Diszkrételemes talajszimulációk gyorsítása grafikus processzorok használatával

Szerző: Nagy Dániel

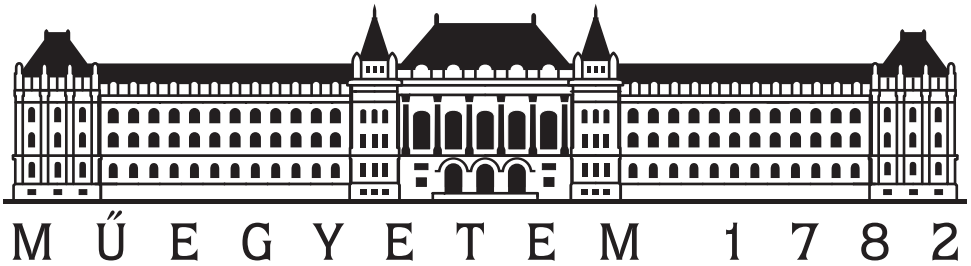
Konzulens:

Dr. Tamás Kornél (BME–GT3)

Pásthy László (BME–GT3)

2023. november 9.

2023. évi Intézményi TDK Konferencia



Tartalomjegyzék

Absztrakt

Diszkételemes módszereket (DEM) elsősorban szemcsés anyagok szimulációjára alkalmaznak. A módszer lényege, hogy a kisméretű diszkrét szemcsék (elemek) közti kölcsönhatások leírásával szimulálható egy granulált anyag makroszkopikus viselkedése. Diszkrételemes szimulációkat a műszaki tudományok számtalan területén alkalmaznak, kezdve a mezőgazdaságtól, ahol a talaj kisméretű szemcsékkel modellezhető, egészen az építészetig, ahol a téglaépületek mechanikája vizsgálható a téglák (elemek) közötti kölcsönhatások leírásával. A módszer rendkívül számítás- és memória igényes, hiszen egy tipikus DEM szimuláció több tízezer elemet tartalmaz, ráadásul minden elem több másik elemmel is érintkezhet, így több százezer kölcsönhatás számítására kerül sor egyetlen időlépésben. A numerikus stabilitás ráadásul kicsi időlépést igényel a legtöbb esetben.

Grafikus processzorok (GPU-k) hatékonysága abban rejlik, hogy egyetlen gépi ciklus alatt képesek ugyanazt a műveletet több tízezer egymástól független adaton elvégezni. GPU-k alkalmazása már elterjedt közönséges differenciálegyenlet-rendszerek paramétertanulmányainak megoldására (lásd a CUDA C++-ra épülő MPGOS könyvtárat és Julia-ra épülő DifferentialEquations.jl könyvtárat). Az elmúlt években megjelent új GPU architektúrák azonban már lehetőséget biztosítanak nagy sebességű adatátvitelre és a párhuzamosan futó szálak közötti hatékony kommunikációra, ezáltal az utóbbi években már komplexebb feladatok szimulációjára is alkalmasak. Ezen felül az újonnan épült szuperszámítógépek egyre nagyobb arányban tartalmaznak GPU alapú számítási kapacitást, például az újonnan átadott magyarországi Kommondor szuperszámítógép 80%-a GPU-s számítási kapacitás.

Diszkrételemes szimulációk gyorsítására szintén használhatók GPU-k. Az elkészült GPUDEM saját fejlesztésű CUDA C++ alapú moduláris könyvtár lényege, hogy minden GPU-n futó szálhoz egy szemcsét rendel. A könyvtár képes tetszőleges számú és méretű gömb alakú elem szimulációjára. A szimuláció tetszőleges számú különböző anyagot tartalmazhat, illetve bármilyen háromszögekkel határolt geometria hozzáadható és mozgatható. Az érintkezések számításához jelenleg a Hertz-Mindlin kapcsolati modell került implementálásra, azonban a moduláris felépítésnek köszönhetően könnyen hozzáadhatók más modellek. A könyvtár Paraview kompatibilis kimeneti fájlokat generál, így az eredmények vizualizációja gyors és egyszerű.

A dolgozatban bemutatásra kerül a könyvtár felépítése és validációja, az alkalmazott numerikus módszerek és a program alkalmazása szemcsék ülepítésére és kultivátoros talajlazításra. Saját fejlesztésű könyvtár esetén a validáció elengedhetetlen, ehhez egy, az EDEM szoftverben készült ülepítés került reprodukálásra azonos kezdeti feltételek és paraméterek mellett. A GPU-s gyorsításnak köszönhetően a GPUDEM 13-szor gyorsabbnak bizonyul az EDEM-nél. A dolgozat végső célja a GPUDEM gyakorlati alkalmazhatóságának a demonstrálása, egy nedves homok talajban végzett kultivátoros mérés eredményeinek a reprodukálása által, amely differenciál-evolúció alapú paraméteroptimalizációval valósul meg.

Abstract

Discrete element methods (DEM) are mainly used for simulating granular materials. The essence of the method is to simulate the macroscopic behaviour of a granular material by describing the interactions between small discrete particles (elements). Discrete-element simulations are used in a wide range of engineering disciplines, from agriculture, where soil can be modelled with small particles, to architecture, where the mechanics of brick buildings can be studied by describing the interactions between bricks (elements). The method is extremely computational and memory intensive, as a typical DEM simulation contains tens of thousands of elements, and each element may interact with several other elements, so hundreds of thousands of interactions are calculated in every single time step. Moreover, numerical stability requires small time steps in most cases.

The efficiency of Graphics Processing Units (GPUs) lies in their ability to perform the same operation on tens of thousands of independent pieces of data in a single machine cycle. GPUs are already widely used to solve parameter studies of ordinary differential equation systems (see the MPGOS library based on CUDA C++ and the DifferentialEquations.jl library based on Julia). However, new GPU architectures that have emerged in recent years now offer the potential for high-speed data transfer and efficient communication between threads running in parallel, making them suitable for simulating more complex tasks. In addition, an increasing proportion of supercomputers built in recent years include GPU-based computing capacity, for example 80% of the newly delivered Kommondor supercomputer in Hungary has GPU computing capacity.

GPUs can also be used to accelerate discrete element simulations. The self-developed GPUDEM is a CUDA C++-based modular library designed to assign each particle to a thread running on a GPU. The library can simulate any number and size of spherical elements. The simulation can contain any number of different materials, any geometry bounded by triangles can be added and moved, and the Hertz-Mindlin contact model is currently implemented for calculating the contacts, but other models can be easily added due to the modular architecture. The library generates Paraview compatible output files, making the visualisation of results fast and easy.

The paper describes the design and validation of the library, the numerical methods used and the application of the program to particle sedimentation and soil-sweep interaction. In the case of a self-developed library, validation is essential, for which a sedimentation in the EDEM software has been reproduced under identical initial conditions and parameters. Thanks to GPU acceleration, GPUDEM proves to be four times faster than EDEM. In line with the needs of high-performance computing, the work also includes code profiling - i.e. testing the efficiency and computational complexity of the program - and scalability testing. The final goal of the thesis is to demonstrate the practical applicability of GPUDEM by reproducing the results of a cultivator measurement in sand soil, which is realized with differential-evolution-based parameter optimization.

Jelölések

A dolgozatban használt jelölések

Jelölés	Dimenzió	Megnevezés
$\boldsymbol{a}$	$\mathrm{m/s^2}$	gyorsulás vektor
d	m	szemcse-szemcse vagy szemcse-fal távolság
e	1	ütközési együttható
E	Pa	Young-modulus
$\boldsymbol{F}$	N	erő
G	Pa	nyírási-modulus
m	kg	szemcsetömeg
$\boldsymbol{M}$	Nm	forgatónyomaték vektor
N_P	1	szemcsék száma
N_M	1	anyagok száma
$\boldsymbol{n}$	1	STL háromszög normálvektor
$\boldsymbol{p}$	m	STL háromszög 1. csúcsának helyvektora
$\boldsymbol{p}_{s\to i}$	m	s szemcse középpontból s és i közötti érintkezési pontba mutató vektor
$\boldsymbol{r}$	m	egyik szemcséből a másikba mutató vektor
R	m	szemcse sugár
$\boldsymbol{s}$	m	STL háromszög 2. csúcsának helyvektora 1-ből
$\boldsymbol{t}$	m	STL háromszög 3. csúcsának helyvektora 1-ből
$\boldsymbol{u}$	m	pozíció vektor
$\boldsymbol{v}$	m/s	sebesség vektor
β	1	csillapítási tényező
$\boldsymbol{\beta}$	$\mathrm{rad/s^2}$	szöggyorsulás vektor
δ_n	m	normálirányú átfedés
$\boldsymbol{\delta}_t$	m	tangenciális-irányú átfedés
Δ_t	s	időlépés
θ	$\mathrm{kg \cdot m^2}$	tehetetlenségi nyomaték
μ	1	csúszási súrlódási együttható
μ_0	1	tapadási súrlódási együttható
μ_r	1	gördülési súrlódási együttható
ν	1	Poisson-szám
ρ	$\mathrm{kg/m^3}$	sűrűség
σ	1	koordináta az STL háromszög vektorainak rendszerében
τ	1	koordináta az STL háromszög vektorainak rendszerében
$\boldsymbol{\omega}$	rad/s	szögsebesség vektor
$\square^*$		ekvivalens mennyiség
$\square^{\{n\}}$		n. időlépésbeni mennyiség

1. Bevezetés

A talaj megfelelő gondozása és művelése kiemelt fontosságú az emberiség számára, mivel a talaj minőségének megőrzése és a fenntartható gazdálkodás lehetővé teszi az élelmiszertermelés növelését és a természeti erőforrások megóvását. Bár a talajművelés több ezer éves tudomány, azonban mind a mai napig tart a technológia fejlődése és jelennek meg újabb talajművelő szerszámok. Napjainkban a környezetbarát talajművelés egyre inkább előtérbe kerül talajművelő szerszámok tervezése esetén, ennek lényege az energiatakarékosság, a talajerózió szabályozása és a károsanyag-kibocsátás csökkentése [1]. A megfelelő talajművelő szerszám megtervezése, sőt még kiválasztása is nehéz feladat, hiszen ez a talajtípustól függően eltérő lehet.

A talaj-szerszám kölcsönhatások matematikai modellezése lehetővé teszi a korábban ismertett problémák elméleti megoldását, például lehetőség nyílik egy kultivátor numerikus optimalizációjára, az energiahatékonyság növelése érdekében. A modellezésre használhatók analitikus modellek, azonban ezek mindössze az eke adott sebességgel való mozgatásához szükséges húzerőt adják meg [2]. Az analitikus modell előnye, hogy validáció után az optimális szántási mélység és sebesség meghatározható [3], azonban a geometria optimalizációja ezzel a módszerrel nem lehetséges. Másik lehetőség végeselem módszer (VEM) alkalmazása a talaj szimulációjára. A VEM előnye, hogy lehetőség nyílik a szerszám-geometria optimalizációjára és a talaj mozgásának a szimulációjára [4], a módszer hátránya azonban, hogy folytonos közeget feltételez, amely nem mindig teljesül. Végeselem módszer használatával nem modellezhető a talaj deformációja, a repedések megjelenése a talajban és a talajszemcsék áramlása sem [5].

Az utóbbi másfél évtizedben elterjedt a diszkrételemes módszer (DEM) alkalmazása talajszimulációkra. A DEM előnye, hogy segítségével figyelembe vehető a talaj inhomogenitása és lehetőség nyílik a talajszemcsék áramlásának a meghatározására is [6]. Diszkrételemes módszerrel számos mezőgazdaságban jelentkező probléma szimulálható és megoldható, többek között lehetséges a vonóerő meghatározásán túl egy kultivátoros művelés során kialakuló repedések vizsgálata is [7], sőt az eljárás továbbfejleszthető és a talajban jelen lévő származadványok is figyelembe vehetők a DEM-el [8].

1.1. Diszkrételemes Módszerek

A diszkrételemes módszert az 1970-es évek elején vezette le először Peter Alan Cundall a PhD dolgozatában, hogy sziklákból álló struktúrák összeomlását és dőlését vizsgálja [9]. A DEM-et azóta a műszaki tudományok számos területén használják kezdve az építészettől – ahol például téglaépületek földrengésbiztonságának a meghatározására használható [10]– egészen a korábban már említett talajművelésig.

A DEM az anyagot szemcsék sokaságaként modellezi, ezek a szemcsék az elemek, amelyek lehetnek például a hasáb alakú téglák az építészetben vagy gömbökből összetett talajszemcsék. Manapság azonban már tetszőleges háromszögek által határolt poliéder elemek is használhatók, viszont ezek a modellek lényegesen bonyolultabbak és több számítást igényelnek [11]. Az elemek mind elmozdulhatnak és elforoghatnak, így 3D-ben egy elem 6 szabadsági fokkal rendelkezik. A diszkrét elemes eljárások lényege [12] alapján, hogy

- a szemcsék közötti kapcsolatok határozzák meg azok mozgását,
- a szemcsék egy szilárd anyagot reprezentálnak és
- a szemcsék mozgása miatt kapcsolatok megszűnhetnek és újak jöhetnek létre.

Egy DEM talajszimuláció a következő részekből áll, a szemcsefelhő (azaz az kezdeti elemek) létrehozása és a geometria meghatározása után:

1. Kapcsolatkeresés: A szemcse-szemcse és a szemcse-geometria kapcsolatok meghatározása. Számos algoritmus létezik, a legegyszerűbb a „brute force”, azaz nyers erő használata és minden lehetséges szemcsepár érintkezésének a vizsgálata. Szofisztikáltabb módszer a számítási tartomány alá egy hálót helyezni, és csak a szomszédos cellákban lévő szemcsék közötti kapcsolatot vizsgálni [13].

2. Kapcsolati modell: A szemcsékre ható erők meghatározása egy adott érintkezési modell szerint. Az érintkezési modell lehet például a Hertz modell, amely a szemcsék összenyomódásából adódó elasztikus erőt veszi csak figyelembe vagy a Hertz-Mindlin modell amely ezt kiegészíti a tangenciális merevséggel és ma az egyik legelterjedtebb modell [14]. A Hertz-Mindlin modell alapja a szemcseütközés során az átfedés hatására fellépő erők meghatározása, ebből adódóan nem minden esetben teljesül az energiamegmaradás, energia alapú modellekkel azonban ez a probléma is megoldható [15]. Léteznek továbbá modellek amelyek a szemcsék közti adhéziót vagy a szemcsék alakváltozását is figyelembe veszik.

3. Időlépés: A szemcsére ható erőkből gyorsulások számítása és a szimuláció időbeni előreléptetése a megfelelő numerikus módszerekkel. A DEM-ben explicit időléptetést használnak néhány kivételtől eltekintve (például amikor viszkoelasztikus erők számítása szükséges [16]). A legegyszerűbb az explicit egylépéses Euler-módszer alkalmazása, amelyben a pozíciószámításnál a Taylor-sorfejtésből adódó magasabbrendű tagokat is figyelembe lehet venni [17]. Egylépéses Runga-Kutta módszerek alkalmazása nem jellemző, mivel azok esetén köztes értékek kiszámítása szükséges, amely sok szemcse esetén rendkívül időigényes. Többlépéses módszerek több korábbi időlépés alapján határozzák meg a következő lépést, így többletszámítás nélkül használhatók magasabb rendű módszerek, mint például Adams-Bashforth – Adams-Moulton korrektor-prediktor formulák [16].

1.2. A számítástechnika fejlődése

A DEM szimulációk az 1970-es években még csak kevesebb mint 100 elemet tartalmaztak [9]. A számítástechnika fejlődésével lehetőség nyílt nagyobb szimulációkra, manapság néhány esetben már több millió szemcsét is alkalmaznak [18, 19]. Nagyméretű numerikus szimulációk esetén a feladat párhuzamosítása elengedhetetlen a futási idő csökkentése érdekében. A nagy teljesítményű számítástechnikában a párhuzamosítás hagyományos módja több processzor és processzormag használata.

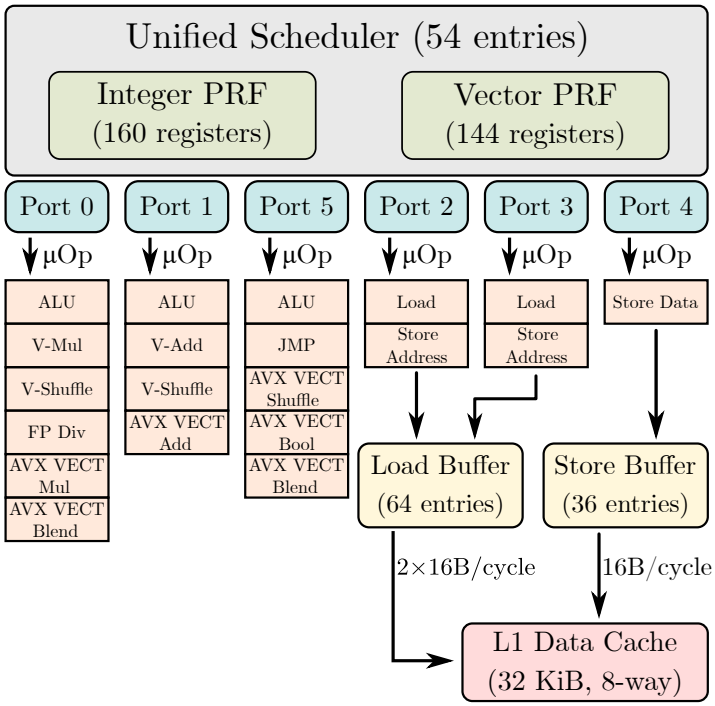
A hagyományos processzorok (CPU-k) képesek rendkívül komplex utasítások elvégzésére, azonban egyetlen gépi ciklus alatt mindössze egy művelet elvégzésére kerül sor, egy adaton. Ez az *egy utasítás egy adat* (SISD – single instruction single data) architektúra az, ahogyan a CPU-k az elmúlt évszazadban működtek. Manapság a CPU által végzett számítások már több módon is párhuzamosíthatók:

1. Egy CPU több maggal is rendelkezik, amelyek egymástól teljesen függetlenül működnek, az Intel 13. generációs i7 processzorai már 16 maggal rendelkeznek, ez a *több utasítás több adat* (MIMD – multiple instruction multiple data) architektúra.

2. Egy CPU mag több portot tartalmaz és minden port képes egymástól függetlenül mikroműveletek végrehajtására, ez látható az 1.1 ábrán. Megfigyelhető például, hogy a `V-Mul` (vektor szorzás), `V-Add` (vektor összeadás), `Store`, `Load` műveletek párhuzamosan végrehajthatók (lásd 0-3 portok).

3. Modern CPU-kon akár 16 elemből álló vektoron is végrehajthatók műveletek egyszerre, a szaknyelv ezt vektorizációnak nevezi, ezen vektorműveleteket az AVX utasításkészlet definiálja Intel processzorok esetén.

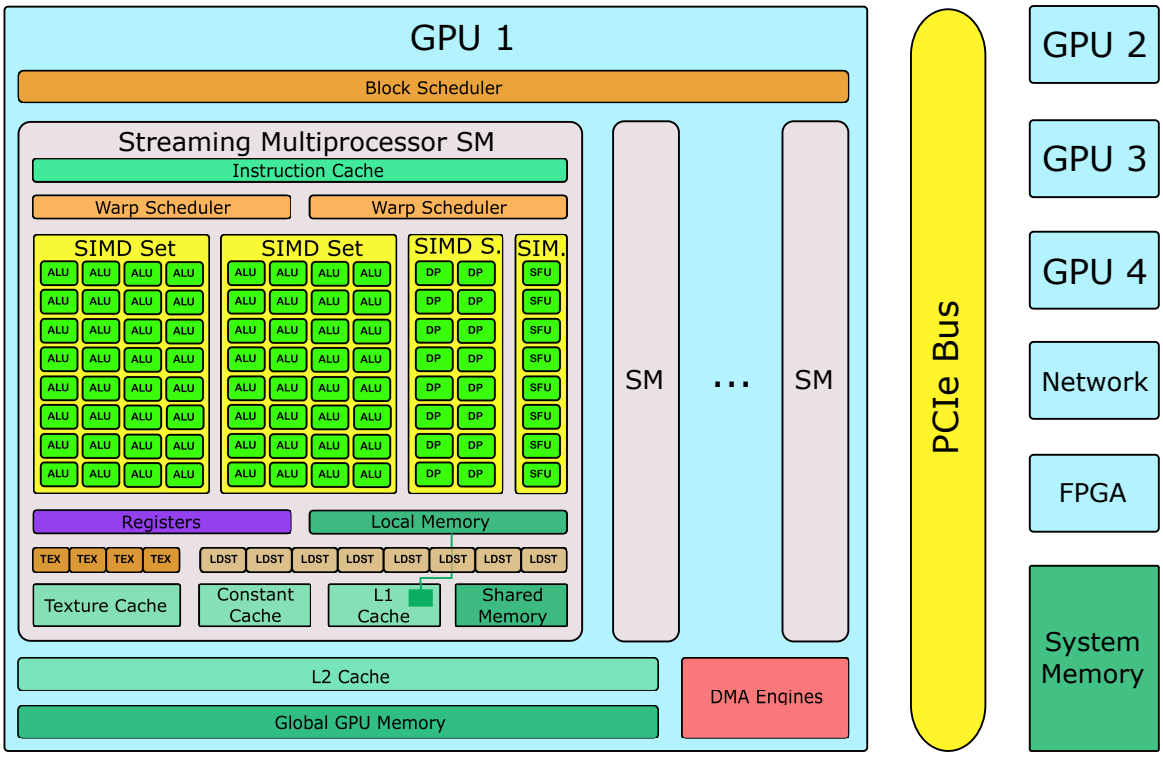
A CPU alapú nagy teljesítményű számítástechnika jellemezően az 1. pontban leírt processzormag szintű párhuzamosságra fókuszál. Bár léteznek eszközök a portok párhuzamos kihasználására, azonban ezek alkalmazása numerikus szimulációk során specifikus programkódot igényel. A legújabb fordítók képesek bizonyos esetekben a program automatikus vektorizációjára és az utasítások átrendezésére a párhuzamosság növelése érdekében, de ezek közel sem elegendők a számítási kapacitások maximális kihasználására [20].



1.1. ábra. Az Intel IvyBridge CPU architektúrája. A feladatütemező (Unified Scheduler) választja ki a végrehajtásra kerülő mikroműveleteket (µOp). A mikroműveletek egy-egy porton végrehajtott utasításokat jelentenek, ezek lehetnek általános logikai műveletek (ALU), összeadás (V-Add), szorzás (V-Mul), memória betöltés (Load), tárolás (Store)... . A betöltés és tárolás műveletek egy átmeneti pufferbe (Store/Load Buffer) tárolódnak, amíg nem kerülnek sorra és férnek hozzá a gyorsítótárhoz (Data Cache).

1.3. Grafikus processzorok

A grafikus processzorok (GPU-k) az *egy utasítás több adat* (SIMD – single instruction multiple data) elvére épülnek. Azaz, egy utasítást, például szorzást vagy összeadást egyszerre több különböző adaton képesek elvégezni, lehetővé téve nagy mennyiségű ugyanolyan művelet elvégzését egyszerre. A SIMD architektúrát kihasználva a 2010-es évek elején jelent meg az első tudományos számításra specializálódott grafikus processzor, az NVIDIA Tegra és ezzel váltak a grafikus processzorok által gyorsított numerikus szimulációk elterjedté. Jelenleg a számítástechnika fejlődése a specializált eszközök, mint például grafikus és tenzor processzorok (GPU-k és TPU-k) használata irányába halad, például a 2023-ban átadott magyarországi Kommondor szuperszámítógép teljesítményének 80%-át GPU-k adják.



1.2. ábra. Egy GPU általános felépítése [21]. A GPU PCIe buszon keresztül csatlakozik a CPU-hoz és éri el a rendszer memóriát (System). Egy GPU több multiprocesszort tartalmaz (Streaming Multiprocesszor) amelyek pedig tartalmázzák az egy utasítás több adat elvnek megfelelően működő számítási egység tömböket (SIMD Set), ezek álhatnak általános logikai egységekből (ALU), lebegőpontos műveleti egységeből (DP). A GPU rendelkezik regiszter (Registers), lokális (Local), gyorsítótár (Cache), osztott (Shared) és globális (Global GPU) memóriával.

1.3.1. Felépítése

A GPU alapvető számítási egysége a szál, egyszerre a GPU-n több százmillió szál is jelen lehet, a CPU szálakkal ellentétben a GPU szálak jellemzően lényegesen kevesebb adaton dolgoznak, szakmai szóval lényegesen *könnyebbek*. Ez jól szemléltethető két vektor összeadása során ($C = A + B$). Egy CPU-n egy szál végigmegy minden elemen ahogyan az 1. kódrészlet mutatja, ez egyszerűen egy ciklussal kerül leírásra. A GPU-n a ciklus elhagyásra kerül, ahogy ez a 2. kódrészletben látható. A GPU-n minden egyes szál egy saját azonosítóval (index) rendelkezik és ez alapján minden szál egy másik adaton végzi el az összeadást.

