



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

NUMERIKUS ANALÍZIS TANSZÉK

Pont a poliéderben feladat robusztus és hatékony megoldása

Témavezető:

Bálint Csaba, Fábián Gábor
egyetemi adjunktus

Szerző:

Vajda Fábiusz
programtervező informatikus MSc

Budapest, 2024

Tartalomjegyzék

1. Absztrakt	3
2. Bevezetés	4
3. Elmélet	5
3.1. 2D eset	5
3.1.1. Sugármetszés	5
3.1.2. Körülfordulási szám	6
3.1.3. Négyesfa (Quadtree)	6
3.2. 3D eset	7
3.2.1. Sugárkövetés	7
3.2.2. Nyolcasfa (Octree)	7
3.2.3. Fél-él adatszerkezet	8
4. Pont a sokszögben	9
4.1. Sokszög generálás	9
4.1.1. Konvex burkok összekötése	9
4.1.2. Vízszintes tengely szerinti összekötés	10
4.1.3. Radiális sokszög generálása	11
4.2. A Körülfordulási szám algoritmus gyorsítása	12
4.3. Legközelebbi oldal vizsgálata	13
4.3.1. Hegyesszőgek közelében lévő pontok lekezelése	14
4.3.2. Térfelosztás	15
4.4. Az implementáció	15
5. Pont a poliéderben	17
5.1. Poliéder generálás	17
5.2. Sugármetszés határesetei	18
5.2.1. A sugár és a lap egy síkon van	18

5.2.2.	A sugár egy élen megy át	18
5.2.3.	A sugár egy csúcson megy át	19
5.3.	Legközelebbi lap vizsgálata	20
5.3.1.	Térfelosztás	21
5.4.	A program	21
6.	Eredmények	23
6.1.	2D	23
6.1.1.	Változó oldalszám	23
6.1.2.	Változó pontszám	24
6.2.	3D	25
6.2.1.	Változó lapszám	25
6.2.2.	Változó pontszám	27
6.2.3.	Változó lapszám és pontszám	30
7.	Összegzés	33
7.1.	Továbbfejlesztési lehetőségek	33
	Irodalomjegyzék	35

1. fejezet

Absztrakt

A pont a sokszögben (Point In Polygon, PIP) probléma egy jól ismert geometriai feladat. Egy adott n oldalú sokszög esetén k pontot kell különböző osztályokba sorolnunk aszerint, hogy azok a sokszög belsejében, a sokszög határán vagy azon kívül helyezkednek el. Az elterjedt megoldások, mint a sugárkövetésre és a körülfordulási számra épülő algoritmusok, jellemzően minden egyes pont osztályozásához a sokszög összes oldalát számításba veszik, ami nagy számú pont esetén hatékonysági problémákat vet fel.

Hasonló a feladat a probléma háromdimenziós általánosítása esetén is, amikor a térbeli pontokat aszerint osztályozzuk, hogy egy n háromszöglapból álló poliéder belsejébe, határára vagy külsejébe esnek. A sugárkövetés algoritmus itt is alkalmazható, a probléma viszont ugyanaz, mint a kétdimenziós esetben: nagy számú pont esetén az osztályozás túlzottan költséges lehet.

A dolgozat célja, egyrészt az eddigi sugárkövetés és körülfordulási szám algoritmusok fejlesztése és bizonyos határesetek kezelése, másrészt egy hatékonyabb megoldási módszer kidolgozása mind a két-, mind a háromdimenziós esetben, térfelosztó adatszerkezetek alkalmazásával. Ezek az adatszerkezetek lehetővé teszik az előfeldolgozási fázisban a tér szisztematikus felosztását, amely révén az algoritmus kiértékelési szakasza jelentős gyorsulást érhet el. Cél még az előbb említett térfelosztásnak a különböző leállási feltételeinek a vizsgálata. A dolgozat továbbá célul tűzi ki az új megoldások összehasonlítását a meglévő algoritmusokkal, azok hatékonyságát vizsgálva két- és háromdimenzióban.

