DIPLOMADOLGOZAT

Szabó Bence

2025

Magyar Agrár- és Élettudományi Egyetem

Szent István Campus

Műszaki Intézet

Gépészmérnök mesterképzési szak

Szemcsehalmazok okozta kopás modellezése

Belső konzulens:	Prof. Dr. Keppler István Egyetemi tanár
Belső konzulens intézete/tanszéke:	Műszaki Intézet Gépszerkezettani Tanszék
Külső konzulens:	Dr. Habil Safranyik Ferenc Tudományos munkatárs
Készítette:	**Szabó Bence**

Gödöllő
2025

TARTALOMJEGYZÉK

1 BEVEZETÉS

1.1. A választott témakör ismertetése

Az űrkutatás az emberiség történetében mindig is extrém kihívásnak bizonyult, az univerzum szinte felfoghatatlan összetettsége miatt. A kutatások során folyamatosan új és eddig ismeretlen műszaki és egyéb problémákat kell a szakembereknek megoldani. A földön kívül elvégzett missziók során számos olyan meghibásodást, károsodást tapasztaltak a kutatók az egyes ott alkalmazott eszközökön, amelyek a földi viszonylatban egyáltalán nem, vagy nem olyan erős mértékben jelentek meg. A legszemléletesebb példák erre a Hold és Mars missziók során a meghibásodott, vagy funkciójukat részben, vagy egészben elvesztő műszaki berendezések, űrjárművek az erősen koptató környezet következtében. Rankin és munkatársai (2020) cikkükben leírják a Marson küldetést teljesítő „Curiosity" űrjáró meghibásodásait, amelyek többek között a marsi por erősen koptató közegére vezethető vissza. A legsúlyosabb problémák a jármű kerekeivel adódtak, amelyek nagy mértékben elkoptak, roncsolódtak a felszín durva hatásai miatt, de ezeken kívül számos további hajtáselemet érintő problémát is feltártak, amelyek ugyancsak a marsi por hatására vezethetők vissza. Sukumaran és munkatársai (2023) kutatásukban a Hold porának erős abrazív hatását is vizsgálták, amelyre egyik példa az Apollo program során alkalmazott űr szkafandereken keletkezett kopások majd az ebből következő meghibásodások. A földön kívüli talajok vizsgálatával számos további tanulmány foglalkozik, továbbá a Magyar Agrár- és Élettudományi Egyetemen is zajlanak kutatások a témában, valamint korábbi szakdolgozat projektem kapcsán szereztem némi tapasztalatot a numerikus szimulációs módszerek alkalmazhatóságával kapcsolatban, így ezen megfontolások mentén esett a választásom jelen témakörre.

1.2. Célkitűzések

Dolgozatomban azt a célt tűztem ki, hogy létrehozzak egy háromtest – ábráziós kopás vizsgálatot diszkrét elemes szimulációs módszer segítségével, mégpedig egy ingyenesen elérhető, nyílt forrás kódú szoftver a Yade Dem alkalmazásával. A szimuláció elvi alapjául a laboratóriumi körülmények között alkalmazott Pin-on-disk módszert választottam, azonban úgy módosítom, hogy koptató anyagot adagolok a forgó tárcsára, amely így megfelel a háromtest – ábráziós kopási modellnek. Ezek alapján elkészítek egy folyamatosan állandó sebességgel mozgó felületet, amely a koptató korongot hivatott megvalósítani és erre a mozgó felületre fogom rászórni a koptató anyagot, jelen esetben a marsi talaj egy általam egyszerűsített és elkészített modelljét, majd erre a szállított koptatóanyag folyamra fogom gravitációs módon

ráejteni a koptatandó testet. Megfelelő számú kontakt elérése esetén a test és a talaj között pedig további összeszorító erőt fogok alkalmazni, amely így biztosítja az állandó megfelelő nagyságú és irányú normális irányú erőt, amely a súrlódáshoz elengedhetetlen. A marsi port oly módon készítem el, hogy a szoftverben elérhető elemi gömb részecskéket összeragasztom, így kialakítva több szemcse típust, amelyekből aztán meghatározott összetételű talaj keverékeket készítek. Létrehozok egy további szimulációt is, amellyel az ily módon kialakított koptató közeg belső súrlódási kúpszögét tudom vizsgálni, majd a szimulációk eredménye alapján össze tudom hasonlítani a valós minták jellemző értékeivel. A kapott vizsgálati eredmények alapján pedig be fogom mutatni, hogy a különböző összetételű abrazív közegek hogyan befolyásolják a koptatás intenzitását.

2 SZAKIRODALMI ÁTTEKINTÉS

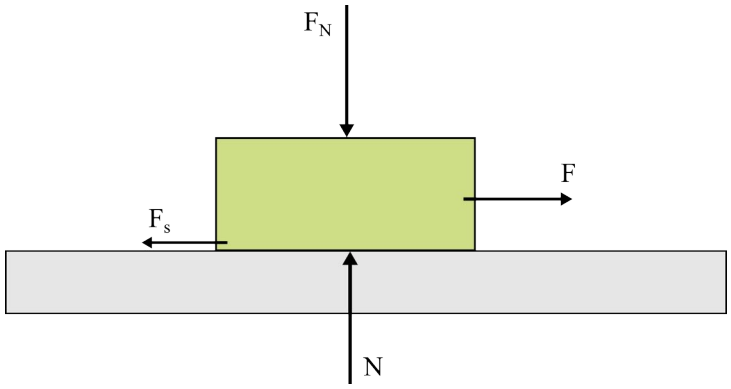
Ebben a fejezetben részletesen feltárom és bemutatom a hazai és nemzetközi szakirodalmakban fellelhető információkat a kopás és az azzal összefüggésbe hozható releváns tudományterületek kapcsán.

2.1 A kopási mechanizmusok

2.1.1 Súrlódás jelensége

Abban az esetben, ha két szilárd, sík felületet egymásra helyezünk azok között lokális érintkezési területek fognak kialakulni. Amennyiben az egyik felületet elmozdítjuk a másikhoz képest, úgy a másik felületen olyan erők ébrednek, amelyek a mozgást akadályozni kívánják. Ezt a jelenséget súrlódásnak az ébredő erőt pedig súrlódási erőnek nevezzük (Hutchings - Shipway, 2017). A gyakorlatban jellemzően szilárd testek felületei érintkeznek egymással, amelyre egy egyszerű példát az 1.ábra mutat, ahol az F_N [N] normál irányú erő az, amely a zöld színnel jelölt testet a szürkére színezett felülethez nyomja, N [N] az az ellenerő, amelyet a felület kifejt a testre, másként megfogalmazva F_N ellenereje, F [N] nagyságú erőt fejtünk ki a testre, F_s [N] pedig az az erő, amely a test elmozdulását kívánja megakadályozni.

1. ábra: Súrlódás állapota két test érintkezése esetén
(Forrás: Saját kép)



Abban az esetben, ha F erő nagysága megegyezik F_s nagyságával, úgy a test nyugalomban marad, ezt statikus súrlódási erőnek nevezzük, amely tipikusan F_{s0} jelöléssel adható meg. Tovább növelve F nagyságát, a test elkezd csúszni a felületen ebben az esetben a mozgást akadályozó erőt dinamikus, vagy mozgási súrlódási erőnek nevezzük, jelölése F_s. Az érintkezési felületen ható normál irányú (F_N) és a mozgás megindításához, majd fenntartásához szükséges $(F_{s0}$ *és* $F_s)$ erők közötti összefüggést a klasszikus Coulomb – egyenlet írja le a következő alakban:

(1)

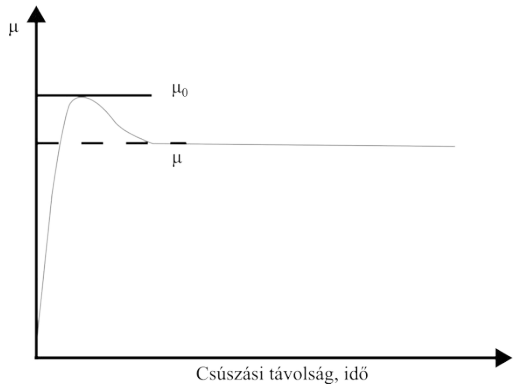
$$\mu = \frac{F_s}{F_N} \qquad \text{valamint} \qquad \mu_0 = \frac{F_{s0}}{F_N}$$

Ezek alapján megállapítható, hogy az erők között a μ [-] súrlódási együttható teremt kapcsolatot, amelyre az alábbi megállapítások tehetők (Fazekas és munkatársai, 2018; Al-Samarai - Al-Douri, 2024; Hutchings – Shipway, 2017):

1. A súrlódási erő arányos a felületeket összeszorító terheléssel, egyenlettel kifejezve:
$$F_s = \mu \cdot F_N \tag{2}$$
2. A súrlódási erő független az érintkező felületek nagyságától, továbbá nem függ a test csúszási sebességétől.
3. A súrlódási tényező függ azonban az érintkező felületek minőségétől és az anyagminőségtől.
4. Megállapítható, hogy a statikus, vagy nyugvó súrlódási tényező nagyobb, mint a dinamikus, vagy mozgásbéli súrlódási tényező.

A 2 ábra szemlélteti a súrlódási tényező értékének alakulását a csúszási távolság, vagy az idő függvényében.

2. ábra: Súrlódási tényező alakulása
(Forrás: Al-Samarai - Al-Douri, 2024)



A csúszási súrlódáson kívül megkülönböztetünk további súrlódási módokat, amelyek közül érdemes megemlíteni a gördülési súrlódást, amely egy felület és egy gördülő test érintkezési felületén alakul ki, tipikusan csapágyazások esetén jellemző, azonban mértéke jelentősen kisebb a csúszási súrlódásétól (Al-Samarai - Al-Douri, 2024; Hutchings – Shipway, 2017), valamint a fúró és ütközésszerű súrlódási módok sem relevánsak a dolgozatom szempontjából, így ezek részletes ismertetésébe nem megyek bele.

Egymással érintkező felületek között több különböző súrlódási állapot is felléphet, amelyek az alábbiak szerint csoportosítható (Dreyer - Gergye 2012; Hutchings – Shipway, 2017):

- Száraz súrlódás esetén az érintkező felületek egymáson kenőanyag nélkül mozdulnak el.
- Határréteg súrlódásról beszélünk, amikor vékony filmréteg tapad meg a felületeken, amely így csökkenti a súrlódási tényező értékét.
- Folyadék súrlódás akkor alakul ki, amikor a felületek között összefüggő folyadék, vagy gáz réteg alakul ki, amely felveszi az ébredő terheléseket. Ebben az esetben a két felület nem érintkezik egymással, a súrlódási tényező értékét a kenő közeg belső súrlódása határozza meg. A belső súrlódás fogalma részletesen a 2.4. fejezetben kerül ismertetésre.
- Vegyes súrlódás esetén a felületek között van kenőréteg, azonban ennek ellenére kialakul a részleges kontakt a két test között.

A mérnöki gyakorlatban a súrlódás, mint mechanikai folyamat tehát megkerülhetetlen, továbbá számos esetben nem kívánt károsodásokat képes okozni.

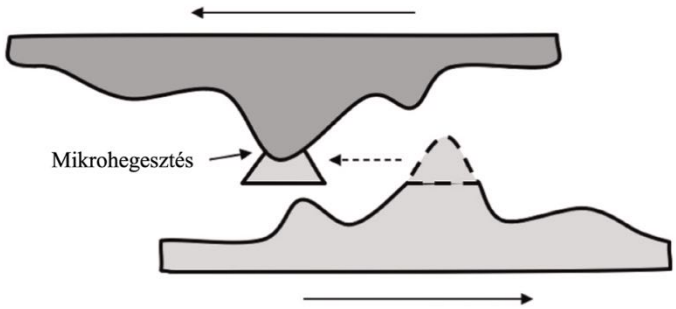
2.1.2 Kopás

Az előző fejezetben ismertetettek alapján két szilárd felület érintkezése és egymáshoz képest vett relatív elmozdulása esetén anyagveszteség lép fel, amelyet kopásnak nevezünk (Fazekas és munkatársai, 2018; Dreyer – Gergye, 2012; Al-Samarai - Al-Douri, 2024). Három fő kategóriába sorolhatóak a jellemző kopásmechanizmusok. Mechanikai folyamatról beszélünk akkor, amikor a felületek elcsúsznak egymáson, eközben pedig olyan terhelések ébrednek, amelyek plasztikus, azaz maradó alakváltozással járó vagy éppen ellenkezőleg, rideg töréses felületi sérüléseket okoznak, amely így anyag elhordást, ebből következően pedig kopást eredményez. Ezzel összefüggésbe hozható a második fő csoport, amely a termikus hatások okozta anyagvesztést foglalja magába. A felületek egymáson való elmozdulása minden esetben hőtermeléssel jár, amely hő a kölcsönhatásban résztvevő testek hőmérsékletét növeli. Vizsgálva a felületek hőmérsékletét az tapasztalható, hogy a változás mértéke olyan nagy lehet, amely befolyásolja az anyagok mechanikai tulajdonságait, mint például a folyáshatár, szakítószilárdság, vagy keménység értékét. Végül pedig külön kategóriába sorolhatók a kémiai kopásmechanizmusok, amelyek során a súrlódó felületek, anyagok a környezettel kölcsönhatásba lépve olyan szerkezeti változásokat szenvednek, amely szintén anyag eltávolításhoz vezet. Ilyen reakció lehet például felületek oxidációja, amely negatívan befolyásolja a kopási folyamatokat (Dreyer – Gergye, 2012; Al-Samarai - Al-Douri, 2024). A valóságban ezen mechanizmusok egymástól nem egyértelműen elválaszthatók, bizonyos

mértékben mind megjelenik és befolyásolja az alkatrészek kopását. Ezen túlmenően a kopásformák további jellemző típusokba sorolhatók az alábbi csoportosítás szerint (Fazekas és munkatársai, 2018; Dreyer – Gergye, 2012; Al-Samarai - Al-Douri, 2024; Hutchings – Shipway, 2017; Swain és munkatársai, 2020):

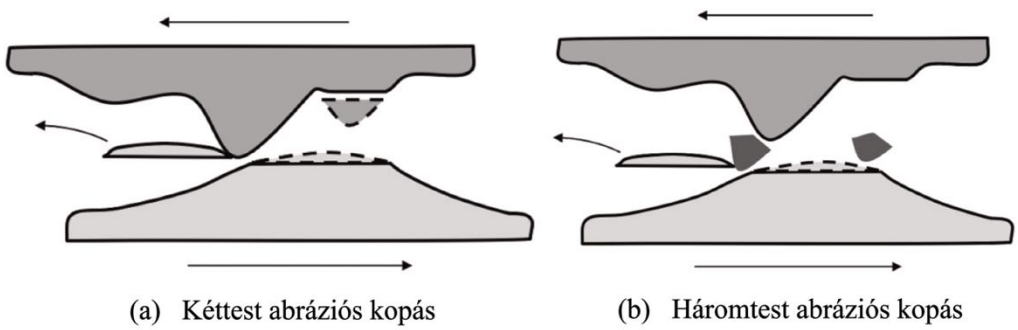
Adhéziós kopás: Tipikusan fémes felületek egymáson történő elcsúszása közben alakul ki, különösen akkor, amikor a két érintkező terület között nagy felületi nyomás alakul ki, azaz nagy terhelések esetén. Ahogyan ez a 3.ábrán látható ekkor az alkatrészeken az érdességből adódó felületi csúcsok képlékeny alakváltozást szenvednek, a keményebb anyagú komponens leszakítja ezeket a puhább testről, továbbá össze is tapadnak a levált részek, így kialakítva az anyag leválasztást és elhordást.

3. ábra: Adhéziós kopás folyamata
(Forrás: A la Monaca és munkatársai, 2021)



Abráziós kopás: Jellemzően két felület egymáson történő elcsúszása közben alakul ki az abrázió jelensége. Ebben az esetben a keményebb alapanyagú test felületi érdességből adódó kiálló érdes csúcsai karcolásokat, barázdákat hoznak létre a lágyabb anyagban. A leforgácsolt részecskék azonban nem tapadnak hozzá egyik testhez sem, hanem elhordásra kerülnek, ezt a folyamatot kéttest abráziós kopásnak nevezzük. Ahogyan azt a 4. (b) ábra is bemutatja létezik háromtest abráziós folyamat is, ebben az esetben apró, idegen részecskék kerülnek a két felület közé, felkeményednek, amelyek így, mint koptató anyag vesznek részt a kölcsönhatásban ezzel felerősítve a koptató hatás mértékét.

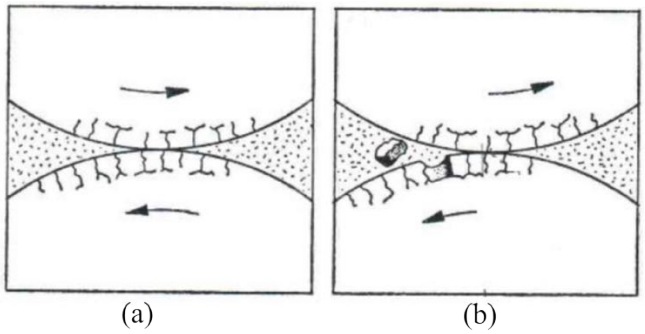
4. ábra: Abráziós kopás folyamatai
(Forrás: A la Monaca és munkatársai, 2021)



Legtipikusabb példa erre az esetre, amikor homok szemcsék kerülnek az alkatrészek közé, amelyek így jelentősen képesek növelni a nem kívánt kopás mértékét. Ebből kifolyólag könnyen belátható, hogy a mérnöki gyakorlatban az egyik leggyakrabban megjelenő kopásmechanizmus, ugyanis a gépészeti berendezések a legtöbb esetben poros környezetben dolgoznak. Dolgozatom szempontjából ugyancsak kiemelt fontosságú az abrazív kopás mechanizmusa, ugyanis az általam szimulálni kívánt eset tipikusan háromtest abráziós folyamat, ebből kifolyólag a 2.1.3.fejezetben részletesen ismertetem a vonatkozó mechanizmusokat, összefüggéseket.

Fáradásos kopás (Pitting): Ebben az esetben az érintkező felületek, folyamatos, ismétlődő igénybevételnek vannak kitéve, amely terhelés kifárasztja az alkatrészek külső rétegeit, mikrorepedések jelennek meg, majd ennek következtében a felületi kipattogzás kezdődik. Leggyakrabban gördülő felületek esetén jelentkezik ez a típusú kopás, ahogyan azt a 5.ábra is szemlélteti.

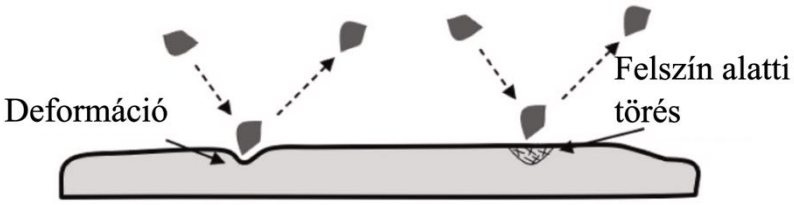
5. ábra: Fáradásos kopás folyamata (a) mikrorepedések kialakulása, (b) felület kipattogzása
(Forrás: Fazekas és munkatársai, 2018)



A mérnöki gyakorlatban tipikusan például fogaskerekek fogfelületén, gördülő csapágyak felületein alakul ki.

Eróziós kopás: Ilyen típusú kopás elsősorban abban az esetben fordul elő, amikor egy szilárd alkatrész áramló közegben például szennyvíz, poros gázok, homok működik, ilyenkor a kontinuum részecskéi nekiütköznek a szilárd felületnek, amelyen így apró repedések keletkeznek, majd onnan részecskék töredeznek ki, amely folyamatot a 6.ábra szemlélteti.

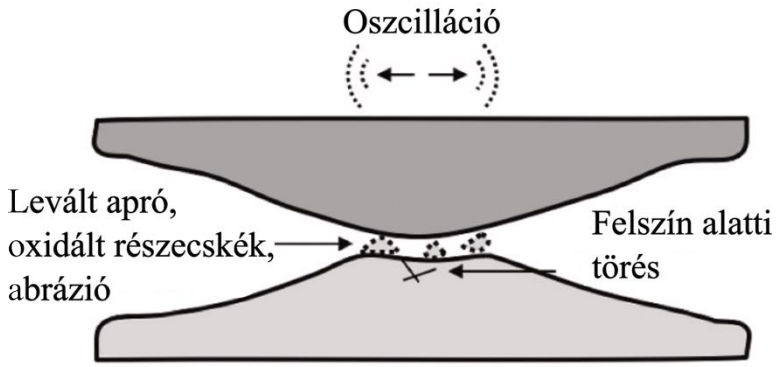
6. ábra: Az eróziós kopás folyamata
(Forrás: A la Monaca és munkatársai, 2021)



Az eróziós kopás a gépészetben tipikusan például gázturbina lapátok felületén alakul ki.

Fretting kopás: A kopási folyamatok egy speciális esete, ugyanis az érintkező felületek ezesetben nem mozdulnak el egymáson, azonban az alkatrészekben a terhelések hatására rezgések alakulnak ki, melynek jellemző frekvencia tartománya 5 – 100 Hz, amplitúdója 1 – 300 μm. Ahogyan azt a 7.ábra bemutatja, az oszcilláló mozgás hatására a felületből apró részecskék töredeznek ki, amelyek aztán oxidálódnak, így tovább erősítve a kopás folyamatát. A gépészetben jellemzően az illesztett kapcsolatoknál, mint például tengely – csapágy gyűrű alakul ki.

7. ábra: Fretting kopási mechanizmus
(Forrás: A la Monaca és munkatársai, 2021)



Az ismertetett kopási folyamtok mellett további típusok is felsorolhatóak, viszont azok a dolgozatom szempontjából kevésbé relevánsak, így ezeket nem részletezem.

2.1.3 Archard kopási törvény

Az előző fejezetekben bemutatottak alapján tehát kimondható, a kopás legalább annyira függ a csúszási körülményektől például normál irányú terhelés, csúszási sebesség, mint a kölcsönhatásban résztvevő anyagok tulajdonságaitól. Az első kopással kapcsolatos összefüggéseket Holm 1946-ban fogalmazta meg. Megállapítása szerint a kopás során eltávolított anyag térfogata és a csúszási távolság között kapcsolat áll fenn, amelyet a tényleges érintkezési felület nagyságához kapcsolt. Később Archard 1953-ban megalkotta a kopás egyenletét, amely kimondja, hogy az elkoptatott anyag térfogata egyenesen arányos a felületre ható normális irányú terheléssel és a csúszási távolsággal, valamint fordítottan arányos a kopó felület keménységével, az arányossági tényező pedig a kopás jellemző mechanizmusától függően kerül meghatározásra.

Ezek alapján tehát csúszó testekre vonatkozóan, az elkopott anyag mennyisége a 3. összefüggésben megadottak alapján számítható (Hutchings és munkatársai, 2017; Lyu és munkatársai, 2025; Zmitrowicz, 2006).

$$V = k\frac{F_N \cdot s}{H} \tag{3}$$

ahol:

V – az elkopott anyag térfogata [m^3]

k – kopási együttható [-]

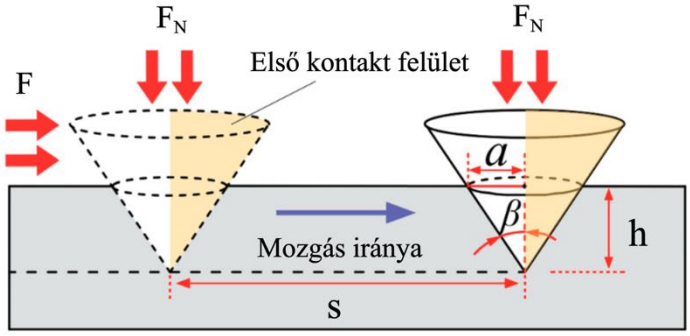
F_N – a felület normál irányú terhelése [N]

s – csúszási távolság [m]

H – a kölcsönhatásban résztvevő puhább anyag keménysége [MPa]

Az összefüggés egyszerűsége jól megmutatja azokat a tényezőket, amelyek ténylegesen befolyásolják a kopási folyamatokat, amely elsősorban fémekre lett megalkotva. A 8.ábrán jól látható, hogy koptató részecskék esetén hogyan és miből adódnak a számításhoz szükséges értékek, *h* a részecskék által okozott bemarás mélységét jelöli, amely az elkopott térfogatból *(V)* és a kopásban résztvevő felület nagyságából számítható.

8. ábra: Abrazív kopás számítási modellje
(Forrás: Lyu és munkatársai, 2025)



Az Archard-féle alap összefüggésen túl számos további kiegészítés, módozat létezik a kopás mértékének meghatározására, amelyek figyelembe veszik például a hőmérsékletváltozás okozta hatásokat, azonban dolgozatom esetén az ismertetett módszert fogom a szimulációba integrálni.

2.2 Laboratóriumi vizsgálatok a kopás megállapítására

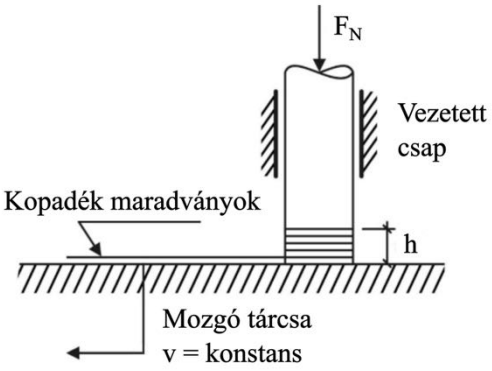
Az elméleti összefüggések validálása valós fizikai tesztek, mérések segítségével lehetséges. Ezen vizsgálatok különösen fontosak, ugyanis laboratóriumi körülmények között minden

esetben garantálható a környezeti változók, valamint a beállítási paraméterek állandósága, így biztosítva egzakt összehasonlíthatóságot az egyes mérési eredmények között. A tesztek relatíve olcsón és gyorsan elvégezhetők, a szükséges berendezés és a módszer általában standard által definiált, ennek ellenére ezen módszerek nem alkalmasak teljes gépszerkezetek vizsgálatára. A koptatási vizsgálati módszerek jelentős részét már az 1998-as években kidolgozták és standardizálták. Gee és Owen (1998) publikációjukban részletesen felsorolják a manapság is használatos eljárásokat.

2.2.1 Pin – on – Disk eljárás

Az egyik legelterjedtebben alkalmazott módszer, az úgynevezett Pin – on – Disk eljárás, amely során egy próbatestet nyomunk egy állansó sebességgel forgó tárcsa felületére a 9.ábrának megfelelő elrendezésben, majd az így kialakult súrlódási viszonyok következtében bekövetkezik a kopás, végeredményként pedig vizsgálható a próbatest, és/vagy a tárcsa kopásának mértéke. A módszer beállítási és működési paramétereit, valamint az eredmények értékelésének metódusait részletesen az ASTM G-99 szabvány definiálja.

9. ábra: Pin - on - disk teszt apparátus, ahol: h – a próbatestből lekopott anyag magassága
(Forrás: Zmitrowicz, 2006)



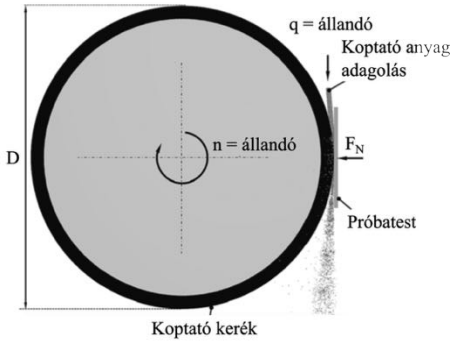
A pin – on – disk elsősorban a két test egymáson történő elcsúszásából adódó kopást vizsgálja, ebből kifolyólag fém – fém kontaktus esetén adhéziós kopás mérésére alkalmas ahogyan azt Lu és Yeh (2023) alkalmazta kutatásukban, amikor rozsdamentes acélok kopásállóságát hasonlították össze. Azonban a módszer alkalmas abráziós kopási vizsgálatok lefolytatására is mégpedig oly módon, hogy a hagyományos fém – fém anyagpárosítás helyett valamilyen abrazív koptató tárcsát alkalmazunk. Ez lehet például egy külön koptató réteg a tárcsa felületén, de előfordulhat olyan alkalmazásban is, amikor koptató szemcséket szórnak a tárcsa felületére így kialakítva a háromtest kopási modellt. Ramakrishnan Easwar (2020) kutatásában hamu és gránit koptatóanyag a tárcsa felületére történő adagolásával valósította meg a háromtest állapotot. Ezen tanulmány továbbá arra is rávilágít, hogy a két test súrlódási állapot tipikusan

nagyobb kopást eredményez a háromtest esetnél, ugyanis utóbbi esetén a koptató részecskék felhalmozódnak és eltartják egymástól a felületeket, így csökkentve a koptatás intenzitását.

2.2.2 „Dry sand/rubber wheel” eljárás

Magyar megfogalmazásban a száraz homok vagy gumikerék koptatási módszer során egy gumírozott felületű forgó tárcsa palást felületének nyomjuk neki a próbatestet, mindeközben egy folyamatos koptató anyag áramot juttatunk ezek érintkezési zónájába, így kialakítva a háromtest kopási állapotot, a 10.ábrán bemutatott elrendezésben.

10. ábra: "Dry sand/rubber wheel" teszt apparátus, ahol: D - a koptató kerék átmérője, n - a koptató kerék fordulatszáma, q - az adagolt koptatóanyag tömegárama
(Forrás: Grasser és munkatársai, 2024)

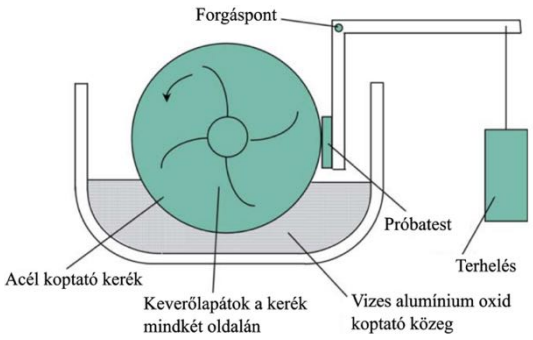


A tesztre vonatkozó beállítási, működtetési és értékelési módszereket ugyancsak szabvány írja elő, amely az ASTM G65.

2.2.3 NASA specifikus koptatási teszt

Az ismertetett két koptatási vizsgálaton kívül számos továbbit lehetne még felsorolni, amelyeknek az elve nagyon hasonló az előzőekben bemutatott típusokhoz, azonban ezeket jelen dolgozatban mélyebben nem ismertetem. Érdemes viszont megemlíteni egy specifikus még nem standardizált vizsgálati módszert, amelyet a NASA fejlesztett tovább egy meglévő szabványos metódus alapján, adaptálva azt a Holdon és a Marson megtalálható körülményekhez. A kiindulási teszt környezetet az ASTM B611 szabvány definiálja, amely áll egy koptatóanyag tároló edényből, aluminium oxid vizes keverékéből, amelyben egy lapátokkal ellátott acél keverő és koptató tárcsa forog. A koptatandó próbatest egy karos mechanizmus segítségével a koptató kerék palástjához van hozzányomva, így a normális irányú terhelés nagysága jól szabályozható és mindig azonos. A teszt berendezés fizikai megvalósítását a 11. ábra szemlélteti.

11. ábra: ASTM B611 koptatási teszt apparátus
(Forrás: Gant - Gee, 2006)



Ezt alapul véve Kobrick és munkatársai (2010) úgy módosította a módszert, hogy a koptató közeget nem keverték össze vízzel, az acél koptató kereket neoprene anyagúra cserélték és koptató anyagként marsi és hold talaj szimulánsokat használtak. A tesztek során megbízható eredményeket kaptak egy egyszerű, könnyen kezelhető módszerrel, amely jól alkalmazható az űrkutatásban.

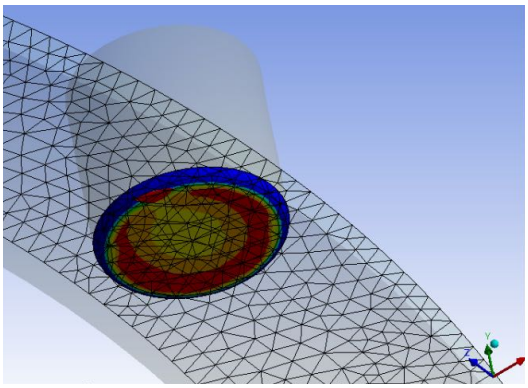
2.3 Kopásvizsgálatok numerikus módszerekkel

Bármennyire egyszerűek és könnyedén elvégezhetők a korábban bemutatott laboratóriumi koptatási tesztek, a szimulációs szoftverek fejlődésével és az elérhető számítási kapacitás növekedésével lehetőség van olyan összetett virtuális vizsgálatok elkészítésére, amelyekkel megfelelő keretrendszerben, jó megbízhatósággal, a valósággal összeegyeztethető eredményeket kaphatunk a szimuláció útján.

2.3.1 Kopás szimulációja végeselem analízis (FEM) alkalmazásával

Sueresh és munkatársai (2017) kutatásukban Ansys Workbench végeselem szoftver alkalmazásával vizsgálták a kopási mélység alakulását és az érintkezési felület nyomás eloszlását pin – on – disk teszt apparátus esetén, a szoftverben kopás mechanizmust a már korábban ismertetett Archard modell alapján határozták meg. A kapott eredményeket összevetették általuk elvégzett valós laboratóriumi értékekkel, amelyek alapján megállították, hogy a szimulációs modell és a valóság közel azonos eredményeket hozott, körülbelül 7% eltérést mértek. Továbbá azt is megállapították, hogy a kopás a próbatest éleinél kezdődik, majd halad tovább a belső keresztmetszet felé, amelyet a 12.ábra szemléltet.

12. ábra: Pin-on-disk FEM szimulációja
(Forrás: Sueresh és munkatársai, 2017)

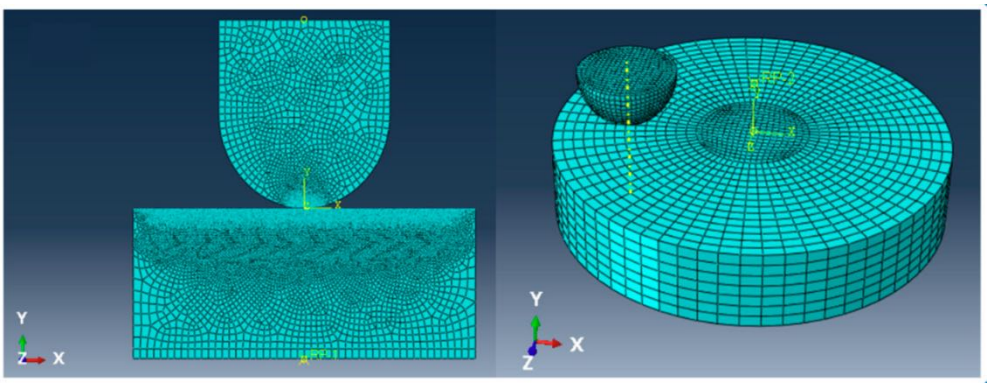


A módszer egyik limitációja, hogy a súrlódási együttható értékét a gyakorlatban meg kell mérni, csak ezután állítható be helyesen a szoftverben, ezek alapján tehát minden új anyagpárosítás esetén szükséges egy előzetes tapasztalati mérés az együttható mérésére vonatkozóan. Másik korlátozó tényező, amely a kapott eredmények helyességét is befolyásolja a testek hálózására vonatkozik. Minél nagyobb a hálózás „felbontása” tehát aminél kisebb elemi háromszögekre osztjuk fel a testeket annál pontosabb a kapott eredmény ugyanakkor szignifikánsan növeli a számítási kapacitás igényét, így az ilyen módszerrel elvégzett szimulációk egyik fontos lépése a hálózás optimalizálása.

Yan és munkatársai (2024) publikációjukban azt vizsgálták, hogyan alkalmazható a végeselem módszer különböző kopási problémák vizsgálatára, Abaqus szoftvert alkalmazásával. Elkészítettek egy virtuális Pin – on – disk teszt apartátust, amely esetén két különböző módon határozták meg a kopási mechanizmust. Az egyik módszer a már ismertetett Archard összefüggés alapú, ezen kívül integrálták a szoftverbe az energia disszipáció elvén működő módozatot is. Az energia disszipáció formula a súrlódás következtében fellépő energiákból számítja az elkopott anyagmennyiséget, amely képes kezelni például a hőmérséklet növekedés

okozta hatásokat is, így sokkal pontosabb végeredményt biztosíthat. A szimulációs környezet kialakítását a 13.ábra szemlélteti.

13. ábra: Pin -on -disk kopásvizsgálat FEM szimulációja
(Forrás: Yan és munkatársai, 2024)



Ezenkívül számos további alkalmazási lehetőséget mutatnak be kutatásukban, mint például fogaskerekek fogprofiljának kopásvizsgálata Archard modell alkalmazásával, ízületi protézisek elemzése. Megállapításuk a módszer alkalmazhatóságával kapcsolatban, hogy rendkívül széleskörűen alkalmazható, ugyanakkor jelenleg még nem képes teljes mértékben felváltani a laboratóriumi teszteket. Napjainkban még kutatási fázisban van a végeselem szimulációval végrehajtott kopás vizsgálat.

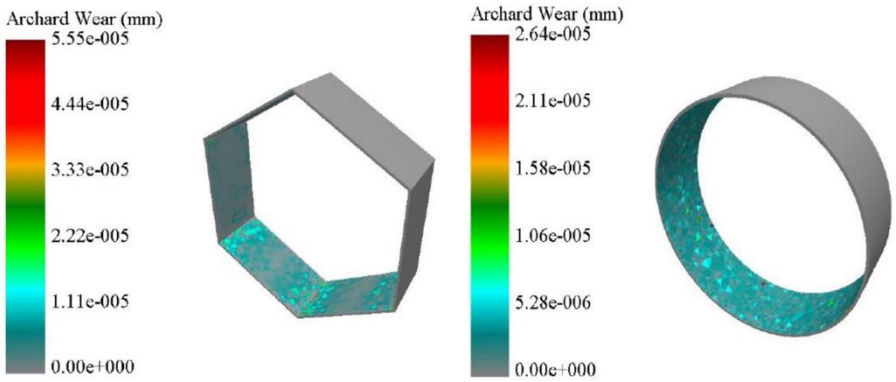
2.3.2 Kopás szimulációja diszkrét elemes szimuláció (DEM) alkalmazásával

A diszkrét elemes módszer kifejezetten szemcsés anyagok viselkedésének vizsgálatára alkalmas, ebből kifolyólag számos kutatásban alkalmazzák a módszert koptatás szimulációjára is. A legtipikusabb alkalmazási területei a talajművelő eszközök, földmunkagépek talajjal érintkező kopó alkatrészeinek, bányászatban használatos golyós malmok belső felületének elemzése geometriai kialakítás és kopás szempontjából. Az iparban több szoftver is elérhető, amelyek rendelkeznek beépített kopási modellekkel, mint például EDEM, Rocky DEM, azonban ezen szoftverek rendkívül drágák. További lehetőség olyan nyílt forráskódú szoftver alkalmazása, amelyek nem rendelkeznek ilyen beépített funkcióval azonban szabadon fejleszthető és integrálhatók új funkciók az alap programba ilyenek például a Yade DEM, LIGGGHTS.