1. Kódrészlet. Két vektor A és B összeadása és az eredmények eltárolása CPU-n.

```
for(int i = 0; i < N; i++)
{
  C[i] = A[i] + B[i];
}
```

2. Kódrészlet. Két vektor A és B összeadása és az eredmények eltárolása GPU-n.

```
int i = calculateThreadIndex();
C[i] = A[i] + B[i];
```

Minden GPU tartalmaz több *Streaming Multiprocesszort* (SM), amelyek egymástól függetlenül működnek, ez a hierarchia legmagasabb szintje, ezek a multiprocesszorok tartalmazzák a

számítási egységeket. A GPU memória többszintű, a globális memória és az L2 Cache a teljes GPU-hoz tartozik, ezentúl minden multiprocesszor saját regiszterekkel, lokális memóriával, L1 Cache-el és saját felhasználó által programozható osztott memóriával rendelkezik [21]. A GPU-k általános felépítését az 1.2. ábra mutatja. A globális memória a legmagasabb szintű memória, ami jellemezően több gigabájt méretű és a GPU bármely részéről elérhető, azonban a sebessége rendkívül lassú. A gyakran használt adatok automatikusan bekerülnek az L1 Cache-be ami már lényegesen gyorsabb elérést biztosít. A regiszter memória mint az 1.2. ábrán is látható közvetlenül a számítási egységek (`ALU`, `DP` és `SFU`) mellett helyezkedik el, így ennek az elérése rendkívül gyors. A regiszter memória közvetlenül a felhasználó által programozható és minden szálhoz saját regiszterek rendelhetők.

A szálakat először a multiprocesszorokhoz kell rendelni, amely blokkonként valósul meg, egy blokk több szálat tartalmaz, és a szálak száma egy blokkban a felhasználó által állítható. A blokkok tovább osztódnak úgynevezett „warp"-okra, egy warp mérete a jelenlegi NVIDIA GPU architektúrákon 32 szál. Minden multiprocesszor rendelkezik warp-ütemezővel (Warp Scheduler), amely akkor indít el egy warpot, ha van rendelkezésre álló warp és elegendő szabad számítási egység. A warpok a már korábban említett SIMD struktúra szerint működnek, azaz a warpon belül minden szál ugyanazt a műveletet végzi egy adott időpontban, azonban más-más adaton, ezért érhető el a GPU-val hatalmas számítási kapacitás a CPU-hoz képest [21]. Ezen megközelítés hátránya az elágazások kezelése, hiszen ekkor elképzelhető, hogy minden szálnak más-más műveletet kellene végrehajtani, ezt nevezzük szál-divergenciának.

1.3.2. Fejlesztés grafikus processzorra

A legelterjedtebbb fejlesztői környezet a CUDA, ami egy NVIDIA általt fejlesztett platform és párhuzamos programozási modell, amely kizárólag NVIDIA GPU-kon használható [22]. A másik elterjedt platform az OpenCL [23], amely bármilyen osztott memóriával rendelkező párhuzamos környezetben működik (például GPU-k, CPU-klaszterek). A CUDA fő előnye a nagyobb sebesség és bővebb eszköztár, amely elsősorban annak köszönhető, hogy a programozási modellt és a grafikus processzort is az NVIDIA adja ki, amely magas szintű integrációt tesz lehetővé. A nyilvánvaló hátrány ebből következően a platformfüggőség. A dolgozatban a CUDA programozási modell kerül alkalmazásra, egyrészt a nagyobb elérhető sebesség miatt, másrészt pedig a mai GPU alapú szuperszámítógépek szinte kivétel nélkül NVIDIA GPU-kat tartalmaznak, így CUDA kompatibilisek. A CUDA több programnyelven is implementálva van, azonban a legtöbb lehetőséget és a legnagyobb sebességet a C++ biztosítja, így ez kerül alkalmazásra.

Numerikus szimulációk több módon is implementálhatók GPU-kra. Az implementációk a következő módon csoportosíthatók:

1. *perGPU*: egy számítási feladatot a teljes GPU-hoz rendelünk. Ekkor a GPU-n lévő összes szál együtt dolgozik és old meg egy problémát, például mátrix-vektor műveletek végezhetők el ilyen módon egy VEM szimulációban [24].

2. *perBlock*: egy számítási feladatot egy GPU blokkhoz rendelünk. Ekkor a GPU-n lévő blokkokban lévő szálak közösen dolgoznak a feladaton, de a blokkok egymástól függetlenül számolnak. Előfordulhat, hogy a blokkok közötti kommunikáció szükséges, ez szinkronizációs pontok beszúrásával valósítható meg. A módszer előnye, hogy egyszerre több 100 különböző számítási feladat végezhető. A *perBlock* megközelítést a néhány éve megjelent *cooperative groups* CUDA könyvtár teszi lehetővé [25], azonban ennek használata még nem terjedt el a numerikus szimulációk területén.

3. *perThread*: egy számítási feladatot egy szálhoz rendelünk. Ekkor minden szál egyedül dolgozik, de egyszerre több ezer szál több ezer különböző adaton képes számolni. A szálak közötti kommunikáció szinkronizációs pontok beszúrásával megvalósítható, így csatolások is figyelembe vehetők. A módszer különösen jól használható például differenciálegyenletek paramétertanulmányainak a megoldására [20].

1.3.3. DEM szoftverek GPU-ra

GPU alapú DEM szimulációk már évek óta léteznek, azonban ezen programok nem elterjedtek a felhasználók körében. A legtöbb esetben a GPU-s DEM szimulációk egy-egy kutatócsoport saját készítésű programjai egy-egy specifikus problémára, amely nagyszámú szemcsét igényel [26, 27]. A masszívan párhuzamos GPU architektúra azonban a megszokott DEM implementációtól lényegesen eltérő logikát és felépítést igényel, amennyiben a GPU erőforrásait a lehető legjobban szeretnénk kihasználni. Az eltérő GPU-s implementációt azonban a szakirodalom jelenleg nem tárgyalja részletesen, a fókusz az alkalmazásokon van.

Léteznek teljesen általános problémákra használható GPU alapú DEM megoldók, mint például a Chrono::GPU, amellyel több tízmillió szemcse is szimulálható [28], a programcsomag hátránya jelenleg, hogy csak egyféle anyag használható és minden szemcse azonos méretű. Az Altair EDEM® is rendelkezik opcióval ahhoz, hogy a számítások GPU-n fussanak, azonban a tapasztalatok szerint a GPU-s megoldás nem gyorsabb.

1.4. Célkitűzés

A fő cél egy saját fejlesztésű moduláris GPU-s DEM megoldó készítése CUDA C++-ban, amely megbízhatóan használható sok szemcsét tartalmazó talajszimulációk készítésére és a GPU-s gyorsításnak köszönhetően legalább egy nagyságrenddel gyorsabb mint a meglévő szoftverek. Feltétel, hogy a programban tetszőleges számú és méretű szemcse, anyag és peremfeltétel megadható legyen, illetve legalább egy mozgatható geometria is hozzáadható legyen, amely a mezőgazdasági alkalmazás miatt fontos. A modularitás lényege, hogy különféle kapcsolatkeresési algoritmusok, kapcsolati modellek és időlépés sémák alkalmazhatók legyenek és a program kiegészíthető legyen a jövőben. Először a Hertz-Mindlin kapcsolati modell kerül implementálásra első és másodrendű időlépés sémákkal. A program fejlesztése során a *perThread* megközelítés kerül alkalmazásra, ahol minden GPU szál egyetlen szemcsét számol. A program hatákonyságának a növelése érdekében a következő irányelvek betartásra kerülnek:

- Globális memória használat optimalizálása: mint az 1.3.1. fejezetben említésre került, a globális memória rendkívül lassú. Amennyiben a globális memóriából van szükség adatra, akkor viszont érdemes minél több adatot egyszerre betölteni egymás melletti memóriacímekről, hiszen a teljes sávszélesség csak ekkor használható ki.

- Explicit regiszterhasználat: a gyakran használt változók és paraméterek a regisztermemóriába explicit betöltésre kerülnek, ez elősegíti a globális memória használatának a csökkentését is.

- Optimalizáció fordítási időben: a program fordítása (angol szóval *compilation*) azaz gépi kód generálása a C++ kódból. A C++ 2011-ben megjelent szabványa óta lehetőség van fordítási időben előre elvégezni számításokat [29], illetve ez alapján állítható, hogy egy adott programrész belekerüljön-e a végső gépi kódba, ami lehetővé teszi a moduláris felépítést teljesítményveszteség nélkül.

- Osztás minimalizálás: az osztás legalább egy nagyságrenddel lassabb mint a szorzás [30], ezért előre kiszámításra kerülnek a többször elvégzendő osztás műveletek.
- Speciális műveletek minimalizálása: a hatványozás, gyökvonás, exponenciális és trigonometrikus függvények kiértékelése rendkívül lassú, így ezek kerülendők az osztáshoz hasonlóan.

A hatékony DEM program készítésén túl a dolgozatnak számos további célja is van:

1. Az 1.3.3. fejezetben említésre került, hogy a GPU-s DEM alkalmazások során az eljárás GPU specifikus részei nem kerülnek jellemzően bemutatásra. A fentebb felsorolt optimalizációs irányelvek a DEM számos lépésében használhatók, amelyek szintén bemutatásra kerülnek.
2. A lényegesen alacsonyabb futási időből következően a dolgozat további célja
   (a) a DEM-es talajművelés szimulációk paraméter érzékenységi vizsgálata és
   (b) az anyagparaméterek optimalizációja a mérések reprodukálása érdekében.

   A paraméter érzékenységi vizsgálatból fontos következtetések nyerhetők arról, hogy az anyag és kapcsolati paraméterek hogyan befolyásolják a vonóerőt egy kultivátoros talajművelés vagy szántás során. Amennyiben egy szimuláció elég hamar lefut és az eredményekre nem kell órákat, esetleg napokat várni, akkor lehetséges a paraméteroptimalizáció evolúciós algoritmussal. A végső cél a dolgozatban egy paraméterkalibráció a mérési eredmények reprodukálása érdekében, evolúciós algoritmus használatával.
3. A DEM használata során számos további kérdés felmerül, például, hogy milyen időlépési sémát válasszuk. A szakirodalom jellemzően az elsőrendű explicit Euler módszert alkalmazza [16], azonban ez a választás jellemzően nem kerül indoklásra. Továbbá, több algoritmus is létezik a kapcsolatkeresésre, de nem létezik egyetlen legjobb algoritmus és a jó választás általában feladatfüggő. A dolgozat célja különböző időlépés sémák és kapcsolatkeresési eljárások bemutatása és ajánlások megfogalmazása a különböző esetekre.

2. A program felépítése

A teljes GPUDEM program CPU, GPU és fordítási időben kiszámolt részt is tartalmaz. A programcsomag fordításának idejében már ismertnek kell lennie a szemcsék és anyagok számának, a választott kapcsolati modellnek, kapcsolatkeresési algoritmusnak és időléptetési módszernek. Ezek lehetővé teszik a programkód fordító általi optimalizálását. A CPU fő feladata a kezdeti feltételek beolvasása, anyagparaméterek beállítása, a memória kezelése és a kimeneti fájlok elkészítése. A GPU a számítás-intenzív részt végzi el, mint a kapcsolatkeresést, erőszámítást és időléptetést. A teljes felépítés a 2.1. ábrán látható. A következő fejezetek a DEM algoritmus GPU specifikus megvalósítását mutatják be, mint a szemcsék tárolását a memóriában, a szemcse-szemcse érintkezés és a szemcse-fal érintkezést, az erőszámítást, az időbeni megoldást és a kimeneti fájlok elkészítését. A számítástechnikai, fizikai és matematikai részletek együttesen kerülnek bemutatásra, hiszen mint látni fogjuk ezek számos esetben nem különíthetők el egymástól.

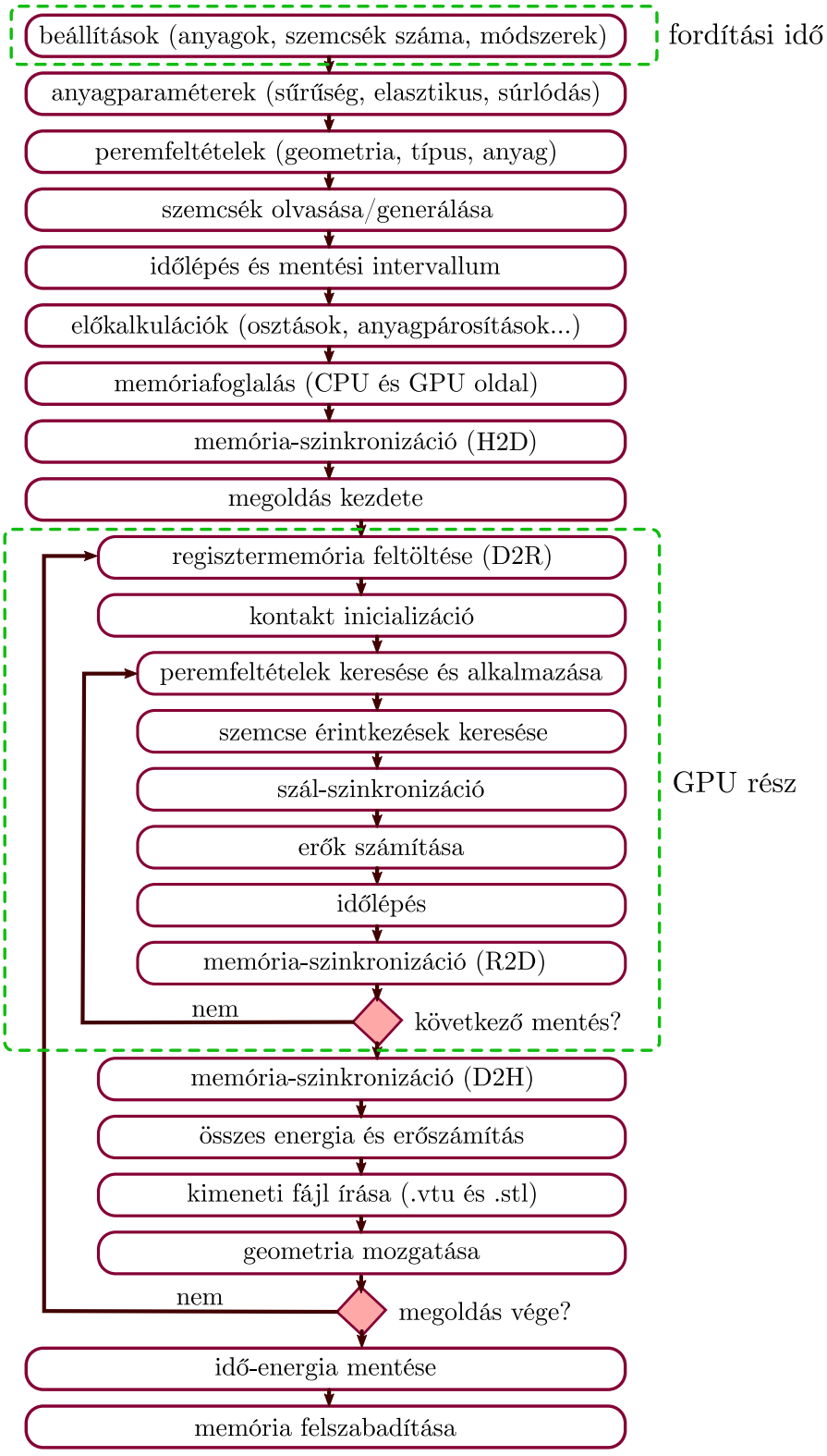
A program a már korábban említett *perThread* megközelítést használja, azaz minden szemcséhez egy szál rendelődik. A 2.1. ábrán a kerettel jelölt GPU rész elején annyi szál indul párhuzamosan, amennyi szemcse található a szimulációban. A szemcseszám, azonban lényegesen nagyobb, mint a GPU-ban lévő CUDA magok száma, így nem kerül minden egyes szál egyszerre számítási erőforrásokhoz. Ez olyan helyzeteket teremthet, hogy a szálak nem ugyanott járnak a számításban, amely nem fizikai eredményeket okozna. Ennek az elkerülése érdekében a programban szinkronizációs pontok találhatók minden időlépés során, hogy a szálak megvárják egymást.

2.1. táblázat. A szemcsék tulajdonságai, amelyeket a GPUDEM tárol

Jelölés	Név
$\boldsymbol{u}$	Pozíció
$\boldsymbol{v}$	Sebesség
$\boldsymbol{a}$	Gyorsulás
$\boldsymbol{\omega}$	Szögsebesség
$\boldsymbol{\beta}$	Szöggyorsulás
$\boldsymbol{F}$	Erő
$\boldsymbol{M}$	Nyomaték
R	Sugár
m	Tömeg
θ	Tehetetlenség
$\mathrm{m_{id}}$	Anyag
R^{-1}	Sugár^{-1}
m^{-1}	Tömeg^{-1}
θ^{-1}	Tehetetlenség^{-1}
$\mathrm{c_{id}}$	Cella index

2.2. táblázat. Az anyagok tulajdonságai, amelyeket a GPUDEM tárol

Jelölés	Név
ρ	Sűrűség
E	Young-modulus
G	Nyírási-modulus
ν	Poisson tényező
e	Ütközési tényező
μ	Csúszási súrlódás egyh.
μ_0	Tapadási súrlódás egyh.
μ_r	Gördülési súrlódás egyh.
β	Csillapítási tényező



2.1. ábra. A GPUDEM felépítése. Minden blokk a program egy elemét jelöli, a ciklusok visszafelé mutató nyíllal vannak ábrázolva. A felső zöld keret már fordítási időben számításra kerül, a középső zöld keret pedig a program GPU-n futó részét mutatja. Rövidítések magyarázata: H2D - Host to Device (másolás CPU-ról GPU-ra); D2H - Device to Host (másolás GPU-ról CPU-ra); R2D - Register to Device (másolás GPU regiszterekből GPU globális memóriába); D2R - Device to Register (másolás GPU globális memóriából GPU regiszterekbe).

2.1. Szemcsék

A következőkben jelölje N_P a szemcsék számát és N_M az anyagok számát. Minden szemcse rendelkezik a 2.1. táblázatban felsorolt tulajdonságokkal. A szemcsék egyik tulajdonsága az anyaguk, amelyek külön struktúrában vannak tárolva, a szemcsék csak a megfelelő anyagsorszámot tárolják. Egy szimulációban több különböző anyag is megadható és minden szemcséhez hozzárendelésre kerül a korábban említett anyagsorszám. Az anyagok tulajdonságai a 2.2. táblázatban láthatók. Minden szemcsének a tömege és tehetetlenségi nyomatéka meghatározásra kerül a megadott sűrűség alapján, mivel a szemcsék gömb alakúak ezért

$$m = \frac{4}{3} \cdot R^3 \cdot \pi \cdot \rho \text{ és} \tag{2.1}$$

$$\theta = \frac{2}{5} \cdot m \cdot R^2. \tag{2.2}$$

A szemcsék sugarának, tömegének és tehetetlenségének a reciproka is tárolásra kerül, mivel ezeket minden időlépésben használni kell és mint az 1.4. fejezetben bemutatásra került az osztás költséges művelet. Minden lehetséges anyag-anyag érintkezéshez előre kiszámításra kerülnek az ekvivalens anyagparaméterek, az E^* ekvivalens Young-modulus és a G^* ekvivalens nyírásimodulus, amely a Hertz-Mindlin modell szerint [31],

$$\frac{1}{E^*} = \frac{1-\nu_i^2}{E_i} + \frac{1-\nu_j^2}{E_j} \quad \text{és} \tag{2.3}$$

$$G^* = \frac{E^*}{2+\nu_i+\nu_j} \quad \text{ahol} \quad i = 0 \ldots N_M - 1, \quad j = 0 \ldots N_M - 1 \tag{2.4}$$

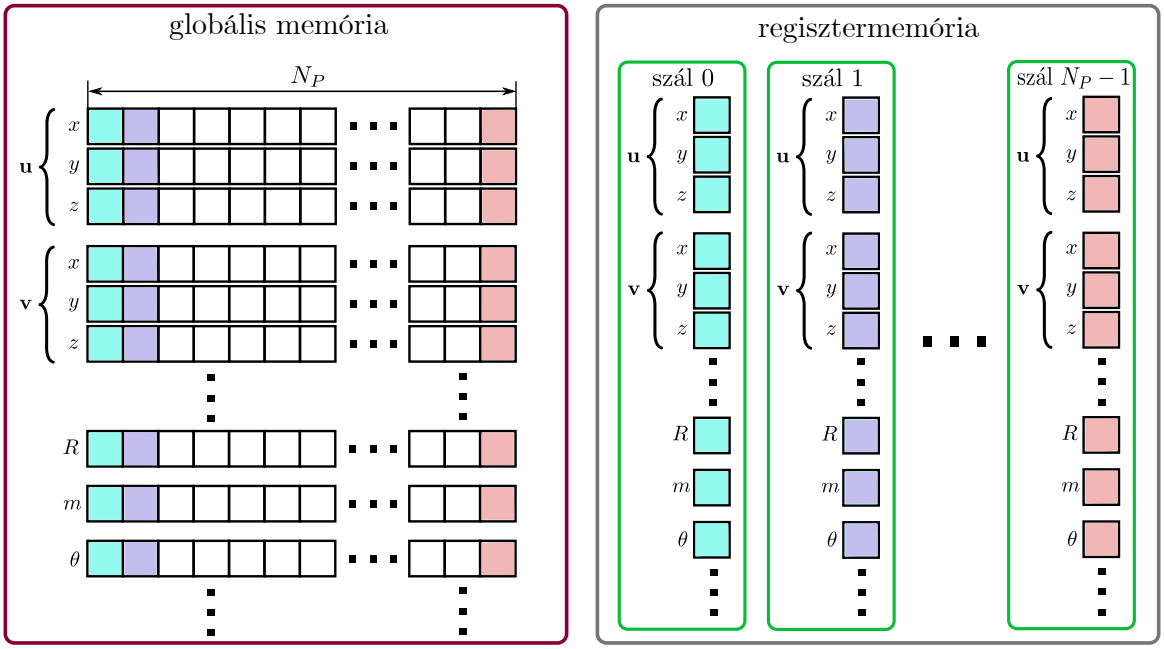
és ahol ν a Poisson tényező. Az ekvivalens ütközési tényező e^*, ekivalens súrlódási együttható μ^*, tapadási súrlódási együttható μ_0^* és gördülési súrlódási együttható μ_r^* a felhasználó által választott módon számolható. A választható számítási módok a minimum, maximum, átlag és harmonikus közép. Az ekvivalens ütközési tényező meghatározható a csillapítási tényező[32]

$$\beta = \frac{\log(e^*)}{\sqrt{\log(e^*)^2 + \pi^2}}. \tag{2.5}$$

Minden szemcse minden tulajdonsága tárolásra kerül a globális memóriában és onnan kerül beolvasásra a lényegesen gyorsabb regisztermemóriába a GPU-s programrész legelején. A 2.2. ábrán látható a bal oldalt, hogy kerül a globális memóriában tárolásra minden adat. A memóriaolvasás hatékony, amennyiben az adatok egymás mellől kerülnek beolvasásra, ahogyan az 1.4. fejezetben említésre került. A 3. kódrészlet mutatja a szemcsék tulajdonságainak a beolvasását a regiszter memóriába. Az első sorban meghatározásra kerül a szálindex, fontos megjegyezni, hogy a szálindex úgy kerül kiszámításra, hogy az egy warpban lévő szálak egymás melletti indexekkel rendelkeznek. Ahogyan korábban említésre került az 1.3.1. fejezetben egy warp 32 szálat tartalmaz. Ez a gyakorlatban azt jelenti, hogy a $0-31$ indexű szálakat az 0. warp számolja, a $32-63$ indexű szálakat az 1. warp és így tovább. A 3-5. sorban kerül a pozíció beolvasásra. Mivel egy warpon belül minden szál ugyanazt a műveletet végzi el, ezért a 3. sorban a 0. warp esetén a $0-31$ indexről kerülnek beolvasásra a szemcsék x pozíciói. Ahogyan a 2.2. ábrán látható, ezek egy folytonos memóriaterületen – azaz egymás mellett – helyezkednek el, ilyen esetben a GPU-k képesek egyetlen memóriaoperációval az egész memóriablokkot beolvasni a regisztermemóriába. A 4. és 5. sorban ugyanígy párhuzamosan kerül beolvasásra az y és z koordináta.

3. Kódrészlet. A regiszer memória párhuzamos feltöltése. A kódrészlet N_P példányban fut le, minden szemcse (azaz szál) esetén más-más adatok kerülnek betöltésre.

```
1 int tid = calculateThreadIndex(); //szál index
2 struct registerMemory rmem; //regiszter memória
3 rmem.u.x = globalmem.u.x[tid];
4 rmem.u.y = globalmem.u.y[tid];
5 rmem.u.z = globalmem.u.z[tid];
6 ...
7 rmem.R = globalmem.R[tid];
8 ...
```



2.2. ábra. A szemcsék tárolása a globális memóriában és a regiszterekben. Az azonos szinű cellák ugyanazon szemcse adatait tartalmazzák. A sorok összefüggő N_P nagyságú memóriaterületeket jelentenek, ahol N_P a szemcsék száma. Minden adat így kerül tárolásra, mint például a pozíció ($\boldsymbol{u}$), sebesség ($\boldsymbol{v}$), sugár (R), tömeg (m) és tehetetlenségi nyomaték (θ).

2.2. Szemcse-szemcse érintkezés

Szemcse-szemcse érintkezések könnyen detektálhatók, hiszen azok gömb alakúak. A 2.3a ábrán látható két kapcsolatban lévő szemcse. Két szemcse – legyen az $i.$ és $j.$ indexű – érintkezik amennyiben

$$R_i + R_j > d, \tag{2.6}$$

ahol $d = |\boldsymbol{u}_i - \boldsymbol{u}_j|$ a szemcsék közötti távolság. Az érintkezések detektálására a legegyszerűbb algoritmus végigmenni a szemcséken, minden szemcsepár között kiszámolni a távolságot és a 2.6. egyenlet szerint eldönteni, hogy a szemcsék érintkeznek-e. Ez a „brute force” megközelítés, azaz a nyers erő alkalmazása a feladatra. A probléma ezzel, hogy a távolság kiszámítása rendkívül időigényes, hiszen a

$$d = \sqrt{(u_{x,i} - u_{x,j})^2 + (u_{y,i} - u_{y,j})^2 + (u_{z,i} - u_{z,j})^2} \tag{2.7}$$

kifejezés gyökvonást tartalmaz és a szemcsepárok száma a szemcseszámmal négyzetesen nő.