2. fejezet

Bevezetés

A pont a poligonban és a pont poliéderben feladat jelenlegi megoldásainak az a gyengesége, hogy nagy mennyiségű lekérdezési pont vizsgálatánál nem működnek elég hatékonyan. Ezen módszerekkel az elsődleges probléma az, hogy minden pontot új feladatként fognak fel, elfelejtve mindent, amit az előző pont vizsgálatánál megtudtak a sokszögről vagy testről. Erre lehet egy jobb megoldás ha valamilyen térfelosztó módszer segítségével valamennyi információt elmentünk a testről és ez alapján gyorsabban feldolgozzuk a szükséges pontokat.

A dolgozatban sok az átfedés a A new point containment test algorithm based on preprocessing and determining triangles című cikkel [1], ennek a legfőbb oka az, hogy nem tudtunk ennek a cikknek a létezéséről, viszont a mi dolgozatunk jobban kitér a különböző leállási feltételek vizsgálatára a térfelosztásnál.

A következő fejezetben bevezetek pár szükséges elméleti fogalmat a feladattal kapcsolatban. Az ezt követő két fejezetben részletezem a fejlesztéseimet kettő, majd három dimenzióban. Az utolsó fejezetben pedig az eredményeket fogom megmutatni és az ezekből leszűrhető tanulságokat részletezem.

3. fejezet

Elmélet

Ebben a fejezetben áttekintjük azokat a fogalmakat, adatszerkezeteket, algoritmusokat, amelyeket a későbbiekben használunk.

3.1. 2D eset

A pont a sokszögben probléma (Point in Polygon, PIP röviden), annak az eldöntése hogy egy pont egy zárt sokszög belsejében, vagy azon kívül helyezkedik el.

Egy gyakorlati felhasználása a problémának annak az eldöntése, hogy egy földrajzi koordináta benne van-e egy bizonyos országban vagy államban.

3.1.1. Sugármetszés

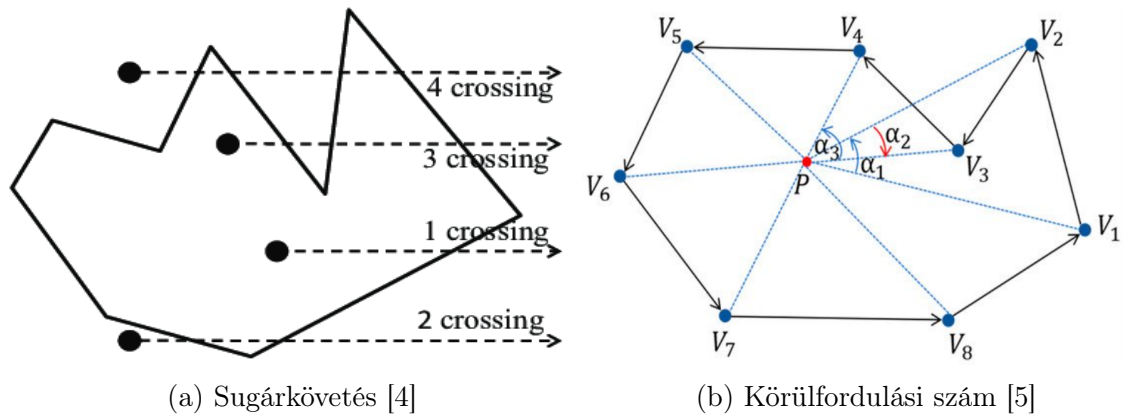
A problémára az egyik elterjedt megoldás a Sugármetszés (Raycasting), aminek az a lényege, hogy a vizsgált pontból egy félegyenest húzunk és megszámloljuk, hogy páros vagy páratlan sokszor metszette-e el az oldalakat [2]. Mivel minden nem csúcsebeli oldal metszésnél változik, hogy a sugár kívül vagy belül van a sokszögben, és mivel az utolsó oldal metszés után mindenképpen kívül van, ezért tudjuk, hogy páratlan metszés esetén belül, páros metszés esetén kívül volt a vizsgált pont.

