with a 5mm width at 45 degrees. The trimmed edges were intended to minimize the effect of specimen edges been broken during the rocking motion, since the specimen were made of rigid materials and vulnerable when subjected to sudden collisions. The detailed information of the granite specimens can be found in table 3 below.

Table 3 Information of the granite specimens tested in the laboratory experiments.

Specimen	Width (m)	Height (m)	Thickness（m）	Mass (kg)
1	0.25	1.000	0.754	503
2	0.17	1.000	0.502	228
3	0.12	1.000	0.375	120
Stacked top	0.15	0.600	0.400	97
Stacked bottom	0.20	0.600	0.550	178
base	1.0	0.250	0.750	500

For specimen 1, free rocking testing was done, and the author compared the experimental results with one discrete element simulation and one numerical tool. The parameters were adjusted to match the experimental results. Next, sinusoidal base motion was imposed onto the base plate and specimen 2 was subjected to sinusoidal ground excitation. Specimens stacked top and stacked bottom were stacked together and later tested under sinusoidal ground motion as stacked block case. The experimental results for specimen 2 under sinusoidal ground excitation were obtained and shown below in table 4.

Table 4 Typical rocking motion space and maximum rocking angle (degrees) for harmonic motion (specimen 2) (Peña, Lourenço, and Campos-Costa 2008b)

Constant Sine	Amplitude (mm)					
(Hz)	3	4	5	6	8	10
0.5	N-R					
1.0						
1.5						
2.0		N-R	N-R	N-R	3.04	3.93
2.5	N-R	N-R	2.27	2.33	2.63	
3.0	1.36	1.49	1.85	1.87	2.15	
3.3	1.35	1.49	1.57	1.42	2.03	
5.0		0.95	1.14	1.23		

N-R: no rocking

The table presents a spectrum of rocking motion scenarios for the granite specimen 2, spanning from a ground motion case with an amplitude of 3 mm and a frequency of 5.0 Hz to one with a 10 mm amplitude and a 2.0 Hz frequency. When the block is subjected to higher frequencies paired with larger amplitudes, there's a pronounced tendency for the block to tilt and potentially topple. Conversely, when exposed to lower frequencies coupled with smaller amplitudes, the likelihood of the block initiating a rocking motion is substantially reduced, suggesting that such conditions may be insufficient to trigger dynamic instability.

4.4.1 Modelling establishment in physics engines

When it came to replicating trimmed blocks—specifically, specimen 2 from the study conducted by Pena et al. (Peña, Lourenço, and Campos-Costa 2008a) The process required a different approach. In Blender, the utilization of a Python script was pivotal in crafting a model block with trimmed edges. The script employed the Bevel modifier to achieve the desired geometry. Initially, a cube object was generated within the scene, with its dimensions and scale being reset to default unit values. Subsequently, the Bevel modifier was applied to the object. This modifier was configured to execute a precise 5mm cut at each edge, set at a 45-degree angle, effectively trimming the block's corners. The following code snippet for this operation would resemble the following structure. This process ensures that the block's geometry accurately reflects the specified design parameters, a necessary step before progressing to the simulation phase of the research:

```
bpy.ops.object.mode_set(mode='EDIT')

  bpy.ops.mesh.select_all(action='SELECT')

  bpy.ops.mesh.bevel(offset_type='OFFSET', offset=0.005 / math.sqrt(2), segments=1, profile=0.5)

  bpy.ops.object.mode_set(mode='OBJECT')
```

Following the formation of the trimmed block, the testing setup for the perfect rectangular block was replicated. The trimmed block was positioned on a base plate, which was subsequently subjected to a predetermined sinusoidal base motion. This consistent approach allowed for a direct comparison between the behavior of the perfect rectangular block and the trimmed block under identical testing conditions. The detailed arrangement of the testing setup is crucial for ensuring that the sinusoidal motion applied to the base plate accurately translates to the block, thereby facilitating a reliable assessment of the block's response to dynamic loading scenario.

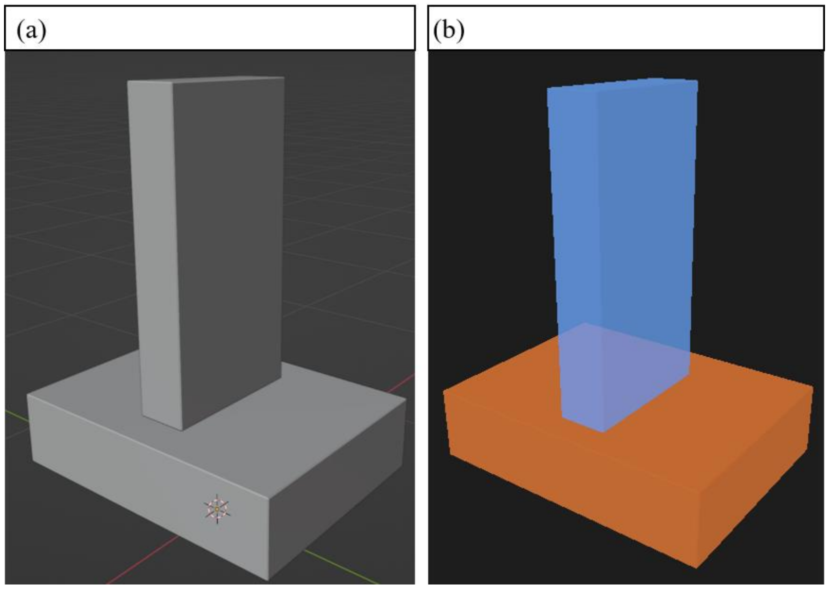


Figure 4.7 (a)Trimmed model for specimen 2 in Blender (b) Trimmed model for specimen 2 in Vortex

Once crafted, the model was exported as an FBX file and subsequently imported into Vortex Studio to undergo additional simulation procedures. Within Vortex, the convex mesh was chosen for collision detection, a decision aimed at accommodating the modified edges of the block. This selection is indicative of the flexibility within the simulation tools to adapt to geometric alterations and ensures that the nuances of the block's shape are accurately represented in the ensuing collision and interaction analyses.

4.4.2 Specimen 1 (the fine tuning of the numerical tools)

The laboratory experimental result for a 0.2 alpha initial angle of rotation case for specimen 1 was conducted and the angle of rotation was plotted against the DEM and CCRR simulation outcome in the original study. The researchers have performed fine tuning on the DEM and CCRR numerical tools to match the experimental results. In their study, the friction angle was assumed to be 30 degrees for the four specimens, and then the damping was fitted according to the experimental value in order to obtain the best correlation. (Peña et al. 2007) The following figure 4.8 shows the plots from the original study and with additional results from the Housner's prediction, Blender, and Vortex.

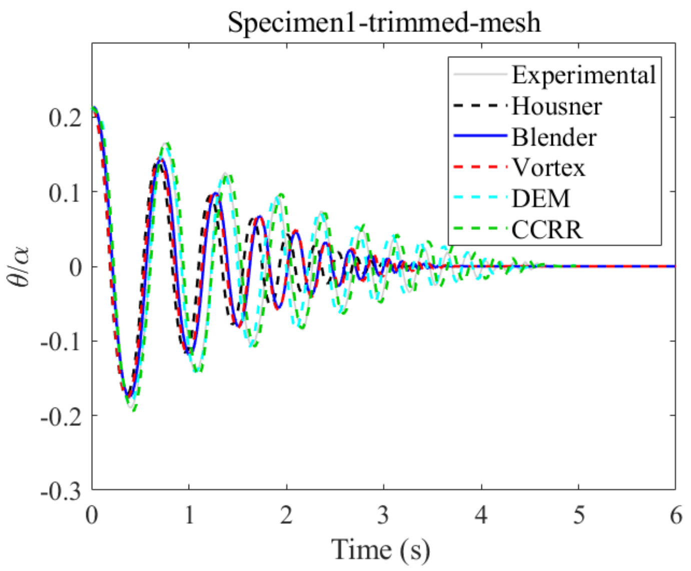


Figure 4.8 Specimen 1 rocking motion tested using Blender, Vortex, Housner, DEM, CCRR and plotted against the experimental result

From the figure, it could be observed that during a free rocking event for a slender member, the experimental results deviate a significant amount from the analytical prediction by Housner's equation where no damping and perfect rigid body response was assumed. Also, although trimmed corners were implemented to mitigate the effects of corner breaking and damaging during the testing, it was still difficult to eliminate during the actual testing process. However, with the forced rocking testing the results from Blender and Vortex are rather perfectly capturing the motion of the experimental result. In the following figure 4.9, the plot of the angle of rotation results from the Blender and Vortex using the trimmed model matched the experimental curve very well.

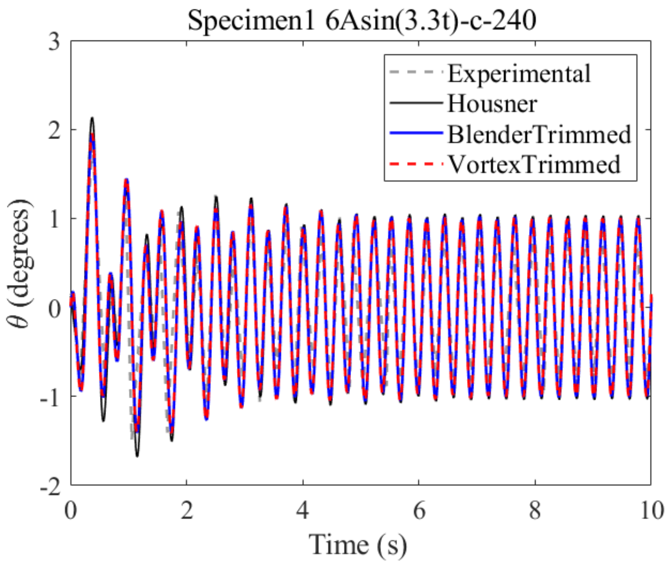


Figure 4.9 Specimen 1 under 6Asin(3.3t) trimmed model ran with 240 fps

It is important to highlight that the choice of contact geometry can significantly impact simulation results. For the Vortex simulations referenced in this case, a box geometry was employed for the contact model. This particular selection is reflected in the figure, where the simulation results demonstrate a relatively stable amplitude and consistent overall behavior. Subsequent sections will dig deeper into the implications of contact geometry on simulation stability, illustrating that while it can influence the outcomes, the choice of geometry should ultimately be selected to fit the specific requirements and goals of the desired application. Such a strategic selection is crucial for achieving a balance between accurate representation and computational efficiency in the context of simulation scenarios.

4.4.3 Specimen 2 (sinusoidal ground motion)

In the original study, the researchers executed forced rocking motion tests on specimen 2 utilizing a standard shake table. The outcomes of these tests were thoroughly recorded in Table 4, which captures the maximum rocking angle observed for each trial. Building upon this, the current study extends the investigation by conducting forced rocking simulations on trimmed versions of specimen 2, utilizing both Blender and Vortex physics engines for the simulation process. This approach facilitated a detailed comparison across four distinct ground motion scenarios, each with a 6 mm amplitude. The results of these simulations are comprehensively presented in Figure 4.10.

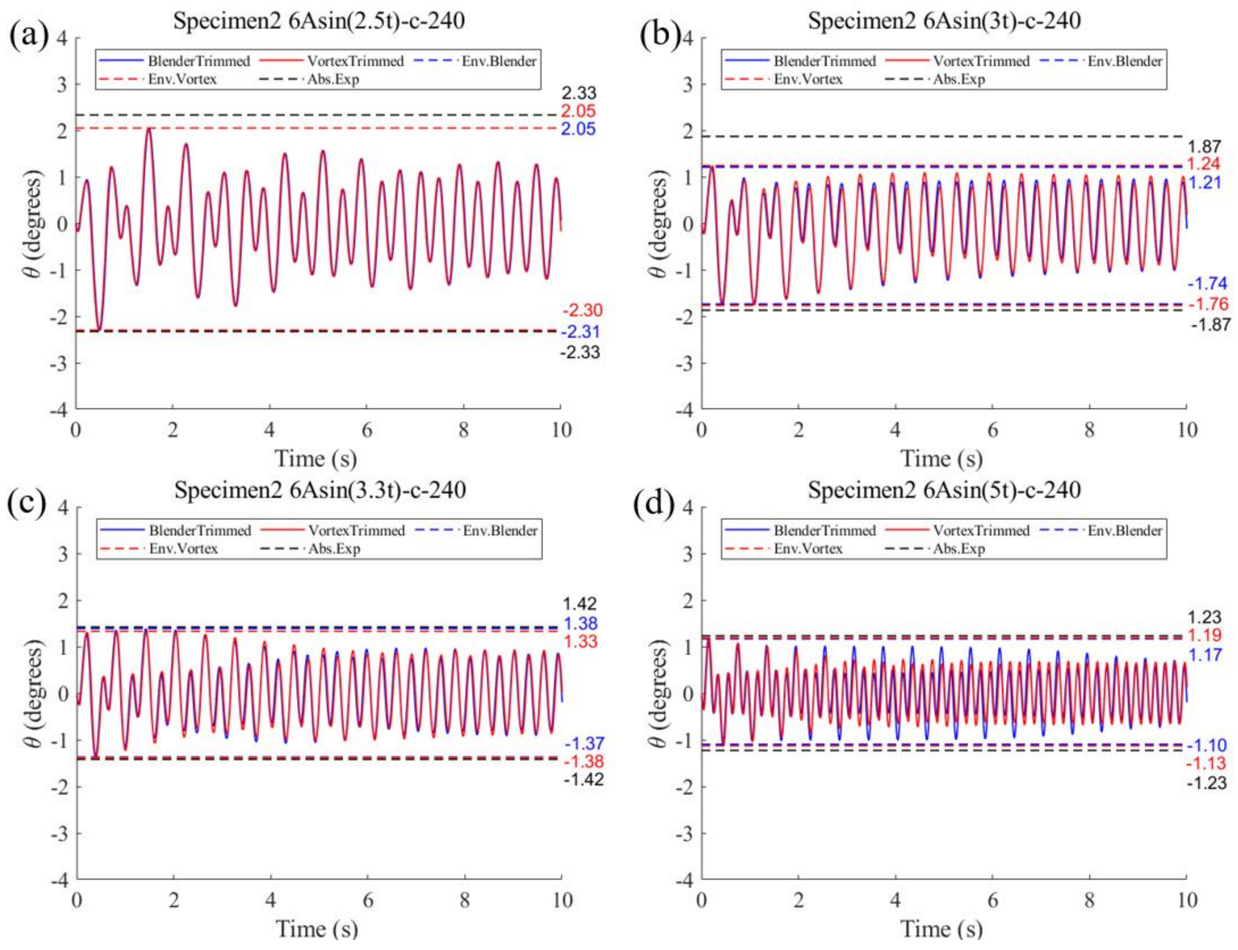


Figure 4.10 (a) specimen 2 under 6Asin(2.5t) (b) specimen 2 under 6Asin(3t) (c) specimen 2 under

6Asin(3.3t) (d) specimen 2 under 6Asin(5t)

The black dashed line portrays the maximum angle of rotation as determined by the actual experimental data. Meanwhile, the blue dashed line corresponds to the extremities of positive and negative angles derived from the Blender simulations, and the red dashed line similarly represents the range of angles from the Vortex Studio simulations. In Figure 4.10, it was evident that the outcomes from Vortex Studio more closely align with the experimental results in almost all the scenarios presented. This great resemblance may stem from Vortex Studio's more sophisticated contact modeling approach, coupled with its enhanced interpretation of collision geometries. (CM Labs Simulations 2016) Such advanced features likely contribute to Vortex Studio's ability to replicate the physical experiment's discrepancies more accurately, thus offering a more reliable virtual representation of the specimen's dynamic behavior under forced rocking conditions.

4.4.4 Stacked blocks under sinusoidal ground motion

A comparative study was undertaken for the stacked blocks using both Blender and Vortex physics engines. Given that the experimental data did not provide a clear and definitive result for comparison, an additional analysis was performed using 3DEC to serve as an alternative means of validation for the results obtained from the Blender simulations. This step was crucial, as the interaction between the stacked blocks introduced added complexity to the system's overall behavior. Due to this complexity, the differences in the contact modeling approaches employed by each physics engine became more obvious. Therefore, these variations allowed for a more detailed observation and analysis of how each engine handles the involved interaction of forces and contacts within the stacked block configuration.

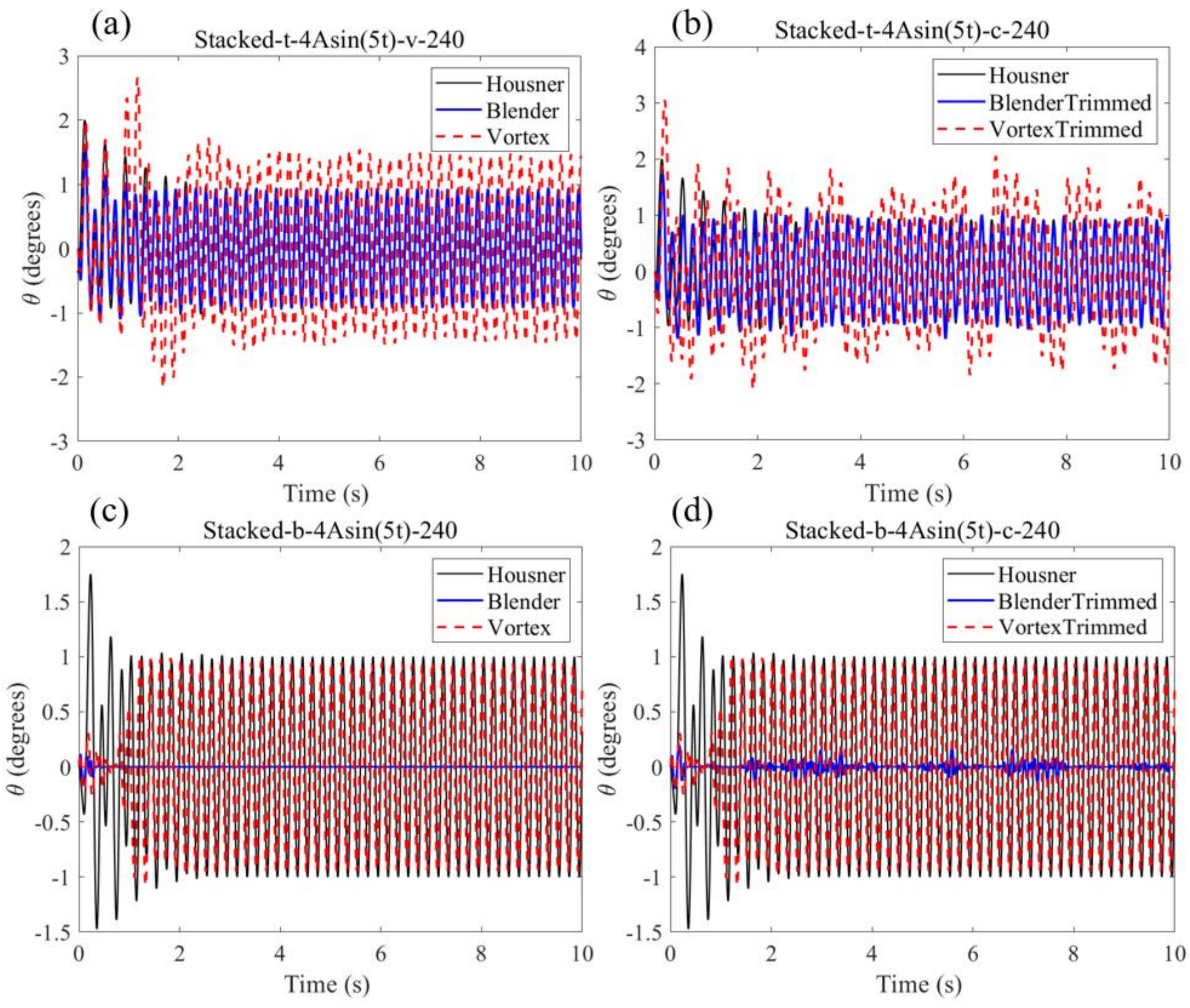


Figure 4.11 (a) untrimmed stacked top block under 4Asin(5t) (b) trimmed stacked top block under 4Asin(5t) (c) untrimmed stacked bottom block under 4Asin(5t) (d) trimmed stacked bottom block under 4Asin(5t)

As depicted in Figure 4.11, the untrimmed models exhibited rocking behavior in Blender that was not accurately captured, which can be attributed to limitations within the contact model. However, once the edges were trimmed, the simulation of the bottom block's rocking behavior in Blender showed a marked improvement. This suggests that the geometry of the collision shape significantly affects the simulation's reliability. Furthermore, the friction model used in Blender presents another area for potential enhancement. It currently contributes to inaccuracies not only in simulations of squat blocks but also in those involving forced rocking motions at high frequencies.

