Ionhajtómű 3D modellezése

TDK Dolgozat

Vida Krisztián

2022. november 1.

Konzulens: Reichardt András

Tartalomjegyzék

1. Összefoglaló

Az utóbbi években a kisméretű műholdak esetében is felvetődött az ionhajtóművek alkalmazása. A létrehozott plazmát egy elektrosztatikus gyorsítórács segítségével megfelelő sebességre gyorsítjuk, majd a fúvókán keresztül távozik az űreszközön kívülre. A fúvókák felületén elhelyezett elektródákra kapcsolt feszültség segítségével befolyásolható a plazmanyaláb útjának alakítása és ezáltal a műhold pályájának módosítása.

TDK dolgozatom célja a fenti elgondolások ellenőrzésére alkalmas szimulációs környezete létrehozása. Az elektródák terét elektrosztatikus közelítéssel számítjuk ki. A numerikus megoldásra végeselemes módszer (FEM – Finite Element Method) alkalmazásával jutunk. A kapott elektromos tér alapján a gyorsított ionok mozgása kiszámítható. Munkám során egy hajtómű fúvókáját modellező háromdimenziós geometriát hoztam létre és vizsgáltam. A létrehozott geometrián különböző végeselem hálózási módszereket alkalmaztam, majd az így kapott hálóra összeállítottam a szükséges differenciálegyenlet rendszert, amit MATLAB használatával oldottam meg.

Dolgozatomban elsősorban a GMSH mint külső végeselem hálózó szoftver által biztosított lehetőségeket használtam és vizsgáltam. A geometria könnyű és gyors változtathatósága, valamint a MATLAB-ban történő megoldás érdekében, ezekre alkalmas függvénykönyvtárakat hoztam létre. A továbbiakban az így kapott eredményeimet szeretném bemutatni.

2. Abstract

In recent years, the use of ion propulsion has been considered for small satellites. The control associated with small size can be implemented inside the nozzles. The generated plasma is accelerated to a suitable velocity by means of an electrostatic accelerator grid and then ejected through the nozzle to the outside of the spacecraft. A voltage applied to the electrodes on the surface of the nozzles can be used to influence the path of the plasma and thus modify the satellite's orbit.

The aim of my TDK thesis is to create a simulation environment to verify the above ideas. The field of the electrodes is calculated using electrostatic approximation. The numerical solution is obtained using the Finite Element Method (FEM). The resulting electric field is used to calculate the accelerated ion motion. In my work, a three-dimensional geometry modeling the nozzle of an engine was created and investigated. I applied various finite element meshing methods on the generated geometry and then compiled the required differential equation system on the resulting mesh, which I solved using MATLAB.

In my thesis I mainly used and investigated the possibilities provided by GMSH as an external finite element meshing software. To make the geometry easy and quick to change and to solve in MATLAB, I created libraries suitable for these. In the following I would like to present my results.

3. Bevezetés

3.1. Elméleti áttekintés [1][2]

Induljunk ki a Maxwell-egyenletek differenciális alakjából:

$$\nabla \times \vec{\mathbf{B}} = \vec{\mathbf{J}} + \frac{\partial \vec{\mathbf{E}}}{\partial t} \tag{1}$$

$$\nabla \times \vec{\mathbf{E}} = -\frac{\partial \vec{\mathbf{B}}}{\partial t} \tag{2}$$

$$\nabla \cdot \vec{\mathbf{B}} = 0 \tag{3}$$

$$\nabla \cdot \vec{\mathbf{D}} = \varrho \tag{4}$$

Továbbá szükségünk lesz a konstitúciós egyenletekre is:

$$\vec{\mathbf{D}} = \varepsilon_0 \varepsilon \vec{\mathbf{E}} \tag{5}$$

$$\vec{\mathbf{B}} = \mu_0 \mu \vec{\mathbf{H}} \tag{6}$$

$$\vec{\mathbf{J}} = \sigma \vec{\mathbf{E}} \tag{7}$$

Ahol $\vec{\mathbf{B}}$ a mágneses indukció, $\vec{\mathbf{J}}$ az áramsűrűség, $\vec{\mathbf{E}}$ a térerősség, $\vec{\mathbf{D}}$ az elektromos eltolás t az idő, ϱ a töltéssűrűség, ε_0 a vákuum permittivitása, ε a relatív permittivitás, μ_0 a vákuum permeabilitása, μ a relatív permeabilitás, σ pedig a közeg vezetőképessége.

A probléma elektrosztatikus modellel közelíthető az állandó geometria és a lassan változó elektródapotenciálok miatt. Emiatt $\frac{\partial \vec{\mathbf{E}}}{\partial t}$ elhanyagolható, továbbá az egyes ionok által keltett mágneses teret ($\frac{\partial \vec{\mathbf{H}}}{\partial t}$) is elhanyagolhatjuk.

Ezek fényében elektrosztatikai modellel élhetünk, tehát bevezethető a skalárpotenciál:

$$\vec{\mathbf{E}} = -\nabla \phi \tag{8}$$

Ekkor felírható az elektrosztatikus modellre a Laplace-Poisson egyenlet, amely megoldását végeselem módszerrel végeztem a MATLAB segítségével.

$$\nabla \varepsilon_0 \nabla \phi = 0 \tag{9}$$

Az egyenlőség jobb oldalán 0 szerepel, mert a világűr töltéssűrűségét elhanyagolhatjuk.

3.2. Peremfeltételek

A peremfeltételek a külvilág hatását jelenítik meg a vizsgált zárt tartomány szempontjából. Kétféle peremfeltételt alkalmazhatunk, az első a Dirichlet-típusú peremfeltétel, amely esetében a potenciál előírt, a második pedig a Neumann-típusú, amely esetében a potenciál normális szerinti deriváltja előírt.

4. A GMSH használata [3]

A GMSH program segítségével speciális végeselem hálókat lehet előállítani, attól függően, hogy mi a célunk. Minden esetben az a fontos, hogy a rendelkezésre álló számítási kapacitást a lehető legjobb módon használjuk ki, illetve, hogy a modell fizikailag helytálló legyen.

4.1. Geometria megadása

A geometria megadása a *Geometry* modulban lehetséges. Itt lehetőség van pontokat definiálni az *xyz* koordinátái segítségével, valamint az így hozzáadott pontokra illeszthetünk, egyeneseket, köríveket és sok más alakzatot is. Korábbi munkám során kétdimenziós alakzatokkal foglalkoztam, amelyeket most háromdimenziós esetekre terjesztek ki.

Egy projekt létrehozásakor mindig létrejön egy *.geo* kiterjesztésű fájl, amely script-ként szerkeszthető egyszerűen például Notepad használatával az *Edit script*-re kattintva. Minden módosítás, amit a gmsh grafikus interfészén végzünk az ennek a script-nek egy soraként jelenik meg. Ennélfogva tulajdonképpen bármilyen módosítást végrehajthatunk egyaránt a script szerkesztésével vagy a grafikus interfész használatával. Ha a script-ben történt változtatásokat meg szeretnénk jeleníteni, akkor a szerkesztés elmentése után a *Reload script*-re kattintva végbemennek a változtatások. Korábbi munkám során a grafikus felületen található parancsokat használtam és mutattam be, azonban háromdimenziós esetben néha már nehézkesz a megfelelő pontra vagy egyenesre kattintani, mert akár fedhetik egymást, ezért ezúttal gyakran a script szerkesztése mellett döntöttem.

Elsőként a pontok koordinátáit szükséges megadni. Ezt a grafikus felület használatával az *Elementary entities* fület lenyitva az *Add* opción belül a *Point* parancsra kattintva tehetjük meg. Ekkor felugrik egy ablak, ahol megadhatjuk az elhelyezni kívánt pont *xyz* koordinátáit, valamint a *Prescribed mesh size at point* paramétert, ami azt jelenti, hogy a létrejövő pont környezetében a projekt alapértelmezett végeselem háló méretéhez képest mennyire legyen finom vagy durva a háló. Ugyanez a folyamat a *.geo* fájl szerkesztésével a következőképpen néz ki:

Point(ID) = {x, y, z, mesh size at point};

Miután vannak pontjaink, ezekre egyeneseket vagy akár görbéket illeszthetünk. A gmsh felületén ezt az *Add* → *Line* parancsra kattintva tehetjük meg. Ekkor ki kell választani a létrehozni kívánt egyenes kezdő- majd végpontját. A script-ben való megadás a pont megadásához hasonló módon történik:

Line(ID) = {start_point, end_point};

Körív megadása is egyszerűen lehetséges, abban az esetben a *Circle* parancs három paramétert vár, a körív kezdőpontjának ID-ját, a kör középpontjának ID-ját, illetve a körív végpontjának ID-ját.

A pontokat összekötő egyenesek, illetve ívek megadása után ezekből felületeket kell definiálnunk. Ez elsőre a grafikus interfész használatával egyszerűbbnek tűnhet, ugyanis szintén az *Add* menün belül a *Plane surface* opciót választva a síkidomot határoló vonalakra kattintva

tehetjük meg. Míg a script-es megadás során először egy *Curve Loop* definiálása szükséges, azonban itt figyelni kell a körüljárásra. Tehát, ha például egy négyszöget szeretnénk definiálni az $1 \to 2 \to 3 \to 4 \to 1$ pontok körüljárásával, akkor az ezeket a pontokat összekötő vonalak ID-jára hivatkozva tehetjük meg. Azonban, ha a 2-es és 3-as pont között olyan módon hoztunk létre egy egyenest a *Line* paranccsal, hogy annak a kezdőpontja a 3-as pont és a végpontja a 2-es pont, akkor annak az ID-jára egy negatív előjellel kell hivatkozni. Ebben a példában tehát így nézne ki a script-nek ezen sora:

$Curve\ Loop(ID) = \{Line_1, -Line_2, Line_3, Line_4\};$

Ezt követően a *Plane Surface* paranccsal a korábban megadott *Curve Loop*-ból létrehozhatjuk a kívánt felületet.

A felületek létrehozása után ezekből térfogatot állíthatunk össze. Az eddigiekhez megszokott módon a grafikus interfész használatával az *Add* menün belül a *Volume* opciót választva, majd a kívánt határfelületek kijelölésével tehető meg. A srcipt-ben a kétdimenziós alakzatokhoz hasonlóan kell eljárni. Most a *Surface Loop* parancs használatával, aminek paramétereinek a sík felületek ID-ját kell megadnunk. Majd az így létrehozott *Surface Loop*-ból megalkothatjuk a térfogatot a *Volume* paranccsal.

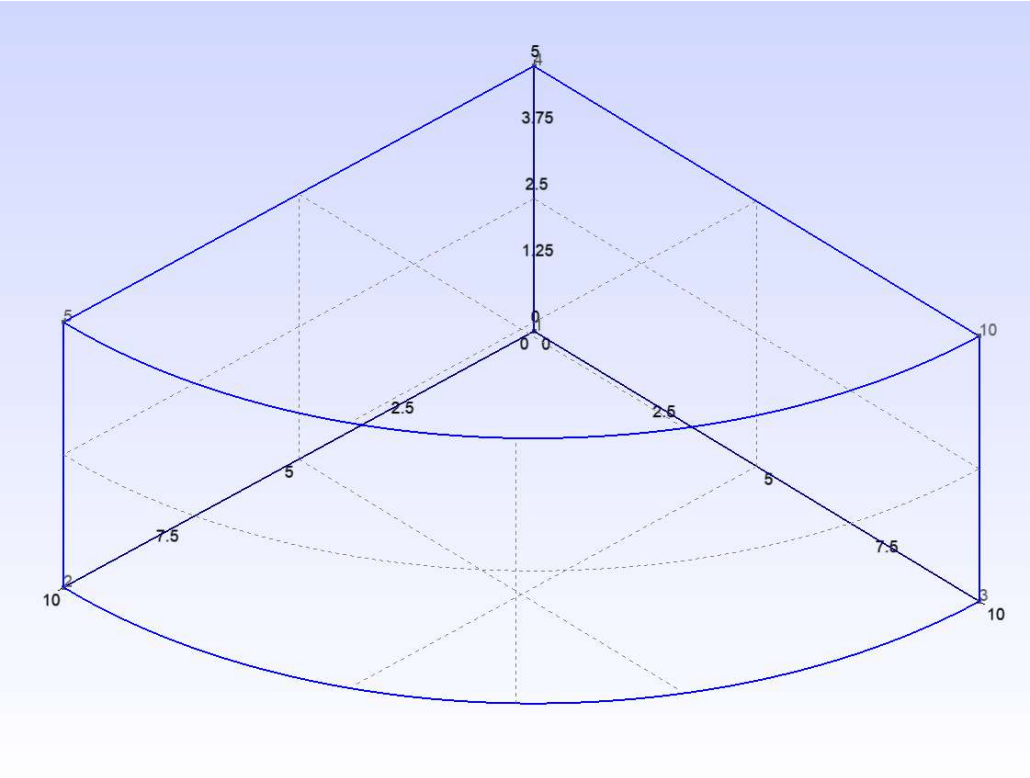
Bizonyos speciális esetekben egy térfogat létrehozását jelentősen megkönnyíthetjük, illetve felgyorsíthatjuk. Például, ha egy forgástestet szeretnénk konstruálni, akkor elegendő a megfelelő sík felületet létrehozni, amit az *Extrude* $\to$ *Rotate* paranccsal megforgathatunk. Vagy ha egy hasábot szeretnénk létrehozni, akkor vagy az *Extrude* $\to$ *Translate*-re kattintva, vagy pedig script-ben egyszerűen az *Extrude* parancs használatával.

Az 1. ábrán látható egy ilyen módon megalkotott egyszerű geometria, ahol egy negyedkör alapú egyenes hasábot hoztam létre.

4.2. Speciális hálózási lehetőségek

A GMSH különböző lehetőségeket biztosít egyéni végeselem háló konstruálására. A háló létrehozása a *Mesh* modulban tehető meg. Itt van lehetőségünk különböző felbontású végeselem háló megadására, ha ezzel nem élünk, akkor kétdimenziós esetben egyszerűen a *2D* parancsra kattintva automatikusan létrejön egy olyan háromszögháló, amelyben a leghosszabb oldal nagysága legfeljebb az *Options - Mesh* menüben megadott *Element size factor* értékével lehet egyenlő. Háromdimenziós esetben pedig a *3D* parancsra kattintva a fent leírt paramétereknek megfelelő tetraéderes háló jön létre.

Azonban ha változó felbontású hálót szeretnénk kapni, akkor azt *Define* menüpontban kínált lehetőségek közül választva tehetjük meg. Az egyik leghasznosabb opció mind közül a *Size fields*, ugyanis itt lehetőség van többek között kör vagy téglalap területeket vagy gömb, téglatest és egyéb térfogatokat definiálni, amik nem jelennek meg objektumként a szimulációs térben, viszont megadható bennük az alapértelmezettnél kisebb *Element size factor*, ami azt eredményezi, hogy azon a tartományon belül jobb lesz a háló felbontása. Ezen kívül a *Size at points* segítségével a kijelölt pontoknál tudunk sűrűbb hálót elérni, illetve a *Transfinite* menün



1. ábra. Egy egyszerű geometria bemutatása

belül, a *Curve* opciót választva azt tudjuk megadni, hogy az általunk kijelölt görbéken vagy egyeneseken a végeselem hálónak hány pontja legyen.

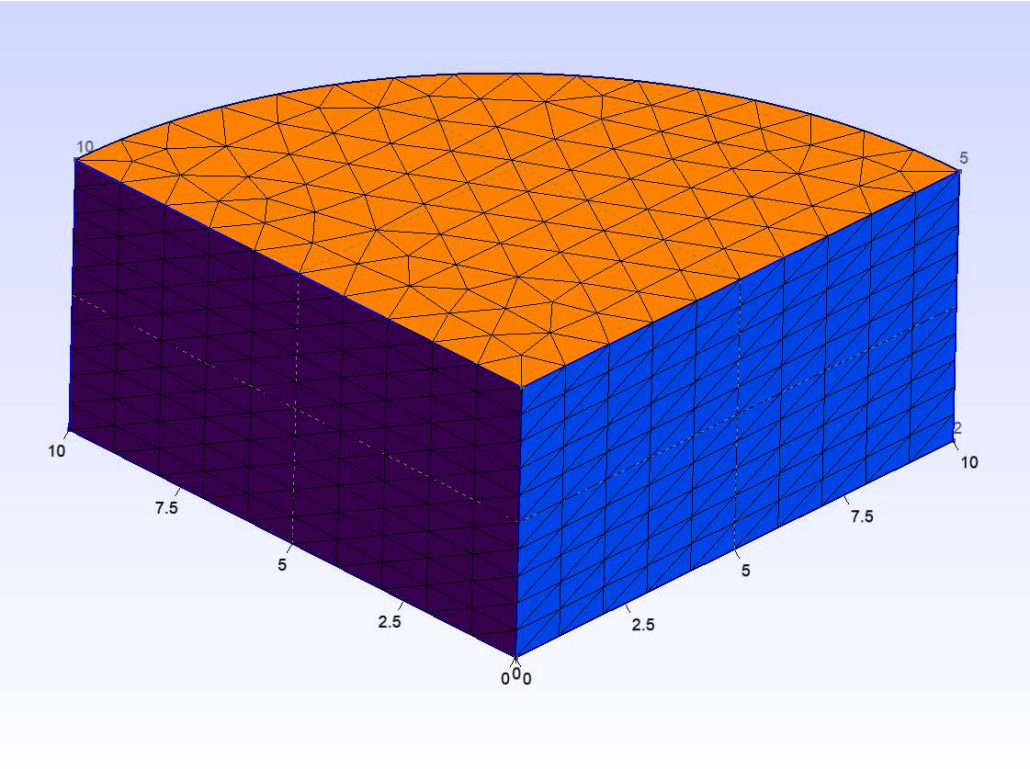
Ha el akarunk térni az alapértelmezett háromszög, vagy háromdimenziós esetben tetraéderes szerkezetű hálózástól, azt a *Recombine* parancs megfelelő használatával tehetjük meg. Ezt vagy a grafikus interfészen használjuk és rákattintunk arra a felületre, ahol négyszögrácsos hálót szeretnénk kapni, vagy pedig a script-ben a *Recombine Surface {surface_ID};* paranccsal tehetjük meg. Fontos megjegyezni, hogy csak azokon a felületeken lesz érvényes a *Recombine* parancs, amelyekre alkalmazzuk, tehát ilyen módon háromszög alapú egyenes hasábokból felépülő végeselem hálót is létre lehet hozni.

Egy másik nagyon fontos opció, amit használhatunk háromdimenziós hálózás során, az a *Layers* parancs. Ezt leginkább az *Extrude* paranccsal együtt érdemes használni, például a következő módon:

$Extrude\{0,0,5\}\{Surface\{1\};Layers\{10\};\}$

Ezzel az történik, hogy a síkbeli alakzatot, amit korábban létrehoztunk, z irányba kinyújtjuk 5 egység hosszan, és ellátjuk 10 réteggel.