A gyors számítás érdekében egy egyszerű sugárral szokás számolni, például az x tengellyel párhuzamos vektor mentén. Ez lehetővé tesz egy másik gyorsítást is. Egy x tengely menti félegyenes nem tud olyan oldalt elmetezni, ahol mindkettő végpont a pont fölött van, vagy mindkettő végpont a pont alatt van. Így a metszeteket csak azokkal az oldalakkal kell kiszámolni, ahol a két végpont a vizsgált pont ellentétes

oldalain helyezkedik el. Mivel egy vízszintes egyenes mindenképpen elmetesz egy olyan szakaszt aminek a végpontjai ellentétes oldalon helyezkednek el, ezért csak azt kell vizsgálni, hogy az egyenes-szakasz metszés a vizsgált pont melyik oldalán történik, hogy ezáltal tudjuk, hogy a félegyenes is metszi-e a szakaszt. (3.1a. ábra).

3.1.2. Körülfordulási szám

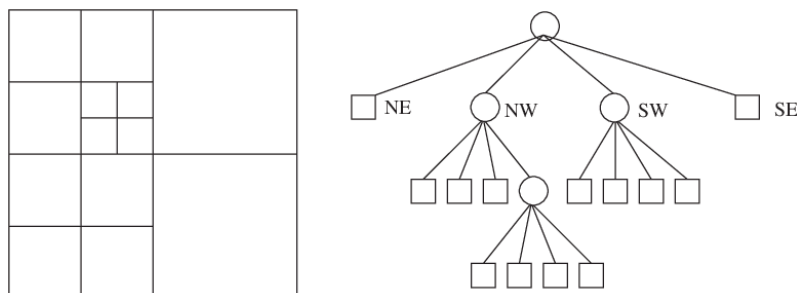
Egy másik megoldás a PIP problémára a körülfordulási szám (Winding number) meghatározásán alapuló algoritmus. Itt a vizsgált pontból egyeneseket húzunk az oldalak végpontjaihoz, majd az így kapott szögeket előjelesen összeadjuk [3]. Zárt és nem önmetsző sokszög esetén két lehetséges eredményt kaphatunk: 2π , ha pont a sokszögön belül van, vagy 0, ha a sokszögön kívül. Szemléletesen itt azt vizsgáljuk, hogy az oldalak a pont körül forognak vagy a pont mellett. (3.1b. ábra).



3.1. ábra. Pont a sokszögben feladat eddigi megoldásai

3.1.3. Négyesfa (Quadtree)

A négyesfa (quadtree) gyökere a sokszögnek a befoglaló doboza lesz, ami egy olyan x és y tengellyel párhuzamos oldalú téglalap, aminek a határai a pontoknak a minimum és maximum x illetve y koordinátái. A gyökeret felosztjuk négy részre, ezek lesznek a gyökér gyerekei. Ezután a kapott cellákat szintén felosztjuk négy részre és így folytatjuk a fa építését egészen addig, ameddig áganként nem teljesül egy leállási feltétel [6]. Ez a leállási feltétel lehet például egy adott oldalszám a cellában, de lehet csak egy bizonyos mélység is, hogy ne menjen a végtelenségig. Ez egy előfeldolgozási folyamat, hogy később lekérdezésnél logaritmikus időben meg lehessen találni egy ponthoz közeli lokális geometriát. (3.2. ábra)



3.2. ábra. Négyes fa [7]

3.2. 3D eset

A probléma a három dimenzióban a pont a poliéderben feladat, ami nevéből adódóan azt dönti el, hogy a pont egy tömör test belsejében vagy azon kívül helyezkedik el.