Kimani és munkatársai (2024) bányászati golyósmalmok belső kialakításának kopástani hatásait modellezték kutatásukban EDEM szoftver segítségével, a kopási mechanizmus a szoftverben elérhető Archard összefüggésén alapult. Koptatóanyagként elemi szemcsékből összeragasztott clumpokat használtak, amellyel kvarc szemcséket modellezték, az így megalkotott szemcsehalmaz rézsűszögét lemérték a valóságban és összevetették a szimulált

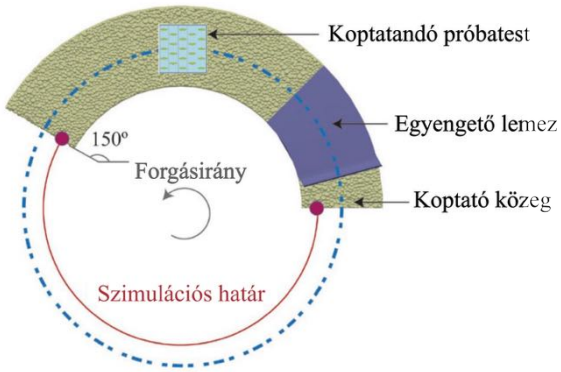
értékekkel ezáltal körülbelül 2,5%-os eltérést kaptak a két érték között. A szemcsés anyagokra vonatkozó rézsűszög és annak mérési lehetőségeit részletesen a 2.4. fejezetben fogom kifejteni. A szimulációt megelőzően elvégezték a kopási vizsgálatot egy valós kicsinyített golyósmalom tesztberendezésen, majd lefuttatták a teszteket a virtuális környezetben is. Ahogyan az a 14.ábrán látható két különböző geometriájú dob kialakítást teszteltek, melynek eredményeként azt kapták, hogy a henger alakú dob sokkal egyenletesebb és sokkal kisebb mértékű kopást szenved el, mint a hatszögletű profil, ahol a sarkok környezetében jelentősebb az anyagvesztés, mint a felületek többi részén. Ez pedig a két alakzatban kialakuló különböző anyagáramnak köszönhető. A valós és a szimulált kopási eredmények között körülbelül 3%-os eltérét kaptak, amely igazán jónak mondható.

14. ábra: Bányászati malom dob profil kialakításának DEM szimulációja
(Forrás: Kimani és munkatársai, 2024)



Yan és munkatársai (2022) kutatásukban szintén EDEM szoftverrel vizsgálták sima és domború felületek közötti kopás különbség alakulását, Archard modell alkalmazásával. Ehhez egy kaviccsal megtöltött körpályát készítettek, amelynek felső felületén körbehúzták az elkoptatandó különböző kialakítású testeket, ahogyan azt a 15.ábra szemlélteti.

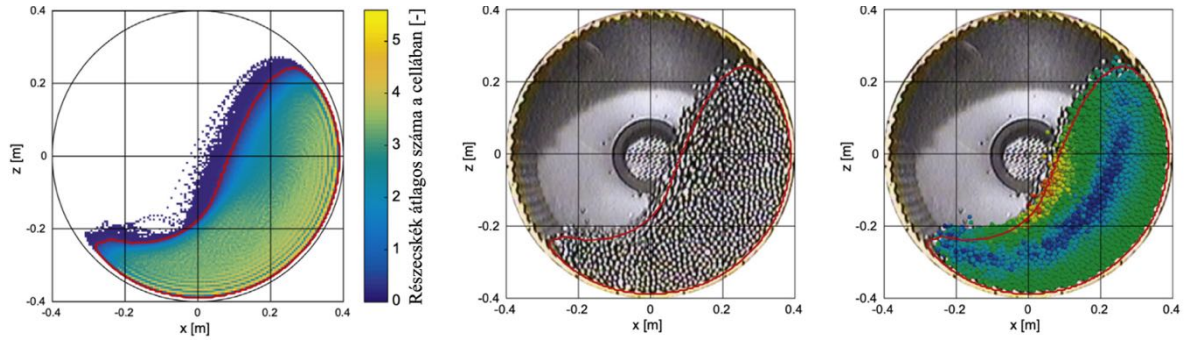
15. ábra: Kavicságyas koptató vizsgálat DEM szimulációja
(Forrás: Yan és munkatársai, 2022)



Ahhoz, hogy a szimuláció futtatási, számítási igényét lecsökkentsék a virtuális térben mindössze egy 150°-os szeletet modelleztek a teljes körpálya helyett. Kísérletük eredményéül azt kapták, hogy sima felület esetén körülbelül 6%-os eltérést tapasztaltak a valóság és a szimuláció között, azonban a domború felületű minta esetén körülbelül 25% volt a különbség, amely különbség feltehetően a kopási modell korlátaira és szimuláció során alkalmazott egyszerű elemi gömb részecskékre vezethető vissza.

Boemer és Ponthot (2017) bányászati golyósmalom dob belső felületének kopását vizsgálta Yade DEM szoftver segítségével. Több különböző kopási modellt integráltak a szoftverbe és külön-külön tesztelték azokat. A malom belső geometriáját elemi háromszögekből álló felületből alakították ki. Az így kialakított alakzat deformációját oly módon követték nyomon, hogy az elemi háromszögek helyzetét módosították a számított kopás mélységéből. Különös figyelmet kellett fordítania a csatlakozó sarokpontok újra generálására és a csatlakozó felületek frissítésére, ezen kívül simítást is kellett alkalmazni a felületek esetén, amely így a valósághoz hasonló kopási képet eredményezett. Az általuk megalkotott teszt környezet virtuális, valóságos és hibrid kialakításait a 16.ábra mutatja be.

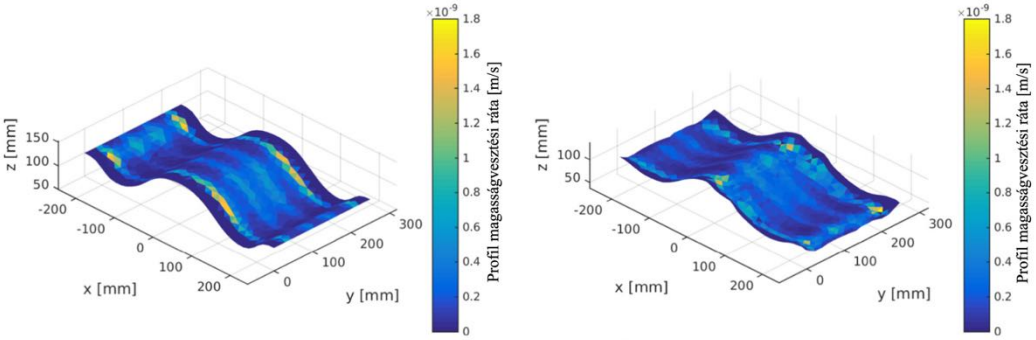
16. ábra: Bányászati golyósmalom belső dob lemez vizsgálata DEM módszerrel.
Balra: szimuláció részecskesűrűség diagramja. *Középen*: Fénykép a malomban zajló folyamatról. *Jobbra:* A szimuláció és a fénykép szuperpozíciója.
(Forrás: Boemer - Ponthot, 2017)



A valós anyagvesztési értékeket összevetve a különböző vizsgált kopási modellek által kapott eredményekkel azt kapták, hogy az úgynevezett tangenciális csillapítási energia disszipációs mechanizmus volt a legközelebb a valósághoz. A módszer lényege, hogy az energiavesztést a részecskék tangenciális irányú ütközési energiájával, csúszás nélkül számszerűsíti.

A malom dobjában a belső felületi szekciók kopási és geometria megváltozási értékeit háromdimenziós diagramban ábrázolták, amelyet a 17.ábra mutat be.

17. ábra: Kopás és geometria eloszlás diagramok
(Forrás: Boemer - Ponthot, 2017)

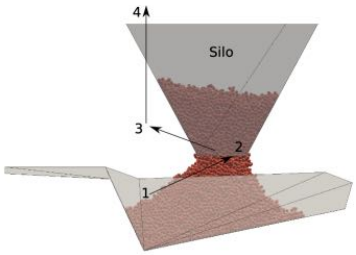


(a) Geometria és kopás eloszlás 2383 óra után.

(b) Geometria és kopás eloszlás 33043 óra után.

Rojas és munkatársai (2019) szintén Yade szoftver segítségével szimulálta bányászati dömper plató felületének rakodási és leürítési folyamat következtében kialakuló kopását, amelynek szimulációs megvalósítását a 18.ábra szemlélteti.

18. ábra: Bányászati dömper plató töltési (a) és ürítési (b) folyamata
(Forrás: Rojas és munkatársai, 2019)



(a) Plató megtöltési folyamata . A töltő siló a jelölt pozíciókat veszi fel a rakodás során.

(b) Plató ürítési folyamata.

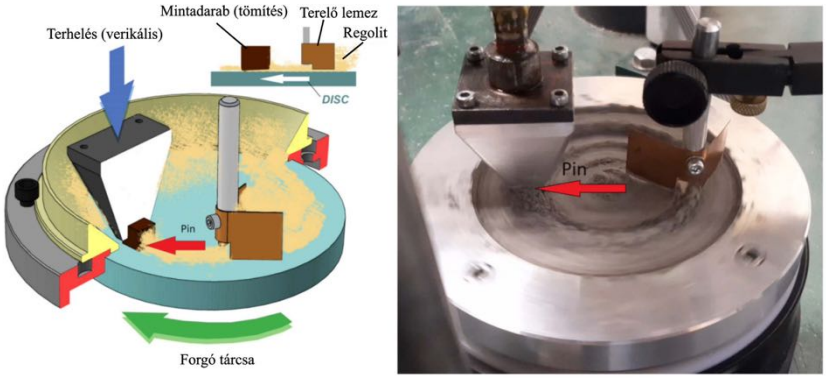
Vizsgálataikhoz az Archard kopási modellt használták, azonban azzal a kikötéssel, hogy a kopás mértékét relatív értékben adták meg nem pedig a képletben meghatározott ténylegesen elkopott térfogatban, ezzel a változtatással a H keménység és k kopási együttható értékei kiestek a képletből. Különböző részecskékből álló koptató anyaggal hajtották végre a teszteket. A koptató szemcsehalmaz minden esetben csak egy típusú részecskét tartalmazott, amelyek a következők voltak: egyszerű gömb, vagy három gömbből alkotott háromszög, vagy négy

gömbből alkotott gúla, vagy pedig tetraéder. Megállapították, hogy a kapott eredmények körülbelül 20%-os hibahatáron belül voltak, rakodáskor elsősorban a plató fülkéhez közeli része kopik, ürítéskor pedig a hátsó zóna. Továbbá azt is megállapították, hogy az alkalmazott szemcsék kialakítása erőteljesen befolyásolja a kopás mértékét, a legpontosabb eredményeket a tetraéder kialakítással kapták, ez egyszerű gömbös kialakítás túlságosan könnyedén képes elgördülni egymáson, így a kopás mértéke jelentősen lecsökken ilyen típusú koptatóanyag alkalmazása esetén.

2.4 Talajok a Hold és a Mars felszínéről

A földön kívüli talajokat a szakirodalmak legtöbbször regolitoknak nevezik, amely definíció szerint egy a szilárd halmazállapotú bolygók felszínét borító, összefüggő, laza szerkezetű geológiai anyag, amely magában foglalja az aprózódott kőzeteket, porokat, valamint a Földön a talajréteget, amelyet a gravitáció, szél, víz vagy a jég hordott el és rakott le (Huggett, 2023). Ahogyan azt a bevezetőben ismertettem a földön kívüli regolitok számos nehézséget okoznak az űrbéli bolygókon végzett tevékenységek során. Ezt alapulvéve Kalácska és munkatársai (2024) Pin – on – Disk koptatási módszerrel vizsgálták fém és politetrafluoretilén tömítések kopását, különböző regolit szimulánsok hozzá adagolásával. A teszt berendezés felépítését a 19.ábra szemlélteti, amelyen jól látható, hogy egy terelő lemezt alkalmaztak, amely mindig összeterelte a koptató anyagot pontosan a mintadarab nyomvonalára.

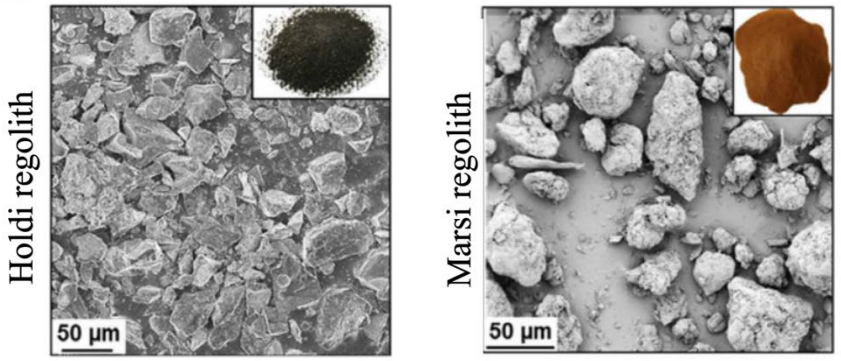
19. ábra: Acél - politetrafluoretilén Pin-on-Disk vizsgálata regolit hozzáadásával
(Forrás: Kalácska és munkatársai, 2024)



Eredményül azt kapták, hogy a holdi porok jelentősebb kopást eredményeztek, mint a marsi minták. Ez a tény arra vezethető vissza, hogy jelentős geometriai eltérés figyelhető meg a két égitest mintája között. A holdi regolit részecskéi éles szélekkel, sarkokkal rendelkeznek, ugyanis légkör hiányában nincs olyan külső hatás, amely folyamatosan mozgásban tartaná azokat, így nem gördülnek el egymáson, nem kerekítik le egymást, ezzel megtartva az aprózódás utáni éles kialakítást. Ezzel ellentétben a marson fellelhető földinél kisebb gravitáció

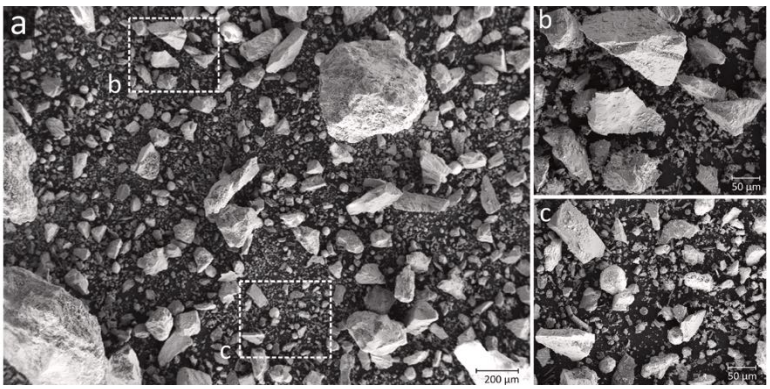
és a légkör miatt a részecskéket a szél folyamatosan mozgásban tartja, amelyek így bizonyos mértékben lekerekednek, így egy a holdinál kevésbé abraziv regolitoth képezve (Kalácska és munkatársai, 2024). A részecskék alakja rendkívül sokféle lehet, egy – egy jellemző kialakítást a 20.ábra szemlélteti.

20. ábra: Holdi és marsi talajok részecskéi
(Forrás: Jakus és munkatársai, 2017)



Az űrmissziók során mindkét égitestről gyűjtöttek talajmintákat különböző helyszínekről, amelyeket aztán a kutatók laboratóriumban elemeztek. Ezen információk alapján elkészítették a földön elérhető ásványokból az egyes talajminták helyettesítő szimulánsait, amelyeket egyedi jól beazonosítható névvel láttak el, mint például LHS-1, LMS-1 stb. holdi, vagy pedig JEZ-1, MGS-1 stb. marsi minták. Ezen egyedi talaj keverékek szabadon megvásárolhatóak. A marsi talajokat tovább elemezve azon belül is az MGS-1 szimulánst, amely az egyik legáltalánosabban alkalmazott talajminta, a pásztázó elektronmikroszkóppal készített felvételek alapján jól látható, hogy számos nagyobb éles ködarab is jelen van az egyébként jellemzően lekerekített sarkokkal rendelkező kis méretű szemcsék között, ahogyan azt a 21.ábra szemlélteti.

21. ábra: MGS-1 szimuláns szemcséi pásztázó elektronmikroszkóppal készített felvétel
(Forrás: Nababan és munkatársai, 2025)



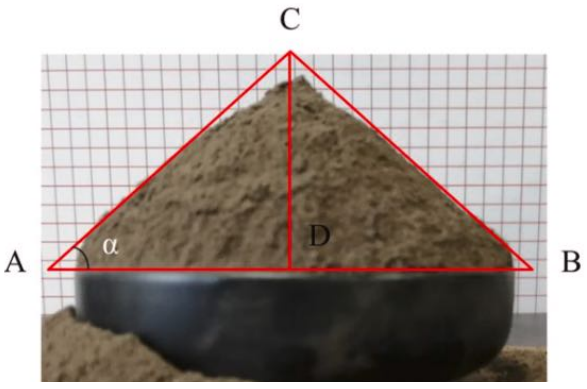
A variánsok szemcseméret eloszlásáról részletes információt tartalmaz azok hivatalos adatlapja, ezentúl számos további paramétert rögzít, mint például sűrűség, rézsűszög, amely értékeket az 1.táblázat foglal össze, ahol a D10, D30, D50 és D90 értékek a részecskék jellemző átmérőjét adja meg százalékos eloszlás szerint.

1. táblázat: MGS-1 marsi talaj szimuláns általános paraméterei
(Forrás: MGS-1 Spec Sheet 003-05-001-0523; Long-Fox - Britt, 2023)

MGS-1 marsi szimuláns tulajdonságai	
Sűrűség nem összenyomott állapotban:	1,29 g/cm^3
Átlagos részecske átmérő	90 µm
Medián részecske átmérő:	60 µm
D10	5,19 µm
D30	19,96 µm
D50	49,3 µm
D90	205,48 µm
Átlagos rézsűszög:	38,9°
Maximum rézsűszög:	43,6°

Annak érdekében, hogy a szimuláció során a koptató anyag paraméterei jó közelítéssel megegyezzen a valós talaj szimulánséval a legegyszerűbb módszer annak rézsűszögét megvizsgálni a valóságban és a szimulációban egyaránt. Halmozott szemcsés anyagok esetén a rézsűszög a legáltalánosabb definíció szerint az a maximális vízszintes síktól mért szög, amelyre az anyag halmozható a lejtő összeomlása nélkül, amelyet a 22.ábrán az α szög jelöl (Al-Hashemi - Al-Amoudi, 2018).

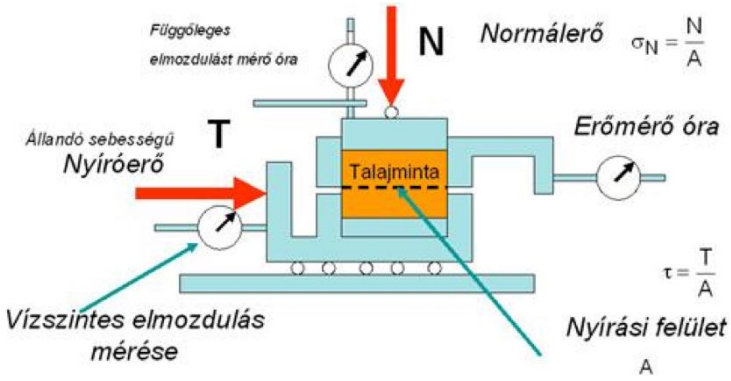
22. ábra: Rézsűszög laboratóriumi mérése
(Forrás: Zhu és munkatársai, 2021)



Szemcsés anyagok rézsűszöge összefüggésben áll az anyag statikus súrlódási együtthatójával és a belső súrlódási szöggel, abban az esetben, amikor a szemcsék bizonyos mértékben leülepedtek és a szemcseméret 5µm-nél nagyobb a rézsűszög és a belső súrlódási szög értéke néhány fok különbséget mutat. (Al-Hashemi - Al-Amoudi, 2018; Metcalf, 1966).

A belső súrlódási szög kapcsolatot teremt a szemcsés anyagok normál- és nyírófeszültsége között, amely értékek elsősorban laboratóriumi direkt nyíróvizsgálatokkal mérhetők a 23.ábrának megfelelően.

23. ábra: Direkt nyíróvizsgálat elvi vázlata
(Forrás: Faur – Szabó, 2011)



Fontos megemlíteni, hogy a szemcsék között csak akkor lép fel nyírási ellenállás, ha a súrlódó felületekre nullától nagyobb normális irányú erő hat. Ezek alapján tehát a talajok nyírószilárdsága a 4. összefüggés szerint az alábbi módon határozható meg (Radjai – Azéma, 2009):

$$\tau = c + \sigma \cdot tan\varphi \tag{4}$$

ahol:

τ - a talajban ébredő nyírófeszültség [Pa]

c – a talaj kohéziója [Pa]

σ - a talajra ható nyomófeszültség [Pa]

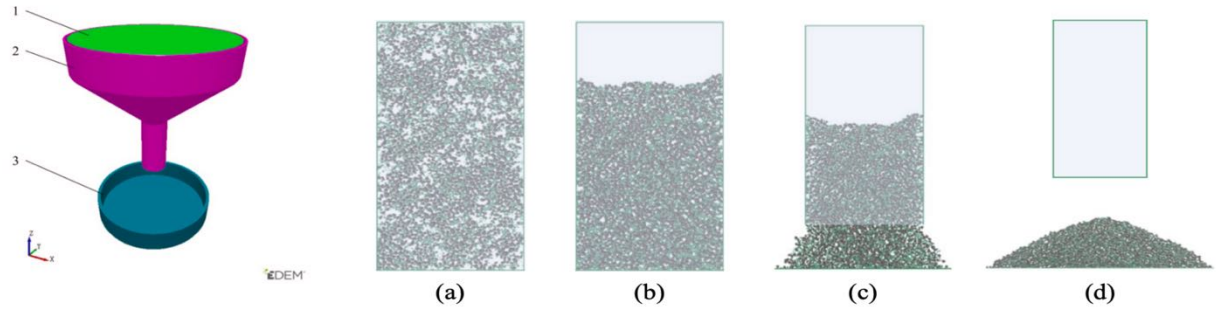
φ - a belső sűrlódási szög [°]

Annak érdekében, hogy a különböző talaj modelleket megfelelő módon lehessen szimulációs környezetben alkalmazni, szükséges tehát annak viselkedését megvizsgálni a valóságban és virtuális környezetben is.

Zhu és munkatársai (2021) és Zhou és munkatársai (2024) diszkrét elemes szimulációs módszer alkalmazásával különböző a valós holdi talajokat egyszerűsített módon leképező szemcsehalmazokat validált azok rézsűszögének vizsgálatával. Mindkét esetben elemi gömbök összeragasztásával alakítottak ki a valós szemcséket helyettesítő összetett részecskéket, amelyekkel aztán elvégezték a vizsgálatokat. Egyik esetben a tölcsérből történő kifolyási

módszert, másik esetben pedig a tölcsérből történő kifolyás, valamint a cső felemelés teszteket hajtották végre, elvi felépítésük a 24.ábra szemlélteti.

24. ábra: Tölcsérből történő kifolyás (balra) és cső felemelés (jobbra) tesztek DEM környezetben: 1-Szemcsék előállítása, 2-Tölcsér, 3-Tálca, (a) Szemcse előállítás, (b) Ülepítés, (c) Henger felemelés, (d) Rézsűszög kialakulása
(Forrás: Zhu és munkatársai, 2021; Zhou és munkatársai, 2024)



A vizsgálatok után azt kapták, hogy mindkét esetben 2 – 3 % hibahatár körül volt a szimulált tölcsérből történő kifolyás és a valós ugyanilyen módon végrehajtott laboratóriumi teszt eredménye, a cső felemelés módszerrel ennél valamivel nagyobb 6% körüli eltérést kaptak az ugyanilyen módon elvégzett laboratóriumi vizsgálathoz képest.

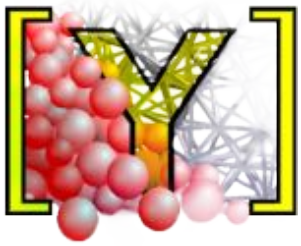
3 KOPÁS NUMERIKUS VIZSGÁLATA

Ebben a fejezetben részletesen ismertetem az alkalmazott szimulációs környezetet, amely két fő vizsgálatot foglal magában. Az egyik a modellezett talajszerkezetek rézsűszögének mérése, a másik pedig maga a koptatási szimuláció, amely előre meghatározott paraméterek megváltoztatásának koptatásra gyakorolt hatásának vizsgálata.

3.1 Yade – dem szimulációs környezet

Dolgozatomban a fejezet bevezetőjében is megfogalmazott problémák vizsgálatát a Yade DEM nyílt forráskódú szimulációs keretrendszerben végeztem el, amely szabadon bárki számára elérhető a https://yade-dem.org/doc/ webcímen érhető el.

25. ábra: Yade diszkrét elemes szimulációs szoftver
(Forrás:https://yade-dem.org/wiki/Logo)



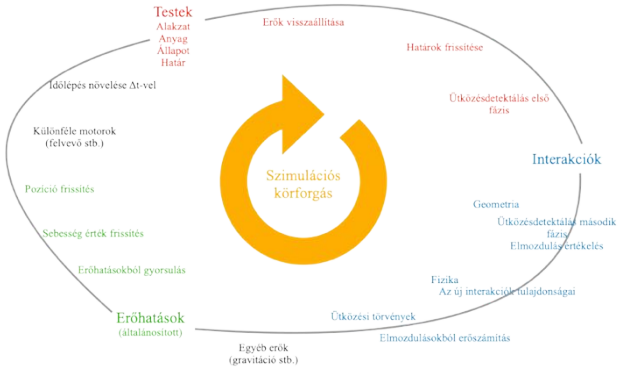
A program sajátossága, hogy a szimulációs környezete szabadon alakítható az igényeknek megfelelően programozás útján például Python kódrendszerben. Számos beépített funkció, modul elérhető a szoftverben, amelyekből némi programozási ismeret birtokában könnyedén felépíthetőek különböző virtuális vizsgálatok, ezen túlmenően lehetőség van szabadon egyedi funkciókat fejleszteni és beintegrálni a szoftverbe (Šmilauer és munkatársai, 2021). A Yade nem rendelkezik beépített kopás számítási modullal, így ezt a funkciót teljes egészében én fejlesztettem és integráltam a programba, amely a fejezet későbbi szakaszában kerül bemutatásra.

3.1.1 Yade működése

A Yade-ben és általában a diszkrét elemes szimulációs szoftverekben folyamatosan egy úgynevezett szimulációs ciklus fut, amely lépésről lépésre lefut és minden esetben elvégzi a lépéseket az alábbiak szerint. A futtatási sorrend tipikusan a részecskék szimulációba integrálásának sorrendjében zajlik, tehát elsőként az újonnan létrehozott elemek közötti ütközéseket vizsgálja meg a program, ezt követően meghatározza az új kölcsönhatásokat. Majd ezután vizsgálja a már kialakult interakciókat, elsőként a bekövetkező alakváltozásokat értékeli, majd ezen értékekből kiszámítja az ébredő feszültségeket, végül pedig aktiválja az erőhatásokat

a kölcsönhatásokban résztvevő részecskékre (Šmilauer és munkatársai, 2021). A körfolyamat során lezajló lépéseket részletesen a 26.ábra mutatja be.

26. ábra: Yade szimulációs ciklus, minden esetben a "Testek" pontból kiindulva
(Forrás: Šmilauer és munkatársai 2021)

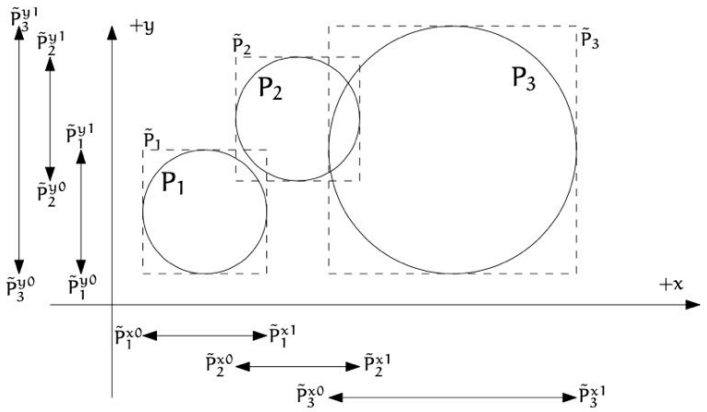


A szoftver az ütközések detektálását az úgynevezett sweep and prune algoritmus segítségével hajtja végre. Ennek a módszernek a lényege, hogy az egyes alakzatokat például részecske, felület, fal egy tengelyhez igazított határolókerettel (Aabb) veszi körül, amely legyen $\tilde{\mathrm{P}}_i$. A határolókeret megadható annak felső és alsó sarokpontjaival a térben ($\in \mathbb{R}^3$) ahol, $\tilde{P}_i^{x0}$, $\tilde{P}_i^{x1}$ a minimum és maximum koordinátái $\tilde{\mathrm{P}}_i$-nek at x tengely mentén és így tovább a többi tengelyre is. Két Aabb átfedése meghatározható az egyes különálló intervallumok átfedésének konjunkciójából a tengelyek mentén az alábbiak szerint ($w \in \{x, y, z\}$):

$$\left(P_i \cap P_j\right) \neq \emptyset \leftrightarrow \ \wedge \ \left[\left(\left(\tilde{P}_i^{w0},\ \tilde{P}_i^{w1}\right) \cap \left(\tilde{P}_j^{w0},\ \tilde{P}_j^{w1}\right)\right) \neq \emptyset\right] \tag{5}$$

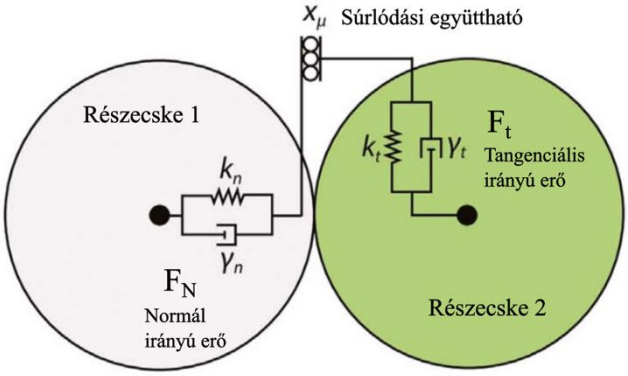
Az egyes határoló keretek térbeli átfedése akkor valósul meg, amikor azok minden tengely mentén metszik egymást. A 27.ábra alapján megállapítható, hogy $\left(\tilde{\mathrm{P}}_1 \cap \tilde{\mathrm{P}}_2\right) \neq \emptyset$ és $\left(\tilde{\mathrm{P}}_2 \cap \tilde{\mathrm{P}}_3\right) \neq \emptyset$azaz a határoló keretek átfedésben vannak, fontos azonban megjegyezni, hogy $(P_1 \cap P_2) = \emptyset$, tehát a P_1 és P_2 részecskék még nincsenek átfedésben (Šmilauer és munkatársai, 2021).

27. ábra: Átfedés érzékelés algoritmus kétdimenziós ábrázolása
(Forrás: Šmilauer és munkatársai 2021)



Miután megtörtént az ütközés érzékelése a különböző mozgásokat és kölcsönhatásokat, amelyek a részecskék között, vagy a részecskék és felületek között alakulnak ki kontakt törvények határozzák meg. Diszkrét elemes szimulációs környezetben az egyik legelterjedtebben alkalmazott a Hertz-Mindlin féle kontakt modell. Ennek lényege, hogy a kölcsönhatás során fellépő erőhatások két fő komponensre osztja, a normál és tangenciális irányokra, amelyet a 28.ábra szemléltet (Kotrocz – Kerényi, 2018; Zhao és munkatársai, 2024).

28. ábra: Két részecske kontaktja Hertz-Mindlin modell szeint
(Forrás: Capozzi és munkatársai 2018)



A kontakt során ébredő normál irányú erő az alábbi összefüggés segítségével számítható:

$$F_N = \frac{4}{3} \cdot E^* \cdot \sqrt{R^*} \cdot \delta_N^{\frac{3}{2}} \tag{6}$$

ahol:

F_N – a kontakt során ébredő normál irányú erő [N]

E^* - a rugalmassági modulus [MPa]

R^* - a részecskék sugara [mm]

δ_N – a kontakt során a részecskék átfedése [mm]

A tangenciális erő pedig az alábbi egyenlettel határozható meg:

$$F_t = -S_t \cdot \delta_t = -8 \cdot G^* \cdot \sqrt{R^* \cdot \delta_N} \cdot \delta_t \tag{7}$$

ahol:

F_t – a kontakt során ébredő tangenciális irányú erő [N]

G^* - a nyírási modulus [MPa]

δ_t – a kontakt során a részecskék átfedése [mm]

A tangenciális irányú erő nagysága nem lehet nagyobb, mint a normál irányú komponensé, a kettő erő egymástól való függőségét a korábban már ismertetett Coulumb súrlódási egyenlet írja le, miszerint:

$$F_t \leq F_N \cdot \mu_s \tag{8}$$

ahol:

μ_s – a súrlódási együttható [-]

A részecskék, valamint a részecskék és felületek között további kölcsönhatások ebből következően pedig erők ébrednek például a gravitáció következtében, amelyeket a szoftver hozzáad a korábban számolt értékekhez. A mozgásegyenleteket a Yade Leapfrog integrálás segítségével oldja meg. Az erők ismeretében már kiszámítható az egyes részecskék gyorsulása, amely a klasszikus Newton törvénnyel felírható az alábbiak szerint:

$$\ddot{u}^{\circ} = \frac{F}{m} \tag{9}$$

ahol:

$\ddot{u}^{\circ}$ - a részecske gyorsulása ebben az időpillanatban $\left[\frac{m}{s^2}\right]$

F – a szemcsére ható kontakt erő ebben az időpillanatban [N]

m – a részecske tömege [kg]

Véve az egyenlet Δt szerinti deriváltját meghatározható annak következő időpillanatbéli pozíciója. Egyszerű gömb részecskékre ugyancsak a Newtont törvényt alkalmazva megadható azok orientációja az alábbiak szerint:

$$\dot{\omega}_i^o = \frac{T_i}{I_{11}} \tag{10}$$

ahol:

$\dot{\omega}_i^o$ - szöggyorsulás ebben az időpillanatban $\left[\frac{rad}{s^2}\right]$

T_i – a nyomaték ebben az időpillanatban [Nm]

I_{11} – a tehetetlenségi nyomaték $[kg \cdot m^2]$

Több elemi gömb szemcséből alkotott úgynevezett clumpok esetén azonban ez az elemi módszer nem alkalmazható, ennél jóval összetettebb és számítási teljesítmény igényesebb módszerrel adható meg ezen szemcsék orientációja. Számos további mozgásegyenlettel kapcsolatos összefüggés megtalálható a Yade dokumentációjában.

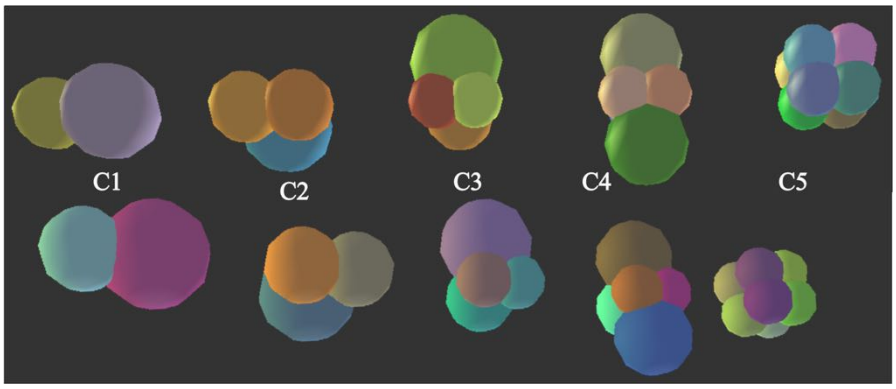
3.2 Rézsűszög numerikus vizsgálata

A fő koptatási szimulációt megelőzően először egy olyan virtuális környezetet állítottam össze, amely alkalmas a korábban ismertetett talaj rézsűszög vizsgálat elvégzésére. Ez a lépés azért fontos, mert így össze tudom vetni az általam lemodellezett talajminták szimulációs eredményeit a földön kívüli regolitiokkal elvégzett valós mérésekkel, ebből adódóan a későbbi koptatási teszteket a valósággal összehasonlított koptatóanyaggal tudom elvégezni. Fontos kiemelni, hogy jelen és a később bemutatásra kerülő kopási szimuláció alapvető paraméterei, mint például kontakt modell, koptató szemcsék kialakítása, jellemzői megegyeznek.

3.2.1 Regolit szemcsék kialakítása

Első lépésként elkészítettem azokat a változó szemcse alakzatokat, amelyeket felhasználva el tudtam készíteni a meghatározott koptató regolith keverékeket. Öt fő szemcseszerkezetet definiáltam az egészen egyszerű szinte gömbszerűtől az összetettebb, „éles” kialakításúig. Ahogyan azt a 29.ábra szemlélteti az egyes elemi gömbök egymásba érnek, így megalkotva az összetettebb alakzatokat.

29. ábra: Kialakított koptató szemcsék
(Forrás: Saját kép)



Minden egyes szemcse típushoz készítettem egy sablont, amely az egyes részecskék pontos helyét tartalmazza az alábbiak szerint:

c1 = [((0, 0, 0), 0.75), ((1.2, 0, 0), 1.0)]
c2 = [((0, 0, 0), 0.75), ((1.1, 0, 0), 0.75), ((0.55, 1.2, 0), 1.0)]
c3 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 1.0)]
c4 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 0.9), ((0.4, 0.3, -1.0), 0.9)]
c5 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0, 0.8, 0), 0.6), ((0.8, 0.8, 0), 0.6), ((0, 0, 0.8), 0.6), ((0.8, 0, 0.8), 0.6), ((0, 0.8, 0.8), 0.6), ((0.8, 0.8, 0.8), 0.6)]

Az egyes elemi gömbök középpontjának koordinátáit az x – y – z tengelyek mentén a belső zárójelben megadott értékek határozzák meg *((0, 0, 0), 0.75)*, a gömb sugarát pedig a külső zárójelben található érték definiálja. A szimulációban a generált szemcsék méretét nem az itt definiált érték adja, hanem egy könnyen megadható *rMin* és *rMax* változó. A későbbi kopás

vizsgálatot figyelembe véve jelen szimulációhoz is az *Ip2_FrictMat_FrictMat_MindlinPhys* kontakt modellt alkalmaztam, ez ugyanis kifejezetten alkalmas szemcsés anyagok vizsgálatára, továbbá képes kezelni azon paramétereket, mint például a normál irányú kontakt erő vagy az érintkezési távolság, amelyek az Archard kopási összefüggéshez szükségesek. A koptató szemcsék méretét a korábban ismertetett MGS-1 marsi talaj adatlapjában is közölt méret eloszlás alapján határoztam meg. A szemcsék méretét a 100 - 110 µm-es mérettartományban határoztam meg, amely érték a 90 µm-es átlagos szemcseméret felett van, azonban kedvezőbb a szimuláció erőforrás igény szempontjából. Szándékosan szűk mérettartományt határoztam meg, ezzel elkerülve a túl nagy és túl kicsi szemcsék okozta esetleges torzító hatásokat. A megadott méret a clumpok teljes közelítő méretét adja meg, ugyanis előállításuk során a program kiszámolja az egyes elemi gömbökből alkotott részecskék térfogatát, amelyből visszaszámolja a méretet oly módon, mintha az egy elemi gömb lenne, így jó közelítéssel megadva a teljes szemcse méretét.