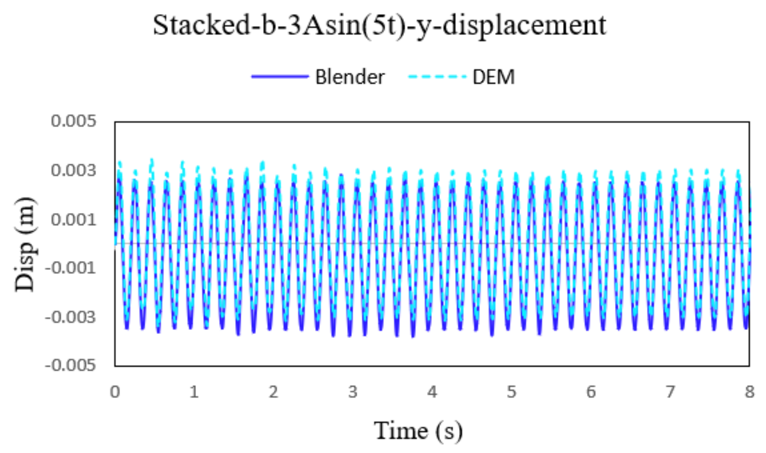


Figure 4.12 Comparison of y-axis displacement in Blender and 3DEC for stacked bottom block under 3Asin(5t) ground motion case

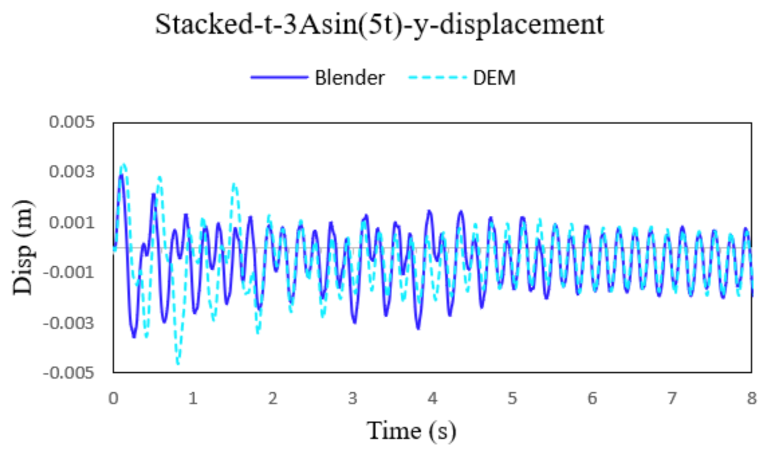


Figure 4.13 Comparison of y-axis displacement in Blender and 3DEC for stacked top block under 3Asin(5t) ground motion case

Figures 4.12 and 4.13 present a comparative analysis of the y-displacement over time between simulations conducted in Blender and those performed with 3DEC, which utilizes DEM. Given that DEM simulations are renowned for their proven accuracy and sophisticated contact modeling capabilities, it is a logical deduction to assert that, as demonstrated in the figures, Blender achieves a high level of precision in simulating the overall response of the rocking behavior in stacked blocks. This comparison underscores Blender's capability to closely replicate the dynamic response as modeled by the more established 3DEC simulations, thereby validating Blender's utility in accurately capturing complex interactions within a stacked block system under dynamic conditions. The experimental results were not consistent with the simulation results from both DEM and physics engines, therefore in this case, only DEM and physics engine results were presented. In fact, in the original study, the author stated that most of the experimental results were not repeatable. This equipment choice may have introduced both systematic and gross errors during the experimentation process.

4.5 Next steps - application of physics engines to the building-scale macro-block collapse analysis of URM façades

Building on the results discussed earlier, the forthcoming stage of the research will extend the use of physics engines to the analysis of large-scale masonry structures. The next step includes conducting a collapse and debris distribution analysis on a building-scale URM façade, which is part of a real-world case study involving a five-story heritage building located in Belfast's historic center, UK. A discrete block model representing this structure has been developed and imported into both Blender and Vortex for simulation purposes. These simulations will be performed as pushover analyses, with the outcomes being benchmarked against the DEM analysis results from the preceding study by Malomo and DeJong (2022) An illustration of the model setup within Blender is provided in Figure 4.14.

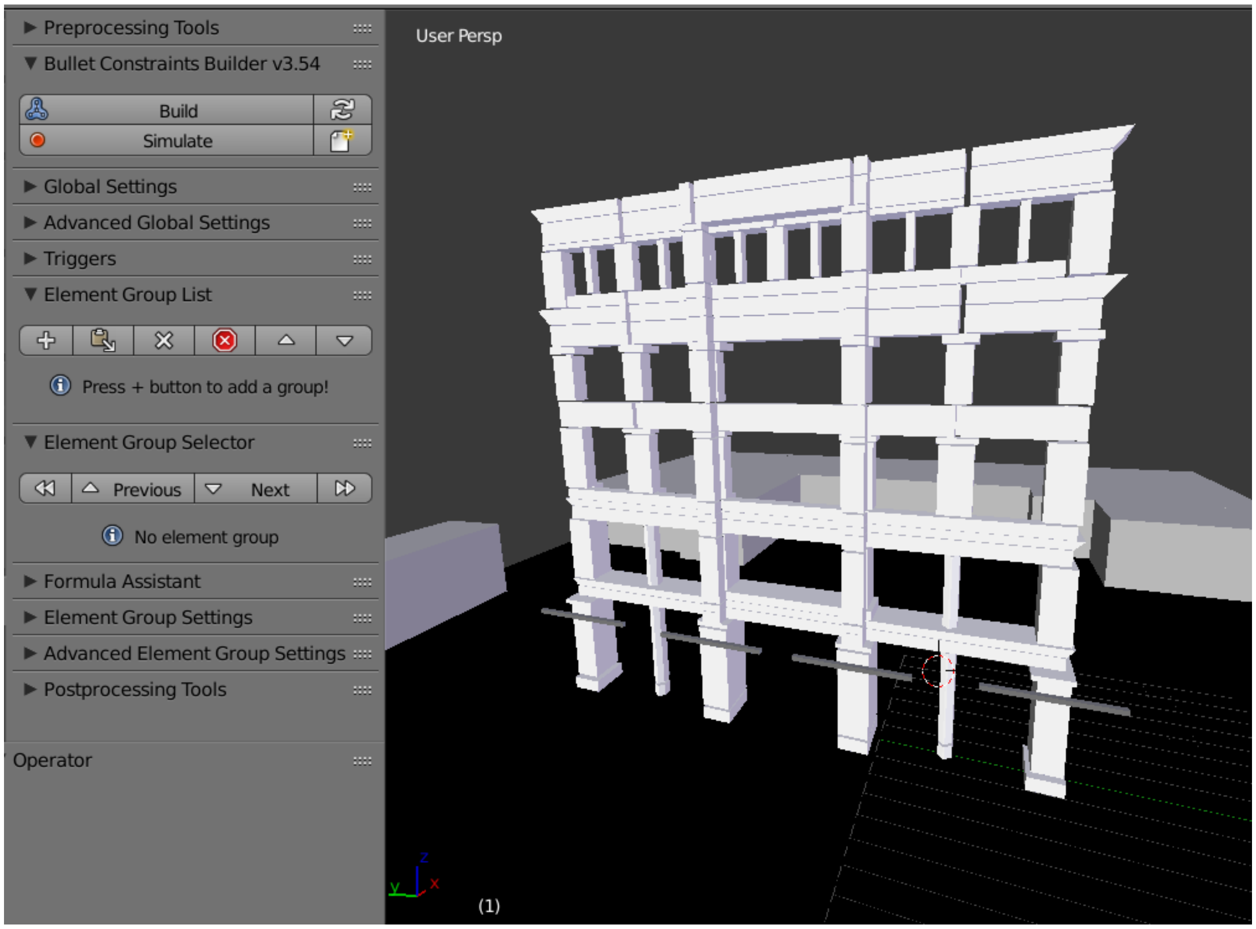


Figure 4.14 The model formulation of the URM façade in Blender

The model formulation in Blender utilized an add-on called Bullet Constraint Builder (BCB), which enabled efficient definition of constraints between objects. Users could categorize objects based on element groups, allowing simulations to be run with a single click. In this façade simulation, four push bars provided an initial acceleration from 0 mm/s to 1 mm/s over a 1-second interval, followed by a constant velocity of 1 mm/s until structural failure occurred. The resultant debris distribution from the Blender simulation was then compared with outcomes from discrete element simulations conducted using 3DEC, as illustrated in Figure 4.15.

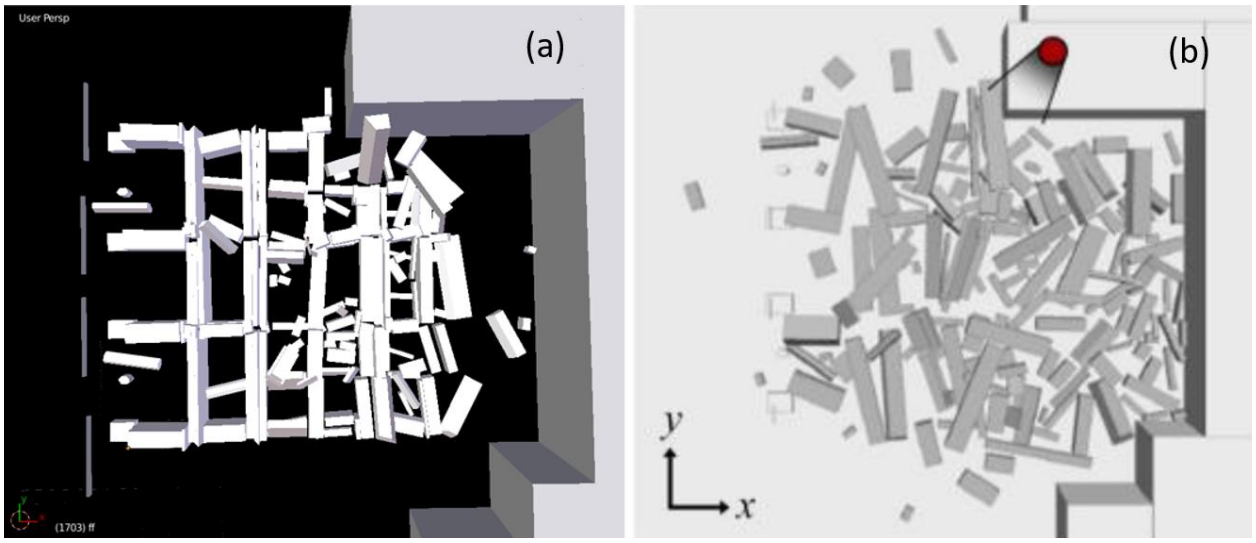


Figure 4.15 URM façade debris distribution pattern in a) Blender and b) 3DEC (Malomo and DeJong 2022)

For the ground floor push-over case, the BCB simulation in Blender showed similarities to the 3DEC results. However, a limitation was noted in BCB's inability to precisely tune normal and shear stiffness to match those in 3DEC; only preset values were available. In this instance, the 'masonry walls' preset was employed to simulate the structural response more accurately.

In Vortex studio, the formulation of the model was done according to the same procedure as the rocking block case, where each block was assigned a part and mechanism collision shape individually. However, this case due to the fact that Vortex studio uses a direct solver, the complex façade would generate a too large number of contacts and therefore the simulation would be unrealistic unless a iterative solver was implemented. The formulation of the model is shown in figure 4.16.

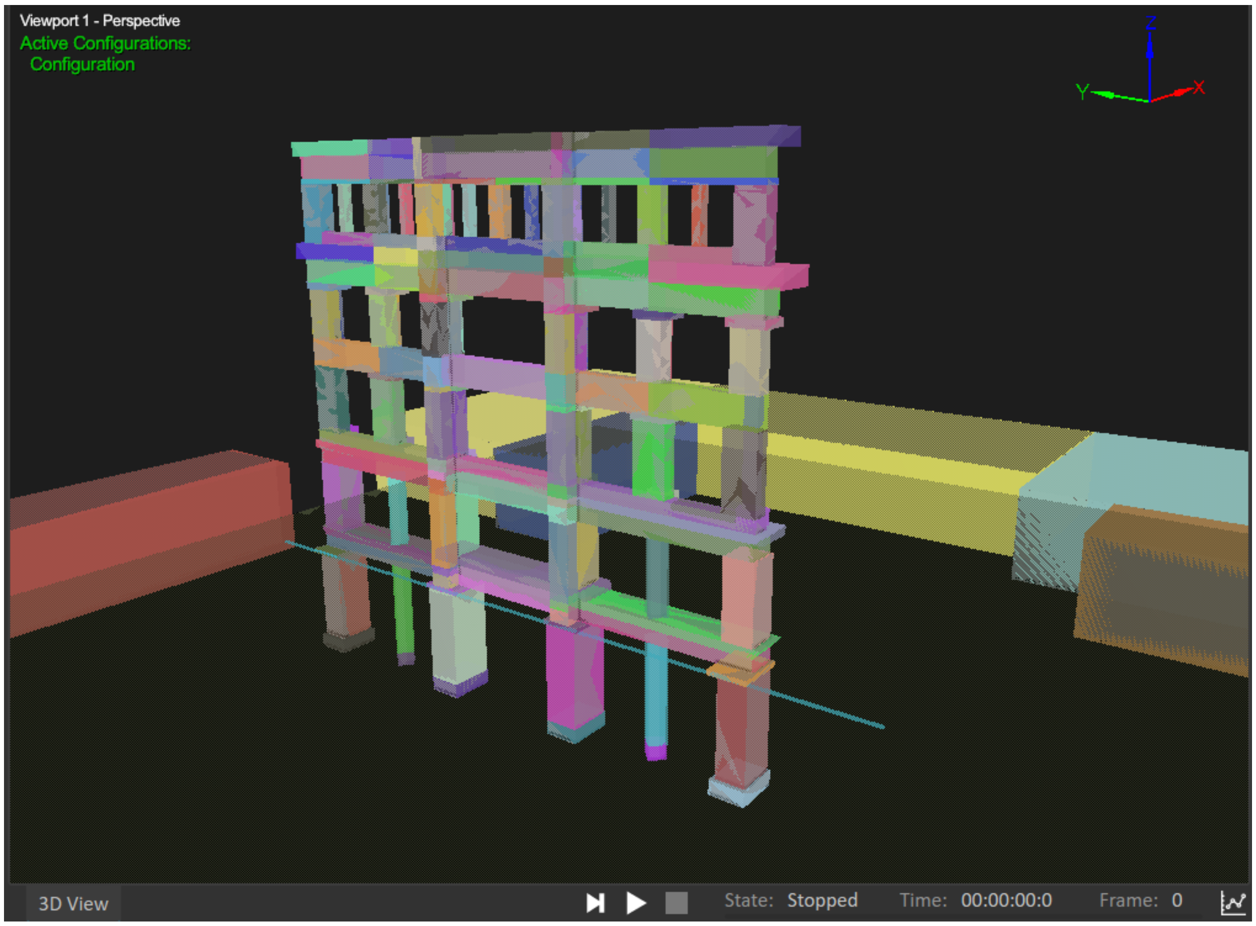


Figure 4.16 The model formulation of the URM façade in Vortex

This ongoing work has proposed a promising avenue for utilizing physics engines, highlighting their potential efficiency advantages. When compared to traditional DEM or FEM alternatives, the use of physics engines for macro-analysis of large-scale structures could offer significant improvements in terms of both time and resource expenditure.

Chapter 5 – Conclusion

This study rigorously evaluated the capabilities of two renowned physics engines, Blender and Vortex, which are acclaimed for producing visually plausible simulations of rigid body dynamics within the video gaming and digital animation sectors. The aim was to determine the precision with which these engines predicted the free rocking motion of macro-blocks, representative of components found in unreinforced masonry (URM) structures. A parametric analysis was performed, encompassing a diverse range of aspect ratios and initial rotation angles. The results from these simulations were benchmarked against the analytical solutions provided by Housner's established equations for rocking motion. The comparison of the numerical results with the analytical benchmarks indicated that discrepancies from the theoretical values for simple rocking motion were minimal for both engines; the simulated motions closely followed the predicted theoretical paths. Notably, the results obtained with Blender showed significant improvements over those documented in existing research. This advancement was realized through a detailed examination of the various settings within Blender, which was discussed extensively in the study. Specifically, the Blender models were highly sensitive to the collision detection parameters and the contact models that were selected.

For both Blender and Vortex, the size of the time step appeared as a crucial factor affecting the accuracy of the simulations. Larger time steps were associated with a loss of precision in the results. Accordingly, it was recommended to use a sufficiently small time-step and a higher number of solver iterations when running simulations. For the purposes of this study, a sub-step of 10,000 steps per second and 1,000 solver iterations, the maximum permitted by the Blender interface, were utilized. For Vortex, the simulations were conducted with 10,000 steps per second and a maximum of 50 pivoting steps with the direct solver.