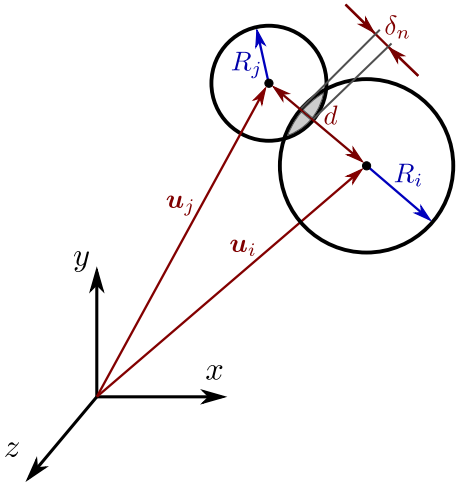
A szimuláció gyorsítható, hogyha a teljes számítási tartományt behálózzuk és csak az azonos, illetve szomszédos cellákban lévő szemcsék esetén kerül a d távolság meghatározásra. A 2.3b ábra mutatja, egy téglatest alapú tartomány felbontását $3 \times 2 \times 2$ cellára, a cellák mérete $\Delta x \times \Delta y \times \Delta z$. A tartomány minden irányban rendelkezik egy minimum és maximum értékkel ($x_{\min}, x_{\max}$ stb...). A következő módon rendelhetünk 3D-ben egy szemcsét egyértelműen egy cellához,

$$c_{\mathrm{id}} = \left\lfloor \frac{x - x_{\min}}{\Delta x} \right\rfloor + N_x \left\lfloor \frac{y - y_{\min}}{\Delta y} \right\rfloor + N_x \cdot N_y \left\lfloor \frac{z - z_{\min}}{\Delta z} \right\rfloor, \tag{2.8}$$

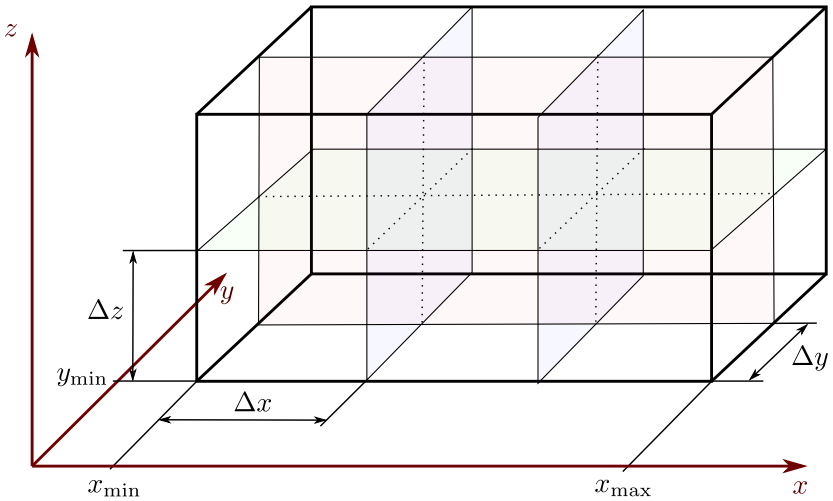
ahol c_{id} a cellaindex, N_x, N_y a cellák száma x, y irányban és $\lfloor \square \rfloor$ a floor függvény, amely pozitív számok esetén a szám egész részét adja. Ezután már csak azt kell megvizsgálni, hogy a két szemcse (i és j indexxel jelölt) azonos vagy szomszédos cellában található-e. Ez teljesül, ha

$$c_{\mathrm{id},i} = c_{\mathrm{id},j} + \begin{Bmatrix} 1 \\ 0 \\ -1 \end{Bmatrix} \cdot 1 + \begin{Bmatrix} 1 \\ 0 \\ -1 \end{Bmatrix} \cdot N_x + \begin{Bmatrix} 1 \\ 0 \\ -1 \end{Bmatrix} \cdot N_x \cdot N_y, \tag{2.9}$$

ahol a kapcsos zárójelek közötti számokat összesen 27 féle kombinációban választhatjuk.



(a) Két szemcse ütközése, a szemcsék sugara R, középpontjaik távolsága d, normál átfedésük δ_n és pozíciójuk $\boldsymbol{u}$.



(b) A számítási tartomány alatt lévő háló, ahol a cellák mérete $\Delta x \times \Delta y \times \Delta z$. A háló minden irányban a megadott minimum és maximum érték között terjed ki (lásd $x_{\min}$, $x_{\max}$...).

2.3. ábra. Szemcse-szemcse érintkezések detektálása és a detektálást felgyorsító háló felépítése.

A cellaindex és a cellaszomszédok meghatározása után két lehetőség adódik. Az egyszerűbb, hogyha továbbra is minden szemcsepáron végigmegyünk, azonban a d távolság csak akkor kerül kiszámításra, hogyha a 2.9. egyenlet teljesül, viszont ez 3D-ben 27 feltétel ellenőrzését igényli, így ez nem gyorsítja fel a számítást lényegesen. A másik lehetőség a szakirodalomban „cell-linked list" (CLL) néven ismert eljárás használata, ahol minden cellához eltároljuk, hogy jelenleg melyik szemcsék találhatók benne [13]. A GPU-n, ez az eljárás szál-divergenciához és ezáltal teljesítményveszteséghez vezet, hiszen minden cellában más-más mennyiségű szemcse található, azonban nagyszámú szemcse esetén nagyságrendekkel kevesebb kapcsolat kiszámolása szükséges. A GPUDEM-ben mindkét kapcsolatkeresési algoritmus implementálásra került.

A CLL módszer implementálása közel sem triviális párhuzamos architektúrákon. A probléma forrása, hogy párhuzamosan végrehajtásra kerülő szálak esetén, több szál is ugyanazt

a memóriaterületet használná, ez úgynevezett versenyhelyzetet eredményez. A 4. kódrészlet mutatja a cellalista feltöltését a globális memóriában. A kódban `globalmem.cellCount` egy $N_C = N_x \cdot N_y \cdot N_z$ méretű tömb, ami azt tartalmazza, hogy eddig hány szemcsét tartalmaz egy adott cella és `globalmem.linkedCellList` tartalmazza, hogy melyik cella mely indexű szemcséket foglalja magába. A `globalmem.linkedCellList` tömb mérete $N_C \cdot N_{P,C}$, ahol $N_{P,C}$ a cellában lévő szemcsék maximális száma. Megeshet, hogy egyszerre több szemcse is ugyanabban a cellában van, azaz a 3. sorban ugyanaz a `cid` cellaindex. Amennyiben ezen szemcsékhez tartozó szálak közel egyszerre hajtják végre a műveleteket, akkor 4. sorban probléma jelentkezik. A szál ami először ér el a 4. sorba inkrementálja a globális memóriában lévő változó `globalmem.cellCount[cid]` értékét, azonban az utasítástól számítva több száz cikluson keresztül még a régi érték található a globális memóriában a memória késleltetése miatt. Ez idő alatt, hogyha egy másik szál ugyanezen memóriaterületről olvas, akkor a helytelen korábbi értéket fogja beolvasni és az 5. sorban mindkét szál ugyanazon memóriaterületre fog írni, tehát a korábban beírt adat felülíródik.

4. Kódrészlet. Szálak közötti verseny a CLL kapcsolatkeresési eljárás detektálása során.

```
1 int tid = ...; //szálindex a szemcséhez
2 ...
3 int cid = ...; //cellaindex, amiben a szemcse van
4 int cell_count =globalmem.cellCount[cid]++;//szemcsék száma a cellában
5 globalmem.linkedCellList[cid*N + cell_count] = tid;
```

A CUDA definiál úgynevezett atomikus kifejezéseket, amelyek garántálják, hogy globális memória írás esetén más szál nem fér hozzá a megadott adathoz, mindaddig amíg az új adat ténylegesen meg nem jelent a memóriában. A 4. sorban a CUDA által definiált `atomicInc()` függvény használata garantálja a fentebb ismertett helyzet elkerülését.

A szemcse-szemcse érintkezések során továbbá az ekvivalens sugár R^* és tömeg m^* kiszámítása szükséges, amely szintén a Hertz-Mindlin kapcsolati modell szerint valósul meg [31], azaz

$$R^* = \frac{1}{R_j^{-1} + R_i^{-1}} \quad \text{és} \tag{2.10}$$

$$m^* = \frac{1}{m_j^{-1} + m_i^{-1}}, \tag{2.11}$$

ahol minden R^{-1} és m^{-1} már a szimuláció előtt kiszámításra került, így mindössze egy-egy összeadás, majd osztás elvégzése szükséges ebben a lépésben.

2.3. Szemcse-fal érintkezés

A fal a határoló geometriát, illetve a szemcsék közé helyezett talajművelő szerszámot jelenti, amelyek mind háromszögek által határolt geometriaként kerülnek megadásra STL fájlformátumban. Az STL fájlok a háromszögek csúcspontjait és a felületi normálisát tartalmazzák az 5. kódrészletben látható módon, ahol az első háromszög a $(0,0,0)$, $(1,0,0)$ és $(0,1,0)$ pontok által definiált. A GPUDEM-be bármilyen ASCII formátumú STL fájl beolvasható, a geometria a program futása során a GPU lokális memóriájában kerül eltárolásra. A lokális memória használatának az előnye a nagyobb sebességű adatelérés, azonban a mérete limitált, így mindössze néhány száz háromszög által határolt geometriák adhatók csak meg (a maximum méret a beállítások és a GPU típusának a függvénye).

5. Kódrészlet. Az STL fájlok felépítése.

```
solid
    facet normal 0.0 0.0 1.0
    outer loop
        vertex 0.0 0.0 0.0
        vertex 1.0 0.0 0.0
        vertex 0.0 1.0 0.0
    endloop
    endfacet
    ...
    ...
endsolid
```

Az STL fájlok alkalmazásának az előnye, hogy rendkívül könnyen detektálhatók a gömb és a háromszög közötti érintkezések. Legyen $\boldsymbol{p}$ a háromszög egyik csúcsába mutató vektor, $\boldsymbol{s}$ és $\boldsymbol{t}$ pedig mutasson a háromszög többi csúcsába $\boldsymbol{p}$-ből, ahogyan a 2.4. ábrán is látható. A vektorok kifeszítik az S síkot, amely a következőképp írható fel,

$$S := \boldsymbol{p} + \boldsymbol{s}\sigma + \boldsymbol{t}\tau, \tag{2.12}$$

ahol τ és σ koordináták az $(\boldsymbol{s}, \boldsymbol{t})$ koordinátarendszerben. A háromszögön belüli pontokra a következő egyenlőtlenségek teljesülnek,

$$\sigma \geq 0, \quad \tau \geq 0 \text{ és } \sigma + \tau \leq 1. \tag{2.13}$$

Legyen $\boldsymbol{n}$ az $\boldsymbol{s}$ és $\boldsymbol{t}$ vektorok által kifeszített, S síkra merőleges egységvektor. Amennyiben az $\boldsymbol{u} - \boldsymbol{p}$ vektort felírjuk az $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ koordinátarendszerben egyből megkapjuk a σ, τ és d koordinátákat, amely koordináták láthatók a 2.4. ábrán is. A transzformációs mátrix az $(\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}})$ és $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ koordinátarendszer között felírható, mint

$$\mathbf{T} = \begin{pmatrix} \boldsymbol{s}^t \\ \boldsymbol{t}^t \\ \boldsymbol{n}^t \end{pmatrix}. \tag{2.14}$$

Ezután a koordináták már egyszerűen számíthatók,

$$\begin{pmatrix} \sigma \\ \tau \\ d \end{pmatrix} = \mathbf{T}^{-t}(\boldsymbol{u} - \boldsymbol{p}), \tag{2.15}$$

ahol $\mathbf{T}^{-t}$ a $\mathbf{T}$ mátrix inverzének a transzponáltja, és d ténylegesen a szemcse $\boldsymbol{u}$ középpontjának a távolságát adja a síktól, hiszen az $\boldsymbol{n}$ bázisvektor egységnyi. A számítások és a tárolt adatok száma tovább csökkenthető, hiszen

$$\mathbf{T}^{-t} = \begin{pmatrix} \tilde{\boldsymbol{s}}^t \\ \tilde{\boldsymbol{t}}^t \\ \tilde{\boldsymbol{n}}^t \end{pmatrix} \text{ és } \sigma = \tilde{\boldsymbol{s}} \cdot (\boldsymbol{u} - \boldsymbol{p}), \quad \tau = \tilde{\boldsymbol{t}} \cdot (\boldsymbol{u} - \boldsymbol{p}), \quad d = \boldsymbol{n} \cdot (\boldsymbol{u} - \boldsymbol{p}), \tag{2.16}$$

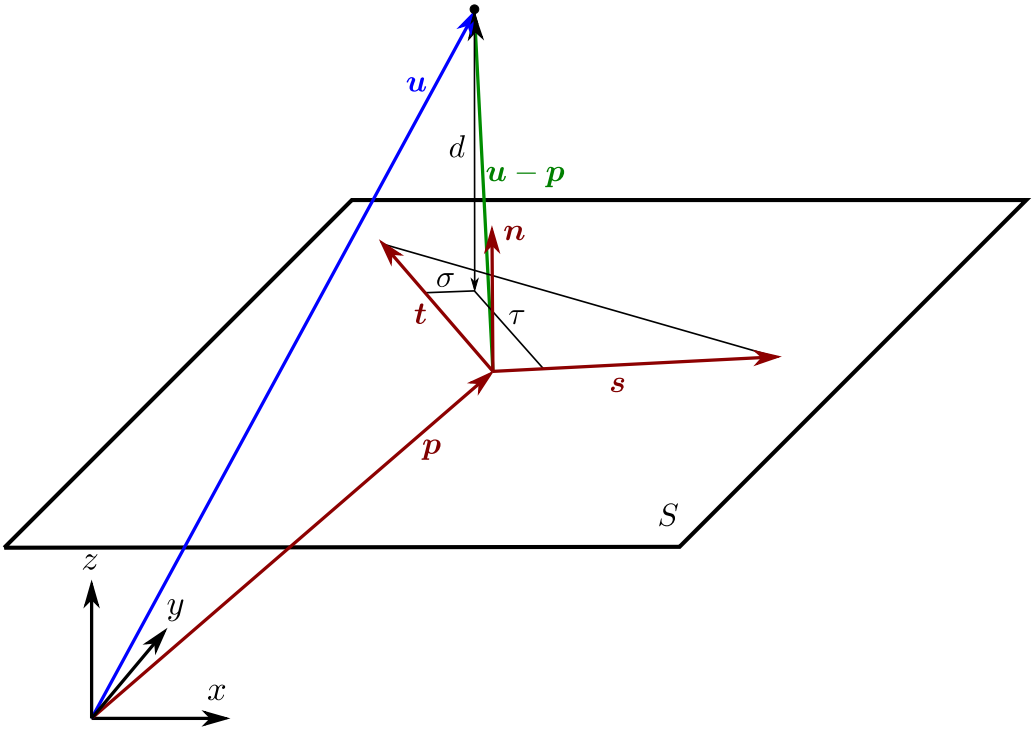
ahol a hullámmal jelölt vektorok a $\mathbf{T}^{-t}$ mátrix sorai. A fenti egyenletben az $\tilde{\boldsymbol{n}}$ vektorra nincsen szükség, mivel a d távolság egyszerűen az $\boldsymbol{u} - \boldsymbol{p}$ merőleges vetülete $\boldsymbol{n}$-re, tehát skalárszorzással számítható. A kiszámolt koordináták alapján az érintkezés már könnyen detektálható. A háromszög és egy gömb érintkezik amennyiben a következő két feltétel teljesül:

1. A gömb középpontja közelebb van a háromszög S síkjához mint annak a sugara, azaz $d < R$.

2. A gömb középpontjának a vetülete az S síkon a háromszög belsejében található, azaz teljesülnek a 2.13. egyenlőtlenségek.

Az ismertetett algoritmus ráadásul meglehetősen hatékony, mindössze 4 vektor tárolása szükséges a koordinátatranszformáció és az eltolás leírásához $(\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{t}}, \boldsymbol{n}, \boldsymbol{p})$. Ráadásul ezek már a szimuláció elején kiszámolhatók minden egyes háromszögre, amíg a megadott geometria csak transzlációs mozgást végez, hiszen a transzláció során csak a $\boldsymbol{p}$ vektor módosul. A továbbiakban a szemcse-fal érintkezés ugyanúgy számítható mint a szemcse-szemcse érintkezés, mindössze az ekvivalens sugár és tömeg definíciója más,

$$R^* = R \quad \text{és} \quad m^* = m. \tag{2.17}$$



2.4. ábra. Háromszög és gömb érintkezése, ahol S a háromszöget meghatározó $\boldsymbol{s}$ és $\boldsymbol{t}$ vektorok által kifeszített sík, $\boldsymbol{s}$ a háromszög csúcsának helyvektorai, $\boldsymbol{u}$ a szemcse középpontjának helyvektora. A σ, τ és d koordináták a háromszöghöz rendelt $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ koordinátarendszerben.

2.4. Kinematika

Miután a szemcse-szemcse és a szemcse-fal érintkezések meghatározásra kerültek, a következő lépés az érintkezés relatív kinematikai leírása. Először a szemcsék közötti relatív sebesség kerül meghatározásra, amely az s és a i indexű szemcse között a következő

$$\tilde{\boldsymbol{v}}_{s\to i} = \boldsymbol{v}_s - \boldsymbol{v}_i + \Big(\boldsymbol{\omega}_s R_s + \boldsymbol{\omega}_i R_i\Big) \times \boldsymbol{r}_{s\to i} \tag{2.18}$$

ahol $\boldsymbol{r}_{s\to i} = (\boldsymbol{u}_i - \boldsymbol{u}_s)/d$ az s szemcséből az i felé mutató egységvektor, amely a 2.5a ábrán látható a sebesség és szögesebesség vektorokkal együtt. Álló fallal való érintkezés során a $\boldsymbol{v}_i$ és a

$\boldsymbol{\omega}_i$ vektorok értelemszerűen nullák, továbbá fal esetén az $\boldsymbol{r}_{s\to i}$ vektor helyére a megfelelő háromszög $\boldsymbol{n}$ normálvektora kerül. A későbbiekben szükséges az $\tilde{\boldsymbol{v}}_{s\to i}$ vektor normál és tangenciális kompense, amelyek

$$\tilde{\boldsymbol{v}}_{s\to i,n} = \left(\tilde{\boldsymbol{v}}_{s\to i}\cdot \boldsymbol{r}_{s\to i}\right)\boldsymbol{r}_{s\to i} \quad \text{és} \tag{2.19}$$

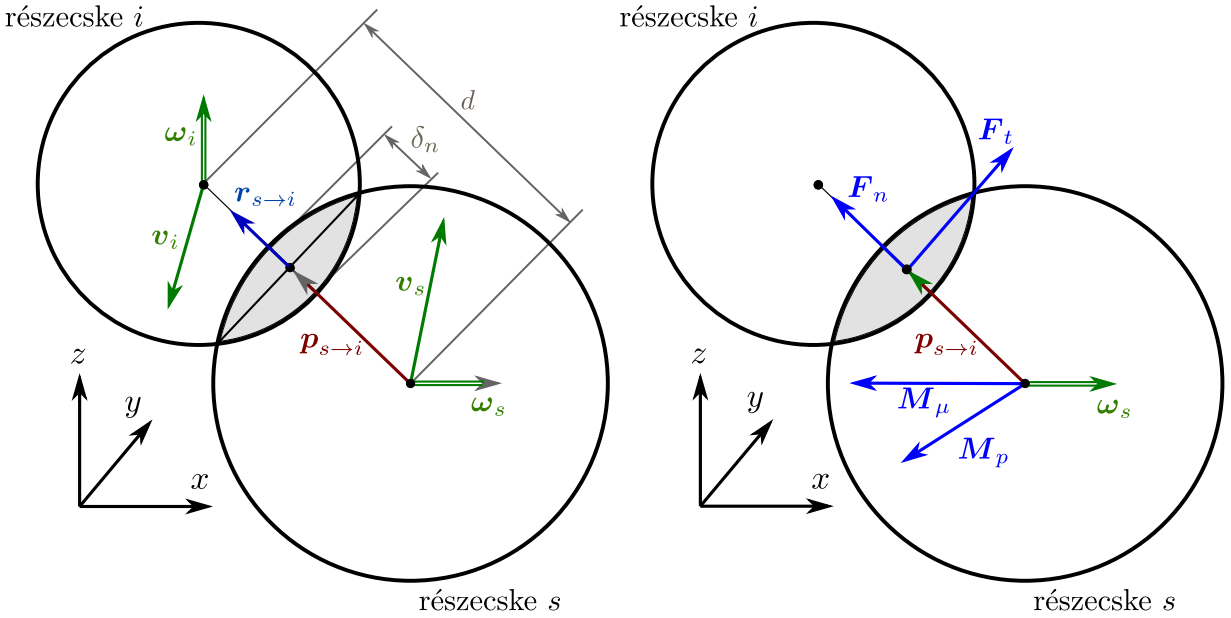
$$\tilde{\boldsymbol{v}}_{s\to i,t} = \tilde{\boldsymbol{v}}_{s\to i} - \tilde{\boldsymbol{v}}_{s\to i,n}. \tag{2.20}$$

Továbbá szükségeltetik az s és i indexxel jelölt szemcse tangenciális és normál átfedése, azaz δ_n és $\boldsymbol{\delta}_t$, amelyek [33]

$$\delta_n = d - (R_s + R_i) \quad \text{és} \tag{2.21}$$

$$\boldsymbol{\delta}_t^{\{n+1\}} = \boldsymbol{\delta}_t^{\{n\}} + \Delta t\cdot \tilde{\boldsymbol{v}}_{s\to i}, \quad \boldsymbol{\delta}_t^{\{0\}} = \boldsymbol{0}, \tag{2.22}$$

ahol $\boldsymbol{\delta}_t^{\{n\}}$ a tangenciális átfedés vektor az n. időlépésben amikor az s és i indexxel rendelkező szemcse érintkezik. Megjegyezendő, hogy minden szemcse minden érintkezéséhez más-más δ_n és $\boldsymbol{\delta}_t$ tartozik, az egyszerűség kedvéért az indexek innentől azonban elhagyásra kerülnek, a továbbiakban az s. és i. szemcse érintkezéséről lesz szó.



(a) A szemcsék sebessége és a $\boldsymbol{p}_{s\to i}$ vektor definíciója.

(b) Erők és nyomatékok két szemcse érintkezése során.

2.5. ábra. A szemcsék közötti erőszámításhoz szükséges mennyiségek 3D-ben. Az ábrákon $\boldsymbol{v}$ a sebesség vektor, $\boldsymbol{\omega}$ a szögsebesség vektor, $\boldsymbol{F}_n$ a normálerő, $\boldsymbol{F}_t$ a szemcsék között ható tangenciális erő, $\boldsymbol{M}_p$ az $\boldsymbol{F}_t$ erő hatásából adódó forgatónyomaték, $\boldsymbol{M}_\mu$ a gördülési ellenállásból adódó forgatónyomaték, d a szemcsék távolsága, δ_n a szemcsék normál átfedése és a $\boldsymbol{p}_{s\to i}$ vektor a szemcseközéppontból az ütközési pontba mutató vektor.