A 2. ábrán látható a korábban létrehozott geometriára elkészített háló, a fent részletezett *Layers* segítségével megadott 10 vízszintes réteggel. A jobb átláthatóság kedvéért a végeselem háló elemeinek felületét színekkel kitöltve jelenítettem meg, így jobban láthatóak a vízszintes rétegek, valamint, hogy továbbra is háromszög alapú minden elem, tehát a háló tetraéderes szerkezetű.



2. ábra. Példa rétegzett halózásra

4.3. Létrejövő msh fájl szerkezete

A *Save Mesh* paranccsal lehet kimenteni a kialakult végeselem hálót egy *.msh* kiterjesztésű fájlba. Ez tulajdonképpen egy speciális *.txt* kiterjesztés, aminek a felépítése megtalálható a GMSH dokumentációjában. A fájl számos a formátumot illető adattal indul, amelyek számomra közvetlenül most nem voltak jelentősek.

A számomra fontos információkat a végeselem háló pontjainak x, y és z koordinátái, illetve a hálót alkotó tetraéder elemek adatai jelentették. A pontok a *$Nodes* string után az *$EndNodes* string-ig érkeznek sajátos struktúrában. A fájl megadja minden ilyen pont esetében ciklikusan egymást követő sorokban, hogy melyik objektumhoz tartozik (pont, egyenes, felület vagy térfogat ID-ja), illetve maguknak a pontoknak a sorszámát is, amit a pontok koordinátái követnek. Számomra jelen esetben csak a pontok koordinátái az értékes adatok, így azoknak a soroknak a tulajdonságaira koncentráltam, amik ezeket tartalmazzák. Ezeknek és csak ezeknek a soroknak közös tulajdonsága, hogy három double értéket tartalmaz köztük szóközt használva elválasztó karakterként.

A másik nagy jelentőséggel bíró információ a végeselem hálót alkotó tetraéder elemek tulajdonságai, egész pontosan az, hogy melyik négy pont alkotja őket. Ezek az adatok az *$Elements* és *$EndElements* string-ek között találhatóak. Itt is szerepel, hogy melyik objektum pontjai következnek a felsorolásban, viszont 1, illetve 2 dimenziós alakzatok (pontok, egyenesek, síkok) is megjelennek itt. Azonban ezeknek az objektumoknak az adott sorban az első adata a dimenziója, tehát könnyen meg lehet találni az egyetlen háromdimenziós térfogatban található

elemeket. Miután megvan, hogy honnan következnek a 3D felület tetraéderes elemei, az azt követő sorokban pedig ezek jelennek meg négy egymást követő integer formájában, melyek közül az első egy ID, amire nincs szükségünk, a maradék négy szám pedig a tetraédert alkotó négy pont sorszáma.

5. Mesh fájl beolvasása MATLAB-ba

5.1. A beolvasás feltételei [4]

A beolvasó kódnak nemcsak annyi a feladata, hogy egy speciális *txt* fájl sorait eltároljuk egy tömbben, hanem, hogy ebből később egy PDE modellt létre lehessen hozni. Viszont a szokásos PDE Toolbox grafikus interfésze vagy parancssoros megadás helyett most egy már elkészített hálóból kell létrehozni a modellt. Erre a célra a MATLAB beépített *geometryFromMesh* függvénye ad lehetőséget, amely bemenetéül egy PDE modellt, egy nodes és egy elements mátrixot vár el. Modellként létrehozhatunk egy üres PDE modellt, amit a átadhatunk a függvénynek. A nodes és elements mátrixoknak azonban pontosan olyan struktúrájúaknak kell lenniük, mint amilyet a MATLAB is csinálna ha az általa biztosított PDE Toolboxot használva alkottuk volna meg a geometriát.

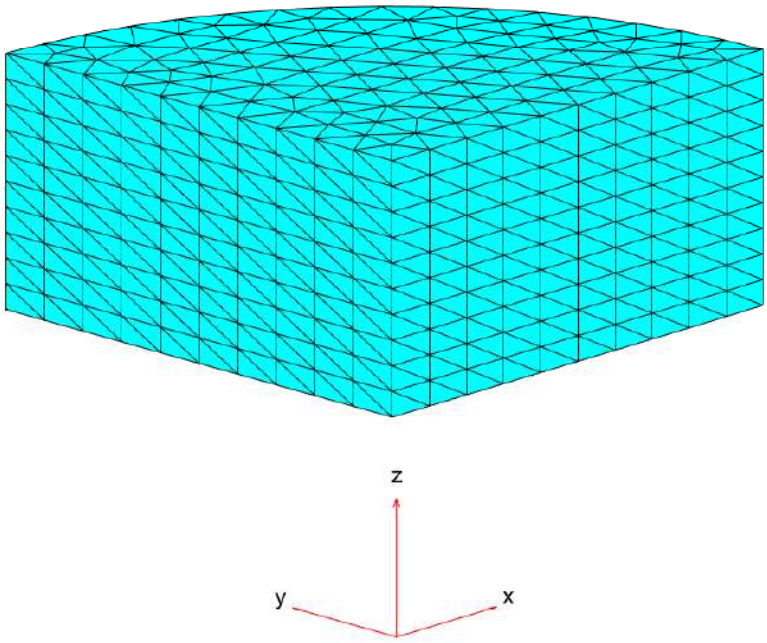
5.2. A beolvasó kód működése

A *.msh* fájlnak a 4.3. fejezetben bemutatott tulajdonságait kihasználva, a korábbi Önálló laboratórium munkám során megírt kétdimenziós beolvasó alapján megírtam egy háromdimenziós beolvasó programot, amely egy GMSH által generált háromdimenziós *.msh* kiterjesztésű fájlt képes beolvasni és abból létrehozni a PDE modellt.

A 3. ábrán látható a beolvasó program által immáron MATLAB-ba importált végeselem háló, amely megegyzik a GMSH-ben létrehozottal.

5.3. A beolvasás sebessége

A fájl megnyitása és az azon való végigfutás igényli a legnagyobb mértékben a számítási kapacitást, emiatt a beolvasás első lépésében az egész *.msh* fájl sorait eltárolom egy string tömb változóban, amin ezután könnyen és gyorsan lehet műveleteket végezni. Ilyen módon kellően gyors, néhány tizedmásodperces nagyságrendű futásidőt sikerült elérni a beolvasással.



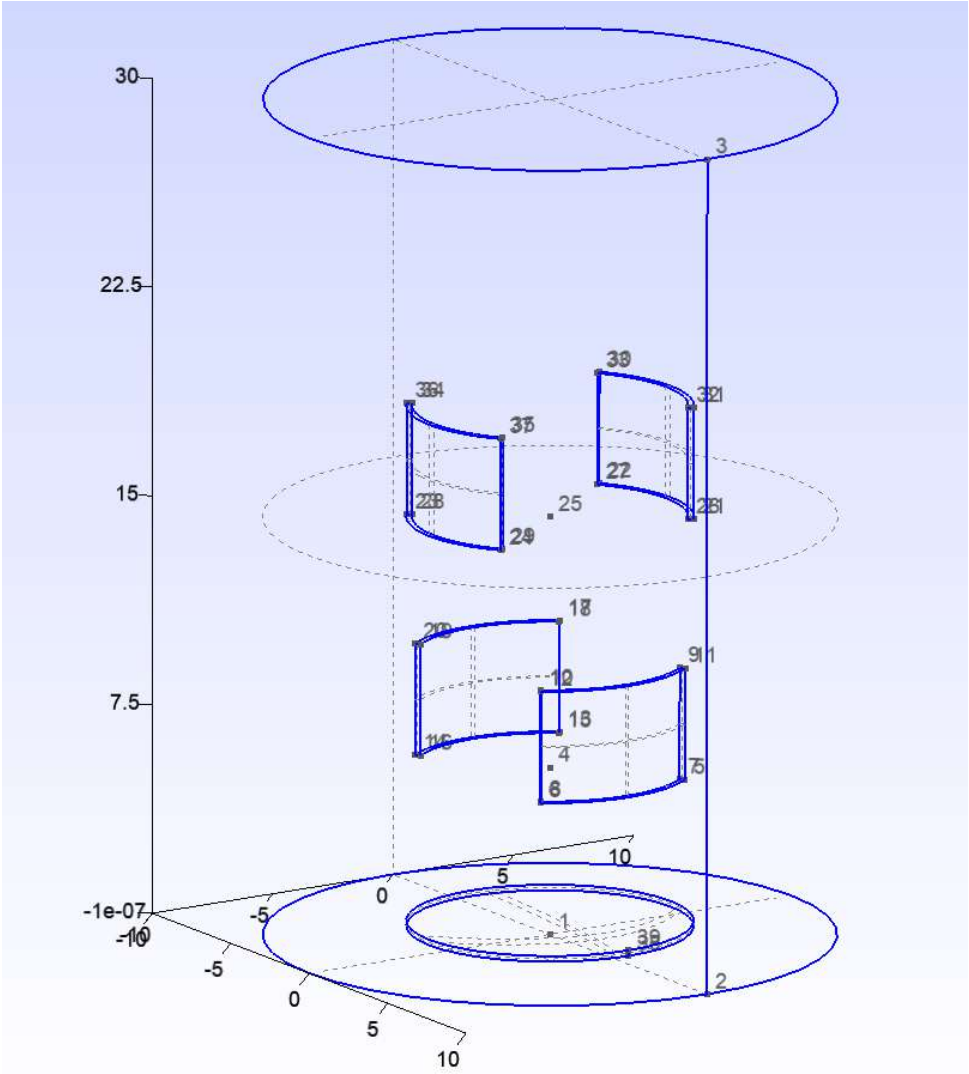
3. ábra. A háló beolvasva MATLAB-ba

6. Bonyolultabb geometria megadása

A korábban bemutatott egyszerű geometria GMSH-ben való létrehozása és beolvasása után létrehoztam az ionhajtóművel modellező összetettebb geometriát. Ez egy nagyobb, hengeres szimulációs térből, a kör alapú gyorsítórácsból, illetve az irányváltoztatásra szolgáló elektródapárokból áll. A modellben két pár elektródát használtam, amelyek elrendezése egymáshoz képes 90°-kal van elforgatva, így biztosítva egyaránt a vízszintes, illetve függőleges irányú elmozdulás lehetőségét.

6.1. A geomtria létrehozása GMSH-ben

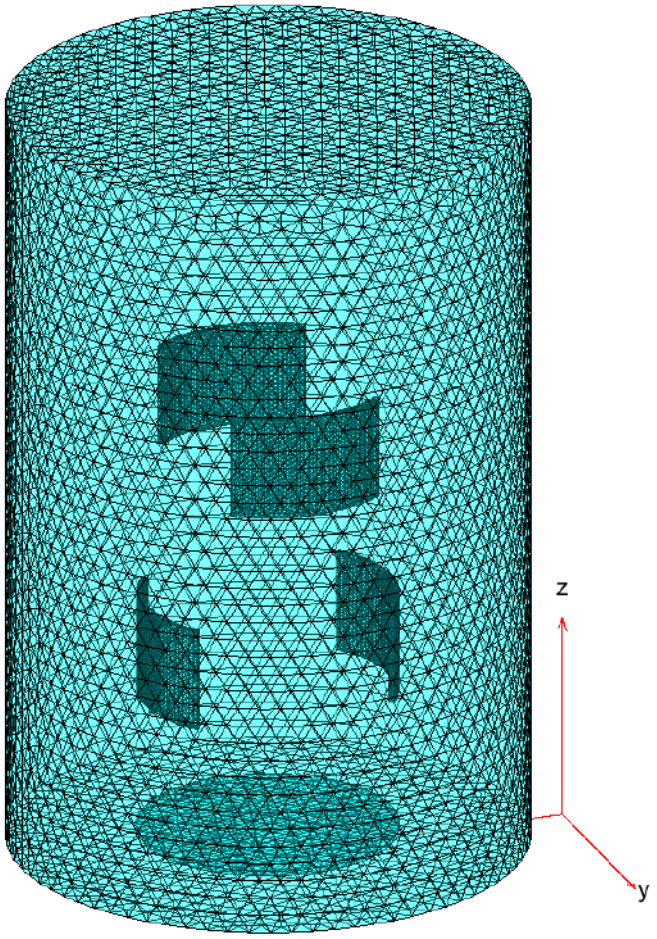
A fent leírt modell megalkotását a 4.1. fejezetben részletezett módon végeztem. Elsőként definiáltam egy 10 egység sugarú és 30 egység magasságú hengert, ami a szimulációs teret alkotja. Ezután létrehoztam a gyorsítórácsot modellező 5 egység sugarú, 0.2 egység vastagságú hengert, majd az elektródapárokat, amelyek egyenként egy 0.2 egység vastagságú, 4 egység magasságú körcgyűrűcikk alapú egyenes hasábokként kerültek modellezésre. Az így megalkotott geometria a 4. ábrán látható.



4. ábra. Az ionhajtómű modellje GMSH-ben

6.2. A végeselem háló elkészítése

Mint ahogy a 4.2. fejezetben bemutatásra került, a GMSH számos lehetőséget biztosít különféle végeselem hálók megalkotására. Munkám egyik célja azt vizsgálni, hogy a különböző jellegű hálók létrehozása, hogyan befolyásolja a megoldás minőségét és annak gyorsaságát. Első lépésben az alapból adódó legegyszerűbb esetet vizsgáltam, amikor a GMSH által létrehozott tetraéderes hálót használtam egyéb változtatások nélkül. Ennek az esetnek a vizsgálatához az *Element size factor* értékét 0.2 egység nagyságúra választottam annak érdekében, hogy a háló felbontása legalább akkora legyen, mint az elektródák vastagsága. A geometriához ilyen módon létrehozott hálót ezután elmentettem *.msh* kiterjesztésben, amit beolvastam MATLAB-ba. Az így kapott hálót 79727 pont és 465054 tetraéder alkotja. A MATLAB-ba sikeresen beolvasott végeselem háló és az abból megalkotott geometria az 5. ábrán látható.

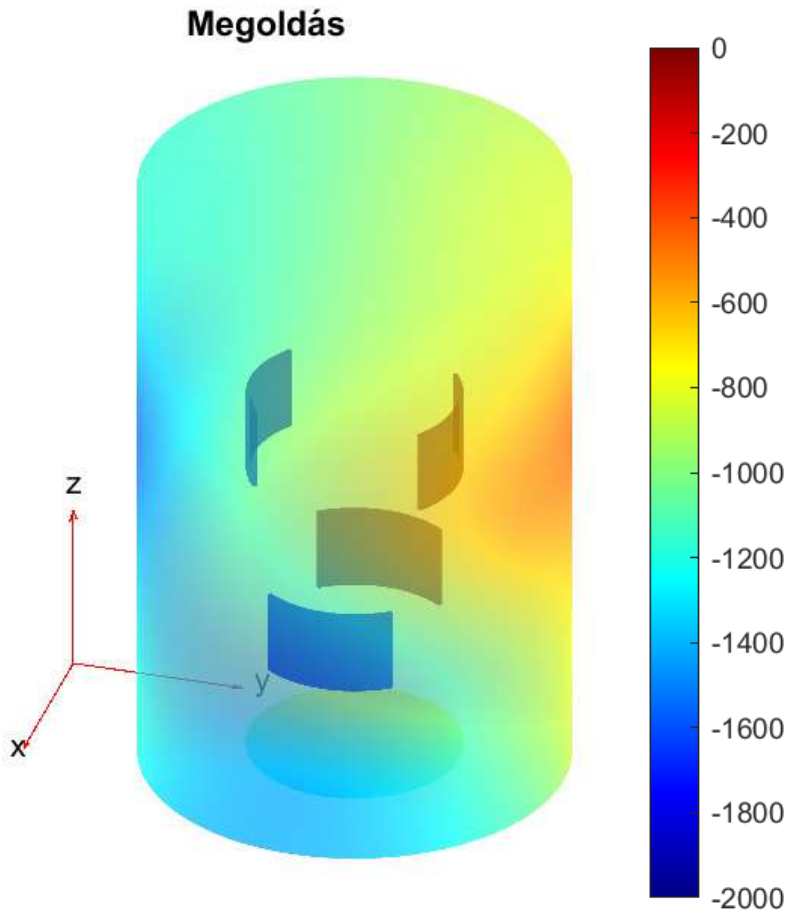


5. ábra. Az ionhajtómű modellje MATLAB-ban

6.3. Az elektrosztatikus probléma megoldása

Példaként az elektródák potenciálját úgy választottam meg, hogy az alsó gyorsítórácsot modellező elektróda $0V$ potenciálú legyen az elektródapárok egyik elektródája szintén $0V$ potenciálú, a másik pedig $-2kV$ potenciálú legyen. Ezen feszültség értékek egy egyaránt x és y irányú kanyarodást modelleznek. Az ehhez szükséges peremfeltételek megadására és az így adódó Laplace-Poisson egyenlet megoldására a MATLAB beépített pdetoolbox-ával végeztem, mivel a korábbi Önálló laboratórium munkám során megírt elektrosztatikus megoldó bár működött, de iteratív megoldás révén nagyságrendekkel lassabban futott le.

A szimulációs tér külső határaira tehát egy-egy Neumann típusú peremfeltételt adtam meg, az elektródák felületeire pedig Dirichlet típusú peremfeltételeket a fent leírt paraméterekkel. Ezután meghívtam a MATLAB pde megoldó függvényét, ami a 6. ábrán látható megoldást eredményezte.