Ezt követően megadtam az anyagjellemzőket mind a koptatóanyagra, valamint definiáltam két típusú acélt, egy hagyományos és egy alacsony súrlódási tényezővel rendelkezőt a rézsűszög méréshez szükséges objektumokhoz (Xia és munkatársai, 2019; Boemer, 2015). A beállított anyagjellemzők értékeit a 2. táblázat foglalja össsze.

2. táblázat: Anyagjellemzők értékei
(Forrás: MGS-1 Spec Sheet 003-05-001-0523; Morgan és munkatársai, 2018)

	Regolith paraméterek		Acél paraméterek	
	Irodalmi érték	Beállított	Irodalmi érték	Beállított
Young modulus [Mpa]	10 - 70	40	210 000	200
Sűrűség [kg/m3]	~1300	1300 / 2600	7850	7800
Poisson [-]	0,1 - 0,4	0,25	0,3	0,3
Belső súrlódási szög [°]	30 - 40	34 / 38 / 44	19 - 32	30 (10)

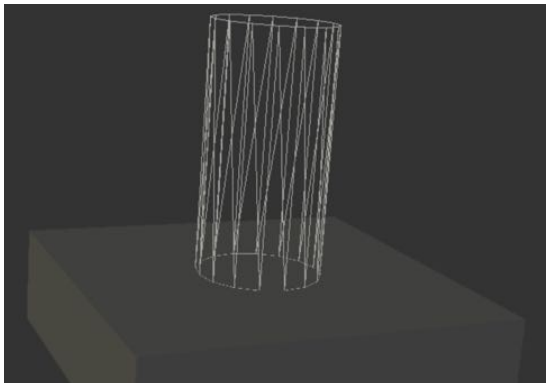
Az acél esetén a Young modulus értékét 210 GPa helyett 210 MPa értékre állítottam, ugyanis annak értéke nagymértékben befolyásolja a szimuláció időlépését, minél nagyobb a beállított érték az időlépés értéke annál kisebb, továbbá futtatási instabilitást okozhat. Az anyagtulajdonságok megadása a következő módon történt:

```
FrictMat(young=40e6, density=1300, poisson=.25, frictionAngle=math.radians(38), label='regolith'),
FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(30), label='steel')
FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(10), label='tube_low_friction')
```

3.2.2 Szimuláció alap felépítése

Az alap paraméterek definiálása után elkészítettem a vizsgálathoz szükséges alap geometriákat, úgymint, a mindkét végén nyitott csövet, amely összefogja a talajszemcséket és egy bázis testet, amelynek felső síkján történik a rézsűszög mérés kísérlet. A cső méretét az Altair weboldalán közzétett rézsűszög mérés javasolt beállításai alapján határoztam meg, amely az alábbi linken érhető el: Altair rézsűszög beállítási útmutató. Az ott leírt információk alapján a cső átmérőjének körülbelül a szemcseátmérő húszsorosának kell lennie, magasságának pedig ennek kétszeresének. Az általam választott szemcseátmérő tartomány 100 – 150 µm, ezek alapján a cső keresztmetszetét 5 mm átmérőre, magasságát pedig 10 mm-re állítottam be. Ezt a csövet egy alaplapra helyeztem el, amelynek feladata egy sík bázisfelületként megtartani a rajta elhelyezkedő koptató talaj halmazt. A szimuláció alap felépítését a 30.ábra mutatja be.

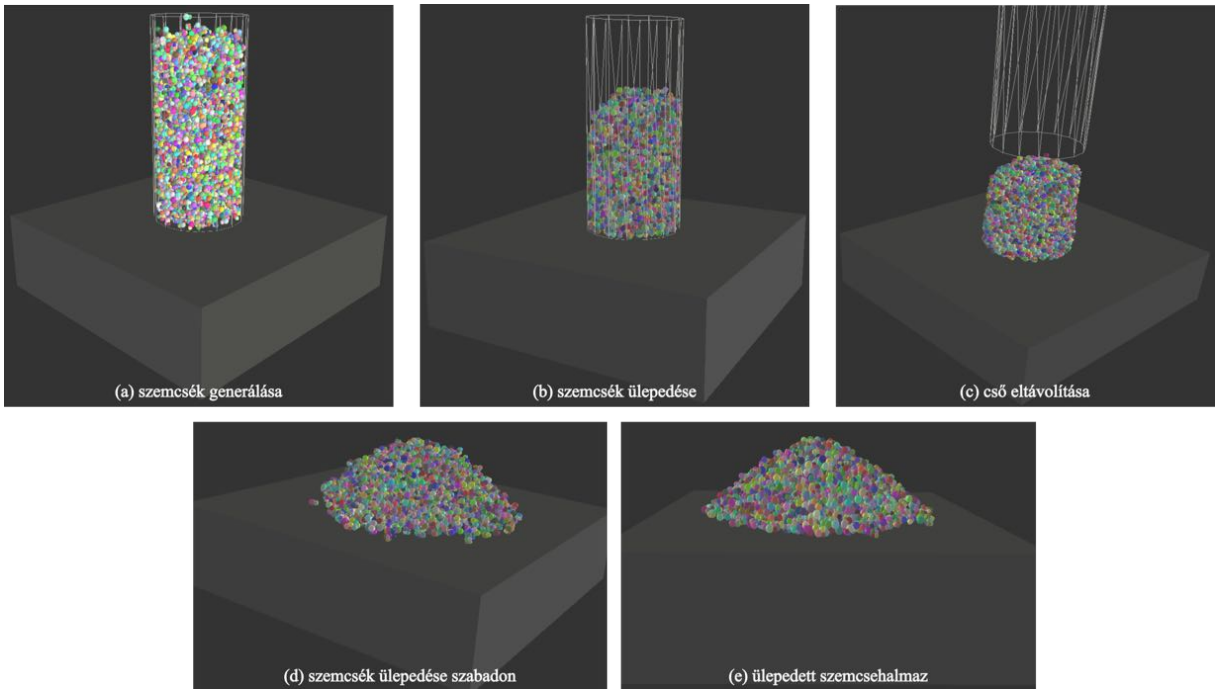
30. ábra: Rézsűszög vizsgálat szimuláció alap felépítése
(Forrás: Saját kép)



A cső talajjal való megtöltését a Yade *makeCloud()* funkciójával valósítottam meg, amely egy véletlen eloszlású, átfedés nélküli szemcse felhőt hoz létre a definiált cső geometria belsejében. A koptatóanyag összetétele pontosan meghatározható, ugyanis az egyes típusú szemcsék százalékos aránya definiálható az egyedileg fejlesztett *clumpTypePercentages = [20, 20, 20, 20, 20]* funkció segítségével. Az előállított szemcsék legenerálása után a szimuláció futtatás első fázisa lezárult, ezután a következő szakasz az ülepítés, amikor a létrehozott clumpok a gravitáció ellenében leülepednek. Ahhoz, hogy kontrolláltan és előre meghatározott értékig történjen minden esetben ez a folyamat az anyaghalmaz *unbalancedForce()* értékét figyeli a program. Ez a funkció a testekre ható átlagos összegző erőt hasonlítja össze a kölcsönhatásokból ébredő átlagos erő nagyságával, 0 az értéke abban az esetben, amikor teljesen stabil a rendszer, ugyanis ekkor a két erő kioltja egymást. Jelen esetben az *unbalancedForce()* értékét 0,12-re választottam, amely már kellően stabil állapotot jelez. Annak érdekében, hogy biztosítva legyen, hogy a szemcsék elérték a stabil állapotot, a program

a határérték elérése után nyolcszor ellenőrzi egymás után 1000 iterációnként a rendezetlenség mértékét, amint mind a nyolc mérés sikeres volt a szimuláció átvált a következő fázisba, amely a cső felemelését és a szemcsehalmaz újbóli stabilizálódását foglalja magában. Annak érdekében, hogy a cső felhúzása minél kevesebb interakciót okozzon a szemcsehalmazon, ennek egy olyan acél alapanyagot definiáltam, amely súrlódást befolyásoló tényezője alacsonyabb a hagyományos acélnál. A cső a felfelé történő mozgási sebességét az Altair ajánlása apján határoztam meg 0,2 mm/s sebességre. Miután az objektum teljesen elhagyta a szimulációs teret a szemcsehalmaz elkezd újra ülepedni és kialakítani egy rézsűállapotot, amelynek eredményekét a szemcsék kúp alakzatban rendeződnek, amelynek mérhető a rézsűszöge a korábban ismertetett mérési módszer szerint. A szimuláció egyes lépéseit a 31.ábra szemlélteti a véglegesnél nagyobb szemcseméretet alkalmazva, ezzel megkönnyítve és meggyorsítva a tesztelési és hibajavítási folyamatokat.

31. ábra: Rézsűszög szimuláció legfőbb lépései
(Forrás: Saját kép)



A szimulációs eredmények utólagos eltárolása és feldolgozhatósága érdekében, valamint a futtatás közbeni ellenőrzése és felügyelete miatt nagy mennyiségű információt dolgoz fel, jelenít meg és tárol el a program. Futtatás közben a terminál ablakban szövegesen minden 2000 iteráció után megjelenik az aktuális folyamat megnevezése, a szimulációban eltelt virtuális idő, a szemcsék száma és az *unbalancedForce()* értéke. Az utolsó ülepítési folyamat során nem határoztam meg olyan kiegyensúlyozatlansági erő értéket, amit elérve a program automatikusan megállítaná azt, minden esetben manuálisan vizsgáltam és állítottam meg a futtatást, azonban

a manuális határtérték ezesetben is 0,12. A próba futtatások alapján azt tapasztaltam, hogy ezen érték alá nem érdemes menni, mert szinte egyáltalán nem módosítja a kapott végeredményt viszont a futtatási időt nagy mértékben növelheti.

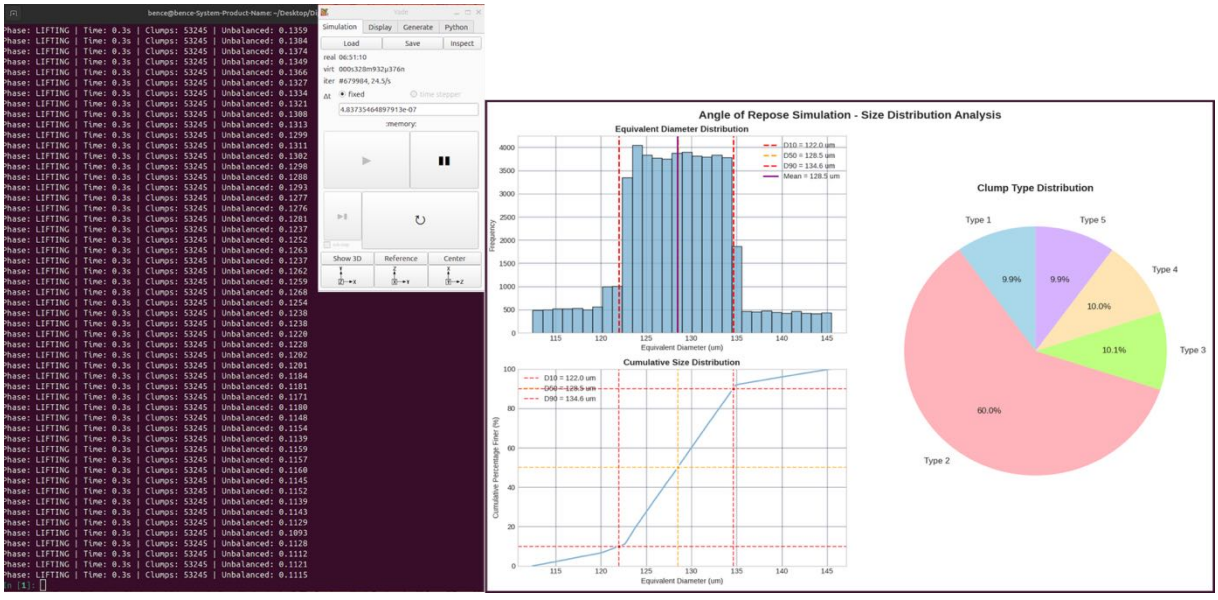
3.2.3 Szimulációs adatok gyűjtése

Ahogyan az a korábbi fejezetben röviden ismertetésre került a futtatás során nagy mennyiségű adatot rögzít a program az utólagos adatfeldolgozás és kiértékelés érdekében. A szimulációban összesen négy különböző módon kerülnek a paraméterek rögzítésre.

Az első és legegyszerűbb a korábban már ismertetett szöveges információ megjelenítés a terminál ablakban, diagramos alakban a szemcsék méret eloszlását és pontos összetételét jeleníti meg a program.

Elkészítettem egy átmérő eloszlás hisztogramot, amely az egyes átmérőket ábrázolja az előfordulás gyakoriságának függvényében, emellett egy kumulatív átmérő eloszlás diagramot, amely azt adja meg, hogy a vizsgált szemcsék hány százaléka kisebb egy adott átmérőnél, továbbá a diagramokon megjelenik a kritikus D10, D50 és D90 értékek is, amelyek azt mutatják meg, hogy a részecskék 10, 50 és 90 százaléka kisebb, mint a mellette kiszámolt érték. Ezen két diagram teljeskörű képet ad a szimuláció során generált részecskék méretéről. Az adatok számításához a korábban a szemcsék előállításánál ismertetett elemi szemcsék térfogatából visszaszámolt közelítő sugár és átmérő értékekkel dolgozik a program. Ezt kiegészítendő a szemcsék keverési aránya egy torta diagramban százalékosan kerül kijelzésre. A bemutatott két vizualizációs megoldást a 32.ábra szemlélteti.

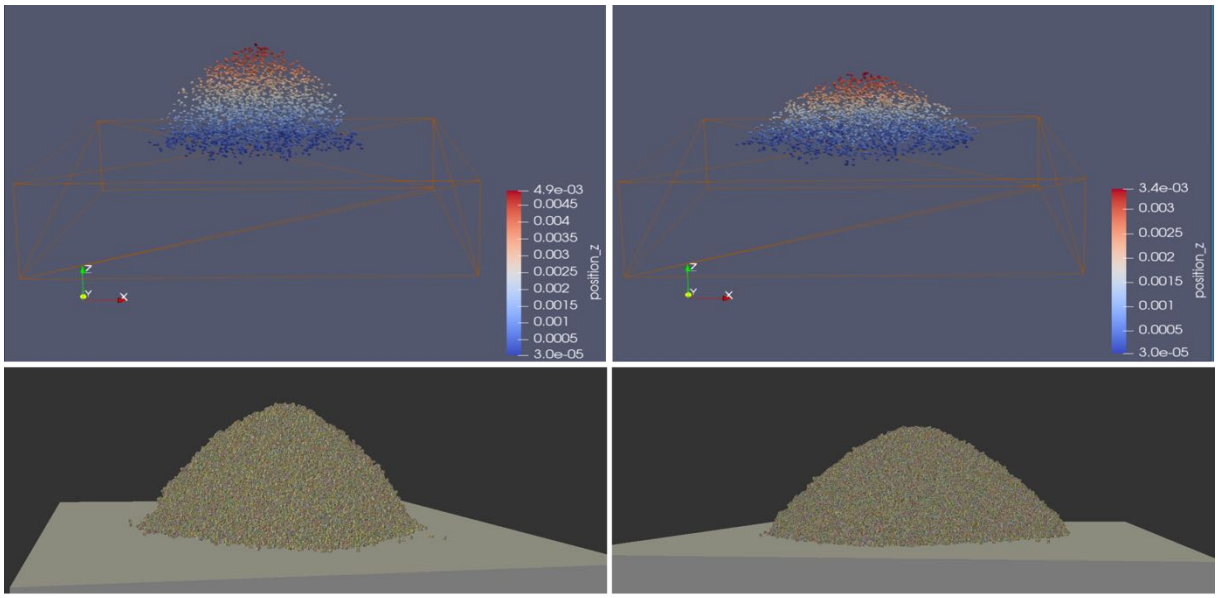
32. ábra: Szimuláció közben megjelenített adatok
(Forrás: Saját kép)



Ezen túlmenően elkészítettem egy adat exportálási funkciót, amely a *completeSimulationExport()* parancs megadására kiexportálja a szimuláció adatait az indítástól a parancs megadásig terjedően. Ez a funkció két különböző .csv fájlt hoz létre. Egyet, amelyik a szimuláció alap adatait, mint például anyagjellemzők, geometriai adatok, részecskék eloszlását adja meg és egy másik külön fájlt, amelyik folyamatosan 2000 iterációnként rögzíti ugyan azon értékeket, amelyek a terminálban kerülnek kiírásra a futtatás során.

Mindezeken túl a szimulációban közreműködő összes objektum kiexportálásra kerül VTK fájlformátumban, amely lehetővé teszi a későbbi vizualizációt és a visszajátszhatóságot, így lehetőséget biztosítva az utólagos elemzésekhez. Három fő csoportra osztottam a kiexportált alakzatokat az alábbiak szerint, alaplap, cső és szemcsék, ennek oka, hogy így külön - külön vizsgálhatók az egyes objektumok. Az adatvizualizációt és az elemzést megkönnyítendő olyan adatok is kiexportálásra kerülnek, mint például a szemcsék esetén azok típusa, tömege, sebessége stb. A fájlokat a szimuláció minden 500 iteráció után elkészíti és elmenti egy külön mappa és fájlszerkezetbe, így azok utólagosan rendkívül egyszerűen benyithatóak egy adatvizualizációs szoftverbe. Dolgozatomban erre a célra a Paraview programot használtam, amelyet alkalmazás közben a 33.ábra mutat be.

33. ábra: Utólagos adatvizualizáció Paraview segítségével
(Forrás: Saját kép)



(a) szemcsehalmaz ülepedés kezdete, fent Paraview, lent Yade vizualizáció (b) ülepedett szemcsehalmaz, fent Paraview, lent Yade vizualizáció

A fentebbi ábrán jól látható a különbség, a Paraview segítségével történő utólagos adatmegjelenítés és a szimuláció során elérhető a Yade Dem szoftverbe beépített háromdimenziós vizualizációs felületek között. Első esetben a szimuláció során számított és tárolt adatok visszaillleszthetők jelen esetben például a szemcsehalmaz magassága a grafikus

környezetbe, így iterációról – iterációra visszajátszható és elemezhető a szimuláció, kiegészítve mindezt színezési térképpel és érték kijelző skálával a még pontosabb adatfeldolgozás érdekében. Ebben az esetben az elmentett elemek száma okozza a limitációt, ugyanis minél magasabb ez a szám annál nagyobb mennyiségű adatot generál a program, túl nagy elemszám esetén egy teljes rézsűszög szimuláció képes 40 – 60 GB mennyiségű adatot generálni. Az ábrán megfigyelhető, hogy jelentős szemcseszámbeli eltérés tapasztalható a két módszer között. A második esetben a grafikus elemzések lefolytatására csupán valós időben van lehetőség, ugyan a szoftverben van arra mód, hogy a számított értékek alapján készítsen színtérképet és azt a modell környezetbe illessze, azonban visszamenőlegesen adatot megjeleníteni ilyen formában nem képes.

3.3 Kopásvizsgálati szimuláció

A dolgozat fő témáját képző koptatási vizsgálathoz létrehoztam egy egyedi szimulációs környezetet, amelynek alapjául a gyakorlatban elterjedt Pin – on - Disk koptatási módszert vettem kiegészítve azt egy koptató szemcse réteggel az elkoptatandó test és a szemcséket szállító felület között, így a próbatest a részecskék elcsúszásából és elgördüléséből következő kölcsönhatások révén szenvedi el a kopást.

3.3.1 Szimuláció alap felépítése és működése

A koptatási szimulációhoz a korábban a 3.2.1. fejezetben részletesen ismertetett regolit szemcséket használtam fel, pontosan olyan módon, mint ahogyan azt korábban definiáltam. Az alkalmazott kontakt modell csakúgy, mint a rézsűszög vizsgálati szimuláció esetén jelen esetben is az *Ip2_FrictMat_FrictMat_MindlinPhys* volt. Továbbá az anyagjellemzők megadása is pontosan ugyan azon módszer segítségével került meghatározásra, mint a korábbi vizsgálat során:

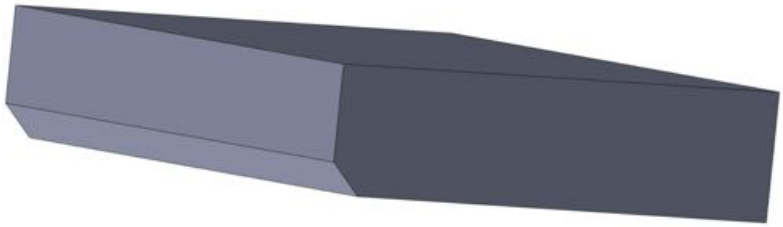
FrictMat(young=40e6, density=1300, poisson=.25, frictionAngle=math.radians(38), label='regolith'),
FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(30), label='steel')

Ezen beállítások egyezősége a két szimuláció esetén kiemelten fontos, ugyanis csak egyező paraméterek esetén kapcsolható össze a rézsűszög vizsgálat és a későbbi koptatási szimuláció eredménye.

A teszt környezet felépítését tekintve áll egy elkoptatandó próbatestből, amelynek szimulációban skálázott mérete minden esetben az alábbi: X irányú hosszúsága 20 mm, Y irányú szélessége 18 mm és Z irányú magassága pedig 10 mm. A próbatest -X irányú alsó élére készítettem egy letörést, amelynek célja, hogy megkönnyítse a koptató szemcsék alácsúszását,

a korai teszt futtatások során ugyanis problémám akadt azzal, hogy a szemcsék nem tudtak kellő számban bejutni a próbadarab alá. Az elkoptatandó geometriát szilárdtest modellként Solidworks 3d tervező szoftverben készítettem el, amelyet a 34.ábra mutat be, majd pedig .STL formátumban exportáltam ki, amely fájlt asztán a szimuláció beolvas és minden esetben skáláz a korábban megadott méretre, amennyiben ez a folyamat szükséges.

34. ábra: A próbatest modellje SolidWorks 2025 programban elkészítve
(Forrás: Saját kép)



A beolvasott .STL formátumú geometria köré először készít egy határoló dobozt a program, amely segítségével meg tudja határozni annak méreteit, majd az elemi háromszögekből létrehozza az objektum felületeit, végül ezen információkból elvégzi a skálázást. Ezt követően elkészíti a végleges felületeket, beállítja az anyagjellemzőket, kiszámítja a felületeket, tömegközéppontokat, a teljes térfogatot ebből pedig a tömeget. Utolsó lépésként egyesíti a különálló felületeket egy merev testté, így a felületek egymáshoz képest nem képesek elmozdulni. Ennek a módszernek a hátránya, hogy a kopás következtében történő felületi változásokat vizuálisan nem követi le. Létezik módszer arra módszer, hogy az egyes felületek képesek legyenek lekövetni a kopás következtében adódó torzulást, azonban ennek kódolási komplexitása és számítási igénye rendkívül nagy, így jelen szimulációt az ismertetett egyszerűsített módon valósítottam meg.

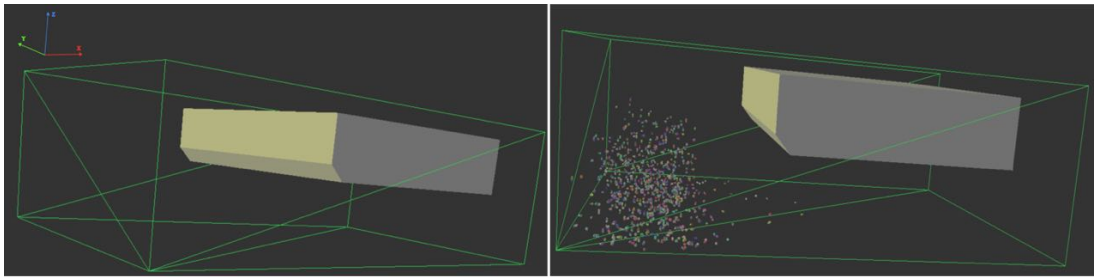
A következő lényeges komponens egy felületekből kialakított +Z és +X irányokban nyitott doboz, melynek -Z belső felülete egy konvejorhoz hasonlóan anyag transzport funkciókat tölt be. X irányú hosszúságát 32 mm, Y irányú szélességét 20 mm, Z irányú magasságát pedig 10 mm értékekre határoztam meg, amelyet az alábbi kódsorral definiáltam:

geom.facetBox(center=(12e-3, 0, 5e-3), extents=(16e-3, 10e-3, 5e-3), wallMask=29, wire=True, material=steelId, color=(0, 1, .3), mask=1, fixed=True)

A geometria létrehozását pedig a *O.bodies.append()* parancs segítségével hajtottam végre. A doboz a szimulációban átlátszó, drótvázként jelenik meg, amelyet a *wire=True* kódrészlet definiál. Ennek lényege, hogy a futtatás közben megkönnyítse a folyamatok vizuális vizsgálatát.

Ezt követően a koptató szemcsefolyamot oly módon integráltam a szimulációba, mintha azok egy szabad kifolyásból esnének a konvejor dobozba. Az alap tesztkörnyezet felépítését a 35.ábra szemlélteti.

35. ábra: A koptatási szimuláció alap felépítése
(Forrás: Saját kép)



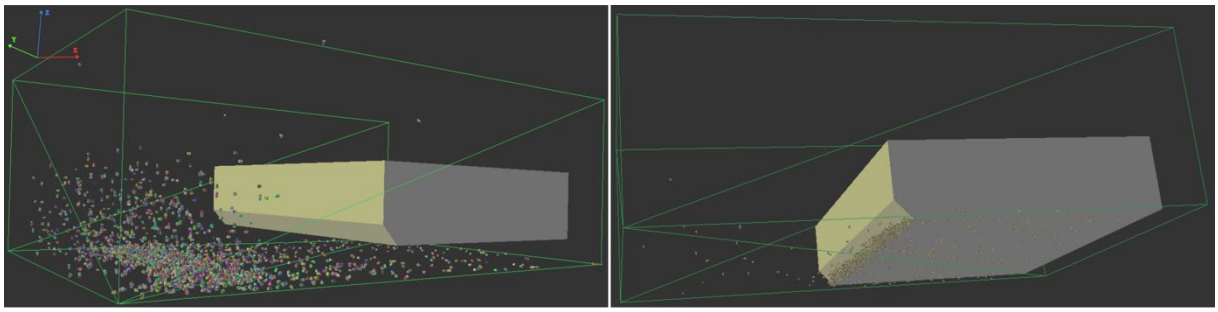
(a) A szimuláció alap felépítése, koptatandó próbatest és a konvejor doboz (b) A szimuláció generált szemcsékkel kiegészített alap felépítése

A koptatóanyag már korábban bemutatott alap clump típusokból épül fel, az egyes alkotók százalékos aránya pontosan meghatározható az általam definiált *clumpTypePercentages = [20, 20, 20, 20, 20]* funkcióval. Az egyes részecskék térfogatát, tömegét és számát a program folyamatosan számítja és nyomon követi, a futtatási teljesítmény optimalizálása céljából meghatároztam egy maximális clump számot, jelen esetben 1800 darabot, amely egyszerre megjelenhet a szimulációban *MAX_ACTIVE_CLUMPS = 1800*, ezenkívül meghatároztam az egy ciklusban maximálisan előállítható részecskék számát, ezzel elkerülve azok egymással történő átfedését és az ebből következő nem kívánt interakciókat, valamint mindezek mellett a pontos tömegáram is megadható, ezen három paraméter alapján jól beállítható egy olyan szemcse előállítási metódus, amely kellően nagy mennyiségű szemcsét állít elő, ugyanakkor képes elkerülni a nem kívánt átfedésben történő részecske generálást.

A konvejor funkciót oly módon alakítottam ki, hogy a program figyeli a kontaktokat a szemcsék és a doboz -Z belső felülete között, abban az esetben, ha észlel kontaktot megvizsgálja a szemcsék +X irányú és a beállított konvejor szállítási sebesség, amely ugyancsak +X irányú közötti különbséget. Ezen sebesség különbségből kiszámítja azt a szükséges gyorsítási erőt, amelyet az egyes szemcsékre kell gyakorolni ahhoz, hogy elérjék a kívánt szállítási sebességet. Ez a módszer tehát nem a valóságban működő súrlódáson alapuló anyagszállítást valósítja meg, hanem egy felügyelt és pontosan szabályozható erő vezérlés alapú transzport. Célja ennek a módszernek, hogy az anyagtovábbításból eredő eltéréseket a lehető legjobban minimalizáljam. Abban az esetben, amikor a szemcsék kölcsönhatásba kerülnek a próbatesttel egy a súrlódásból eredő erő kerül kiszámításra, amely az aktuális súrlódási viszonyoknak megfelelően csökkentik az azokra ható erőt ebből kifolyólag pedig a szemcsék mozgási sebességét, így téve valósághűvé a szemcsék viselkedését az interakció során.

Az elkoptatandó alakzat csak akkor kezd el a gravitáció ellenében zuhanni, amikor alatta már egy kialakult közel homogén szemcsefolyam található, ezt egy meghatározott +X irányú jelző pont kialakításával értem el, amely akkor aktiválódik, ha azon áthaladtak a szemcsék, a folyamat a 36.(a) ábrán jól látható. Annak érdekében, hogy a próbatest a koptatás során ne mozduljon el a pozíciójából a Z irányú tengely kivételével minden tengely mentén történő elmozdulás és elfordulás ellen rögzítve van, így biztosítva annak rögzítését, amelyet a 36. (b) ábra szemléltet.

36. ábra: A próbatest és a koptató szemcsék interakciója
(Forrás: Saját kép)



(a) A koptató szemcsék transzportja és a próbatest ejtése (b) A koptatóanyag és a próbatest interakciója

Annak érdekében, hogy a próbadarab Z irányba ne tudjon ellenállás nélkül elmozdulni kialakítottam egy rá ható – Z normális irányú erőt, amely az állandó előfeszítést hivatott megvalósítani, ez a terhelés azonban csak azután aktiválódik, miután a próbatest és a szemcsék között legalább 10 kontakt kialakult. A terhelő erő értékét 10 N nagyságban határoztam meg, amely könnyedén módosítható a *wearbody_force_magnitude = 10* változó segítségével.

3.3.2 Koptatással kapcsolatos funkciók megvalósítása

A szimuláció legfontosabb funkciója, hogy képes legyen a próbatest által elszenvedett kopás mértékét számítani és nyomon követni. Az elkoptatott anyagmennyiséget a már korábban a 2.1.3. fejezetben bemutatott Archard féle összefüggés alapján határoztam meg, amely funkció nem érhető el a Yade beépített moduljaiban, így egyedileg fejlesztettem jelen vizsgálati környezethez.

Az elkoptatandó test felületekből azon belül is elemi háromszögekből épül fel, az első lépés, hogy a program meghatározza a kialakult kontaktok számát ezen elemi felületek és a szemcsék között. Ezt követően a kialakult kontaktoknál meghatározza az ott ébredő normális irányú erők nagyságát, majd pedig számolja a számolja a kölcsönhatásban résztvevő objektumok sebességeit, ezen információkból pedig a csúszási távolságot. A kopási együtthatót és a kopásban résztvevő puhább anyag keménységét konstans változókként definiáltam. Ezen érékeket a szakirodalmakban leírtak alapján definiáltam szem előtt tartva azt, hogy a lehető

legnagyobb kopás intenzitási beállításokat alkalmazzak a szimuláció gyorsabb lefutása érdekében. A kopási együttható egy dimenzió nélküli szám, amely, az jelzi, hogy inkább forgácsolásból, vagy pedig deformációból adódó koptatást okoz az idegen tárgy a próbatesten, tipikus tartománya 10^{-1} től 10^{-6} értékig terjed. A nagyobb számérték azt jelzi, hogy puhább anyag és éles szemcsék interakciója történik (Bhushan, 2013). A keménység tekintetében a marsi szimulánsok jellemzően bazalt alapúak, így referenciaként ennek értékeit vettem, amely körülbelül 1600 – 2000 MPa között van (Gopaul és munkatársai, 2022), míg az S420 szerkezeti acélé körülbelül 550 – 680 MPa (S420 acél adatlapja). Ezek alapján a szimulációs beállításokat az alábbi kódrészlet adja meg

```
k_archard = 10e-2 # [-]
wearbody_hardness = 5e6 # [MPa]
```

Minden szükséges paraméter definiálása után, így már megadható a kopási térfogat számításához szükséges összefüggés, amelyet a kódsorban az alábbi módon definiáltam.

```
# Archard wear equation: V = k * F_n * s / H
wear_volume = k_archard * normal_force * sliding_distance / wearbody_hardness
wear_volume = max(0.0, min(wear_volume, 1e-8))
facet_wear_volume += wear_volume
```

Annak érdekében, hogy elkerüljem az esetleges, véletlenszerűen előforduló túl nagy mértékben kiugró, vagy bármilyen okból a negatív értékű térfogat változást készítettem egy limitáló funkciót, amely folyamatosan figyeli az iterációnként számított kopás mértékét és csak abban az esetben adja hozzá a kumulált kopási értékhez, ha az a meghatározott minimum és maximum tartomány között van. A kezdeti tesztelési fázisban előfordult olyan eset, amikor több szemcse oly mértékben felgyorsult a generálás közbeni interakció során, hogy nekiütközve a próbatestnek az átlagostól jelentős mértékben eltérő kopást okozott azon.

Ezen hibák megakadályozását hivatott szolgálni a bevezetett limitáló funkció továbbá a szemcsék mozgását is figyelemmel kíséri a program és abban az esetben, ha a szemcsék túlzott mértékben felgyorsulnának, vagy forognának egy csillapító funkció visszaállítja azokat a jellemző mozgási paraméterek tartományába. Ezen túlmenően számos kopással kapcsolatos információ kiszámolásra és tárolásra kerül a későbbi adatkiértékelés és vizualizáció céljából. Kialakítottam egy cél kopási értéket, amelyet 0,2 mm^3 térfogat veszteségben határoztam meg a próbatest esetén. A futtatást nem állítja meg automatikusan ezen határérték elérése esetén a program, a nyomon követhetőséget azonban javítja.

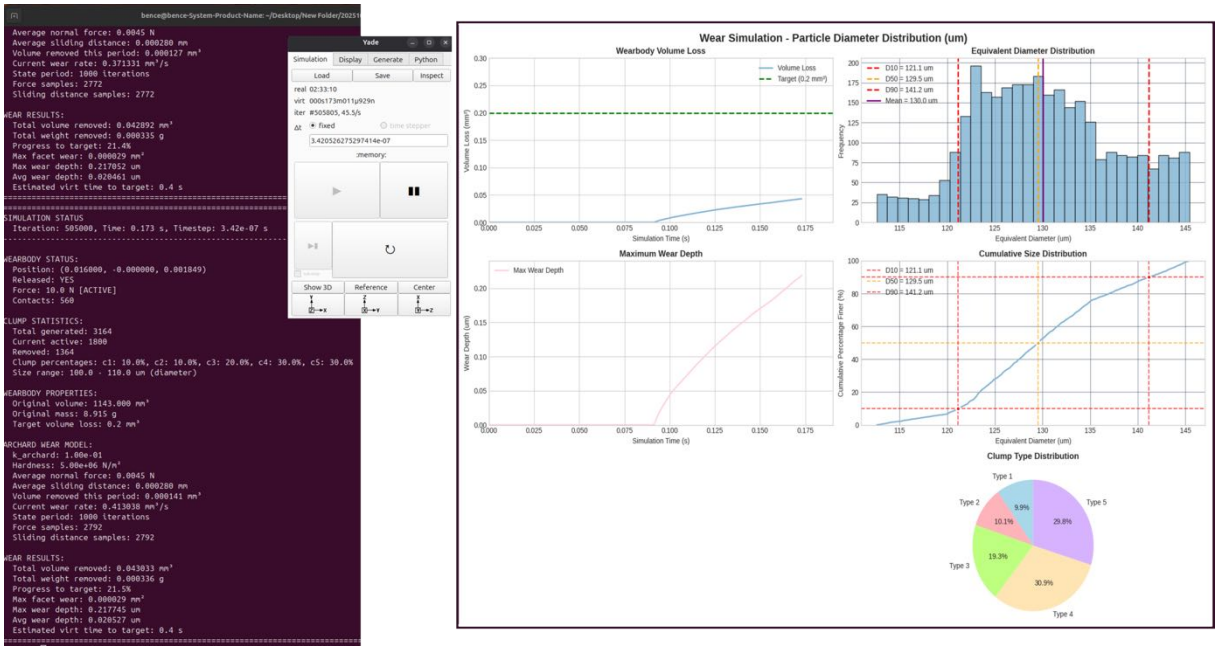
3.3.3 Dokumentáció és adatgyűjtés

Korábban a 3.2.3 fejezetben már részletesen ismertem a rézsűszög vizsgálati szimulációnál alkalmazott adat gyűjtési, tárolási és megjelenítés módszereket, amelyeket a koptatási környezetbe nagy egyezőséggel integráltam, azonban hozzáadtam néhány további kiegészítés, hogy még több információt tudjak tárolni az eredményekről. Összesen most is négy különböző módon történik az adatok kezelése, hasonlóan, mint ahogyan az a rézsűszög szimuláció esetén bemutatásra került.

Elsőként a terminál ablakban közölt paramétereket érdemes említeni, amely megjelenít minden olyan alap futtatási információt, mint például az aktuális iteráció száma, a koptatóanyag összetétele és méret tartománya, a próbatest pozíciója stb. ezen túlmenően olyan a kopással összefüggő értéket, amely a gyors nyomon követést jelentősen megkönnyíti mint például az átlagos csúszási távolság és normális irányú erők 1000 iterációnkénti átlagos értékét, a teljes elkoptatott anyag térfogatát, a számított virtuális időt, amely a cél kopási érték eléréséhez szükséges stb.