The study revealed that physics engines could accurately simulate the simple rocking motion of macro-blocks, with a discrepancy of less than 2% when compared to analytical solutions for aspect ratios of rocking macro-blocks ranging from 0.5 to 2.5. The simulation accuracy was critically dependent on the precise selection and adjustment of collision detection and margin parameters. Additionally, the findings underscored the remarkable efficiency of physics engines; for instance, a 10-second free rocking motion simulation of a macro-block was completed in just 4 seconds on a standard office laptop.

Given the limitations of physics engines, the constraints between the elements of structures often necessitate the use of additional add-ons in most physics engines. In many structures, it is essential to define constraints between structural members. In masonry structures, as explored in this study, modeling the shear stiffness provided by mortar in the façade is required. In such instances, additional add-ons to the engines can be utilized, such as the Bullet Constraint Builder for Blender.

The investigation into the forced rocking simulation process uncovered those factors such as the friction model, frame rate, and the detail of sub-steps were instrumental in obtaining accurate results from the physics engines. During the tests involving stacked block configurations, the engines demonstrated their potential for simulating more complex structural interactions, indicating their suitability for broader applications in structural analysis. The next phase of research involved applying these engines to the simulation of a large-scale structural model—a building-scale URM façade from a historic five-story building in Belfast—where a pushover analysis was conducted. These simulations were then compared with the results from previous DEM studies, validating the physics engines' utility in more complex and extensive modeling endeavors.

Reference

Augusti, Giuliano, and Anna Sinopoli. 1992. “Modelling the Dynamics of Large Block Structures.” *Meccanica* 27: 195–211. https://doi.org/10.1007/BF00430045.

Blender Documentation Team. 2023. “Blender 3.6 Reference Manual.” Blender Documentation Team. 2023. https://docs.blender.org/manual/en/latest/index.html.

Bui, T. T., A. Limam, V. Sarhosis, and M. Hjiaj. 2017. “Discrete Element Modelling of the In-Plane and out-of-Plane Behaviour of Dry-Joint Masonry Wall Constructions.” *Engineering Structures* 136: 277–94. https://doi.org/10.1016/j.engstruct.2017.01.020.

Burman, B. C., P. A. Cundall, and O. D.L. Strack. 1980. “A Discrete Numerical Model for Granular Assemblies.” *Geotechnique*. https://doi.org/10.1680/geot.1980.30.3.331.

Cappelli, Enrico, Angelo Di Egidio, and Fabrizio Vestroni. 2020. “Analytical and Experimental Investigation of the Behavior of a Rocking Masonry Tuff Wall.” *Journal of Engineering Mechanics* 146: 4020048. https://doi.org/10.1061/(ASCE)EM.1943-7889.0001775.

Casapulla, Claudia, Linda Giresini, and Paulo B. Lourenço. 2017. “Rocking and Kinematic Approaches for Rigid Block Analysis of Masonry Walls: State of the Art and Recent Developments.” *Buildings* 7 (3). https://doi.org/10.3390/buildings7030069.

Chen, Wei Bin, Wan Huan Zhou, and Zhen Yu Yin. 2023. “Recent Development on Macro–Micro Mechanism of Soil-Structure Interface Shearing Through DEM.” *Archives of Computational Methods in Engineering* 30 (3): 1843–62. https://doi.org/10.1007/s11831-022-09854-0.

CM Labs Simulations. 2016. “Theory Guide: Vortex Software’s Multibody Dynamics Engine,” 1–65.

Coumans, Erwin, and All Rights. 2009. “Bullet 2 . 74 Physics SDK Manual.” *Physics*.

Cundall, Peter A. 1971. “A Computer Model for Simulating Progressive, Large-Scale Movement in Blocky Rock System.” In *Proceedings of the International Symposium on Rock Mechanics*, 8:129–36.

DeJong, Matthew J., and Elias G. Dimitrakopoulos. 2014. “Dynamically Equivalent Rocking Structures.” *Earthquake Engineering and Structural Dynamics* 43 (10): 1543–63. https://doi.org/10.1002/eqe.2410.

Dimitri, R, L De Lorenzis, and G Zavarise. 2011. “Numerical Study on the Dynamic Behavior of Masonry Columns and Arches on Buttresses with the Discrete Element Method.” *Engineering Structures* 33 (12): 3172–88. https://doi.org/https://doi.org/10.1016/j.engstruct.2011.08.018.

Epic Games, Inc. 2023. “Unreal Engine 5.3 Documentation.” Unreal Engine 5.3 Documentation. 2023. https://docs.unrealengine.com/5.0/en-US/physics-sub-stepping-in-unreal-engine/#:~:text=Sub-stepping takes the total,delta time is set to.

Galvez, Francisco, Marta Giaretton, Shannon Abeling, Jason Ingham, and Dmytro Dizhur. 2018. *Discrete Element Modeling of a Two Storey Unreinforced Masonry Scaled Model*.

Galvez, Francisco, Stefano Segatta, Marta Giaretton, Kevin Walsh, Ivan Giongo, and Dmytro Dizhur. 2018. “FE and DE Modelling of Out-Of-Plane Two Way Bending Behaviour of Unreinforced Masonry Walls.” In .

Galvez, Francisco, Luigi Sorrentino, Dmytro Dizhur, and Jason Ingham. 2022. “Damping Considerations for Rocking Block Dynamics Using the Discrete Element Method.” *Earthquake Engineering & Structural Dynamics* 51. https://doi.org/10.1002/eqe.3598.

Housner, George W. 1963. “The Behavior of Inverted Pendulum Structures during Earthquakes.” *Bulletin of the Seismological Society of America* 53 (2): 403–17. https://doi.org/10.1785/bssa0530020403.

Ishiyama, Yuji. 1984. “Motions of Rigid Bodies and Criteria for Overturning by Earthquake Excitations.” *Bulletin of the New Zealand Society for Earthquake Engineering* 17: 24–37. https://doi.org/10.5459/bnzsee.17.1.24-37.

Kovacs, Michael Andrew, and Lydell Wiebe. 2019. “Controlled Rocking CLT Walls for Buildings in Regions of Moderate Seismicity: Design Procedure and Numerical Collapse Assessment.” *Journal of Earthquake Engineering* 23 (5): 750–70. https://doi.org/10.1080/13632469.2017.1326421.

Lachanas, Christos, and Dimitrios Vamvatsikos. 2021. “Rocking Incremental Dynamic Analysis.” *Earthquake Engineering & Structural Dynamics* 51. https://doi.org/10.1002/eqe.3586.

Lemos, José V. 2007. “Discrete Element Modeling of Masonry Structures.” *International Journal of Architectural Heritage* 1 (2): 190–213. https://doi.org/10.1080/15583050601176868.

Lindemann, Patrick. 2009. “The Gilbert-Johnson-Keerthi Distance Algorithm.” *Algorithms in Media Informatics*. http://www.medien.ifi.lmu.de/lehre/ss10/ps/Ausarbeitung_Beispiel.pdf.

Lourenço, Paulo B. 2002. “Computations on Historic Masonry Structures.” *Progress in Structural Engineering and Materials* 4 (3): 301–19. https://doi.org/10.1002/pse.120.

Ma, Q. T., S. Parshottam, and M. Montalla. 2018. *Modelling Rocking Behaviour Using Physics Engine Simulation. 11th National Conference on Earthquake Engineering 2018, NCEE 2018: Integrating Science, Engineering, and Policy*. Vol. 2.

Malomo, Daniele, and Matthew J. DeJong. 2022. “Post-Fire Collapse Assessment of the Bank Buildings (Belfast, UK) Masonry Façade via Discrete Element Macro-Analysis.” *Structures* 35: 1002–9. https://doi.org/10.1016/j.istruc.2021.12.005.

Malomo, Daniele, Anjali Mehrotra, and Matthew J. DeJong. 2021. “Distinct Element Modeling of the Dynamic Response of a Rocking Podium Tested on a Shake Table.” *Earthquake Engineering and Structural Dynamics* 50 (5): 1469–75. https://doi.org/10.1002/eqe.3404.

Mehrotra, Anjali, and Matthew DeJong. 2017. “The Performance of Slender Monuments During the 2015 Gorkha, Nepal, Earthquake.” *Earthquake Spectra* 33 (October): S321–43. https://doi.org/10.1193/120616EQS223M.

Nishida, Kiwamu. 2017. “Ambient Seismic Wave Field.” *Proceedings of the Japan Academy. Series B, Physical and Biological Sciences* 93 (7): 423–48. https://doi.org/10.2183/pjab.93.026.

Peña, Fernando, Paulo B. Lourenço, and Alfredo Campos-Costa. 2008a. "Experimental Dynamic Behavior of Free-Standing Multi-Block Structures under Seismic Loadings." *Journal of Earthquake Engineering* 12 (6): 953–79. https://doi.org/10.1080/13632460801890513.

———. 2008b. "Experimental Dynamic Behavior of Free-Standing Multi-Block Structures under Seismic Loadings." *Journal of Earthquake Engineering* 12 (6): 953–79. https://doi.org/10.1080/13632460801890513.

Peña, Fernando, Francisco Prieto-Castrillo, Paulo Lourenco, A Costa, and Jose Lemos. 2007. "On the Dynamics of Rocking Motion of Single Rigid-Block Structures."

Penna, Andrea, Paolo Morandi, Maria Rota, Carlo Filippo Manzini, Francesca da porto, and Guido Magenes. 2013. "Performance of Masonry Buildings during the Emilia 2012 Earthquake." *Bulletin of Earthquake Engineering* 12. https://doi.org/10.1007/s10518-013-9496-6.

Pulatsu, Bora, Ece Erdogmus, Paulo B Lourenço, Jose V Lemos, and Kagan Tuncay. 2020. "Simulation of the In-Plane Structural Behavior of Unreinforced Masonry Walls and Buildings Using DEM." *Structures* 27: 2274–87. https://doi.org/https://doi.org/10.1016/j.istruc.2020.08.026.

Q., Ma, Parshottam S., and M Montalla. 2018. "Modelling Rocking Behaviour," no. July.

Scattarreggia, Nicola, Daniele Malomo, and Matthew DeJong. 2022. "A New Distinct Element Meso-Model for Simulating the Rocking-Dominated Seismic Response of RC Columns." *Earthquake Engineering & Structural Dynamics* 52. https://doi.org/10.1002/eqe.3782.

Shamy, Usama El, and Natasha Zamani. 2012. "Discrete Element Method Simulations of the Seismic Response of Shallow Foundations Including Soil-Foundation-Structure Interaction." *International Journal for Numerical and Analytical Methods in Geomechanics* 36 (10): 1303–29. https://doi.org/https://doi.org/10.1002/nag.1054.

So, Emily, and Robin Spence. 2013. "Estimating Shaking-Induced Casualties and Building Damage for Global Earthquake Events: A Proposed Modelling Approach." *Bulletin of Earthquake Engineering* 11 (1): 347–63. https://doi.org/10.1007/s10518-012-9373-8.

Vassiliou, Michalis, Kevin Mackie, and Bozidar Stojadinovic. 2016. "A Finite Element Model for Seismic Response Analysis of Deformable Rocking Frames." *Earthquake Engineering & Structural Dynamics* 46. https://doi.org/10.1002/eqe.2799.

Vlachakis, Georgios, Anastasios I Giouvanidis, Anjali Mehrotra, and Paulo B Lourenço. 2021. "Numerical Block-Based Simulation of Rocking Structures Using a Novel Universal Viscous Damping Model" 147 (11): 1–17. https://doi.org/10.1061/(ASCE)EM.1943-7889.0001985.

Xiao, Junsen, Kenta Tozato, Shuji Moriguchi, Yu Otake, and Kenjiro Terada. 2023. "Quantification of the Contribution Ratio of Relevant Input Parameters on DEM-Based Granular Flow Simulations." *Soils and Foundations* 63 (6): 101378. https://doi.org/https://doi.org/10.1016/j.sandf.2023.101378.

Xing, Yi Si, Xiaoping P. Liu, and Shao Ping Xu. 2010. "Efficient Collision Detection Based on AABB Trees and Sort Algorithm." *2010 8th IEEE International Conference on Control and Automation, ICCA 2010*, 328–32. https://doi.org/10.1109/ICCA.2010.5524093.

Xu, Zhen, Xinzheng Lu, Hong Guan, Bo Han, and A Ren. 2014. “Seismic Damage Simulation in Urban Areas Based on a High-Fidelity Structural Model and a Physics Engine.” *Natural Hazards* 71. https://doi.org/10.1007/s11069-013-0972-8.

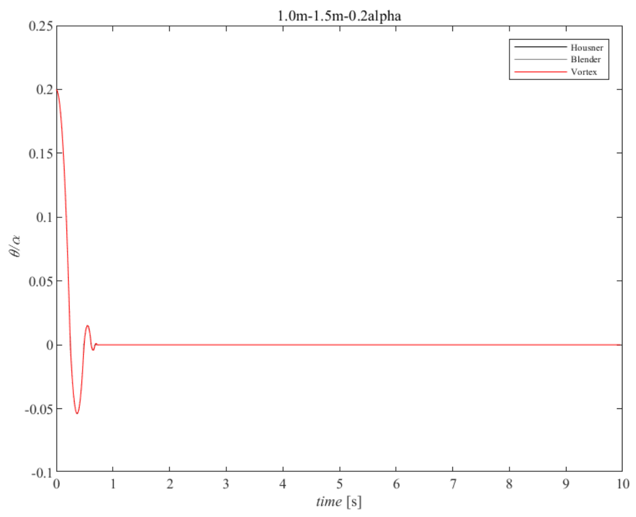
Zheng, Zhe, Yuan Tian, Zhebiao Yang, and Xinzheng Lu. 2020. “Hybrid Framework for Simulating Building Collapse and Ruin Scenarios Using Finite Element Method and Physics Engine.” *Applied Sciences (Switzerland)*. https://doi.org/10.3390/app10124408.

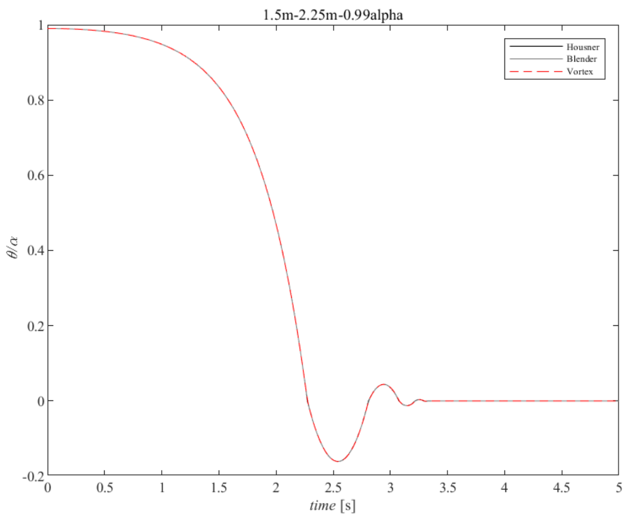
Zhong, Chiyun, and Constantin Christopoulos. 2021. “Finite Element Analysis of the Seismic Shake-Table Response of a Rocking Podium Structure.” *Earthquake Engineering and Structural Dynamics* 50 (4): 1223–30. https://doi.org/10.1002/eqe.3397.

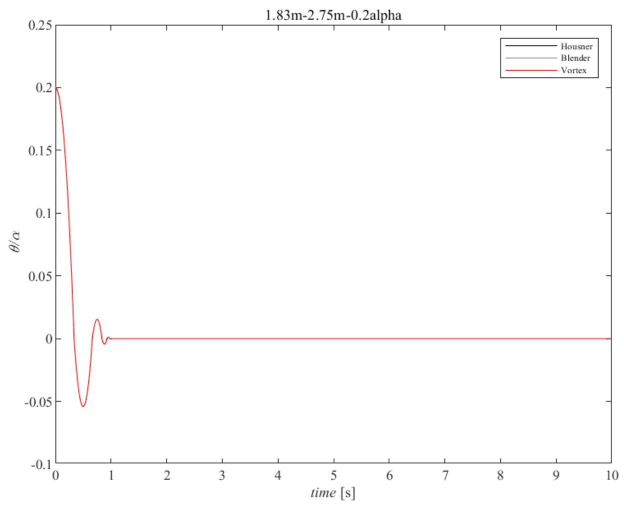
Zienkiewicz, O.C, R.L Taylor, and J.Z Zhu. 2005. “Finite Element Method for Solid and Structural Mechanics,” no. 1: 6–8.