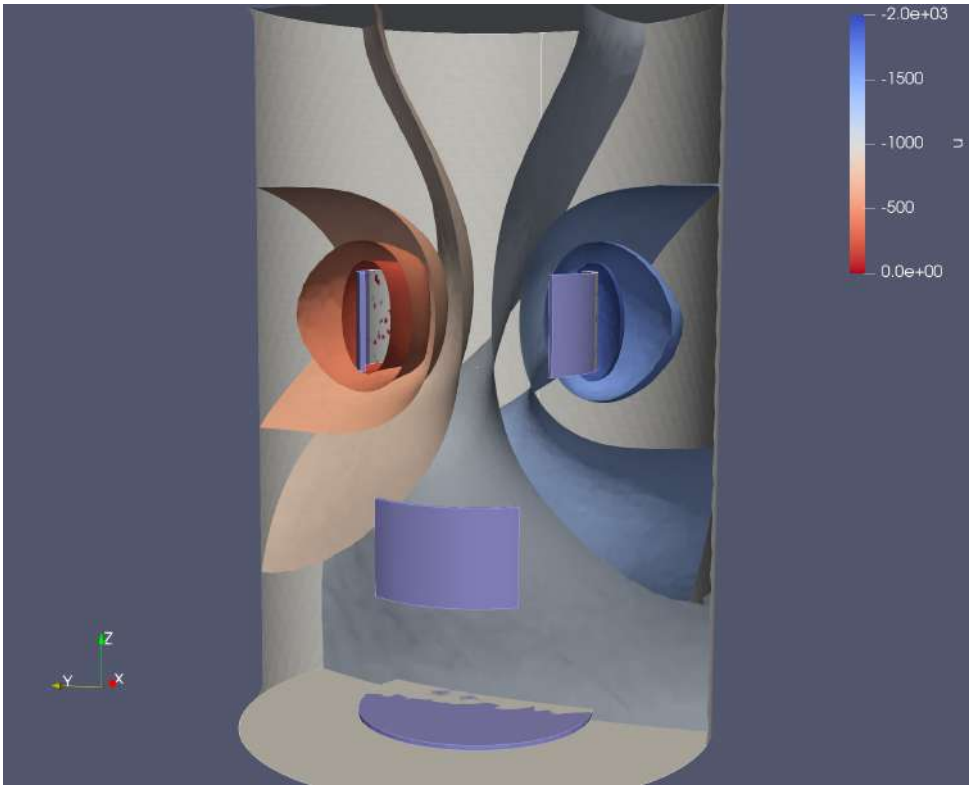
6. ábra. Megoldás MATLAB pdetool használatával

6.4. Ábrázolás Paraview segítségével

Mivel a MATLAB ábrázolási lehetőségei korlátozottak, így a fenti ábra sem mutat sokat abból, hogy hogyan alakul a potenciál eloszlása az elektródák közül, ezért érdemes ezt egy erre alkalmasabb célszoftver a Paraview használatával végezni.

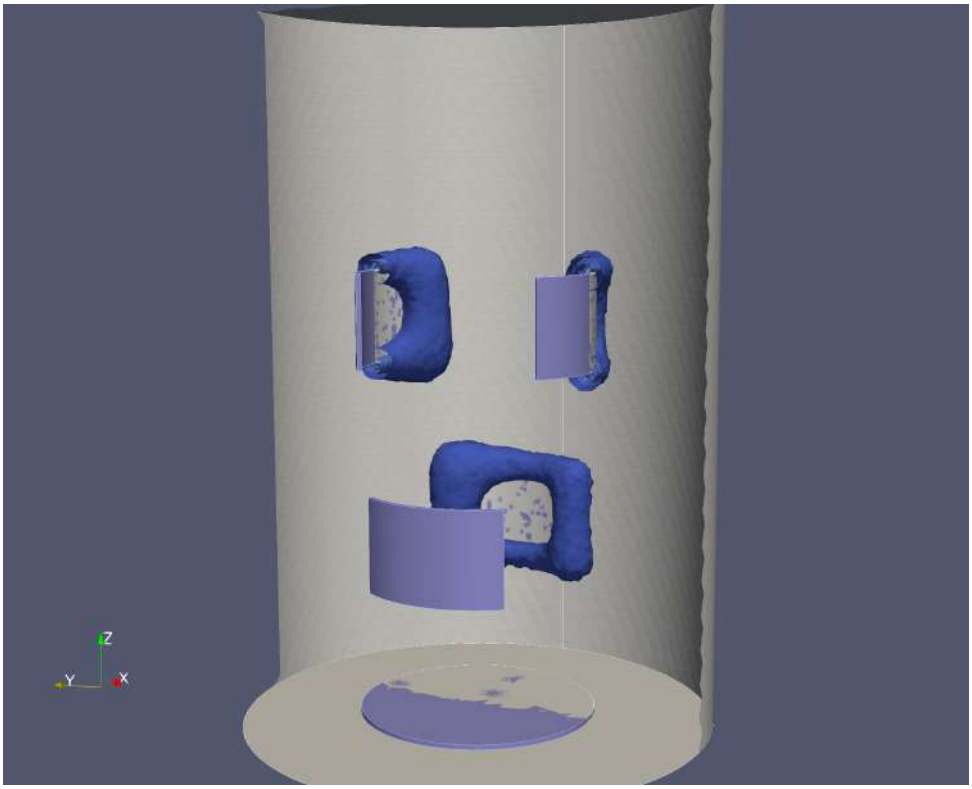
A GMSH lehetőséget biztosít többek között a végeselem háló *.vtk* kiterjesztésben való exportálására, amit a Paraview képes kezelni. Továbbá a MATLAB-ban már elvégzett megoldás egy *.m* fájba kimentve is használható ilyen módon a megoldás elemzésére.

A 7. ábrán látható az elektrosztatikai probléma megoldásának ábrázolása a belső, ekvipotenciális felületekkel kibővítve.



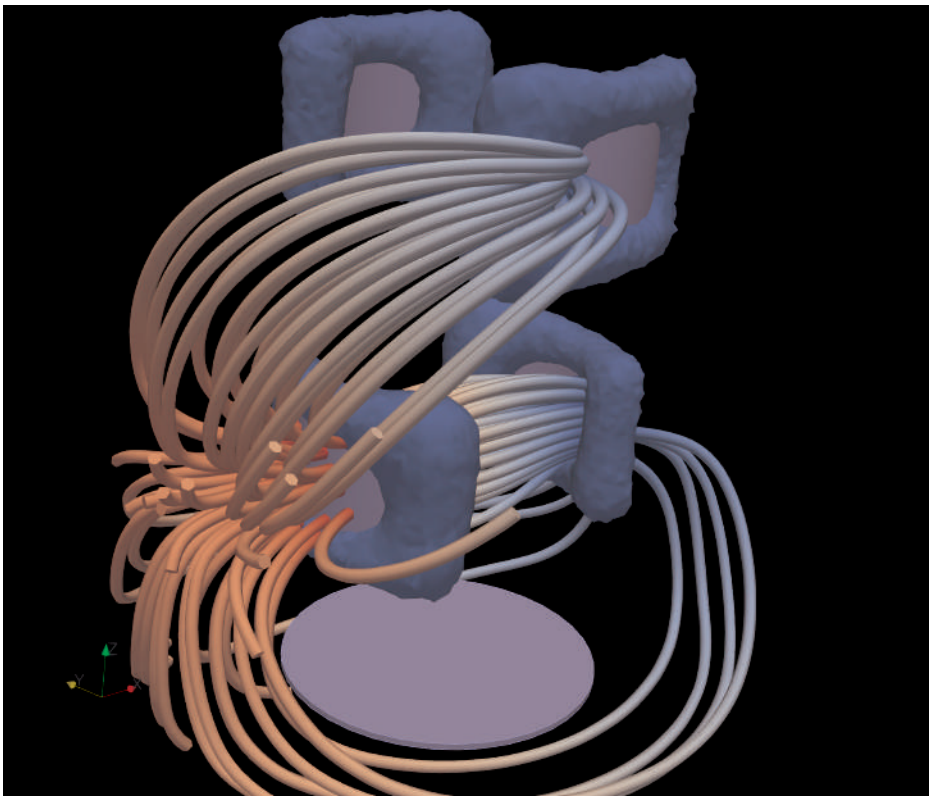
7. ábra. Ekvipotenciális felületek ábrázolása Paraview segítségével

Lehetséges továbbá az azonos energiájú részek ábrázolása is, amely a 8. ábrán látható. Megfigyelhető, hogy az elrendezés energiájának jelentős része az elektródáknál, illetve azoknak is főként a sarkainál sűrűsödik.

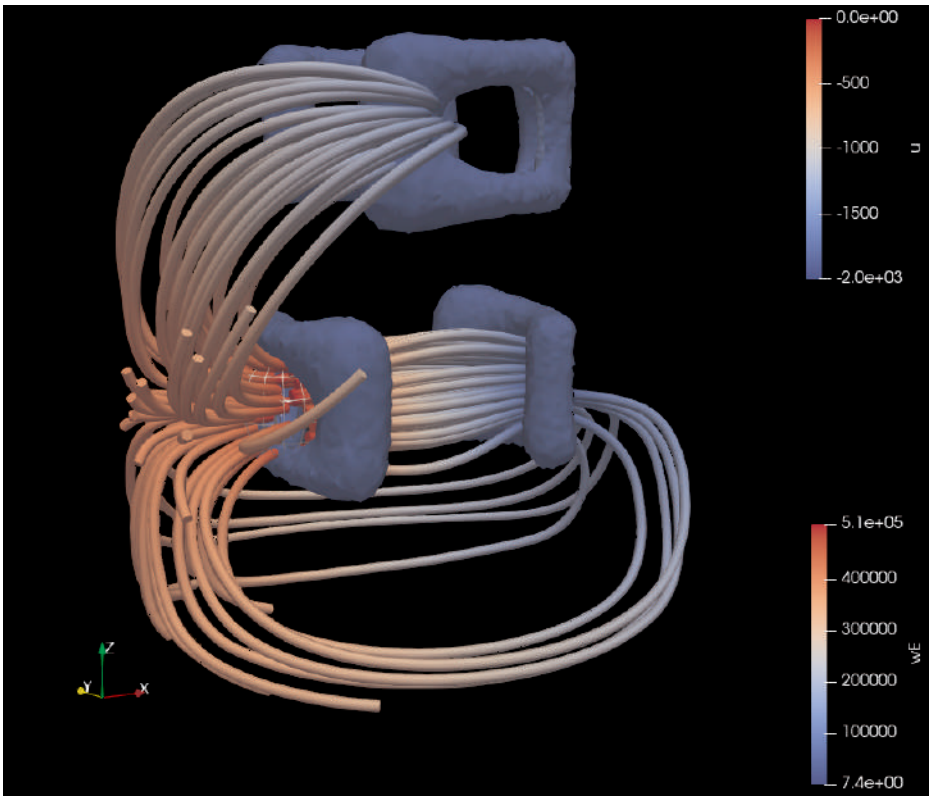


8. ábra. Azonos energiájú felületek

A 9. illvetve a 10. ábrákon pedig az erővonalak alakulása látható, az azonos energiájú felületek mellett. Utóbbi ábrán az elektródák nincsenek megjelenítve.

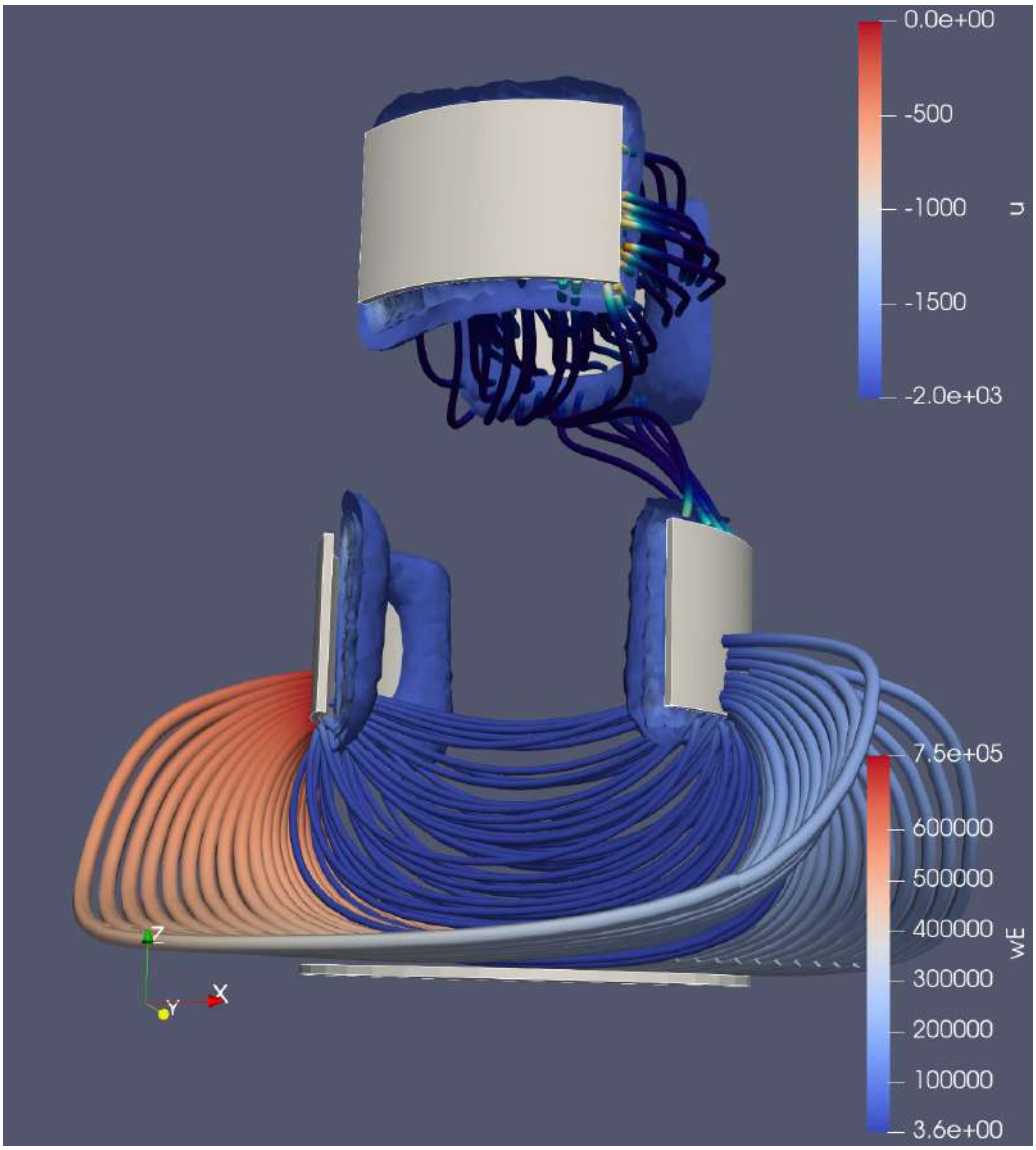


9. ábra. Erővonalak alakulása



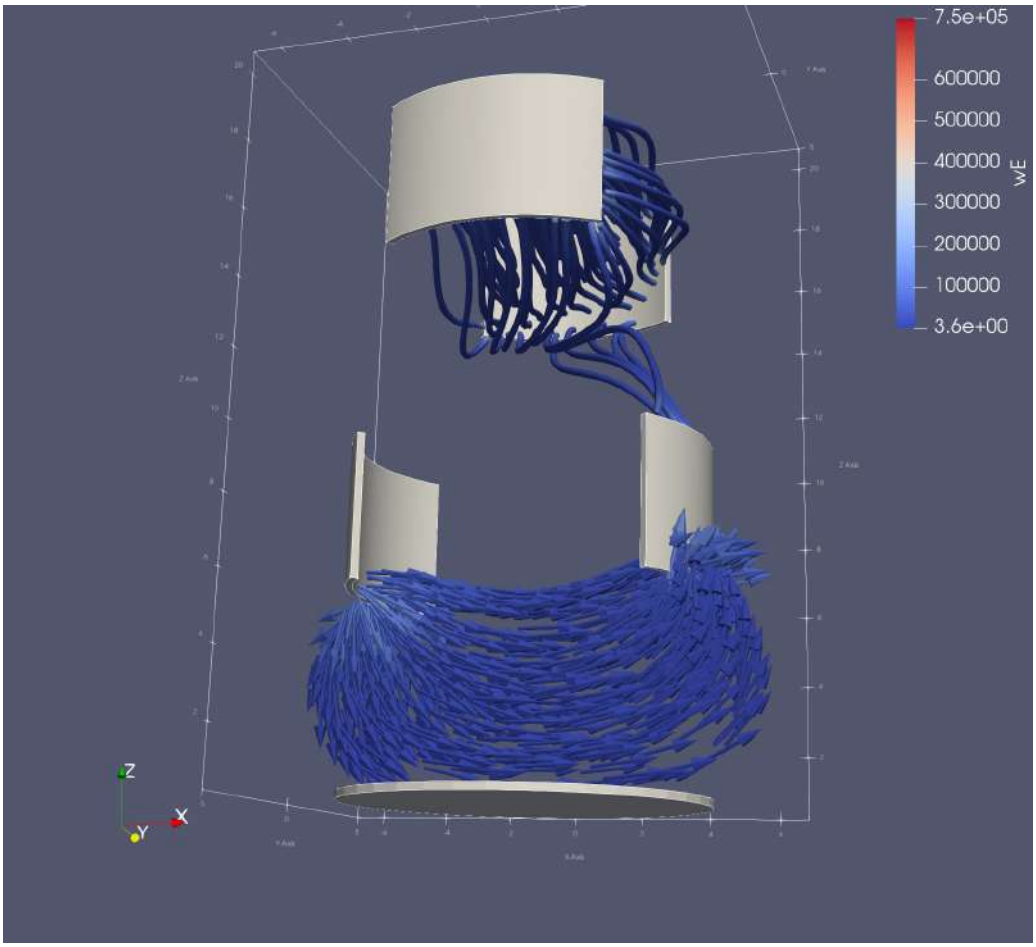
10. ábra. Erővonalak alakulása az elektródák nélkül

A 11. ábrán látható az azonos energiájú felületeknek csak azon része, amelyek az elektródák közötti térrészben helyezkednek el. Ezen az ábrán már jobban látszik, hogy ezek valójában is különálló rétegek, amelyek viszont nagyon közel helyezkednek el egymáshoz képest. Továbbá az erővonalak láthatóak még az ábrán olyan módon, hogy az alsó elektródapár alsó harmada és a gyorsítórácsot modellező elektróda között, illetve a felső elektródapár között kerültek megjelenítésre.



11. ábra. Az azonos energiűjú felületek rétegződése, az erővonalak speciális ábrázolása mellett

Végül pedig a 12. ábrán látható az erővonalak mentén a térerősség vektorok nagyságának és irányának az alakulása.



12. ábra. Az erővonalak mentén a térerősségvektorok

7. Geometria létrehozásának gyorsítása

Ahogy az korábban bemutatásra került GMSH-ben lehetőség van a grafikus interfész használatával, vagy a *.geo* kiterjesztésű fájl szerkesztésével a geometria megadására, azonban ez viszonylag körülményes. Ha egy adott geometriát szeretnénk vizsgálni különböző elektródapotenciálokkal, akkor ezt a folyamatot ugyan elég egyszer elvégezni, viszont ha változtatni szeretnénk az elektródák elrendezését, akkor még hasonló geometria esetén is, a már meglévőt alapul véve és azt módosítva is nehézkes. Márpedig nemcsak egyféle hajtómű modellt lehet elképzelni a gyakorlatban, ennélfogva a különböző elektródaelrendezések modellezése is indokolt. Az alábbiakban elektródák elrendezésének kiválasztásához és a leíráshoz tartozó megfontolásokat szeretném részletesen kifejteni.

7.1. Geometria

A szóba kerülő elrendezéseknél két alapvető dolgot kell meggondolni:

- szintek száma és a külső burkoló felület

  A szintek száma legyen 1 és 3 között. Ezzel elérhető a megfelelő tagoltság, de még reális.