2.5. Erőszámítás

Az erőszámítás a Hertz-Mindlin modell szerint valósul meg [31]. A műveletek számának a csökkentése érdekében, a következő mennyiség kerül bevezetésre,

$$R_\delta = \sqrt{R^*\cdot \delta_n}. \tag{2.23}$$

Ezután számítható az egyenértékű normál és nyíró merevség,

$$S_n = 2E^* \cdot R_\delta \quad \text{és} \tag{2.24}$$

$$S_t = 8G^* \cdot R_\delta. \tag{2.25}$$

A korábban ismertetett mennyiségekből már számítható a rugalmasságból adódó normálerő $\boldsymbol{F}_{ne}$ és tangenciális erő $\boldsymbol{F}_{te}$, illetve a csillapításból adódó normálerő $\boldsymbol{F}_{nd}$ és tangenciális erő $\boldsymbol{F}_{td}$,

$$\boldsymbol{F}_{ne} = -\frac{4}{3} E^* R_\delta \delta_n \boldsymbol{r}_{s\to i}, \tag{2.26}$$

$$\boldsymbol{F}_{te} = -\boldsymbol{\delta}_t \cdot S_t, \tag{2.27}$$

$$\boldsymbol{F}_{nd} = -2\sqrt{\frac{5}{6}} \cdot \beta \cdot \sqrt{S_n \cdot m^*} \cdot \tilde{\boldsymbol{v}}_{s\to i,n} \quad \text{és} \tag{2.28}$$

$$\boldsymbol{F}_{td} = -2\sqrt{\frac{5}{6}} \cdot \beta \cdot \sqrt{S_t \cdot m^*} \cdot \tilde{\boldsymbol{v}}_{s\to i,t}. \tag{2.29}$$

A normál és tangenciális erők a rugalmasságból és csillapításból származnak, azaz

$$\boldsymbol{F}_n = \boldsymbol{F}_{ne} + \boldsymbol{F}_{nd} \quad \text{és} \tag{2.30}$$

$$\boldsymbol{F}_t = \boldsymbol{F}_{te} + \boldsymbol{F}_{td}, \tag{2.31}$$

ezek az erők láthatók a 2.5b ábrán. A szemcsék megcsúsznak egymáson, amennyiben a tapadási feltételel nem teljesül, azaz

$$|\boldsymbol{F}_t| > \mu_0^* \cdot |\boldsymbol{F}_n|, \tag{2.32}$$

ahol μ_0^* a 2.1 fejezetben bemutatott ekvivalens tapadási súrlódási tényező. Ekkor a tangenciális erő a következő módon kerül csökkentésre:

$$\boldsymbol{F}_{t,\text{új}} = \mu^* \cdot \boldsymbol{F}_t \cdot \frac{|\boldsymbol{F}_n|}{|\boldsymbol{F}_t|}, \tag{2.33}$$

ahol μ^* az ekvivalens csúszási súrlódási együttható, amely μ_0^*-hoz hasonlóan számítható. Így $|\boldsymbol{F}_{t,\text{új}}| = \mu^* \cdot |\boldsymbol{F}_n|$ és $|\boldsymbol{F}_{t,\text{új}}|$ iránya megegyezik $|\boldsymbol{F}_t|$ irányával. Az új módosított erővel már számítható a szemcsére ható erő, amely egy érintkezésből származik

$$\boldsymbol{F} = \boldsymbol{F}_n + \begin{cases} \boldsymbol{F}_t, & \text{ha} \quad |\boldsymbol{F}_t| \le \mu_0^* \cdot |\boldsymbol{F}_n| \\ \boldsymbol{F}_{t,\text{új}}, & \text{ha} \quad |\boldsymbol{F}_t| > \mu_0^* \cdot |\boldsymbol{F}_n| \end{cases}. \tag{2.34}$$

A tangenciális erő forgatónyomatékot okoz, mivel a hatásvonala nem megy át a szemcse középpontján, a nyomaték

$$\boldsymbol{M}_p = \boldsymbol{p}_{s\to i} \times \boldsymbol{F}_t, \tag{2.35}$$

ahol $\boldsymbol{p}_{s\to i}$ az s és i szemcse érintkezési pontjába mutató vektor, aminek definíciója a 2.5. ábrákon látható. A gördülési súrlódás figyelembe vétele egy a szögsebesség vektor irányába mutató nyomatékot eredményez [32],

$$\boldsymbol{M}_\mu = -\mu_r^* \cdot |\boldsymbol{F}_n| \cdot |\boldsymbol{p}_{s\to i}| \cdot \frac{\boldsymbol{\omega}_s}{|\boldsymbol{\omega}_s|}. \tag{2.36}$$

A szemcsére ható forgatónyomaték egy érintkezésből a gördülési súrlódás figyelembe vételével tehát,

$$\boldsymbol{M} = \boldsymbol{M}_p + \boldsymbol{M}_\mu. \tag{2.37}$$

Egy szemcse, több fallal és szemcsével is érintkezhet, a 2.18.-2.37. egyenletekben leírt számításokat tehát minden szemcse esetén el kell végezni az összes érintkezésre. Azt, hogy az s. szemcse hány fallal és másik szemcsével érintkezik $N_{C,s}$ adja meg, $N_{C,\max}$ pedig az érintkezések maximum számát jelöli. Az s indexxel jelölt szemcsére ható teljes erő és nyomaték az összes érintkezésből származó erő és nyomaték összegzéséből adódik,

$$\boldsymbol{F}_{s,\text{total}} = \sum_{j=0}^{N_{C,s}-1} \boldsymbol{F}_{s,j} \quad \text{és} \tag{2.38}$$

$$\boldsymbol{M}_{s,\text{total}} = \sum_{j=0}^{N_{C,s}-1} \boldsymbol{M}_{s,j}, \quad \text{ahol} \quad N_{C,s} \leq N_{C,\max}, \tag{2.39}$$

ahol $\boldsymbol{F}_{s,j}$ és $\boldsymbol{M}_{s,j}$ az s-el jelölt szemcsére ható erő és nyomaték annak j indexxel jelölt érintkezéséből.

2.6. Időbeni megoldás

Az időbeni megoldás előállításához először a gyorsulások meghatározásra szükséges. Gömb akalú szemcsékről lévén szó,

$$\boldsymbol{a}_s^{\{n\}} = m_s^{-1} \cdot \boldsymbol{F}_{s,\text{total}}^{\{n\}} + \boldsymbol{g} \quad \text{és} \tag{2.40}$$

$$\boldsymbol{\beta}_s^{\{n\}} = \theta_s^{-1} \cdot \boldsymbol{M}_{s,\text{total}}^{\{n\}}, \tag{2.41}$$

ahol $\boldsymbol{a}_s^{\{n\}}$ az s. szemcse gyorsulása és $\boldsymbol{\beta}_s^{\{n\}}$ az s. szemcse szöggyorsulása az n. időlépésben. Az egyenletben $\boldsymbol{g}$ a gravitációs gyorsulás vektora. A gyorsulásokból már kiszámolható a megadott lépésközzel az új sebesség, pozíció és szögsebesség, erre több időlépés séma választható.

Euler-módszer

A legegyszerűbb választás az Euler-módszer, ekkor a sebesség $\boldsymbol{v}_s$, pozíció $\boldsymbol{u}_s$ és szögsebesség $\boldsymbol{\omega}_s$ a következő módon kerül meghatározásra,

$$\boldsymbol{v}_s^{\{n+1\}} = \boldsymbol{v}_s^{\{n\}} + \Delta t \cdot \boldsymbol{a}_s^{\{n\}}, \tag{2.42}$$

$$\boldsymbol{u}_s^{\{n+1\}} = \boldsymbol{u}_s^{\{n\}} + \Delta t \cdot \boldsymbol{v}_s^{\{n\}} \quad \text{és} \tag{2.43}$$

$$\boldsymbol{\omega}_s^{\{n+1\}} = \boldsymbol{\omega}_s^{\{n\}} + \Delta t \cdot \boldsymbol{\beta}_s^{\{n\}}, \tag{2.44}$$

ahol Δt a lépésköz. A pozíciószámítás során figyelembe vehetők magasabb rendű tagok is [16] és a pozícióra ezáltal pontosabb becslés adható, ha

$$\boldsymbol{u}_s^{\{n+1\}} = \boldsymbol{u}_s^{\{n\}} + \Delta t \cdot \boldsymbol{v}_s^{\{n\}} + \frac{1}{2}\Delta t^2 \boldsymbol{a}_s^{\{n\}}. \tag{2.45}$$

Amennyiben feltételezzük, hogy egy időlépés során a gyorsulás végig állandó, akkor a 2.45. egyenlet a pontos pozíciót adja meg, ezért a GPUDEM-ben ez az időlépés séma az „Exact” nevet kapta. A gyorsulás természetesen nem állandó az időlépés alatt, a valóságban a szemcse legkisebb elmozdulása is megváltoztatja az erőket és ezáltal a gyorsulást.

Adams-módszer

A megoldóban implementálásra került a másodrendű Adams-Bashforth – Adams-Moulton időléptetés is [34, 35]. A módszer előnye, hogy másodrendben pontos, azonban eggyel több időlépés eltárolását igényli. A módszer lényege, hogy a sebesség explicit módok kerül kiszámításra, Adams-Bashforth módszerrel,

$$\boldsymbol{v}_s^{\{n+1\}} = \boldsymbol{v}_s^{\{n\}} + \Delta t \cdot \left(\frac{3}{2}\boldsymbol{a}_s^{\{n\}} - \frac{1}{2}\boldsymbol{a}_s^{\{n-1\}} \right) \quad \text{és} \tag{2.46}$$

$$\boldsymbol{\omega}_s^{\{n+1\}} = \boldsymbol{\omega}_s^{\{n\}} + \Delta t \cdot \left(\frac{3}{2}\boldsymbol{\beta}_s^{\{n\}} - \frac{1}{2}\boldsymbol{\beta}_s^{\{n-1\}} \right). \tag{2.47}$$

Ezután az implicit Adams-Moulton módszerrel kerül meghatározásra a pozíció,

$$\boldsymbol{u}_s^{\{n+1\}} = \boldsymbol{u}_s^{\{n\}} + \frac{1}{2} \cdot \Delta t \cdot \left(\boldsymbol{v}_s^{\{n\}} + \boldsymbol{v}_s^{\{n+1\}} \right), \tag{2.48}$$

ahol $\boldsymbol{v}_s^{\{n+1\}}$ már ismert a 2.46. egyenletből. A módszer több memóriát igényel, hiszen az eggyel régebbi gyorsulás és sebesség értékeket is tárolni kell. Az első lépés során, azaz $n = 0$ esetén a 2.46. és a 2.47. egyenletekben nincsen információnk $\boldsymbol{a}_s^{\{-1\}}$ és $\boldsymbol{\beta}_s^{\{-1\}}$ értékéről, így az első lépésben mindig a 2.42.-2.44. egyenletekben leírt Euler-módszer kerül alkalmazásra. Léteznek magasabb rendű Adams-módszerek, amelyek előnye, hogy nagyobb időlépéssel is megfelelően pontos megoldást adnak. A feladat jellegéből adódóan az időlépés nem növelhető lényegesen, hiszen a szemcsék mozgása során kapcsolatok folyamatosan jönnek létre és szűnnek meg, amely ütközések időbeni felbontása szükséges, így magasabb rendű numerikus módszerek implementációja nem indokolt.

2.7. Ki/bemeneti fájlok

A kezdeti szemcse eloszlás beolvasása és a szemcsék bizonyos időközönkénti elmentése elengedhetetlen. A GPUDEM alapvetően `csv` és `vtu` kiterjesztésű fájlok olvasására és írására képes. A `csv`, azaz vesszőkkel határolt szöveg (*comma separated text*) egy egyszerű szövegfájl. A GPUDEM-be a következő fejléccel rendelkező fájlok olvashatók be:

```
x, y, z, R
```

ahol az első három oszlopba a szemcsék x, y, z koordinátái kerülnek, a negyedik oszlopba pedig a szemcsék sugarai és minden sor egy szemcsét ír le. Ugyanilyen típusú kimeneti fájlok is létrehozhatók.

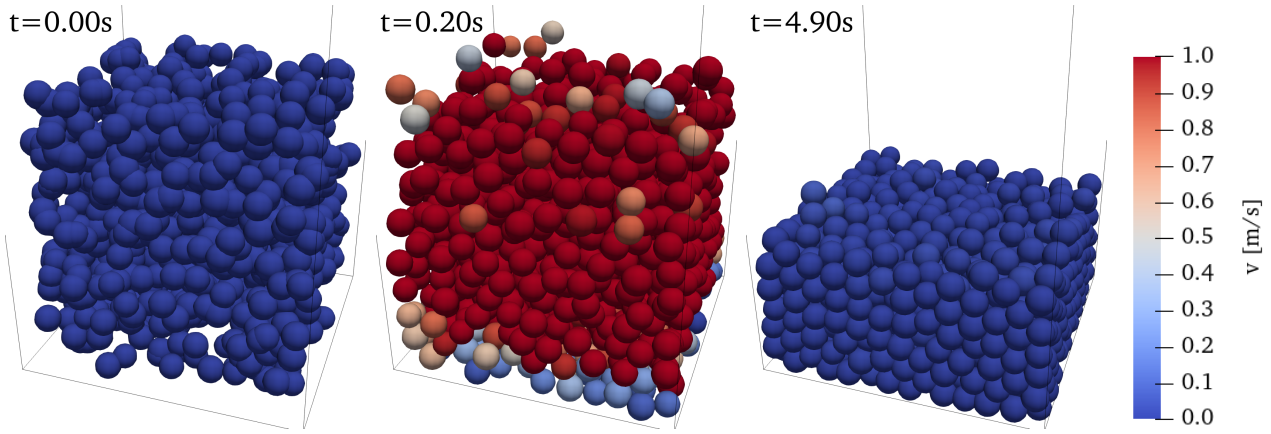
A `vtu` fájlokat a nyílt-forráskódú VTK Visualization Toolkit definiálja [36]. A fájltípus előnye, hogy közvetlenül beolvasható Paraview-ba és hozzáadható minden szemcséhez bármilyen további tetszőleges tulajdonság, például a szemcsék sebessége, a szemcsékre ható erők vagy akár a szemcsék anyaga. A Paraview egy nyílt-forráskódú adatvizualízációs szoftver [37, 38], amely különösen alkalmas nagy mennyiségű adat kezelésére így akár több százezer szemcse megjelenítésére is.

3. Tesztelés

Az elkészült GPUDEM tesztelése elengedhetetlen. A tesztelés során a legfontosabb szempont, hogy a szimuláció valós eredményeket adjon. További szempont az implementált módszerek hatékonyságának a vizsgálata és a program futásának az elemzése a GPU-n (skálázás és profilozás).

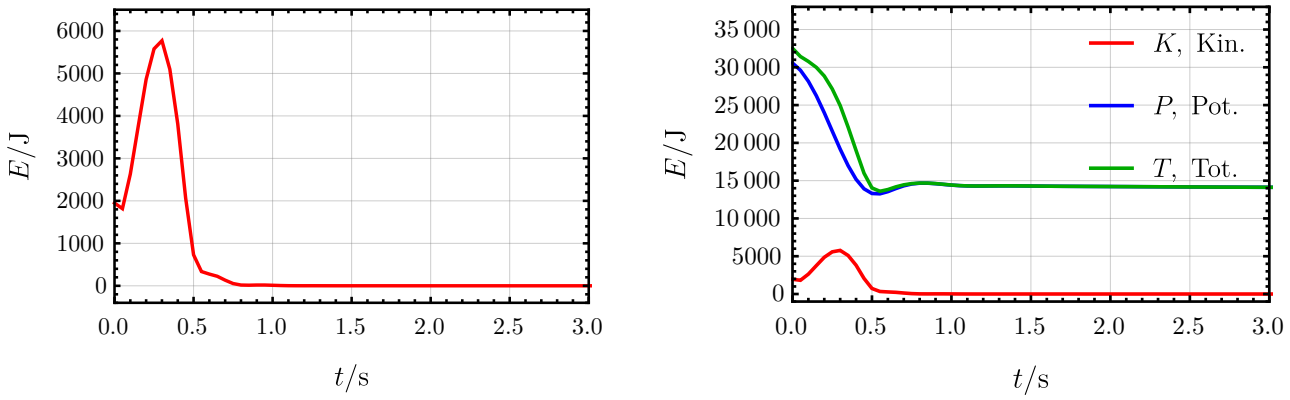
3.1. Gravitációs ülepítés

Az ülepítés lényege egy megfelelő kezdeti szemcseeloszlás létrehozása további szimulációkhoz. Gravitációs ülepítés esetén a szemcsék kezdetben véletlenszerű helyekre kerülnek egy adott tartományban, a gravitáció hatására leesnek és egymásra rétegződnek. Az ülepítés során elvárás, hogy minden szemcse sebességge nullához tartson. Az ülepítésre egy példa a 3.1. ábrán látható, kezdetben a szemcsepozíciók véletlenszerűen kerültek generálásra egyenletes eloszlás szerint, ez a kezdeti szemcseeloszlás látható a 3.1a ábrán. A 3.1c ábrán láthatók a már teljesen leülepedett szemcsék, ekkor már minden szemcse sebessége közel nulla. Az első szimuláció beállításai a 6.2. táblázatban találhatók az #1 oszlopban.



(a) Kezdeti szemcsék (b) Szemcsék 0.2 s után (c) Az ülepített szemcsék

3.1. ábra. $N_P = 1024$ szemcse ülepítése a GPUDEM-ben. Az ábrák színezése a sebességvektor nagysága alapján történt a jobb oldalt látható skála szerint.



(a) Kinetikus energia az ülepítés során

(b) Kinetikus, potenciális és $T = P + K$ energia az ülepítés során

3.2. ábra. Az energiák $N_P = 1024$ szemcse ülepítése során a GPUDEM-ben.

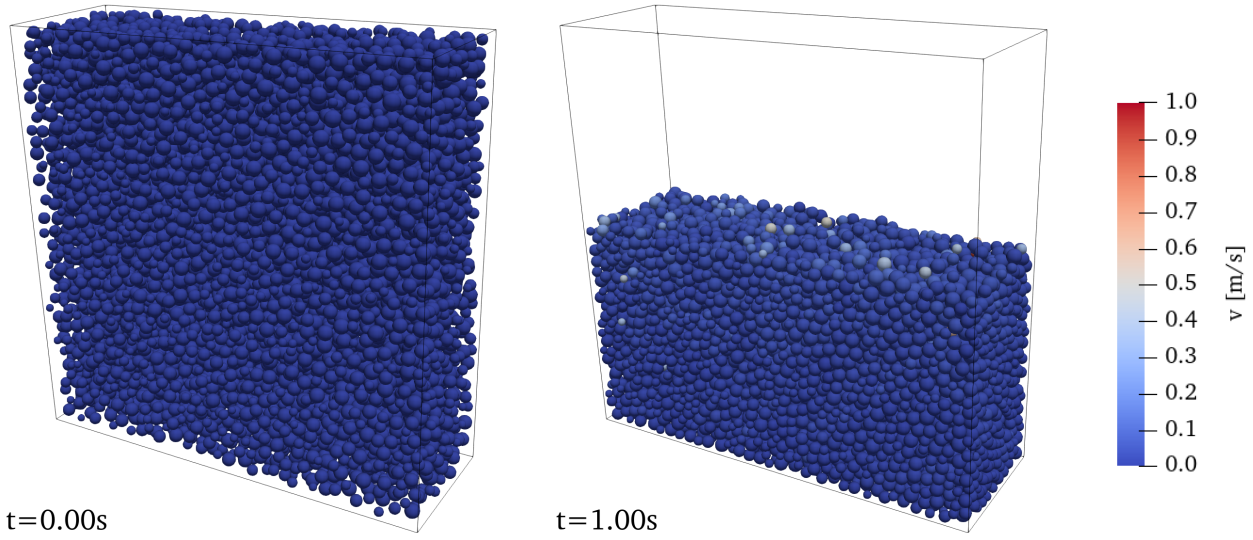
Az ülepítés sikerességét a kinetikus és potenciális energia monitorozásával is meg lehet állapítani. A szemcsehalmaz teljes kinetikus energiája

$$K = \frac{1}{2} \sum_{i=0}^{N_P-1} \Big(m_i \cdot \boldsymbol{v}_i \cdot \boldsymbol{v}_i + \theta_i \cdot \boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_i \Big), \tag{3.1}$$

ahol $\boldsymbol{v}_i$ az i. szemcse sebessége és $\boldsymbol{\omega}_i$ a szögsebessége. A szemcsehalmaz potenciális energiája

$$P = \sum_{i=0}^{N_P-1} m_i \cdot g \cdot \boldsymbol{u}_{z,i}, \tag{3.2}$$

amennyiben a gravitációs gyorsulás vektora $\boldsymbol{g} = [0, 0, -g]^t$ és ahol $\boldsymbol{u}_{z,i}$ az i. szemcse z koordinátája. A 3.1. ábrán látható szimuláció során számolt kinetikus energiát a 3.2a ábra mutatja. Megfigyelhető, hogy kezdetben a kinetikus energia növekszik, hiszen a szemcsék esés közben folyamatosan gyorsulnak, majd a kinetikus energia csökkenni kezd, ahogy egyre több szemcse leülepedett és végül nullához konvergál. A kinetikus energia nem csökken sosem pontosan nullára a véges időlépésből és a kerekítési hibákból adódóan. A 3.2b ábrán a kinetikus energia K, potenciális energia P és a kettő összege $T = P + K$ került ábrázolásra. Megfigyelhető, hogy a $T = P + K$ energia nem csökken a szimuláció során végig, ennek az oka, hogy a szemcsék az ütközés és átfedés során rugalmas energiát tárolnak. A 3.2. ábrákon összességében látszik, hogy az ülepítés jól működik, hiszen a kinetikus energia nullához tart a potenciális pedig egy konstans értékhez konvergál.



(a) Kezdeti szemcsék (b) Az ülepített szemcsék

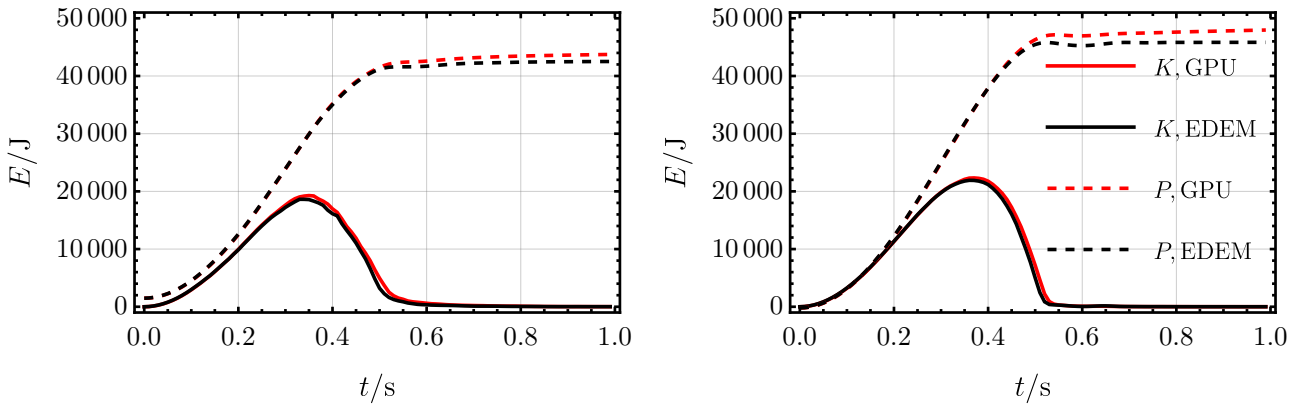
3.3. ábra. $N_P = 8519$ szemcse ülepítése a GPUDEM-ben. Az ábrák színezése a sebességvektor nagysága alapján történt a jobb oldalt látható skála szerint.

3.2. Validáció

A validáció célja meggyőződni, hogy a programban minden módszer megfelelően került implementációra, azaz nincsenek benne elírások és logikai hibák. A validáció egy módja az összehasonlítás más programcsomagokkal, amelyek hasonló módszereket alkalmaznak. Az egyik

legelterjedtebb DEM szoftver az EDEM, amelyben számos kapcsolati modell, például a Hertz-Mindlin modell is implementálva van. Két eset került összehasonlításra, az egyikben 509 db a másikban pedig 8519 db szemcse került alkalmazásra. Mind az EDEM-ben és a GPUDEM-ben ugyanaz a kezdeti szemcseeloszlás került megadásra, amely a 3.3a ábrán látható. Továbbá, ugyanazon anyagparaméterek és időlépés került beállításra, ahogyan a 6.2. táblázat #2 oszlopában látható. Mindkét szoftverben 1 s alatt minden szemcse sebessége közel nulla lesz, az ülepített szemcsék a 3.3b ábrán láthatók.

A potenciális és kinetikus energia ábrázolásra került a 3.4. ábrán. Megfigyelhető, hogy a szemcsefelhő potenciális és kinetikus energiája mindkét esetben nagyon hasonló. A kinetikus energia minden esetben nullához tart, ahogy az elvárható. A potenciális energiában kisebb eltérés adódik egy idő után, ez valószínűleg a GPUDEM és az EDEM implementációjának a különbsége okozza. Az EDEM forráskódja nem ismert, így pontosan nem tudni, hogyan került a kapcsolatkeresés és a Hertz-Mindlin modell implementációra.



(a) $N_P = 509$ (b) $N_P = 8519$

3.4. ábra. Az energiák K és $-P$ az ülepítés során GPUDEM-ben és EDEM-ben.

A futási idők összehasonlítása a 3.1. táblázatban látható. Feltüntetésre került továbbá, hogy melyik program milyen hardveren került futtatásra. Az 509 szemcsét tartalmazó eset közel nyolcszor gyorsabbnak bizonyult GPU használatával, míg a nagyobb probléma esetén 13-szor gyorsulás figyelhető meg. Érdemes megjegyezni, hogy még a 8519 szemcsét tartalmazó eset sem használja ki teljesen a GPU erőforrásait, ennél lényegesen nagyobb szimulációk is futtathatók.