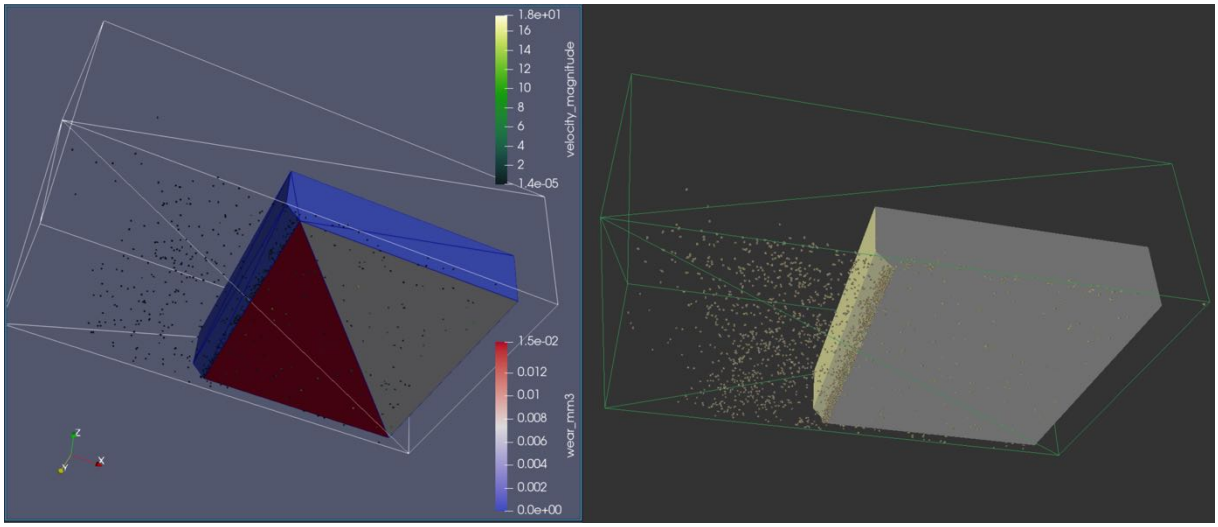
Ezenkívül a program valós időben megjelenít és frissít különböző diagramokat, amelyek egy része a már korábban ismertetett koptatóanyag összetételére vonatkoznak, továbbá egy diagramban az elkoptatott térfogat kerül ábrázolásra a szimulációs idő függvényében és egy utolsó diagram, amely a kopási árok mélységét jeleníti meg ugyancsak a szimulációs idő függvényében, az ismertetett adat megjelenítő módokat a 37.ábra szemlélteti.

37. ábra: Valós idejű adatmegjelenítési módok a Yade felületén
(Forrás: Saját kép)



Az adatok elmentéséért jelen esetben is a *completeSimulationExport()* egyedileg definiált funkció a felelős, amely táblázatos formában menti ki a szimulációval kapcsolatos általános paramétereket, a szemcsék méretével, összetételével kapcsolatos adatokat, továbbá a koptatással kapcsolatos a szimulációs időhöz kapcsolt értékeket, amelyek a dolgozat eredményeinek meghatározásában jelentős szerepet töltenek be. Végül pedig az összes objektum, elkülönítve kiexportálásra kerül VTK fájlformátumban, amely a későbbi Paraview szoftverben történő elemzést teszi lehetővé. A 38.ábrán kerül bemutatásra az eltérés aYade beépített felülete és az utólagos Paraview szoftverben történő visszajátszás között.

38. ábra: Paraview vizualizáció (balra), Yade beépített grafikus megjelenítés (jobbra)
(Forrás: Saját kép)



Az ábrán jól látható, hogy Paraview szoftverben az elkoptatott anyagmennyiség függvényében a program képes színezési térképet készíteni a próbatestre, így lehetőséget biztosítva például a kopás eloszlásának az elemzésére, amely kék – piros átmenetű színezéssel jelenik meg az ábrán. A Paraview elemzésén megfigyelhető ahogyan a próbadarab felülete elemi háromszögekből épül fel, amelyre a Yade képes a számításokat elvégezni. Ezen túlmenően megjeleníthető például a szemcsék haladási sebességére vonatkozó információ, amely az ábrán zöld árnyalatú színezéssel került megjelenítésre. Yade-ben ezen információk nem kerültek megjelenítésre, a hatékonyabb futtatás érdekében.

3.3.4 Vizsgálati paraméterek

A szimulációk futtatásával azt vizsgáltam, hogy a koptatóanyag egyes paramétereinek megváltoztatása hogyan befolyásolja a próbatest kopásának az intenzitását. Annak érdekében, hogy az elvégzett tesztek jól nyomon követhetők legyenek táblázatos formában foglaltam össze az egyes variánsokat. Két fő megközelítést alkalmaztam, melynek egyik eleme az abraziv közeg

szemcseösszetételének pontosan definiált összetétele. A korábban a 29.ábrán bemutatott egyes alap szemcsetípusokból alakítottam ki nyolc különböző keveréket, melyeknek százalékos összetétele a 3.táblázatban látható összefoglalva.

3. táblázat: Koptatóanyag keverékek specifikációja
(Forrás: Saját szerkesztés)

Koptatóanyag keverék megnevezése	A keverékek százalékos összetétele az egyes típusú szemcsékből
eq	C1 (20%), C2 (20%), C3 (20%), C4 (20%), C5 (20%)
c5	C1 (10%), C2 (10%), C3 (10%), C4 (10%), **C5 (60%)**
c4	C1 (10%), C2 (10%), C3 (10%), **C4 (60%)**, C5 (10%)
c3	C1 (10%), C2 (10%), **C3 (60%)**, C4 (10%), C5 (10%)
c2	C1 (10%), **C2 (60%)**, C3 (10%), C4 (10%), C5 (10%)
c1	**C1 (60%)**, C2 (10%), C3 (10%), C4 (10%), C5 (10%)
m1	C1 (10%), C2 (20%), C3 (10%), C4 (30%), C5 (30%)
m2	C1 (10%), C2 (10%), C3 (20%), C4 (30%), C5 (30%)

A vizsgálati paraméterek megváltoztatásának másik eleme azok anyagjellemzőinek pontos definiálása. Definiáltam hét különböző anyagjellemzőt, amelyeket a 4.táblázatban foglaltam össze. Először elkészítettem a V1 – es verziót, amely definiálja az alap paramétereket, majd pedig minden egyes új verzió esetén változtattam egy paramétert. Az eredetihez képest módosított paramétereket színekkel jelöltem, oly módon, hogy az azonos jellemzők megegyező színnel kerültek feltűntetésre.

4. táblázat: Koptatóanyag szimulációs beállítási paraméterei
(Forrás: Saját szerkesztés)

Koptató szemcsék: V1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	2600
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	38
rmin [um]	0,04	50
rmax [um]	1000	55

Koptató szemcsék: V2 MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	38
rmin [um]	0,04	50
rmax [um]	1000	55

Koptató szemcsék: V3 MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	34
rmin [um]	0,04	50
rmax [um]	1000	55

Koptató szemcsék: V4 MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	44
rmin [um]	0,04	50
rmax [um]	1000	55

Koptató szemcsék: V2_s1 MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	38
rmin [um]	0,04	30
rmax [um]	1000	35

Koptató szemcsék: V2_s MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	38
rmin [um]	0,04	40
rmax [um]	1000	45

Koptató szemcsék: V2_b MGS-1		
Paraméter	**Valós érték**	**Szimuláció beállítás**
Young modulus [Pa]	10 - 50e6	40e6
Sűrűség[kg/m3]	1300	1300
Poisson [-]	0,15 - 0,35	0,25
Súrlódási szög [°]	34 - 40	38
rmin [um]	0,04	60
rmax [um]	1000	65

Ezen két fő, előre jól meghatározott paraméter halmazból párosítottam össze az egyes futtatások peremfeltételeit, amely párosításokat a későbbiekben bemutatásra kerülő eredmények fejezetben minden esetben feltüntetek.

3.3.5 A futtatási környezet bemutatása

A szimulációkat egy dedikált asztali számítógépen futtattam, amelyben egy Intel Core i7 - 14700K 28 magos processzor, valamint 32 GB RAM kapott helyet. A gépen natívan Ubuntu 25.04 operációs rendszert használtam, az általam alkalmazott Yade verzió pedig a 2025.2.0 volt.

Ezen konfiguráció mellett a programot a több szálas futtatási módon indítottam el, azaz indításkor a -j paranccsal megadtam a használni kívánt processzor magok számát. Számos összehasonlító tesztet végeztem, amely alapján esetemben azt az eredményt kaptam, hogy a legoptimálisabban az az eset, amikor az összes processzormagot használja a program. Esetemben tehát a -j28 parancs megadásával indítottam minden esetben minkét szimulációt. Ezen konfiguráció és beállítások mellett az egyes futtatások a rézsűszög vizsgálat esetén 12 – 24 óra közötti, a koptatási szimulációk 20 – 80 óra közötti futtatási időt igényeltek az ismertetett, előre definiált cél értékek eléréséhez.

Érdemes továbbá azt megjegyezni, hogy a szimulációk nagy mértékben RAM igényesnek bizonyultak, a rézsűszög vizsgálatok 20 – 30 GB mennyiséget, míg a koptatási tesztek 10 – 15 GB területet kívántak.

Mindkét szimuláció teljes kódja megtalálható a mellékletek fejezetben.

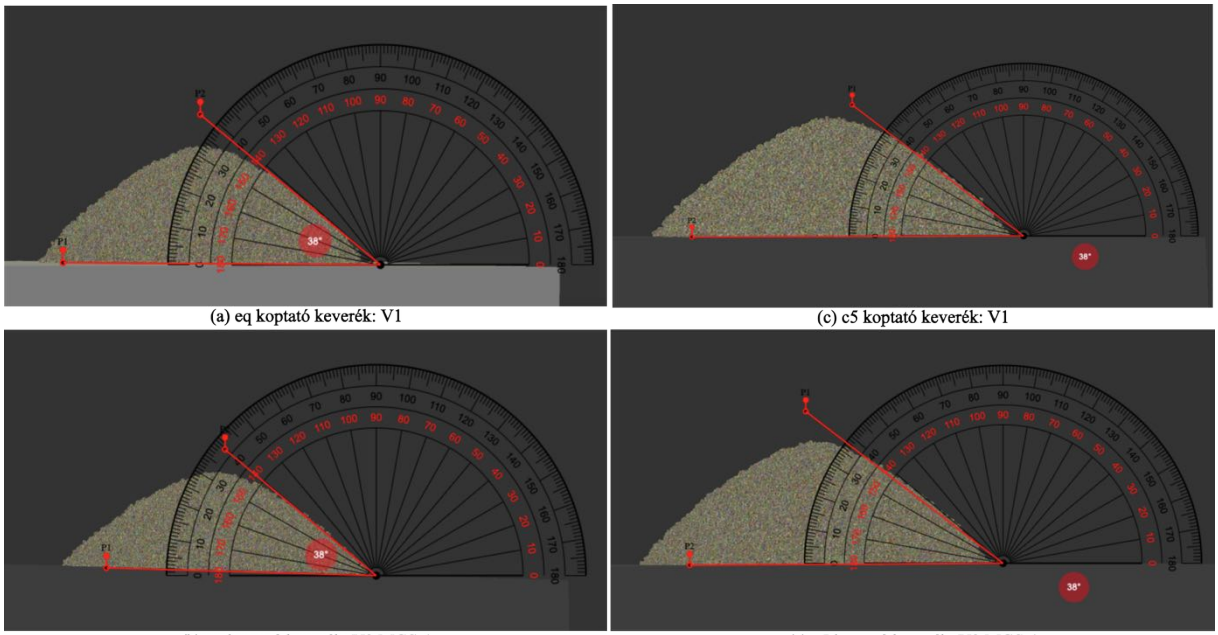
4 EREDMÉNYEK

Ebben a fejezetben részletesen ismertetem a lefuttatott rézsűszög mérési és koptatási vizsgálati szimulációs eredményeket a megváltoztatott koptatóanyag paraméterek függvényében.

4.1 Rézsűszög vizsgálati eredményei

A koptatóanyag rézsűszög értékeit utólagosan a szimulációk lefuttatását követően volt lehetőségem megmérni. A kapott szemcsehalmazok geometriájáról képet készítve, majd ezen képekre szögmérővel a rézsűszögeket berajzolva és megmérve az eredmények jó közelítéssel megegyeztek az MGS-1 marsi regolit 003-05-001-0523 számmal ellátott specifikációs adatlapjában meghatározott átlagos 38,9° rézsűszög értékkel. A szemcseösszetétel és a sűrűség jelen esetben nem okozott jelentős eltérést az értékekben, a 39.ábrán jól látható, hogy a mért értékek 38° közelében alakultak.

39. ábra: Rézsűszög mérés eredményei
(Forrás: Saját kép, szimulációs eredmények alapján)



(a) eq koptató keverék: V1

(c) c5 koptató keverék: V1

(b) eq koptató keverék: V2 MGS-1

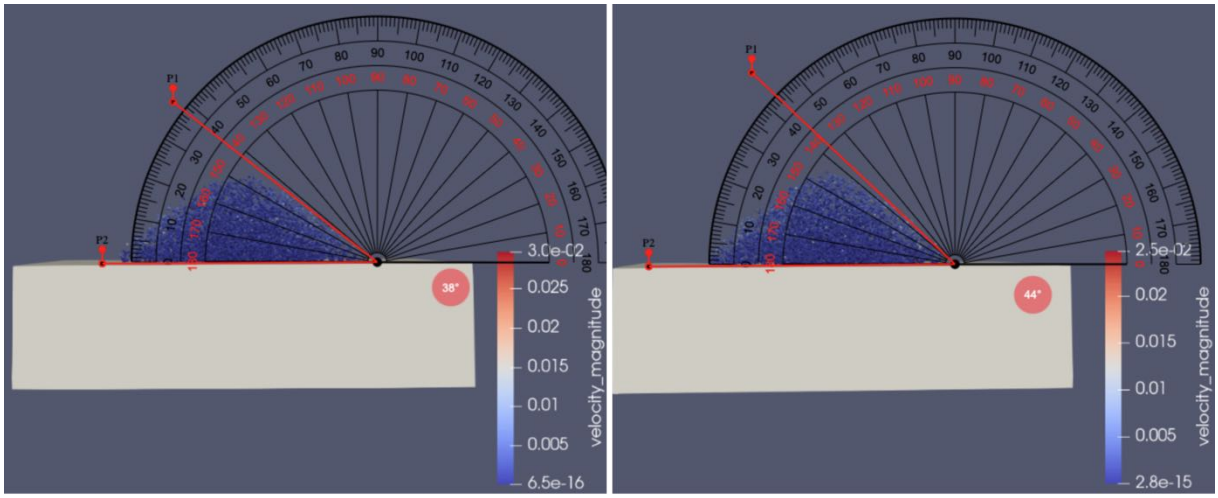
(c) c5 koptató keverék: V2 MGS-1

A 4.táblázat alapján mind a V1 és a V2 MGS-1 súrlódási szögét 38° értéknek határoztam meg, ezek alapján, tehát a rézsűszög értékben eltérést előzetesen nem vártam. A sűrűség különbség hatására ugyanazon összetétel mellett a sűrűbb szemcsehalmaz hamarabb elérte a meghatározott belső rendezetlenségi erő, azaz az UnbalancedForce értéket, a szemcsék nagyobb tömegéből adódóan.

Összehasonlítást végeztem olyan koptatóanyag specifikációkkal is, amikor eltérő súrlódási szög értéket állítottam be az egyes variánsokhoz azonos szemcseösszetétel mellett. A méréseket

ezúttal Paraview alapján végeztem el. A kapott eredmény jól közelítette az elvártakat, ugyanis a V4 MGS-1 anyaghoz definiált 44° súrlódási szög érték jól mérhető volt a vizsgálat után. Az eredmények a 40. ábrán láthatóak, továbbá a skálákon a szemcsék mozgási sebessége látható, amely szintén az ülepedett állapot elérésének vizsgálatát segíti.

40. ábra: A koptatóanyag súrlódási szögének hatása a rézsűszög mérésre
(Forrás: Saját kép, szimulációs eredmények alapján)



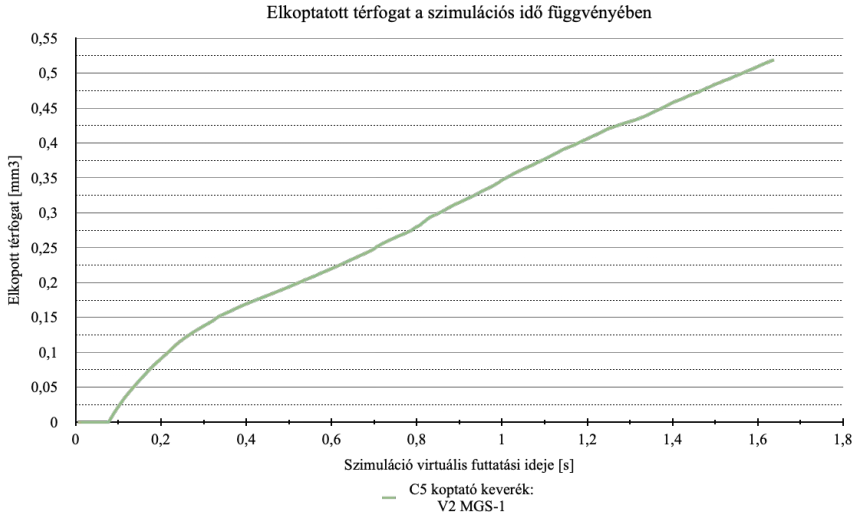
(a) c5 koptató keverék: V2 MGS-1, Paraviewben megjelenítve (b) c5 koptató keverék: V4 MGS-1, Paraviewben megjelenítve

A szögértékeket utólagosan egy online eszköz segítségével mértem, amely az alábbi linken érhető el: https://protractormaster.online

4.2 Kopási vizsgálat eredményei

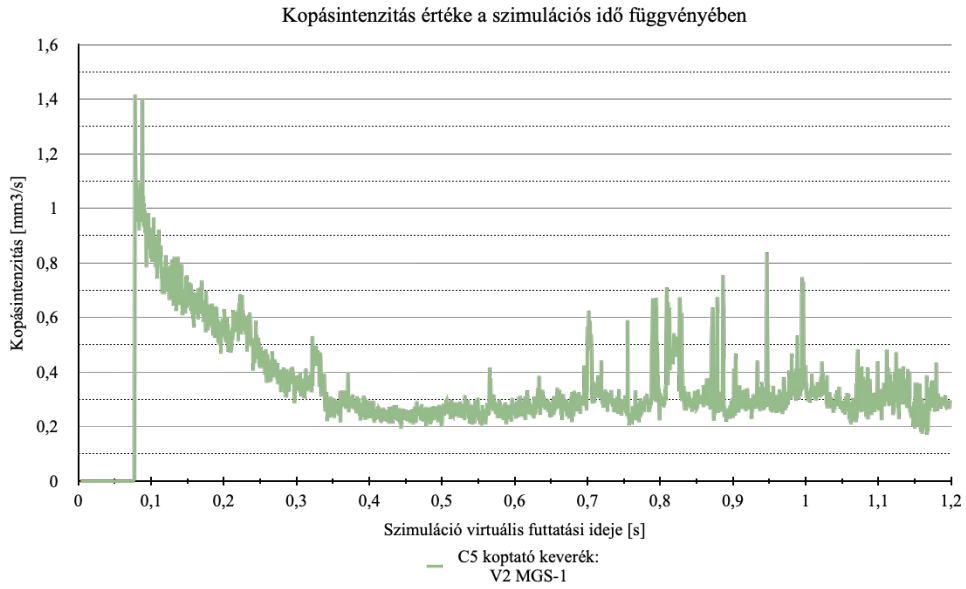
Számos koptatási vizsgálatot lefuttatva két alapvető megállapítás tehető az elkészített szimuláció működésével kapcsolatban. A 41.ábrán megjelenített diagram alapján kimondható, hogy a próbatest anyagvesztése a kopási folyamat során közel lineáris, amely megfelel a szakirodalomban definiált jellemző kopási karakterisztikának (Gee és munkatársai, 2005).

41. ábra: Próbatest anyagveszteségének diagramja
(Forrás: Saját kép, szimulációs eredmények alapján)



Másik megállapítás a 42.ábra alapján tehető, amely szerint a kopásintenzitás értéke az idő függvényében csökken. Ezen jelenség ugyancsak számosa szakirodalomban megtalálható, mint jellemző karakterisztika a kopási folyamatok során.

42. ábra: A pillanatnyi kopásintenzitás diagramja
(Forrás: Saját kép, szimulációs eredmények alapján)

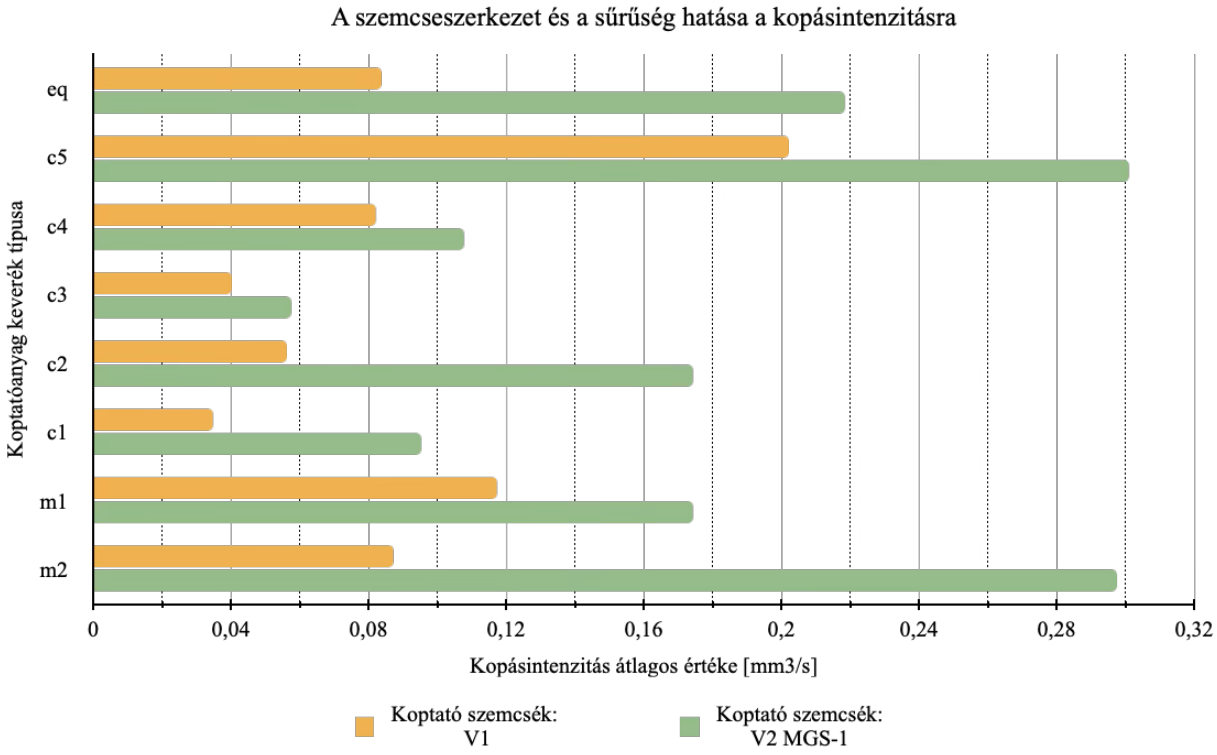


Ezen tapasztalások alapján tehát kimondható, hogy az elkészített szimuláció a kopás alapvető jellemzőit képes jól visszaadni a bemutatott esetekben, így ezek alapján a paraméterek megváltoztatásának a kopásra gyakorolt hatását érdemes jelen virtuális környezettel vizsgálni.

4.2.1 Szemcseszerkezet hatása a kopás intenzitására

Az első vizsgált paraméter a koptató szemcsehalmaz összetételének hatása a kopásintenzitásra. A 43.ábra alapján az a következtetés vonható le, hogy az összetetteb, élesebb kialakítású szemcsékből álló koptatóanyag intenzívebb koptatást eredményezett, a vizsgálatokat két különböző sűrűség beállítással is elvégeztem, a kapott eredmények mindkét esetben nagyon hasonlóak és megfelelnek a korábbam megfogalmazott megállapításnak.

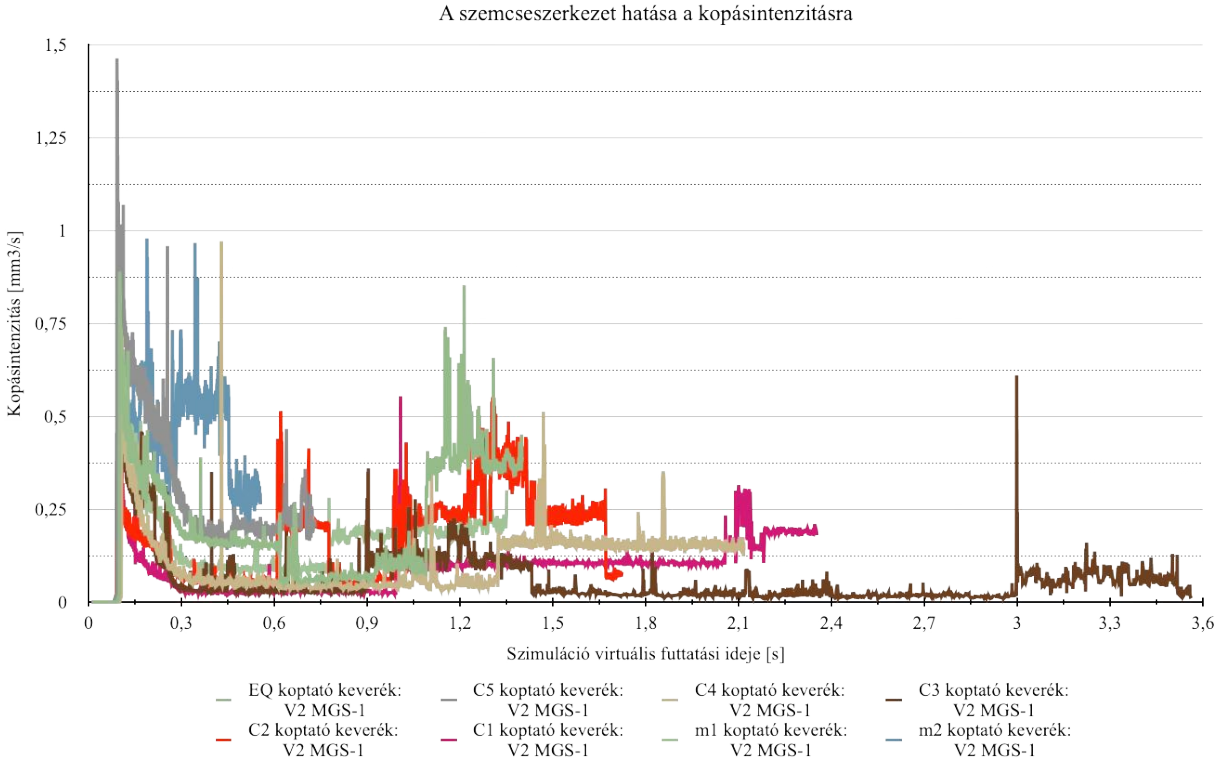
43. ábra: Koptatóanyag összetételének koptatásra gyakorolt hatása
(Forrás: Saját kép, szimulációs eredmények alapján)



A jelenség azzal magyarázható, hogy az élesebb, összetettebb kialakítása elemi szemcsék sokkal inkább hajlamosak összeakadni, továbbá az inkább szögletes geometriájuk miatt sokkal kevésbé képesek elgördülni. Mivel az elgördülésük korlátozott, így sokkal inkább jellemző az ilyen típusú szemcsék forgácsoló hatása. Az irodalomkutatás során megismert irodalmak alapján a koptató szemcsék geometriájának hatása a valóságban megegyezik a szimulációban tapasztalt jelenségekkel.

A pillanatnyi kopásintenzitás értékeket ábrázolva a szimulációs idő függvényében, amely a 44.ábrán látható a görbék jól lekövetik az átlagos kopásintenzitás értékek alakulását. Továbbá az egyes adatsorok hossza jó visszajelzést ad arra, hogy melyik összetételű koptatóanyag mennyi idő alatt volt képes a kitűzött 0,2 mm^3 anyagveszteséget elkoptatni a próbatestből.

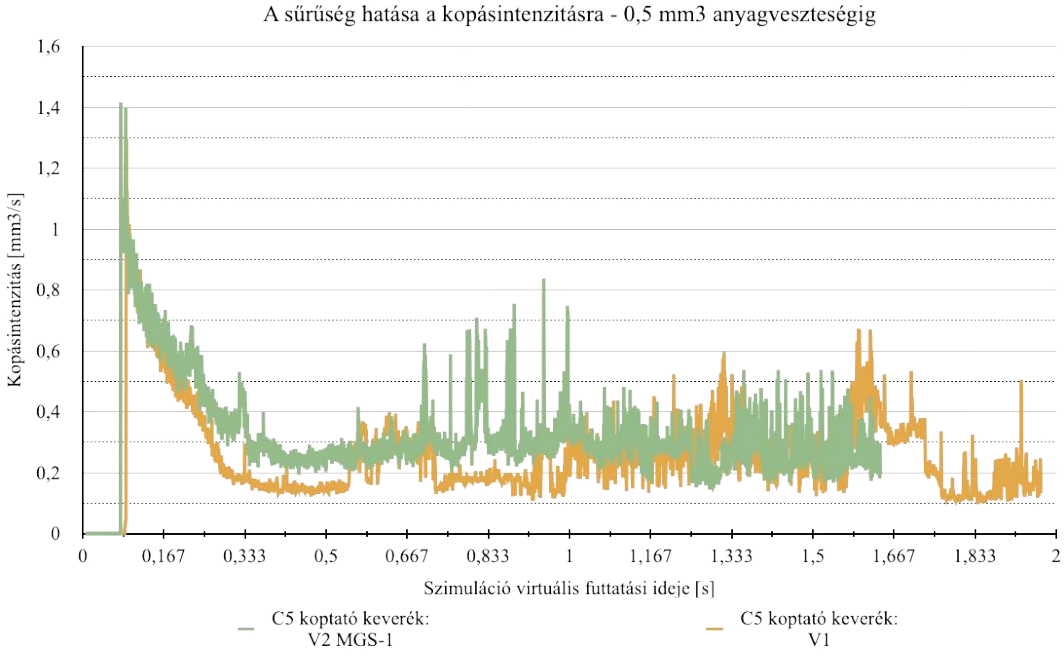
44. ábra: Pillanatnyi kopásintenzitás értékek különböző koptatóanyag összetételek esetén
(Forrás: Saját kép, szimulációs eredmények alapján)



4.2.2 A koptatóanyag sűrűségének hatása a kopás intenzitására

A koptatóanyag sűrűségének kopás intenzitásra gyakorolt hatását megvizsgálva azt az eredményt hozta a szimuláció, hogy bár kezdetben magasabb kopást generál a kisebb sűrűségű koptatóanyag, azonban hosszabb távon ez az intenzitás lecsökken, majd egy idő után a nagyobb sűrűségű anyag okoz intenzívebb koptatást, amely trend a 45.ábrán bemutatott diagramon jól kivehető.

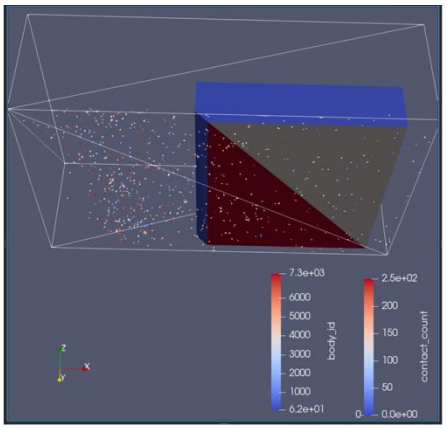
45. ábra: Abraziv anyag sűrűségének hatása a kopásra
(Forrás: Saját kép, szimulációs eredmények alapján)



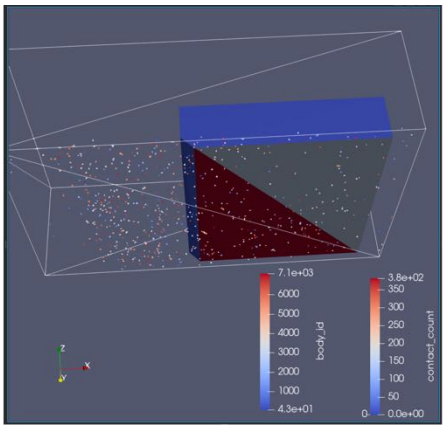
A szakirodalomban leírtak alapján a sűrűbb koptatóanyag jellemzően agresszívebb kopást eredményez, ugyanis az egyes szemcsék nagyobb tömeggel ebből következően pedig nagyobb mozgási energiával rendelkeznek, amelyek a koptató hatást növelik (Xia és munkatársai, 2019). Esetemben megvizsgálva a szimuláció futási paramétereit a kapott eredmények azzal magyarázhatók, hogy a kisebb sűrűségű koptatóanyagból a konvejor kezdetben sokkal nagyobb mennyiséget képes szállítani, mint a nagyobb sűrűségűből, így szimuláció korai fázisában a kontaktok száma jelentősen nagyobb kisebb sűrűség esetén, amely így nagyobb kezdeti kopást eredményez. Hosszabb távon azonban a kontaktok száma normalizálódik mindkét esetben, sőt a nagyobb sűrűséggel rendelkező koptatóanyag által kialakított kontaktok száma meg is haladja az alacsonyabb sűrűségűét. Tehát a futtatás közben vagy egy időpillanat, amikor a kopási trendek elmetszik egymást, innentől kezdve a sűrűbb közeg intenzívebb koptató hatást gyakorol a próbatestre.

Paraviewben visszajátszva és elemezve a szimulációt, a jelenséget a 46.ábra szemlélteti, ahol a contact_count megnevezésű skálák kezdeti és a szimuláció végére kialakult érintkezések számát jelenítik meg a két különböző sűrűségű koptatóanyag esetén. Az ábrán jól látható, hogy a kezdeti érintkezések száma a sűrűbb közeg esetén körülbelül 250-re tehető, míg a kevésbé sűrűbb anyag esetén ez a szám 380 körül van. A futtatás végére a trend azonban megfordul, ekkor a sűrűbb koptatóanyag nagyjából 660 kontaktot alakított ki, míg a kisebb sűrűségű közeg esetén 510 körül alakul ez a szám.

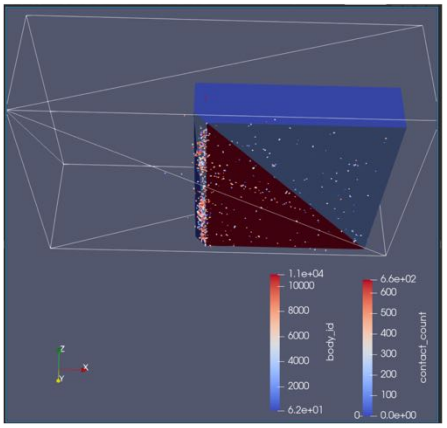
46. ábra: Koptató szemcsék és a próbatest közötti kontaktok száma különböző sűrűség esetén
(Forrás: Saját kép, szimulációs eredmények alapján)



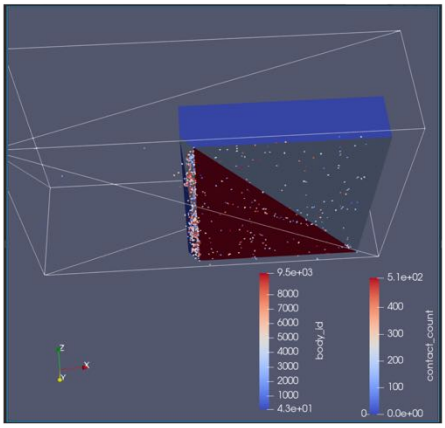
(a) V1 koptatóanyag kezdeti kontaktok száma



(b) V2 MGS-1 koptatóanyag kezdeti kontaktok száma



(c) V1 koptatóanyag kontaktok száma a futtatás végén



(d) V2 MGS-1 koptatóanyag kontaktok száma a futtatás végén

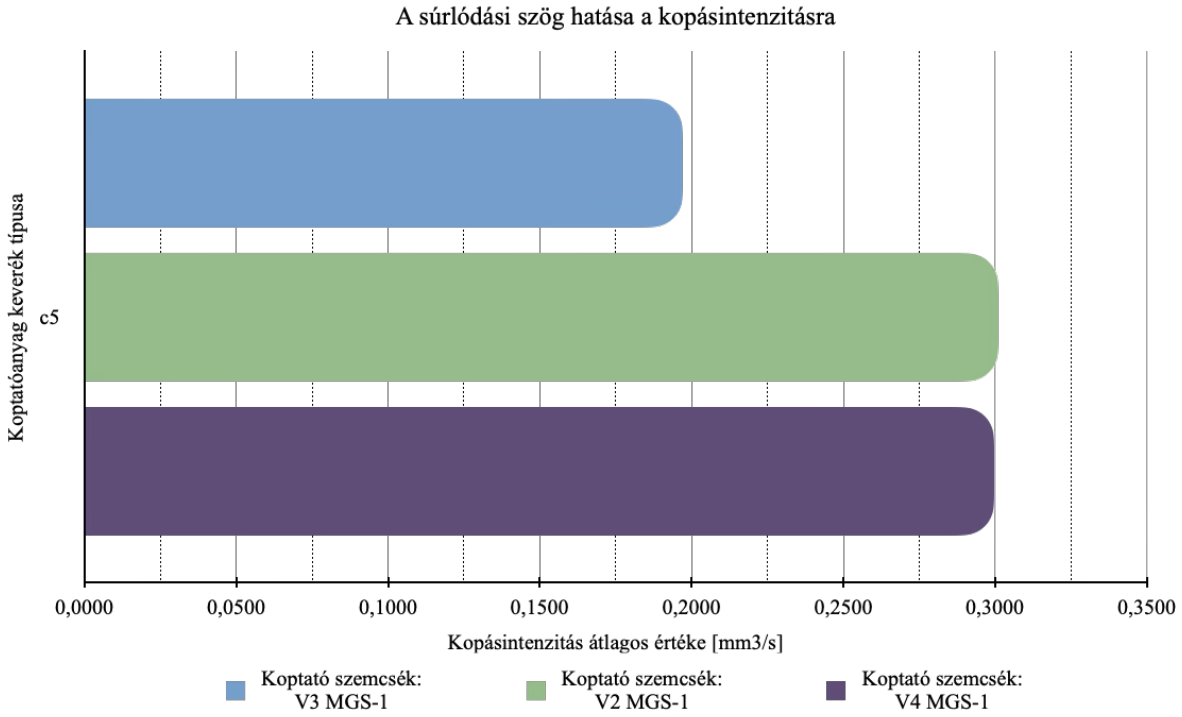
A kopásintenzitási tendenciák teljes bizonyosságú igazolására a szimuláció még hosszabb távon történő futtatásával lenne lehetőség.