- szinteken belüli elektróda elrendezés

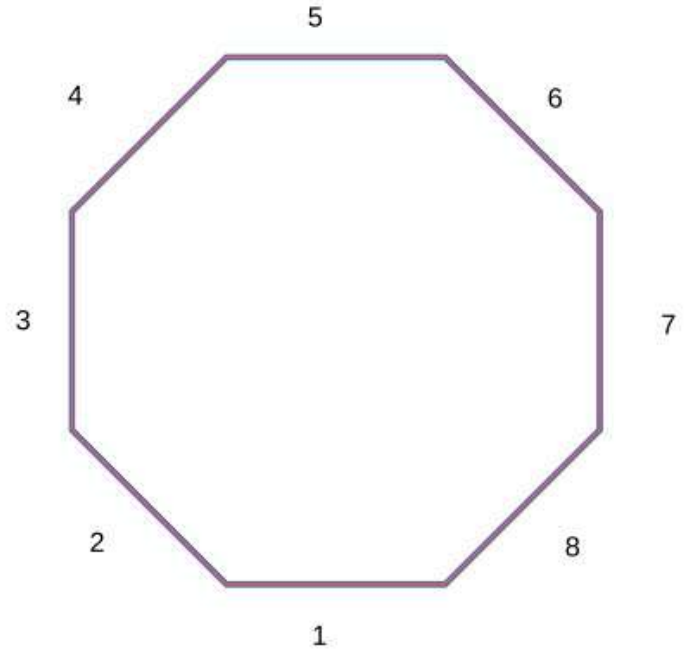
  A megfelelő szinten belüli elrendezés a külső befoglalótól is függ, de hengerszimmetrikus esetben a kör alakú a választás. Ennek hátránya az ívelt elektróda. Ha kettőnél több elektródát szeretnénk alkalmazni, akkor célszerű kör alakú befoglalót használni.

7.2. Lehetséges elrendezések - szinten belül

Olyan elrendezéseket vizsgáljunk , amelyeknél 8 szegmensnek van helye. Az egyes szegmensekre sorszámukkal lehet hivatkozni.

7.2.1. Oktagon

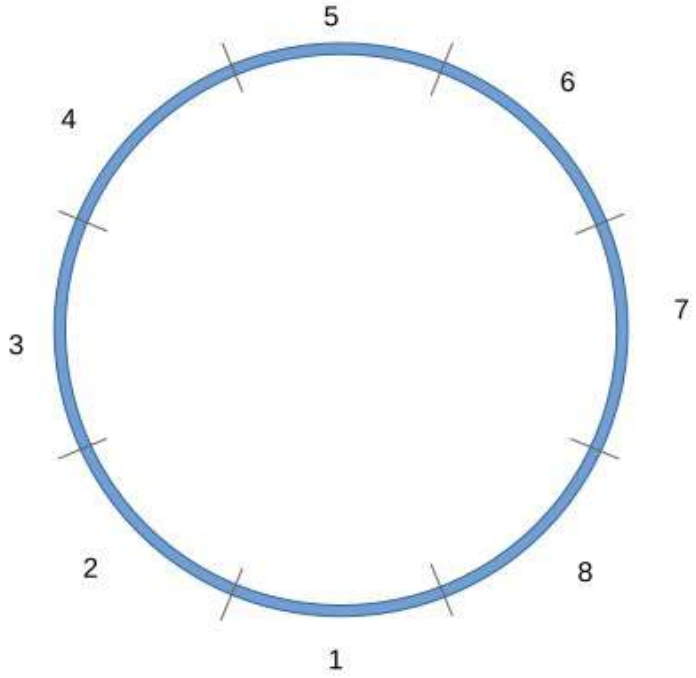
A geometria leírásához szükséges az elektródák sorszámának megadása, amelyek használatban vannak. Ebben elrendezésben a szomszédos szegmenseknek nem lehet különböző potenciálja, mert a határon illetve annak közelében numerikus problémákat okozna. Egy-egy szegmens (középponti) ívszöge 45°, (360°/8 alapján). Az elrendezés előnye, hogy egyenes (síklap felületű) elektródákat tartalmaz.



13. ábra. Nyolcszögletű elrendezés

7.2.2. Kör

Tipikusan jó meggondolás, ha a fúvóka külső kontúrja hengeres illetve hengerszimmetriával rendelkezik. A szegmensek "összeérnek", amiből következik, hogy a közvetlenül egymás melletti szekciók nem lehetnek különböző potenciálra kötve.



14. ábra. Kör alakú elrendezés

Néhány (értelmes) lehetséges elektróda használati map:

- 1,2 és 5,6 használva (szimmetrikus)
- 1,3,5,7 használva (szimmetrikus)
- 1,4,6 használva (aszimmetrikus)
- 1,2,4,7 használva (aszimmetrikus)

7.2.3. Elválasztott körcikkek

Az előző elrendezéshez hasonló, azonban az egyes szekciók között van egy elválasztó tartomány.

Legyen a szekciók középponti nyílásszöge (α) azonos, és az elválasztó tartomány nyílásszöge is azonos (ϵ). Ekkor

$$8 \cdot (\alpha + \epsilon) = 360^\circ \qquad \Longrightarrow \qquad \alpha + \epsilon = 45^\circ$$

Értelmes választás pl. $\alpha = 40^\circ$ és $\epsilon = 5^\circ$, mert ekkor már van megfelelő elválasztó távolság az esetleges szomszédos elektródák között is.

7.2.4. Leíró paraméterek

A megadandó paraméterek, amelyeket a konfigurációs fájlnak kell tartalmaznia:

- R_p - kör (külső) sugara
- d - elektróda vastagság
- h - elektróda magasság
- N - szekciók száma
- α - szekció nyílásszöge
- map - meghajtott szekciók listája (ahol van elektróda)

Olyan paraméterek, amelyek az előzőekből adódnak (nem kell megadni, mert számíthatóak):

- L - elektróda hossza ($\mathrm{L} = R_p \cdot \alpha$)
- ϵ - elválasztó rész nyílásszöge ($\epsilon = 45^\circ - \alpha$)
- r_0 - lekerekítés sugara ($r_0 = \mathrm{d}/10$)

7.3. Geometriát létrehozó MTALAB kód

A fent részletezett elektródaelrendezésekkel és geometria létrehozásával kapcsolatos gondolatok alapján elkészítettem MATLAB-ban egy függvénykönyvtárat (függelék B), amellyel lehetőség van a 7.2.4. fejezetben ismertetett leíró paraméterek megadásával egy GMSH geometria létrehozására, ezzel jelentősen meggyorsítva a különböző elektróda elrendezések megadását és azok vizsgálatát. A program létrehoz egy üres *.geo* kiterjesztésű fájl, amit feltölt a konfigurációs fájlban megadott leíró paraméterek alapján. Azonban a leíró paraméterek listáját az implementáció során fellépő problémák alapján kis mértékben módosítottam, mivel alapvetően háromszintes elektródaelrendezéseket vizsgáltam.

Ekkor ugyanis, a 7.2.2. fejezetben leírtak szerint, ha minden második szekcióban szerepel meghajtott elektróda, akkor a szekciók számától függetlenül az elrendezés sajátosságából adódóan, a középső szinten az elektródaelrendezés α szöggel való elforgatásával értelmes geometriát kapunk, azaz a szintek között nincs átfedés, ezáltal jobbak a lehetőségek a fúvókából kilépő töltéssel rendelkező részecskék irányítására. Nem ez a helyzet azonban például, ha az elektródakiosztás (1,2,5,6) szerinti, ekkor ugyanis $2 \cdot \alpha$ szöggel lenne célszerű elforgatni a középső szintet. Emiatt hozzáadtam az eredeti *map* leíró paraméteren (azaz a meghajtott szekciók listáján) túl, a középső szinten (ami minden páros számú szintet jelent, de három szint esetében ez egyszerűen csak a középső) meghajtott szekciók listáját is, mint bemeneti paraméter (*map_mid*).

Ezen túl meghagytam a lehetőséget, hogy egy-egy elektróda a saját szekcióját túl, a mellette elhelyhezkedő "üres" szekcióba túlnyúljon, tehát nagyobb α szöget lehessen megadni, mint a szekciók számából adódó hozzá tartozó középponti szög (8 szekció esetén 45°). Ekkor viszont szükség van az ϵ elektródákat elválasztó rész nyílásszögére is, mint bemeneti paraméterre, amelyet az általam használ konfigurációs fájlok is tartalmaznak.

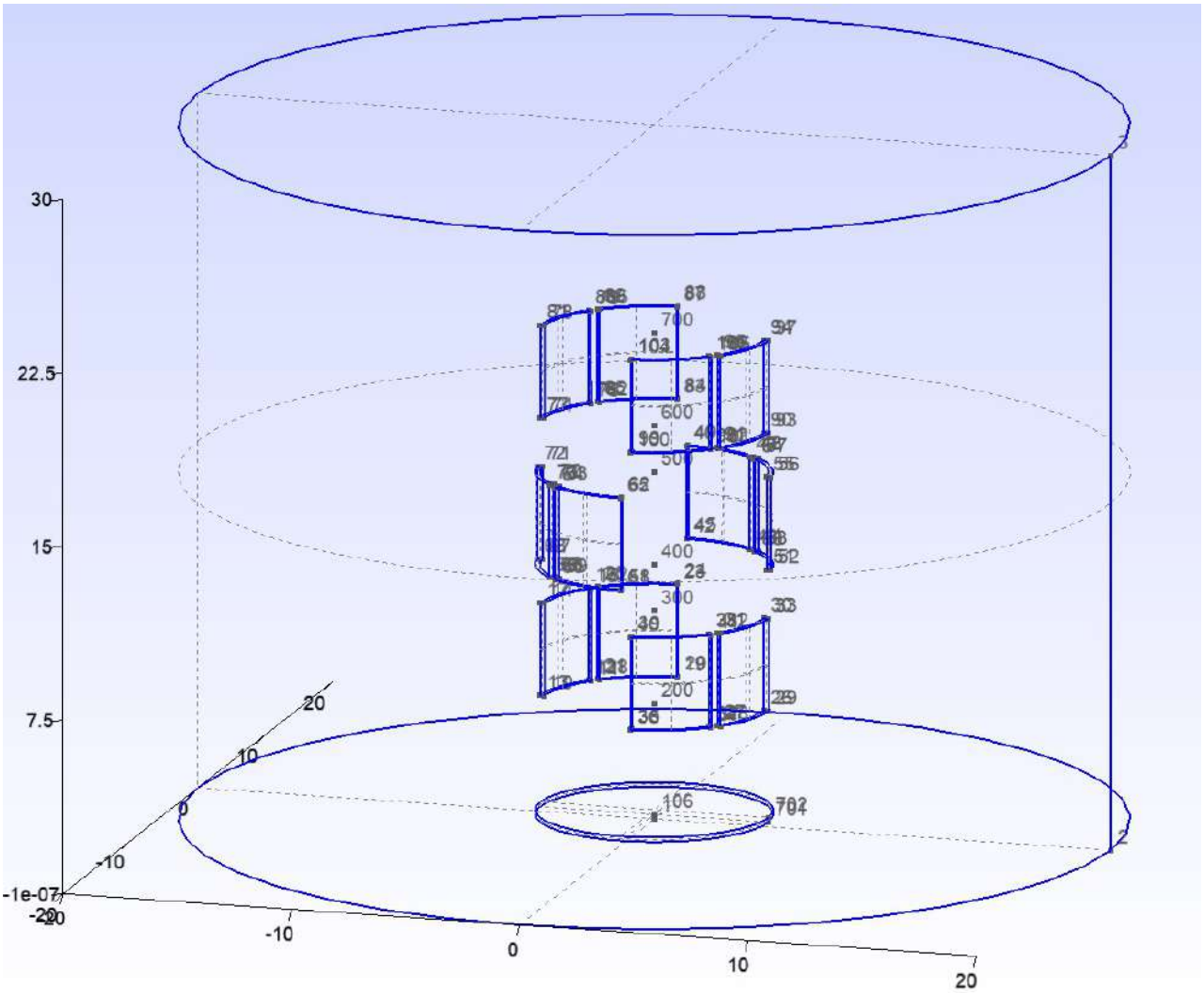
Ennek implementálásával a korábban akár órákig is eltartó geometria fálj létrehozása (∼500 soros *.geo* fájl), vagy az ilyen bonyolultabb geometriák esetén a még ennél is lassabban használható grafikus interfésszel történő nehézkes megadás néhány másodpercre csökkent, így könnyen és gyorsan adódott lehetőség különböző elrendezések vizsgálatára.

7.3.1. A geometria létrehozó kód működése

Az alábbiakban néhány példán szeretném bemutatni, hogy különböző megadott leíró paraméterek alapján milyen elektródaelrendezés jön létre GMSH-ben:

Leíró paraméterek:

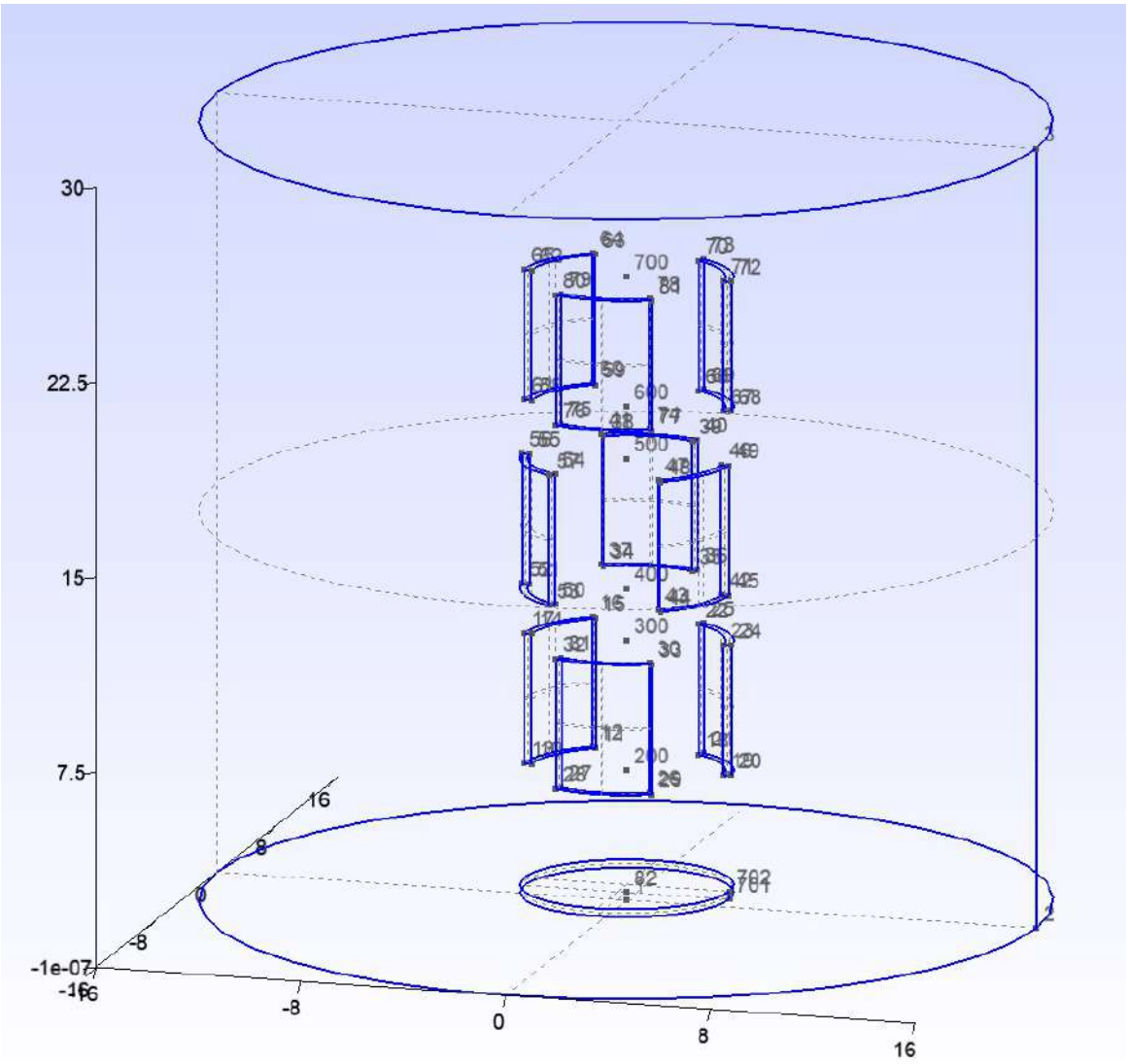
- Rp = 5
- d = 0.2
- h = 4
- N = 8
- alfa = 45
- map = [1 2 5 6]
- map_mid = [3 4 7 8]
- eps = 5



15. ábra. Első példa egy elektródaelrendezésre

Leíró paraméterek:

- Rp = 4
- d = 0.3
- h = 5
- N = 6
- alfa = 60
- map = [1 3 5]
- map_mid = [2 4 6]
- eps = 5



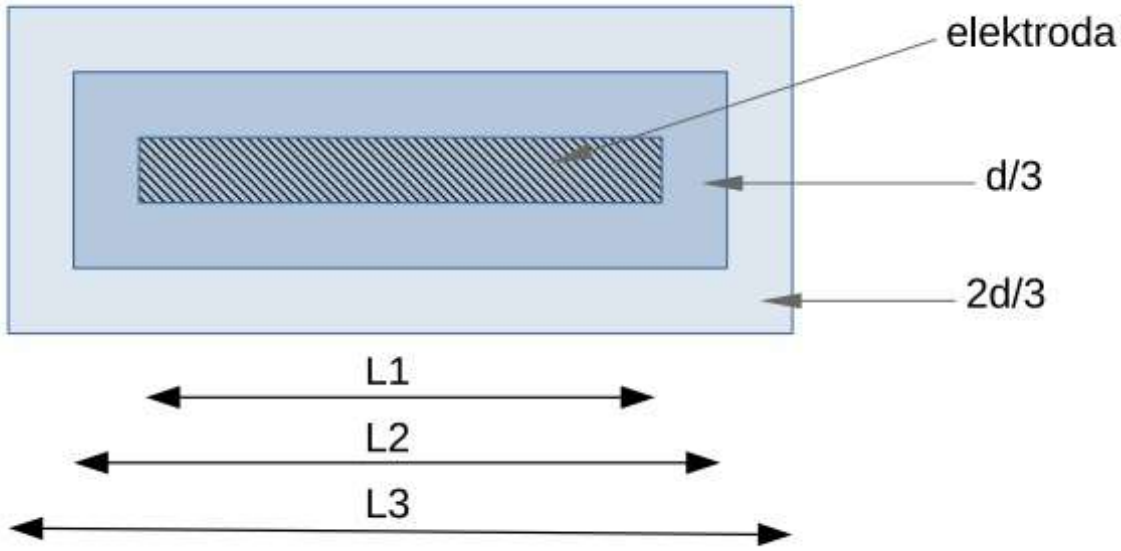
16. ábra. Második példa egy elektródaelrendezésre

7.4. Hálózási kérdések

Mint ahogy az a 4.2. és a 6.1. fejezetekben bemutatásra került, a GMSH számos lehetőséget biztosít a változó felbontású végeselem háló megvalósítására. A modellezés és szimulációnak a célja pedig, hogy a numerikus számításokkal a valóságot a lehető legjobban tudjuk közelíteni, így a végeselem háló létrehozásakor bizonyos szempontokat szem előtt kell tartani.