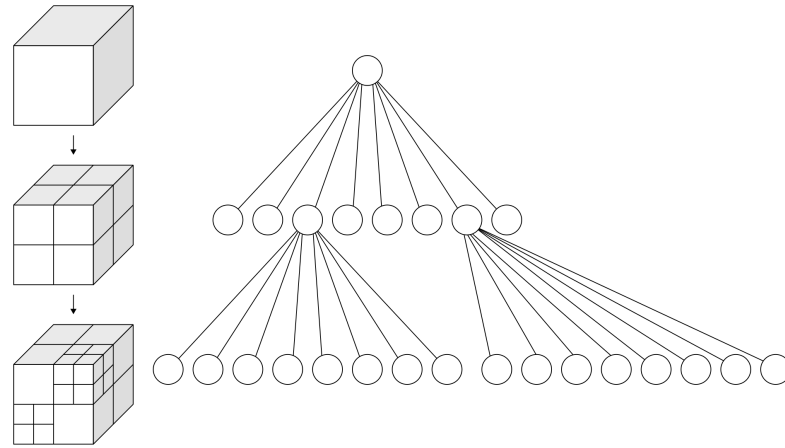
Egy gyakorlati felhasználása a problémának az ütközés vizsgálatnál van, ugyanis ekkor az egyik testnek vehetjük tetszőlegesen sűrű pontfelosztását és ellenőrizhetjük, hogy ezekből a pontokból benne van-e bármelyik a másik testben, ha igen, akkor ütközés történt.

3.2.1. Sugárkövetés

A Sugárkövetés ebben az esetben is használható. Ekkor azt vizsgáljuk, hogy a pontból húzott félegyenes páros vagy páratlan sokszor metszi el a lapokat. Páratlan sok metszés esetén a testen belül, páros sok metszés esetén a testen kívül van a pont. Ebben az esetben is egyszerűbbek a számítások ha a sugarat például az x tengellyel párhuzamosan húzzuk. Továbbá, ebben az esetben nem kell azokat a lapokat vizsgálni, ahol a lap mindegyik csúcsa ugyanazon az oldalán helyezkedik el a pontnak y vagy z koordináta szerint. A hatékony félegyenes-háromszög metszések meghatározásához a Möller-Trumbore algoritmust használtam [8].

3.2.2. Nyolcasfa (Octree)

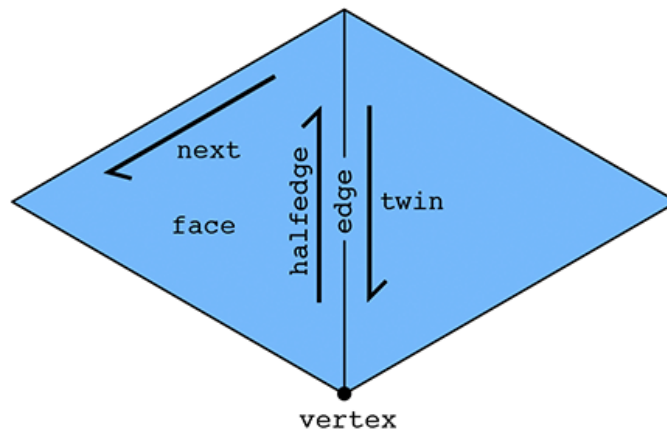
Az Octree vagy nyolcas fa a Quadtree három dimenziós megfelelője. Ahelyett, hogy a síkot osztanánk négy részre itt a teret osztjuk nyolc részre [9]. Az előny szintén az, hogy előfeldolgozással logaritmus időben lehet lekérdezni a ponthoz közeli lokális geometriát. (3.3. ábra)



3.3. ábra. Nyolcas fa [10]

3.2.3. Fél-él adatszerkezet

A Fél-él (Half-edge) adatszerkezet egy hatékony tárolása a test oldalainak, ami-
ben a szomszédsági viszonyok könnyen ellenőrizhetők [11]. A nevéből adódóan fél-
éleket tárol el, ami azt jelenti, hogy az élnek csak az egyik laphoz tartozó része jelenti
az egységet. A fél-élekben el van tárolva a lap, amihez tartoznak, a párjuk, az előző
fél-él és a következő fél-él, valamint az él végpontjai (3.4. ábra).