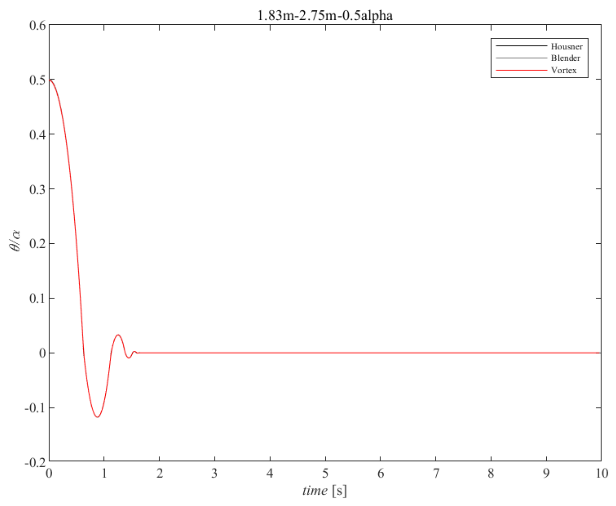
Appendix I – simulation results unpresented in the main text

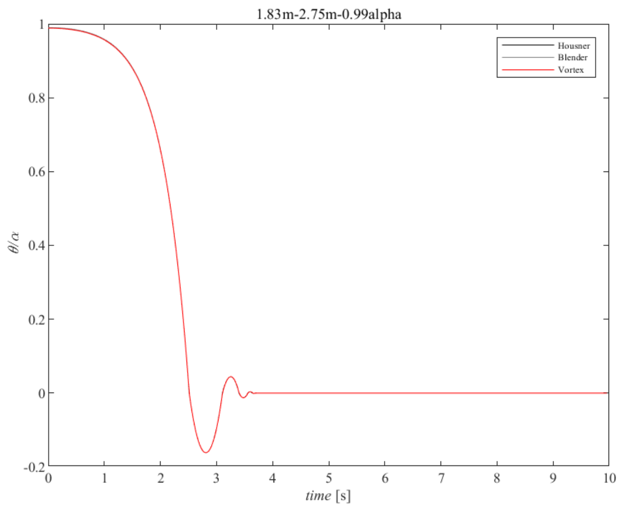
Part 1 – free rocking simulations 1.5 aspect ratio

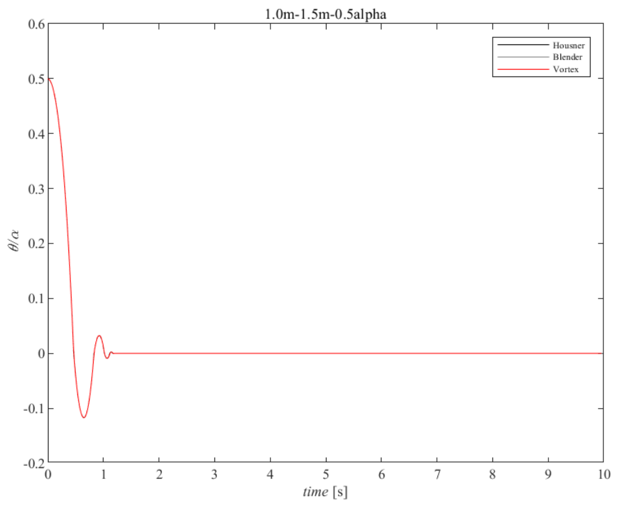


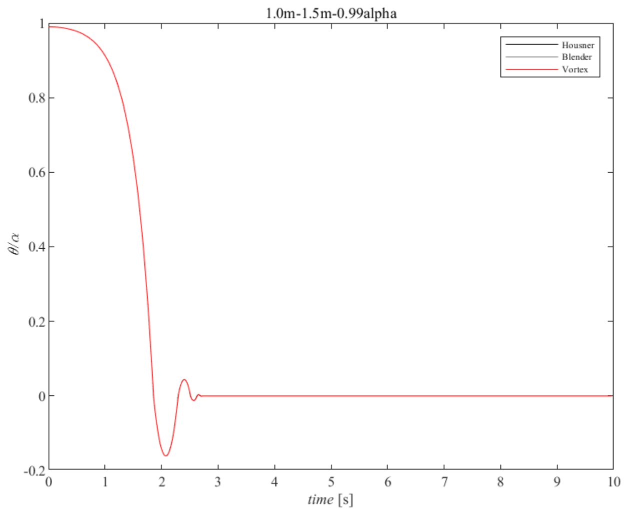


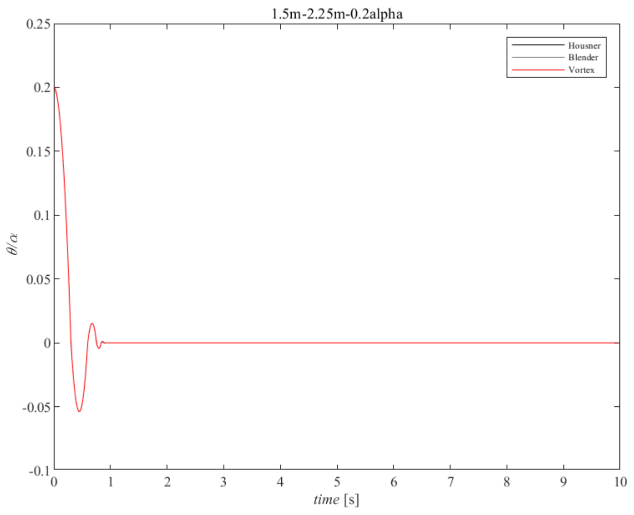


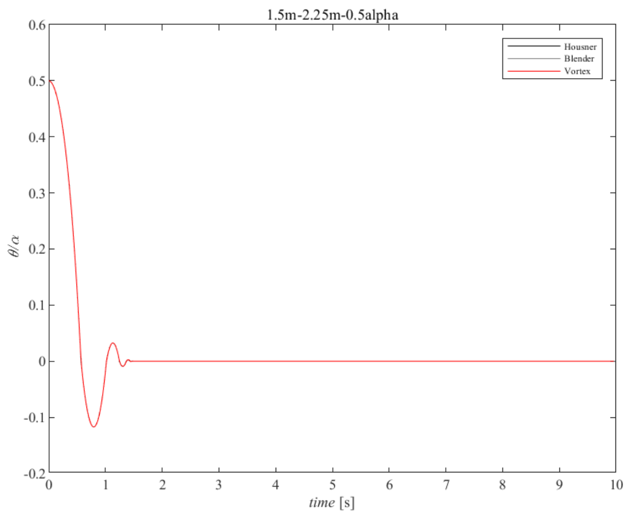



1.83m-2.75m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

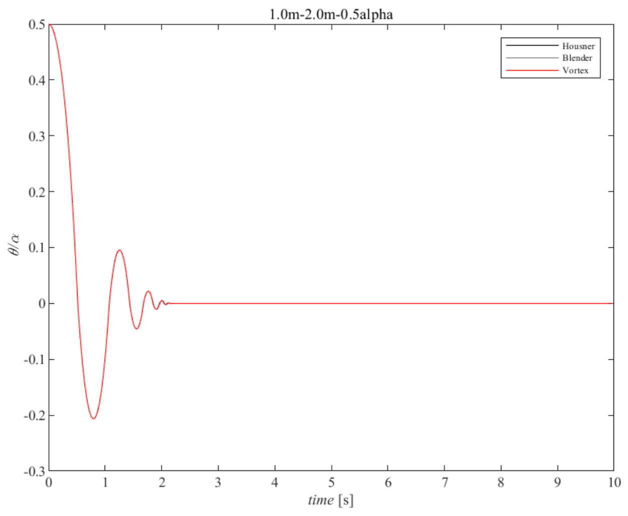

1.0m-1.5m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]

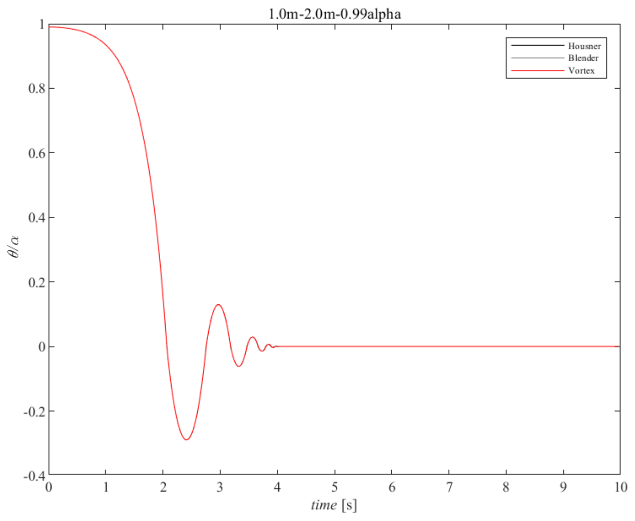

1.0m-1.5m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

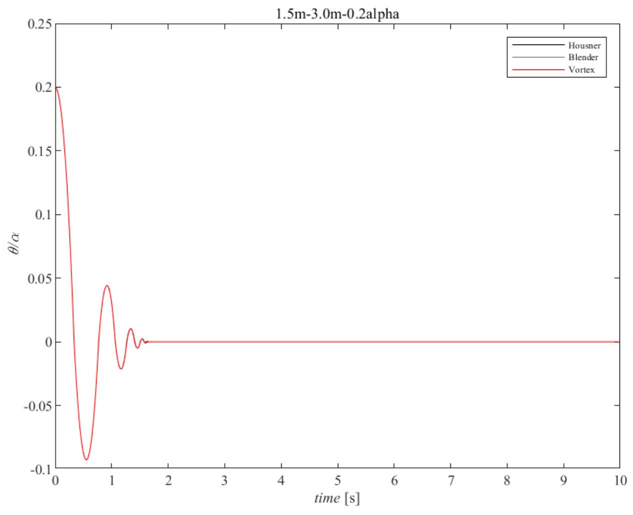

1.5m-2.25m-0.2alpha
Housner
Blender
Vortex
θ/α
time [s]

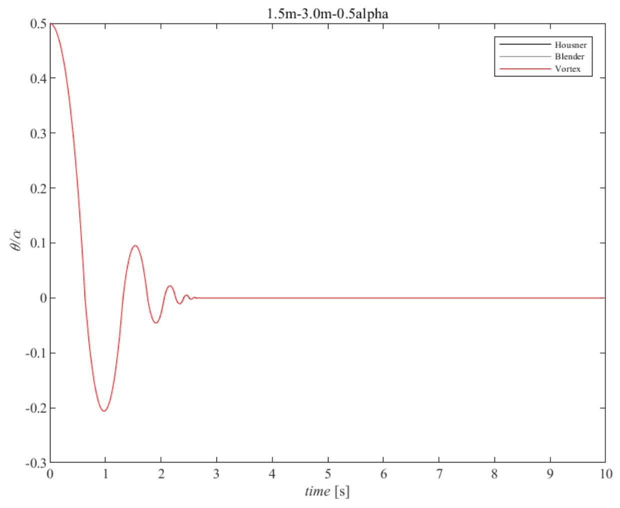

1.5m-2.25m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]

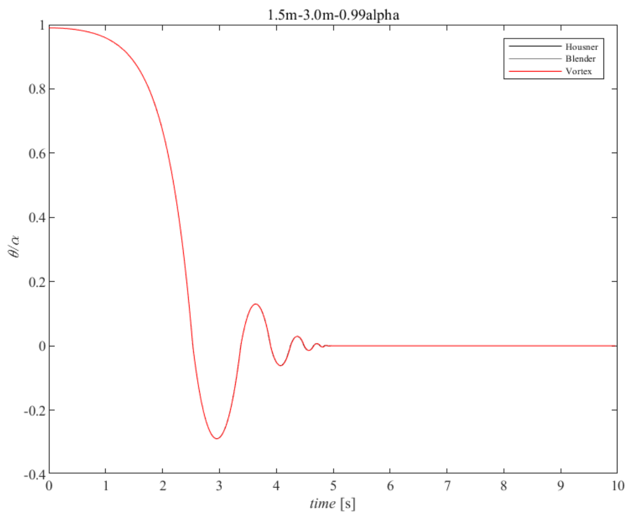
Part 1 – free rocking simulations 2.0 aspect ratio

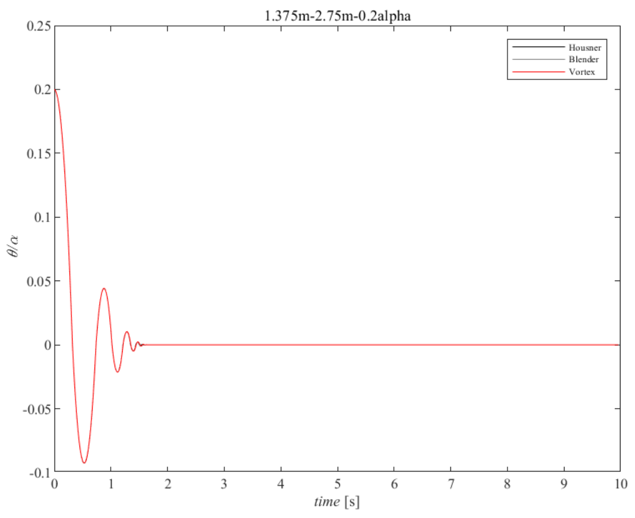


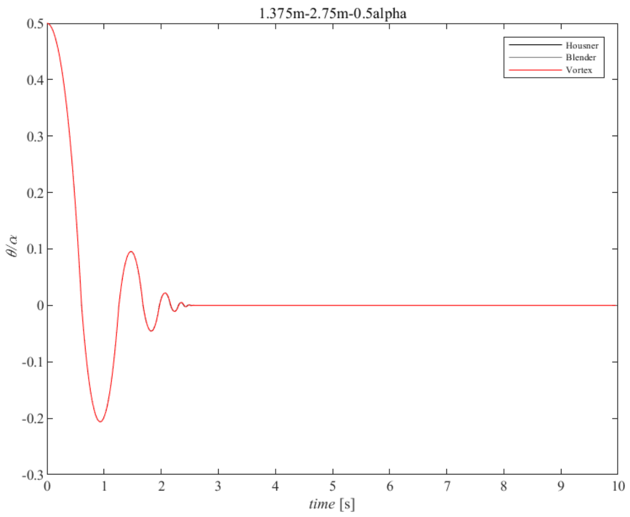


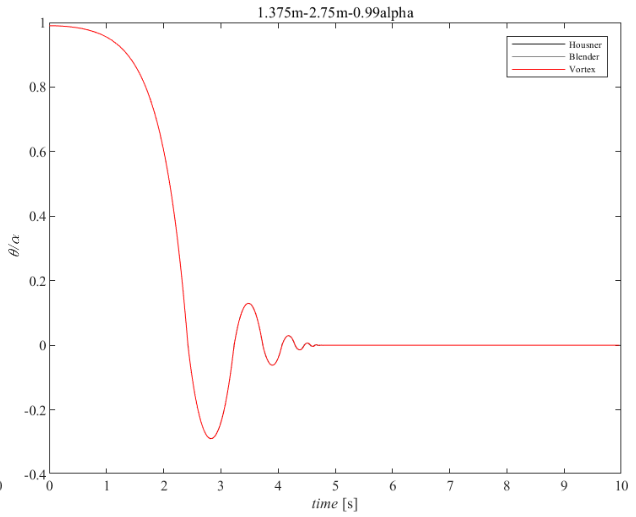


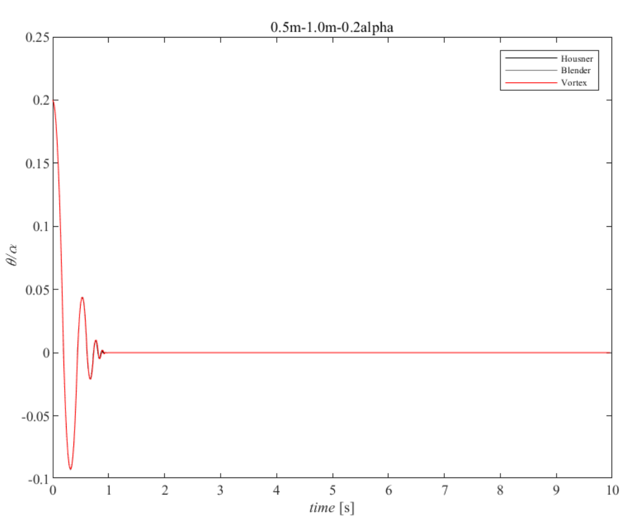


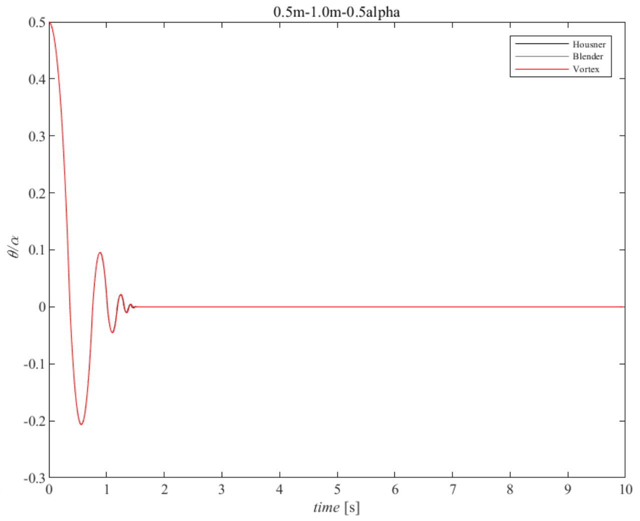


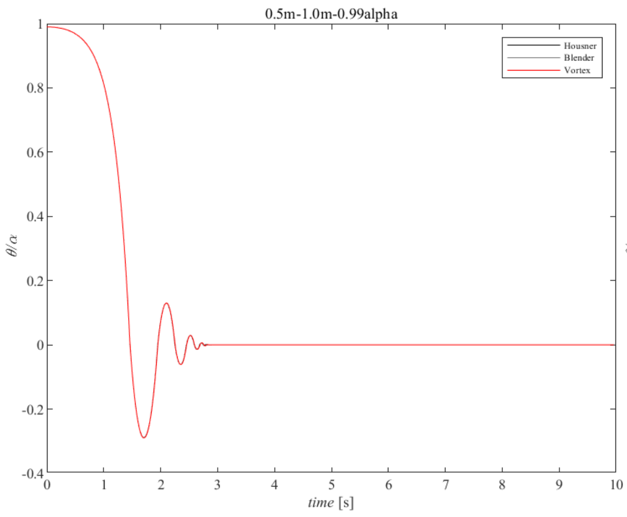


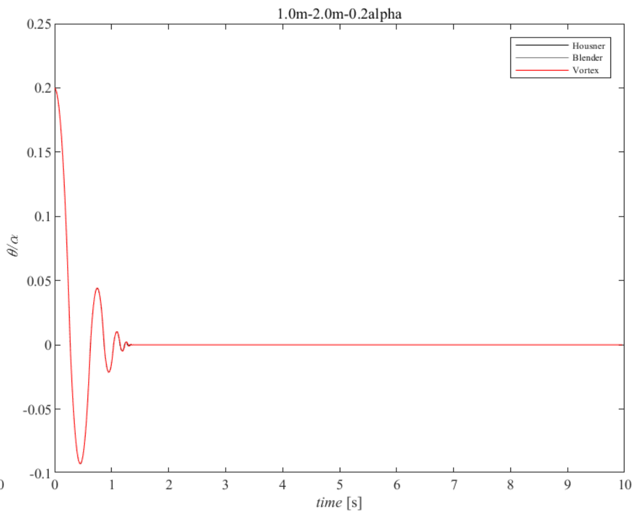

1.375m-2.75m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]


1.375m-2.75m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

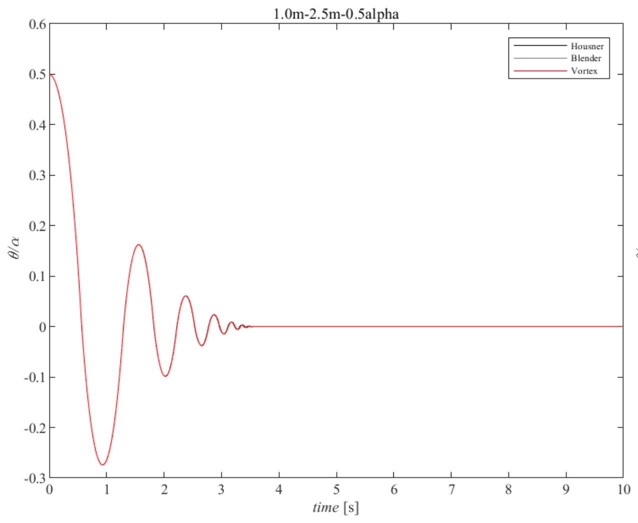

0.5m-1.0m-0.2alpha
Housner
Blender
Vortex
θ/α
time [s]