Fontos, hogy az elektródák környezetében, ahol nagyok a potenciálok kellően nagy felbontású hálót alkalmazzunk, annak érdekében, hogy az azokban a pontokban számított potenciál értékek között ne legyen túl nagy különbség és ezáltal a kialakuló tér is jobban közelít a valóságoshoz.

Ennek konkrét számszerűsítése az alábbi ábrán látható:



17. ábra. A végeselem háló kialakítása az elektródák környezetében

ahol:

- L1 az elektróda hossza (erre nem hálózunk, mert ekvipotenciális felület)
- L2 = L1 + 2 · d a belső zóna hossza, ahol a maximális mesh méret d/3
- L3 = L2 + 2 · d a külső zóna hossza, ahol a maximális mesh méret 2d/3

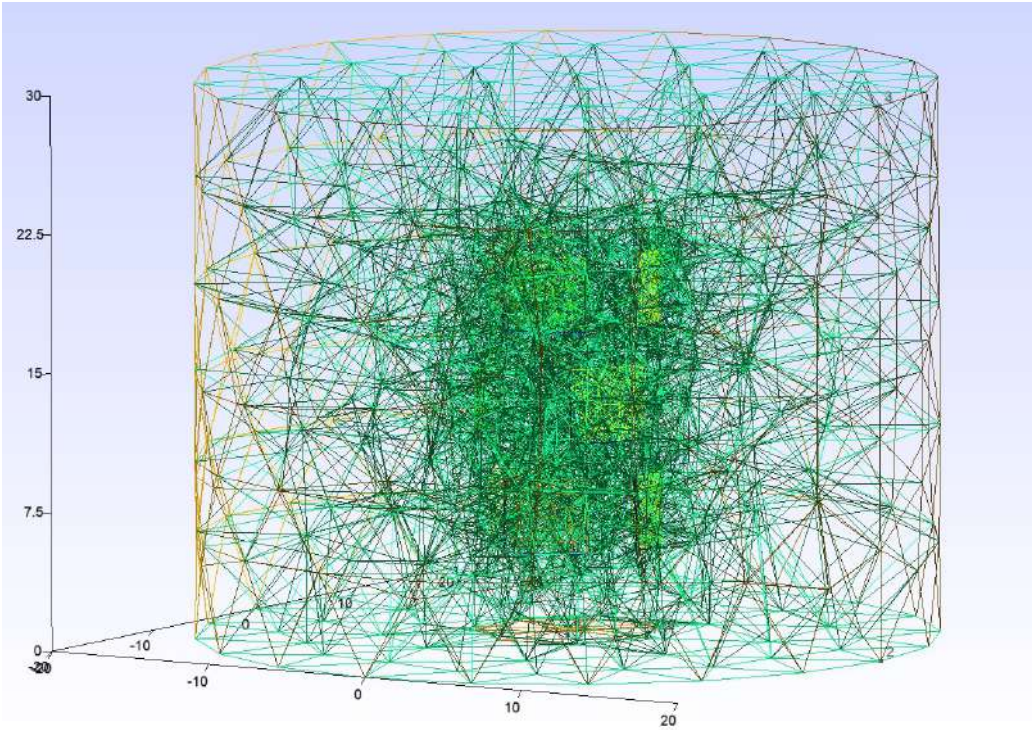
A maradék térre pedig egy értelmesen megválasztott méretet kell alkalmazni. Ugyanis, ha az egész szimulációs térre ilyen felbontású végeselem hálót alkalmaznánk, akkor feleslegesen sok elemű lenne a háló, ami miatt túl nagy lenne a belőle kinyerhető *.msh* fájl és ez jelentősen megnövélni a megoldás kiszámítási idejét.

7.4.1. Végeselem háló size field használatával

A GMSH egyik speciális lehetősége a végeselem háló testreszabására a *size field* definiálása. Ekkor lehetőségünk van különböző alakzatokat definiálni, például téglatestet, hengert, gömböt stb. Ezek nem jelennek meg geometriai alakzatként a szimulációs térben, viszont definiálható az elemi hálóméret, azaz a végeselem hálót alkotó tetraéderek maximális oldalhossza az alakzaton kívül és belül egyaránt.

Ennek hátránya azonban, hogy a GMSH nem tud több ilyen *size field*-et kezelni a számunkra megfelelő módon. Azaz, hogy például az egyes elektródákat körbevegyük ilyen alakzatokkal, vagy az elektródák közti hengeres térben két ilyen *size field* hengert hozzunk létre úgy, hogy a külső, az elektródákhoz közeli hengerben nagyobb legyen a háló felbontása, a belső térrészben, ahol elegendő lenne a kisebb felbontás, ott pedig ritkább legyen a végeselem háló.

Ennélfogva ezt a módszert csak úgy lehetne használni, hogy a az elektródákon kívüleső további térre alapértelmezetten megadott hálóméretnél egy sűrűbb hálózású térrészt alakítunk ki az elektródák között. Azonban a keletkező fájlméretet és a számítási időt szem előtt tartva, ha olyan méretet definiálunk, ami megfelelő a belső tér számítására, akkor viszont sérülnek az előző fejezetben leírtak, mégpedig, hogy az elektródák közvetlen környezetében az elektródavastagsággal arányos legyen a háló felbontása. Az így kapott ritkább háló az alábbi ábrán látható:



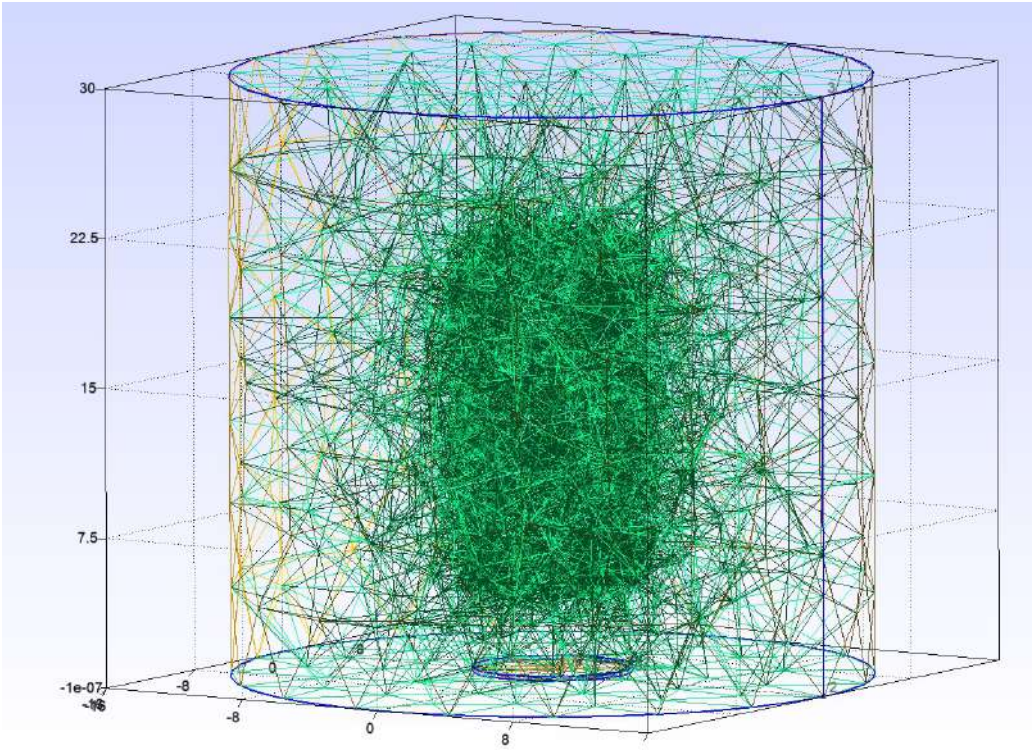
18. ábra. Példa ritkább hálóra

Ennek előnye, hogy viszonylag kevés, mindössze 2321 pont és 11687 elem alkotja a hálót, ami elég gyors megoldást tesz lehetővé, viszont a numerikus megoldás az elektródák környezetében problémás lehet.

7.4.2. Végeselem háló mesh size at point használatával

Egy másik kézenfekvő lehetőség az lenne, hogy az elektródákat alkotó pontok definiálásakor megadjuk hozzájuk a *mesh size at point* paramétert, amelyet a fent leírtak alapján d/3 nagyságúnak választunk. Ezzel viszont az a probléma, hogy nem jól definiált az, hogy ez a paraméter az adott pontnak mekkora környezetében legyen érvényes és mivel az egymással szemben elhelyezkedő elektródák mindegyikére definiáljuk, ezért befelé a tér közepe felé sem csökken az elemszám kellő mértékben.

Ez azt eredményezi, hogy bár az elektródák közvetlen környezetében megfelelő lesz a numerikus megoldás felbontása, azonban a szimulációs tér közepén feleslegesen nagy felbontású marad a háló, ami indokolatlanul nagy fájlméreteket és lassú megoldást eredményez. Az így kapott háló az alábbi ábrán látható:



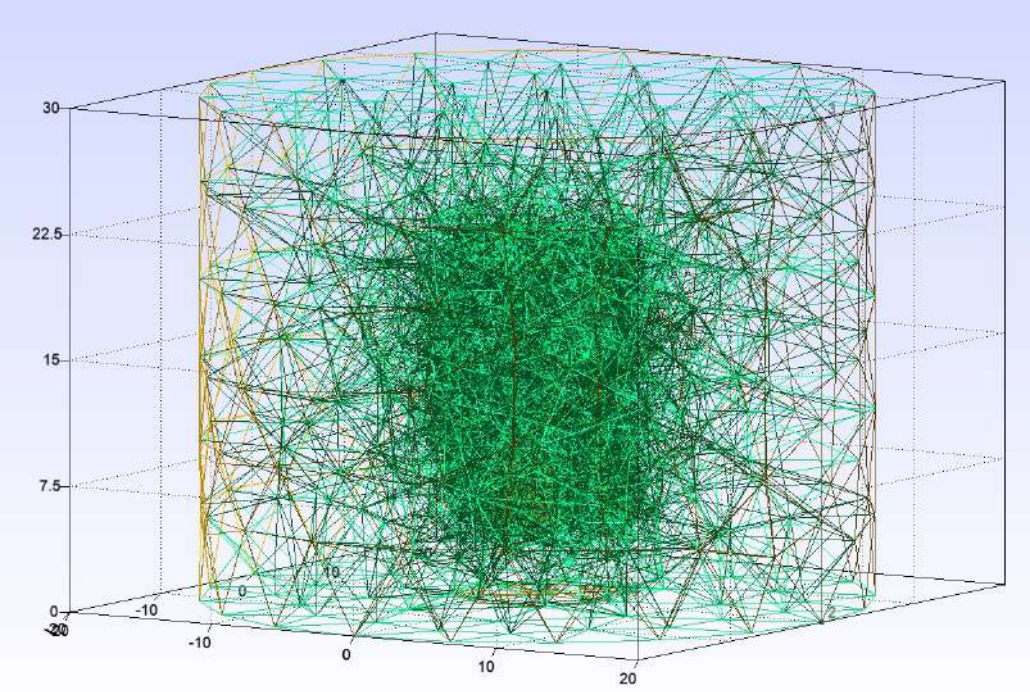
19. ábra. Példa sűrűbb hálóra

Az így kapott háló azonban túlságosan nagy méretű *.msh* fájl eredményez (235 MB), mert 819199 pontból és 4907440 tetraéder elemől áll, ami indokolatlanul hosszú számítási időt eredményez.

7.4.3. Végeselem háló transfinite használatával

Mivel az előbbi két módszer nem megfelelő végeselem hálót eredményezett, ezért indokolt volt egy harmadik módszer használata, ami a *transfinite* parancs alkalmazását jelenti. Ezzel lehetőség van a pontokat összekötő egyenesekre (vagy görbékre) megadni, hogy azt hány részre ossza fel a program, tehát, hogy hány háló elem keletkezzen ezek mentén. Az elektródák környezetében szükséges megfelelő felbontás eléréséhez tehát az azokat alkotó rövid, a vastagságukat megadó oldalegyenesek mentén ezt a paramétert 3-ra, a hosszabb határolóvonalak mentén pedig 60-ra választva ott megfelelő végeselem hálót kapunk.

Az előző módszerhez hasonlóan itt is egyúttal az elektródák közti térben is besűrűsödik a háló, azonban messze nem olyan mértékben és így még kezelhető méretű *.msh* fájlt eredményez. Az így kapott végeselem háló az alábbi ábrán látható:



20. ábra. Példa megfelelő felbontású hálóra

Az így kapott végeselem háló már optimálisnak mondható a modellt illetően, mivel a eleget tesz a megszabott hálózási feltételeknek, de még nem eredményez túl nagy számítási kapacitásigyént. Ennek 106206 pontja és 614059 eleme van, ami még teljesen jól kezelhetőnek és számíthatónak számít. Az itt bemutatott hálók néhány fontos paraméterét az alábbi táblázat összegzi.

	Ritka	Közepes	Sűrű
Pontok száma	2321	106206	819199
Elemszám	11687	614059	4907440
Beolvasási idő	2.6s	22.9s	183s

1. táblázat. Különböző hálók paraméterei

8. Részecskemozgás vizsgálata

A gyors és egyszerű geometria létrehozás és hálózás után következhet az elektródák közti térben a töltéssel rendelkező részecskék mozgásának vizsgálata. A cél annak a meghatározása, hogy a töltött, tömeggel rendelkező részecske egy adott pontból, megfelelő sebességgel indítva, milyen pályán fog haladni. A sok részecske egymástól függetlenül mozog. Ennek a folyamatnak a lépései a következők.

1. Elektromos térerősség meghatározása
2. Pálya kiszámítása a térerősségek és kezdeti állapot ismeretében
3. Utófeldolgozás a sok részecske által futott pályák analízisével

Ezen lépések között még vannak adattároláshoz kapcsolódó átalakítási feladatok, de ezek az egyes függvényeken belül lettek megvalósíva. Az egyes modulok (függvények) között fájlokon keresztül történik a kommunikáció. Ennek több célja is van. Egyrészről nem kell sorban végrehajtani az összes függvényt minden alkalommal. Másrészről az egyes részeken belül megvalósítható párhuzamosítás lehetősége is adódik. Harmadrészt a feldolgozás később is elvégezhető, akár másik számítógépen. A fent felsorolt lépéseket a következőkben szeretném részletesen bemutatni.

8.1. Elektromos térerősség meghatározása

A beolvasó kód által előállított geometriából létrejön a modell, amely megoldása a 6.3. fejezet alapján történik. Először a a határfeltételek és PDE-együtthatók beállítása történik meg, majd ezt követi a megoldás. A számítás végén itt történik a megoldás elmentése is. Egyrészt egy *.vtk* kiterjesztésű fájlba, ami a ParaView által értelmezhető szöveges xml-fájl, amely a teljes megoldást tartalmazza. Ennek célja a ParaView-ban történő ábrázolás. Másrészt pedig *.mat* kiterjesztésben is, aminek célja a további felhasználás lehetőségének biztosítása a "nyers" elmentett adatokkal. Ezt használja fel később a részecskemozgás számító modul.

8.2. A pálya kiszámítása

Miután megkaptuk az elektrosztatikai modell megoldását, azaz a háló pontjaiban a potenciálok értékét, megvizsgálhatjuk, hogy ebben a térben hogyan mozog egy töltéssel rendelkező részecske. Ez alapvetően a részecske mozgásegyenletének megoldásával történik három dimenzióban.

A részecske háromdimenziós mozgását a Descartes-koordinátarendszerben vizsgáljuk. A rá ható erőt pedig a korábban megoldott Laplace-Poisson egyenlet megoldásából kapott potenciálokból számított térerősségek segítségével számolhatjuk a következő módon:

$$m\frac{\partial^2\vec{\mathbf{r}}}{\partial t^2} = Q\vec{\mathbf{E}} = \vec{\mathbf{F}} \tag{10}$$

ahol m a részecske tömege, $\vec{\mathbf{r}}$ a részecske helyvektora (a gyorsítórács közepe azaz a $(0,0,0)$ pont az origó), t az idő, Q pedig az részecske töltése. Az $\vec{\mathbf{r}}$ helyvektort szétbontva x, y és z irányú komponenseire, az alábbi differenciálegyenlet rendszert kapjuk:

$$\begin{cases} \ddot{x} = \frac{F_x}{m} = -\frac{1}{m}\frac{\partial U}{\partial x} \\ \ddot{y} = \frac{F_y}{m} = -\frac{1}{m}\frac{\partial U}{\partial y} \\ \ddot{z} = \frac{F_z}{m} = -\frac{1}{m}\frac{\partial U}{\partial z} \end{cases} \tag{11}$$

Mivel ismert, hogy a sebességvektor a helyvektor idő szerinti deriváltja, ezért a 11. egyenletrendszert a sebességre is felírhatjuk a következő módon:

$$\begin{cases} \dot{v_x} = \frac{F_x}{m} = -\frac{1}{m}\frac{\partial U}{\partial x} \\ \dot{v_y} = \frac{F_y}{m} = -\frac{1}{m}\frac{\partial U}{\partial y} \\ \dot{v_z} = \frac{F_z}{m} = -\frac{1}{m}\frac{\partial U}{\partial z} \end{cases} \tag{12}$$

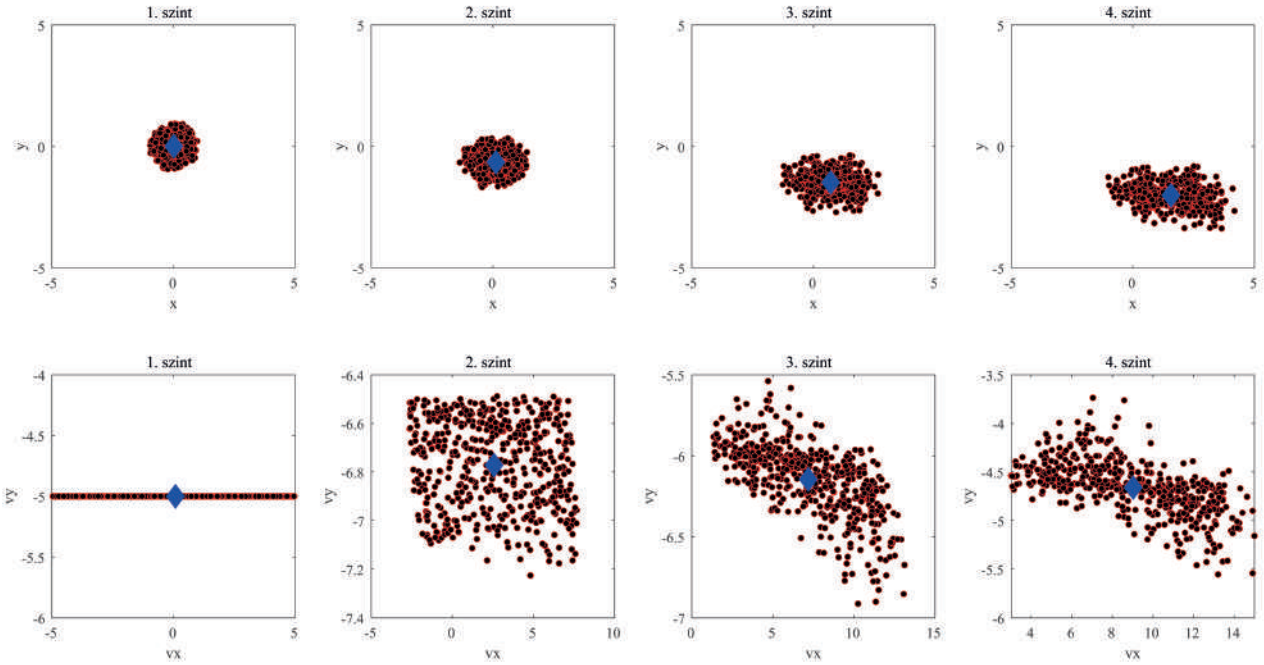
A mozgásegyenlet és a részecskék pályájának meghatározása után az adatok kimentésre kerülnek olyan módon, hogy az tartalmazza az egyes pontokban a részecskék x, y és z koordinátáit, valamint az x, y és z irányú sebességkomponenseket minden egyes időpillanatban.