3.4. ábra. Fél-él adatszerkezet [12]

4. fejezet

Pont a sokszögben

4.1. Sokszög generálás

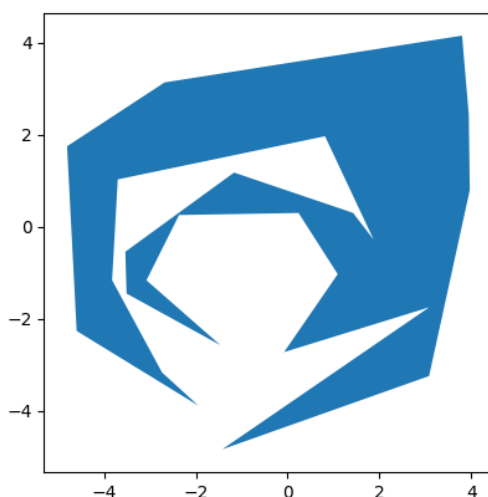
A robusztus mérésekhez tetszőleges oldalszámú véletlen sokszögek generálása volt szükséges. Ehhez nem elég véletlen pontokat generálni és azokat tetszőlegesen összekötni, mert így könnyen keletkezhetnek önmetszések. Bizonyos algoritmusok így is működnének, de önmetsző esetben nem olyan triviális, hogy mi a poligon külseje és belseje, továbbá a gyakorlati felhasználás szempontjából is kevésbé van jelentősége ennek az esetnek. A gyakorlatban általában nincsenek önmetsző poligonok, ezért ezzel az esettel nem foglalkoztam. A sokszögek tárolásához egy tömböt célszerű használni, amiben a csúcspontok vannak felsorolva pozitív forgásirányban.

A programban három fő módszert használtam a sokszögek készítésre, mindhárom esetben generáltam $n \in \mathbb{N}$ db véletlen pontot egy véges térrészben, majd a módszertől függő módon kötöttem ezeket össze.

4.1.1. Konvex burkok összekötése

Az első módszer a konvex burkok [13] összekötése. Nevéből adódóan itt a legenerált pontoknak vettem a konvex burkát [14], majd a maradék pontoknak a konvex burkát, és ezt ismételtam addig, ameddig volt még pont. Mivel a konvex burkok tartalmazzák az összes maradék pontot, ezért nincsenek önmetszések a különböző rétegek között. A keletkezett burkokat egy rekurzív algoritmussal kötöttem össze kívülről belülré haladva. Az eddigi csúclánc végpontjait az algoritmus összeköti a következő konvex burok végpontjaival egy „hidat” formálva. Itt előfordulhat, hogy a híd önmetszésben van a belső burokkal, ebben az esetben a belső burok egy szom-

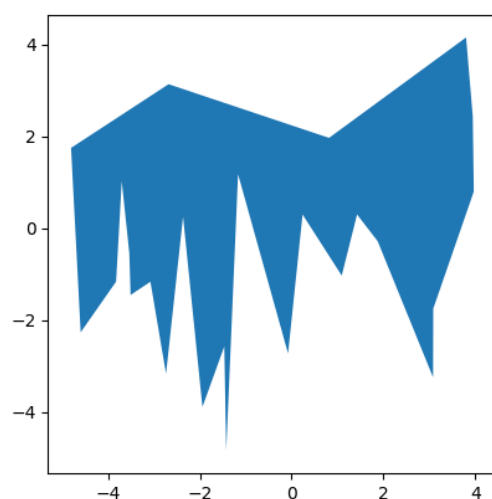
szédos oldalával próbálunk meg hidat formálni, és ezt addig próbáljuk, ameddig találunk egy hidat, ami nincs metszésben semmivel. Ezután a kezdeti és a végpontot úgy mozgatjuk el, hogy a belső burkon legyenek, hogy folytatható legyen a rekurzív algoritmus. Meggondolható, hogy a legbelső burok csúcsainak a körülfordulási iránya minden lépésben megfordul, hiszen a hidak mentén az oldalak felkanyarodnak az új burokra, majd elindulnak azon abba az irányba ahonnan jöttek. A legbelső pontoknál előfordulhat, hogy nem maradt elég pont egy konvex alakzathoz, tehát csak egy vagy kettő pont található az utolsó szinten, ilyenkor is össze lehet kötni a maradék pontokat az eddigi poligon végpontjaival az előzőekhez hasonlóan (4.1. ábra).