3.1. táblázat. A szimuláció futási ideje

Program	$N_P = 509$	$N_P = 8519$	Hardver	Teljesítmény
EDEM	8 s	63 s	Intel Core i7-4700MQ	218 GFLOPS
GPUDEM	1.0 s	4.8 s	GeForce RTX 3060 Ti	16.2 TFLOPS

3.3. Skálázhatóság

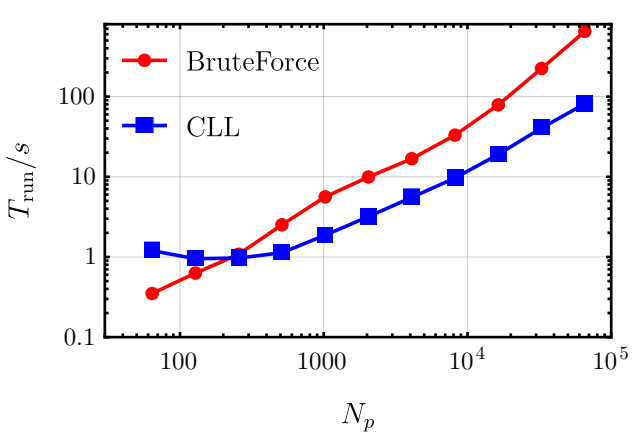
Az előző két példán már láthattuk, hogy a GPUDEM valós eredményeket ad ülepítés esetén. A kérdés, hogy változik a program futási ideje a szemcseszám változtatásával. A skálázás vizsgálatához ugyanazon paraméterek, de más-más szemcseszám mellett kerültek elvégzésre szimulációk, mind a BruteForce és a CLL kapcsolatkeresési eljárással. A BruteForce kapcsolatkeresési

eljárás lényege, hogy minden lehetséges szemcsepár között kiszámításra került a távolság, a CLL pedig a számítási tartományhoz egy hálót rendel és csak a szomszédos cellákban lévő szemcsék között kerül a távolság kiszámításra, a két eljárás a 2.2. fejezetben került részletesen bemutatásra. A szimulációk minden esetben $\Delta t = 1 \cdot 10^{-4}$ s lépésközt használtak és 2500 lépést tettek meg. A beállított lépésköz mellett a szimulációk stabilak, azaz az ülepítés során a kinetikus energia nullához konvergál, minden beállított szemcseszám mellett. A paraméterek a 6.2. táblázatban láthatók a #3 oszlopban. A futási időket (T_{run}) a 3.5a ábra mutatja a szemcseszám függvényében logaritmikus diagrammon. Kicsi szemcseszám esetén ($N_P < 250$) a BruteForce megközelítés hatékonynak bizonyul, nagy szemcseszám esetén ($N_P > 10^4$) azonban a futási idő már közel háromszorosára növekszik a szemcseszám duplázásával, például

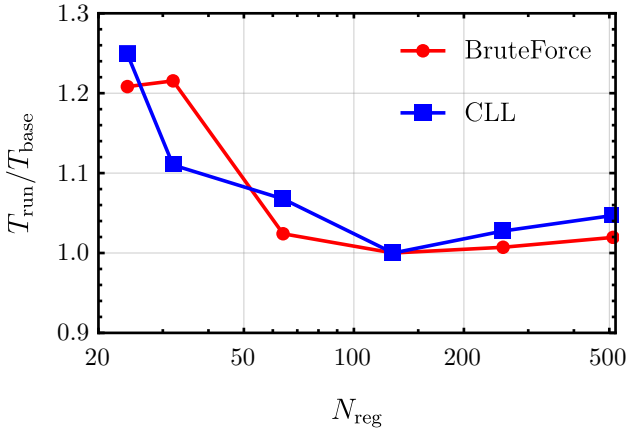
$$\frac{T_{\text{run}}(N_P = 32768)}{T_{\text{run}}(N_P = 16384)} = 2.84 \quad \text{és} \quad \frac{T_{\text{run}}(N_P = 65536)}{T_{\text{run}}(N_P = 32768)} = 2.91, \tag{3.3}$$

tehát a skálázás nem lineáris. A gyorsabb növekedés oka, hogy a műveletek száma N_P^2-el arányos a kapcsolatkeresés során, azonban a futási idő növekedés mégsem négyszereződik a szemcseszám duplázásával, mivel a GPU erőforrásai még 10^5 szemcse használata során sincsenek teljesen kihasználva.

A CLL kapcsolatkeresési eljárás esetén a futási idő a szemcseszám növelésével először csökken. Az ok, hogy a háló ebben az esetben $N_C = 28^3 = 21952$ cellát tartalmaz és minden cellában maximum 8 szemcse lehet, azonban a cellákat minden időlépés előtt újra kell inicializálni, és mivel a szálak száma alacsony, ezért lassú végigmenni minden cellán, hiszen egy szálnak sokkal több cellával kell foglalkoznia. Ez egészen $N_P \approx 250$-ig igaz, mivel egyre több szál kerül alkalmazásra és egyre gyorsabb lesz a cellák frissítése. A szemcseszám további növelése már a futási idő növekedésével jár, kétszer annyi szemcsével számolni nagyjából kétszer annyi ideig tart. Ebben rejlik a CLL eljárás előnye, hiszen itt a műveletek száma N_P-vel arányos, így a skálázás lineáris. Összességében megállapítható, hogy a CLL kapcsolatkeresési eljárás nagy szemcseszám esetén lényegesen gyorsabb és a szemcseszám növelésével egyre nagyobb a különbség a kettő futási idejében, $N_P = 65536$ szemcse esetén a CLL már nyolcszor gyorsabbnak bizonyult.



(a) A futási idő T_{run} a szemcseszám N_p függvényében.



(b) A futási idő T_{run} a szálankénti regiszterszám N_{reg} függvényében. T_{base} a futási idő 128 regiszter használatával.

3.5. ábra. A #3 ülepítés futási ideje különböző kapcsolatkeresési algoritmusok (BruteForce és CLL) alkalmazása mellett.

3.4. Regiszterhasználat

A GPU-s alkalmazások egy fontos beállítása a szálankénti regiszterszám. A regisztermemória a számítási egységekhez közel helyezkedik el és rendkívül gyors elérésű, ahogy az 1.3.1. fejezetben bemutatásra került. A szálankénti regiszterszám N_{reg} növelésének két ellentétes hatása van:

1. A regiszterszám növelésével egyre több adat kerül a számítási egységekhez közel tárolásra, így a memóriaolvasás és írás átlagosan gyorsabb lesz.

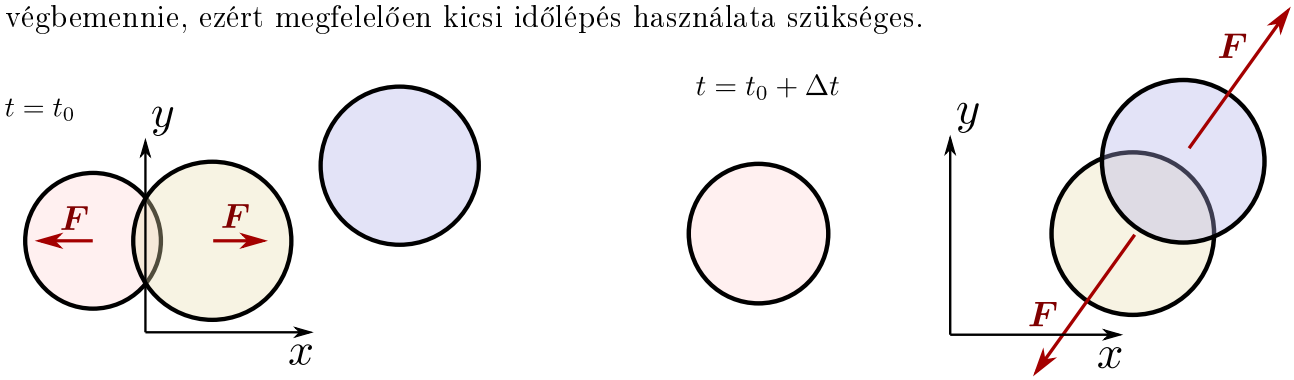
2. Minél több regisztert használ egy szál, annál kevesebb szál fér el egy SM-en (multiprocesszoron), mivel a rendelkezésre álló memória mennyisége limitált, így egyszerre kevesebb rendelkezésre álló szál lesz és a számítási egységek kihasználtsága csökkenhet.

A megfelelő regiszterszám beállításával a program futási idejét gyorsíthatjuk. A 3.5b ábrán került ábrázolásra a normált futási idő a regiszterszám függvényében mindkét kapcsolatkeresési eljárás használata során. A futási idő T_{run} minden esetben a T_{base}-el, azaz a 128 regiszter melletti futási idővel került osztásra, $T_{\text{base}} = 30.73\,\text{s}$ a BruteForce eljárás esetén és $T_{\text{base}} = 8.72\,\text{s}$ a CLL esetén. Az ábrán látható, hogy az elején a regiszterszám növelésével a futási idő csökken egészen $N_{\text{reg}} = 128$-ig, ekkora még nem jelentkezik a GPU kihasználtságának a csökkenése, mivel összességében nincsen túl sok regiszter használva. A regiszterszám további növelése azonban már lényegesen csökkenti a szálak számát, ami elfér egy SM-en, így a futási idő újból nőni kezd.

A regiszterszám változtatása természetesen nem befolyásolja az eredményeket, mindössze a program memóriahasználatát módosítja. A továbbiakban $N_{\text{reg}} = 128$ kerül alkalmazásra, mivel ez bizonyul a leghatékonyabbnak a 3.5b ábra alapján.

3.5. Időlépés

Az időlépés növelése lehetővé teszi a futási idő csökkentését, azonban túl nagy időlépés stabilitásvesztéssel járhat. A stabilitásvesztés egyszerűen magyarázható. Tegyük fel, hogy két szemcse összeütközött t_0 időpontban ahogyan a 3.6a ábrán látható. Az ütközés során a szemcsére a szemcsék közötti átfedéssel arányos erő hat, ha az időlépés nagy és ezzel az erővel számolt gyorsulással továbbszámolunk akkor egy időlépés alatt a szemcse túlságosan is elmozdulhat. Ez az elmozdulás olyan nagy is lehet, hogy rögtön átfedésbe kerül egy másik szemcsével ahogyan a 3.6b ábra mutatja, azonban ebben az esetben a szemcsére az előzőnél is nagyobb erő hat, és a következő időlépésben még nagyobb lesz az elmozdulása. Az ütközéseknek fokozatosan kell végbemennie, ezért megfelelően kicsi időlépés használata szükséges.



(a) A szemcsék $t = t_0$-nál ütköznek.

(b) A szemcsék $t = t_0 + \Delta t$-nél.

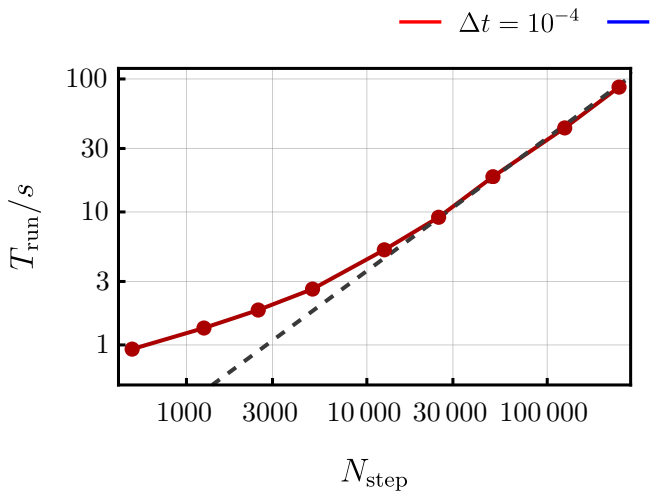
3.6. ábra. Szemcsék nem-fizikai ütközése túl nagy időlépés esetén.

A korábban a 3.1. fejezetben ismertett ülepítés elvégzésre került különböző időlépések mellett. A szimuláció hossza $T = 2.5\,\mathrm{s}$, az időlépés pedig $\Delta t = 1 \cdot 10^{-5}\,\mathrm{s} \ldots 1 \cdot 10^{-2}\,\mathrm{s}$ között került állításra. A lépésszám N_{step} azt adja meg, hogy összesen hány időlépést kell elvégezni a szimulációban, ami

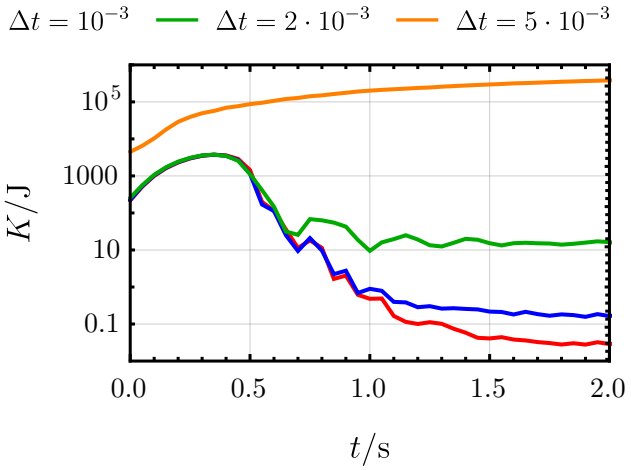
$$N_{\mathrm{step}} = \frac{T}{\Delta t}. \tag{3.4}$$

A futási idő a lépésszám függvényében a 3.7a ábrán látható. Az ábrán a szaggatott vonal a lineáris futási idő növekedést mutatja, megfigyelhető, hogy ha a lépésszám elég nagy $N_{\mathrm{step}} > 10000$ akkor kétszerannyi időlépés megtétele kétszerezi a futási időt, ahogyan az várható. Alacsonyabb lépésszám esetén ez nem igaz, hiszen a teljes futási idő egyre kevesebb része lesz a DEM számítás és arányaibban jelentősebbek az egyéb műveletek, mint a kimenetek írása, inicializálás és memóriakezelés. A teljes szimuláció során például 50-szer mentésre kerülnek a szemcsepozíciók és sebességek, azonban ezen kimeneti fájlok írását a CPU végzi, tehát ez a rész nem párhuzamosított.

A túl nagy időlépés ($\Delta t > 10^{-3}\,\mathrm{s}$) nem fizikai eredményekhez vezet az ülepítés során, ahogyan a 3.7b ábrán látható. Az ábrán megfigyelhető, hogyha az időlépés nagy, a szemcsefelhő kinetikus energiája (K) nem tart nullához, ahogyan az elvárható lenne. Ennek az oka, hogy az időlépés túl nagy és az ütközéseknél túl gyorsan alakulnak ki nagy átfedések, ahogyan az a 3.6. ábrán ismertetésre került. Az optimális időlépés megtalálása feladatfüggő, hiszen az függ a szemcsék méretétől, anyagától és a geometriától.



(a) A futási idő az időlépések számának N_{step} függvényében.

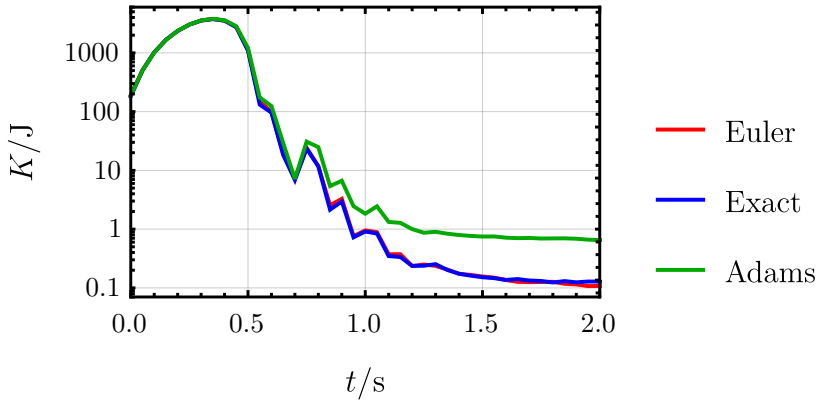


(b) A kinetikus energia logaritmikus diagrammon a szimuláció során különböző időlépések mellett.

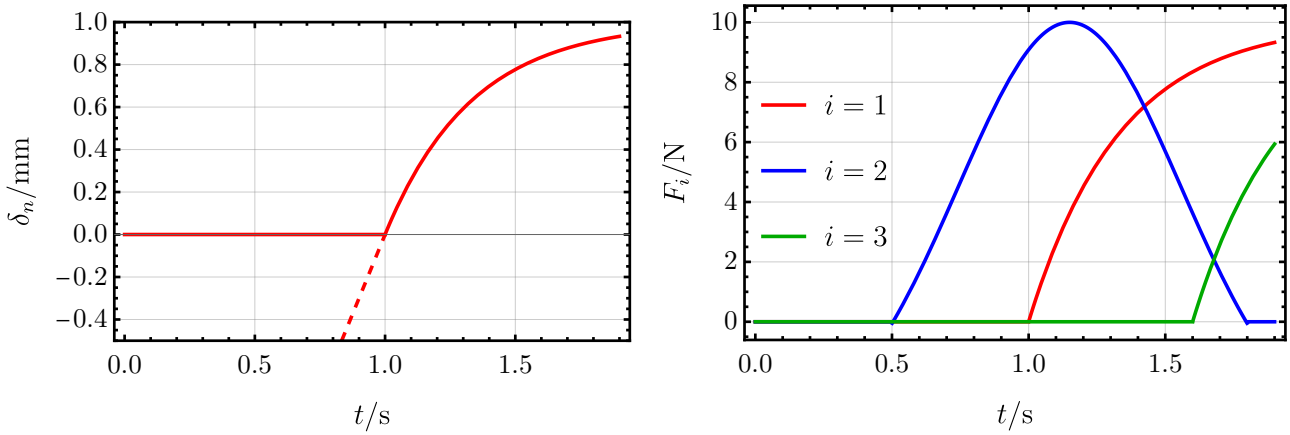
3.7. ábra. A #3 ülepítés különböző időlépések mellett.

A $\Delta t = 1 \cdot 10^{-3}\,\mathrm{s}$ időlépés mellett kipróbálásra került minden implementált időlépési séma, a futási időben nem mutatkozik különbség, hiszen az időlépés számításigénye elhanyagolható a kapcsolatkeresés és erőszámításhoz képest. A 3.8. ábrán látható a kinetikus energia K a szimulációk során. Az elsőrendű Euler-módszer esetén a szemcsefelhő kinetikus energiája közel 0.1 J-ra csökken. Az Exact-módszer az Euler-módszert egészíti ki a másodrendű tag $a \cdot \Delta t^2/2$ figyelembe vételével a pozíciószámítás során, ahogyan a 2.45. egyenletben bemutatásra került. A kinetikus energia ebben az Exact-módszer használata során is szinte hasonlóan csökken. A másodrendű Adams prediktor-korrektor módszer használata során viszont egy nagyságrenddel

több kinetikus energia marad vissza, habár a módszer rendje magasabb, ez itt mégsem jelent nagyobb pontosságot.



3.8. ábra. A kinetikus energia változása ugyanazon szimulációban $\Delta t = 1 \cdot 10^{-3}\,\mathrm{s}$, de különböző időléptetési módszerek alkalmazása mellett.



(a) A δ_n változása egy ütközés során

(b) Egy szemcsére ható erők több ütközésből

3.9. ábra. Magyarázat a nem folytonosságra a szemcseérintkezés során.

A differenciálegyenletek numerikus módszereiből ismert tény, hogy azok konvergenciarendje csak megfelelően folytonos esetekben garantált [39]. A szemcsék közötti ütközések elején azonban az erő változása nem folytonos, például a normálerő arányos a normálátfedéssel, azaz

$$\boldsymbol{F}_{ne} \propto \delta_n \tag{3.5}$$

ahol $\delta_n = d - (R_s + R_i)$, amely bár látszólag folytonos azonban csak akkor kerül értelmezésre, ha $\delta_n > 0$. A 3.9a ábrán látható a tangenciális átfedés egy esetben ahol két szemcse $t = 1\,\mathrm{s}$-től érintkezik, megfigyelhető hogy mivel $\delta_n < 0$ esetén a függvényérték nulla (nem történik ütközés), ezért $t = 1\,\mathrm{s}$-nél a derivált nem folytonos. Amennyiben a numerikus megoldás egy időlépést tesz például $t = 0.95\,\mathrm{s}$-tól $\Delta t = 0.1\,\mathrm{s}$ lépésközzel, akkor az időlépés során a függvény deriváltjába szakadás következik be. Ilyen esetekben a konvergencia elsőrendű, azaz a hiba Δt-vel arányos, hiszen nem teljesül a folytonossági feltétel [40]. A probléma megoldása az érintkezések időpontjának a pontos detektálása, és a mozgásegyenlet numerikus megoldása a folytonos szakaszokon, az időlépés megfelelő módosításával. Azonban ahogyan a 3.9b ábra mutatja, kapcsolatok folyamatosan jönnek létre és szünnek meg a szemcsék között, és már csak egy szemcse számításához

is rengeteg nem folytonos pont megtalálása szükséges. A DEM-ben azonban több ezer szemcse található, így minden ütközés pontos detektálása szinte lehetetlen. Éppen ezért nem kerülnek magasabb rendű numerikus módszerek alkalmazásra, hiszen a konvergenciarend megtartásának a feltétele az ütközések pontos detektálása, amely ekkora szemcseszám mellett nem megvalósítható. A szakirodalomban jellemzően az Euler-módszer kerül alkalmazásra [41, 42], azonban a választást jellemzően nem indokolják. A dolgozat további részében az elsőrendű Euler-módszer kerül alkalmazásra.

3.6. Profilozás

A profilozás lényege a szűk keresztmetszet – azaz a program legerőforrásigényesebb részének – megtalálása, alacsony szintű számlálók adatai alapján, mint például a memória és számítási műveletek száma vagy az adott utasítás gyakorisága. A modern NVIDIA GPU-kon az NVIDIA NSight Compute alkalmazás használható a programok teljeskörű profilozására.

A 3.2. táblázat mutatja a profilozás fontosabb eredményeit 3 különböző szemcseszám mellett, megfigyelhető, hogy a futási idő közel lineárisan nő. A szimuláció beállításai a 6.2. táblázat #3 oszlopában láthatók, a profilozás során végig az CLL kapcsolatkeresési eljárás került alkalmazásra és mindössze 100 időlépés lett végrehajtva. A 3.2. táblázatból látható, hogy az elkészült diszkrételemes kód alapvetően a GPU memóriáját jobban használja, mint a számítási erőforrásait, a GPU-s irodalom ezt *memórialimitált* problémának nevezi [43]. A memóriahasználat a 3.10. ábrán látható, megfigyelhető, hogy a globális memória elérése az L1 és L2 Cache gyorsítótáron keresztül valósul meg. A GPU programozásban amikor globális memória (felső zöld téglalap az ábrán) műveletről beszélünk az nem feltétlenül a GPU memória (device memory, jobb oldalt középső téglalap) olvasását és írását jelenti, előfordulhat, hogy a keresett adat rendelkezésre áll az L1 vagy L2 cache gyorsítótárban. A profilozás eredménye alapján az L1 gyorsítótárban az esetek 12.5%-ban található meg a keresett adat, az L2 gyorsítótárban pedig 70.4%-ban. Összességében megállapítható a 3.10. ábráról, hogy a GPU memória kihasználtsága rendkívül magas.

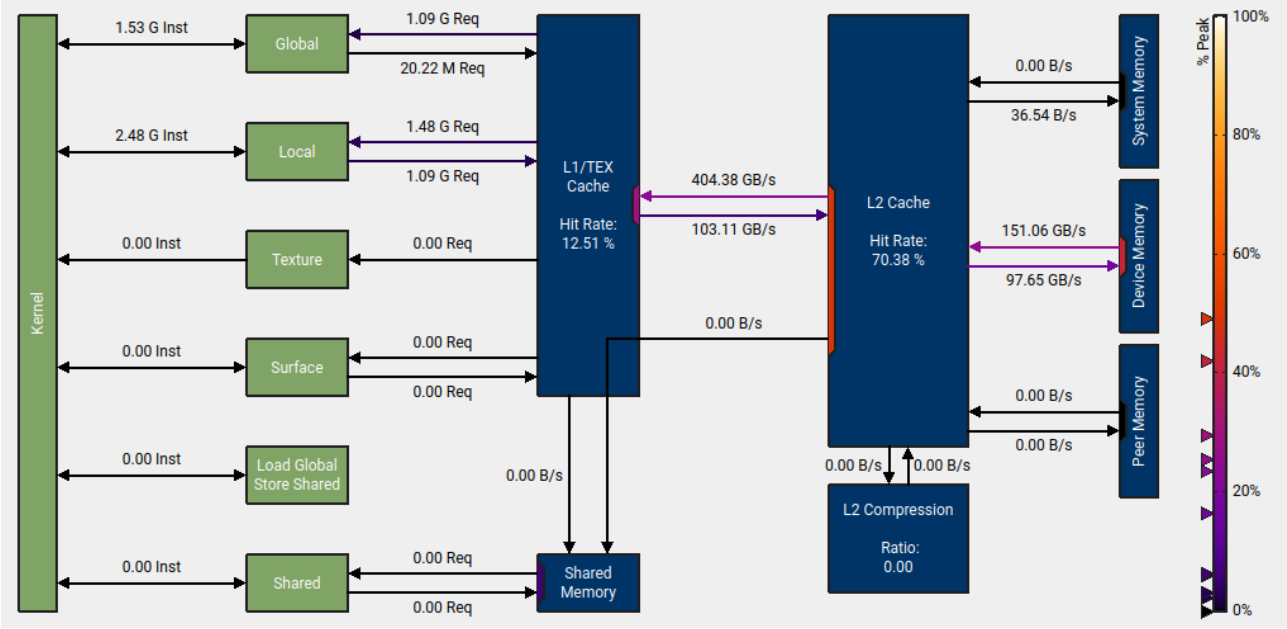
3.2. táblázat. A szimulációk beállításai

Szemcseszám	$2^{13} = 8192$	$2^{14} = 16384$	$2^{15} = 32768$
Futási idő	490 ms	869 ms	1846 ms
Számítási erőforrás kihasználtsága	13.30%	22.03%	27.03%
Memóriasávszélesség kihasználtsága	27.65%	40.92%	49.54 %
Lebegőpontos műveletek	$241.0 \cdot 10^6$	$741.6 \cdot 10^6$	$2511.1 \cdot 10^6$
Utasításszám/ciklus	0.22	0.32	0.36
L1 Cache találati ráta	20.68 %	15.90 %	12.51 %
L2 Cache találati ráta	76.06 %	77.28 %	70.38 %
Nincs várakozó warp	94.09 %	91.46 % %	90.76 %
Warp elakadás: memóriafüggés (scoreboard)	15.79	25.88	57.21
Warp elakadás: akadály (barrier)	6.55	7.23	8.95
Warp elakadás: várakozás (wait)	2.48	2.60	2.67
Warp elakadás: elágazás kezelés (branch)	1.88	1.78	1.13

A 3.2. táblázatból továbbá megállapítható, hogy a szemcseszám növelésével a memória és számítási kihasználtság is nő, ebből az is következne, hogy a futási idő nem duplázódik a szemcseszám duplázásával. Az adatokat tovább elemezve, megállapítható, hogy a lebegőpontos

műveletek száma a szemcseszám duplázódásával közel háromszorosára emelkedik, aminek oka feltehetően, hogy a szemcsék közötti kapcsolatok száma nem lineárisan nő a szemcseszámmal az ülepítéses szimuláció elején. A szemcseméretet tizedére csökkentve 8192 szemcse esetén a műveletek száma $73.2 \cdot 10^6$, 16384 szemcse esetén pedig a műveletek száma $177.9 \cdot 10^6$, amely hatalmas visszaesés a műveletek számában a 3.2. táblázatban látottakhoz képest, pedig csak a szemcseméret változott, de ezáltal kevesebb kapcsolat van a szemcsék között az ülepítés kezdetén.