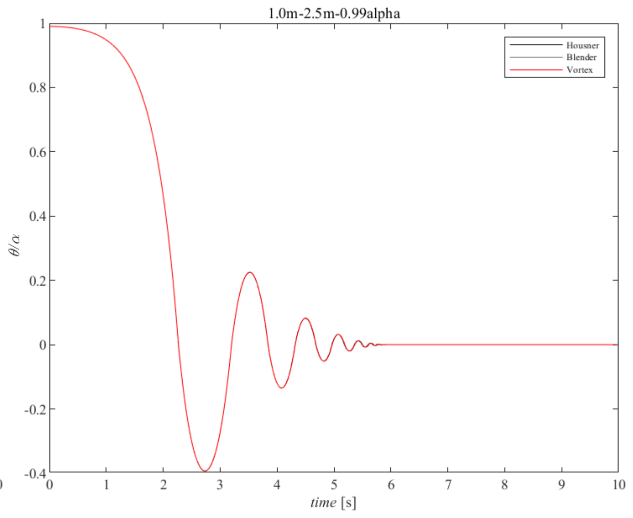

0.5m-1.0m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]

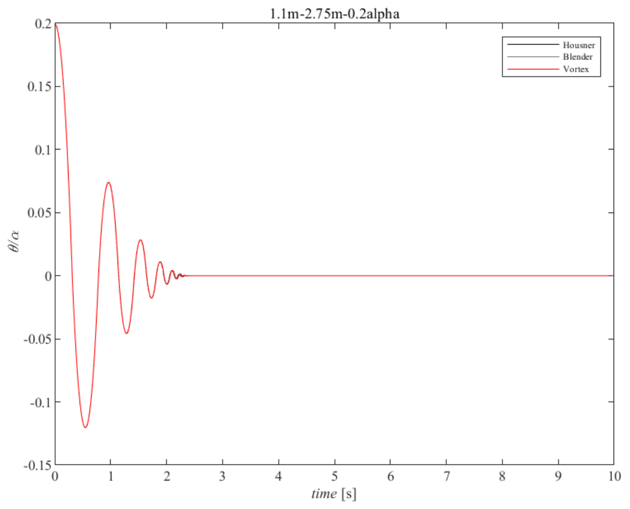

0.5m-1.0m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

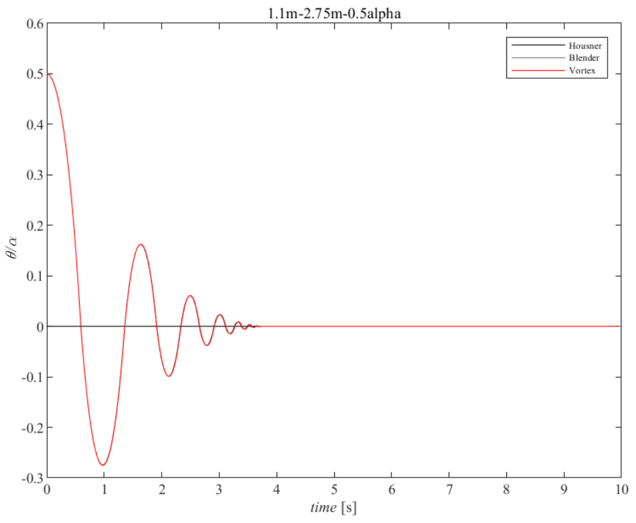

1.0m-2.0m-0.2alpha
Housner
Blender
Vortex
θ/α
time [s]

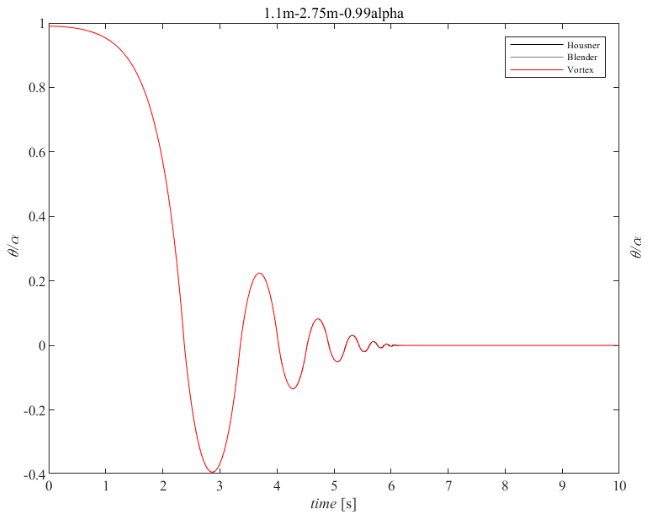
Part 1 – free rocking simulations 2.5 aspect ratio

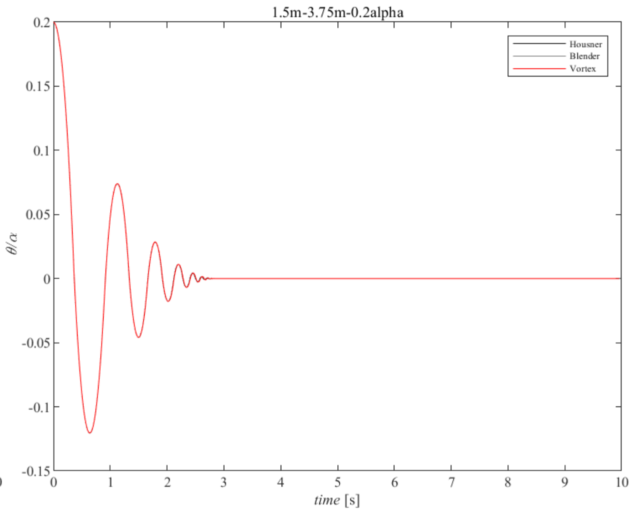


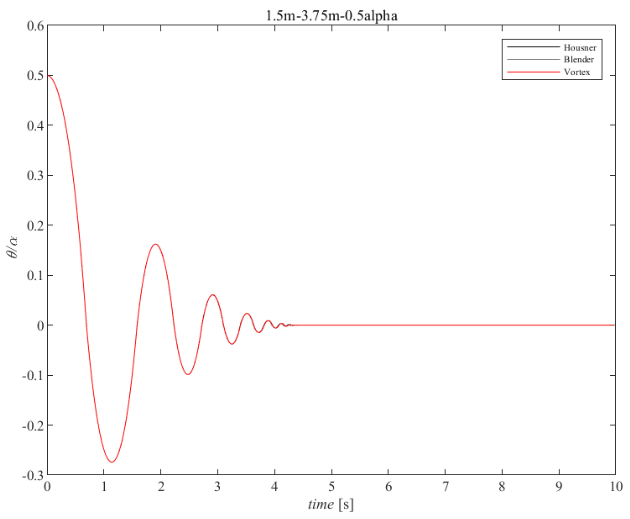


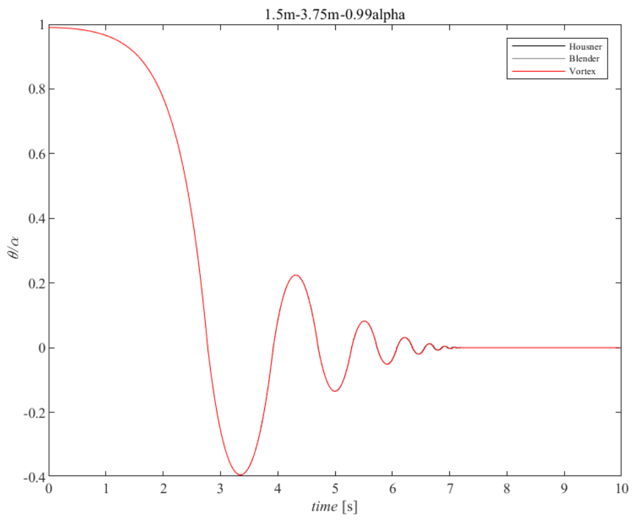


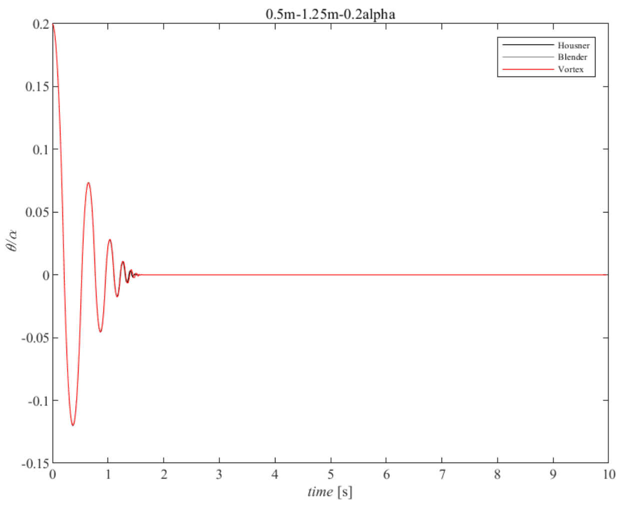


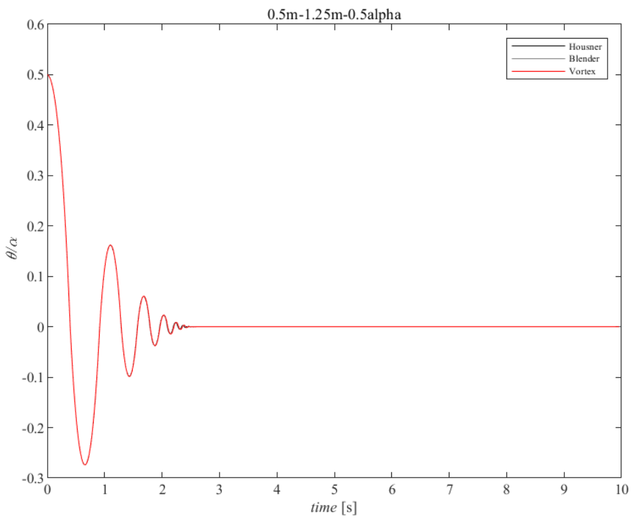


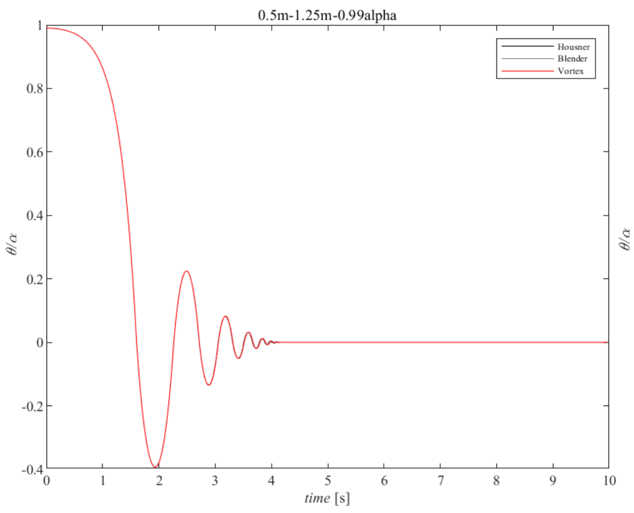


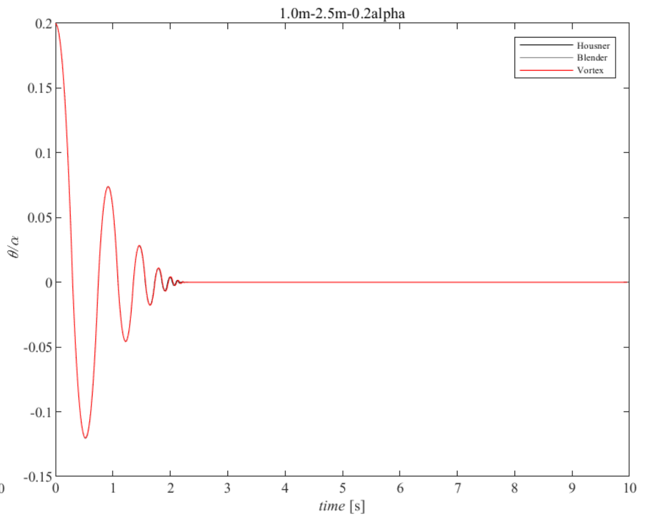

1.5m-3.75m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]


1.5m-3.75m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

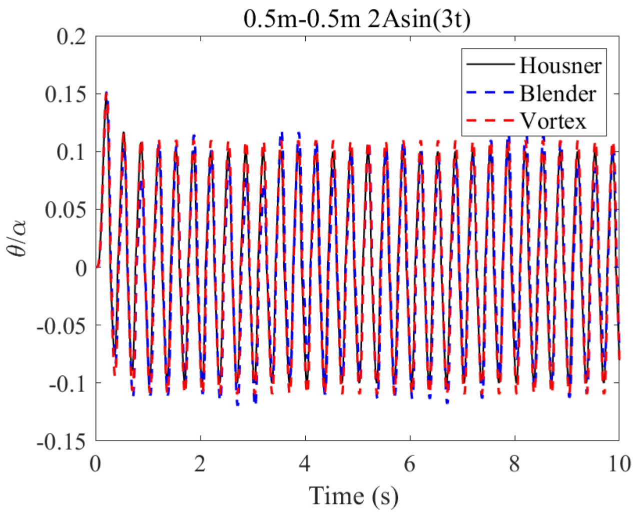

0.5m-1.25m-0.2alpha
Housner
Blender
Vortex
θ/α
time [s]

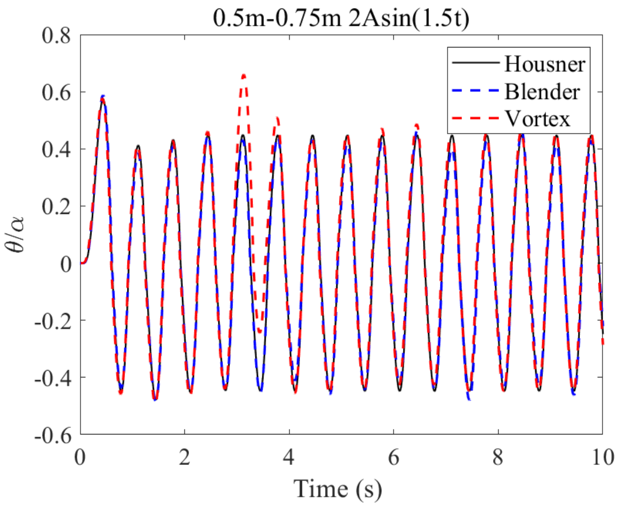

0.5m-1.25m-0.5alpha
Housner
Blender
Vortex
θ/α
time [s]

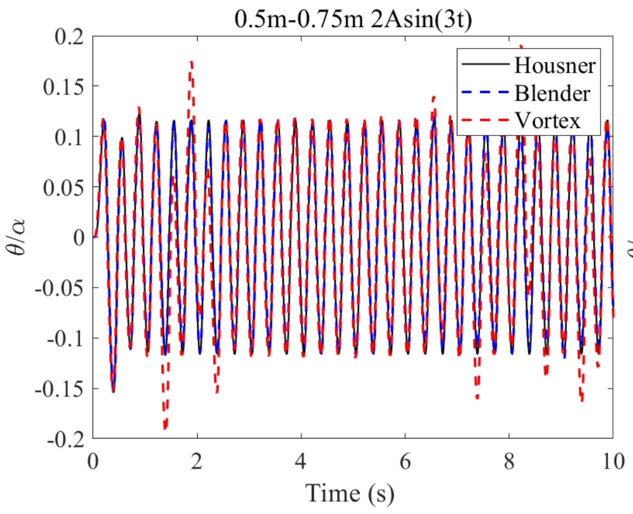

0.5m-1.25m-0.99alpha
Housner
Blender
Vortex
θ/α
time [s]

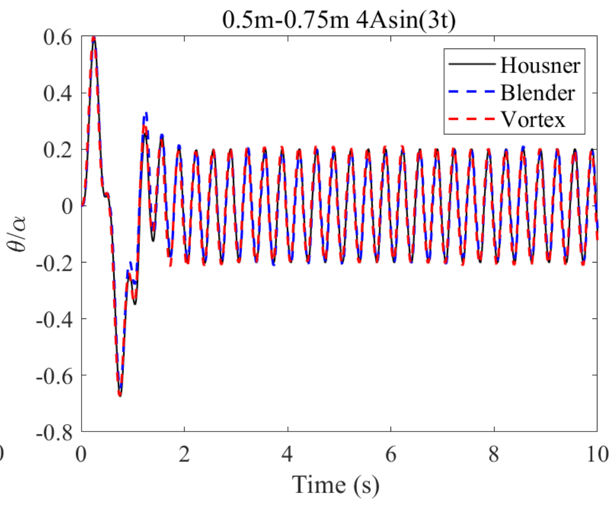

1.0m-2.5m-0.2alpha
Housner
Blender
Vortex
θ/α
time [s]