4.1. ábra. Konvex burkok összekötése 25 pontra

4.1.2. Vízszintes tengely szerinti összekötés

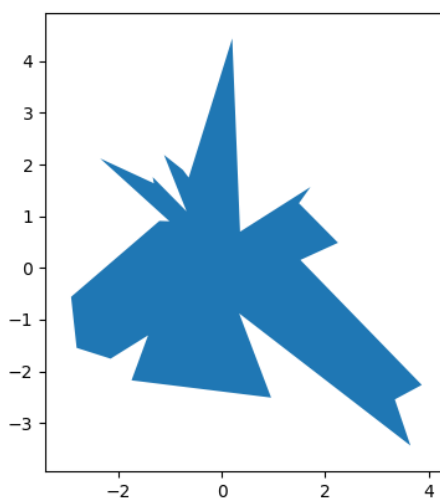
A második módszerben a pontokat x koordináta szerint rendeztem, majd az első és az utolsó pontra illesztettem egy egyenest. Ezután az egyenes feletti és az egyenes alatti pontokat sorban összekötöttem (4.2. ábra). Így az alsó és felső oldalak között biztos nincs önmetszés mert másik félsíkon vannak, továbbá az egyenes azonos oldalán lévő oldalak között sincsen önmetszés hiszen az x koordináta pontról pontra növekszik.



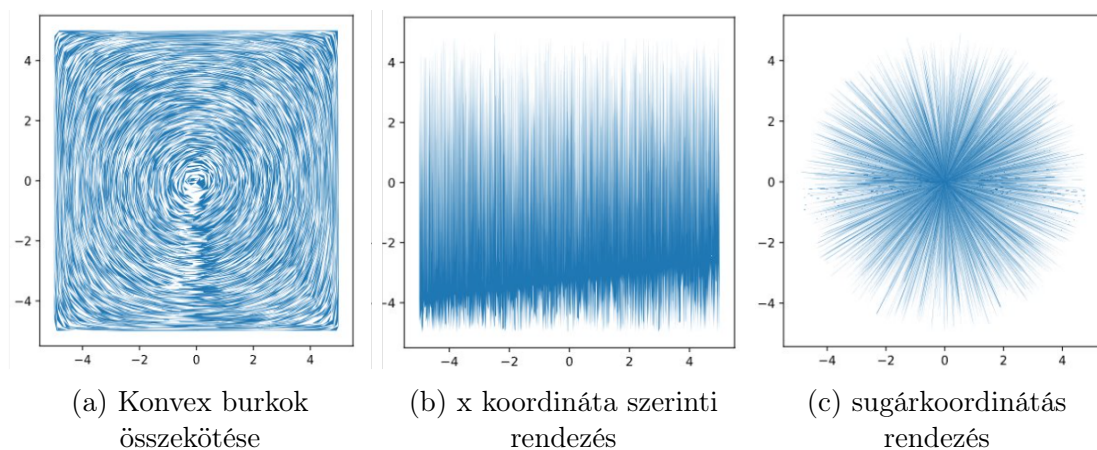
4.2. ábra. x -tengely szerinti összekötés 25 pontra

4.1.3. Radiális sokszög generálása

A harmadik módszerben a pontokat átváltottam polárkoordináta-rendszerbe [15], majd a szög koordináta értékek szerint kötöttem őket össze sorban (4.3. ábra). Az így keletkezett poligon egy szabálytalan csillag lesz. A második esethez hasonlóan a rendezés itt is kizárja az önmetszés lehetőségét.