4.2.3 A koptatóanyag súrlódási szögének hatása a kopás intenzitására

Elvégezve a súrlódási szög paraméterének megváltoztatásával kapcsolatos vizsgálatokat az eredmények azt mutatják, hogy a V3-as 34° súrlódási szöggel rendelkező koptató szemcsék rendelkeztek a legkisebb koptató hatással, ezt követi szorosan egymás mellett a V2-es 38°

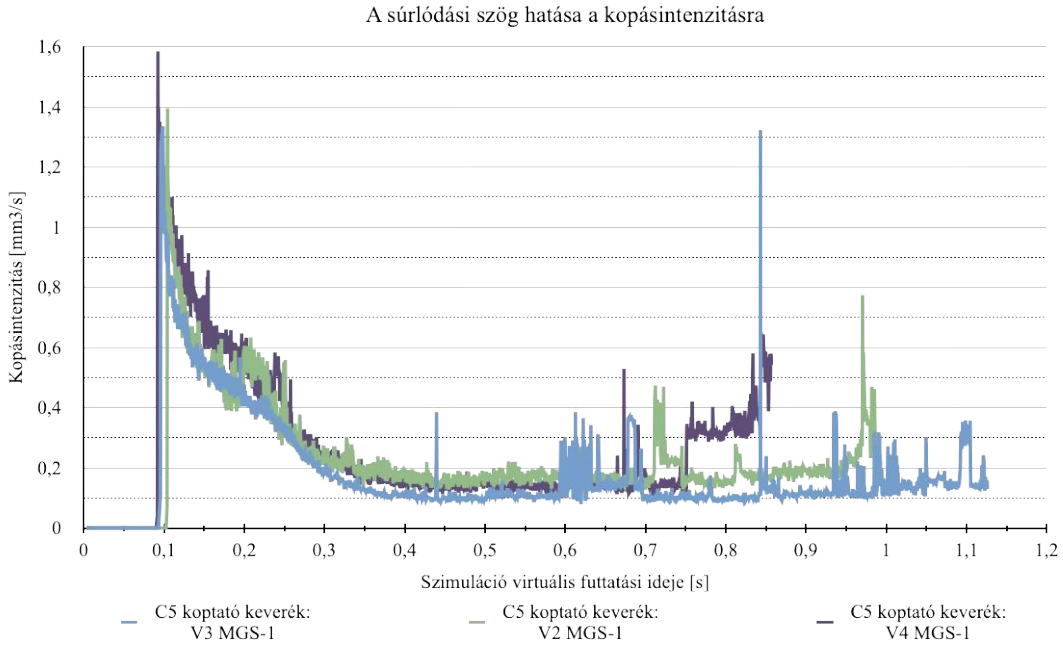
súrlódási szög beállítással rendelkező anyag, valamint a V4-es minta, amelynek súrlódási szöge 44°, ezen trendek jól megfigyelhetők az átlagos kopásintenzitás értékein a 47.ábrán.

47. ábra: Átlagos kopásintenzitás alakulása a koptatóanyag súrlódási szögének függvényében
(Forrás: Saját kép, szimulációs eredmények alapján)



A 48.ábrán megfigyelve a szimulációs idő függvényében ábrázolt pillanatnyi kopásintenzitás értékeit az látható, hogy a lila színnel jelölt V4-es variáns egy idő után jóval agresszívebben koptat, mint a másik két tulajdonságú anyag.

48. ábra: Pillanatnyi kopásintenzitási görbék a koptatóanyag súrlódási szögének függvényében
(Forrás: Saját kép, szimulációs eredmények alapján)

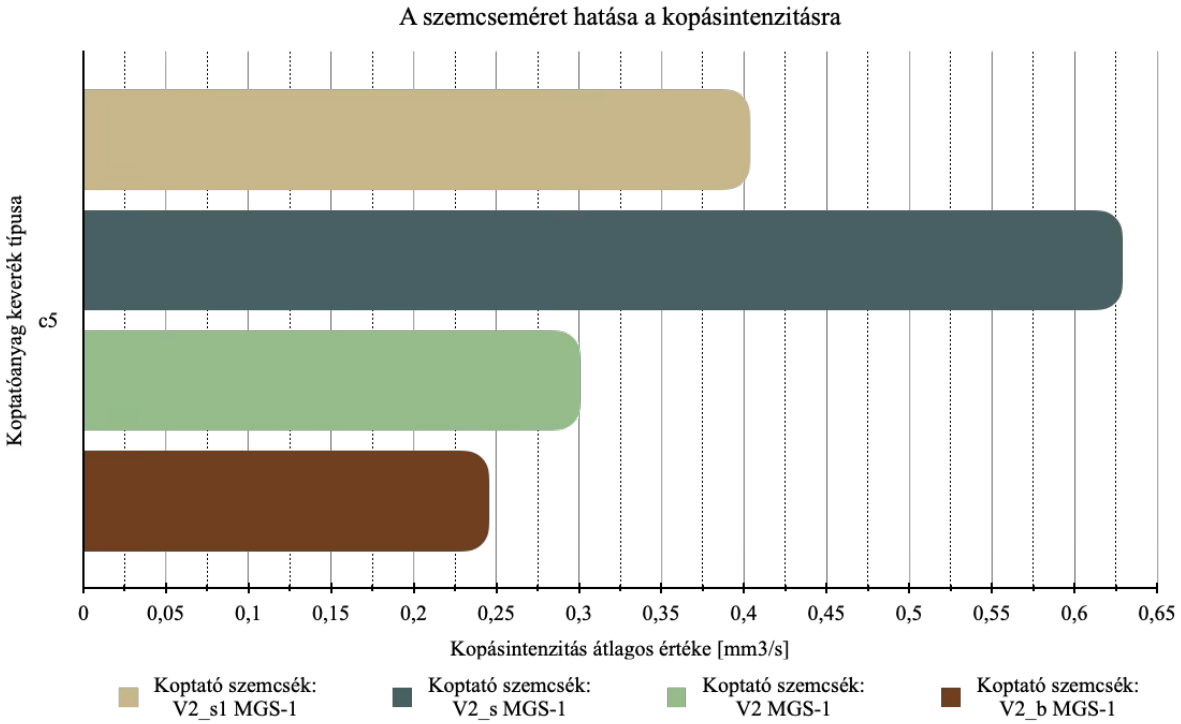


A kapott eredmények jól közelítenek a szakirodalomban leírtakkal, ugyanis általános megállapítás, hogy a magasabb súrlódási szög vagy más megfogalmazásban a súrlódási együttható értéke nagyobb elgördüléssel szembeni ellenállást eredményez a szemcsék esetén, amelyek így inkább barázdákat vágnak az érintkező felületbe, ezzel növelve a koptatás intenzitását. A szimuláció többszöri, hosszabb távú futtatása és a kapott eredmények átlagolása jelentősen javítaná a kapott eredmények pontosságát.

4.2.4 A szemcseméret hatása a kopás intenzitására

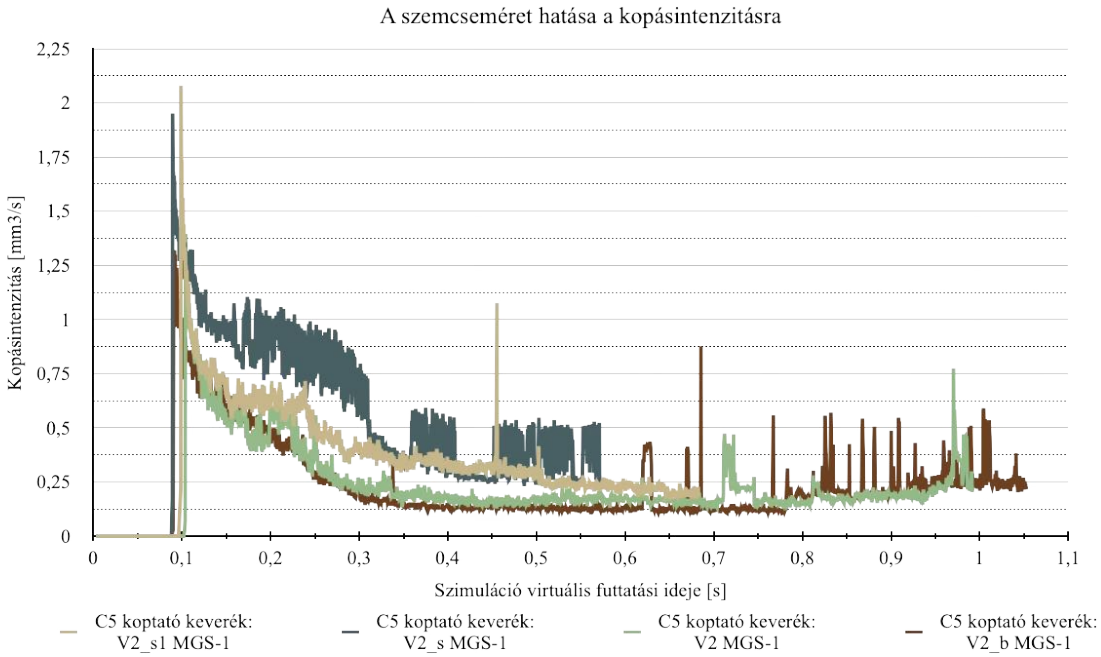
Végül pedig a koptatóanyag szemcseméret megváltozásnak a kopásintenzitásra gyakorolt hatását vizsgáltam. A kapott eredményekből megállapítható, hogy létezik egy olyan szemcseméret tartomány, amely a legoptimálisabb a koptatás hatásának maximalizálása szempontjából. Ezen ideális mérettartományon kívül eső, tehát az ennél kisebb és nagyobb szemcseméret is csökkenti a kopás erősségét. A 49.ábrán megjelenített diagram a szimuláció során az átlagos kopásintenzitási értékeket hasonlítja össze a szemcseméret függvényében. Jelen szimuláció esetén az optimális átmérő tartomány a V2_s MGS-1 szimulánst alkalmazva 80 – 85 µm közé esett, a második legintenzívebb koptatást a V2_s1 MGS-1anyag 60 – 65 µm méretű szemcséi okozták, ezt követi a V2 MGS-1 variáns 100 – 110 µm jellemző átmérővel, majd pedig a V2_b MGS-1 szimuláns 120 – 130 µm szemcseméret beállításokkal.

49. ábra: Átlagos kopásintenzitás értékek a koptatóanyag jellemző szemcseátmérője függvényében
(Forrás: Saját kép, szimulációs eredmények alapján)



Megvizsgálva az 50.ábrát, amely az egyes változatok pillanatnyi kopásintenzitásának idő szerinti alakulását mutatja, a korábban tett megállapítás itt is igazolható, ugyanakkor az is látható, hogy az idő előrehaladtával a trendek közelítenek egymáshoz.

50. ábra: Pillanatnyi kopásintenzitás alakulása a szemcseméret függvényében
(Forrás: Saját kép, szimulációs eredmények alapján)



A kapott eredmények a szakirodalomban leírtakkal jól összeegyeztethető, ugyanis az általános megállapítás szerint egy bizonyos mérettartomány alatt a szemcsék kisebb kopást generálnak, melynek egyik oka, hogy a kis méretű szemcsék inkább adhéziós módon koptatják a felületet, a felületen ejtett bemarások nem oly mélyek, mint nagyobb szemcseméret esetén, valamint fennáll a szemcsék felgyülemlésének lehetősége, amelyek kedvezőlenül hatnak a kopásra. Nagy általánosságban az optimálisnál nagyobb szemcseméret, ugyan nem oly jelentős mértékben, de tovább növeli a kopásintenzitás értékét (Trevisiol és munkatársai, 2017). Ennek szimulációval történő igazolására a vizsgálatok hosszabb távú és többszöri futtatásával lenne lehetőség.

5 KÖVETKEZTETÉSEK ÉS JAVASLATOK

A kapott eredmények alapján megállapítható, hogy az elkészített virtuális rézsűszög mérési és a kopás vizsgálati szimulációs módszerek képesek reprezentálni a valóságban mért és tapasztalt jelenségeket. A koptatóanyag paramétereinek egyesével történő megváltoztatása és az ebből következő koptatási eredmények jól közelítettek a szakirodalomban leírtakkal. Ezek alapján kimondható, hogy a Yade Dem szoftverbe az általam egyedileg fejlesztett koptatási mechanizmus képes a valósággal összeegyeztethető eredményeket adni, azonban számos olyan további fejlesztési lehetőség van a jelenlegi megvalósításban, amely az elkészített modell pontosságát képes lenne tovább növelni.

Jelen esetben annak érdekében, hogy a kopási vizsgálatok rövid idő alatt képesek legyenek lefutni olyan mértékű vizsgálati cél értéket határoztam, amely ezt lehetővé teszi, így az általam tesztelt futtatási idők alapján a definiált elérni kívánt térfogatveszteséget 0,2 mm^3 értékben határoztam meg. Kobrik és munkatársai (2010) által elvégzett laboratóriumi kopási vizsgálatok során acél próbatest esetén az elkoptatott térfogat maximális értéke 8 mm^3 volt. A kapott eredményekből és a szakirodalmi vonatkozások alapján a még pontosabb eredmények elérése érdekében javasolt a definiált cél anyagveszteség mértékét lényegesen tovább növelni, ezzel lehetővé téve a nagyobb mennyiségű adatok gyűjtését, ebből kifolyólag pedig a pontosabb adat kiértékeléseket.

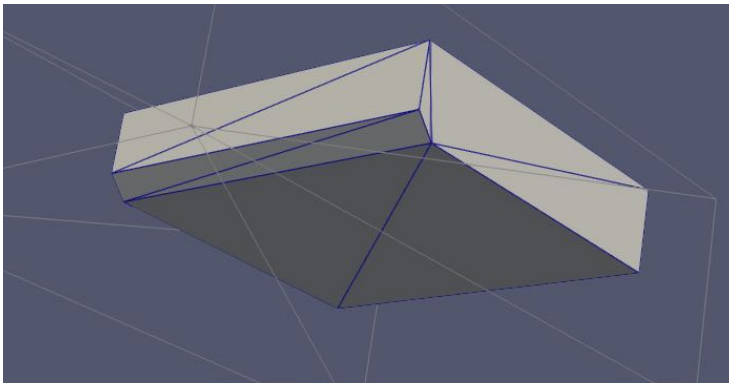
A futtatások ismétlési pontossága az általam megadott peremfeltételekkel körülbelül 15 – 25% nagyságúra adódott, amely érték jól közelít a szakirodalmakban leírt tapasztalt szimulációs eredmény ismétlési pontosságával, azonban hosszabb távú és a szimulációk többszöri újra futtatásával, majd pedig ezen adatok átlagolásával az említett pontosság tovább növelhető lenne.

Végeztem vizsgálatokat a szemcseméret tartomány megnövelésével is, amelyet 60 – 110 μm méretben határoztam meg, akkor azonban a kopás intenzitás értéke jelentősen lecsökkent. A jelenség abból következik, hogy a nagyobb méretű szemcsék kitámasztják a próbatestet, így a kisebb méretű részecskék minimális ellenállással haladnak el az alatt, rosszabb esetben egyáltalán nem érintkeznek a mintadarabal, így nem okozva kopást annak felületén. A valóságban ez a jelenség a koptató szemcsék kopásából, illetve aprózódásából következően másként zajlik. A koptató részecskék töredezése, mint funkció nem került kialakításra a szimulációban az erőforrásigény optimalizálása miatt, azonban egy lehetséges jövőbeli vizsgálati cél lehet.

Jelen kidolgozott módszer nem vizsgálja a kopás geometrián történő eloszlását, továbbá nem módosítja a próbatest alakját az anyagveszteség következtében. Ezen funkciók a szimulációba történő integrálása a program számítási igényét nagy mértékben befolyásolnák, így ezen megfontolások miatt általam nem kerültek megvalósításra.

További lehetséges továbbfejlesztési pont a próbatestet felépítő elemi háromszög felületek számának további növelés ezzel pedig az egységnyi területek méretének csökkentése, jelenleg ugyanis, ahogyan az az 51.ábrán látható a próbadarab felületei két elemi háromszögből épülnek fel.

51. ábra: Elkoptatandó próbatest elemi háromszög kialakítása
(Forrás: Saját kép)



Ezzel a módosítással a vizsgálat felbontása nagyban növelhető lenne, továbbá így alkalmassá válna a kopás eloszlás pontos kiszámítására szimuláció. Azonban fontos megemlíteni, hogy az interakciókban résztvevő felületek számának növelése jelentősen megnöveli az iterációnkénti számításokat, amely nagy mértékben lassítja a futtatási sebességet, ebből kifolyólag esetemben a legrosszabb elérhető felbontás mellett döntöttem a számítási igény javára.

6 ÖSSZEFOGLALÁS

Dolgozatom elkészítésével azt a célt tűztem ki, hogy létrehozzak egy olyan diszkrét elemes szimulációt, amely képes a kopást, mint jelenséget vizsgálni egy olyan nyílt forráskódú környezetben, mint a Yade Dem. A program sajátossága, hogy az beépített kopási modullal nem rendelkezik, így ezen funkciót teljes egészében egyedileg hoztam létre és integráltam szoftverbe. A módszer kidolgozását az űrkutatásban megismert holdi és marsi talajok okozta agresszív koptatási hatások inspirálták, ugyanis az elkészített szimulációval lehetőség nyílik ezen nehezen hozzáférhető koptatóanyagok virtuális vizsgálatára.

A kidolgozás során először megismertem a súrlódással és a kopással kapcsolatos jelenségeket, megvizsgáltam a gyakorlatban használatos laboratóriumi koptatási eljárásokat, továbbá felkutattam és kielemeztem a jelenleg elérhető numerikus módszerekkel elvégzett virtuális koptatási kísérleteket. Ezt követően feltártam a holdi és marsi talajok egyedi anyag és geometriai jellemzőit, ezen kutatások során megismertem a talajok rézsűszögének vizsgálati módszerét, amelyet aztán a saját munkám során én is alkalmaztam.

Két külön szimulációt készítettem, amelyek közül az első alkalmas a koptató szemcsék rézsűszögének vizsgálatára, a második pedig a dolgozat célkitűzésében is meghatározott koptatási folyamatot képes modellezni. Az első egy mindkét végén nyitott cső szemcsékkel való megtöltését, azok ülepedését, a cső felemelését és a szabad szemcsehalmaz újbóli ülepedését foglalja magában. Utóbbi esetén a teszt környezet a Pin – on – Disk laboratóriumi kísérletet veszi alapul kiegészítve azt egy abrazív szemcse folyammal, a tárcsa és a próbadarab közé szórva. A kopási mechanizmust az Archard féle összefüggés alapján integráltam a szimulációba.

Lefuttatva a szimulációkat azt az eredményt kaptam, hogy a rézsűszög mért értékei megegyeznek a szakirodalomban definiáltakkal. A koptatási vizsgálatok során a koptató anyag beállítási paramétereit egyesével megváltoztattam és vizsgáltam azok koptatás intenzitásra gyakorolt hatását. A módosított és összehasonlított tulajdonságok közé tartozott a koptató anyag szemcseszerkezete, annak sűrűsége, a súrlódási szög értéke és az alkotó szemcsék mérete. A kapott eredmények alapján kimondható, hogy a szimuláció az alap kopási jelenségeket, mint például a térfogatveszteség lineáris karakterisztikája, vagy a koptatási intenzitás kis mértékű csökkenése az idő függvényében jól reprezentálta, ebből következően pedig a paraméter vizsgálati eredmények is nagy egyezőséget mutattak a szakirodalmakban leírt jelenségekkel.

7 SUMMARY

The objective of my thesis was to develop a discrete element simulation to investigate wear as a phenomenon using an open-source framework called Yade Dem. The program does not have a built-in wear module, therefore I created this function entirely and integrated it into the software. The development of the method was inspired by the aggressive abrasive effects of Lunar and Martian soils known from space research, as the simulation allows for the virtual examination of these hard-to-access abrasive regoliths.

Throughout the development process, I initially acquired knowledge regarding methodologies associated with friction and abrasion, examined the laboratory abrasion procedures used in practice, and investigated and evaluated virtual abrasion tests performed with currently available numerical methods. Then I examined the unique material and geometric properties of Lunar and Martian soils. During this research, I gained insight on the method of testing the angle of repose of soils, which I then implemented in my own project.

I developed two distinct simulations, the first is designed to investigate the angle of repose of abrasive medium, and the second is intended to represent the abrasion process outlined in the thesis aim. The initial simulation is filling a tube that is open at both ends with particles, providing them to settle, elevating the tube, and then allowing the loose particle mixture to settle once more. For the second simulation, the testing environment is derived from the Pin-on-Disk laboratory experiment, enhanced with an abrasive particle flow distributed between the disc and the test specimen. I integrated the wear mechanism into the simulation utilizing Archard's correlation.

Upon performing the simulations, I acquired findings indicating that the measured static angle values align with those stated in the literature. In the abrasion test, I systematically adjusted the properties of the abrasive material individually and analyzed their impact on abrasion intensity. The properties that were adjusted and compared were the particle structure of the abrasive material, its density, the friction angle, and the size of the constituent particles. The results indicate that the simulation accurately represented fundamental wear phenomena, including the linear relationship of volume loss and the gradual reduction in wear intensity over time. Consequently, the parameter test results exhibited a substantial level of alignment with the phenomena stated in the literature.

8 IRODALOMJEGYZÉK

1. A. Al-Samarai, Riyadh & Al-Douri, Yarub. (2024). Friction and Wear in Metals. 10.1007/978-981-97-1168-0.
2. Al-Hashemi, Hamzah & Al-Amoudi, Omar. (2018). A review on the angle of repose of granular materials. Powder Technology. 330. 397-417. 10.1016/j.powtec.2018.02.003.
3. Bhushan B. (2013). Introduction to tribology: John Wiley & Sons
4. Boemer, D. (2015). Discrete Element Method Modeling of Ball Mills Liner Wear Evolution.
5. Boemer, D., & Ponthot, J.-P. (2017). A generic wear prediction procedure based on the discrete element method for ball mill liners in the cement industry. Minerals Engineering, 109, 55–79. https://doi.org/10.1016/J.MINENG.2017.02.014
6. C. Trevisiol, A. Jourani, Salima Bouvier. (2017). Effect of abrasive particle size on friction and wear behaviour of various microstructures of 25CD4 steel. Journal of Physics: Conference Series, 843, pp.012073.
7. Capozzi, Luigi & Barresi, Antonello & Pisano, Roberto. (2018). Supporting data and methods for the multi-scale modelling of freeze-drying of microparticles in packed-beds. Data in Brief. 22. 10.1016/j.dib.2018.12.061.
8. Dreyer, Mathias Roman & Gergye, Tamás. (2012). Kopási folyamatok online mérése radionukleációs technika (RNT) segítségével.
9. Faur, K. B., & Szabó, I..(2011). Geotechnika 1
10. Fazekas, L., Deák, K., & Menyhárt, J..(2018). Gépészeti rendszerek károsodása, javítástechnológiája és karbantartása.
11. Gant, Andrew & Gee, Mark. (2006). Abrasion of tungsten carbide hardmetals using hard counterfaces. International Journal of Refractory Metals & Hard Materials - INT J REFRACT MET HARD MATER. 24. 189-198. 10.1016/j.ijrmhm.2005.05.007.
12. Gee, M G; Owen-Jones, S (1998) Wear testing methods and their relevance to industrial wear problems. NPL Report. CMMT(A)92
13. Gee, Mark & Gant, Andrew & Hutchings, I.M. & Kusano, Yukihiro & Schiffmann, Kirsten & Van Acker, Karel & Poulat, S. & Gachon, Yves & Stebut, J. & Hatto, Peter & Plint, George. (2005). Results from an interlaboratory exercise to validate the micro-scale abrasion test. Wear. 259. 27-35. 10.1016/j.wear.2005.02.092.

14. Gopaul, K., Venkannah, S., & Dauhoo, M. Z. (2022). Wear and flow rate problems on chutes in a rock crushing plant. Tribology and Materials, 1(4), 128–137. https://doi.org/10.46793/tribomat.2022.016
15. Grasser, D., Corujeira Gallo, S., Pereira, M.P., & Barnett, M. (2024). Wear simulation and validation of composites (insert-reinforced matrix) in the dry sand rubber wheel test. Minerals Engineering.
16. Huggett, Richard. (2023). Regolith or soil? An ongoing debate. Geoderma. 432. 10.1016/j.geoderma.2023.116387.
17. Hutchings, I. & Shipway, P.. (2017). Tribology: Friction and wear of engineering materials: Second Edition.
18. Jakus, Adam & Koube, Katie & Geisendorfer, Nicholas & Shah, Ramille. (2017). Robust and Elastic Lunar and Martian Structures from 3D-Printed Regolith Inks. Scientific Reports. 7. 44931. 10.1038/srep44931.
19. Kalácska, Gábor & Barko, Gyorgy & Shegawu, Hailemariam & Kalácska, Ádám & Zsidai, L. & Keresztes, Róbert & Károly, Zoltán. (2024). The Abrasive Effect of Moon and Mars Regolith Simulants on Stainless Steel Rotating Shaft and Polytetrafluoroethylene Sealing Material Pairs. Materials. 17. 4240. 10.3390/ma17174240.
20. Kobrick, Ryan & Budinski, Kenneth & Street, Kenneth & Klaus, David. (2010). Three-Body Abrasion Testing using Lunar Dust Simulants to Evaluate Surface System Materials. 40th International Conference on Environmental Systems, ICES 2010. 10.2514/6.2010-6077.
21. Kotrocz, Krisztián és Kerényi, György (2018) Analysation of the effect of the contact parameters on soil's shear behaviour in discrete element simulations. HUNGARIAN AGRICULTURAL ENGINEERING (34). pp. 5-10. ISSN 0864-7410
22. la Monaca, Andrea & Murray, James & Liao, Zhirong & Speidel, Alistair & Robles Linares Alvelais, José & Axinte, Dragos & Hardy, Mark & Clare, Adam. (2021). Surface integrity in metal machining - Part II: Functional performance. International Journal of Machine Tools and Manufacture. 10.1016/j.ijmachtools.2021.103718.
23. Long-Fox, Jared & Britt, Daniel. (2023). Characterization of planetary regolith simulants for the research and development of space resource technologies. Frontiers in Space Technologies. 4. 10.3389/frspt.2023.1255535.

24. Lu, Chieh-Jung & Yeh, Jien-Wei. (2023). Improved Wear and Corrosion Resistance in TiC-Reinforced SUS304 Stainless Steel. Journal of Composites Science. 7. 34. 10.3390/jcs7010034.
25. Lyu, Xiongfei & Chen, You & Liao, Shao-ming & Fernandez-Steeger, Tomas. (2025). Wear Prediction and Mechanism Study of Tunnel Boring Machine Disc Cutter Breaking in Hard–Soft Rock Considering Thermal Effect. Applied Sciences. 15. 4183. 10.3390/app15084183.
26. Metcalf, J.R. (1966). Angle of repose and internal friction. International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts, 3, 155-161.
27. Morgan, Paul & Grott, Matthias & Knapmeyer-Endrun, Brigitte & Golombek, Matt & Delage, Pierre & Lognonné, Philippe & Piqueux, Sylvain & Daubar, Ingrid & Murdoch, Naomi & Charalambous, Constantinos & Pike, William & Mueller, Nils & Hagermann, Axel & Siegler, Matt & Lichtenheldt, Roy & Teanby, Nick & Kedar, Sharon. (2018). A Pre-Landing Assessment of Regolith Properties at the InSight Landing Site. Space Science Reviews. 214. 10.1007/s11214-018-0537-y.
28. Nababan, Deddy & Pownceby, Mark & Rhamdhani, M.. (2025). Metals extraction on Mars through carbothermic reduction: Mars regolith simulant (MGS-1) characterization and preliminary reduction experiments. Acta Astronautica. 234. 10.1016/j.actaastro.2025.04.050.
29. Radjai, F., & Azéma, E. (2009). Shear strength of granular materials. European Journal of Environmental and Civil Engineering, 13(2), 203–218. https://doi.org/10.1080/19648189.2009.9693100
30. Ramakrishnan Easwar, V. (2020). An Investigation into ‘Squeezing’or Three-Body Wear caused by Abrasive Particles (Dissertation). Retrieved from https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-283552
31. Rankin, Arturo & Maimone, Mark & Biesiadecki, Jeffrey & Patel, Nikunj & Levine, Dan & Toupet, Olivier. (2020). Driving Curiosity: Mars Rover Mobility Trends During the First Seven Years. 1-19. 10.1109/AERO47225.2020.9172469.
32. Rojas Parra, Eduardo & Vergara, Victor & Soto, Rodrigo. (2019). Case study: Discrete element modeling of wear in mining hoppers. Wear. 430. 10.1016/j.wear.2019.04.020.
33. Šmilauer, V., Angelidakis, V., Catalano, E., Caulk, R., Chareyre, B., Chèvremont, W., Dorofeenko, S., Duriez, J., Dyck, N., Eliáš, J., Er, B., Eulitz, A., Gladky, A., Guo, N., Jakob, C., Kneib, F., Kozicki, J., Marzougui, D., Maurin, R., ... Yuan, C. (2021). Yade

documentation: the Yade Project. (3rd ed.) Zenodo. https://doi.org/10.5281/ZENODO.5705394

34. Sukumaran, Abhijith & Zhang, Cheng & Nisar, Ambreen & Agarwal, Arvind. (2023). Recent Trends in Tribological Performance of Aerospace Materials in Lunar Regolith Environment - A Critical Review. Advances in Space Research. 73. 10.1016/j.asr.2023.10.039.
35. Suresh, R. & Kumar, M. & Basavarajappa, Satyappa & Kiran, Dr & Yeole, Mahesh & Katare, Naresh. (2017). Numerical Simulation & Experimental study of wear depth and Contact pressure distribution Of Aluminum MMC Pin on Disc Tribometer.. Materials Today: Proceedings. 4. 11218-11228. 10.1016/j.matpr.2017.09.043.
36. Swain, Biswajit & Bhuyan, Subrat & Behera, Rameswar & Mohapatra, Soumya & Behera, Ajit. (2020). Wear: A Serious Problem in Industry. 10.5772/intechopen.94211.
37. Wainaina Kimani, H., Kimotho Kuria, J., & Mutua, J. (2024). Investigation of Liner Wear on Different Ball Mill Profiles Using Discrete Element Method. JOURNAL OF SUSTAINABLE RESEARCH IN ENGINEERING, 8(1), 68-84. Retrieved from https://jsre.jkuat.ac.ke/index.php/jsre/article/view/186
38. Xia, Rui & Wang, Xuewen & Li, Bo & Wei, Xing & Yang, Zhaojian. (2019). Discrete Element Method- (DEM-) Based Study on the Wear Mechanism and Wear Regularity in Scraper Conveyor Chutes. Mathematical Problems in Engineering. 2019. 1-12. 10.1155/2019/4191570.
39. Xia, Rui & Wang, Xuewen & Li, Bo & Wei, Xing & Yang, Zhaojian. (2019). Discrete Element Method- (DEM-) Based Study on the Wear Mechanism and Wear Regularity in Scraper Conveyor Chutes. Mathematical Problems in Engineering. 2019. 1-12. 10.1155/2019/4191570.
40. Yan, Liang & Guan, Linyi & Wang, Di & Xiang, Dingding. (2024). Application and Prospect of Wear Simulation Based on ABAQUS: A Review. Lubricants. 12. 57. 10.3390/lubricants12020057.
41. Yan, Yunpeng & Helmons, Rudy & Carr, Michael & Wheeler, Craig & Schott, Dingena. (2022). Modelling of material removal due to sliding wear caused by bulk material. Powder Technology. 415. 118109. 10.1016/j.powtec.2022.118109.
42. Zhao, Zhihao & Wu, Mingliang & Jiang, Xiaohu. (2024). A Review of Contact Models' Properties for Discrete Element Simulation in Agricultural Engineering. Agriculture. 14. 238. 10.3390/agriculture14020238.

43. Zhou, Ningxi & Chen, Jian & Tian, Ning & Huang, Juehao & Wu, Peng. (2024). Calibration of Discrete Element Method Parameters for a High-Fidelity Lunar Regolith Simulant Considering the Effects of Realistic Particle Shape. Materials. 17. 4789. 10.3390/ma17194789.
44. Zhu, Jianzhong & Zou, Meng & Liu, Yansong & Gao, Kai & Su, Bo & Qi, Yingchun. (2021). Measurement and calibration of DEM parameters of lunar soil simulant. Acta Astronautica. 191. 10.1016/j.actaastro.2021.11.009.
45. Zmitrowicz, Alfred. (2006). Wear patterns and laws of wear-a review. JOURNAL OF THEORETICAL AND APPLIED MECHANICS. 44. 219-253.
46. Altair rézsűszög beállítási útmutató, webhely: https://community.altair.com/discussion/38721/material-model-calibration-using-static-angle-of-repose-test-with-edem/p1
47. MGS-1 Spec Sheet 003-05-001-0523, wehbely: https://cdn.shopify.com/s/files/1/0398/9268/0862/files/MGS-1_SPEC_SHEET_DEC_2023.pdf?v=1745852729
48. S420 acél adatlapja, webhely: https://metalzenith.com/blogs/steel-properties/s420-steel-properties-and-key-applications-overview