A GPU relatív alacsony erőforrás kihasználtságát az alacsony utasításszám/ciklus szám okozza, ideális esetben minden egyes warp egy műveletet végez el egy ciklus alatt a teljes program során, ilyenkor az utasítás/ciklus=1. A GPUDEM mindösssze 0.22-es utasításszám/ciklust ér el alacsony szemcseszám esetén, amely a szemcseszám növelesével némiképp javul egészen 0.36-ra. Az alacsony utasításszám/ciklus okozója, hogy nincsen rendelkezésre álló warp. Annak ellenére, hogy van számítási kapacitás, ha nincsen rendelkezésre álló warp, akkor a ciklusban nem lesz új warp elindítva, ezen ciklusok aránya több mint 90% a kódban. A probléma okát a warpok elakadási okaiból állapíthatjuk meg. Egy warp átlagosan 15.79 ciklust vár a globális memóriából származó adatokra kicsi szemcseszám esetén és a legnagyobb szemcseszám esetén ez már 57.21 ciklus. A magas várakozási idő okozója, hogy több 10000 szemcse minden adata már nem fér be az L1 cache gyorsítótárba, mint alacsony szemcseszám esetén, ezért az adatokat a lényegesen lassabb elérésű globális memóriából kell betölteni. Mint említésre került, a szinkronizációs pontoknál minden szál megvárja egymást, ez azonban azzal jár, hogy a warpoknak is várakozniuk kell. A ciklusok száma amit a warpok átlagosan fix cikluszámú utasítás eredményére (például lebegőpontos összeadás, szorzás, osztás) várnak a várakozás sorban látható, ez szinte elhanyagolható a memóriafüggésből adódó várakozásokhoz képest.



3.10. ábra. A memóriahasználat 31768 szemcse használata esetén a #3 ülepítés szimulációban. A jobb oldali színskála a különböző memória elemek és azok közötti sávszélesség kihasznaltságát mutatja. A zöld blokkok a CUDA által definiált logikai elemek, mint a globális (Global), lokális (Local), textúra (Texture) és osztott (Shared) memória utasítások. A kék blokkok a GPU valós fizikai elemei, mint a gyorsítótárak (L1, L2 Cache), az osztott memória (Shared Memory), a GPU memória (Device Memory).

További értékes információt ad a programról, hogy a különböző programrészek a warpok elakadásának és a műveletek számának mekkora hányadát adják. A 3.3. táblázatból megállapítható, hogy a warp elakadások több mint fele a kapcsolatkeresés során történik, aminek az oka, a várakozás az adatokra a globális memóriából. A másik lényeges rész és a szinkronizációs pontok, ahol a szálak megvárják egymást, hogy minden szál az új adatokkal dolgozhasson a következő időlépésben. Ez nyilvánvalóan a warpok elakadását okozza.

3.3. táblázat. A warpok elakadásánal aránya és a műveletek száma a különböző programrészekben

Programrész	Warp elakadások aránya	Műveletek számának aránya
Megoldó struktúra	2.33%	2.83%
Geometria	0.28%	1.86%
Kapcsolatkeresés	57.64%	67.66%
Távolságszámítás	4.78%	7.74%
Erőszámítás	1.53%	3.74%
Gyorsulás számítás	0.08%	0.06%
Időlépés	0.00%	0.22%
Szinkronizáció	19.94%	11.43%

A program profilozása számos fejlesztési lehetőségre rámutat. A potenciális fejlesztési lehetőségek a következők:

1. Az L1 és L2 gyorsítótár még hatékonyabb kihasználása a memória megfelelő rendezésével. A kapcsolatkeresés során bármely szemcse bármely másikkal kapcsolatban állhat, így teljesen tetszőleges szálak teljes tetszőleges memóriahelyekről olvasnak adatokat. A gyorsítótárazás kiszámítható adathasználat mellett működik jól, például amikor egy szál mindig ugyanonnan olvas be adatot, azonban ez nem teljesül. Ennek a javítása egy blokkalapú programstruktúrával valósítható meg, ahol a közeli szemcsék egy blokkban és warpban helyezkednek el, amely azonban a GPUDEM szinte teljes újraírását igényelné. Mint az 1.3.2. fejezetben említésre került a *per-block* felépítés lehetséges, azonban numerikus szimulációk során ritkán használják.

2. Kapcsolatok frissítésének az optimalizációja. Amennyiben a program talál két érintkező szemcsét, akkor meg kell nézni, hogy volt-e érintkezés azzal a szemcsével az előző lépésben, és ha volt a tangenciális átfedést az előző tangenciális átfedés alapján kell meghatározni. Ehhez a program a 6. programrészletben látható módon végigmegy minden egyes korábbi kapcsolaton és ellenőrzi, hogy megegyezik-e a mostani index egy korábbi indexxel. Ez az ellenőrzés a warp elakadások több mint 20%-át okozza. Azonban a szemcsék pozíciója nem változik nagyon egy időlépés során és a legtöbb esetben a kapcsolatok sorszáma sem változik. A 7. programrészletben már feltételezve van először, hogy a lépés során a kapcsolat sorszáma (`contacts.count`) nem változott meg. A sorszám változásra csak akkor kerül sor, ha egy kapcsolat megszűnik vagy egy új létrejön, az így megváltoztatott kód már csak az összes warp elakadás kevesebb mint 8%-át okozza. Továbbá a 12. sorban beszúrásra került egy `break` utasítás, ami a for ciklust megtöri, hogyha a korábbi érintkezés már meg lett találva. Ezzel a kis módosítással a program futási ideje 869 ms-ről 521 ms-re csökkent.

6. Kódrészlet. A kapcsolatkeresés régi verziója.

```
int i=...; //a szemcse indexe amivel az adott szál érintkezik
bool wasInContact = false;
for(int j = 0; j < MaxContactNumber; j++)
{
    if(i == contacts.tid_last[j]) //warp elakadás: 20.83%
    {
        wasInContact = true;
    }
}
if(wasInContact) ...
```

7. Kódrészlet. A kapcsolatkeresés új verziója.

```
int i=...; //a szemcse indexe amivel az adott szál érintkezik
bool wasInContact = false;
if(i == contacts.tid_last[contacts.count]) //warp elakadás: 5.92%
{
    wasInContact = true;
}
else for(int j = 0; j < MaxContactNumber; j++)
{
    if(i == contacts.tid_last[j]) //warp elakadás: 1.73%
    {
        wasInContact = true;
        break;
    }
}
if(wasInContact) ...
```

3. Számítások számának a csökkentése a kapcsolatkeresés során. A szemcse-szemcse kapcsolatok során eddig mindig kiszámításra került m^* és R^* ami a műveletek 5%-át adja, azonban erre csak akkor van szükség, ha még nem került korábbi lépés során számításra. Ezzel a javítással a kód futási ideje a korábbi 512 ms-ről 475 ms-re csökkent.

4. A távolságszámítás optimalizációja. A 2.6. egyenlőtlenség szerint két szemcse érintkezik, ha sugaraik összege nagyobb mint a középpontjuk távolsága. Az egyenlőtlenség négyzetre emelve is igaz, tehát

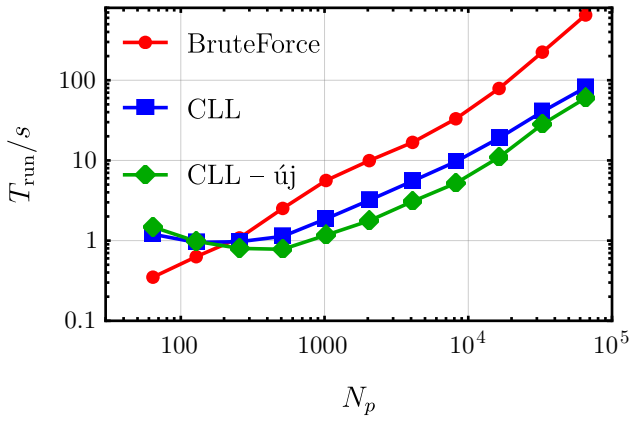
$$(R_i + R_j)^2 > d^2, \tag{3.6}$$

ahol d^2 számítása nem igényel gyökvonást. A gyökvonást elég elvégezni, ha a két szemcse biztosan érintkezik. Ez a módosítás nem okozza a futási idő érezhető változását.

5. Szinkronizáció optimalizálása a felesleges szinkronizációs pontok kiiktatása által. Megállapításra került, hogy egy szinkronizációs pontra nincsen szükség a kódban, ennek az eltávolítása azonban nem okozza a program futási idejének az érezhető változását.

A változásokat végrehajtva a programon a skálázás újból vizsgálatra került. Az új kód (CLL–új) skálázódása a 3.11. ábrán látható a szemcseszám N_P függvényében. A fentebb leírt módosítások a GPUDEM futási idejét megfelezték a $N_P = 10^3 \ldots 10^4$ szemcseszám tartományban. Alacsonyabb szemcseszám esetén nem mutatkozik lényeges eltérés, magasabb szemcseszám

esetén pedig a futási idő a korábbi 75%-a. Ezen esettanulmány rávilágít a profilozás fontosságára a nagy-teljesítményű alkalmazások során, jelen esetben a szűk keresztmetszet – azaz a kapcsolatkeresés – optimalizációjával a futási időt sikerült megfelezni.



3.11. ábra. A futási idő (T_{run}) a szemcseszám (N_P) függvényében különböző kapcsolatkeresési algoritmusok mellett.

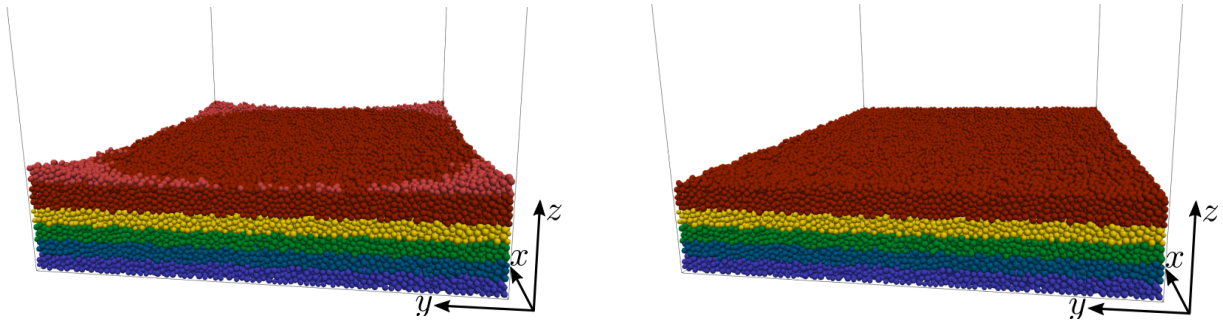
4. Alkalmazás – Talajművelés

Az előző fejezetben bemutatásra került a program működése. Az ülepítési feladatokon már látjuk, hogy a program rendkívül gyors és megfelelő eredményeket ad. Ez a fejezet a GPUDEM néhány lehetséges alkalmazását mutatja be, amely során egy kultivátoros talajművelés kerül szimulációra. A kultivátoros talajművelés célja a talaj fellazítása, ami növeli a vízfelvevő képességet és lehetővé teszi a gyökerek könnyebb behatolását a talajba. A következő példa célja bemutatni, hogy a GPUDEM alkalmazható gyakorlati problémák vizsgálatára is, mint például a kultivátorra ható erők meghatározására a talajművelés során.

4.1. Beállítások

A szimuláció során egy nedves homok talajban került végighúzásra egy kultivátor. A nedves homok talaj DEM paraméterei és a szimuláció további beállításai a 6.2. táblázat #4 oszlopában kerültek összefoglalásra [7]. A szimulációs tartomány hossza $l = 1\,\mathrm{m}$ és szélessége $w = 0.7\,\mathrm{m}$, a talajszemcsék mérete $R = 4.8\,\mathrm{mm} \pm 0.8\,\mathrm{mm}$. A talajszemcsékből 14 cm vastag talajréteg kerül kialakításra és a kultivátor 10 cm mélyen kerül elhelyezésre a talajban.

A szimuláció első lépése a talaj előkészítése, amely ülepítéssel valósul meg. Az ülepítés szimulációjára a 3.1. fejezet már kitért. Az ülepítés során $N_P = 147\,456$ szemcse került alkalmazásra, az ülepített szemcsék a 4.1a ábrán láthatók. A különböző rétegek, amelyek azonos DEM paraméterekkel rendelkeznek, magasság szerint más-más színnel kerültek jelölésre az ábrán, továbbá megfigyelhető, hogy a szemcsék a tartomány sarkainál vastagabban rétegződnek. Ez nem kívánatos jelenség, így minden olyan szemcse eltávolításra került amely középpontja 14 cm felett van. A 4.1b ábrán látható a szemcsehalmaz a felső réteg eltávolítása után, aminek a felülete már teljesen sima. Az felső szemcsék eltávolítása után $N_P = 146\,180$ szemcse maradt vissza.

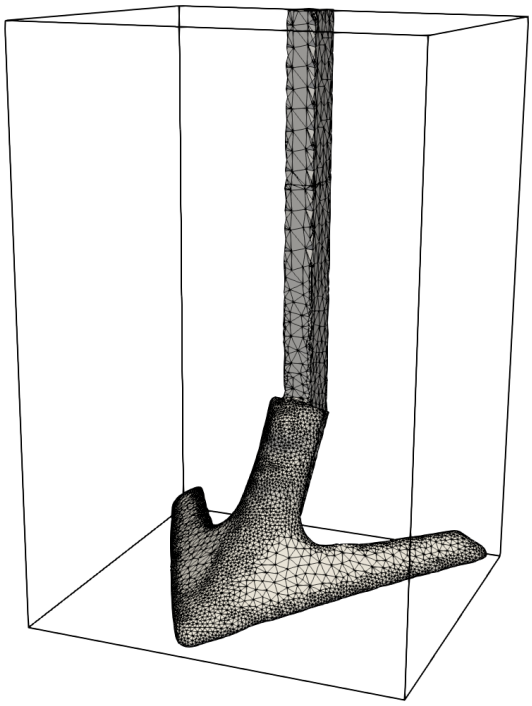


(a) A szemcsék az ülepítés után

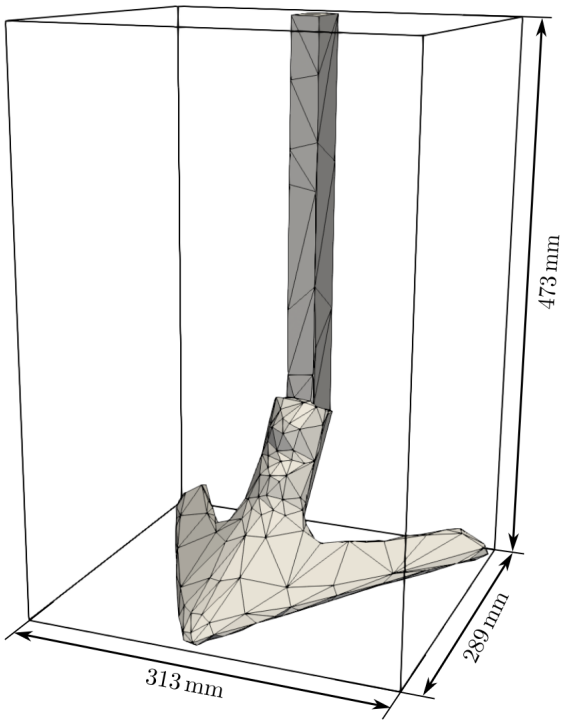
(b) A legfelső szemcsék eltávolítása után

4.1. ábra. $N_P = 147\,456$ szemcse ülepítése és kiegyenesítése a legfelső szemcsék eltávolítása által.

A második lépés a kultivátor geometria hozzáadása. A szimulált kultivátor modellel korábban laboratóriumi méréseket is végeztek, aminek során készült el a 3D szkennelt változtat. Azért esett erre a modellre a választás, mivel így lehetőség nyílik majd a mérésekkel való összevetésre. Ez a 3D szkennelésből származó modell azonban több mint 14 304 háromszöget tartalmaz, viszont ez a GPUDEM jelenlegi implementációjában nem alkalmazható, ezért a geometria egyszerűsítése szükséges. Az eredeti modell mérete a Blender szoftverben került csökkentésre, az új csökkentett modell mindössze 500 háromszöget tartalmaz, az eredeti és a csökkentett modell összehasonlítása a 4.2. ábrán látható.

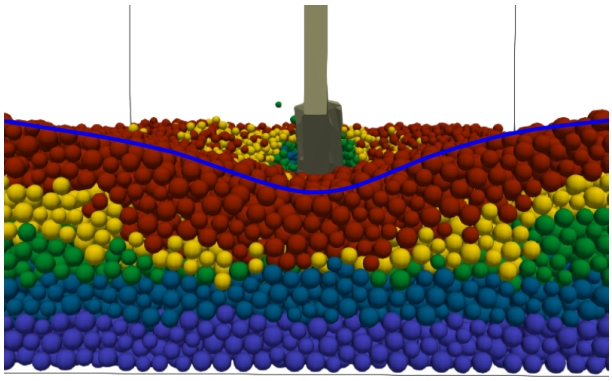


(a) Az eredeti 14 304 háromszöget tartalmazó modell

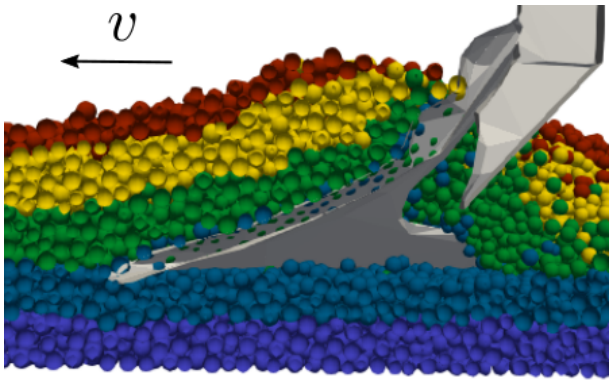


(b) A csökkentett 500 háromszöget tartalmazó modell és befoglaló méretei

4.2. ábra. Az eredeti és a csökkentett háromszögszámú kultivátor modell



(a) A talajművelés után kialakuló felszínprofil. A felszínprofil kék színnel került ábrázolásra.



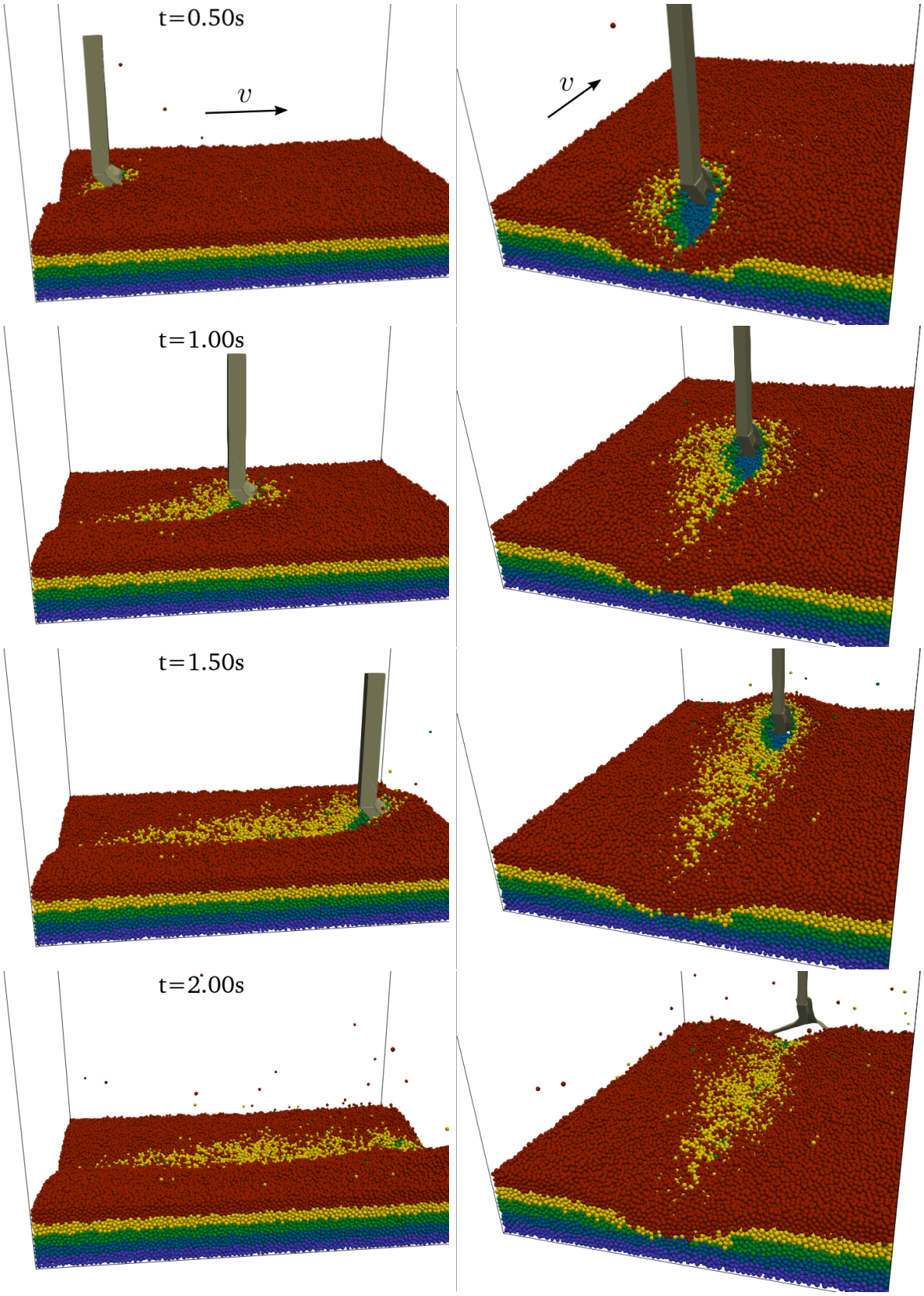
(b) A kultivátor a talajban a talajművelés során. A mozgás irányát v adja meg.

4.3. ábra. A kialakuló felszínprofil és a szemcsék a kultivátor környékén

4.2. Eredmények

Több különböző szimuláció készült más-más munkasebességekkel. Az első szimulációban a kultivátor $v = 0.7\,\mathrm{m/s}$ sebességgel mozog. A szimuláció néhány pillanatfelvétele látható a 4.4. ábrán, megfigyelhető, ahogy a kultivátor fellazítja és megmozgatja a talaj alsóbb rétegeit, miközben a felső vörös réteg szinte érintetlenül marad. A kultivátor továbba egy V alakú barázdát hagy

maga után a talajban ahogyan a 4.3a ábrán látható, a barázda profilja kék színnel ábrázolásra került. Továbbá, az ábrán jól látható, hogy a felső vörös réteg egyben maradt a kultivátor elhaladása után is, mindössze az alsóbb rétegek keverednek meg.



4.4. ábra. Kultivátoros talajsöprés $v = 0.7\,\mathrm{m/s}$ mellett, két különböző szögből nézve

A szimulációban megfigyelhető, hogy a kultivátor előtt a szemcsék megemelkednek majd mögötte visszazuhannak, azonban a folyamat közben az alsóbb rétegek átrendeződnek, ahogy látható a 4.3b ábrán. A szimuláció során meghatározásra kerültek a kultivátorra ható erők. A kultivátorra ható erő számítátható minden egyes szemcse-kultivátor kölcsönhatásból származó erő összegzésével, amelynek az implementálása a programban nem bonyolult. A szemcsékre ható erők egyébként is számításra kerülnek, mindössze egy új globális változót kell létrehozni, amelyhez hozzáadásra kerül minden egyes szemcse-szerszám között ható erő, majd minden időlépés végén nullázódik.