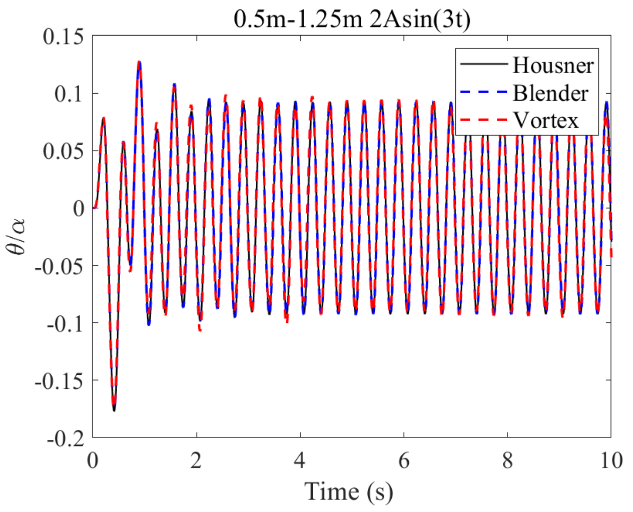
Part 2 – Forced rocking simulations

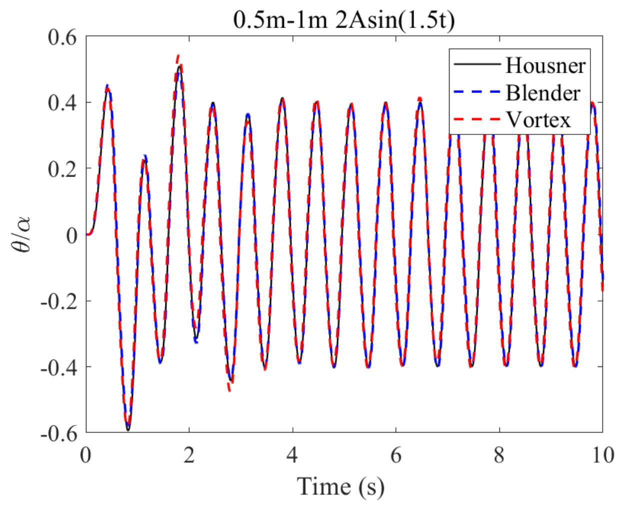


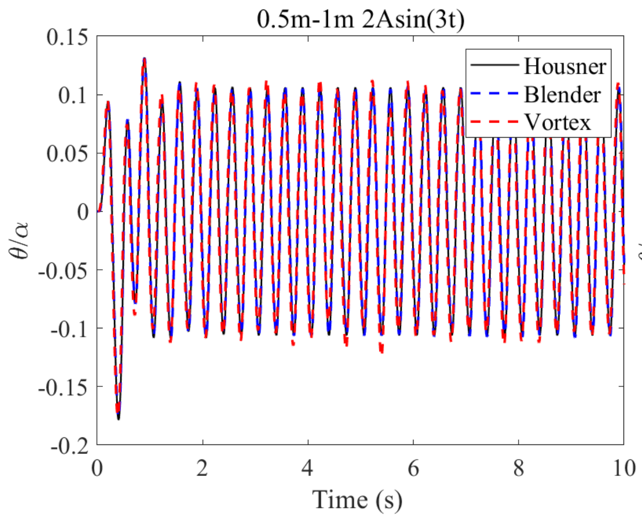


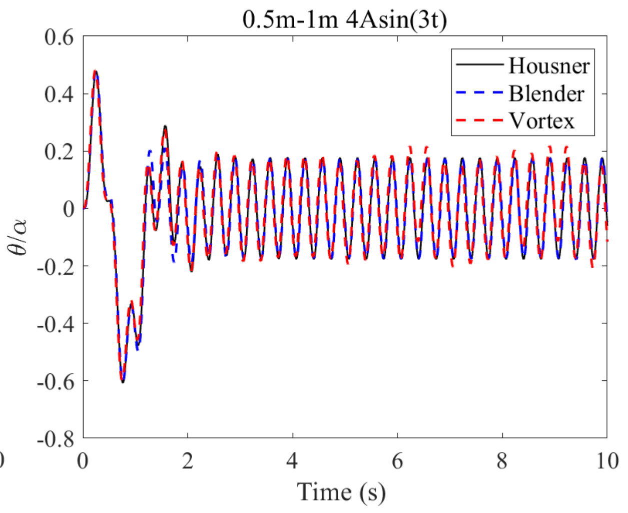


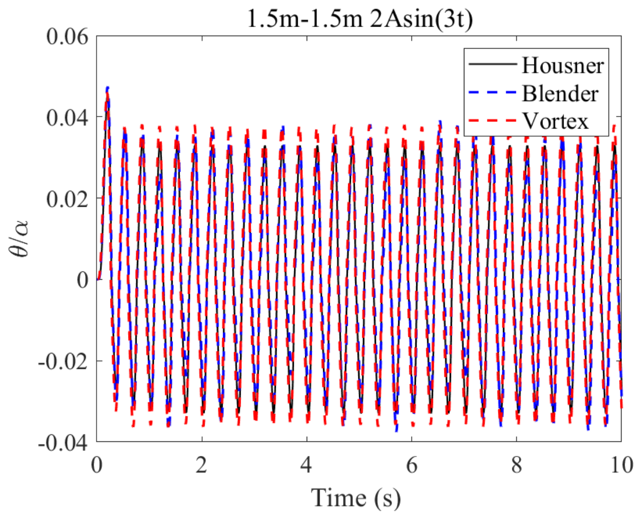


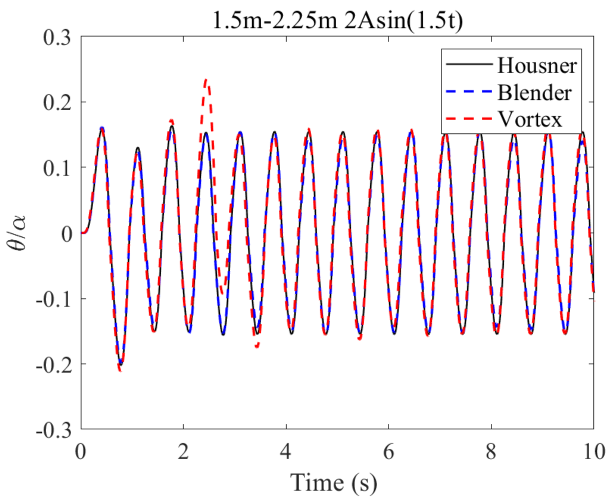


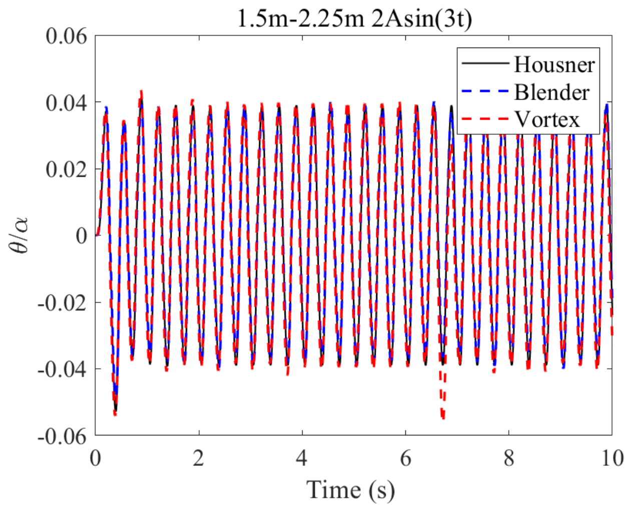


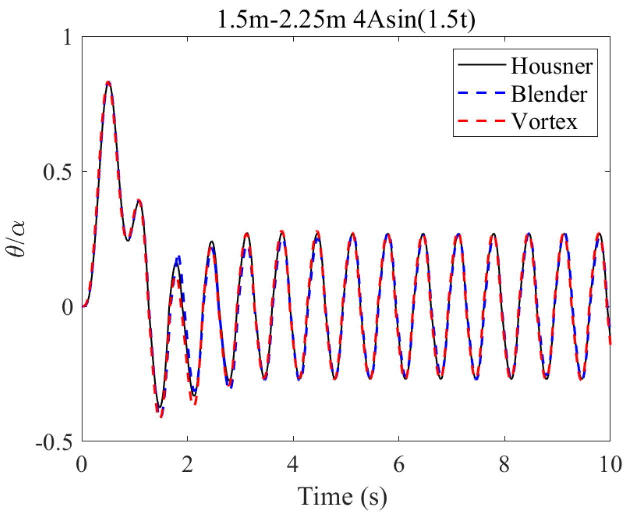

0.5m-1m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

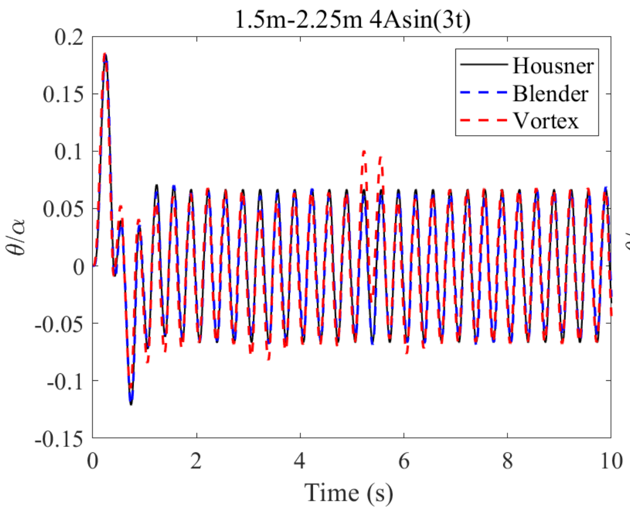
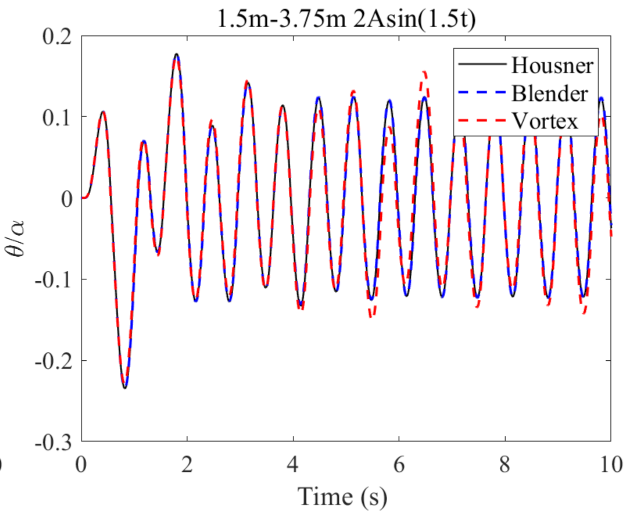
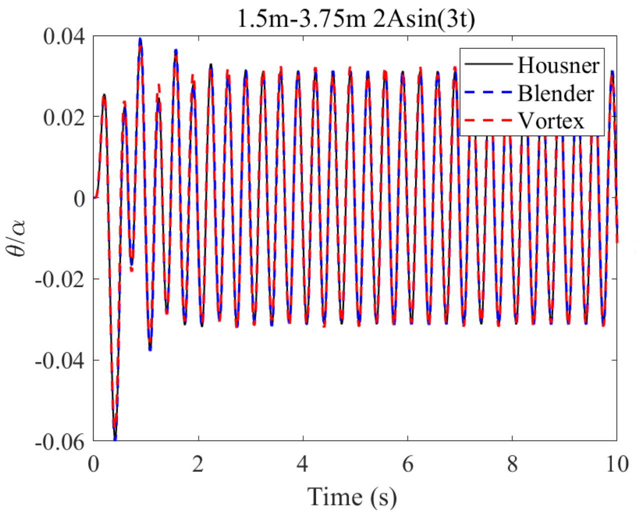
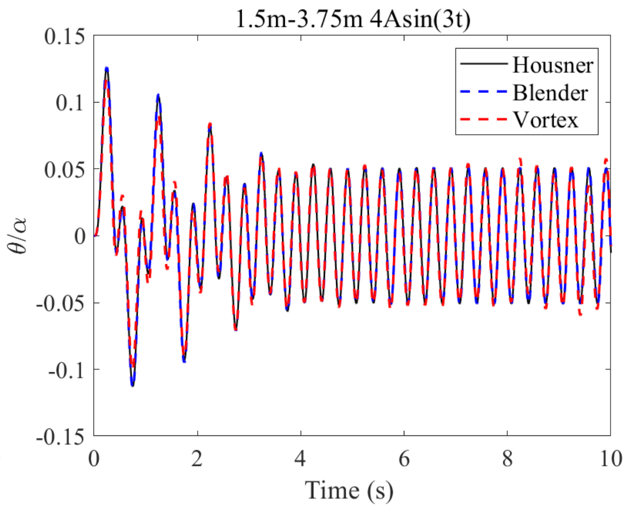
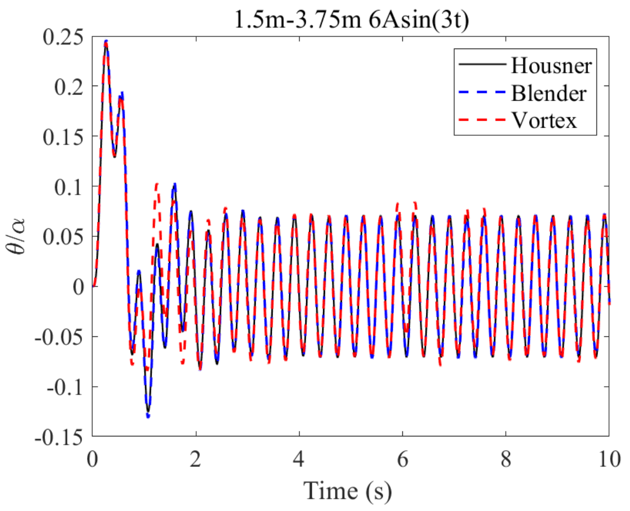
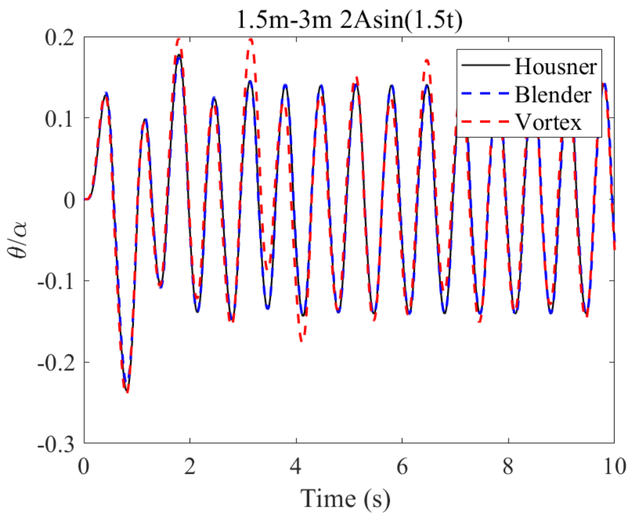