9 ÁBRÁK ÉS TÁBLÁZATOK JEGYZÉKE

9.1 Ábrajegyzék

10 MELLÉKLETEK

I. Melléklet: Rézsűszög vizsgálati szimuláció script

```
# Angle of repose simulation

# 1. Imports
from yade import utils, geom
from yade import Vector3
import numpy as np
import math
import random
import matplotlib.pyplot as plt
from matplotlib import cm
from yade import pack
import time
import csv
from datetime import datetime
import os

# 2. Main parameters
rMin = 0.05e-3
rMax = 0.055e-3
clumpTypePercentages = [20, 20, 20, 20, 20]

tube_diameter = 5e-3
tube_height = 10e-3
tube_center = Vector3(0, 0, tube_height/2)
ground_size = 15e-3

fill_height = 10e-3
packing_density = .8
sphere_packing_radius = (rMin + rMax) / 2
min_distance_between_clumps = 1.5 * rMax

unbalanced_force_threshold = 0.12
lifting_speed = 20e-4

# 3. Material definition
shotsId, steelId = O.materials.append([
    FrictMat(young=40e6, density=1300, poisson=.25, frictionAngle=math.radians(38), label='shots'),
    FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(30), label='steel'),
])
tubeId = O.materials.append(
    FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(10), label='tube_low_friction') # Lowe
)

# 4. Data handling
clump_size_data = []
size_distribution_bins = 20
time_series_data = []

# 5. Unbalanced force and RandomRadius
def calculateUnbalancedForce():
    try:
        unbalanced = utils.unbalancedForce()
        return unbalanced
    except:
        return 1.0

def getRandomRadius():
    return random.uniform(rMin, rMax)

# 6. Simulation state
class FineParticleState:
    def __init__(self):
        self.phase = "PACKING"
        self.currentMass = 0.0
        self.currentVolume = 0.0
        self.clumpsGenerated = 0
        self.particleCount = 0
        self.phase_start_time = 0.0
        self.tube_lift_start_z = tube_center[2]

        self.unbalanced_force_history = []
        self.force_check_interval = 1000
        self.stable_force_count = 0
        self.required_stable_checks = 8

        self.packing_complete = False
        self.clump_ids = []

        self.initial_settling_time = 3.0

state = FineParticleState()

# 7. Clump related functions
def createClumpTemplates():
    c1 = [((0, 0, 0), 0.75), ((1.2, 0, 0), 1.0)]
    c2 = [((0, 0, 0), 0.75), ((1.1, 0, 0), 0.75), ((0.55, 1.2, 0), 1.0)]
    c3 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 1.0)]
    c4 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 0.9), ((0.4, 0.3, -1.0), 0.9
        ((0, 0, 0.8), 0.6), ((0.8, 0, 0.8), 0.6), ((0, 0.8, 0.8), 0.6), ((0.8, 0.8, 0.8), 0.6)]
    return [c1, c2, c3, c4, c5]

clumpTemplates = createClumpTemplates()

def createClump(clump_type, center_pos):
    template = clumpTemplates[clump_type]
    base_radius = getRandomRadius()

    center_x = center_pos[0]
    center_y = center_pos[1]
    center_z = center_pos[2]

    colors = [(0.2, 0.8, 0.2), (0.8, 0.2, 0.2), (0.2, 0.2, 0.8), (0.8, 0.8, 0.2), (0.8, 0.2, 0.8)]
    color = colors[clump_type]

    spheres = []
    total_volume = 0
    max_radius = 0

    for rel_pos, radius_factor in template:
        sphere_radius = base_radius * radius_factor
        max_radius = max(max_radius, sphere_radius)
        abs_x = center_x + rel_pos[0] * base_radius
        abs_y = center_y + rel_pos[1] * base_radius
        abs_z = center_z + rel_pos[2] * base_radius

        s = utils.sphere((abs_x, abs_y, abs_z), sphere_radius, material=shotsId)
        s.color = color
        s.mask = 3
        spheres.append(s)
        total_volume += (4/3) * math.pi * (sphere_radius**3)

    # Track size distribution
    equivalent_radius = ((3 * total_volume) / (4 * math.pi)) ** (1/3)
    clump_size_data.append((base_radius, clump_type, equivalent_radius, total_volume, max_radius))

    total_mass = total_volume * O.materials[shotsId].density
    return spheres, total_mass, total_volume, base_radius, max_radius

# 8. Ground and tube creation
def createSimpleFlatGround():
    ground_thickness = 4e-3

    ground = utils.box(
        center=(0, 0, -ground_thickness/2),
        extents=(ground_size/2, ground_size/2, ground_thickness/2),
        material=steelId,
        fixed=True,
        color=(0.1, 0.1, 0.1)
    )

    ground_id = O.bodies.append(ground)
    print(f"Ground created: {ground_size*1000:.1f} x {ground_size*1000:.1f} mm")
    return ground_id

def createOpenEndedTube(center, radius, height, material, segments=16):
    tube_bodies = []

    bottom_vertices = []
    top_vertices = []

    for i in range(segments):
        angle = 2 * math.pi * i / segments
        x = center[0] + radius * math.cos(angle)
        y = center[1] + radius * math.sin(angle)

        z_bottom = center[2] - height/2
        bottom_vertices.append(Vector3(x, y, z_bottom))

        z_top = center[2] + height/2
        top_vertices.append(Vector3(x, y, z_top))

    for i in range(segments):
        next_i = (i + 1) % segments

        facet1 = utils.facet([bottom_vertices[i], bottom_vertices[next_i], top_vertices[i]],
```



```
                             material=material, color=(0.9, 0.9, 1.0))
        facet2 = utils.facet([bottom_vertices[next_i], top_vertices[next_i], top_vertices[i]],
                             material=material, color=(0.9, 0.9, 1.0))

        tube_bodies.append(O.bodies.append(facet1))
        tube_bodies.append(O.bodies.append(facet2))

    print(f"Tube created: {radius*1000:.1f} mm radius, {height*1000:.1f} mm height")
    return tube_bodies

# 9. Initialization
ground_id = createSimpleFlatGround()
tube_radius = tube_diameter / 2
tube_bodies = createOpenEndedTube(tube_center, tube_radius, tube_height, tubeId)

# 10. Cylindrical packing
def createCylindricalPacking():

    # Calculate clump target number
    tube_radius = tube_diameter / 2
    cylinder_volume = math.pi * (tube_radius * 0.85)**2 * fill_height
    avg_clump_volume = (4/3) * math.pi * sphere_packing_radius**3 * 2.5
    target_num_clumps = int(cylinder_volume * packing_density / avg_clump_volume)

    print(f"Target clumps: {target_num_clumps}")
    print(f"Cylinder radius: {tube_radius*1000:.2f} mm")
    print(f"Fill height: {fill_height*1000:.1f} mm")

    cylinder_pred = pack.inCylinder(
        (0, 0, 0),                  # bottom center
        (0, 0, fill_height * 0.9),  # top center (90% fill)
        tube_radius * 0.85          # radius (85% of tube radius)
    )

    sp = pack.SpherePack()
    sp.makeCloud(
        minCorner=Vector3(-tube_radius, -tube_radius, 0),
        maxCorner=Vector3(tube_radius, tube_radius, fill_height * 0.9),
        rMean=sphere_packing_radius,
        rRelFuzz=0.05,
        num=target_num_clumps * 2,
        periodic=False,
        porosity=1-packing_density
    )

    filtered_sp = pack.SpherePack()
    for center, radius in sp:
        if cylinder_pred(center):
            filtered_sp.add(center, radius)

    sp = filtered_sp
    if len(sp) == 0:
        print("Clumps cant be generated")
        return 0, 0, 0, []

    print("Creating clumps")
    clumps_created = 0
    total_mass = 0
    total_volume = 0
    clump_ids = []

    for i, (center, radius) in enumerate(sp):
        # Select clump type based on percentages
        clump_type = random.choices([0, 1, 2, 3, 4], weights=clumpTypePercentages, k=1)[0]

        # Create clump
        spheres, clump_mass, clump_volume, base_radius, max_radius = createClump(clump_type, center)

        # Add to simulation
        clpId, sphId = O.bodies.appendClumped(spheres)

        # Set initial conditions
        clump_body = O.bodies[clpId]
        clump_body.state.vel = Vector3(0, 0, 0)
        clump_body.state.angVel = Vector3(0, 0, 0)
        clump_body.mask = 3

        # Set sphere properties
        for sphere_id in sphId:
            if O.bodies[sphere_id]:
                O.bodies[sphere_id].state.vel = Vector3(0, 0, 0)
                O.bodies[sphere_id].state.angVel = Vector3(0, 0, 0)

        clump_ids.append(clpId)
        total_mass += clump_mass
        total_volume += clump_volume
        clumps_created += 1

        if (i + 1) % 25 == 0 or i == len(sp) - 1:
            print(f"  Created {i + 1}/{len(sp)} clumps...")

    print(f" Packing completed ")
    print(f"Clumps created: {clumps_created}")
    print(f"Total mass: {total_mass*1000:.3f}g")
    print(f"Total volume: {total_volume*1e9:.2f}mm³")

    print(f"Size data collected: {len(clump_size_data)} entries")
    update_plot()

    return clumps_created, total_mass, total_volume, clump_ids

# 11. Plotting
try:
    plt.ion()
    plt.style.use('seaborn-v0_8-whitegrid')
except:
    try:
        plt.style.use('seaborn-whitegrid')
    except:
        pass

COLORS = {
    'primary': '#7FB3D3',
    'secondary': '#B8E6B8',
    'accent': '#FFD1DC',
    'warning': '#FFEB9C',
    'info': '#D1C4E9',
    'grid': '#2C3E50',
    'text': '#2C3E50'
}

CLUMP_COLORS = ['#A8D8EA', '#FFB3BA', '#BFFF7F', '#FFE5B4', '#D7B3FF', '#FFC3A0']

fig = plt.figure(figsize=(14, 10))
gs = fig.add_gridspec(2, 2, height_ratios=[1, 1], width_ratios=[1, 1])

ax1 = fig.add_subplot(gs[0, 0])  # Top left: Equivalent diameter distribution
ax2 = fig.add_subplot(gs[1, 0])  # Bottom left: Cumulative size distribution

ax3 = fig.add_subplot(gs[:, 1])  # Right side: Clump type distribution

fig.suptitle('Angle of Repose Simulation - Size Distribution Analysis', fontsize=16, fontweight='bold')

plt.rcParams.update({
    'font.size': 10,
    'axes.grid': True,
    'grid.alpha': 0.6,
    'grid.color': '#2C3E50'
})

ax1.set_title('Equivalent Diameter Distribution', fontweight='bold')
ax1.set_xlabel('Equivalent Diameter (um)')
ax1.set_ylabel('Frequency')
ax1.grid(True)

ax2.set_title('Cumulative Size Distribution', fontweight='bold')
ax2.set_xlabel('Equivalent Diameter (um)')
ax2.set_ylabel('Cumulative Percentage Finer (%)')
ax2.grid(True)

ax3.set_title('Clump Type Distribution', fontweight='bold')

plt.tight_layout()
plt.show(block=False)

last_plot_update_iter = 0
plot_update_interval = 200

def update_plot():
    global last_plot_update_iter

    try:
        print(f"Updating plots with {len(clump_size_data)} data points")

        update_size_distribution_plots()
```

```
            fig.canvas.draw()
            fig.canvas.flush_events()
            plt.pause(0.01)
        except:
            pass

    except Exception as e:
        print(f"Plot update error: {e}")

def update_size_distribution_plots():
    if len(clump_size_data) < 1:
        print("No size data for plotting")
        return

    try:
        equiv_diameters_um = [data[2] * 2 * 1e6 for data in clump_size_data]
        clump_types = [data[1] for data in clump_size_data]

        print(f"Processing {len(equiv_diameters_um)} diameter values")
        print(f"Diameter range: {min(equiv_diameters_um):.1f} - {max(equiv_diameters_um):.1f} um")

        ax1.clear()
        ax2.clear()
        ax3.clear()

        # Calculate percentiles
        d10 = np.percentile(equiv_diameters_um, 10)
        d50 = np.percentile(equiv_diameters_um, 50)
        d90 = np.percentile(equiv_diameters_um, 90)
        mean_diameter = np.mean(equiv_diameters_um)

        # 1. Equivalent diameter distribution
        ax1.hist(equiv_diameters_um, bins=30, alpha=0.8, color=COLORS['primary'], edgecolor='black')
        ax1.axvline(d10, color='red', linestyle='--', linewidth=2, label=f'D10 = {d10:.1f} um')
        ax1.axvline(d50, color='orange', linestyle='--', linewidth=2, label=f'D50 = {d50:.1f} um')
        ax1.axvline(d90, color='red', linestyle='--', linewidth=2, label=f'D90 = {d90:.1f} um')
        ax1.axvline(mean_diameter, color='purple', linestyle='-', linewidth=2, label=f'Mean = {mean_diameter:.1f} um'
        ax1.set_xlabel('Equivalent Diameter (um)')
        ax1.set_ylabel('Frequency')
        ax1.set_title('Equivalent Diameter Distribution', fontweight='bold')
        ax1.legend()
        ax1.grid(True)

        # 2. Cumulative distribution
        sorted_diameters = np.sort(equiv_diameters_um)
        cumulative_percent = np.arange(1, len(sorted_diameters) + 1) / len(sorted_diameters) * 100

        ax2.plot(sorted_diameters, cumulative_percent, color=COLORS['primary'], linewidth=2)
        ax2.axhline(10, color='red', linestyle='--', alpha=0.7)
        ax2.axhline(50, color='orange', linestyle='--', alpha=0.7)
        ax2.axhline(90, color='red', linestyle='--', alpha=0.7)
        ax2.axvline(d10, color='red', linestyle='--', alpha=0.7, label=f'D10 = {d10:.1f} um')
        ax2.axvline(d50, color='orange', linestyle='--', alpha=0.7, label=f'D50 = {d50:.1f} um')
        ax2.axvline(d90, color='red', linestyle='--', alpha=0.7, label=f'D90 = {d90:.1f} um')
        ax2.plot(d10, 10, 'o', color='red', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax2.plot(d50, 50, 'o', color='orange', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax2.plot(d90, 90, 'o', color='red', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax2.set_xlabel('Equivalent Diameter (um)')
        ax2.set_ylabel('Cumulative Percentage Finer (%)')
        ax2.set_title('Cumulative Size Distribution', fontweight='bold')
        ax2.legend()
        ax2.grid(True)
        ax2.set_ylim(0, 100)

        # 3. Clump type pie chart
        type_counts = [clump_types.count(i) for i in range(5)]
        type_percentages = [count/len(clump_types)*100 for count in type_counts]
        active_types = [(i, pct) for i, pct in enumerate(type_percentages) if pct > 0]

        if active_types:
            active_indices = [item[0] for item in active_types]
            active_percentages = [item[1] for item in active_types]
            active_labels = [f'Type {i+1}' for i in active_indices]
            active_colors = [CLUMP_COLORS[i] for i in active_indices]

            ax3.pie(active_percentages, labels=active_labels, colors=active_colors,
                    autopct='%1.1f%%', startangle=90, textprops={'fontsize': 12})
            ax3.set_title('Clump Type Distribution', fontweight='bold', fontsize=14)
        else:
            ax3.text(0.5, 0.5, 'No Data Available', ha='center', va='center',
                     transform=ax3.transAxes, fontsize=14, fontweight='bold')
            ax3.set_title('Clump Type Distribution', fontweight='bold', fontsize=14)

        plt.tight_layout()
        print("Plots updated successfully")

    except Exception as e:
        print(f"Size distribution plot error: {e}")

# 12. .CSV data export
def recordTimeSeriesData():
    try:
        unbalanced_force = calculateUnbalancedForce()
        current_phase = state.phase if 'state' in globals() else "INITIALIZATION"

        time_series_data.append({
            'iteration': O.iter,
            'simulation_time_s': O.time,
            'unbalanced_force': unbalanced_force,
            'phase': current_phase,
            'total_clumps': len(clump_size_data),
            'timestep_s': O.dt
        })

        if len(time_series_data) > 10000:
            time_series_data.pop(0)

    except Exception as e:
        print(f"Time series recording error: {e}")

def completeSimulationExport():
    export_dir = "complete_simulation_export_angle_repose"
    if not os.path.exists(export_dir):
        os.makedirs(export_dir)

    timestamp = f"{O.iter:06d}_{int(O.time*1000):06d}"
    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    if clump_size_data:
        equiv_diameters_um = [data[2] * 2 * 1e6 for data in clump_size_data]
        d10 = np.percentile(equiv_diameters_um, 10)
        d50 = np.percentile(equiv_diameters_um, 50)
        d90 = np.percentile(equiv_diameters_um, 90)
        mean_diameter = np.mean(equiv_diameters_um)
        std_diameter = np.std(equiv_diameters_um)
        min_diameter = min(equiv_diameters_um)
        max_diameter = max(equiv_diameters_um)

        clump_types = [data[1] for data in clump_size_data]
        type_counts = [clump_types.count(i) for i in range(5)]
        type_percentages = [count/len(clump_types)*100 for count in type_counts]
    else:
        d10 = d50 = d90 = mean_diameter = std_diameter = 0
        min_diameter = max_diameter = 0
        type_counts = [0] * 5
        type_percentages = [0] * 5

    # Calculate total mass and volume
    total_volume = sum([data[3] for data in clump_size_data])
    total_mass = total_volume * O.materials[shotsId].density

    # 1. Main sim to .CSV
    main_csv_filename = os.path.join(export_dir, f"main_simulation_data_{timestamp}.csv")
    try:
        with open(main_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)


            writer.writerow(['Export_Timestamp', current_time])
            writer.writerow(['Export_Iteration', O.iter])
            writer.writerow(['Export_Simulation_Time_s', f"{O.time:.6f}"])
            writer.writerow(['Timestep_s', f"{O.dt:.6e}"])
            writer.writerow(['Packing_Method', 'Cylindrical_MakeCloud'])
            writer.writerow([])

            writer.writerow(['--- Material properties ---'])
            writer.writerow(['Shots_Young_Modulus_Pa', O.materials[shotsId].young])
            writer.writerow(['Shots_Density_kg_m3', O.materials[shotsId].density])
            writer.writerow(['Shots_Poisson_Ratio', O.materials[shotsId].poisson])
            writer.writerow(['Shots_Friction_Angle_deg', math.degrees(O.materials[shotsId].frictionAngle)])
            writer.writerow([])

            writer.writerow(['--- Particle parameters ---'])
            writer.writerow(['Particle_Radius_Min_um', rMin * 1e6])
            writer.writerow(['Particle_Radius_Max_um', rMax * 1e6])
```



```
            writer.writerow(['Particle_Diameter_Min_um', rMin * 2 * 1e6])
            writer.writerow(['Particle_Diameter_Max_um', rMax * 2 * 1e6])
            writer.writerow([])

            # Geometry Parameters
            writer.writerow(['--- Geometry parameters ---'])
            writer.writerow(['Tube_Diameter_mm', tube_diameter * 1000])
            writer.writerow(['Tube_Height_mm', tube_height * 1000])
            writer.writerow(['Fill_Height_mm', fill_height * 1000])
            writer.writerow(['Ground_Size_mm', ground_size * 1000])
            writer.writerow(['Packing_Density', packing_density])
            writer.writerow([])

            # Current Status
            writer.writerow(['--- Current status ---'])
            writer.writerow(['Total_Clumps_Generated', len(clump_size_data)])
            writer.writerow(['Total_Volume_mm3', total_volume * 1e9])
            writer.writerow(['Total_Mass_g', total_mass * 1000])
            writer.writerow(['Current_Phase', state.phase if 'state' in globals() else "Unknown"])
            writer.writerow([])

            # Size Distribution Statistics
            writer.writerow(['--- Size distribution  ---'])
            writer.writerow(['Equivalent_Diameter_D10_um', d10])
            writer.writerow(['Equivalent_Diameter_D50_um', d50])
            writer.writerow(['Equivalent_Diameter_D90_um', d90])
            writer.writerow(['Mean_Diameter_um', mean_diameter])
            writer.writerow(['Std_Diameter_um', std_diameter])

        print(f"Main simulation data exported: {main_csv_filename}")
    except Exception as e:
        print(f"Main simulation export failed: {e}")

    # 2. Time series .CSV
    time_series_csv_filename = os.path.join(export_dir, f"time_series_data_{timestamp}.csv")
    try:
        with open(time_series_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow(['Iteration', 'Simulation_Time_s', 'Unbalanced_Force', 'Phase', 'Total_Clumps', 'Timestep

            for data_point in time_series_data:
                writer.writerow([
                    data_point['iteration'],
                    f"{data_point['simulation_time_s']:.6f}",
                    f"{data_point['unbalanced_force']:.6f}",
                    data_point['phase'],
                    data_point['total_clumps'],
                    f"{data_point['timestep_s']:.6e}"
                ])

        print(f"Time series data exported: {time_series_csv_filename}")
    except Exception as e:
        print(f"Time series export failed: {e}")

# 13. .VTK export
vtk_export_enabled = True
vtk_export_interval = 500
vtk_output_dir = "vtk_export_angle_repose"
vtk_file_prefix = "angle_repose_simulation"

def setupVTKExport():
    vtk_dir = vtk_output_dir
    if not os.path.exists(vtk_dir):
        os.makedirs(vtk_dir)

    def exportSeparateObjectVTKs():
        if not vtk_export_enabled or O.iter % vtk_export_interval != 0:
            return

        timestamp = f"{O.iter:06d}"

        try:
            exportGroundPlateVTK(timestamp)
            exportTubeVTK(timestamp)
            exportClumpsVTK(timestamp)
        except Exception as e:
            print(f"VTK export error: {e}")

    def exportGroundPlateVTK(timestamp):
        vtk_filename = f"{vtk_output_dir}/ground_plate_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': []}
        cell_data = {'body_id': []}
        vertex_count = 0

        for body in O.bodies:
            if body is None or not hasattr(body, 'shape'):
                continue

            try:
                if hasattr(body.shape, 'extents') and body.id == ground_id:
                    pos = body.state.pos
                    extents = body.shape.extents


                    box_vertices = [
                        [pos[0] - extents[0], pos[1] - extents[1], pos[2] - extents[2]],
                        [pos[0] + extents[0], pos[1] - extents[1], pos[2] - extents[2]],
                        [pos[0] + extents[0], pos[1] + extents[1], pos[2] - extents[2]],
                        [pos[0] - extents[0], pos[1] + extents[1], pos[2] - extents[2]],
                        [pos[0] - extents[0], pos[1] - extents[1], pos[2] + extents[2]],
                        [pos[0] + extents[0], pos[1] - extents[1], pos[2] + extents[2]],
                        [pos[0] + extents[0], pos[1] + extents[1], pos[2] + extents[2]],
                        [pos[0] - extents[0], pos[1] + extents[1], pos[2] + extents[2]]
                    ]

                    all_vertices.extend(box_vertices)

                    faces = [
                        [0, 1, 2], [0, 2, 3],  # bottom
                        [4, 7, 6], [4, 6, 5],  # top
                        [0, 4, 5], [0, 5, 1],  # front
                        [2, 6, 7], [2, 7, 3],  # back
                        [0, 3, 7], [0, 7, 4],  # left
                        [1, 5, 6], [1, 6, 2]   # right
                    ]

                    for face in faces:
                        all_cells.append([3, vertex_count + face[0], vertex_count + face[1], vertex_count + face[2]])
                        all_cell_types.append(5)
                        cell_data['body_id'].append(body.id)

                    vel_mag = body.state.vel.norm()
                    for _ in range(8):
                        point_data['velocity_magnitude'].append(vel_mag)

                    vertex_count += 8
                    break
            except:
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types, point_data, cell_data, "Ground Plate"

    def exportTubeVTK(timestamp):
        vtk_filename = f"{vtk_output_dir}/tube_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': []}
        cell_data = {'body_id': []}
        vertex_count = 0

        for body_id in tube_bodies:
            try:
                body = O.bodies[body_id]
                if body is None or not hasattr(body, 'shape'):
                    continue

                if hasattr(body.shape, 'vertices'):
                    facet_vertices = [body.state.pos + body.shape.vertices[i] for i in range(3)]

                    for v in facet_vertices:
                        all_vertices.append([v[0], v[1], v[2]])

                    all_cells.append([3, vertex_count, vertex_count + 1, vertex_count + 2])
                    all_cell_types.append(5)  # VTK_TRIANGLE
                    vertex_count += 3

                    vel_mag = body.state.vel.norm()
                    for _ in range(3):
```

```
                        point_data['velocity_magnitude'].append(vel_mag)

                    cell_data['body_id'].append(body.id)
                except:
                    continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types, point_data, cell_data, "Tube")

    def exportClumpsVTK(timestamp):

        vtk_filename = f"{vtk_output_dir}/clumps_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': [], 'radius_mm': [], 'clump_type': []}
        cell_data = {'body_id': [], 'mass_kg': []}
        vertex_count = 0

        particle_count = 0
        max_particles = 80000

        for body in O.bodies:
            if body is None or not hasattr(body, 'shape') or body.mask != 3:
                continue

            try:
                if hasattr(body.shape, 'radius'):
                    pos = body.state.pos
                    radius = body.shape.radius
                    vel_mag = body.state.vel.norm()

                    clump_type = 0
                    if hasattr(body.shape, 'color'):
                        color = body.shape.color
                        if abs(color[1] - 0.8) < 0.1:
                            clump_type = 1
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.2) < 0.1:
                            clump_type = 2
                        elif abs(color[2] - 0.8) < 0.1:
                            clump_type = 3
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.8) < 0.1:
                            clump_type = 4
                        elif abs(color[2] - 0.8) < 0.1 and abs(color[0] - 0.8) < 0.1:
                            clump_type = 5

                    all_vertices.append([pos[0], pos[1], pos[2]])
                    all_cells.append([1, vertex_count])
                    all_cell_types.append(1)
                    vertex_count += 1

                    point_data['velocity_magnitude'].append(vel_mag)
                    point_data['radius_mm'].append(radius * 1000)
                    point_data['clump_type'].append(clump_type)

                    cell_data['body_id'].append(body.id)
                    cell_data['mass_kg'].append(body.state.mass)

                    particle_count += 1

                    if particle_count >= max_particles:
                        break
            except:
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types, point_data, cell_data,
                         f"Clump Particles ({particle_count} spheres)")

    def writeVTKFile(filename, vertices, cells, cell_types, point_data, cell_data, description):
        with open(filename, 'w') as f:
            f.write("# vtk DataFile Version 3.0\n")
            f.write(f"{description}\n")
            f.write("ASCII\n")
            f.write("Dataset Unstructured_grid\n")

            f.write(f"Points {len(vertices)} float\n")
            for v in vertices:
                f.write(f"{v[0]:.6e} {v[1]:.6e} {v[2]:.6e}\n")

            total_cell_size = sum(len(cell) for cell in cells)
            f.write(f"Cells {len(cells)} {total_cell_size}\n")
            for cell in cells:
                f.write(" ".join(map(str, cell)) + "\n")

            f.write(f"Cell_Types {len(cells)}\n")
            for cell_type in cell_types:
                f.write(f"{cell_type}\n")

            if point_data:
                f.write(f"Point_data {len(vertices)}\n")
                for key, values in point_data.items():
                    if values:
                        f.write(f"Scalars {key} float 1\n")
                        f.write("Lookup_table default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n")

            if cell_data:
                f.write(f"Cell_data {len(cells)}\n")
                for key, values in cell_data.items():
                    if values:
                        f.write(f"SCALARS {key} float 1\n")
                        f.write("Lookup_table default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n")

    return exportSeparateObjectVTKs

exportSeparateObjectVTKs = setupVTKExport()

# 14. Simu control and check
def checkSystemStability():
    if state.phase != "SETTLING":
        return False

    if O.iter % state.force_check_interval != 0:
        return False

    unbalanced_ratio = calculateUnbalancedForce()
    state.unbalanced_force_history.append(unbalanced_ratio)

    if len(state.unbalanced_force_history) > 10:
        state.unbalanced_force_history.pop(0)

    if unbalanced_ratio < unbalanced_force_threshold:
        state.stable_force_count += 1
        print(f"Stable check {state.stable_force_count}/{state.required_stable_checks}: "
              f"Unbalanced force = {unbalanced_ratio:.4f}")
    else:
        state.stable_force_count = 0
        print(f"System not stable: Unbalanced force = {unbalanced_ratio:.4f}")

    if state.stable_force_count >= state.required_stable_checks:
        print("System stable Switching to lifting phase.")
        switchToLifting()
        return True

    return False

def performPackingFilling():
    if state.phase != "PACKING" or state.packing_complete:
        return

    clumps_created, total_mass, total_volume, clump_ids = createCylindricalPacking()

    state.currentMass = total_mass
    state.currentVolume = total_volume
    state.clumpsGenerated = clumps_created
    state.clump_ids = clump_ids

    particle_count = 0
    for clump_id in clump_ids:
        if O.bodies[clump_id] and hasattr(O.bodies[clump_id], 'shape') and hasattr(O.bodies[clump_id].shape, 'members
            particle_count += len(O.bodies[clump_id].shape.members)

    state.particleCount = particle_count
    state.packing_complete = True

    switchToSettling()

def switchToSettling():
    state.phase = "SETTLING"
    state.phase_start_time = O.time
    state.stable_force_count = 0
```



```
    state.unbalanced_force_history.clear()

    for engine in O.engines:
        if isinstance(engine, NewtonIntegrator):
            engine.damping = 0.8

    print(f" Settling ")
    print(f"Generated {state.clumpsGenerated} clumps ({state.particleCount} particles)")
    print(f"Monitoring unbalanced force (threshold: {unbalanced_force_threshold})")

def switchToLifting():
    state.phase = "LIFTING"
    state.phase_start_time = O.time
    state.tube_lift_start_z = tube_center[2]

    for engine in O.engines:
        if isinstance(engine, NewtonIntegrator):
            engine.damping = 0.2

    print(f" LIFTING PHASE ")

def liftTube():
    if state.phase != "LIFTING":
        return

    elapsed_time = O.time - state.phase_start_time
    lift_distance = lifting_speed * elapsed_time

    for body_id in tube_bodies:
        try:
            body = O.bodies[body_id]
            if body is None:
                continue

            current_pos = body.state.pos
            new_z = state.tube_lift_start_z + lift_distance + (current_pos[2] - state.tube_lift_start_z)
            body.state.pos = Vector3(current_pos[0], current_pos[1], new_z)
        except:
            continue

def controlClumpVelocities():
    if state.phase not in ["SETTLING", "LIFTING"]:
        return

    max_velocity = 2.0
    max_angular_velocity = 50.0

    for clump_id in state.clump_ids:
        try:
            body = O.bodies[clump_id]
            if body is None:
                continue

            vel = body.state.vel
            if vel.norm() > max_velocity:
                body.state.vel = vel.normalized() * max_velocity

            angVel = body.state.angVel
            if angVel.norm() > max_angular_velocity:
                body.state.angVel = angVel.normalized() * max_angular_velocity

            if state.phase == "SETTLING" and (O.time - state.phase_start_time) < state.initial_settling_time:
                if vel[2] > 0:
                    body.state.vel = Vector3(vel[0] * 0.5, vel[1] * 0.5, vel[2] * 0.1)

        except:
            continue

def fine_particle_monitor():

    if O.iter % 100 == 0:
        recordTimeSeriesData()

    if O.iter % 2000 == 0:
        unbalanced = calculateUnbalancedForce()

        print(f"Phase: {state.phase} | Time: {O.time:.1f}s | "
              f"Clumps: {state.clumpsGenerated} | "
              f"Unbalanced: {unbalanced:.4f}")

        if state.phase == "SETTLING":
            if (O.time - state.phase_start_time) > state.initial_settling_time:
                for engine in O.engines:
                    if isinstance(engine, NewtonIntegrator) and engine.damping > 0.3:
                        engine.damping = 0.3
                        print("Reduced damping to 0.3")

        checkSystemStability()

# 15. Simulation engines
O.engines = [
    ForceResetter(),
    InsertionSortCollider([Bo1_Sphere_Aabb(), Bo1_Facet_Aabb(), Bo1_Box_Aabb()],
                          verletDist=2*rMax),
    InteractionLoop(
        [Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom(), Ig2_Box_Sphere_ScGeom()],
        [Ip2_FrictMat_FrictMat_MindlinPhys(
            en=MatchMaker(matches=((steelId, shotsId, 0.3), (shotsId, shotsId, 0.4),
                                   (tubeId, shotsId, 0.3), (steelId, steelId, 0.3))),
            es=MatchMaker(matches=((steelId, shotsId, 0.3), (shotsId, shotsId, 0.4),
                                   (tubeId, shotsId, 0.3), (steelId, steelId, 0.3)))
        )],
        [Law2_ScGeom_MindlinPhys_Mindlin(label='contactLaw', neverErase=False)]
    ),
    NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.8, label='newton'),
    PyRunner(command='performPackingFilling()', iterPeriod=100, label='packingFiller'),
    PyRunner(command='liftTube()', iterPeriod=50, label='tubeLifter'),
    PyRunner(command='controlClumpVelocities()', iterPeriod=100, label='velocityController'),
    PyRunner(command='fine_particle_monitor()', iterPeriod=2000, label='fineMonitor'),
    PyRunner(command='exportSeparateObjectVTKs()', iterPeriod=vtk_export_interval, label='vtkExporter'),
    DomainLimiter(lo=(-ground_size/2, -ground_size/2, -15e-3),
                  hi=(ground_size/2, ground_size/2, 150e-3), iterPeriod=1000, label='domLim'),
]

# 16. Timestep
critical_dt = min(
    utils.PWaveTimeStep(),
    utils.SpherePWaveTimeStep(rMin, O.materials[shotsId].density, O.materials[shotsId].young)
)
O.dt = 1.2 * critical_dt

# Visualization setup
try:
    from yade import qt
    v = qt.View()
    v.upVector = (0, 0, 1)
    v.viewDir = (-1, -1, -0.5)
    v.center(median=False)

except ImportError:
    print("Visualization not available")

O.saveTmp()

# 17. Startup info
print(f"Material properties:")
print(f"  Shots: density={O.materials[shotsId].density} kg/m³, friction={math.degrees(O.materials[shotsId].frictionAn
print(f"  Steel: density={O.materials[steelId].density} kg/m³, friction={math.degrees(O.materials[steelId].frictionAn
print(f"  Particle size: {rMin*2*1000:.3f} - {rMax*2*1000:.3f} mm (DIAMETER)")
print(f"Simulation base data:")
print(f"  Tube diameter: {tube_diameter*1000:.1f} mm")
print(f"  Fill height: {fill_height*1000:.1f} mm")
print(f"  Ground size: {ground_size*1000:.1f} mm")
print("Available commands::")
print("  completeSimulationExport()         - Export all data to CSV files")
print("  update_plot()                      - Update size distribution plots")
```