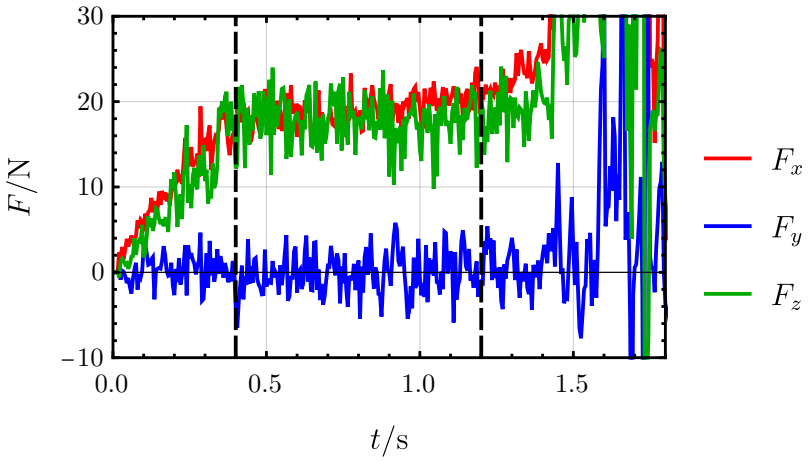
A kultivátorra ható x, y, z irányba ható erők a 4.5. ábrán láthatók. A kultivátor az x irányba halad, ennek megfelelő ebben az irányban egy közel $F_x \approx 20\,\mathrm{N}$ vonóerőt kell kifejteni a mozgatáshoz. Az y síkra a kultivátor szimmetrikus, ennek ellenére a szemcsék inhomogenitásából adódan egy kisebb nulla körül oszcilláló erő jelentkezik. A z irányban szintén egy nagyobb erő jelentkezik, a kultivátort a szemcsék lefelé nyomják, ahogyan a 4.3b ábrán látható. Ebből tehát következik, hogy a z pozíció konstanson tartása érdekében, a pozitív z irányba erőt kell kifejteni. Az ábrán három intervallumot különböztethetünk meg:

1. Kezdetben ($t < 0.4\,\mathrm{s}$) a vonóerő növekszik, ahogy a kultivátor behatol a szemcsehalmazba.

2. A középső szakaszon a vonóerő közel konstans, de a talaj inhomogenitásából adódó fluktációk megfigyelhetők a konstans érték körül.

3. A szimuláció végén ($t > 1.2\,\mathrm{s}$), ahogyan a kultivátor elhagyja a tartományt, a szemcsék a tartomány széle és a kultivátor között összenyomásra kerülnek, ami egy nem valós erőnövekedést eredményez. A 4.4. ábrán $t = 2\,\mathrm{s}$-nél megfigyelhetők a tartományban repkedő szemcsék, amelyet szintén az erős összenyomás okoz, ahogyan a kultivátor elhagyja a tartományt.

A második intervallumban a vonóerő egy konstans érték között fluktuál, ami megfelel a szakirodalomnak [8]. Az első és harmadik intervallumban tapasztalt jelenségek nem valósak, ezek a szimuláció speciális beállításából adódnak.

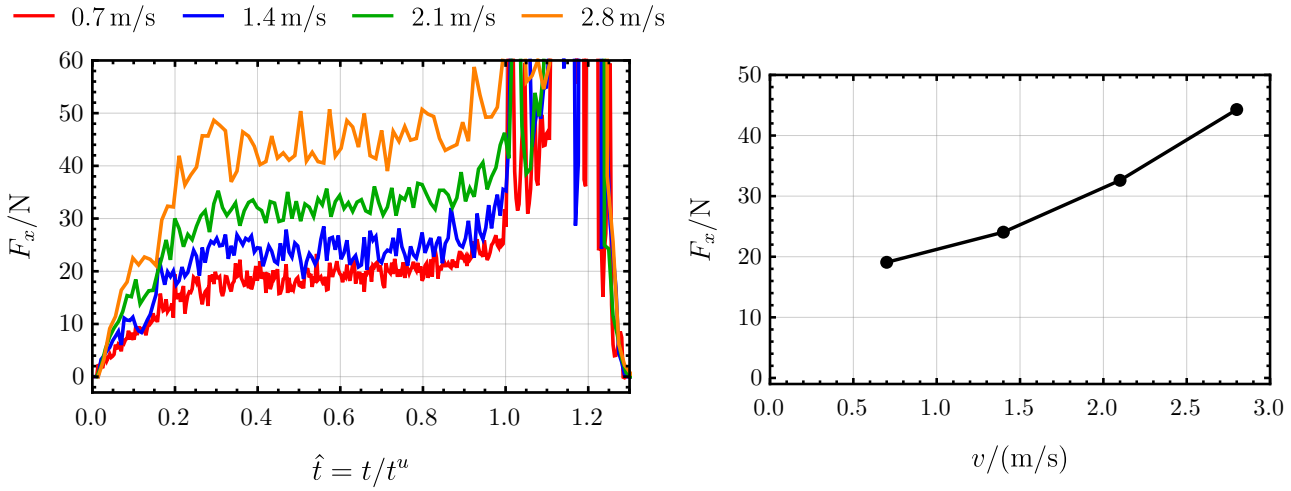


4.5. ábra. A kultivátorra ható erők a talajlazítás során

A következő szimulációkban a kultivátor sebessége növelésre került. Az összehasonlíthatóság érdekében, az idő dimenziótlanításra kerül, a következő módon

$$\hat{t} = \frac{t}{t^u} \quad \text{és} \quad t^u = \frac{l}{v}, \tag{4.1}$$

ahol $\hat{t}$ a dimenziótlan idő, t^u az időtartam ami alatt a kultivátor végighalad a tartományon, l a tartomány hossza és v a kultivátor sebessége. A 4.6a ábrán látható a vonóerő a dimenziótlan idő $\hat{t}$ függvényében. A szakirodalomnak megfelelően nagyobb sebességeknél nagyobb vonóerő szükséges a szimulációban is [8]. Mindegyik szimuláció során megfigyelhető a korábban ismertetett három különböző intervallum. A szimulációk elején a vonóerő nő, majd a vonóerő egy konstans érték körül oszcillál, a szimuláció végén ahogy a kultivátor elhagyja a szimulációs tartományt a vonóerő megemelkedik. A fizikailag valós tartomány nagyjából $0.3 < \hat{t} < 0.8$ között található, ez az a rész, amikor a kultivátor már teljesen a szimulációs tartományban van és még nem ért közel a tartomány végéhez, tehát a szemcsék nem kezdtek el összenyomódni. A $0.3 < \hat{t} < 0.8$ tartományban meghatározásra került az átlagos vonóerő, ami a sebesség függvényében a 4.6a ábrán látható.



(a) A vonóerő a dimenziótlan idő függvényében különböző sebességek mellett

(b) A vonóerő a kultivátor sebességének a függvényében

4.6. ábra. A vonóerő alakulása különböző sebességek mellett

4.3. Paraméterek hatása

A mérésben $v \approx 0.7\,\mathrm{m/s}$ sebesség mellett $F_x \approx 310\,\mathrm{N}$ vonóerőt rögzítettek homokos talajban. A pontos értékek és a talaj paraméterei a 6.1. táblázatban láthatók. A szimuláció nem modellezi tökéletesen a talajt, hiszen a talajszemcsék a valóságban lényegesen kisebbek mint az itt beállított 4.8 mm. A talajparaméterek változtatásával azonban lehetséges olyan beállítások megtalálása, amely visszadja a kísérletben mért erőt. A következőkben a talajszemcsék sűrűségének (ρ), mechanikai paramétereinek (E, G), ütközési tényezőjének (e) és a súrlódási tényezőinek a hatása kerül elemzésre. A cél, olyan paraméterek megtalálása, amellyel visszakapjuk a mért $F_x \approx 310\,\mathrm{N}$ erőt.

Az eddigi több mint 148 ezer elemet tartalmazó szimuláció közel fél óráig fut. A kalibrációhoz olyan szimulációt kell készíteni, ahol néhány perc alatt már meghatározható a vonóerő, ehhez a szemcseszám csökkentésre kerül. A cél, hogy az egyszerűsítés ellenére, továbbra is elkülöníthető legyen egy közel állandósult szakasz. A paraméterek változtatása során az ülepítést minden paraméterkombinációra részben újra kell csinálni, például E csökkentésével a szemcsék jobban összenyomhatók, ráadásul ezáltal a talajvastagság is változik. Az ülepítés után következhet a kultivátor végighúzása. A következő lépésekkel érhető el, hogy bármely paraméterkombináció esetén ugyanazon kezdeti talajvastagság legyen létrehozva, így ugyanolyan

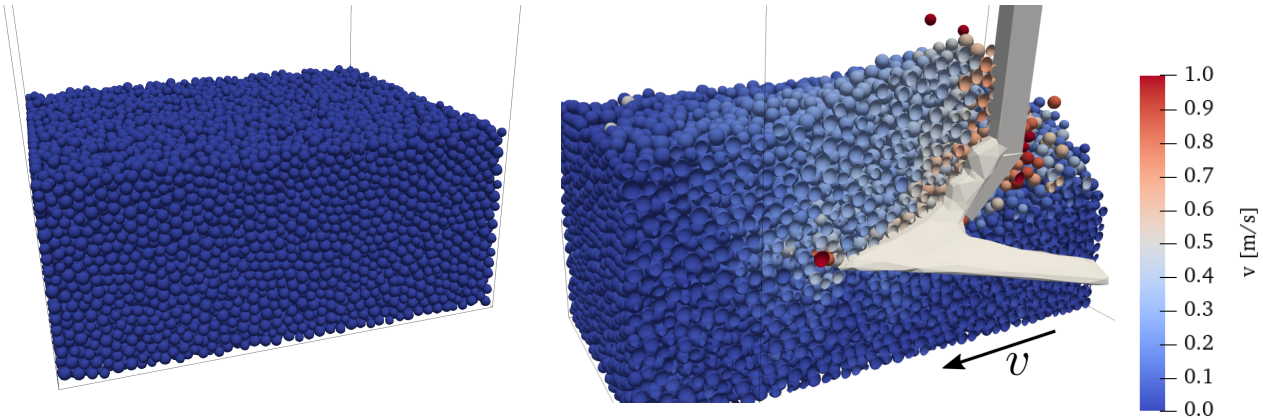
feltételek mellett lehessen végighúzni a kultivátort:

1. Kezdetben 38912 darab, $R = 8.5\,\text{mm} \pm 1.5\,\text{mm}$ sugarú szemcse kerül ülepítésre egy $0.7 \times 0.6\,\text{m}$-es tartományban, amely szemcsék a paraméterek függvényében nagyjából $30-40\,\text{cm}$ vastagon rétegződnek.

2. A szemcsékből eltávolításra kerül minden aminek a középpontja 30 cm felett van, így minden esetben egy hasonló 30 cm vastag talajréteg kerül létrehozásra. Az ülepített szemcsék a 4.7a ábrán láthatók.

3. Az utolsó lépés a kultivátor végighúzása $v = 0.7\,\text{m/s}$-el és az erők összegzése. A vonóerő a $0.4\,\text{s} < t < 0.8\,\text{s}$ tartományban kerül kiátlagolásra, így a tartomány szélei nem befolyásolják a szimuláció eredményeit. A kultivátor végighúzásáról egy pillanatkép látható a 4.7b ábrán.

A felsorolt lépések automizációra kerültek és a teljes futási idő mindössze $10-30\,\text{s}$. A további paraméterek a 6.2. táblázat #5 oszlopában láthatók. Ez az elkészített automatizált kód már használható a kultivátoros talajművelés paraméterfüggő vizsgálatára.



(a) Az ülepített részecsék a felső réteg levágása után

(b) A kultivátor a talajban a szimuláció során

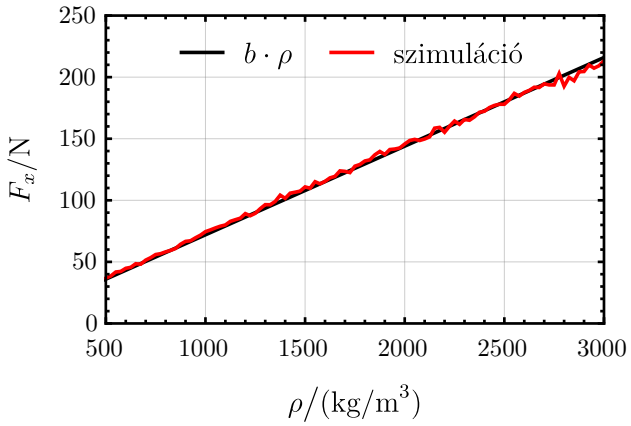
4.7. ábra. A paraméterérzékenységi vizsgálathoz létrehozott szemcsehalmaz az ülepítés után és a talajművelés során. A színezés a sebességvektor nagyságával arányos a skála szerint.

4.3.1. Szemcsesűrűség hatása

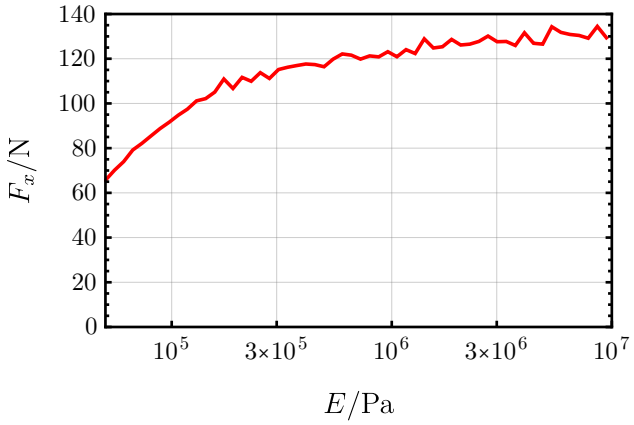
Elsőként a szemcsesűrűség hatása került vizsgálatra, a szemcserűség a $\rho = 500\,\text{kg/m}^3 \ldots 3000\,\text{kg/m}^3$ tartományban került változtatásra, összesen 100 szimuláció készült egyenletes beosztással, a további paraméterek a 6.2. táblázat #5 oszlopában láthatók. A szimulációkban a vonóerő lineáris növekedése figyelhető meg a sűrűség növekedésével, ahogyan a 4.8a. ábrán látható. A szimuláció eredményeire egy egyenes illeszthető a következő alakban,

$$F_x = b \cdot \rho, \quad \text{ahol} \quad b = 0.072\,\text{N}/(\text{kg/m}^3). \tag{4.2}$$

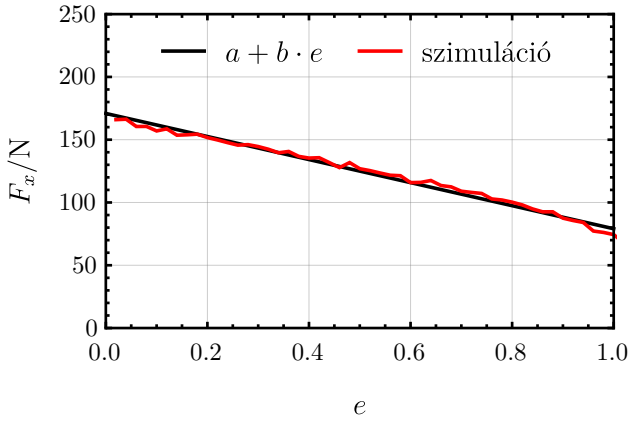
A szakirodalom szerint a talaj sűrűsége lineárisan korrelál a szükséges vonóerővel [44], és ezt a GPUDEM szimuláció is megerősíti. A sűrűségnövekedés várhatóan tényleg a vonóerő növekedését okozza, hiszen ezáltal a talajszemcsék nehezebbek, tehát nagyobb erőt kell kifejteni a megmozgatásukhoz.



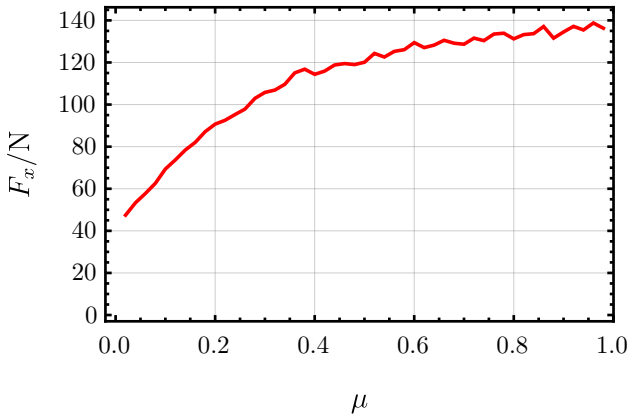
(a) A szemcsesűrűség hatása a vonóerőre



(b) A Young-modulus hatása a vonóerőre.



(c) Az ütközési tényező hatása a vonóerőre



(d) A csúszási súrlódási tényező hatása.

4.8. ábra. A vonóerő alakulása különböző anyagparaméterek mellett

4.3.2. Young-modulus hatása

Másodikként a Young-modulus került változtásra az $E = 5 \cdot 10^4\,\mathrm{Pa} \ldots 1 \cdot 10^7\,\mathrm{Pa}$ tartományban, a további paraméterek a 6.2. táblázat #5 oszlopában láthatók. Összesen 100 különböző szimuláció készült logaritmikus beosztással. A szimulációkban megfigyelhető a vonóerő növekedése a 4.8b ábra szerint, az ábrán az x tengely skálája logaritmikus. Látható, hogy ha a Young-modulus elég nagy ($E \approx 3 \cdot 10^6\,\mathrm{Pa}$), akkor az E értékének további növelése már nem igazán jelenti a vonóerő növekedését.

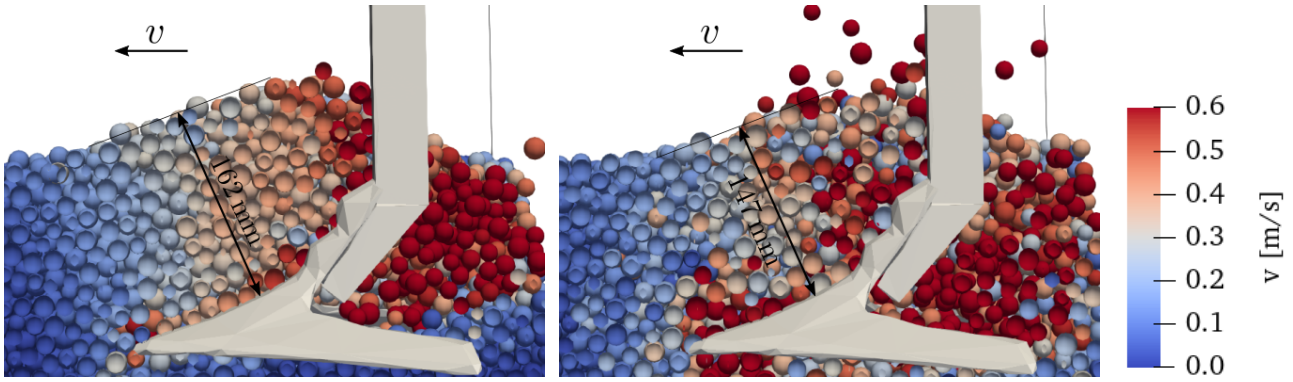
4.3.3. Ütközési tényező hatása

Az üközési tényező hatásának a vizsgálata során szintén 100-100 szimuláció került elvégzésre. Az ütközési tényező hatása a 4.8c ábrán látható, megfigyelhető a vonóerő lineáris csökkenése az ütközési tényező növekedésével. A szimuláció eredményeire egyenes illeszthető a következő alakban,

$$F_x = a + b \cdot e, \quad \text{ahol} \quad a = 170.9\,\mathrm{N} \quad \text{és} \quad b = -91.8N. \tag{4.3}$$

Ha az ütközési tényező kicsi, akkor a szemcsék jobban összeragadnak, a jobban összeragadt szemcsék között a súrlódás pedig nagyobb, ezáltal nagyobb erőkifejtés szükséges a kultivátor mozgatásához. Ahogyan a 4.9. ábrán látható, alacsony ütközési tényező esetén (például $e = 0$) lényegesen homogénabb tömörödési zóna jön létre, amely vastagabb és több szemcsét tartalmaz.

Nagyobb ütközési tényező esetén a kultivátor feletti talajréteg vastagsága lecsökken és egy kisebb tömörödési zóna jön létre, amely mozgatása kisebb erőt igényel, hiszen kevesebb szemcsét tartalmaz.



(a) $e = 0$, a kultivátor feletti talajréteg vastagsága 162 mm

(b) $e = 1$, a kultivátor feletti talajréteg vastagsága 147 mm

4.9. ábra. A kultivátor körüli részecskék sebessége $e = 0$ és $e = 1$ ütközési tényező esetén. A részecskék színezése a sebességvektor nagyságával arányos a színskála szerint. Az ábrákon jelölésre került a kultivátor feletti talajréteg vastagsága.

4.3.4. Súrlódási tényező hatása

A csúszási (μ) és tapadási (μ_0) súrlódási tényezőpár együtt került változtatásra úgy, hogy a csúszási súrlódási tényező megválasztása után a tapadási a csúszási súrlódási tényezőnél mindig 10%-kal nagyobbra lett beállítva, azaz azaz $\mu_0 = 1.1\mu$. A szimuláció eredményei a 4.8d ábrán láthatók, ahol megfigyelhető a vonóerő növekedése a súrlódási tényező növekedésével. A vonóerő növekedés egyszerűen magyarázható ebben az esetben, hiszen a súrlódás növelésével a szemcsék egyre nehezebben csúsznak el egymáson, tehát nagyobb erő kifejtése szükséges a megmozgatásukhoz.

4.4. Kalibráció evolúciós algoritmussal

4.4.1. Differenciál-evolúció

A 4.3. fejezetben bemutatásra került a különböző anyagparaméterek hatása a vonóerőre. A paraméterek állítgatásával, azaz kalibrációjával megtalálhatók azon beállítások, amelyek visszaadják a mérési eredményeket. A kalibráció során a paraméterek állítgatása általános esetben kézzel történik, szimulációkat egymás után futtatva és kiértékelve, és ezek alapján a paramétereket módosítva. Azonban ez viszonylag sok időt és emberi erőforrást igénylő folyamat. Ezért ehelyett a paraméterkalibráció automatizálásra került egy evolúciós algoritmussal, amely célja, a mérésben tapasztalható erő reprodukálása. A differenciál-evolúció került alkalmazásra, amely folytonos problémákra alkalmazható adott megszoritások mellett [45]. A megszoritások a kontrolváltozók értékeit limitálják, a 4.1. táblázatában láthatók a kontrolváltozók és azok határai. A differenciál evolúciós algoritmusban NP jelöli az egyedek számát (jelen esetben a kontrollváltozókból álló paraméterkombinációk számát), ezen egyedek kezdetben véletlenszerűen kerülnek generálásra a 4.1. táblázatban leírt határok betartásával. Minden $\mathbf{x}$ egyed a következő kontrolváltozókat (elemeket) tartalmazza,

$$\mathbf{x} = [\rho \quad E \quad \nu \quad e \quad \mu \quad \mu_r]^t. \tag{4.4}$$

A cél a paraméterek kalibrálása, hogy a mérésben tapasztalt $F_{\text{mérés}} = 310\,\text{N}$ erőt visszakapjuk. A fitnesz függvény ennek megfelelően

$$f(\mathbf{x}) = \left|F_x(\mathbf{x}) - F_{\text{mérés}}\right|, \tag{4.5}$$

ahol $F_x(\mathbf{x})$ a vonóerő a szimulációban a kontrolváltozók függvényében. A kezdeti egyedek generálása és a fitnesz függvény definiálása után az evolúció a következő módon zajlik minden generációban, minden egyeden:

1. Kiválaszásra kerül három másik, $\mathbf{x}$-től különböző egyed, $\mathbf{a}, \mathbf{b}$ és $\mathbf{c}$. Ezekből létrehozásra kerül egy $\mathbf{y}$ egyed,
$$\mathbf{y} = \mathbf{a} + F(\mathbf{b} - \mathbf{c}), \tag{4.6}$$
ahol F a differenciál-súlyozás.

2. Az új $\mathbf{y}$ és a régi $\mathbf{x}$ egyed keresztezésre kerül. Mégpedig minden elem CR valószínűséggel kerül lecserélésre $\mathbf{x}$-ben, de egy elem véletlenszerűen mindig lecserélésre kerül. Így jön létre egy új $\tilde{\mathbf{x}}$ elem.

3. Az új $\tilde{\mathbf{x}}$ egyed jobb mint a régi $\mathbf{x}$, amennyiben a fitnesz értéke kisebb, azaz
$$f(\tilde{\mathbf{x}}) \leq f(\mathbf{x}), \tag{4.7}$$
ahol f a fitnesz. Amennyiben a 4.7. egyenlőtlenség teljesül, akkor az $\mathbf{x}$ egyed lecserélésre kerül $\tilde{\mathbf{x}}$-re.

4. Az 1-3. pontok elvégzésre kerülnek minden egyeden, majd következik a következő generáció.

A paraméteroptimalizációhoz a választott egyedszám $NP = 60$, a differenciál-súly $F = 0.8$ és a keresztezési arány $CR = 0.5$. A differenciál-evolúcióhoz egy saját C++ implementáció készült, amely minden függvénykiértékeléshez, azaz erőmeghatározáshoz meghívja a GPUDEM könyvtárat. A beállított paraméterek mellett egy generációban 60 függvénykiértékelés szükséges, ennek megfelelően egy generáció futási ideje 20-25 perc.