8.3. A mozgás utófeldolgozása

Itt az előzőekben kiszámított részecske útvonalak elemzése történik. Most a különböző magasságú szinteken való részecske áthaladások vizsgálata történik. Ezek a szint áthaladások esetén kapott térbeli koordináták (x,y,z) és sebesség értékek (v_x,v_y,v_z) kerülnek vizsgálat alá. A jobb szemléltethetőség és valóság közelítése érdekében itt száz részecske mozgását szimuláljuk, melyeknek bizonyos szórással adott a kezdeti helyzetük valamint a kezdősebességük x, y és z koordinátája. A sebességvektor z irányú komponense a domináns a modellben, ezért annak értéke úgy került megválasztásra, hogy van egy maximum és minimum értéke, amelyek között egy random függvény adja meg végül a sebességkomponens értékét, olyan módon, hogy még a minimálisan definiált z irányú sebesség esetén is értelmes pályája legyen a mozgó részecskének. Ezzel szemben az x és y irányú kezdősebesség komponenseket kisebbnek feltételezve, ugyanakkora maximum és minimum értékkel kerültek megadásra.

A modul bemenete tehát az előző pontban megkapott egyes részecske útvonalak, kimenete pedig azok együttes ábrázolása különböző módokon.

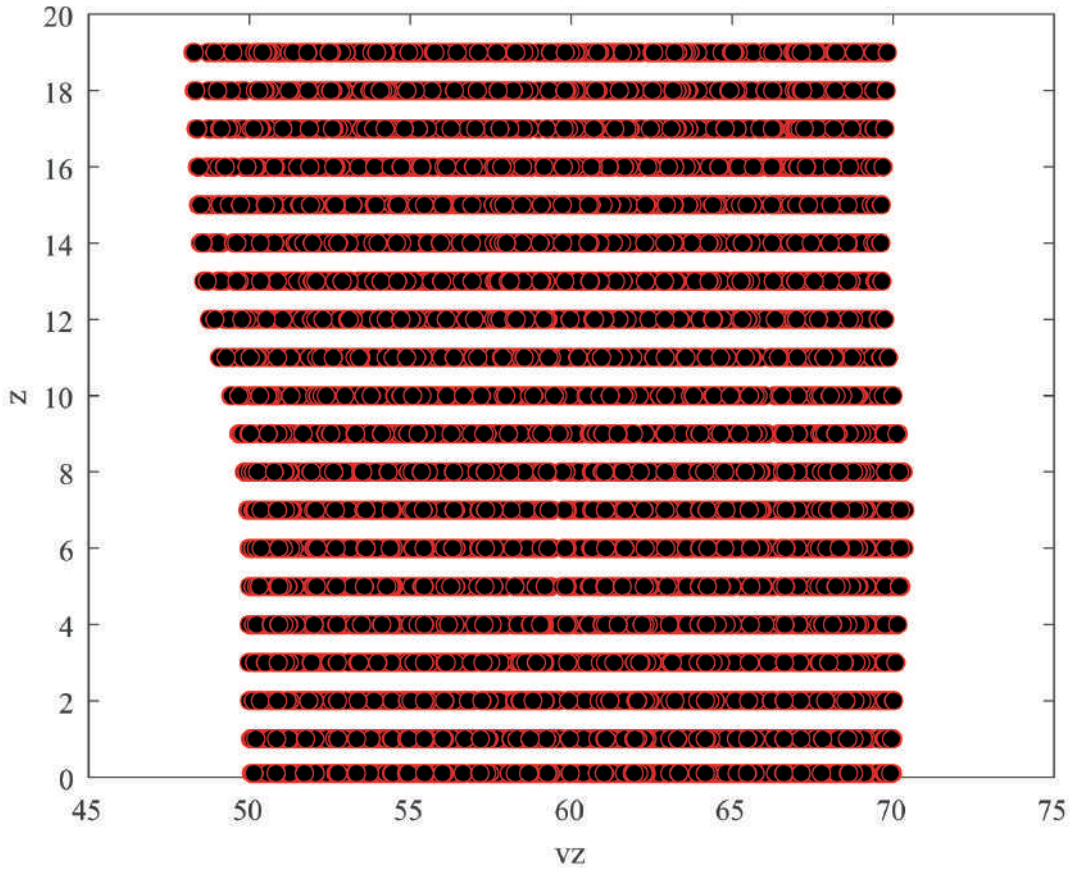
Az egyik érdekes megközelítése az ábrázolásnak a részecskék mozgásának és helyzetének alakulása a különböző szinteken való áthaladásokkor. Ennek ábrázolása az alábbi ábrán látható.



21. ábra. A részecskék mozgása a különböző szinteken való áthaladáskor

A 21. ábrán látható a részecskék mozgásának alakulása az elektródák közti térben haladás során. Felül az $x-y$ síkban való elhelyezkedésük alakülása látható, ahol az 1. szint a gyorsítórács szintje, ahonnan kilépnek a töltéssel rendelkező részkecskék, a 2.,3., illetve a 4. szint pedig az egyes elektródapárok szintjei. Alul pedig a részecskék x és y irányú sebességkomponenseinek iránya látható ugyanilyen felosztásban. Mindkét esetben az ábrán kék jelöléssel szerepel a ponthalmaz átlaga, az x és y irányú komponensek számtani közepének számításából.

Ez azonban csak az $x-y$ síkbeli ábrázolást mutatja be, ezért célszerű a z irányú mozgás valamilyen módon való ábrázolása is. Az alábbi ábrán látható a részecskék z irányú sebesség-komponensének alakulása az elektródák közti térből kifelé haladva.



22. ábra. A függőleges sebességkomponens eloszlása az egyes szinteken

8.4. Különböző esetek vizsgálata

Az előzőekben csak egy példa került bemutatásra a töltéssel rendelkező részecskék mozgásának utófeldolgozásának lehetőségeire. Azonban lehetőség van bizonyos szisztémák, elgondolások mentén vizsgálatokat végezni. Erre lehet egy példa, hogy egy egyszerű gyorsítást vizsgálunk, tehát hogy az űreszközt a haladási irányában szeretnénk gyorsítani, oldalirányban pedig nem. Ezt úgy lehet elérni, hogy a fúvóka különböző szintjein elhelyezkedő elektródáknak azonos potenciált választunk, valamint ezen potenciálok értéke a gyorsítórácstól (ami $0V$ potenciálú) kifelé haladva fokozatosan csökkennek. Itt fontos megjegyezni, hogy ezen potenciálok értéke negatív (kivéve az első szinten), mert pozitív töltéssel rendelkező, gyorsítani kívánt részecskét feltételezünk, tehát a potenciálok abszolút értéke növekszik kifelé haladva, viszont az értékük negatív. Az első szinten azért nem probléma a pozitív potenciál, mert alapból feltételezünk egy z irányú kezdősebességet a kilépő részecskék számára, amivel könnyedén áthalad az első szintig tartó enyhén lassító téren, utána viszont folyamatos gyorsítás érhető el, továbbá így nagyobb lehet a potenciálkülönbség az első és második szint elektródái között és így a gyorsító hatás is.

Erre egy konkrét példa tehát, hogyha az elektródaszinteket alulról felfelé, azaz a gyorsítórácstól kifelé (A,B,C) módon indexeljük, akkor egy lehetséges potenciálkiosztás ennek vizsgá-

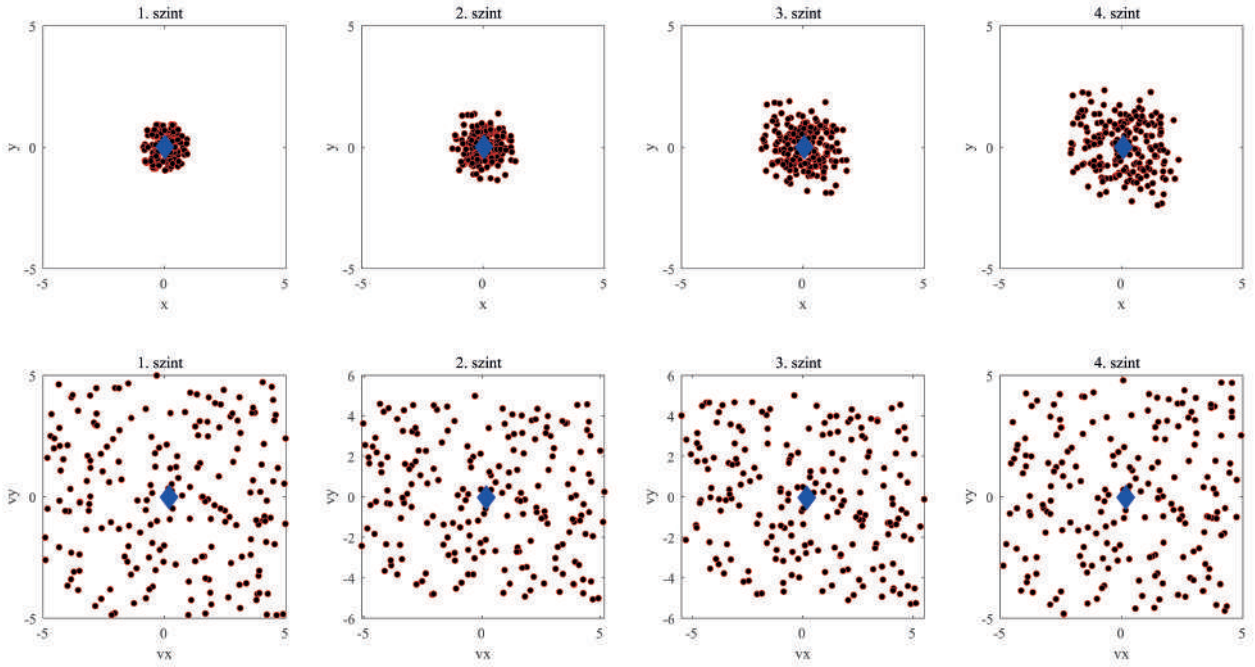
latára a következő:

	Elektródapár egyik tagjának potenciálja	Elektródapár másik tagjának potenciálja
A szint	$100V$	$100V$
B szint	$-200V$	$-200V$
C szint	$-UH$	$-UH$

2. táblázat. Elektródák potenciálkiosztása egyszerű gyorsítás esetén

Ahol UH értéke a változtatható paraméter, amelynek függvényében vizsgáljuk a kilépő-töltéssel rendelkező részecskék mozgását. Ennek a paraméternek megválasztható értékei és a vizsgáladó esetek például $UH = [250, 500, 750, 1000]V$ potenciálok esetén érdekesek.

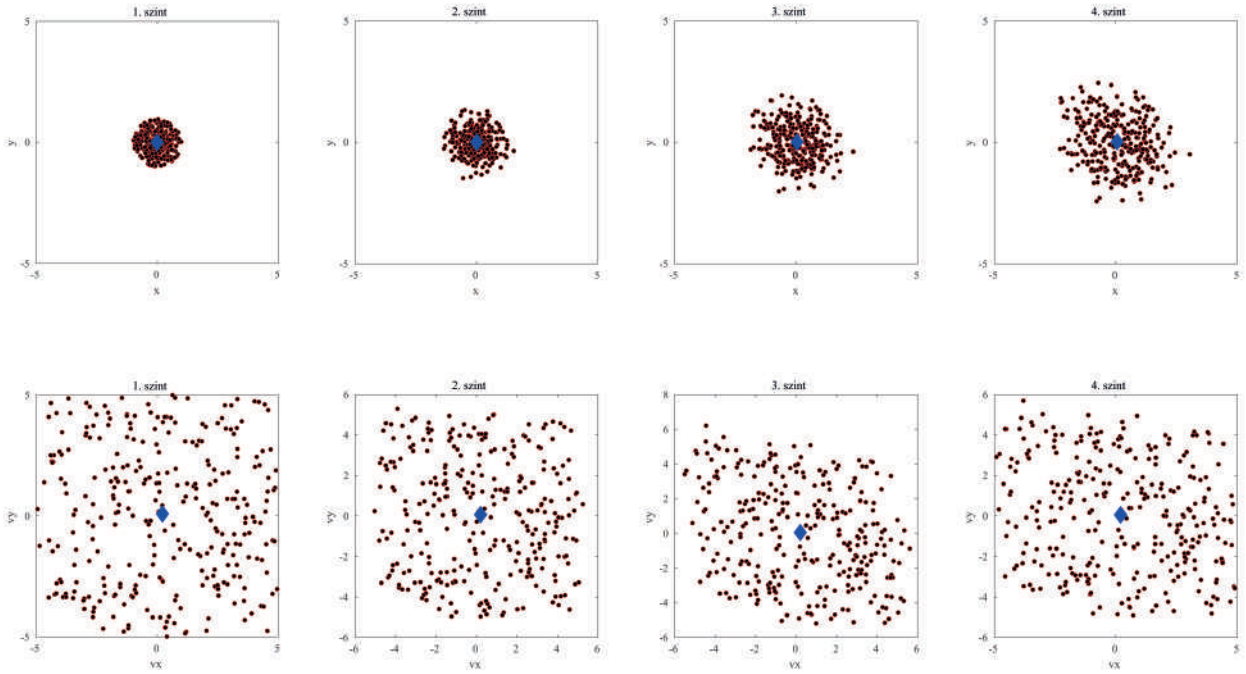
Ezen esetek vizsgálata előtt még egy érdekes lehetőséget is érdemes bemutatni itt. Mégpedig, hogy egyfajta inicializációs futtatásként megnézzük, hogy mi történik, akkor, ha a C szint nincs meghajva. Ekkor az elektródák potenciálja a numerikus számításokban NaN értéket kap, amitől Neumann peremfeltétel marad definiálva rajtuk, tehát úgy viselkednek, mintha ott sem lennének a szimulációs térben. Az így kapott eredmények az alábbi ábrán láthatóak:



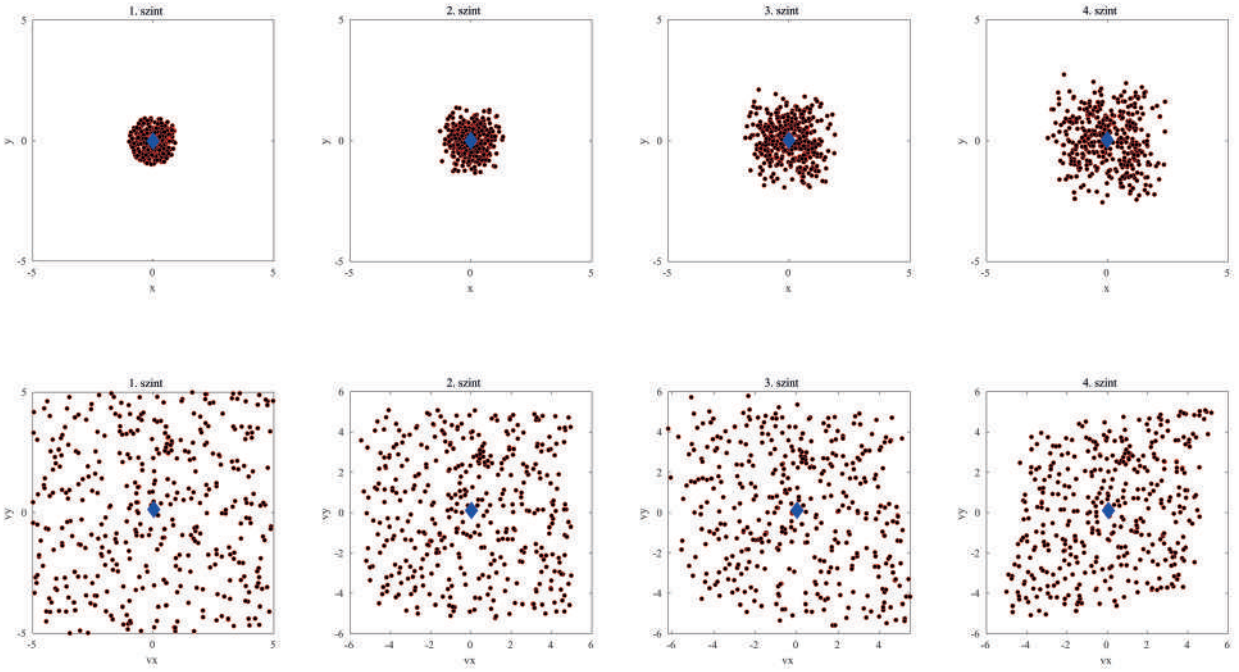
23. ábra. Egyszerű gyorsítás C szinten lévő elektródák meghajtása nélkül

Az ábra értelmezése megegyezik a 21. ábráéval, a töltéssel rendelkező részecskék helyzetét és sebességét láthatjuk rajta az ott leírtak alapján.

Ezt követően megvizsgálhatjuk, hogy hogyan alakul a különböző szinteken a részecskék mozgása, ha a C szinten lévő elektródapár is meg van hajtva, ez az alábbi ábrákon látható.