II. Melléklet: Kopásvizsgálati szimuláció script

```python
# Wear simulation

# 1. Imports
from yade import utils, geom, Vector3, Matrix3, pack, ymport
import numpy as np
import math
import random
import matplotlib.pyplot as plt
from matplotlib import cm
import json
import os
import pickle
import time
import csv
from datetime import datetime

# 2. Main parameters
rMin = 0.05e-3
rMax = 0.055e-3
massFlowRate = 1.0/1000
MAX_ACTIVE_CLUMPS = 1000
maxClumpsPerCycle = 3
clumpTypePercentages = [20, 20, 20, 20, 20]

conveyor_speed = 0.6
tilt_angle_deg = 0

wearbody_force_magnitude = 10
wearbody_force_contact_threshold = 10
wearbody_force_enabled = True

k_archard = 10e-2 # [-]
wearbody_hardness = 5e6 # [MPa]
TARGET_VOLUME_LOSS_MM3 = 0.2
TARGET_VOLUME_LOSS_M3 = TARGET_VOLUME_LOSS_MM3 * 1e-9

# 3. Material definition
regolithId, steelId = O.materials.append([
    FrictMat(young=40e6, density=1300, poisson=.25, frictionAngle=math.radians(38), label='regolith'),
    FrictMat(young=200e6, density=7800, poisson=.3, frictionAngle=math.radians(30), label='steel'),
])

# 4. Conveyor and Wearbody creation
simulation_start_time = time.time()
wearbodyReleased = False
wearbody_force_active = False

def createConveyor():
    global conveyor_facets, fctIds

    rotation_angle_rad = math.radians(-tilt_angle_deg)
    rotation_center = Vector3(12e-3, 0, 0)
    R = Matrix3(
        math.cos(rotation_angle_rad), 0, math.sin(rotation_angle_rad),
        0, 1, 0,
        -math.sin(rotation_angle_rad), 0, math.cos(rotation_angle_rad)
    )

    fctIds = O.bodies.append(geom.facetBox(
        center=(12e-3, 0, 5e-3),
        extents=(16e-3, 10e-3, 5e-3),
        wallMask=29,
        wire=True,
        material=steelId,
        color=(0, 1, .3),
        mask=1,
        fixed=True
    ))

    for facet_id in fctIds:
        facet = O.bodies[facet_id]
        p0 = facet.state.pos + facet.shape.vertices[0]
        p1 = facet.state.pos + facet.shape.vertices[1]
        p2 = facet.state.pos + facet.shape.vertices[2]
        p0_new = rotation_center + R * (p0 - rotation_center)
        p1_new = rotation_center + R * (p1 - rotation_center)
        p2_new = rotation_center + R * (p2 - rotation_center)
        new_pos = (p0_new + p1_new + p2_new) / 3.0
        facet.state.pos = new_pos
        facet.shape.vertices = [p0_new - new_pos, p1_new - new_pos, p2_new - new_pos]

    conveyor_facets = []
    for b_id in fctIds:
        b = O.bodies[b_id]
        if b.shape.normal[2] > 0.9:
            b.mask = 5
            conveyor_facets.append(b_id)

def applyConveyorForces():
    rotation_angle_rad = math.radians(-tilt_angle_deg)
    conveyor_bodies = [O.bodies[id] for id in conveyor_facets]
    conveyor_direction = Vector3(math.cos(rotation_angle_rad), 0, math.sin(rotation_angle_rad))

    for conveyor_body in conveyor_bodies:
        for i in O.interactions.withBody(conveyor_body.id):
            if not i.isReal:
                continue
            particle_id = i.id1 if i.id2 == conveyor_body.id else i.id2
            particle_body = O.bodies[particle_id]
            if particle_body is None or particle_body.mask != 3:
                continue
            if i.geom.normal[2] > 0.7:
                current_vel = particle_body.state.vel
                vel_along_conveyor = current_vel.dot(conveyor_direction)
                vel_diff = conveyor_speed - vel_along_conveyor
                force_magnitude = 3.0 * particle_body.state.mass * vel_diff / O.dt
                force = force_magnitude * conveyor_direction
                O.forces.addF(particle_id, force)

def createWearbody():
    global wearbodyIds, wearbodyClump, facet_center_positions, facet_areas
    global initialWearbodyVolume, initialWearbodyMass, numFacets
    global wearbodyInitialPos, release_x_threshold, min_z_pos, max_z_pos

    wearbodyPos = (1.6e-2, 0, 7e-3)
    wearbodyExt = (10e-3, 9e-3, 4e-3)
    initialHeight = 7e-3
    wearbodyInitialPos = Vector3(wearbodyPos[0], wearbodyPos[1], initialHeight)

    release_x_threshold = wearbodyPos[0] - wearbodyExt[0]/4
    min_z_pos = 1e-3
    max_z_pos = 10e-3

    stl_facets = ymport.stl('wearbody.STL')

    min_x = min_y = min_z = float('inf')
    max_x = max_y = max_z = float('-inf')

    for facet in stl_facets:
        for v in facet.shape.vertices:
            min_x = min(min_x, facet.state.pos[0] + v[0])
            max_x = max(max_x, facet.state.pos[0] + v[0])
            min_y = min(min_y, facet.state.pos[1] + v[1])
            max_y = max(max_y, facet.state.pos[1] + v[1])
            min_z = min(min_z, facet.state.pos[2] + v[2])
            max_z = max(max_z, facet.state.pos[2] + v[2])

    stl_size_x = max_x - min_x
    stl_size_y = max_y - min_y
    stl_size_z = max_z - min_z
    stl_center = Vector3((min_x + max_x) / 2, (min_y + max_y) / 2, (min_z + max_z) / 2)

    scale_x = 2 * wearbodyExt[0] / stl_size_x if stl_size_x > 0 else 1
    scale_y = 2 * wearbodyExt[1] / stl_size_y if stl_size_y > 0 else 1
    scale_z = 2 * wearbodyExt[2] / stl_size_z if stl_size_z > 0 else 1
    scale = min(scale_x, scale_y, scale_z)

    wearbodyIds = []
    scaled_facets = []

    for facet in stl_facets:
        scaled_vertices = []
        for v in facet.shape.vertices:
            scaled_v = Vector3(
                (facet.state.pos[0] + v[0] - stl_center[0]) * scale,
                (facet.state.pos[1] + v[1] - stl_center[1]) * scale,
                (facet.state.pos[2] + v[2] - stl_center[2]) * scale
            )
            scaled_vertices.append(scaled_v)

        new_facet = utils.facet(
            [wearbodyInitialPos + v for v in scaled_vertices],
            material=steelId,
            color=(0.6, 0.6, 0.6),
            wire=False
        )
```



```python
        new_facet.mask = 3
        new_facet.state.blockedDOFs = ''
        new_facet.dynamic = True
        facet_id = O.bodies.append(new_facet)
        wearbodyIds.append(facet_id)
        scaled_facets.append(O.bodies[facet_id])

    initialWearbodyVolume = calculateSTLVolume(scaled_facets)
    density = O.materials[steelId].density
    initialWearbodyMass = density * initialWearbodyVolume
    numFacets = len(wearbodyIds)
    facetMass = initialWearbodyMass / numFacets

    facet_center_positions = {}
    facet_areas = {}
    for facet_id in wearbodyIds:
        try:
            facet = O.bodies[facet_id]
            if facet is not None:
                vertices = [facet.state.pos + facet.shape.vertices[i] for i in range(3)]
                center = (vertices[0] + vertices[1] + vertices[2]) / 3
                facet_center_positions[facet_id] = center

                v1 = vertices[1] - vertices[0]
                v2 = vertices[2] - vertices[0]
                area = 0.5 * v1.cross(v2).norm()
                facet_areas[facet_id] = area
        except IndexError:
            continue

    for id in wearbodyIds:
        b = O.bodies[id]
        vertices = [b.state.pos + b.shape.vertices[i] for i in range(3)]
        center = (vertices[0] + vertices[1] + vertices[2]) / 3
        v1 = vertices[1] - vertices[0]
        v2 = vertices[2] - vertices[0]
        crossProduct = v1.cross(v2)
        area = 0.5 * math.sqrt(crossProduct.squaredNorm())
        b.state.mass = facetMass
        b.state.inertia = Vector3(facetMass, facetMass, facetMass) / 12.0
        b.state.blockedDOFs = ''
        b.dynamic = True

    wearbodyClump = O.bodies.clump(wearbodyIds)
    clumpBody = O.bodies[wearbodyClump]
    clumpBody.state.vel = Vector3(0, 0, 0)
    clumpBody.state.angVel = Vector3(0, 0, 0)
    clumpBody.state.blockedDOFs = 'xyzXYZ'
    clumpBody.dynamic = False

def calculateSTLVolume(facets):
    total_volume = 0.0
    for facet in facets:
        vertices = [facet.state.pos + v for v in facet.shape.vertices]
        v0, v1, v2 = vertices[0], vertices[1], vertices[2]
        cross_product = v1.cross(v2)
        signed_volume = (1.0/6.0) * v0.dot(cross_product)
        total_volume += signed_volume
    return abs(total_volume)

def countWearbodyContacts():
    total_contacts = 0
    for facet_id in wearbodyIds:
        try:
            facet = O.bodies[facet_id]
            if facet is None:
                continue
            interactions = facet.intrs()
            real_interactions = sum(1 for i in interactions if i.isReal)
            total_contacts += real_interactions
        except IndexError:
            continue
    return total_contacts

def checkWearbodyTrigger():
    global wearbodyReleased
    if wearbodyReleased:
        return

    sample_size = min(20, len(O.bodies))
    for _ in range(sample_size):
        b_id = random.randint(0, len(O.bodies)-1)
        try:
            b = O.bodies[b_id]
            if b is None:
                continue
            if b.isClump and b.id != wearbodyClump and b.mask == 3:
                if b.state.pos[0] >= release_x_threshold:
                    releaseWearbody()
                    return
        except IndexError:
            continue

def releaseWearbody():
    global wearbodyReleased
    if not wearbodyReleased:
        try:
            clumpBody = O.bodies[wearbodyClump]
            if clumpBody is None:
                print("Error: Cannot release wearbody - wearbody no longer exists")
                return

            clumpBody.state.pos = Vector3(wearbodyInitialPos[0], wearbodyInitialPos[1], clumpBody.state.pos[2])
            clumpBody.state.vel = Vector3(0, 0, 0)
            clumpBody.state.angVel = Vector3(0, 0, 0)
            clumpBody.state.blockedDOFs = 'xyXYZ'
            O.engines = O.engines + [PyRunner(command='constrainWearbodyPosition()', iterPeriod=1, label='positionCon
            clumpBody.dynamic = True
            print(f"Wearbody released at iteration {O.iter}, constrained to vertical movement only")
            wearbodyReleased = True
        except IndexError:
            print("Error: Cannot release wearbody - wearbody no longer exists")
            return

def constrainWearbodyPosition():
    global wearbodyReleased
    if not wearbodyReleased:
        return

    try:
        wb = O.bodies[wearbodyClump]
        if wb is None:
            raise IndexError
    except IndexError:
        print(f"Warning: Cannot constrain wearbody position - wearbody (ID: {wearbodyClump}) no longer exists")
        for i, eng in enumerate(O.engines):
            if hasattr(eng, 'label') and eng.label == 'positionConstrainer':
                O.engines.pop(i)
                print("Disabled position constrainer due to missing wearbody")
                break
        return

    initial_x = wearbodyInitialPos[0]
    initial_y = wearbodyInitialPos[1]
    current_pos = wb.state.pos

    if current_pos[0] != initial_x or current_pos[1] != initial_y:
        wb.state.pos = Vector3(initial_x, initial_y, current_pos[2])

    wb.state.vel = Vector3(0, 0, wb.state.vel[2])

    drift_x = current_pos[0] - initial_x
    drift_y = current_pos[1] - initial_y
    if abs(drift_x) > 1e-10 or abs(drift_y) > 1e-10:
        correction_force = Vector3(-drift_x * 1000, -drift_y * 1000, 0)
        O.forces.addF(wearbodyClump, correction_force)

    if current_pos[2] < min_z_pos:
        wb.state.pos = Vector3(initial_x, initial_y, min_z_pos)
        wb.state.vel = Vector3(0, 0, max(0.0, wb.state.vel[2]))
    elif current_pos[2] > max_z_pos:
        wb.state.pos = Vector3(initial_x, initial_y, max_z_pos)
        wb.state.vel = Vector3(0, 0, min(0.0, wb.state.vel[2]))

    if abs(wb.state.vel[2]) > 0.1:
        wb.state.vel = Vector3(0, 0, wb.state.vel[2] * 0.9)

    wb.state.angVel = Vector3(0, 0, 0)

def applyWearbodyForce():
    global wearbody_force_active
    if not wearbody_force_enabled:
        return

    try:
        wb = O.bodies[wearbodyClump]
```

```
        if wb is None:
            return
    except IndexError:
        return

    if wearbodyReleased:
        total_contacts = countWearbodyContacts()
        if total_contacts >= wearbody_force_contact_threshold:
            wearbody_force_active = True

        if wearbody_force_active:
            force_vector = Vector3(0, 0, -wearbody_force_magnitude)
            O.forces.addF(wearbodyClump, force_vector)

def monitorWearbody():
    if O.iter % 200 == 0:
        try:
            clumpBody = O.bodies[wearbodyClump]
            if clumpBody is None:
                return
        except IndexError:
            return

# 5. Clump related functions
targetMass = 0.0
currentMass = 0.0
lastMassUpdateIter = 0
numSphereGen = 0
clumpsGenerated = 0
clumpsRemoved = 0
activeClumpsCount = 0
lastClumpCheckIter = 0
clumpCheckInterval = 1000
fast_count_interval = 100
new_clumps_since_check = 0
removed_clumps_since_check = 0
clump_size_data = []

def createClumpTemplates():
    c1 = [((0, 0, 0), 0.75), ((1.2, 0, 0), 1.0)]
    c2 = [((0, 0, 0), 0.75), ((1.1, 0, 0), 0.75), ((0.55, 1.2, 0), 1.0)]
    c3 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 1.0)]
    c4 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0.4, 0.9, 0), 0.75), ((0.4, 0.3, 1.0), 0.9), ((0.4, 0.3, -1.0), 0.9
    c5 = [((0, 0, 0), 0.6), ((0.8, 0, 0), 0.6), ((0, 0.8, 0), 0.6), ((0.8, 0.8, 0), 0.6),
          ((0, 0, 0.8), 0.6), ((0.8, 0, 0.8), 0.6), ((0, 0.8, 0.8), 0.6), ((0.8, 0.8, 0.8), 0.6)]
    return [c1, c2, c3, c4, c5]

def getRandomRadius():
    return random.uniform(rMin, rMax)

def createClump(clump_type):
    template = clumpTemplates[clump_type]
    base_radius = getRandomRadius()

    clumpOutletCenter = (0, 0, 1e-3)
    clumpOutletExtents = (6e-3, 18e-3, 1e-3)

    x_offset = random.uniform(-1e-3, 1e-3)
    y_offset = random.uniform(-1e-3, 1e-3)
    center_x = clumpOutletCenter[0] + random.uniform(-clumpOutletExtents[0]/2, clumpOutletExtents[0]/2) + x_offset
    center_y = clumpOutletCenter[1] + random.uniform(-clumpOutletExtents[1]/2, clumpOutletExtents[1]/2) + y_offset
    center_z = clumpOutletCenter[2] + random.uniform(0, 5e-3)

    colors = [(0.2, 0.8, 0.2), (0.8, 0.2, 0.2), (0.2, 0.2, 0.8), (0.8, 0.8, 0.2), (0.8, 0.2, 0.8)]
    color = colors[clump_type]

    spheres = []
    total_volume = 0
    max_radius = 0

    for rel_pos, radius_factor in template:
        sphere_radius = base_radius * radius_factor
        max_radius = max(max_radius, sphere_radius)
        abs_x = center_x + rel_pos[0] * base_radius
        abs_y = center_y + rel_pos[1] * base_radius
        abs_z = center_z + rel_pos[2] * base_radius

        s = utils.sphere((abs_x, abs_y, abs_z), sphere_radius, material=regolithId)
        s.color = color
        s.mask = 3
        spheres.append(s)
        total_volume += (4/3) * math.pi * (sphere_radius**3)

    equivalent_radius = ((3 * total_volume) / (4 * math.pi)) ** (1/3)
    clump_size_data.append((base_radius, clump_type, equivalent_radius, total_volume, max_radius))

    total_mass = total_volume * O.materials[regolithId].density
    return spheres, total_mass

def onBodyRemoved(id):
    global removed_clumps_since_check
    try:
        body = O.bodies[id]
        if body is not None and body.isClump and body.mask == 3:
            removed_clumps_since_check += 1
    except:
        pass

def countActiveClumps():
    global activeClumpsCount, lastClumpCheckIter, new_clumps_since_check, removed_clumps_since_check

    if O.iter - lastClumpCheckIter >= clumpCheckInterval:
        active_count = 0
        for body in O.bodies:
            if body is not None and body.isClump and body.mask == 3:
                active_count += 1

        removed_in_interval = (activeClumpsCount + new_clumps_since_check) - active_count - removed_clumps_since_chec
        if removed_in_interval > 0:
            global clumpsRemoved
            clumpsRemoved += removed_in_interval

        activeClumpsCount = active_count
        lastClumpCheckIter = O.iter
        new_clumps_since_check = 0
        removed_clumps_since_check = 0
        return active_count

    return activeClumpsCount + new_clumps_since_check - removed_clumps_since_check

def generateClumps():
    global numSphereGen, currentMass, targetMass, lastMassUpdateIter, clumpsGenerated, new_clumps_since_check

    current_active = countActiveClumps()
    if current_active >= MAX_ACTIVE_CLUMPS:
        return

    iterPassed = O.iter - lastMassUpdateIter
    timeElapsed = iterPassed * O.dt
    targetMass += massFlowRate * timeElapsed
    lastMassUpdateIter = O.iter

    clumps_added = 0
    while currentMass < targetMass and clumps_added < maxClumpsPerCycle and current_active + clumps_added < MAX_ACTIV
        clump_type = random.choices([0, 1, 2, 3, 4], weights=clumpTypePercentages, k=1)[0]
        spheres, clumpMass = createClump(clump_type)

        clpId, sphIds = O.bodies.appendClumped(spheres)
        O.bodies[clpId].state.vel = Vector3(0, 0, -0.2)
        O.bodies[clpId].mask = 3

        currentMass += clumpMass
        numSphereGen += len(sphIds)
        clumpsGenerated += 1
        new_clumps_since_check += 1
        clumps_added += 1

def stabilizeParticles():
    max_safe_velocity = 1.0
    max_safe_angular_velocity = 2000.0

    sample_size = min(100, len(O.bodies))
    bodies_to_check = random.sample(range(len(O.bodies)), sample_size)

    for b_id in bodies_to_check:
        try:
            b = O.bodies[b_id]
            if b is None or not b.isClump:
                continue

            vel = b.state.vel
            vel_magnitude = vel.norm()
            angVel = b.state.angVel
            angVel_magnitude = angVel.norm()

            if vel_magnitude > max_safe_velocity:
                b.state.vel = vel * 0.7
            if angVel_magnitude > max_safe_angular_velocity:
                b.state.angVel = angVel * 0.1
        except:
            continue

def clumptemplate():
    print("Creating clump template 3d view")

    for body in O.bodies:
        if hasattr(body, 'template_clump') and body.template_clump:
            O.bodies.erase(body.id)

    template_radius = 2e-3
    spacing = 8e-3
    base_y = -15e-3
    base_z = 15e-3

    template_clump_ids = []
    colors = [(0.2, 0.8, 0.2), (0.8, 0.2, 0.2), (0.2, 0.2, 0.8), (0.8, 0.8, 0.2), (0.8, 0.2, 0.8)]

    for clump_type in range(5):
        template = clumpTemplates[clump_type]
        center_x = clump_type * spacing
        center_y = base_y
        center_z = base_z
        color = colors[clump_type]

        spheres = []
        for rel_pos, radius_factor in template:
            sphere_radius = template_radius * radius_factor
            abs_x = center_x + rel_pos[0] * template_radius
            abs_y = center_y + rel_pos[1] * template_radius
            abs_z = center_z + rel_pos[2] * template_radius

            s = utils.sphere((abs_x, abs_y, abs_z), sphere_radius, material=regolithId)
            s.color = color
            s.mask = 7
            s.template_clump = True
            spheres.append(s)

        if len(spheres) > 1:
            clpId, sphId = O.bodies.appendClumped(spheres)
            O.bodies[clpId].template_clump = True
            O.bodies[clpId].dynamic = False
            O.bodies[clpId].state.blockedDOFs = 'xyzXYZ'
            template_clump_ids.append(clpId)
        else:
            spheres[0].dynamic = False
            spheres[0].state.blockedDOFs = 'xyzXYZ'
            template_clump_ids.append(spheres[0].id)

    print(f"Created template clumps: c1(Green), c2(Red), c3(Blue), c4(Yellow), c5(Purple)")
    return template_clump_ids

clumpTemplates = createClumpTemplates()

# 6. Archard wear equation
totalVolumeRemoved = 0.0
totalMassLost = 0.0
contact_data = {}
wear_data = []
sim_time = []
facet_wear = {}
relative_wear = {}

STATE_ITER_PERIOD = 1000
last_state_iter = 0
state_period_normal_forces = []
state_period_sliding_distances = []
state_period_volume_removed = 0.0
state_period_start_volume = 0.0

time_series_data = []

wear_grid_x_bins = 20
wear_grid_y_bins = 20
wear_grid = np.zeros((wear_grid_x_bins, wear_grid_y_bins))

def resetStatePeriodTracking():
    global state_period_normal_forces, state_period_sliding_distances, state_period_volume_removed, state_period_star
    state_period_normal_forces = []
    state_period_sliding_distances = []
    state_period_volume_removed = 0.0
    state_period_start_volume = totalVolumeRemoved

def trackWear():
    global totalVolumeRemoved, totalMassLost, contact_data, wear_data, sim_time, facet_wear, wear_grid
    global state_period_normal_forces, state_period_sliding_distances, state_period_volume_removed
    global min_x, max_x, min_y, max_y

    try:
        wb = O.bodies[wearbodyClump]
        if wb is None:
            return
    except IndexError:
        return

    if 'min_x' not in globals():
        min_x = float('inf')
        max_x = float('-inf')
        min_y = float('inf')
        max_y = float('-inf')
        for pos in facet_center_positions.values():
            min_x = min(min_x, pos[0])
            max_x = max(max_x, pos[0])
            min_y = min(min_y, pos[1])
            max_y = max(max_y, pos[1])
        min_x -= 0.5e-3
        max_x += 0.5e-3
        min_y -= 0.5e-3
        max_y += 0.5e-3

    iter_volume_removed = 0.0
    wear_grid.fill(0.0)
    density = O.materials[steelId].density

    for facet_id in wearbodyIds:
        try:
            facet = O.bodies[facet_id]
            if facet is None:
                continue

            facet_wear_volume = 0.0
            interactions = facet.intrs()

            for i in interactions:
                if not i.isReal:
                    continue

                normal_force_vector = i.phys.normalForce
                normal_force = normal_force_vector.norm()
                normal_unit = i.geom.normal
                other_id = i.id1 if i.id2 == facet_id else i.id2

                try:
                    other_body = O.bodies[other_id]
                    if other_body is None or other_body.mask != 3:
                        continue
                except IndexError:
                    continue

                contact_point = i.geom.contactPoint
                facet_vel = facet.state.vel
                other_vel = other_body.state.vel
                rel_vel = other_vel - facet_vel
                normal_component = rel_vel.dot(normal_unit) * normal_unit
                tangential_vel = rel_vel - normal_component
                sliding_speed = tangential_vel.norm()
                sliding_distance = sliding_speed * O.dt

                state_period_normal_forces.append(normal_force)
                state_period_sliding_distances.append(sliding_distance)

                wear_volume = k_archard * normal_force * sliding_distance / wearbody_hardness
                wear_volume = max(0.0, min(wear_volume, 1e-8))
                facet_wear_volume += wear_volume

                if (contact_point is not None and
                    all(not math.isnan(coord) for coord in contact_point)):
                    try:
                        contact_key = f"{int(contact_point[0]*1e6)}_{int(contact_point[1]*1e6)}_{int(contact_point[2]
                        if contact_key not in contact_data:
                            contact_data[contact_key] = {"total_wear": 0, "position": contact_point}
                        contact_data[contact_key]["total_wear"] += wear_volume
                    except (ValueError, TypeError):
```

```
                        pass

                if facet_id in facet_center_positions:
                    pos = facet_center_positions[facet_id]
                    if min_x < pos[0] < max_x and min_y < pos[1] < max_y:
                        x_idx = int((pos[0] - min_x) / (max_x - min_x) * (wear_grid_x_bins-1))
                        y_idx = int((pos[1] - min_y) / (max_y - min_y) * (wear_grid_y_bins-1))
                        if 0 <= x_idx < wear_grid_x_bins and 0 <= y_idx < wear_grid_y_bins:
                            wear_grid[x_idx, y_idx] += facet_wear_volume

                facet_wear[facet_id] += facet_wear_volume
                iter_volume_removed += facet_wear_volume

            except IndexError:
                continue

        totalVolumeRemoved += iter_volume_removed
        totalMassLost = totalVolumeRemoved * density
        state_period_volume_removed += iter_volume_removed

        if wearbodyReleased and O.iter % 50 == 0:
            updateWearbodyColors()

        if O.iter % 500 == 0:
            checkWearTarget()

        if O.iter % 100 == 0:
            calculateRelativeWear()
            updateWearbodyMass()

            if O.iter % 100 == 0:
                current_mass = initialWearbodyMass - totalMassLost
                volume_loss_percent = (totalVolumeRemoved / initialWearbodyVolume) * 100

                max_wear_depth_um, avg_wear_depth_um = calculateWearDepth()

                wear_data.append((O.time, totalVolumeRemoved, current_mass, volume_loss_percent, max_wear_depth_um, avg_w
                sim_time.append(O.time)
                generateWearProfiles()
                update_plot()

def calculateWearDepth():
    if not facet_wear or not facet_areas:
        return 0.0, 0.0

    wear_depths = []
    for facet_id in wearbodyIds:
        if facet_id in facet_wear and facet_id in facet_areas:
            area = facet_areas[facet_id]
            if area > 0:
                depth = facet_wear[facet_id] / area
                wear_depths.append(depth)

    if not wear_depths:
        return 0.0, 0.0

    max_depth_m = max(wear_depths)
    avg_depth_m = sum(wear_depths) / len(wear_depths)

    max_depth_um = max_depth_m * 1e6
    avg_depth_um = avg_depth_m * 1e6

    return max_depth_um, avg_depth_um

def wearToColor(wear_value):
    MAX_WEAR_MM2 = 0.035e-6
    ORIGINAL_WEARBODY_COLOR = (0.6, 0.6, 0.6)

    if wear_value <= 0:
        return ORIGINAL_WEARBODY_COLOR

    normalized_wear = min(wear_value / MAX_WEAR_MM2, 1.0)

    if normalized_wear < 0.1:
        factor = normalized_wear / 0.1
        r = 0.6 * (1 - factor) + 0.4 * factor
        g = 0.6 * (1 - factor) + 0.6 * factor
        b = 0.6 * (1 - factor) + 1.0 * factor
    elif normalized_wear < 0.3:
        factor = (normalized_wear - 0.1) / 0.2
        r = 0.4 * (1 - factor) + 0.0 * factor
        g = 0.6 * (1 - factor) + 1.0 * factor
        b = 1.0 * (1 - factor) + 0.0 * factor
    elif normalized_wear < 0.5:
        factor = (normalized_wear - 0.3) / 0.2
        r = 0.0 * (1 - factor) + 1.0 * factor
        g = 1.0
        b = 0.0
    elif normalized_wear < 0.7:
        factor = (normalized_wear - 0.5) / 0.2
        r = 1.0
        g = 1.0 * (1 - factor) + 0.5 * factor
        b = 0.0
    elif normalized_wear < 0.9:
        factor = (normalized_wear - 0.7) / 0.2
        r = 1.0
        g = 0.5 * (1 - factor) + 0.0 * factor
        b = 0.0
    else:
        factor = (normalized_wear - 0.9) / 0.1
        r = 1.0
        g = 0.0 * (1 - factor) + 1.0 * factor
        b = 0.0 * (1 - factor) + 1.0 * factor

    return (r, g, b)

def updateWearbodyColors():
    if not facet_wear:
        return

    for facet_id in wearbodyIds:
        try:
            facet = O.bodies[facet_id]
            if facet is None:
                continue

            wear_value = facet_wear.get(facet_id, 0.0)
            new_color = wearToColor(wear_value)
            facet.color = new_color

        except IndexError:
            continue

def updateWearbodyMass():
    global totalMassLost
    try:
        wb = O.bodies[wearbodyClump]
        if wb is None:
            return

        new_mass = initialWearbodyMass - totalMassLost
        wb.state.mass = new_mass

        if numFacets > 0:
            new_facet_mass = new_mass / numFacets
            for facet_id in wearbodyIds:
                try:
                    facet = O.bodies[facet_id]
                    if facet is not None:
                        facet.state.mass = new_facet_mass
                        facet.state.inertia = Vector3(new_facet_mass, new_facet_mass, new_facet_mass) / 12.0
                except IndexError:
                    continue
    except IndexError:
        return

def calculateRelativeWear():
    total_wear = sum(facet_wear.values())
    avg_wear = total_wear / len(facet_wear) if len(facet_wear) > 0 else 1.0
    for facet_id in facet_wear:
        relative_wear[facet_id] = facet_wear[facet_id] / avg_wear if avg_wear > 0 else 0.0
    return relative_wear

def generateWearProfiles():
    longitudinal_profile = np.sum(wear_grid, axis=1)
    transverse_profile = np.sum(wear_grid, axis=0)
    return longitudinal_profile, transverse_profile

def checkWearTarget():
    MAX_WEAR_MM2 = 0.035e-6
    if not facet_wear:
        return False

    max_wear = max(facet_wear.values())
    wear_percentage = (max_wear / MAX_WEAR_MM2) * 100
```



```
    if max_wear >= MAX_WEAR_MM2:
        print(" Target wear reached! ")
        print(f"Maximum wear: {max_wear*1e6:.6f} mm² ({wear_percentage:.1f}% of target)")
        return True

    return False

def estimateTimeToTarget():
    if state_period_volume_removed <= 0:
        return "N/A - no wear in current period"

    remaining_volume_m3 = TARGET_VOLUME_LOSS_M3 - totalVolumeRemoved

    if remaining_volume_m3 <= 0:
        return "Target reached!"

    state_period_time_s = STATE_ITER_PERIOD * O.dt
    current_wear_rate_m3_per_s = state_period_volume_removed / state_period_time_s

    if current_wear_rate_m3_per_s <= 0:
        return "N/A - no wear progress"

    estimated_time_s = remaining_volume_m3 / current_wear_rate_m3_per_s

    if estimated_time_s < 60:
        return f"{estimated_time_s:.1f} s"
    elif estimated_time_s < 3600:
        minutes = int(estimated_time_s / 60)
        seconds = int(estimated_time_s % 60)
        return f"{minutes}:{seconds:02d} min"
    else:
        hours = int(estimated_time_s / 3600)
        minutes = int((estimated_time_s % 3600) / 60)
        return f"{hours}:{minutes:02d} h"

def state():
    global last_state_iter, time_series_data

    if O.iter - last_state_iter >= STATE_ITER_PERIOD:
        avg_normal_force = sum(state_period_normal_forces) / len(state_period_normal_forces) if state_period_normal_f
        avg_sliding_distance = sum(state_period_sliding_distances) / len(state_period_sliding_distances) if state_per

        state_period_time_s = STATE_ITER_PERIOD * O.dt
        current_wear_rate_mm3_per_s = (state_period_volume_removed * 1e9) / state_period_time_s if state_period_time_

        real_sim_time = time.time() - simulation_start_time
        hours = int(real_sim_time // 3600)
        minutes = int((real_sim_time % 3600) // 60)
        seconds = int(real_sim_time % 60)
        sim_time_formatted = f"{hours:02d}:{minutes:02d}:{seconds:02d}"

        progress_to_target_percent = (totalVolumeRemoved * 1e9 / TARGET_VOLUME_LOSS_MM3) * 100

        max_wear_depth_um, avg_wear_depth_um = calculateWearDepth()

        time_series_data_point = {
            'sim_time_formatted': sim_time_formatted,
            'virt_time': O.time,
            'iterations': O.iter,
            'total_volume_removed_mm3': totalVolumeRemoved * 1e9,
            'total_weight_removed_g': totalMassLost * 1000,
            'progress_to_target_percent': progress_to_target_percent,
            'avg_normal_force': avg_normal_force,
            'avg_sliding_distance_mm': avg_sliding_distance * 1000,
            'volume_removed_this_period_mm3': state_period_volume_removed * 1e9,
            'current_wear_rate_mm3_per_s': current_wear_rate_mm3_per_s,
            'max_wear_depth_um': max_wear_depth_um,
            'avg_wear_depth_um': avg_wear_depth_um
        }

        time_series_data.append(time_series_data_point)

        last_state_iter = O.iter

    active_clumps = countActiveClumps()

    try:
        wb_pos = O.bodies[wearbodyClump].state.pos
        pos_str = f"({wb_pos[0]:.6f}, {wb_pos[1]:.6f}, {wb_pos[2]:.6f})"
    except:
        pos_str = "(not found)"

    total_contacts = countWearbodyContacts()
    volume_loss_percent = (totalVolumeRemoved / initialWearbodyVolume) * 100
    force_status = "ACTIVE" if wearbody_force_active and wearbody_force_enabled else "INACTIVE"
    force_value = wearbody_force_magnitude if wearbody_force_active else 0.0

    clump_type_str = f"c1: {clumpTypePercentages[0]:.1f}%, c2: {clumpTypePercentages[1]:.1f}%, c3: {clumpTypePercenta

    time_estimate = estimateTimeToTarget()
    progress_percent = (totalVolumeRemoved * 1e9 / TARGET_VOLUME_LOSS_MM3) * 100
    max_facet_wear = max(facet_wear.values()) * 1e6 if facet_wear else 0

    current_avg_normal_force = sum(state_period_normal_forces) / len(state_period_normal_forces) if state_period_norm
    current_avg_sliding_distance = sum(state_period_sliding_distances) / len(state_period_sliding_distances) if state

    state_period_time_s = STATE_ITER_PERIOD * O.dt
    current_wear_rate_mm3_per_s = (state_period_volume_removed * 1e9) / state_period_time_s if state_period_time_s >

    max_wear_depth_um, avg_wear_depth_um = calculateWearDepth()

    TARGET_MASS_LOSS_G = TARGET_VOLUME_LOSS_M3 * O.materials[steelId].density * 1000

    print("=" * 80)
    print("SIMULATION STATUS")
    print(f"  Iteration: {O.iter}, Time: {O.time:.3f} s, Timestep: {O.dt:.2e} s")
    print("-" * 80)
    print()
    print("WEARBODY STATUS:")
    print(f"  Position: {pos_str}")
    print(f"  Released: {'YES' if wearbodyReleased else 'NO'}")
    print(f"  Force: {force_value:.1f} N [{force_status}]")
    print(f"  Contacts: {total_contacts}")
    print()
    print("CLUMP STATISTICS:")
    print(f"  Total generated: {clumpsGenerated}")
    print(f"  Current active: {active_clumps}")
    print(f"  Removed: {clumpsRemoved}")
    print(f"  Clump percentages: {clump_type_str}")
    print(f"  Size range: {rMin*2*1e6:.1f} - {rMax*2*1e6:.1f} um (diameter)")
    print()
    print("WEARBODY PROPERTIES:")
    print(f"  Original volume: {initialWearbodyVolume*1e9:.3f} mm³")
    print(f"  Original mass: {initialWearbodyMass*1000:.3f} g")
    print(f"  Target volume loss: {TARGET_VOLUME_LOSS_MM3:.1f} mm³")
    print()
    print("ARCHARD WEAR MODEL:")
    print(f"  k_archard: {k_archard:.2e}")
    print(f"  Hardness: {wearbody_hardness:.2e} N/m²")
    print(f"  Average normal force: {current_avg_normal_force:.4f} N")
    print(f"  Average sliding distance: {current_avg_sliding_distance*1000:.6f} mm")
    print(f"  Volume removed this period: {state_period_volume_removed*1e9:.6f} mm³")
    print(f"  Current wear rate: {current_wear_rate_mm3_per_s:.6f} mm³/s")

    print(f"  State period: {STATE_ITER_PERIOD} iterations")
    print(f"  Force samples: {len(state_period_normal_forces)}")
    print(f"  Sliding distance samples: {len(state_period_sliding_distances)}")
    print()
    print("WEAR RESULTS:")
    print(f"  Total volume removed: {totalVolumeRemoved*1e9:.6f} mm³")
    print(f"  Total weight removed: {totalMassLost*1000:.6f} g")
    print(f"  Progress to target: {progress_percent:.1f}%")
    print(f"  Max facet wear: {max_facet_wear:.6f} mm²")
    print(f"  Max wear depth: {max_wear_depth_um:.6f} um")
    print(f"  Avg wear depth: {avg_wear_depth_um:.6f} um")
    print(f"  Estimated virt time to target: {time_estimate}")
    print("=" * 80)

    resetStatePeriodTracking()

# 7. Data export and visuals
try:
    plt.ion()
    plt.style.use('seaborn-v0_8-whitegrid')
except:
    try:
        plt.style.use('seaborn-whitegrid')
    except:
        pass

COLORS = {
    'primary': '#7FB3D3',
    'secondary': '#B8E6B8',
    'accent': '#FFD1DC',
    'warning': '#FFEB9C',
    'info': '#D1C4E9',
```

```
    'grid': '#2C3E50',
    'text': '#2C3E50'
}

CLUMP_COLORS = ['#A8D8EA', '#FFB3BA', '#BFFF7F', '#FFE5B4', '#D7B3FF', '#FFC3A0']

last_plot_update_iter = 0
plot_update_interval = 200

def setupVTKExport():
    vtk_dir = 'output/vtk'
    if not os.path.exists(vtk_dir):
        os.makedirs(vtk_dir)

    def exportSeparateObjectVTKs():
        if O.iter % 1000 != 0:
            return

        timestamp = f"{O.iter:06d}"

        try:
            exportWearbodyVTK(timestamp)
            exportConveyorVTK(timestamp)
            exportClumpsVTK(timestamp)
        except Exception as e:
            print(f"VTK export error: {e}")

    def exportWearbodyVTK(timestamp):
        vtk_filename = f"output/vtk/wearbody_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'wear_mm3': [], 'velocity_magnitude': []}
        cell_data = {'facet_id': [], 'contact_count': [], 'area_mm2': []}
        vertex_count = 0

        for facet_id in wearbodyIds:
            try:
                facet = O.bodies[facet_id]
                if facet is None:
                    continue

                facet_vertices = [facet.state.pos + facet.shape.vertices[i] for i in range(3)]

                for v in facet_vertices:
                    all_vertices.append([v[0], v[1], v[2]])

                all_cells.append([3, vertex_count, vertex_count + 1, vertex_count + 2])
                all_cell_types.append(5)
                vertex_count += 3

                v1 = facet_vertices[1] - facet_vertices[0]
                v2 = facet_vertices[2] - facet_vertices[0]
                area = 0.5 * v1.cross(v2).norm()

                wear_value = facet_wear.get(facet_id, 0.0) * 1e9
                vel_mag = facet.state.vel.norm()

                for _ in range(3):
                    point_data['wear_mm3'].append(wear_value)
                    point_data['velocity_magnitude'].append(vel_mag)

                contact_count = sum(1 for i in facet.intrs() if i.isReal)
                cell_data['facet_id'].append(facet_id)
                cell_data['contact_count'].append(contact_count)
                cell_data['area_mm2'].append(area * 1e6)

            except (IndexError, AttributeError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Wearbody with Wear Data")

    def exportConveyorVTK(timestamp):
        vtk_filename = f"output/vtk/conveyor_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': []}
        cell_data = {'facet_id': [], 'contact_count': [], 'is_moving_belt': []}
        vertex_count = 0

        for facet_id in fctIds:
            try:
                facet = O.bodies[facet_id]
                if facet is None:
                    continue

                facet_vertices = [facet.state.pos + facet.shape.vertices[i] for i in range(3)]

                for v in facet_vertices:
                    all_vertices.append([v[0], v[1], v[2]])

                all_cells.append([3, vertex_count, vertex_count + 1, vertex_count + 2])
                all_cell_types.append(5)
                vertex_count += 3

                for _ in range(3):
                    point_data['velocity_magnitude'].append(0.0)

                contact_count = sum(1 for i in facet.intrs() if i.isReal)
                is_moving_belt = 1 if facet_id in conveyor_facets else 0

                cell_data['facet_id'].append(facet_id)
                cell_data['contact_count'].append(contact_count)
                cell_data['is_moving_belt'].append(is_moving_belt)

            except (IndexError, AttributeError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Conveyor Box")

    def exportClumpsVTK(timestamp):
        vtk_filename = f"output/vtk/clumps_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': [], 'clump_type': [], 'radius_mm': [],
                      'angular_velocity_magnitude': []}
        cell_data = {'body_id': [], 'contact_count': [], 'mass_kg': []}
        vertex_count = 0

        clump_count = 0
        for body in O.bodies:
            if body is None or not hasattr(body, 'shape') or body.mask != 3:
                continue

            try:
                if hasattr(body.shape, 'radius'):
                    pos = body.state.pos
                    radius = body.shape.radius
                    vel_mag = body.state.vel.norm()
                    angvel_mag = body.state.angVel.norm()

                    all_vertices.append([pos[0], pos[1], pos[2]])
                    all_cells.append([1, vertex_count])
                    all_cell_types.append(1)
                    vertex_count += 1

                    clump_type = 0
                    if hasattr(body.shape, 'color'):
                        color = body.shape.color
                        if abs(color[1] - 0.8) < 0.1 and abs(color[0] - 0.2) < 0.1:
                            clump_type = 1
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.2) < 0.1:
                            clump_type = 2
                        elif abs(color[2] - 0.8) < 0.1 and abs(color[0] - 0.2) < 0.1:
                            clump_type = 3
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.8) < 0.1:
                            clump_type = 4
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[2] - 0.8) < 0.1:
                            clump_type = 5

                    point_data['velocity_magnitude'].append(vel_mag)
                    point_data['clump_type'].append(clump_type)
                    point_data['radius_mm'].append(radius * 1000)
                    point_data['angular_velocity_magnitude'].append(angvel_mag)

                    contact_count = sum(1 for i in body.intrs() if i.isReal)
                    cell_data['body_id'].append(body.id)
                    cell_data['contact_count'].append(contact_count)
                    cell_data['mass_kg'].append(body.state.mass)

                    clump_count += 1

                    if clump_count > 3000:
                        break

            except (AttributeError, IndexError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Clump Particles")

    def writeVTKFile(filename, vertices, cells, cell_types, point_data, cell_data, description):
        with open(filename, 'w') as f:
            f.write("# vtk DataFile Version 3.0\n")
            f.write(f"{description}\n")
            f.write("ASCII\n")
            f.write("DATASET UNSTRUCTURED_GRID\n")

            f.write(f"POINTS {len(vertices)} float\n")
            for v in vertices:
                f.write(f"{v[0]:.6e} {v[1]:.6e} {v[2]:.6e}\n")

            total_cell_size = sum(len(cell) for cell in cells)
            f.write(f"CELLS {len(cells)} {total_cell_size}\n")
            for cell in cells:
                f.write(" ".join(map(str, cell)) + "\n")

            f.write(f"CELL_TYPES {len(cells)}\n")
            for cell_type in cell_types:
                f.write(f"{cell_type}\n")

            if point_data:
                f.write(f"POINT_DATA {len(vertices)}\n")
                for key, values in point_data.items():
                    if values:
                        data_type = "float" if isinstance(values[0], float) else "int"
                        f.write(f"SCALARS {key} {data_type} 1\n")
                        f.write("LOOKUP_TABLE default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n" if isinstance(val, float) else f"{val}\n")

            if cell_data:
                f.write(f"CELL_DATA {len(cells)}\n")
                for key, values in cell_data.items():
                    if values:
                        data_type = "float" if isinstance(values[0], float) else "int"
                        f.write(f"SCALARS {key} {data_type} 1\n")
                        f.write("LOOKUP_TABLE default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n" if isinstance(val, float) else f"{val}\n")

    return exportSeparateObjectVTKs

def completeSimulationExport():
    export_dir = "complete_simulation_export"
    if not os.path.exists(export_dir):
        os.makedirs(export_dir)

    timestamp = f"{O.iter:06d}_{int(O.time*1000):06d}"
    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    print("Exporting Complete Simulation Data to .CSV")

    if clump_size_data:
        equiv_diameters_um = [data[2] * 2 * 1e6 for data in clump_size_data]
        d10 = np.percentile(equiv_diameters_um, 10)
        d50 = np.percentile(equiv_diameters_um, 50)
        d90 = np.percentile(equiv_diameters_um, 90)
        mean_diameter = np.mean(equiv_diameters_um)
        std_diameter = np.std(equiv_diameters_um)
        min_diameter = min(equiv_diameters_um)
        max_diameter = max(equiv_diameters_um)

        clump_types = [data[1] for data in clump_size_data]
        type_counts = [clump_types.count(i) for i in range(5)]
        type_percentages = [count/len(clump_types)*100 for count in type_counts]
    else:
        d10 = d50 = d90 = mean_diameter = std_diameter = 0
        min_diameter = max_diameter = 0
        type_counts = [0] * 5
        type_percentages = [0] * 5

    avg_normal_force = sum(state_period_normal_forces) / len(state_period_normal_forces) if state_period_normal_force
    avg_sliding_distance = sum(state_period_sliding_distances) / len(state_period_sliding_distances) if state_period_
    current_wear_rate_mm3_per_s = (state_period_volume_removed * 1e9) / (STATE_ITER_PERIOD * O.dt) if O.dt > 0 else 0

    max_wear_depth_um, avg_wear_depth_um = calculateWearDepth()

    TARGET_MASS_LOSS_G = TARGET_VOLUME_LOSS_MM3 * O.materials[steelId].density * 1000

    main_csv_filename = os.path.join(export_dir, f"main_simulation_data_{timestamp}.csv")
    try:
        with open(main_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow(['Export_Timestamp', current_time])
            writer.writerow(['Export_Iteration', O.iter])
            writer.writerow(['Export_Simulation_Time_s', f"{O.time:.6f}"])
            writer.writerow(['Real_Simulation_Duration_s', f"{time.time() - simulation_start_time:.2f}"])
            writer.writerow(['Timestep_s', f"{O.dt:.6e}"])
            writer.writerow([])

            writer.writerow(['Material properties'])
            writer.writerow(['regolith_Young_Modulus_Pa', O.materials[regolithId].young])
            writer.writerow(['regolith_Density_kg_m3', O.materials[regolithId].density])
            writer.writerow(['regolith_Poisson_Ratio', O.materials[regolithId].poisson])
            writer.writerow(['regolith_Friction_Angle_deg', math.degrees(O.materials[regolithId].frictionAngle)])
            writer.writerow(['Steel_Young_Modulus_Pa', O.materials[steelId].young])
            writer.writerow(['Steel_Density_kg_m3', O.materials[steelId].density])
            writer.writerow(['Steel_Poisson_Ratio', O.materials[steelId].poisson])
            writer.writerow(['Steel_Friction_Angle_deg', math.degrees(O.materials[steelId].frictionAngle)])
            writer.writerow([])

            writer.writerow(['Particle parameters'])
            writer.writerow(['Particle_Radius_Min_um', rMin * 1e6])
            writer.writerow(['Particle_Radius_Max_um', rMax * 1e6])
            writer.writerow(['Particle_Diameter_Min_um', rMin * 2 * 1e6])
            writer.writerow(['Particle_Diameter_Max_um', rMax * 2 * 1e6])
            writer.writerow(['Mass_Flow_Rate_kg_s', massFlowRate])
            writer.writerow(['Max_Active_Clumps', MAX_ACTIVE_CLUMPS])
            writer.writerow(['Max_Clumps_Per_Cycle', maxClumpsPerCycle])
            writer.writerow(['Clump_Type_1_Percent', clumpTypePercentages[0]])
            writer.writerow(['Clump_Type_2_Percent', clumpTypePercentages[1]])
            writer.writerow(['Clump_Type_3_Percent', clumpTypePercentages[2]])
            writer.writerow(['Clump_Type_4_Percent', clumpTypePercentages[3]])
            writer.writerow(['Clump_Type_5_Percent', clumpTypePercentages[4]])
            writer.writerow([])

            writer.writerow(['Conveyor parameters'])
            writer.writerow(['Conveyor_Speed_m_s', conveyor_speed])
            writer.writerow(['Conveyor_Tilt_deg', tilt_angle_deg])
            writer.writerow([])

            writer.writerow(['Wearbody parameters'])
            writer.writerow(['Wearbody_Force_Magnitude_N', wearbody_force_magnitude])
            writer.writerow(['Wearbody_Force_Contact_Threshold', wearbody_force_contact_threshold])
            writer.writerow(['Wearbody_Force_Enabled', wearbody_force_enabled])
            writer.writerow(['Wearbody_Force_Active', wearbody_force_active])
            writer.writerow([])

            writer.writerow(['Archard equation'])
            writer.writerow(['K_Archard', k_archard])
            writer.writerow(['Wearbody_Hardness_Pa', wearbody_hardness])
            writer.writerow(['Target_Volume_Loss_mm3', TARGET_VOLUME_LOSS_MM3])
            writer.writerow(['Target_Mass_Loss_g', TARGET_MASS_LOSS_G])
            writer.writerow(['State_Period_Iterations', STATE_ITER_PERIOD])
            writer.writerow([])

            writer.writerow(['Current state'])
            writer.writerow(['Wearbody_Released', wearbodyReleased])
            writer.writerow(['Active_Clumps', countActiveClumps()])
            writer.writerow(['Total_Clumps_Generated', clumpsGenerated])
            writer.writerow(['Total_Clumps_Removed', clumpsRemoved])
            writer.writerow(['Total_Spheres_Generated', numSpheresGen])
            writer.writerow(['Wearbody_Contacts', countWearbodyContacts()])
            writer.writerow(['Initial_Wearbody_Volume_mm3', initialWearbodyVolume * 1e9])
            writer.writerow(['Initial_Wearbody_Mass_g', initialWearbodyMass * 1000])
            writer.writerow(['Total_Volume_Removed_mm3', totalVolumeRemoved * 1e9])
            writer.writerow(['Total_Mass_Lost_g', totalMassLost * 1000])
            writer.writerow(['Volume_Loss_Percent', (totalVolumeRemoved / initialWearbodyVolume) * 100])
            writer.writerow(['Progress_To_Target_Percent', (totalVolumeRemoved * 1e9 / TARGET_VOLUME_LOSS_MM3) * 100]
            writer.writerow(['Max_Facet_Wear_mm2', max(facet_wear.values()) * 1e6 if facet_wear else 0])
```

```
    'grid': '#2C3E50',
    'text': '#2C3E50'
}

CLUMP_COLORS = ['#A8D8EA', '#FFB3BA', '#BFFF7F', '#FFE5B4', '#D7B3FF', '#FFC3A0']

last_plot_update_iter = 0
plot_update_interval = 200

def setupVTKExport():
    vtk_dir = 'output/vtk'
    if not os.path.exists(vtk_dir):
        os.makedirs(vtk_dir)

    def exportSeparateObjectVTKs():
        if O.iter % 1000 != 0:
            return

        timestamp = f"{O.iter:06d}"

        try:
            exportWearbodyVTK(timestamp)
            exportConveyorVTK(timestamp)
            exportClumpsVTK(timestamp)
        except Exception as e:
            print(f"VTK export error: {e}")

    def exportWearbodyVTK(timestamp):
        vtk_filename = f"output/vtk/wearbody_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'wear_mm3': [], 'velocity_magnitude': []}
        cell_data = {'facet_id': [], 'contact_count': [], 'area_mm2': []}
        vertex_count = 0

        for facet_id in wearbodyIds:
            try:
                facet = O.bodies[facet_id]
                if facet is None:
                    continue

                facet_vertices = [facet.state.pos + facet.shape.vertices[i] for i in range(3)]

                for v in facet_vertices:
                    all_vertices.append([v[0], v[1], v[2]])

                all_cells.append([3, vertex_count, vertex_count + 1, vertex_count + 2])
                all_cell_types.append(5)
                vertex_count += 3

                v1 = facet_vertices[1] - facet_vertices[0]
                v2 = facet_vertices[2] - facet_vertices[0]
                area = 0.5 * v1.cross(v2).norm()

                wear_value = facet_wear.get(facet_id, 0.0) * 1e9
                vel_mag = facet.state.vel.norm()

                for _ in range(3):
                    point_data['wear_mm3'].append(wear_value)
                    point_data['velocity_magnitude'].append(vel_mag)

                contact_count = sum(1 for i in facet.intrs() if i.isReal)
                cell_data['facet_id'].append(facet_id)
                cell_data['contact_count'].append(contact_count)
                cell_data['area_mm2'].append(area * 1e6)

            except (IndexError, AttributeError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Wearbody with Wear Data")

    def exportConveyorVTK(timestamp):
        vtk_filename = f"output/vtk/conveyor_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': []}
        cell_data = {'facet_id': [], 'contact_count': [], 'is_moving_belt': []}
        vertex_count = 0

        for facet_id in fctIds:
            try:
                facet = O.bodies[facet_id]
                if facet is None:
                    continue

                facet_vertices = [facet.state.pos + facet.shape.vertices[i] for i in range(3)]

                for v in facet_vertices:
                    all_vertices.append([v[0], v[1], v[2]])

                all_cells.append([3, vertex_count, vertex_count + 1, vertex_count + 2])
                all_cell_types.append(5)
                vertex_count += 3

                for _ in range(3):
                    point_data['velocity_magnitude'].append(0.0)

                contact_count = sum(1 for i in facet.intrs() if i.isReal)
                is_moving_belt = 1 if facet_id in conveyor_facets else 0

                cell_data['facet_id'].append(facet_id)
                cell_data['contact_count'].append(contact_count)
                cell_data['is_moving_belt'].append(is_moving_belt)

            except (IndexError, AttributeError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Conveyor Box")

    def exportClumpsVTK(timestamp):
        vtk_filename = f"output/vtk/clumps_{timestamp}.vtk"

        all_vertices = []
        all_cells = []
        all_cell_types = []
        point_data = {'velocity_magnitude': [], 'clump_type': [], 'radius_mm': [],
                      'angular_velocity_magnitude': []}
        cell_data = {'body_id': [], 'contact_count': [], 'mass_kg': []}
        vertex_count = 0

        clump_count = 0
        for body in O.bodies:
            if body is None or not hasattr(body, 'shape') or body.mask != 3:
                continue

            try:
                if hasattr(body.shape, 'radius'):
                    pos = body.state.pos
                    radius = body.shape.radius
                    vel_mag = body.state.vel.norm()
                    angvel_mag = body.state.angVel.norm()

                    all_vertices.append([pos[0], pos[1], pos[2]])
                    all_cells.append([1, vertex_count])
                    all_cell_types.append(1)
                    vertex_count += 1

                    clump_type = 0
                    if hasattr(body.shape, 'color'):
                        color = body.shape.color
                        if abs(color[1] - 0.8) < 0.1 and abs(color[0] - 0.2) < 0.1:
                            clump_type = 1
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.2) < 0.1:
                            clump_type = 2
                        elif abs(color[2] - 0.8) < 0.1 and abs(color[0] - 0.2) < 0.1:
                            clump_type = 3
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[1] - 0.8) < 0.1:
                            clump_type = 4
                        elif abs(color[0] - 0.8) < 0.1 and abs(color[2] - 0.8) < 0.1:
                            clump_type = 5

                    point_data['velocity_magnitude'].append(vel_mag)
                    point_data['clump_type'].append(clump_type)
                    point_data['radius_mm'].append(radius * 1000)
                    point_data['angular_velocity_magnitude'].append(angvel_mag)

                    contact_count = sum(1 for i in body.intrs() if i.isReal)
                    cell_data['body_id'].append(body.id)
                    cell_data['contact_count'].append(contact_count)
                    cell_data['mass_kg'].append(body.state.mass)

                    clump_count += 1

                    if clump_count > 3000:
                        break

            except (AttributeError, IndexError):
                continue

        if all_vertices:
            writeVTKFile(vtk_filename, all_vertices, all_cells, all_cell_types,
                         point_data, cell_data, "Clump Particles")

    def writeVTKFile(filename, vertices, cells, cell_types, point_data, cell_data, description):
        with open(filename, 'w') as f:
            f.write("# vtk DataFile Version 3.0\n")
            f.write(f"{description}\n")
            f.write("ASCII\n")
            f.write("DATASET UNSTRUCTURED_GRID\n")

            f.write(f"POINTS {len(vertices)} float\n")
            for v in vertices:
                f.write(f"{v[0]:.6e} {v[1]:.6e} {v[2]:.6e}\n")

            total_cell_size = sum(len(cell) for cell in cells)
            f.write(f"CELLS {len(cells)} {total_cell_size}\n")
            for cell in cells:
                f.write(" ".join(map(str, cell)) + "\n")

            f.write(f"CELL_TYPES {len(cells)}\n")
            for cell_type in cell_types:
                f.write(f"{cell_type}\n")

            if point_data:
                f.write(f"POINT_DATA {len(vertices)}\n")
                for key, values in point_data.items():
                    if values:
                        data_type = "float" if isinstance(values[0], float) else "int"
                        f.write(f"SCALARS {key} {data_type} 1\n")
                        f.write("LOOKUP_TABLE default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n" if isinstance(val, float) else f"{val}\n")

            if cell_data:
                f.write(f"CELL_DATA {len(cells)}\n")
                for key, values in cell_data.items():
                    if values:
                        data_type = "float" if isinstance(values[0], float) else "int"
                        f.write(f"SCALARS {key} {data_type} 1\n")
                        f.write("LOOKUP_TABLE default\n")
                        for val in values:
                            f.write(f"{val:.6e}\n" if isinstance(val, float) else f"{val}\n")

    return exportSeparateObjectVTKs

def completeSimulationExport():
    export_dir = "complete_simulation_export"
    if not os.path.exists(export_dir):
        os.makedirs(export_dir)

    timestamp = f"{O.iter:06d}_{int(O.time*1000):06d}"
    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    print("Exporting Complete Simulation Data to .CSV")

    if clump_size_data:
        equiv_diameters_um = [data[2] * 2 * 1e6 for data in clump_size_data]
        d10 = np.percentile(equiv_diameters_um, 10)
        d50 = np.percentile(equiv_diameters_um, 50)
        d90 = np.percentile(equiv_diameters_um, 90)
        mean_diameter = np.mean(equiv_diameters_um)
        std_diameter = np.std(equiv_diameters_um)
        min_diameter = min(equiv_diameters_um)
        max_diameter = max(equiv_diameters_um)

        clump_types = [data[1] for data in clump_size_data]
        type_counts = [clump_types.count(i) for i in range(5)]
        type_percentages = [count/len(clump_types)*100 for count in type_counts]
    else:
        d10 = d50 = d90 = mean_diameter = std_diameter = 0
        min_diameter = max_diameter = 0
        type_counts = [0] * 5
        type_percentages = [0] * 5

    avg_normal_force = sum(state_period_normal_forces) / len(state_period_normal_forces) if state_period_normal_force
    avg_sliding_distance = sum(state_period_sliding_distances) / len(state_period_sliding_distances) if state_period_
    current_wear_rate_mm3_per_s = (state_period_volume_removed * 1e9) / (STATE_ITER_PERIOD * O.dt) if O.dt > 0 else 0

    max_wear_depth_um, avg_wear_depth_um = calculateWearDepth()

    TARGET_MASS_LOSS_G = TARGET_VOLUME_LOSS_M3 * O.materials[steelId].density * 1000

    main_csv_filename = os.path.join(export_dir, f"main_simulation_data_{timestamp}.csv")
    try:
        with open(main_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow(['Export_Timestamp', current_time])
            writer.writerow(['Export_Iteration', O.iter])
            writer.writerow(['Export_Simulation_Time_s', f"{O.time:.6f}"])
            writer.writerow(['Real_Simulation_Duration_s', f"{time.time() - simulation_start_time:.2f}"])
            writer.writerow(['Timestep_s', f"{O.dt:.6e}"])
            writer.writerow([])

            writer.writerow(['Material properties'])
            writer.writerow(['regolith_Young_Modulus_Pa', O.materials[regolithId].young])
            writer.writerow(['regolith_Density_kg_m3', O.materials[regolithId].density])
            writer.writerow(['regolith_Poisson_Ratio', O.materials[regolithId].poisson])
            writer.writerow(['regolith_Friction_Angle_deg', math.degrees(O.materials[regolithId].frictionAngle)])
            writer.writerow(['Steel_Young_Modulus_Pa', O.materials[steelId].young])
            writer.writerow(['Steel_Density_kg_m3', O.materials[steelId].density])
            writer.writerow(['Steel_Poisson_Ratio', O.materials[steelId].poisson])
            writer.writerow(['Steel_Friction_Angle_deg', math.degrees(O.materials[steelId].frictionAngle)])
            writer.writerow([])

            writer.writerow(['Particle parameters'])
            writer.writerow(['Particle_Radius_Min_um', rMin * 1e6])
            writer.writerow(['Particle_Radius_Max_um', rMax * 1e6])
            writer.writerow(['Particle_Diameter_Min_um', rMin * 2 * 1e6])
            writer.writerow(['Particle_Diameter_Max_um', rMax * 2 * 1e6])
            writer.writerow(['Mass_Flow_Rate_kg_s', massFlowRate])
            writer.writerow(['Max_Active_Clumps', MAX_ACTIVE_CLUMPS])
            writer.writerow(['Max_Clumps_Per_Cycle', maxClumpsPerCycle])
            writer.writerow(['Clump_Type_1_Percent', clumpTypePercentages[0]])
            writer.writerow(['Clump_Type_2_Percent', clumpTypePercentages[1]])
            writer.writerow(['Clump_Type_3_Percent', clumpTypePercentages[2]])
            writer.writerow(['Clump_Type_4_Percent', clumpTypePercentages[3]])
            writer.writerow(['Clump_Type_5_Percent', clumpTypePercentages[4]])
            writer.writerow([])

            writer.writerow(['Conveyor parameters'])
            writer.writerow(['Conveyor_Speed_m_s', conveyor_speed])
            writer.writerow(['Conveyor_Tilt_deg', tilt_angle_deg])
            writer.writerow([])

            writer.writerow(['Wearbody parameters'])
            writer.writerow(['Wearbody_Force_Magnitude_N', wearbody_force_magnitude])
            writer.writerow(['Wearbody_Force_Contact_Threshold', wearbody_force_contact_threshold])
            writer.writerow(['Wearbody_Force_Enabled', wearbody_force_enabled])
            writer.writerow(['Wearbody_Force_Active', wearbody_force_active])
            writer.writerow([])

            writer.writerow(['Archard equation'])
            writer.writerow(['K_Archard', k_archard])
            writer.writerow(['Wearbody_Hardness_Pa', wearbody_hardness])
            writer.writerow(['Target_Volume_Loss_mm3', TARGET_VOLUME_LOSS_MM3])
            writer.writerow(['Target_Mass_Loss_g', TARGET_MASS_LOSS_G])
            writer.writerow(['State_Period_Iterations', STATE_ITER_PERIOD])
            writer.writerow([])

            writer.writerow(['Current state'])
            writer.writerow(['Wearbody_Released', wearbodyReleased])
            writer.writerow(['Active_Clumps', countActiveClumps()])
            writer.writerow(['Total_Clumps_Generated', clumpsGenerated])
            writer.writerow(['Total_Clumps_Removed', clumpsRemoved])
            writer.writerow(['Total_Spheres_Generated', numSpheresGen])
            writer.writerow(['Wearbody_Contacts', countWearbodyContacts()])
            writer.writerow(['Initial_Wearbody_Volume_mm3', initialWearbodyVolume * 1e9])
            writer.writerow(['Initial_Wearbody_Mass_g', initialWearbodyMass * 1000])
            writer.writerow(['Total_Volume_Removed_mm3', totalVolumeRemoved * 1e9])
            writer.writerow(['Total_Mass_Lost_g', totalMassLost * 1000])
            writer.writerow(['Volume_Loss_Percent', (totalVolumeRemoved / initialWearbodyVolume) * 100])
            writer.writerow(['Progress_To_Target_Percent', (totalVolumeRemoved * 1e9 / TARGET_VOLUME_LOSS_MM3) * 100]
            writer.writerow(['Max_Facet_Wear_mm2', max(facet_wear.values()) * 1e6 if facet_wear else 0])
```

```
            writer.writerow(['Max_Wear_Depth_um', max_wear_depth_um])
            writer.writerow(['Avg_Wear_Depth_um', avg_wear_depth_um])
            writer.writerow(['Estimated_Time_To_Target', estimateTimeToTarget()])
            writer.writerow([])

            writer.writerow(['Current period'])
            writer.writerow(['Current_Period_Volume_Removed_mm3', state_period_volume_removed * 1e9])
            writer.writerow(['Current_Wear_Rate_mm3_per_s', current_wear_rate_mm3_per_s])
            writer.writerow(['Avg_Normal_Force_N', avg_normal_force])
            writer.writerow(['Avg_Sliding_Distance_m', avg_sliding_distance])
            writer.writerow(['Avg_Sliding_Distance_mm', avg_sliding_distance * 1000])
            writer.writerow(['Normal_Force_Samples', len(state_period_normal_forces)])
            writer.writerow(['Sliding_Distance_Samples', len(state_period_sliding_distances)])
            writer.writerow([])

            writer.writerow(['Size distribution '])
            writer.writerow(['Total_Clumps_Analyzed', len(clump_size_data)])
            writer.writerow(['Equivalent_Diameter_D10_um', d10])
            writer.writerow(['Equivalent_Diameter_D50_um', d50])
            writer.writerow(['Equivalent_Diameter_D90_um', d90])
            writer.writerow(['Mean_Diameter_um', mean_diameter])
            writer.writerow(['Std_Diameter_um', std_diameter])
            writer.writerow(['Min_Diameter_um', min_diameter])
            writer.writerow(['Max_Diameter_um', max_diameter])
            writer.writerow(['Clump_Type_1_Count', type_counts[0]])
            writer.writerow(['Clump_Type_2_Count', type_counts[1]])
            writer.writerow(['Clump_Type_3_Count', type_counts[2]])
            writer.writerow(['Clump_Type_4_Count', type_counts[3]])
            writer.writerow(['Clump_Type_5_Count', type_counts[4]])
            writer.writerow(['Clump_Type_1_Actual_Percent', type_percentages[0]])
            writer.writerow(['Clump_Type_2_Actual_Percent', type_percentages[1]])
            writer.writerow(['Clump_Type_3_Actual_Percent', type_percentages[2]])
            writer.writerow(['Clump_Type_4_Actual_Percent', type_percentages[3]])
            writer.writerow(['Clump_Type_5_Actual_Percent', type_percentages[4]])

        print(f"Main simulation data exported: {main_csv_filename}")
    except Exception as e:
        print(f"Main simulation export failed: {e}")

    time_series_csv_filename = os.path.join(export_dir, f"time_series_data_{timestamp}.csv")
    try:
        with open(time_series_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow([
                'Sim_time_hh_mm_ss', 'Simulation_virt_time_s', 'Iterations',
                'Total_volume_removed_mm3', 'Total_weight_removed_g', 'Progress_to_target_percent',
                'Average_normal_force_N', 'Average_sliding_distance_mm',
                'Volume_removed_this_period_mm3', 'Current_wear_rate_mm3_per_s',
                'Max_wear_depth_um', 'Avg_wear_depth_um'
            ])

            for data_point in time_series_data:
                writer.writerow([
                    data_point['sim_time_formatted'],
                    f"{data_point['virt_time']:.6f}",
                    data_point['iterations'],
                    f"{data_point['total_volume_removed_mm3']:.9f}",
                    f"{data_point['total_weight_removed_g']:.9f}",
                    f"{data_point['progress_to_target_percent']:.6f}",
                    f"{data_point['avg_normal_force']:.6f}",
                    f"{data_point['avg_sliding_distance_mm']:.9f}",
                    f"{data_point['volume_removed_this_period_mm3']:.9f}",
                    f"{data_point['current_wear_rate_mm3_per_s']:.9f}",
                    f"{data_point['max_wear_depth_um']:.9f}",
                    f"{data_point['avg_wear_depth_um']:.9f}"
                ])

        print(f"Time series data exported: {time_series_csv_filename} ({len(time_series_data)} data points)")
    except Exception as e:
        print(f"Time series export failed: {e}")

    size_dist_csv_filename = os.path.join(export_dir, f"size_distribution_data_{timestamp}.csv")
    try:
        with open(size_dist_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow([
                'Base_Diameter_um', 'Clump_Type', 'Equivalent_Diameter_um',
                'Total_Volume_mm3', 'Max_Diameter_um'
            ])

            for base_radius, clump_type, equiv_radius, total_volume, max_radius in clump_size_data:
                writer.writerow([
                    f"{base_radius * 2 * 1e6:.6f}",
                    clump_type,
                    f"{equiv_radius * 2 * 1e6:.6f}",
                    f"{total_volume * 1e9:.9f}",
                    f"{max_radius * 2 * 1e6:.6f}"
                ])

        print(f"Size distribution data exported: {size_dist_csv_filename} ({len(clump_size_data)} clumps)")
    except Exception as e:
        print(f"Size distribution export failed: {e}")

    templates_csv_filename = os.path.join(export_dir, f"clump_templates_{timestamp}.csv")
    try:
        with open(templates_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)

            writer.writerow([
                'Clump_Type', 'Sphere_Index', 'Relative_X', 'Relative_Y', 'Relative_Z', 'Radius_Factor'
            ])

            for clump_type, template in enumerate(clumpTemplates):
                for sphere_idx, (pos, radius_factor) in enumerate(template):
                    writer.writerow([
                        clump_type,
                        sphere_idx,
                        f"{pos[0]:.6f}",
                        f"{pos[1]:.6f}",
                        f"{pos[2]:.6f}",
                        f"{radius_factor:.6f}"
                    ])

        print(f"Clump templates exported: {templates_csv_filename}")
    except Exception as e:
        print(f"Clump templates export failed: {e}")

    if facet_wear:
        wear_facets_csv_filename = os.path.join(export_dir, f"wear_per_facet_{timestamp}.csv")
        try:
            with open(wear_facets_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
                writer = csv.writer(csvfile)

                writer.writerow(['Facet_ID', 'Wear_mm2', 'Center_X_mm', 'Center_Y_mm', 'Center_Z_mm'])

                for facet_id, wear_value in facet_wear.items():
                    if facet_id in facet_center_positions:
                        center = facet_center_positions[facet_id]
                        writer.writerow([
                            facet_id,
                            f"{wear_value * 1e6:.9f}",
                            f"{center[0] * 1000:.6f}",
                            f"{center[1] * 1000:.6f}",
                            f"{center[2] * 1000:.6f}"
                        ])

            print(f"Wear per facet data exported: {wear_facets_csv_filename} ({len(facet_wear)} facets)")
        except Exception as e:
            print(f"Wear per facet export failed: {e}")

    print(".CSV exported")

fig = plt.figure(figsize=(16, 12))
gs = fig.add_gridspec(3, 2, height_ratios=[1, 1, 1], width_ratios=[1, 1])

ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[1, 0])

ax3 = fig.add_subplot(gs[0, 1])
ax4 = fig.add_subplot(gs[1, 1])
ax5 = fig.add_subplot(gs[2, 1])

fig.suptitle('Wear Simulation - Particle Diameter Distribution (um)', fontsize=16, fontweight='bold')

plt.rcParams.update({
    'font.size': 10,
    'axes.grid': True,
    'grid.alpha': 0.6,
    'grid.color': '#2C3E50'
})

volume_loss_line, = ax1.plot([], [], color=COLORS['primary'], linewidth=2, label='Volume Loss')
target_line = ax1.axhline(y=TARGET_VOLUME_LOSS_MM3, color='green', linestyle='--', linewidth=2, label=f'Target ({TARG
ax1.set_title('Wearbody Volume Loss', fontweight='bold')
```



```
ax1.set_xlabel('Simulation Time (s)')
ax1.set_ylabel('Volume Loss (mm³)')
ax1.legend()
ax1.grid(True)

wear_depth_line, = ax2.plot([], [], color=COLORS['accent'], linewidth=2, label='Max Wear Depth')
ax2.set_title('Maximum Wear Depth', fontweight='bold')
ax2.set_xlabel('Simulation Time (s)')
ax2.set_ylabel('Wear Depth (um)')
ax2.legend()
ax2.grid(True)

ax3.set_title('Equivalent Diameter Distribution', fontweight='bold')
ax3.set_xlabel('Equivalent Diameter (um)')
ax3.set_ylabel('Frequency')
ax3.grid(True)

ax4.set_title('Cumulative Size Distribution', fontweight='bold')
ax4.set_xlabel('Equivalent Diameter (um)')
ax4.set_ylabel('Cumulative Percentage Finer (%)')
ax4.grid(True)

ax5.set_title('Clump Type Distribution', fontweight='bold')

plt.tight_layout()
plt.show(block=False)

def update_plot():
    global last_plot_update_iter

    try:
        if len(wear_data) == 0:
            return

        times = [item[0] for item in wear_data]
        volumes_m3 = [item[1] for item in wear_data]
        max_depths_um = [item[4] for item in wear_data]

        if len(times) == 0:
            return

        volume_mm3 = [v * 1e9 for v in volumes_m3]

        volume_loss_line.set_data(times, volume_mm3)
        wear_depth_line.set_data(times, max_depths_um)

        if times:
            ax1.set_xlim(0, max(times) * 1.1)
            ax2.set_xlim(0, max(times) * 1.1)

        max_vol = max(volume_mm3) if volume_mm3 else 0
        max_depth = max(max_depths_um) if max_depths_um else 0
        ax1.set_ylim(0, max(TARGET_VOLUME_LOSS_MM3 * 1.5, max_vol * 1.1))
        ax2.set_ylim(0, max_depth * 1.1 if max_depth > 0 else 1.0)

        update_size_distribution_plots()

        current_iter = O.iter
        if current_iter - last_plot_update_iter >= plot_update_interval:
            try:
                fig.canvas.draw_idle()
                fig.canvas.flush_events()
                last_plot_update_iter = current_iter
            except:
                pass

    except Exception as e:
        print(f"Plot update error: {e}")

def update_size_distribution_plots():
    if len(clump_size_data) < 10:
        return

    try:
        equiv_diameters_um = [data[2] * 2 * 1e6 for data in clump_size_data]
        clump_types = [data[1] for data in clump_size_data]

        ax3.clear()
        ax4.clear()
        ax5.clear()

        d10 = np.percentile(equiv_diameters_um, 10)
        d50 = np.percentile(equiv_diameters_um, 50)
        d90 = np.percentile(equiv_diameters_um, 90)
        mean_diameter = np.mean(equiv_diameters_um)

        ax3.hist(equiv_diameters_um, bins=30, alpha=0.8, color=COLORS['primary'], edgecolor='black')
        ax3.axvline(d10, color='red', linestyle='--', linewidth=2, label=f'D10 = {d10:.1f} um')
        ax3.axvline(d50, color='orange', linestyle='--', linewidth=2, label=f'D50 = {d50:.1f} um')
        ax3.axvline(d90, color='red', linestyle='--', linewidth=2, label=f'D90 = {d90:.1f} um')
        ax3.axvline(mean_diameter, color='purple', linestyle='-', linewidth=2, label=f'Mean = {mean_diameter:.1f} um'
        ax3.set_xlabel('Equivalent Diameter (um)')
        ax3.set_ylabel('Frequency')
        ax3.set_title('Equivalent Diameter Distribution', fontweight='bold')
        ax3.legend()
        ax3.grid(True)

        sorted_diameters = np.sort(equiv_diameters_um)
        cumulative_percent = np.arange(1, len(sorted_diameters) + 1) / len(sorted_diameters) * 100

        ax4.plot(sorted_diameters, cumulative_percent, color=COLORS['primary'], linewidth=2)
        ax4.axhline(10, color='red', linestyle='--', alpha=0.7)
        ax4.axhline(50, color='orange', linestyle='--', alpha=0.7)
        ax4.axhline(90, color='red', linestyle='--', alpha=0.7)
        ax4.axvline(d10, color='red', linestyle='--', alpha=0.7, label=f'D10 = {d10:.1f} um')
        ax4.axvline(d50, color='orange', linestyle='--', alpha=0.7, label=f'D50 = {d50:.1f} um')
        ax4.axvline(d90, color='red', linestyle='--', alpha=0.7, label=f'D90 = {d90:.1f} um')
        ax4.plot(d10, 10, 'o', color='red', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax4.plot(d50, 50, 'o', color='orange', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax4.plot(d90, 90, 'o', color='red', markersize=6, markeredgecolor='white', markeredgewidth=2)
        ax4.set_xlabel('Equivalent Diameter (um)')
        ax4.set_ylabel('Cumulative Percentage Finer (%)')
        ax4.set_title('Cumulative Size Distribution', fontweight='bold')
        ax4.legend()
        ax4.grid(True)
        ax4.set_ylim(0, 100)

        type_counts = [clump_types.count(i) for i in range(5)]
        type_percentages = [count/len(clump_types)*100 for count in type_counts]
        active_types = [(i, pct) for i, pct in enumerate(type_percentages) if pct > 0]

        if active_types:
            active_indices = [item[0] for item in active_types]
            active_percentages = [item[1] for item in active_types]
            active_labels = [f'Type {i+1}' for i in active_indices]
            active_colors = [CLUMP_COLORS[i] for i in active_indices]

            ax5.pie(active_percentages, labels=active_labels, colors=active_colors, autopct='%1.1f%%', startangle=90)
            ax5.set_title('Clump Type Distribution', fontweight='bold')
        else:
            ax5.text(0.5, 0.5, 'No Data Available', ha='center', va='center', transform=ax5.transAxes, fontsize=12, f
            ax5.set_title('Clump Type Distribution', fontweight='bold')

    except Exception as e:
        print(f"Size distribution plot error: {e}")

def cleanup():
    try:
        plt.close(fig)
    except:
        pass

# 8. Sim engines
O.bodyRemoved = onBodyRemoved

exportSeparateObjectVTKs = setupVTKExport()

resetStatePeriodTracking()

critical_dt = min(
    utils.PWaveTimeStep(),
    utils.SpherePWaveTimeStep(rMin, O.materials[regolithId].density, O.materials[regolithId].young)
)

O.engines = [
    ForceResetter(),
    InsertionSortCollider([Bo1_Sphere_Aabb(), Bo1_Facet_Aabb()], verletDist=12e-3),
    InteractionLoop(
        [Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom()],
        [Ip2_FrictMat_FrictMat_MindlinPhys(
            en=MatchMaker(matches=((steelId, regolithId, 0.3), (regolithId, regolithId, 0.4), (steelId, steelId, 0.3)
            es=MatchMaker(matches=((steelId, regolithId, 0.3), (regolithId, regolithId, 0.4), (steelId, steelId, 0.3)
        )],
        [Law2_ScGeom_MindlinPhys_Mindlin(label='contactLaw', neverErase=False)]
    ),
    NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.6),
```

```
    PyRunner(command='applyConveyorForces()', iterPeriod=100, label='conveyorController'),
    PyRunner(command='stabilizeParticles()', iterPeriod=100, label='particleStabilizer'),
    PyRunner(command='generateClumps()', iterPeriod=200, label='clumpGenerator'),
    PyRunner(command='checkWearbodyTrigger()', iterPeriod=50, label='wearbodyTrigger'),
    PyRunner(command='applyWearbodyForce()', iterPeriod=10, label='wearbodyForceApplier'),
    PyRunner(command='trackWear()', iterPeriod=200, label='wearTracker'),
    PyRunner(command='monitorWearbody()', iterPeriod=200, label='wearbodyMonitor'),
    PyRunner(command='exportSeparateObjectVTKs()', iterPeriod=4000, label='vtkSeparateExporter'),
    PyRunner(command='state()', iterPeriod=STATE_ITER_PERIOD, label='stateMonitor'),
    DomainLimiter(lo=(-5e-3, -11e-3, -.2e-3), hi=(28e-3, 11e-3, 13e-3), iterPeriod=200, label='domLim'),
]

# 9. Timestep and more
O.dt = 1.2 * critical_dt
print(f"Timestep: {O.dt} (Critical dt: {critical_dt})")

lastMassUpdateIter = O.iter
countActiveClumps()

O.exitHook = "cleanup()"

createConveyor()
createWearbody()

facet_wear = {facet_id: 0.0 for facet_id in wearbodyIds}

try:
    from yade import qt
    v = qt.View()
    v.upVector = (0, 0, 1)
    v.viewDir = (-1, 0, -.3)
    v.center(median=False)
except ImportError:
    pass

# 10. Startup info
print("Available commands:")
print("- completeSimulationExport()   : Export complete data in CSV format (Excel-ready!)")
print("- clumptemplate()              : Create visualization of all clump types")

O.saveTmp()
```