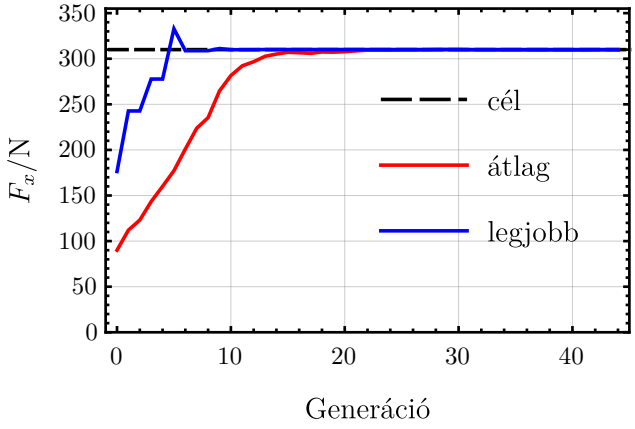
4.1. táblázat. A kontrolváltozók határai az evolúciós algoritmusban

	Kontrolváltozó	Min	Max	Kezdeti eloszlás
ρ	szemcsesűrűség	$500\,\text{kg/m}^3$	$2500\,\text{kg/m}^3$	lineáris
E	Young-modulus	$5 \cdot 10^4\,\text{Pa}$	$1 \cdot 10^7\,\text{Pa}$	logaritmikus
ν	Poisson-szám	0	0.5	lineáris
e	ütközési együttható	0.05	0.95	lineáris
μ	súrlódási együttható	0.05	0.70	lineáris
μ_r	gördülési együttható	0.01	0.30	lineáris

4.4.2. Eredmények

Az optimalizáció 44 generáción keresztül futott 16 órán keresztül és összesen 2700 függvénykiértékelés történt, tehát 2700-szor került végrehajtásra az ülepítés és a kultivátor végighúzása, több mint 30 000 szemcsével minden függvénykiértékelés során. A generációk előrehaladtával a

legjobb egyed és az egyedek átlaga is egyre közelebb kerül a kívánt célhoz, ahogyan a 4.10. ábrán látható. Az ábrán a szaggatott vonal jelöli a mérés során tapasztalt átlagos vonóerőt. A differenciál-evolúció nagyjából 20 generáció után megtalálja az optimum megoldásokat, azonban minden egyed más-más paraméterekkel rendelkezik, azaz az egyedek nem egy globális optimumba konvergálnak, hanem több lokális optimumot találnak meg. Az egyedek csoportosításával, azaz klaszterezésével megtalálhatóak a lokális optimumok [46], a klaszterezés a Wolfram Mathematica beépített `FindClusters` függvényével valósult meg a K-Means algoritmussal.



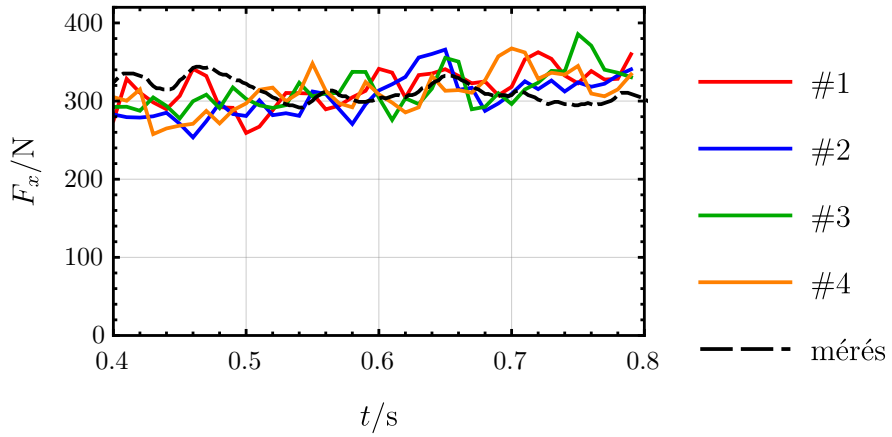
4.10. ábra. A legjobb és az átlagos vonóerő a populációban az evolúciós algoritmus futása során.

4.2. táblázat. A legtöbb egyedet tartalmazó klaszterek átlagértékei és a megadott paraméterek melletti átlagos vonóerő és annak szórása.

	Kontrolváltozó	#1	#2	#3	#4
ρ	szemcsesűrűség	$2082\,\mathrm{kg/m^3}$	$2379\,\mathrm{kg/m^3}$	$1993\,\mathrm{kg/m^3}$	$2387\,\mathrm{kg/m^3}$
E	Young-modulus	$2.979 \cdot 10^6\,\mathrm{Pa}$	$2.305 \cdot 10^6\,\mathrm{Pa}$	$8.960 \cdot 10^6\,\mathrm{Pa}$	$7.978 \cdot 10^6\,\mathrm{Pa}$
ν	Poisson-szám	0.259	0.475	0.227	0.486
e	ütközési együttható	0.063	0.258	0.104	0.113
μ	súrlódási együttható	0.684	0.637	0.684	0.495
μ_r	gördülési együttható	0.229	0.277	0.282	0.264
egyedek száma a klaszterban		5	4	9	10
$\overline{F}_x$	vonóerő	316.0 N	303.9 N	312.5 N	310.5 N
ΔF_x	vonóerő szórása	25.6 N	26.2 N	25.5 N	25.8 N

A klaszterezés után 4 nagyobb csoport azonosítható, amelyek a 4.2. táblázatban láthatók, feltüntetésre került, hogy a csoportok hány egyedet tartalmaznak. Látható, hogy az optimalizálási problémának több megoldása is van. Minden megadott paraméterkombinációra a táblázat tartalmazza a szimulált vonóerőt és annak szórását. A 4.11. ábrán a szimulált vonóerő került ábrázolásra a négy különböző paraméterkombinációra és a mért vonóerő is feltüntetésre került szaggatott fekete vonallal. Minden paraméterkombinációra jó egyezés figyelhető meg a mérés és a szimuláció között az átlagerőre, azonban a szimulációkban nagyobb fluktáció mutatkozik mint a mérésben, ami a szimuláció diszkrét jellegével magyarázható, ugyanis a szimulációban a gömbök mérete meghaladta a valós talajszemcsék méretét. Az optimalizáció továbbfejleszthető és a későbbiekben a szórás szintén megadható mint optimalizálandó mennyiség, ezáltal mégjobb

egyezés is elérhető lenne. Az eset jól demonstrálja, hogy a saját készítésű GPUDEM-el akár egy nap alatt lehetséges a paraméterek kalibrálása és a mérés reprodukálása, amely meglehetősen rövidnek számít populáció alapú algoritmussal történő DEM paraméterkalibrációk esetén. A szakirodalomban található legtöbb esetben egy egyed kiértékelése órákig tart [47, 48], de még csökkentett szemcseszám esetén is több mint 10 percig tart [49].



4.11. ábra. A kultivátorra ható erők a talajlazítás során a különböző paraméterek mellett. A paraméterek a 4.2. táblázatban láthatók.

5. Összegzés

A dolgozatban bemutatásra került a saját fejlesztésű GPUDEM könyvtár. A kód felépítése a 2. fejezetben került bemutatásra. A program minden szemcsét a globális memóriába tárol, azonban erről a regisztermemóriában másolat készül, a program gyorsítása érdekében. Összességében megállapítható, hogy a megoldó struktúra, az időlépés és az erőszámítás rendkívül kicsi része a program futási idejének. A legnagyobb memória és számítási igényt a kapcsolatkeresés jelenti, a program által elvégzett műveletek több mint felét ez adja. Két kapcsolatkeresési eljárás is implementálásra került, a BruteForce és az optimalizált CLL, amelyek a 2.2. alfejezetben bemutatásra kerültek. A CLL kapcsolatkeresési eljárás egy hálót rendel a tartományhoz és ez alapján a szemcséket cellákba sorolja, ez a megközelítés közel tizedére csökkenti a futási időt. Láthattuk, hogy a CLL eljárás még hatékonyabban is implementálható GPU-kra, ahogy a 3.6. fejezetben a profilozás során bemutatásra került. Kicsi módosításokkal a program futási idő tovább feleződött.

Az alacsony futási idő lehetővé teszi egyrészt nagyméretű problémák szimulációját (ahol a szemcseszám több mint 100 000), másrész lehetővé válik a paraméter érzékenységi vizsgálat, hiszen néhány óra alatt több száz szimuláció is készíthető. A 4.2. fejezetben bemutatásra került egy kultivátoros szimuláció több mint 146 000 szemcsével, a szimulációban megfigyelhető volt, hogy a sebesség növelése növeli a kultivátorra ható erőt. Megállapításra került, hogy a szemcsesűrűség – és ezáltal a szemcsetömeg – növekedése lineárisan növeli a vonóerőt. A súrlódási tényező és a Young-modulus növelése szintén növeli a vonóerőt, azonban az összefüggés ezen esetekben nem lineáris. Az ütközési tényező növelése csökkenti a vonóerőt, ezen összefüggések kvalitatívan is magyarázatra kerültek a 4.3. fejezetben.

Differenciál-evolúcióval az anyagparaméterek optimalizálásra kerültek és 4 lehetséges paraméterkombináció került meghatározásra a 4.2. táblázatban, amelyek mind visszaadják a mérés eredményét. Az optimalizáció futási ideje kevesebb volt mint egy nap, pedig 2700 ülepítés és kultivátoros talajművelés történt meg több mint 30 000 szemcsével minden szimulációban.

5.1. Konklúzió

A fő eredmény tehát, hogy **a GPUDEM-el a tizedére csökkenthető a futási idő más szoftverekhez képest**, amely elsősorban GPU-s gyorsításnak köszönhető. Más GPU-s DEM szoftverekkel ellentétben a GPUDEM általánosan használható, bármilyen problémára, illetve több különböző méretű és anyagú szemcsét is alkalmazni lehet. Az alacsony futási időn túl a további célkitűzések is megvalósultak:

1. A GPU specifikus optimalizáció bemutatásra került. A globális memória használat csökkentésre került a gyors elérésű regisztermemória használata által, a memóriaelrendezés és szinkronizáció a 2.1. fejezetben került leírásra. Továbbá, az osztások száma is minimalizálásra került, amelyre egy példa a szemcsetömeg és szemcsesugár reciprokának a tárolása. A profilozás során a 3.6. fejezetben bemutatásra került a kód alacsony szintű működése, és megállapításra került, hogy a kapcsolatkeresés igényli a legtöbb számítási erőforrást.

2. A lényegesen alacsonyabb futási idő lehetővé tette sok szemcse alkalmazását ($N_P > 100\,000$) és az optimalizációt is, amely segítségével sikerült reprodukálni egy nedves homok talajban készült vonóerő mérést. A dolgozatban csak egy egyszerű példa került bemutatásra, de természetesen több mérésre együttesen is lehetséges a paraméteroptimalizáció, a fitnesz függvény megfelelő definiálásával.

3. Az alacsony futási idő lehetővé tette a paraméterérzékenységi vizsgálatot nagyszámú szimuláció készítésével, amely eredményei a 4.8. ábrán kerültek bemutatásra. A vizsgálat során a beállított paraméterek mellett megállapításra került, hogy a szemcsesűrűség növelése lineárisan növeli a szükséges vonóerőt. A Young-modulus és a súrlódási tényező növelése szintén nagyobb vonóerőt eredményez. A szimulációkból megállapításra került, hogy az ütközési tényező növelése a vonóerő csökkenését eredményezi, mivel nagyobb ütközési tényezők esetén kisebb tömörödési zóna alakul ki.

4. A DEM-es szakirodalomban jellemzően Euler-módszert használnak, azonban ezt a választást a legtöbb esetben nem indokolják. A 3.5. fejezetben bemutatásra került, hogy minden időlépési séma elsőrendben pontos, ha nem teljesül a folytonossági feltétel. Mint láttuk, az ütközések során az erők változása nem folytonos, tehát nincsen értelme magasabb rendű numerikus integráló módszerek alkalmazásának. A kapcsolatkeresési algoritmus választása sem egyértelmű. Kicsi problémák esetén ($N_P < 200$) a „BruteForce" megközelítés GPU-kon gyorsabbnak bizonyul, mint a CLL. Természetesen nagyszámú szemcse használata esetén a CLL lényegesen gyorsabb, hiszen ott csak a közeli cellákban lévő szemcsék között kerül meghatározásra a távolság.

5.2. További lehetőségek

A rendkívül gyors futási idő hozzájárulhat a jövőben néhány további valós probléma megoldásához:

1. A GPUDEM-el több szemcse szimulálható megadott idő alatt, mint más szoftverben. Ez lehetővé teszi a szemcseméret csökkentését és a talaj pontosabb modellezését. A talajművelés szimulációja során minden esetben fontos, hogy a szemcsék mérete a geometriához képest kicsi legyen. Amennyiben kicsi geometria részletek hatását szeretnénk vizsgálni, akkor ez különösképp igaz.

2. Az talajparaméterek kalibrálása DEM alkalmazása során számos esetben fontos, hiszen a DEM a valóságosnál nagyobb szemcséket alkalmaz, amelyek alakja is eltérő lehet. Az eltéréseket azzal korrigálhatjuk, hogy az anyagparamétereket módosítjuk, hogy ezáltal a szimulált talaj makroszkopikus viselkedés megegyezzen a valódi talaj viselkedésével. A GPUDEM a jövőben egy megfelelő eszköz lehet más típusú talajok DEM paramétereinek a meghatározására is.

3. A szerszámgeometria optimalizációja szintén lehetséges a GPUDEM-el. A geometriaoptimalizáció első lépése a talajparaméterek kalibrálása, ezután lehetséges például egy eke geometriájának a módosítása és a módosítás hatásának a vizsgálata.

A GPUDEM-be a jövőben egyszerűen beépíthetők más anyagmodellek is és más típusú problémák is vizsgálhatók. A DEM-VEM összeköttetés is könnyedén beépíthető, hiszen a geometria háromszögeire ható erők már most is számítva vannak, ez alapján pedig egy VEM programmal számítható a deformáció és az új deformált geometriával folytatható a szimuláció. Egyedül más alakú szemcsék használata igényelné a program lényegesebb módosítását, hiszen ekkor a kontaktkeresés lényegesen megváltozik, illetve a szemcsék tárolása is több adatot igényel. Jelenleg a GPUDEM használata programozási tudást igényel, hiszen egy C++ könyvtár formájában érhető el. A cél a jövőben egy előfordított program készítése, és a beállítások beolvasása egy bemeneti fájlból. Így a GPUDEM összeköttethető lenne a jelenleg fejlesztés alatt álló BMEDEM-el, amely már grafikus felülettel is rendelkezik, ez végsősoron lehetővé tenné a könnyebb alkalmazhatóságot.

Köszönetnyilvánítás

Köszönet *Dr. Tamás Kornél* konzulensnek az ötleteiért, javaslataiért és a korábbi mérési eredmények rendelkezésre bocsátásáért. Köszönet *Pásthy László* konzulensemnek a segítségéért, és a BMEDEM forráskódjának a megosztásáért, amely nélkül a GPU-s implementáció elkészítése lényegesen időigényesebb lett volna. Köszönet továbbá *Dr. Hegedűs Ferenc* korábbi TDK konzulensemnek, a GPU programozásba való bevezetésért 2019-ben. A Kulturális és Innovációs és Minisztérium ÚNKP-23-5-BME-80 kódszámú Új Nemzeti Kiválóság Programjának a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból finanszírozott szakmai támogatásával készült. A Bolyai János Kutatási Ösztöndíj támogatásával készül.

Hivatkozások

[1] János Péter Rádics és István J Jóri. „Development of 3E tillage system and machinery to challenge climate change impacts". *Periodica Polytechnica Mechanical Engineering* 54.1 (2010), 49–56. old.

[2] C Saunders, RJ Godwin és MJ O'Dogherty. „Prediction of soil forces acting on mouldboard ploughs". *Fourth International Conference on Soil Dynamics, Adelaide*. 2000.

[3] RJ Godwin és tsai. „A force prediction model for mouldboard ploughs incorporating the effects of soil characteristic properties, plough geometric factors and ploughing speed". *Biosystems engineering* 97.1 (2007), 117–129. old.

[4] John M Fielke. „Finite element modelling of the interaction of the cutting edge of tillage implements with soil". *Journal of Agricultural Engineering Research* 74.1 (1999), 91–101. old.

[5] Z Asaf, D Rubinstein és I Shmulevich. „Determination of discrete element model parameters required for soil tillage". *Soil and Tillage Research* 92.1-2 (2007), 227–242. old.

[6] Z Asaf, D Rubinstein és I Shmulevich. „Evaluation of link-track performances using DEM". *Journal of terramechanics* 43.2 (2006), 141–161. old.

[7] Kornél Tamás, István J Jóri és Abdul M Mouazen. „Modelling soil–sweep interaction with discrete element method". *Soil and Tillage Research* 134 (2013), 223–231. old.

[8] Kornel Tamas és Louis Bernon. „Role of particle shape and plant roots in the discrete element model of soil–sweep interaction". *Biosystems Engineering* 211 (2021), 77–96. old.

[9] Peter Alan Cundall. „The measurement and analysis of accelerations in rock slopes". (1971).

[10] Nuno Mendes, Sara Zanotti és José V Lemos. „Seismic performance of historical buildings based on discrete element method: An adobe church". *Journal of Earthquake Engineering* 24.8 (2020), 1270–1289. old.

[11] Benjamin Nassauer, Thomas Liedke és Meinhard Kuna. „Polyhedral particles for the discrete element method: geometry representation, contact detection and particle generation". *Granular matter* 15 (2013), 85–93. old.

[12] Peter A Cundall és Roger D Hart. „Numerical modelling of discontinua". *Engineering computations* 9.2 (1992), 101–113. old.

[13] Ruihuan Cai és tsai. „Modified cell-linked list method using dynamic mesh for discrete element method". *Powder Technology* 340 (2018), 321–330. old.

[14] Józef Horabik és Marek Molenda. „Parameters and contact models for DEM simulations of agricultural granular materials: A review". *Biosystems engineering* 147 (2016), 206–225. old.

[15] YT Feng. „An energy-conserving contact theory for discrete element modelling of arbitrarily shaped particles: Basic framework and general contact model". *Computer Methods in Applied Mechanics and Engineering* 373 (2021), 113454. old.

[16] Harald Kruggel-Emden és tsai. „Selection of an appropriate time integration scheme for the discrete element method (DEM)". *Computers & Chemical Engineering* 32.10 (2008), 2263–2279. old.

[17] Harald Kruggel-Emden és tsai. „A comparative numerical study of particle mixing on different grate designs through the discrete element method". (2007).

[18] JQ Gan, ZY Zhou és AB Yu. „A GPU-based DEM approach for modelling of particulate systems". *Powder Technology* 301 (2016), 1172–1182. old.

[19] Maksym Dosta és Vasyl Skorych. „MUSEN: An open-source framework for GPU-accelerated DEM simulations". *SoftwareX* 12 (2020), 100618. old.

[20] Dániel Nagy, Lambert Plavecz és Ferenc Hegedűs. „The art of solving a large number of non-stiff, low-dimensional ordinary differential equation systems on GPUs and CPUs". *Communications in Nonlinear Science and Numerical Simulation* 112 (2022), 106521. old.

[21] Dániel Nagy és Lambert Plavecz. „Nagyméretű paramétertanulmányok hatékony megoldása CPU-n és GPU-n". *35. OTDK Konferencia* (2020).

[22] Kamran Karimi, Neil G Dickson és Firas Hamze. „A performance comparison of CUDA and OpenCL". *arXiv preprint arXiv:1005.2581* (2010).

[23] Aaftab Munshi. „The opencl specification". *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE. 2009, 1–314. old.

[24] Serban Georgescu, Peter Chow és Hiroshi Okuda. „GPU acceleration for FEM-based structural analysis". *Archives of Computational Methods in Engineering* 20 (2013), 111–121. old.

[25] *NVIDIA Technical Blog*. 2017. URL: `https://developer.nvidia.com/blog/cooperative-groups/`.

[26] Guang-Yu Liu és tsai. „Study on the particle breakage of ballast based on a GPU accelerated discrete element method". *Geoscience Frontiers* 11.2 (2020), 461–471. old.

[27] Nicolin Govender és tsai. „A numerical investigation into the effect of angular particle shape on blast furnace burden topography and percolation using a GPU solved discrete element model". *Chemical Engineering Science* 204 (2019), 9–26. old.

[28] Luning Fang és tsai. „Chrono:: GPU: An open-source simulation package for granular dynamics using the discrete element method". *Processes* 9.10 (2021), 1813. old.

[29] *constexpr (C++)*. 2023. URL: `https://learn.microsoft.com/en-us/cpp/cpp/constexpr-cpp?view=msvc-170`.

[30] Torbjörn Granlund. „Instruction latencies and throughput for AMD and Intel x86 Processors". *Technical report, KTH* (2012).

[31] Laszlo Pasthy, Jozsef Graeff és Kornél Tamás. „Development Of A 2D Discrete Element Software With LabVIEW For Contact Model Improvement And Educational Purposes." *ECMS*. 2022, 203–209. old.

[32] DEM Solutions. „EDEM 2.6 theory reference guide". *Edinburgh, United Kingdom* (2014).

[33] Shahab Golshan és tsai. „Lethe-DEM: An open-source parallel discrete element solver with load balancing". *Computational Particle Mechanics* 10.1 (2023), 77–96. old.

[34] Ernst Hairer, Syvert P Nørsett és Gerhard Wanner. *Solving ordinary differential equations I. Nonstiff problems*. 1993.

[35] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[36] *VTK File Formats - VTK documentation*. 2023. URL: `https://docs.vtk.org/en/latest/design_documents/VTKFileFormats.html#unstructuredgrid`.

[37] Utkarsh Ayachit. *The paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[38] James Ahrens és tsai. „36-paraview: An end-user tool for large-data visualization". *The visualization handbook* 717 (2005), 50038–1. old.

[39] Alfredo Bellen és Marino Zennaro. *Numerical methods for delay differential equations*. Numerical Mathematics és Scietific Computation, 2013.

[40] Gerhard Wanner és Ernst Hairer. *Solving ordinary differential equations II*. 375. köt. Springer Berlin Heidelberg New York, 1996.

[41] Alberto Di Renzo és Francesco Paolo Di Maio. „Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes". *Chemical engineering science* 59.3 (2004), 525–541. old.

[42] Jens A Melheim. „Cluster integration method in Lagrangian particle dynamics". *Computer physics communications* 171.3 (2005), 155–161. old.

[43] John Cheng, Max Grossman és Ty McKercher. *Professional CUDA c programming*. John Wiley & Sons, 2014.

[44] Mustafa Ucgul, John M Fielke és Chris Saunders. „Three-dimensional discrete element modelling (DEM) of tillage: Accounting for soil cohesion and adhesion". *Biosystems Engineering* 129 (2015), 298–306. old.

[45] Vitaliy Feoktistov. *Differential evolution*. Springer, 2006.

[46] John A Hartigan és Manchek A Wong. „Algorithm AS 136: A k-means clustering algorithm". *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), 100–108. old.

[47] Retief Lubbe és tsai. „Bayesian Calibration of GPU–based DEM meso-mechanics Part II: Calibration of the granular meso-structure". *Powder Technology* 407 (2022), 117666. old.

[48] Quang Hung Nguyen. „Machine learning in the calibration process of discrete particle model". B.S. thesis. University of Twente, 2022.

[49] M Javad Mohajeri, Huy Q Do és Dingena L Schott. „DEM calibration of cohesive material in the ring shear test by applying a genetic algorithm framework". *Advanced Powder Technology* 31.5 (2020), 1838–1850. old.

6. Appendix

6.1. táblázat. A mérés beállításai és eredménye

Átlagsebesség	$\overline{v}$	0.706 m/s
Szórás	Δv	0.091 m/s
Mért vonóerő	F_x	309.5 N
Szórás	ΔF_x	14.5 N
Tömeg alapú nedvességtartalom		3.96%

6.2. táblázat. A szimulációk beállításai

Szimuláció	#1	#2	#3	#4	#5
Tart. méret l/m	$2 \times 2 \times 2$	$3 \times 1 \times 3$	$2 \times 2 \times 2$	$1 \times 0.7 \times 2$	$1 \times 0.7 \times 2$
Háló (cellák száma)	–	$25 \times 8 \times 25$	$28 \times 28 \times 28$	–	$32 \times 32 \times 32$
$N_{P,C}$	–	12	8	–	8
$N_{C,\max}$	16	12	12	12	12
Kapcsolatkeresés	BruteForce	CLL	*változó*	BruteForce	CLL
Időlépés módszere	Exact	Exact	Exact	Euler	Euler
Δt/s	$1e-4$	$1e-4$	$1e-4$	$5e-5$	$1e-4$
Szemcseszám N_P	1024	509/8519	*változó*	147456	38912
Szemcsék tulajdonságai					
$\overline{R}$/mm	90	200/40	30	4.8	8.5
$\pm\Delta R$/mm	5	60/12	5	0.8	1.5
$\rho/(\mathrm{kg/m^3})$	1000	2500	1000	1850	1850
E/Pa	$2 \cdot 10^5$	$2.5 \cdot 10^8$	$2 \cdot 10^5$	$2 \cdot 10^6$	$2 \cdot 10^6$
G/Pa	$7.69 \cdot 10^4$	$1 \cdot 10^8$	$7.69 \cdot 10^4$	$7.24 \cdot 10^5$	$7.24 \cdot 10^5$
ν	0.3	0.25	0.3	0.38	0.38
e	0.1	0.5	0.1	0.5	0.5
μ	0.6	0.5	0.6	0.6	0.6
μ_0	0.7	0.5	0.7	0.7	0.7
μ_r	0.05	0.01	0.05	0.03	0.03
Geometria/fal tulajdonságai					
E/Pa	$2 \cdot 10^8$	$2.5 \cdot 10^8$	$2 \cdot 10^8$	$2 \cdot 10^8$	$2 \cdot 10^8$
G/Pa	$7.69 \cdot 10^7$	$1 \cdot 10^8$	$7.69 \cdot 10^7$	$7.69 \cdot 10^7$	$7.69 \cdot 10^7$
ν	0.3	0.25	0.3	0.3	0.3
e	0.1	0.5	0.1	0.5	0.5
μ	0.8	0.5	0.8	0.6	0.6
μ_0	0.9	0.5	0.9	0.7	0.7
μ_r	0.2	0.01	0.2	0.03	0.03