0.5m-1m 4Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

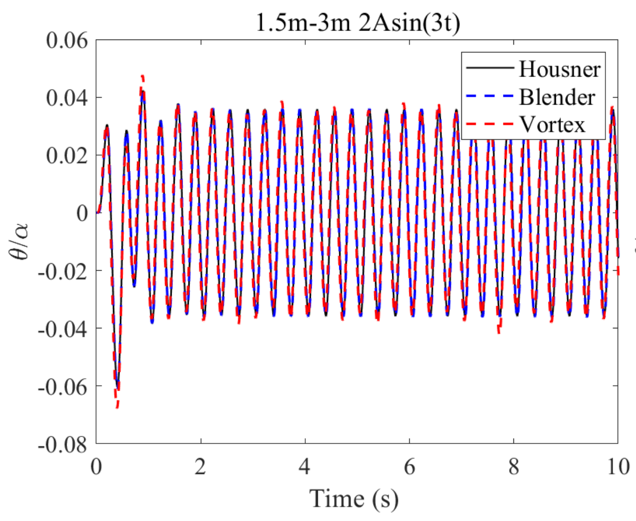

1.5m-1.5m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

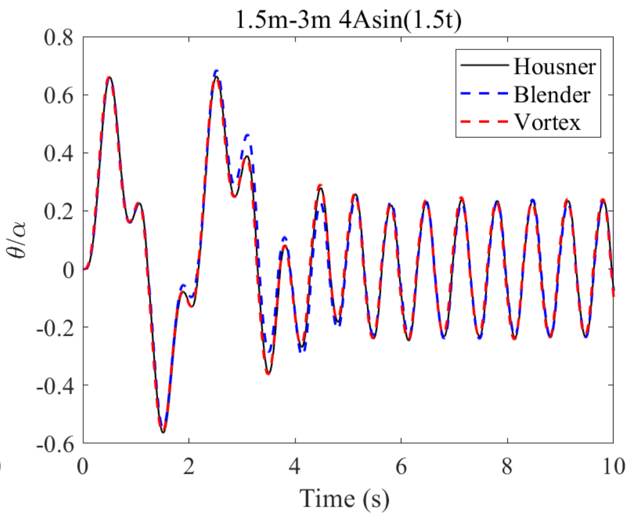

1.5m-2.25m 2Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

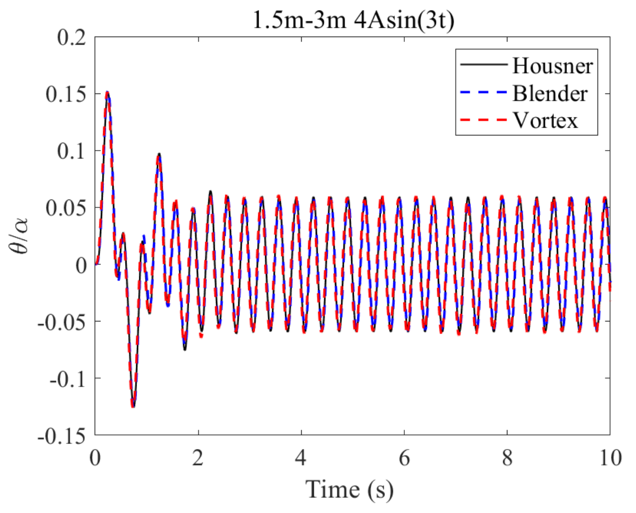

1.5m-2.25m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

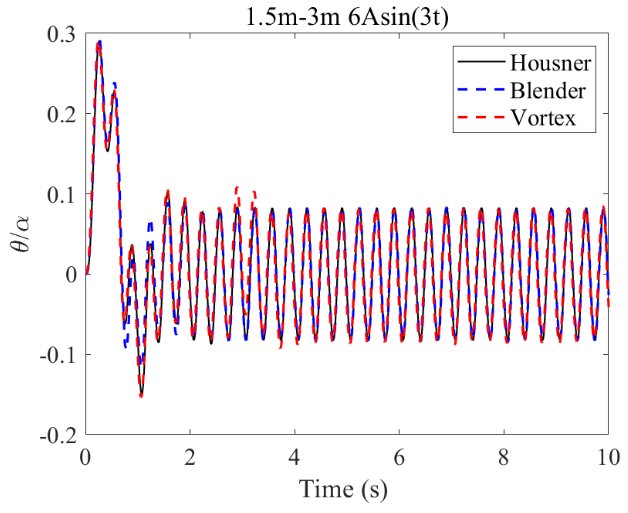

1.5m-2.25m 4Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

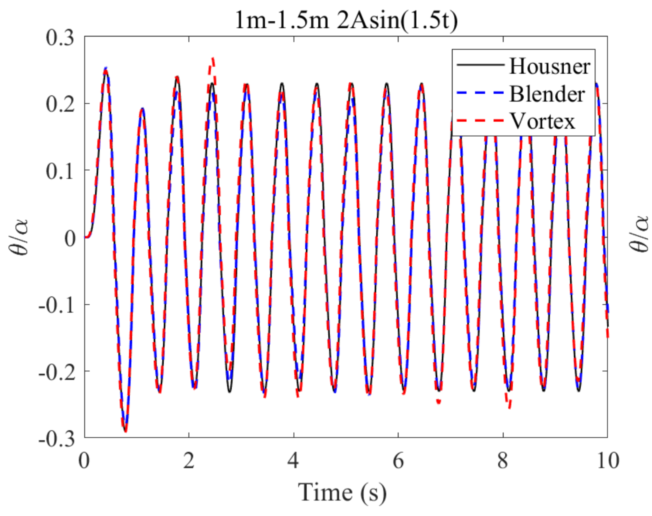

1.5m-2.25m 4Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

1.5m-3.75m 2Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

1.5m-3.75m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

1.5m-3.75m 4Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

1.5m-3.75m 6Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

1.5m-3m 2Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

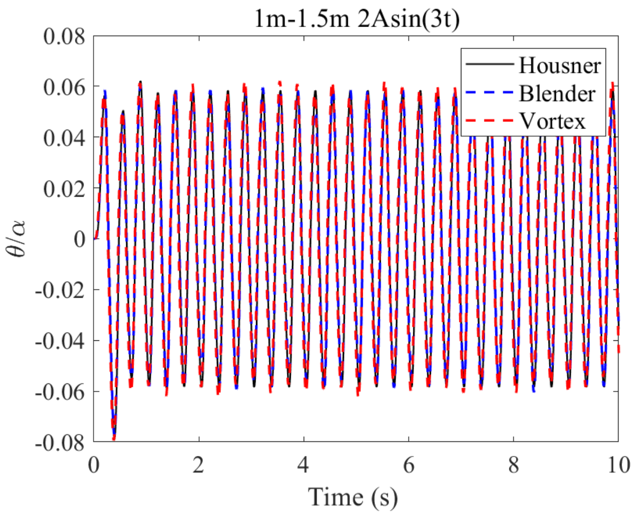

1.5m-3m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

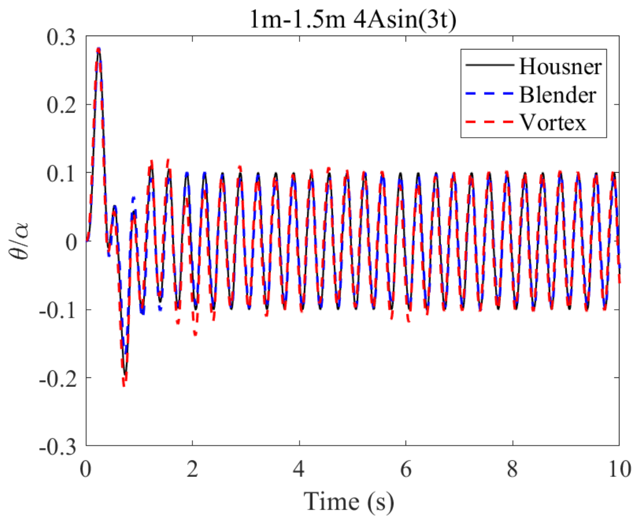

1.5m-3m 4Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

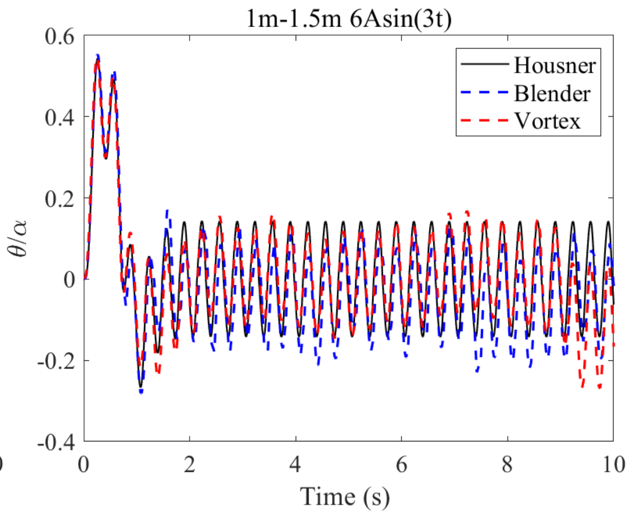

1.5m-3m 4Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

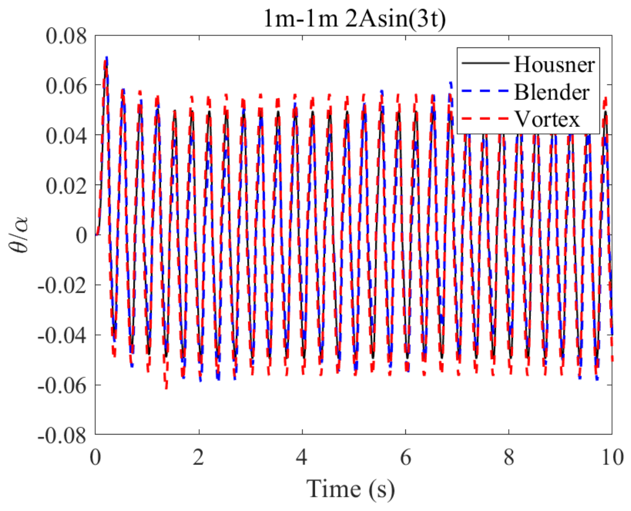

1.5m-3m 6Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

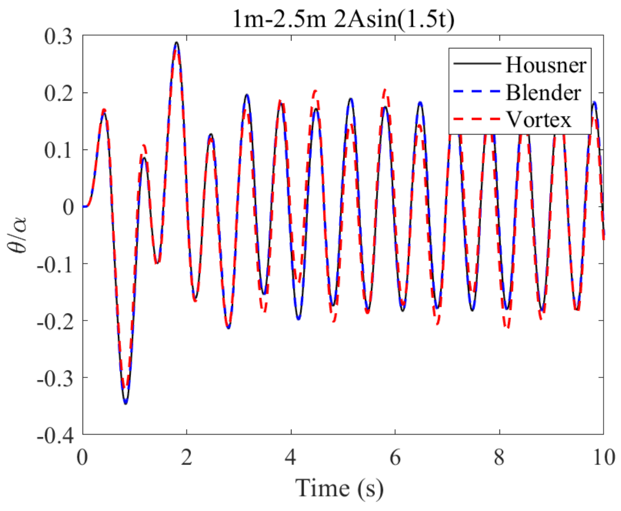

1m-1.5m 2Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

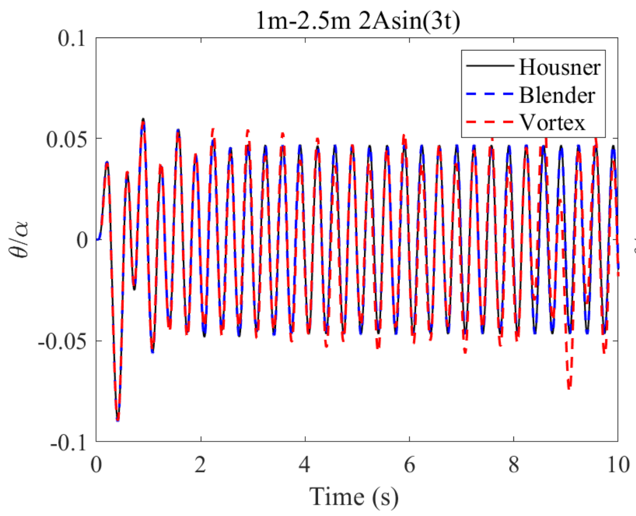

1m-1.5m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

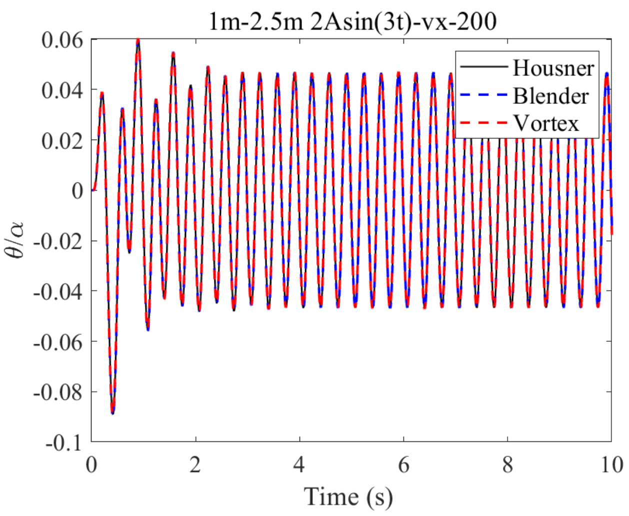

1m-1.5m 4Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

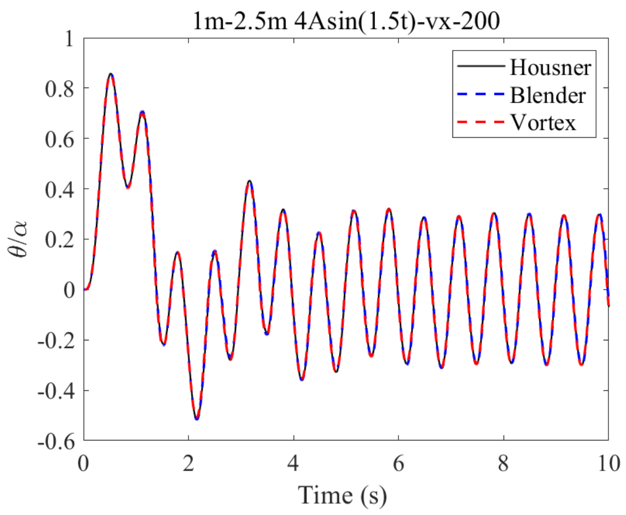

1m-1.5m 6Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

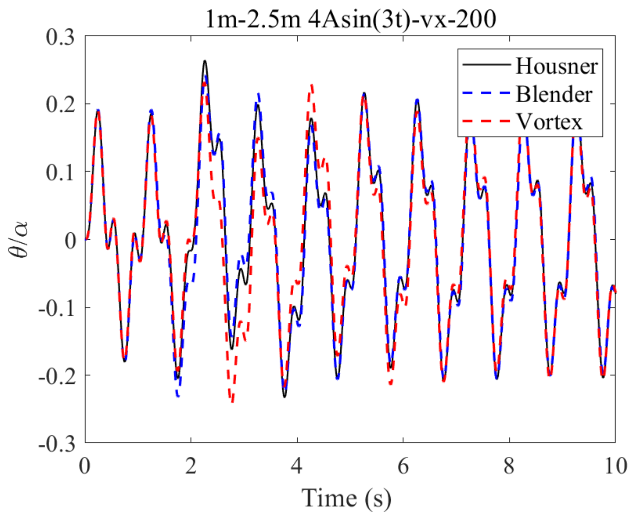

1m-1m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)

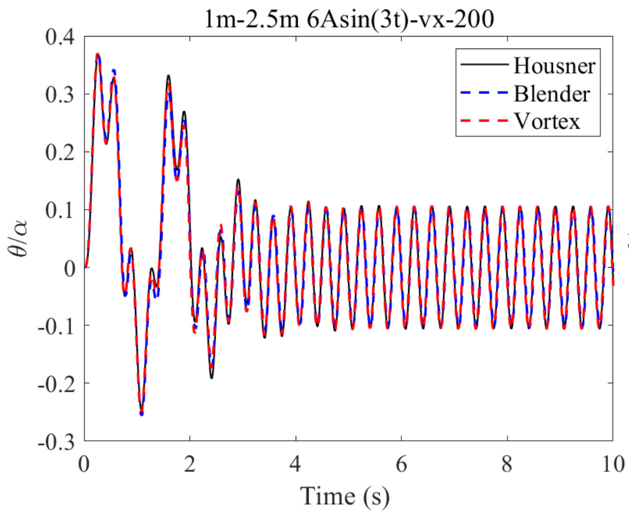

1m-2.5m 2Asin(1.5t)
Housner
Blender
Vortex
θ/α
Time (s)

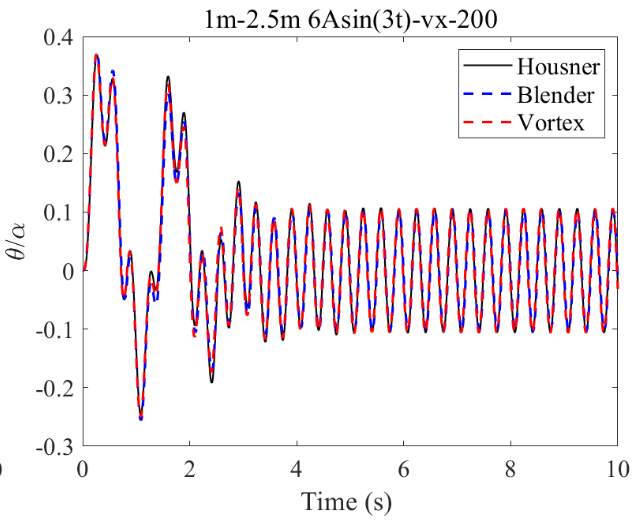

1m-2.5m 2Asin(3t)
Housner
Blender
Vortex
θ/α
Time (s)


1m-2.5m 2Asin(3t)-vx-200
Housner
Blender
Vortex
θ/α
Time (s)

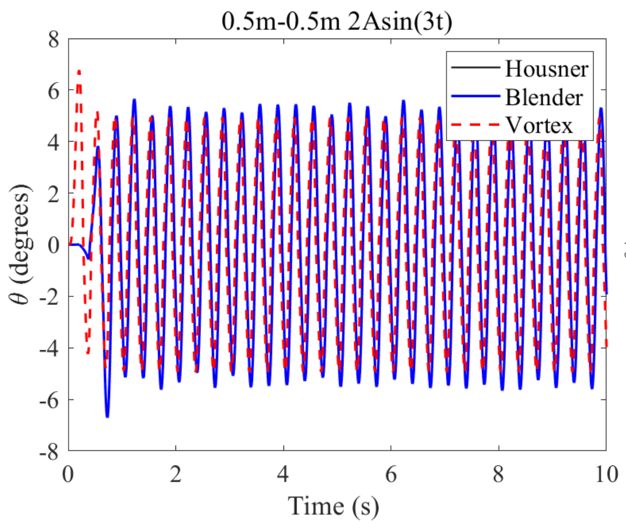

1m-2.5m 4Asin(1.5t)-vx-200
Housner
Blender
Vortex
θ/α
Time (s)

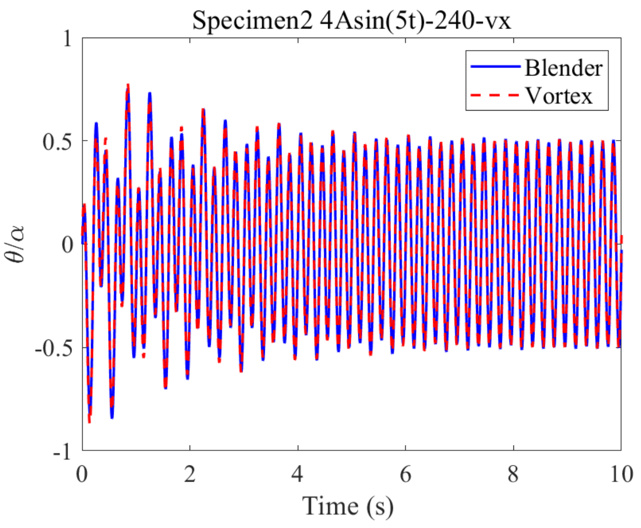

1m-2.5m 4Asin(3t)-vx-200
Housner
Blender
Vortex
θ/α
Time (s)

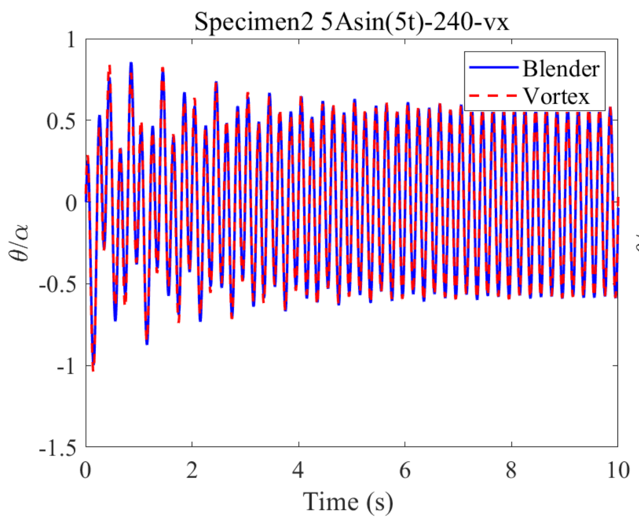

1m-2.5m 6Asin(3t)-vx-200
Housner
Blender
Vortex
θ/α
Time (s)

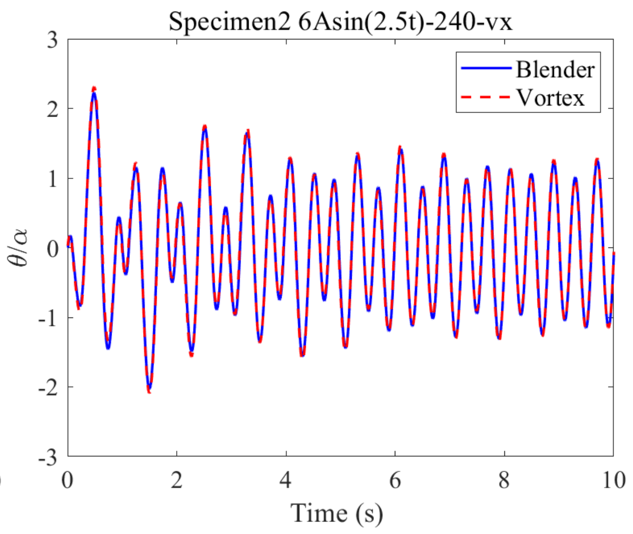

1m-2.5m 6Asin(3t)-vx-200
Housner
Blender
Vortex
θ/α
Time (s)

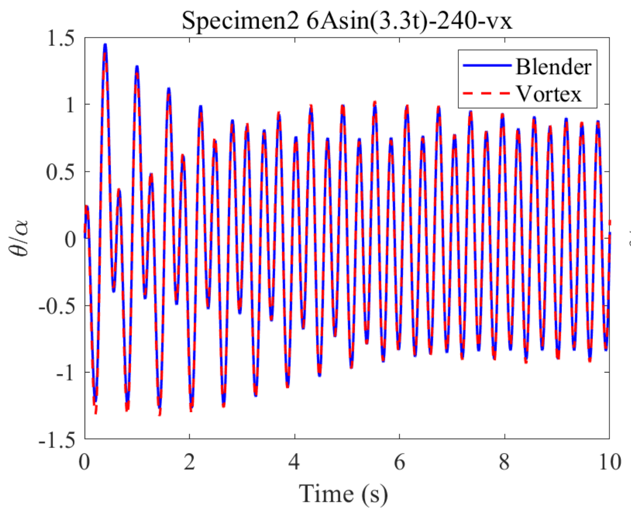
Part 3 – forced rocking for marble macro block models (untrimmed)