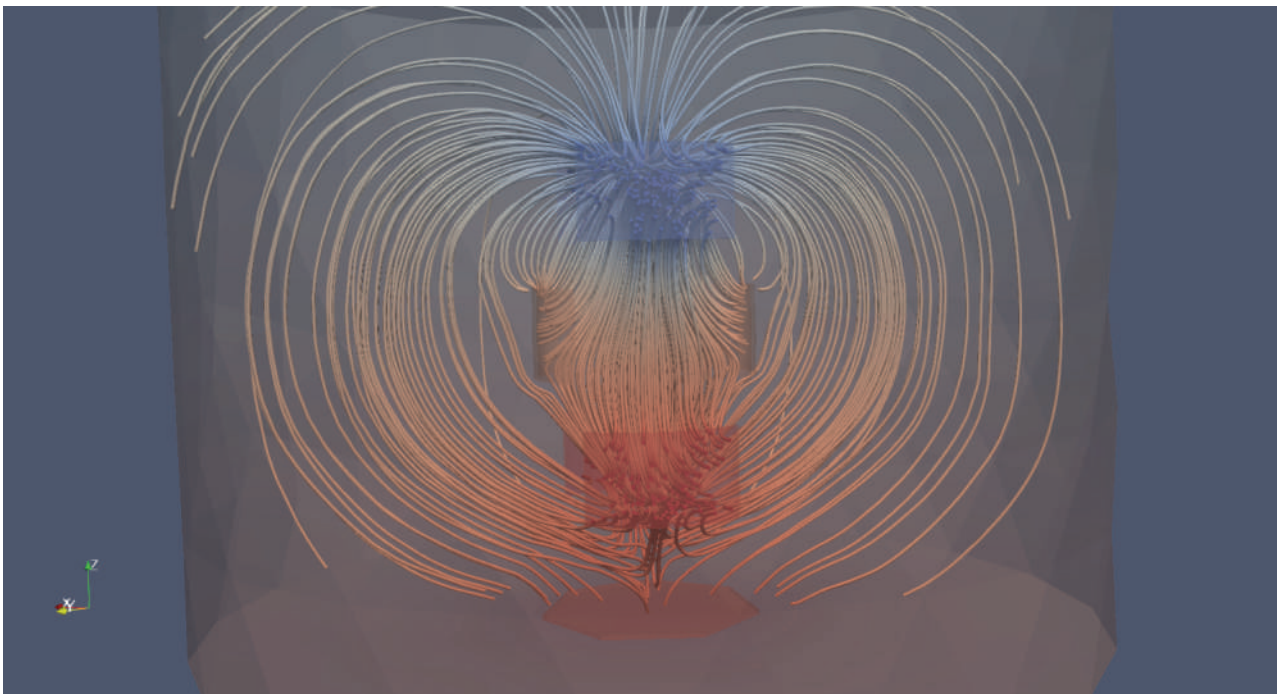


24. ábra. Egyszerű gyorsítás C szinten $-250V$ elektródapotenciállal



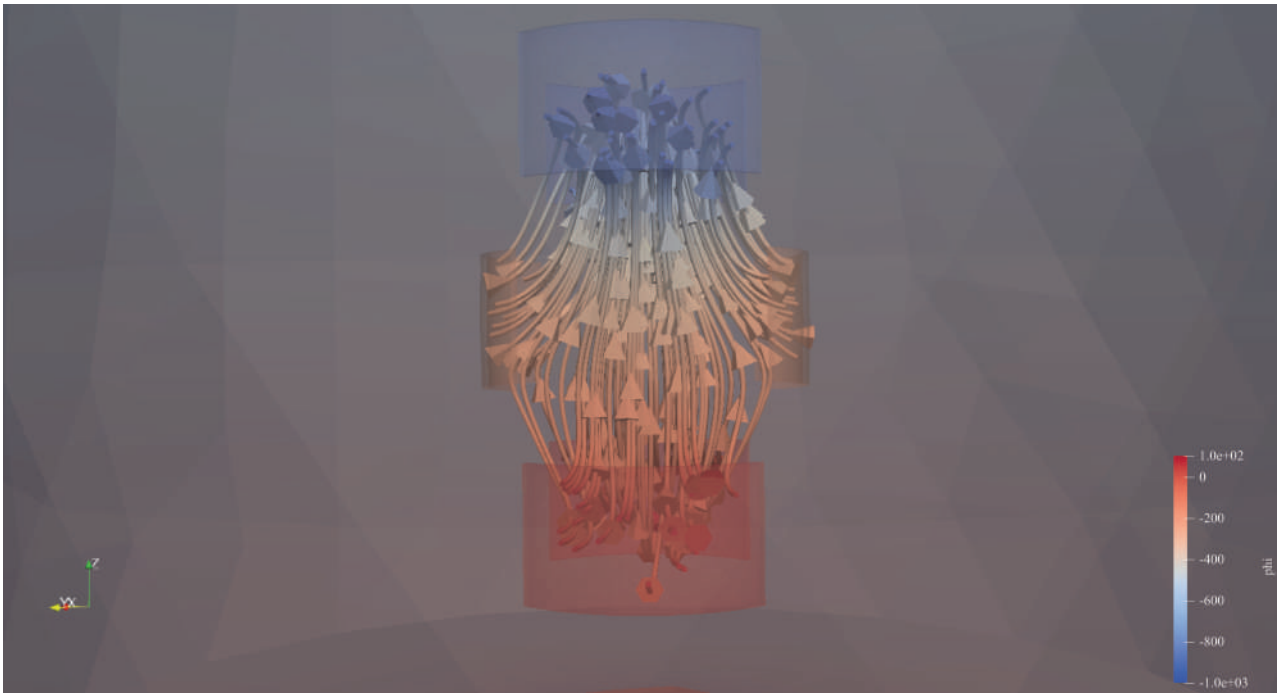
25. ábra. Egyszerű gyorsítás C szinten $-500V$ elektródapotenciállal

Megfigyelhető, hogy a 23., a 24. és a 25. ábrák jellegre nagyon hasonlóak egymáshoz, ami megfelel az elvárásoknak, hiszen itt egy egyszerű z irányú gyorsítást vizsgáltunk, aminek az $x - y$ síkú mozgásra és elhelyezhókedésre nincs hatással. Ennek szemléletesebb értelmézésre szolgál, ha megvizsgáljuk a kialakú erővonalakat és Paraview segítségével ábrázoljuk azt.



26. ábra. Erővonalak ábrázolása egyszerű gyorsítás esetén C szinten $-1000V$ potenciállal

Még ennél is szemléletesebb képet kaphatunk, ha csak az elektródák közti belső térben (ahol a töltéssel rendelkező részecskék mozognak) ábrázoljuk a kialakuló elektromos teret, ami az alábbi ábrán látható.



27. ábra. Belső tér ábrázolása egyszerű gyorsítás esetén C szinten $-1000V$ potenciállal

A 27. ábrán látható, hogy az elektródák közti belső térben, különösen annak középső részén, ahonnan a szimuláció során a részecskék haladtak, a kialakuló tér nagyrészt csak z irányú komponensekkel rendelkezik és a pozitív töltéssel rendelkező részecskéket kifelé gyorsítja, ami

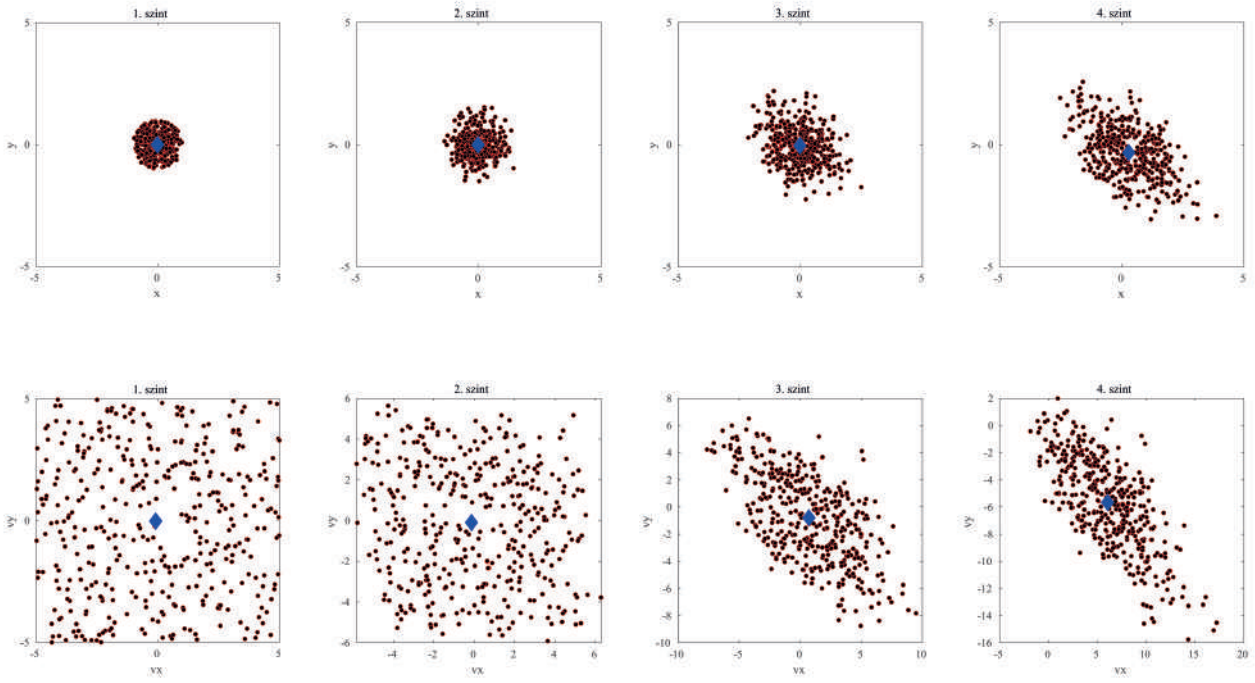
megmagyarázza, hogy ennek az esetnek miért nincs jelentős hatása az $x-y$ síkú mozgásra, ami a 23., a 24. és a 25. ábrákon volt látható.

Ezt követően egy olyan esetet szeretnék bemutatni, ahol hasonlóképpen egy kifelé történő gyorsítás tapasztalható, azonban az elektródapotenciálok nem ilyen fokozatos gyorsítást eredményeznek, hanem hogy az A és B szintek között történik egy jelentős gyorsítás, amit a C szinten lévő elektródák potenciálja hátráltat. Itt az elektródák közti térből kilépő töltéssel rendelkező részecskék z irányú sebességének alakulása az érdekes. Az ehhez tartozó potenciálértékek az alábbi táblázatban szerepelnek.

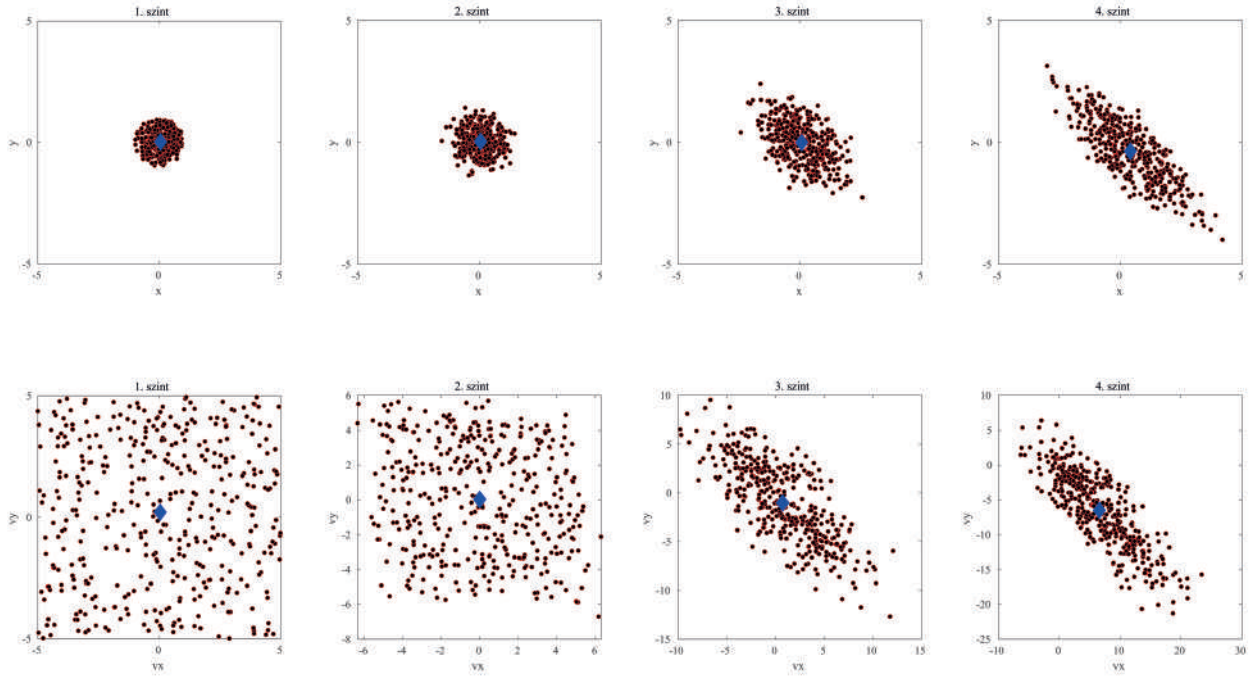
	Elektródapár egyik tagjának potenciálja	Elektródapár másik tagjának potenciálja
A szint	$500V$	$500V$
B szint	$-UH$	$-UH$
C szint	$-500V$	$-500V$

3. táblázat. Elektródák potenciálkiosztása aszimmetrikus gyorsítás esetén

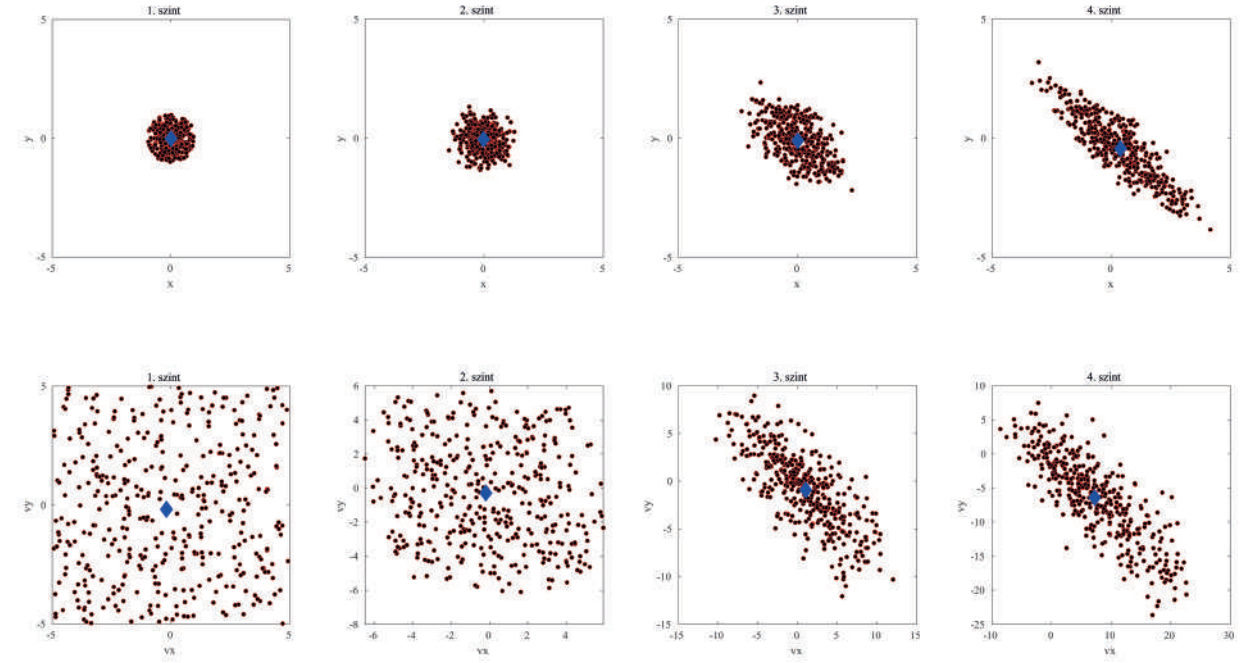
Itt UH értéke a $[250, 750, 1000]V$ potenciálértékeket veszi fel. Az így kapott eredmények a részecskék $x-y$ síkú mozgására vonatkozóan az alábbi ábrákon láthatóak.



28. ábra. Erővonalak ábrázolása aszimmetrikus gyorsítás esetén B szinten $-250V$ potenciállal

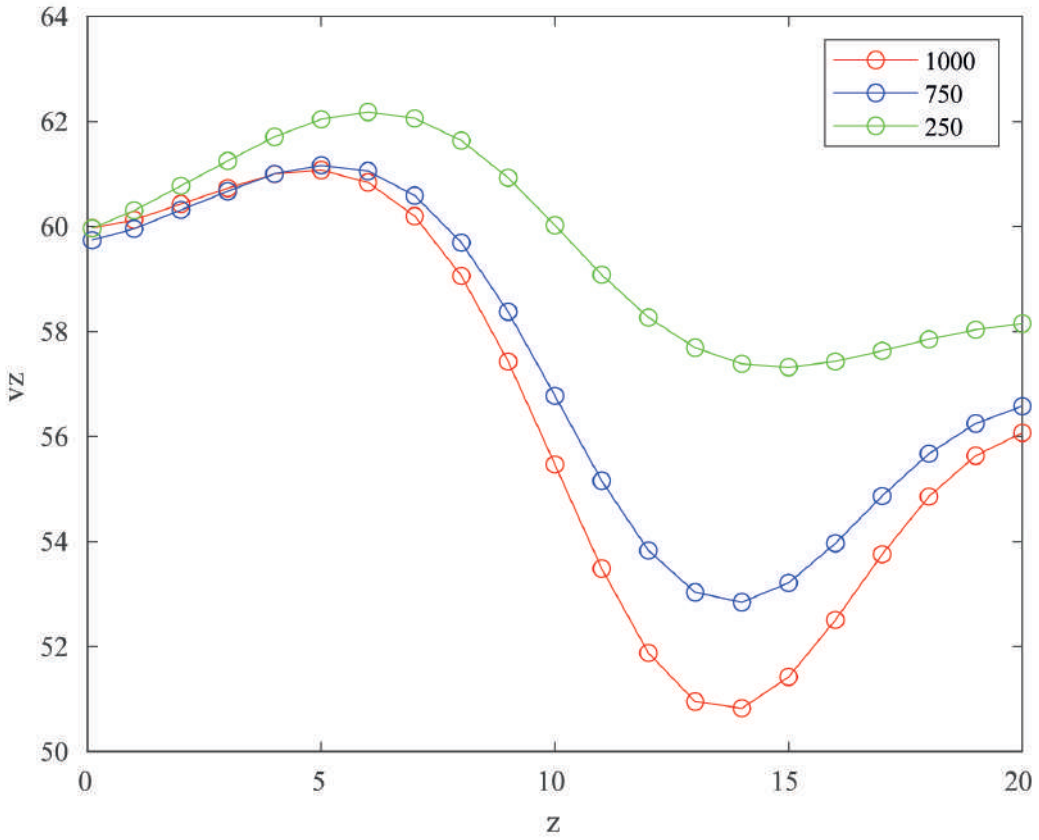


29. ábra. Erővonalak ábrázolása aszimmetrikus gyorsítás esetén B szinten $-750V$ potenciállal



30. ábra. Erővonalak ábrázolása aszimmetrikus gyorsítás esetén B szinten $-1000V$ potenciállal

Itt a 28., a 29. és a 30. ábrákon megfigyelhető az előző esettel ellentétben, hogy az elrendezés amellett, hogy z irányban végez gyorsítást, a kilépő részecskék szimmetrikus elhelyezkedését is módosítja az elektródák közti térben haladás során. Az alábbi ábrán látható a részecskék z irányú sebessége a gyorsítórácstól mért távolság függvényében és erre a B szint elektródapotenciáljának hatása.