11 HALLGATÓI NYILATKOZAT

MATE Szervezeti és Működési Szabályzat
III. Hallgatói Követelményrendszer
III.1. Tanulmányi és Vizsgaszabályzat
6.13. sz. függeléke: A MATE egységes szakdolgozat / diplomadolgozat / záródolgozat / portfólió készítési útmutatója
4.2. sz. melléklete: Nyilatkozat a záródolgozat/szakdolgozat/diplomadolgozat/portfólió nyilvános hozzáféréséről és eredetiségéről (módosítva: 2025. október 16.)

NYILATKOZAT

a diplomadolgozat[1] nyilvános hozzáféréséről és eredetiségéről

A hallgató neve:	Szabó Bence
A Hallgató Neptun kódja:	JDCLMZ
A dolgozat címe:	Szemcsehalmazok okozta kopás modellezése
A megjelenés éve:	2025
A konzulens intézetének neve:	Műszaki Intézet
A konzulens tanszékének a neve:	Gépszerkezettani Tanszék

Kijelentem, hogy az általam benyújtott diplomadolgozat[2] egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, és az irodalomjegyzékben szerepeltettem. Továbbá kijelentem, hogy a dolgozat elkészítése során alkalmazott mesterséges intelligencia-eszközök (pl. szöveggenerálás, nyelvi javítás, fordítás, adatelemzés) használata nem helyettesítette a saját kutatási és alkotói munkámat, azok alkalmazását a források között vagy a módszertani részben feltüntettem, és a szakmai-etikai elvárásoknak megfelelően jártam el.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe. Tudomásul veszem, hogy a megvédett és

- nem titkosított dolgozat a védést követően
- titkosításra engedélyezett dolgozat a benyújtásától számított 5 év eltelte után

nyilvánosan elérhető és kereshető lesz az Egyetem könyvári repozitori rendszerében.

Kelt: Gödöllő, 2025 év október hó 29 nap

Szabó Bence
Hallgató aláírása

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.

Hallgatók, doktoranduszok nyilatkozata mesterséges intelligencia (MI) alkalmazásáról

1. Általános adatok

Hallgató neve:	**Szabó Bence**
Neptun-kódja:	**JDCLMZ**
Képzési szint (a megfelelőt jelölje X-szel):	☐ BSc/BA ☒ MSc/MA ☐ Doktori (PhD) ☐ Egyéb: .....
Tantárgy neve/kódja*:	**Diplomamunka készítés 2.**
A munka címe:	**Szemcsehalmazok okozta kopás modellezése**

* doktori értekezés esetén nem kitöltendő

2. Nyilatkozat az MI használatáról

Alulírott, etikai felelősségem teljes tudatában az alábbi nyilatkozatot teszem:

(Kérjük, válasszon egyet az alábbi lehetőségek közül!)

☐ A) Nem alkalmaztam mesterséges intelligencia rendszert vagy szolgáltatást.

(Amennyiben ezt jelölte, a további táblázatok kitöltése nem szükséges.)

☒ B) Alkalmaztam mesterséges intelligencia rendszert vagy szolgáltatást.

(Kérjük, töltse ki a vonatkozó táblázatokat!)

3. A mesterséges intelligencia használatának részletezése

I. TÁBLÁZAT: Asszisztensi vagy kisebb mértékű felhasználás (pl. fordítás, nyelvi korrektúra, ötletelés stb.)

(Ezen felhasználások esetében a konkrét promptok és válaszok csatolása nem szükséges.)

A felhasználás célja	Alkalmazott MI-eszköz neve és verziója	Érintett rész (ha nem a szöveg egészére vonatkozik)
Szimuláció script ellenőrzése	Claude Sonnet 4	

II. TÁBLÁZAT: Jelentős tartalmi hozzájárulás (pl. egy teljes ábra vagy egy hosszabb szövegrész generálása)

*(Ezekben az esetekben a felhasznált kulcsfontosságú promptok és az MI által adott nyers válaszok dokumentálása és a munka **mellékletében való csatolása szükséges**.)*

A felhasználás célja	Alkalmazott MI-eszköz neve,	Az érintett fejezet / ábra / táblázat pontos sorszáma	A prompt-naplót tartalmazó melléklet

	verziója, elérhetősége		bejegyzésének sorszáma

3/A. Oktató által előírt kiegészítő szabályok (ha vannak)

Amennyiben az adott tantárgy oktatója vagy témavezetője az MI-eszközök használatára vonatkozóan külön szabályokat vagy elvárásokat határozott meg, kérjük, az alábbi mezőben foglalja össze ezeket:

Pl. az MI használatának tilalma bizonyos feladattípusokra; csak konkrét eszköz használata engedélyezett; eltérő hivatkozási elvárások; dokumentációs forma stb.

Oktató vagy témavezető által előírt szabályok:

.....

.....

.....

.....

4. Minden hallgatóra vonatkozó nyilatkozat:

Kijelentem, hogy az MI által esetlegesen generált tartalmakat minden esetben kritikailag felülvizsgáltam, szerkesztettem és a munkába illesztettem. A leadott munka minden eleméért, annak eredetiségéért és tudományos helytállóságáért teljes körű felelősséget vállalok. Tudomásul veszem, hogy a Magyar Agrár- és Élettudományi Egyetem a benyújtott munkát mesterséges intelligencia detektorral ellenőrizheti, és eljárást kezdeményezhet, amennyiben a nyilatkozatom valótlan vagy hiányos.

Kelt: Gödöllő, 2025. október hó 29 nap

Szabó Bence .....

Hallgató aláírása

.....

Konzulens/Témavezető aláírása

12 KONZULENSI NYILATKOZAT

NYILATKOZAT

Szabó Bence (név) (hallgató Neptun azonosítója: JDCLMZ) konzulenseként nyilatkozom arról, hogy a záródolgozatot/szakdolgozatot/diplomadolgozatot/portfóliót[1] áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A diplomadolgozatot a záróvizsgán történő védésre **javaslom / nem javaslom**[2].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[3]

Kelt2025 év október hó 30. nap

belső konzulens

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő aláhúzandó.
[3] A megfelelő aláhúzandó.

MATE Szervezeti és Működési Szabályzat
III. Hallgatói Követelményrendszer
III.1. Tanulmányi és Vizsgaszabályzat
6.13. sz. függeléke: A MATE egységes szakdolgozat / diplomadolgozat / záródolgozat / portfólió készítési útmutatója
7. sz. melléklete: Műszaki Intézet külső konzulensi nyilatkozat

KÜLSŐ KONZULENSI NYILATKOZAT

Szabó Bence (név) (hallgató Neptun azonosítója: JDCLMZ)

külső konzulenseként nyilatkozom arról, hogy a hallgató az előre egyeztetett konzultációkon rendszeresen megjelent.

Kelt: Budapest, 2025. október 30.

külső konzulens

13 KÖSZÖNETNYILVÁNÍTÁS

Szeretném megköszönni témavezetőmnek Prof. Dr. Keppler Istvánnak, hogy lehetőséget biztosított egy ennyire érdekes és műszaki kihívásokkal teli munka elvégzésére, továbbá, hogy iránymutatásaival, minden esetben tovább segített az akadályokon, ezen kívül szeretném megköszönni a pozitív visszajelzéseit, amelyekkel folyamatosan lendületben tartott a dolgozat elkészítése alatt.

Nagy köszönettel tartozom a Yade felhasználói és fejlesztői közösségének, valamint azon kutatóknak, akik nagy tapasztalattal rendelkeznek a dolgozatomban vizsgált speciális területen és készségesen segítettek, minden felmerülő kérdésem és problémám megoldásában.

Végül pedig, de nem utolsó sorban rendkívül hálás vagyok a családomban, amiért mindvégig támogattak a tanulmányaim során és a legnehezebb időszakokban is türelmesek és megértőek voltak velem.