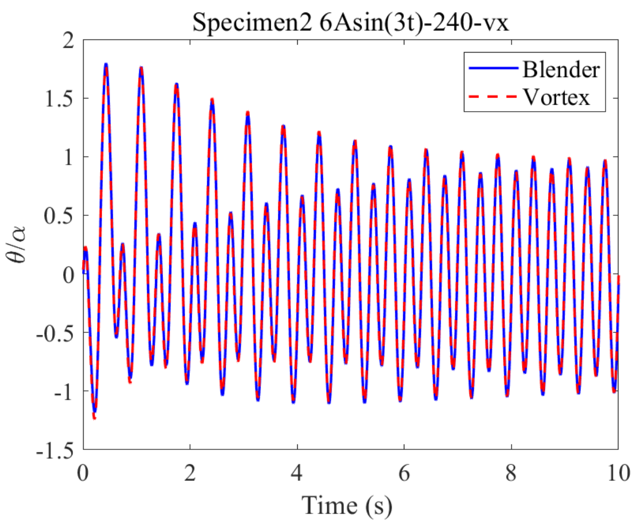


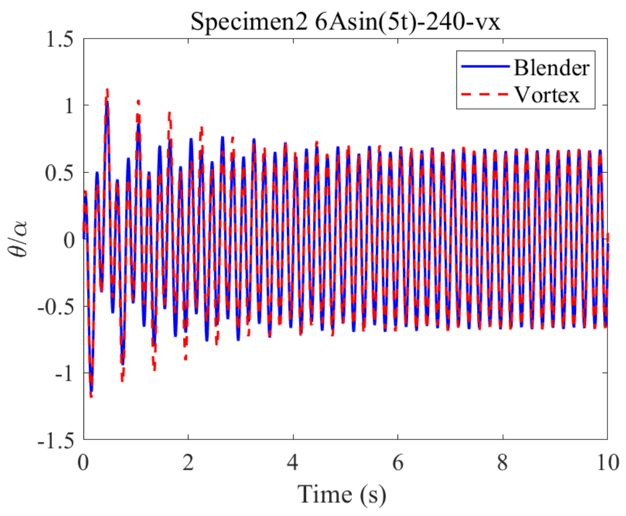
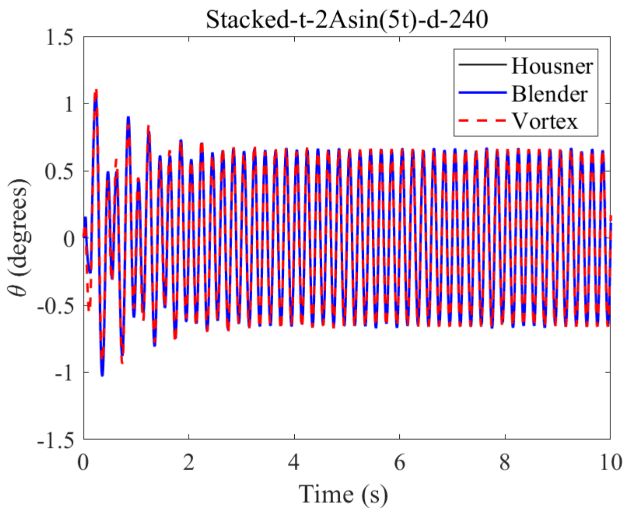
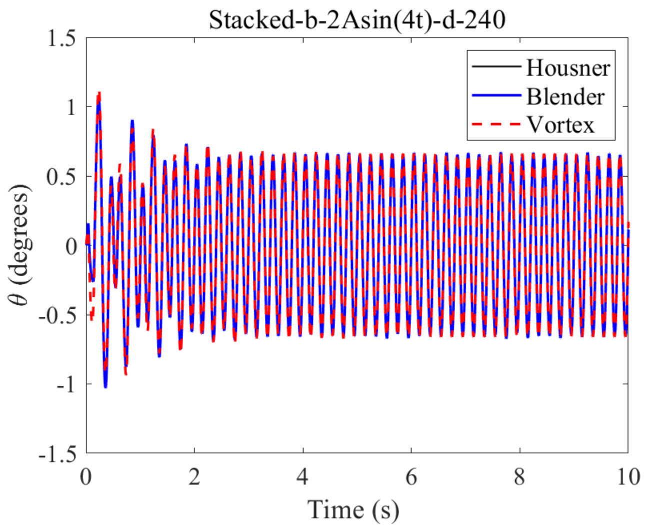




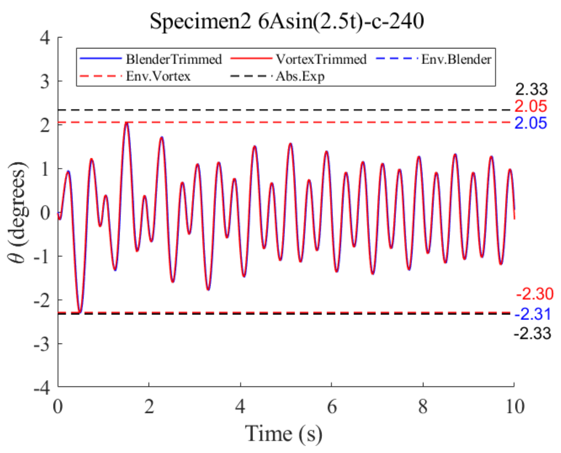


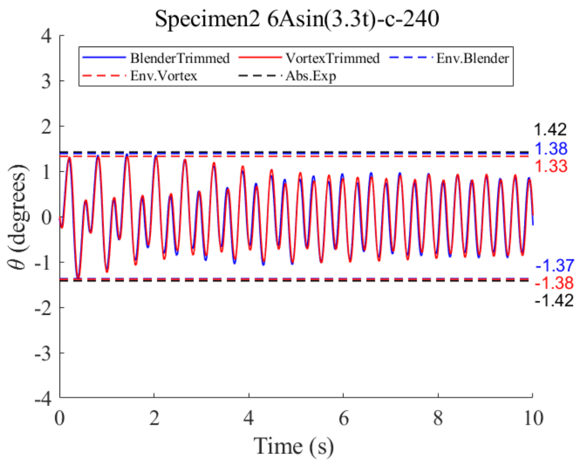


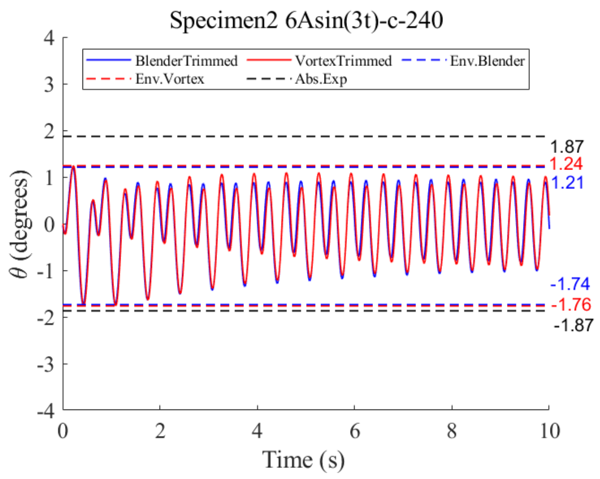


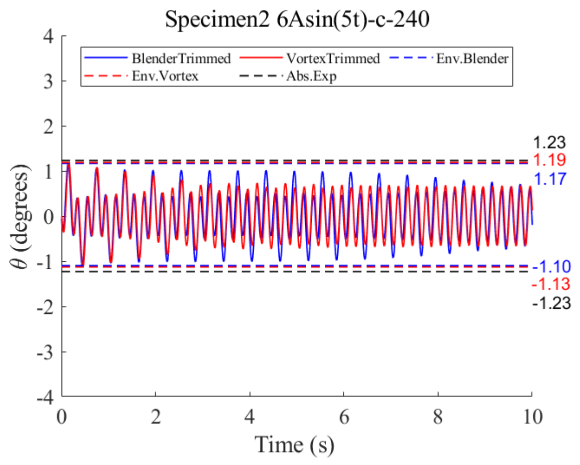

Specimen2 6Asin(5t)-240-vx
Blender
Vortex
θ/α
Time (s)

Stacked-t-2Asin(5t)-d-240
Housner
Blender
Vortex
θ (degrees)
Time (s)

Stacked-b-2Asin(4t)-d-240
Housner
Blender
Vortex
θ (degrees)
Time (s)

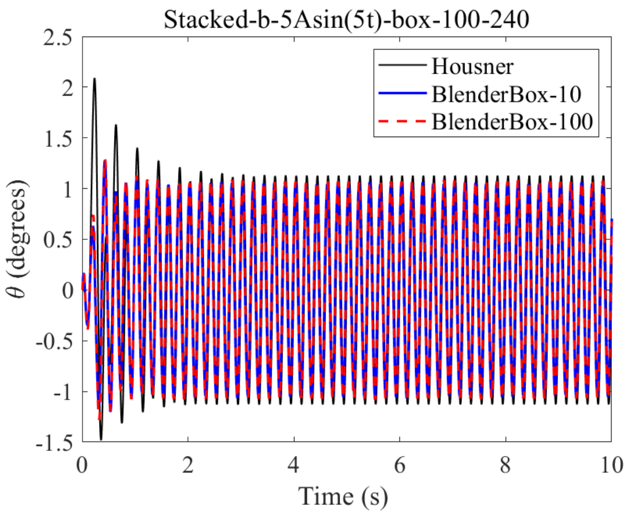
Part 3 – forced rocking for marble macro block models (trimmed + stacked blocks)

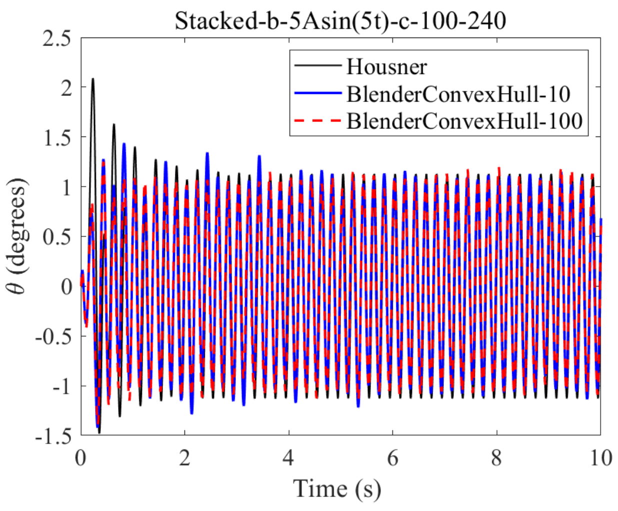


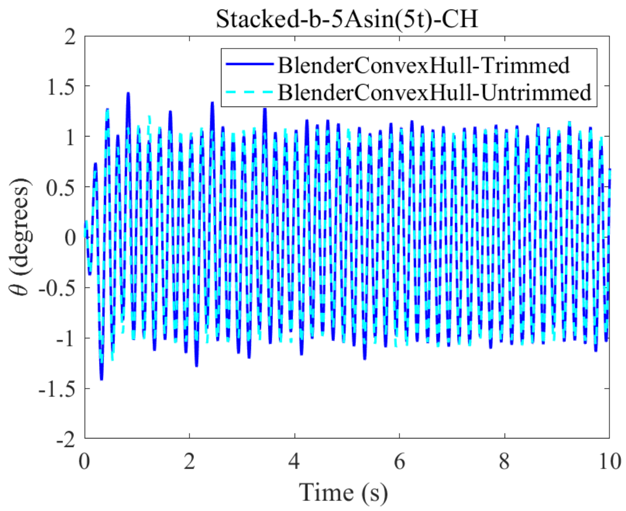


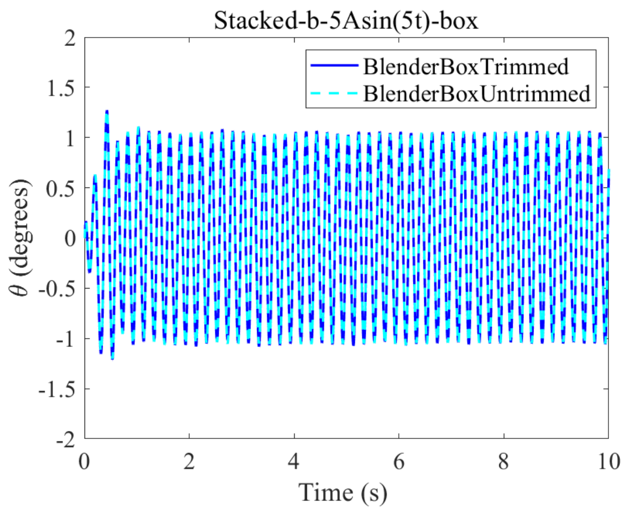


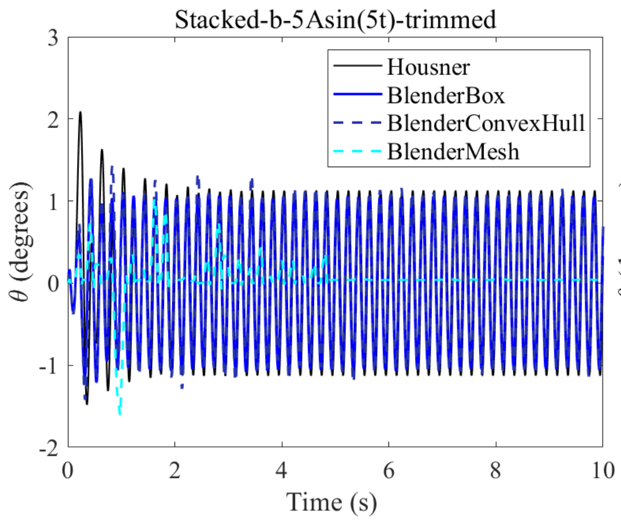


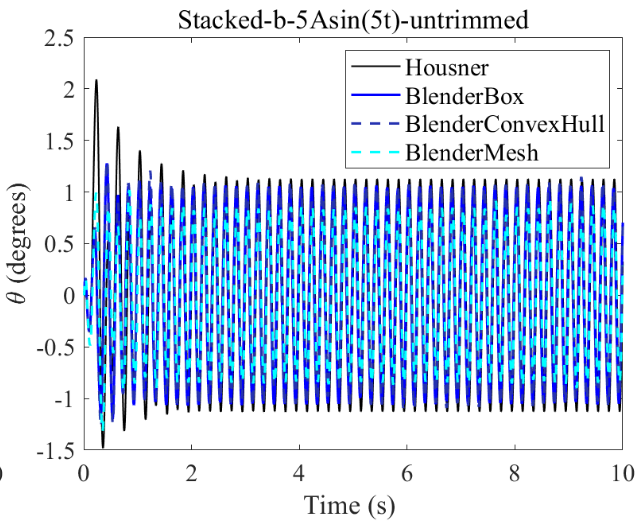





Stacked-b-5Asin(5t)-CH
BlenderConvexHull-Trimmed
BlenderConvexHull-Untrimmed
θ (degrees)
Time (s)


Stacked-b-5Asin(5t)-box
BlenderBoxTrimmed
BlenderBoxUntrimmed
θ (degrees)
Time (s)


Stacked-b-5Asin(5t)-trimmed
Housner
BlenderBox
BlenderConvexHull
BlenderMesh
θ (degrees)
Time (s)


Stacked-b-5Asin(5t)-untrimmed
Housner
BlenderBox
BlenderConvexHull
BlenderMesh
θ (degrees)
Time (s)

Appendix II – scripts utilized in the research process

Python scripts implemented in Blender.

1 – Exporting angle of rotations and translational data to a csv file

```
import bpy
import csv
from itertools import chain
from math import pi, radians, degrees

filepath = "your_file_path /0.25m-1m-0.5alpha.csv"

with open("your_file_path /0.25m-1m-0.5alpha.csv", 'w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file, dialect='excel')
    csv_writer.writerow(("frame","object","locX","locY","locZ","rotX","rotY","rotZ"))

    scene = bpy.context.scene
    frame_current = scene.frame_current

    for frame in range(scene.frame_start, scene.frame_end + 1):
        scene.frame_set(frame)
        for ob in scene.objects:
            if "BLOCK" in ob.name:
                mat = ob.matrix_world
                loc = mat.translation.xzy
                rot = mat.to_euler()
                scale = mat.to_scale()
                csv_writer.writerow(tuple(chain((frame, ob.name), loc, rot)))

    scene.frame_set(frame_current)
```

2 – Importing sinusoidal ground acceleration data (converted to displacement data per frame) from a csv file

```
import bpy
import csv

f = 1

csv_path = "your_file_path/ground motion 0.75m-1m 2Asin(3t)-d-240.csv"

ob = bpy.context.object

with open(csv_path) as file:
    groundmotioncsv = csv.reader(file, delimiter=",")

    for row in groundmotioncsv:

        ob.location = [float(v) for v in row[:3]]
        ob.keyframe_insert("location", frame=f)
        f += 1

        if f == 3000:

            break
```

3 – Creating two trimmed rectangular block model in blender

```
import bpy
import math

def create_block(name, dimensions, mass, location):
    bpy.ops.mesh.primitive_cube_add(location=location)
    cube = bpy.context.object
    cube.scale[0] = dimensions[0] / 2.0  # Divide by 2 because the default cube size is 2
    cube.scale[1] = dimensions[1] / 2.0
    cube.scale[2] = dimensions[2] / 2.0

    bpy.ops.object.transform_apply(scale=True)

    cube.name = name
    bpy.ops.rigidbody.object_add()
    cube.rigid_body.mass = mass
    bpy.ops.object.mode_set(mode='EDIT')
    bpy.ops.mesh.select_all(action='SELECT')
    bpy.ops.mesh.bevel(offset_type='OFFSET', offset=0.005 / math.sqrt(2), segments=1, profile=0.5)
    bpy.ops.object.mode_set(mode='OBJECT')

bpy.ops.object.select_all(action='DESELECT')
bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.delete()

create_block("bottom_block", [1, 0.75, 0.25], 178, [0, 0, 0.60 / 2])
create_block("top_block", [0.25, 0.754, 1], 97, [0, 0, 0.60 + 0.60 / 2])
```

Python script used in Vortex

1 - Implementation of ground motion and extraction of the movement of the object

```
from VxSim import *
from math import *

def on_add_to_universe(self, universe):
pass

def on_remove_from_universe(self, universe):
pass

def pre_step(self):
A_displacement = 5*0.001 # Amplitude of displacement
f = 3.3
t = self.getApplicationContext().getSimulationTime()
dt = self.getApplicationContext().getSimulationTimeStep()
```

```
displacement = A_displacement * sin(f * 2 * pi * t)
velocity = A_displacement * f * 2 * pi * cos(f * 2 * pi * t)
prev_displacement = self.getInput("displacement").getValue()
delta_displacement = displacement - prev_displacement

# Compute the velocity change
velocity_change = delta_displacement / dt
self.getOutput("velocity").setValue(velocity_change)
self.getOutput("displacement").setValue(displacement)

pass

def post_step(self):
tm = self.getInput('tm').value
ori = tm.getRotation()
zaxis = VxVector3(0, 0, 1)
dp = ori[2].dot(zaxis)
if ori[2].y > 0:
ang = -acos(dp)
else:
ang = acos(dp)

self.getOutput('theta').value = ang
pass

def on_state_restore(self, data):
pass

def on_state_save(self, data):

pass
```