31. ábra. Részecskék z irányú sebessége a távolság függvényében aszimmetrikus gyorsítás esetén

Megfigyelhető, hogy ebben az esetben, mivel az első és utolsó szint potenciálja állandó értéken van tartva, így a B szinten minél nagyobb a potenciál abszolút értéke (előjelesen C szint potenciáljánál kisebb), annál kisebb lesz a kilépő részecskék z irányú végesebessége, valamit, ahogy a 28., a 29. és a 30. ábrákon is látható, a részecskék kezdetben szimmetrikus elhelyezkedését is eltorzítja.

9. Összegfoglalás

TDK munkám során kisméretű műholdak esetén használható ionhajtómű modellezésével és szimulációjával foglalkoztam. A szimuláció egy elektrosztatikai probléma megoldását jelentette végeselem módszerrel. Munkám során az így használatos végeselem módszer lehetőségeinek vizsgálatával és fejlesztésével foglalkoztam behatóbban.

A modellalkotás és szimuláció során MATLAB és GMSH környezeteket használtam és az ezek által biztosított lehetőségeket vizsgáltam. A MATLAB által biztosított PDEToolbox számos remek lehetőséget biztosít végeselemű szimulációkra, azonban egy jelentős hátránya, hogy a szimulációs tér teljes egészét egyforma sűrűségűen hálózza, ami azt eredményezi, hogy a számunkra kevésbé fontos részeken feleslegesen nagy felbontásban keletkezik a megoldás, ami feleslegesen nagy számítási kapacitást igényel. Ezzel szembenben a GMSH specifikusan ilyen végeselem hálók optimális létrehozására készült és nagyon jól használható az adott problémához a legjobban illeszkedő háló létrehozására. Számos lehetőséget biztosít a felhasználó számára, hogy tetszés szerint testreszabható hálót készítsen.

Azonban a GMSH-ben létrehozott háló bár elmenthető olyan kiterjesztésben, amit a MATLAB tudna kezelni, egy bizonyos nem túl nagy fájlméret esetén a MATLAB ezt már nem tudja egyszerűen beolvasni. Ezért munkám első lépése során létrehoztan egy függvénykönyvtárat MATLAB-ban, amivel az így készült végeselem háló beolvasható és ebből létrehozza a PDEToolbox számára szükséges geometriát és pdemodel-t. Viszont a GMSH-nek további hátránya, hogy a kívánt elrendezés geometriájának megadása mind szkriptelve, mind pedig a grafikus interfész használatával nehézkes és lassú folyamat. Mivel a különböző elektródaelrendezések vizsgálata is fontos része volt a munkámnak, ezért készítettem egy további függvénykönyvtárat MATLAB-ban, ami lehetőséget biztosít egy előre deifinált paraméterlista megadásával tetszőleges elektródaelrendezésű hengeres geometria létrehozására GMSH-ben.

Ezután következhetett az elektrosztatikai probléma megoldása MATLAB használatával. Erre a PDEToolbox-ot használtam majd az így kapott eredmények további feldolgozása is MATLAB-ban történt. A végeselem háló pontjaiban megkapott potenciál értékekből meghatározható a töltéssel rendelkező részecskék mozgása ebben a térben a mozgásegyenlet háromdimenziós megoldásával. Ezt követően pedig a részecskék mozgásának utófeldolgozását vizsgáltam, egész pontosan azt, hogy hogyan alakul a mozgó részecskék helyzete és sebessége a gyorsítórácsból kilépve és eközben a haladási irányukban a sebességük hogyan változik.

Ábrák jegyzéke

Táblázatok jegyzéke

10. Irodalomjegyzék

Hivatkozások

[1] Matthew N.O. Sadiku, Computational Electromagnetics with MATLAB, Taylor & Francis, 2019, ISBN 978-1-138-55815-1

[2] Vida Krisztián, Kétdimenziós elektrosztatikai probléma egyéni megoldása, Önálló laboratórium beszámoló, 2021

[3] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[4] MATLAB, 2021. 9.10.0.1649659 (R2021a), Natick, Massachusetts: The MathWorks Inc.

[5] Á.Makara,A. Reichardt and L. Csurgai-Horváth, "Visualization and simulation of ion thrusters possibly usable by small satellites", HSPACE-2020-FP-39, Budapest, 2020.

[6] K. Vida, A. Reichardt, "Limits of ion pathline control using electric field", Proc. of 20th International IGTE Symposium, Graz, 2022.

[7] Goetz D, Katz. I, Fundamentals of Electric Propulsion, Wiley, 2008

[8] Jahn R. G., Physics of Electric Propulsion, McGraw-Hill, 1968

függelék A Beolvasó MATLAB kód

```
file = fopen('model_v2.msh');
l=fgetl(file);                % read first record
count = 1;
while ~strcmp(l,'$EndElements')      % loop until find the magic
   record
  l=fgetl(file);
  count = count + 1;
end
A = strings(count,1);
count = 2;
file = fopen('model_v2.msh');
l=fgetl(file);
A(1) = l;
while ~strcmp(l,'$EndElements')      % loop until find the magic
   record
  l=fgetl(file);
  A(count)=l;
  count = count + 1;
end

count = 1;
while ~strcmp(l,'$Nodes')      % loop until find the magic record
  l=A(count);
  count = count + 1;
end

nodeData = str2num(A(count));
numNodes = nodeData(2);
nodes = zeros(3,numNodes);
l = (A(count));
k = count;
i = 1;
space = 0;

while ~strcmp(l,'$EndNodes')      % loop until find the magic
   record
  count = count + 1;
```

```
  B = isspace(l);
  b = sum(B(:) == 1);
  space = space + b;
  k = k + 1;
  if space == 2
    nodes(:,i) = str2num(A(count-1));
    l = (A(count));
    space = 0;
    i = i + 1;
  else
    l = (A(count));
    space = 0;
  end
end

count = count + 1;
k = k + 1;
l=(A(count));

count = count + 1;
k = k + 1;
l=(A(count));

elementsData = str2num(A(count));
numElements = elementsData(2);
elements = zeros(5,numElements);


while ~strcmp(l,'$EndElements')      % loop until find the magic
   record
  B = isspace(l);
  b = sum(B(:) == 1);
  space = space + b;
  k = k + 1;
  count = count + 1;
  l = convertStringsToChars(l);
  if space == 3 && l(1) == '3' && l(2) == ' ' && l(5) == '4' && l
     (4) == ' ' && l(6) == ' '
    i = 1;
    while ~strcmp(l,'$EndElements')
```

```
        count = count + 1;
        l = A(count);
        sor = convertStringsToChars(l);
        if sor(1) == '3' && sor(2) == ' ' && sor(5) == '4' && sor
           (4) == ' ' && sor(6) == ' '
            count = count + 1;
            l = A(count);
        else
        elements(:,i) = str2num(A(count-1));
        k = k + 1;
        i = i + 1;
        end
    end
  else
    l = A(count);
    space = 0;
  end
end

elements( :, all(~elements,1) ) = [];
elements(1,:) = [];

model = createpde;
geometryFromMesh(model,nodes,elements);
pdegplot ( model , 'EdgeLabels', 'on', 'FaceLabels', 'on',"
   FaceAlpha",0.5);
```

függelék B Geometria konstruáló kód

```
run adatok.m
fi_deg = 360/N;
fi_rad = 2*pi/N;
alfa_rad = alfa*2*pi/360;
eps_rad = eps*2*pi/360;
fileID = fopen('ures.geo','w');
fprintf(fileID, '%s %s\n', '// Gmsh project created on', string(
   datetime('now')));
fprintf(fileID, '%s\n', 'SetFactory("OpenCASCADE");');

pointID = 1;
lineID = 1;
surfaceID = 1;
volumeID = 1;

%szimulacios ter (nagy henger) letrehozasa
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', pointID,
   ') = {', 0, 0, 0, 1.0, '};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %d, %d, %s\n', 'Circle(',
   lineID, ') = {', 0, 0, 0, Rp*4, 0, '2*Pi};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Curve Loop(', lineID, ') = {',
   lineID, '};');
lineID = lineID + 1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Plane Surface(', surfaceID, ') =
    {', surfaceID, '};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d, %d, %d%s\n  %s%d%s \n%s\n', 'Extrude {',
   0, 0, 30, '} {', 'Curve{', surfaceID, '};', '}');
surfaceID = surfaceID +1;

lineID = lineID + 1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Curve Loop(', lineID, ') = {',
```

```
   lineID, '};');
surfaceID = surfaceID +1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Plane Surface(', surfaceID, ') =
    {', surfaceID, '};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s\n', 'Surface Loop(',volumeID,') = {1, 2,
   3};');
volumeID = volumeID + 1;

% koriv kozeppontok
kp1ID = 200;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp1ID, ')
    = {', 0, 0, 5, 1.0, '};');

kp2ID = 300;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp2ID, ')
    = {', 0, 0, 5+h, 1.0, '};');

kp3ID = 400;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp3ID, ')
    = {', 0, 0, 5+h+2, 1.0, '};');

kp4ID = 500;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp4ID, ')
    = {', 0, 0, 5+h+2+h, 1.0, '};');

kp5ID = 600;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp5ID, ')
    = {', 0, 0, 5+h+2+h+2, 1.0, '};');

kp6ID = 700;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d, %.1f%s\n', 'Point(', kp6ID, ')
    = {', 0, 0, 5+h+2+h+2+h, 1.0, '};');
```

```
pointID = 10;
lineID = 10;
surfaceID = 10;
surfID = 5;

%terbeli alakzatok letrehozasa
for i = 1:3
   if i == 2
    for k = 1:length(map_mid)
        %pontok
        %alul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos((map_mid(k)-1)*
           fi_rad), (Rp-d)*sin((map_mid(k)-1)*fi_rad), 5+(i-1)*(h
           +2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos(alfa_rad - eps_rad
           + (map_mid(k)-1)*fi_rad), (Rp - d)*sin(alfa_rad -
           eps_rad + (map_mid(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3,
           '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -Rp*cos(alfa_rad - eps_rad + (
           map_mid(k)-1)*fi_rad), Rp*sin(alfa_rad - eps_rad + (
           map_mid(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -Rp*cos((map_mid(k)-1)*fi_rad),
           Rp*sin((map_mid(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3, '};
           ');
        pointID = pointID + 1;

        %felul
        fprintf(fileID, '%s\n', '//+');
```

```
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -(Rp - d)*cos((map_mid(k)-1)*
            fi_rad), (Rp-d)*sin((map_mid(k)-1)*fi_rad), h+5+(i-1)
            *(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -(Rp - d)*cos(alfa_rad - eps_rad
            + (map_mid(k)-1)*fi_rad), (Rp - d)*sin(alfa_rad -
            eps_rad + ((map_mid(k)-1))*fi_rad), h+5+(i-1)*(h+2), d
            /3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -Rp*cos(alfa_rad - eps_rad + (
            map_mid(k)-1)*fi_rad), Rp*sin(alfa_rad - eps_rad + (
            map_mid(k)-1)*fi_rad), h+5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -Rp*cos((map_mid(k)-1)*fi_rad),
            Rp*sin((map_mid(k)-1)*fi_rad), h+5+(i-1)*(h+2), d/3, '
            };');
        pointID = pointID + 1;

        %vonalak
        %alul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
            , ') = {', pointID-8, kp1ID + (i-1)*200, pointID-7, '
            };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
             {', pointID-7, pointID-6, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
            , ') = {', pointID-6, kp1ID + (i-1)*200, pointID-5, '
            };');
```

```
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-5, pointID-8, '};');
        lineID = lineID + 1;

        %fuggoleges
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-8, pointID-4, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-7, pointID-3, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-6, pointID-2, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-5, pointID-1, '};');
        lineID = lineID + 1;

        %felul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
           , ') = {', pointID-4, kp2ID + (i-1)*200, pointID-3, '
           };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-3, pointID-2, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
           , ') = {', pointID-2, kp2ID + (i-1)*200, pointID-1, '
           };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
```

```
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-1, pointID-4, '};');
        lineID = lineID + 1;

        %curve loop and surface
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-12, lineID-11, lineID-10,
            lineID-9, '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-12, lineID-7, -(lineID-4)
           , -(lineID-8), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-11, lineID-6, -(lineID-3)
           , -(lineID-7), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-10, lineID-5, -(lineID-2)
           , -(lineID-6), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-9, lineID-8, -(lineID-1),
            -(lineID-5), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-4, lineID-3, lineID-2,
           lineID-1, '};');
        surfaceID = surfaceID + 1;


    end
else
    for k = 1:length(map)
```

```
        %pontok
        %alul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos((map(k)-1)*fi_rad),
            (Rp-d)*sin((map(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3, '
           };');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos(alfa_rad - eps_rad
           + (map(k)-1)*fi_rad), (Rp - d)*sin(alfa_rad - eps_rad
           + (map(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -Rp*cos(alfa_rad - eps_rad + (map
           (k)-1)*fi_rad), Rp*sin(alfa_rad - eps_rad + (map(k)-1)
           *fi_rad), 5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -Rp*cos((map(k)-1)*fi_rad), Rp*
           sin((map(k)-1)*fi_rad), 5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;

        %felul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos((map(k)-1)*fi_rad),
            (Rp-d)*sin((map(k)-1)*fi_rad), h+5+(i-1)*(h+2), d/3,
           '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
           ', pointID, ') = {', -(Rp - d)*cos(alfa_rad - eps_rad
           + (map(k)-1)*fi_rad), (Rp - d)*sin(alfa_rad - eps_rad
           + (map(k)-1)*fi_rad), h+5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
```

```
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -Rp*cos(alfa_rad - eps_rad + (map
            (k)-1)*fi_rad), Rp*sin(alfa_rad - eps_rad + (map(k)-1)
            *fi_rad), h+5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%.8f, %.8f, %d, %.2f%s\n', 'Point(
            ', pointID, ') = {', -Rp*cos((map(k)-1)*fi_rad), Rp*
            sin((map(k)-1)*fi_rad), h+5+(i-1)*(h+2), d/3, '};');
        pointID = pointID + 1;

        %vonalak
        %alul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
            , ') = {', pointID-8, kp1ID + (i-1)*200, pointID-7, '
            };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
             {', pointID-7, pointID-6, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
            , ') = {', pointID-6, kp1ID + (i-1)*200, pointID-5, '
            };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
             {', pointID-5, pointID-8, '};');
        lineID = lineID + 1;

        %fuggoleges
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
             {', pointID-8, pointID-4, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
             {', pointID-7, pointID-3, '};');
```

```
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-6, pointID-2, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-5, pointID-1, '};');
        lineID = lineID + 1;

        %felul
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
           , ') = {', pointID-4, kp2ID + (i-1)*200, pointID-3, '
           };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-3, pointID-2, '};');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Circle(', lineID
           , ') = {', pointID-2, kp2ID + (i-1)*200, pointID-1, '
           };');
        lineID = lineID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d%s\n', 'Line(', lineID, ') =
            {', pointID-1, pointID-4, '};');
        lineID = lineID + 1;

        %curve loop and plane surface
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-12, lineID-11, lineID-10,
            lineID-9, '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-12, lineID-7, -(lineID-4)
           , -(lineID-8), '};');
```

```
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-11, lineID-6, -(lineID-3)
           , -(lineID-7), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-10, lineID-5, -(lineID-2)
           , -(lineID-6), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-9, lineID-8, -(lineID-1),
            -(lineID-5), '};');
        surfaceID = surfaceID + 1;
        fprintf(fileID, '%s\n', '//+');
        fprintf(fileID, '%s%d%s%d, %d, %d, %d%s\n', 'Curve Loop('
           , surfaceID, ') = {', lineID-4, lineID-3, lineID-2,
           lineID-1, '};');
        surfaceID = surfaceID + 1;


    end
   end
end

%surface (plane surface helyett, mert nem sik)
for surf = 10:(surfaceID-1)
    fprintf(fileID, '%s\n', '//+');
    fprintf(fileID, '%s%d%s%d%s\n', 'Surface(', surfID, ') = {',
       surf, '};');
    surfID = surfID + 1;
end

%gyorsito elektroda letrehozasa
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %.1f, %.2f%s\n', 'Point(', pointID
   , ') = {', 0, 0, d, d/3, '};');
pointID = pointID + 1;
```

```
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %.1f, %d, %d, %s\n', 'Circle(',
   1000, ') = {', 0, 0, d, Rp, 0, '2*Pi};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Curve Loop(', 1000, ') = {',
   1000, '};');
lineID = lineID + 1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Plane Surface(', 1000, ') = {',
   1000, '};');
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d, %d, %.1f%s\n  %s%d%s \n%s\n', 'Extrude {',
    0, 0, d, '} {', 'Curve{', 1000, '};', '}');
surfaceID = surfaceID +1;
surfID = surfID + 1;

lineID = lineID + 1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Curve Loop(', 1002, ') = {',
   1002, '};');
lineID = lineID + 1;
surfaceID = surfaceID +1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d%s\n', 'Plane Surface(', 1002, ') = {',
   1002, '};');
surfID = surfID + 1;
surfaceID = surfaceID +1;
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%d%s%d, %d, %d%s\n', 'Surface Loop(',volumeID,
   ') = {', 1000, 1001, 1002, '};');
volumeID = volumeID + 1;

%surface loop
for volID = 3:((surfID-7)/6 + 2)
    fprintf(fileID, '%s\n', '//+');
    fprintf(fileID, '%s%d%s%d, %d, %d, %d, %d%s\n', 'Surface
       Loop(', volID, ') = {', 5 + (volID-3)*6, 6 + (volID-3)*6,
       7 + (volID-3)*6, 8 + (volID-3)*6, 9 + (volID-3)*6, 10 + (
       volID-3)*6, '};');
    volumeID = volumeID + 1;
```

```
end

%egyetlen terfogatelem letrehozasa
x = 1:1:volID;
str = sprintf('%.0f, ',x);
str = str(1:end-2);
fprintf(fileID, '%s\n', '//+');
fprintf(fileID, '%s%s%s\n', 'Volume(1) = {', str, '};');

fclose(fileID);
```