4.3. ábra. Radiális sokszög 25 pontra

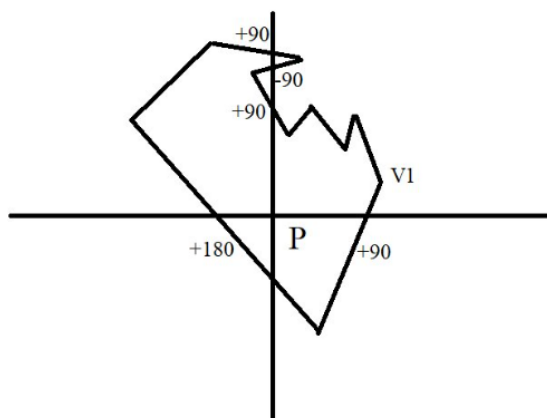


4.4. ábra. Különböző sokszög generálások 10000 pontra

4.2. A Körülfordulási szám algoritmus gyorsítása

A Körülfordulási szám algoritmusban minden oldalra el kell végezni egy arkusz-tangens számítást, ami sok műveletidőt vesz igénybe. Egy hatékonyabb megoldás erre, ha nem törődünk a pontos szögekkel, csak azt vizsgáljuk, hogy hány síknegyed váltás történt a pont körül.

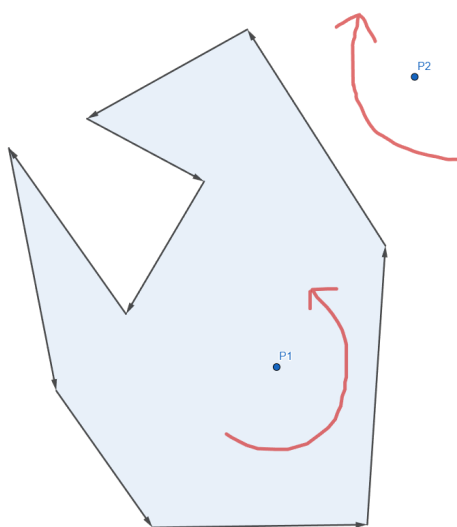
Az, hogy melyik síknegyedben van a jelenlegi csúcs egyszerű vizsgálni, mert csak össze kell hasonlítani a csúcs koordinátáit a vizsgált pont koordinátaival. Amikor végig iterálunk a sokszög csúcsain azt vizsgáljuk, hogy melyik síknegyedben van az oldalak két végpontja. Három eset lehetséges. Első esetben nem történt síknegyed váltás, ekkor nem adunk hozzá semmit a változóhoz amiben tároljuk a szögelfordulások összegét. Második esetben egy szomszédos síknegyedbe léptünk, ekkor hozzáadunk az eredményhez egy derékszöget pozitív vagy negatív előjellel attól függően, hogy melyik síknegyedbe léptünk át. A harmadik esetben az ellenkező oldalon lévő síknegyedbe léptünk, ez az eset igényli a legtöbb számítást ugyanis attól függően, hogy az oldal a pont alatt vagy felett megy át függ az, hogy pozitív vagy negatív előjellel adjuk hozzá az eredményhez az egyenesszöget (4.5. ábra). A Matlab inpolygon függvénye is ez alapján a gyorsítás alapján dolgozik [3].



4.5. ábra. Fejlesztett Körülfordulási szám algoritmus

4.3. Legközelebbi oldal vizsgálata

Mivel a sokszög oldalai pozitív forgásirányban vannak felsorolva egymás után, ezért az oldalak közvetlen környezetében igaz, hogy a belső pontok pozitív szöggel látják az oldalt, a külső pontok pedig negatívval. Ez a tulajdonság igaz mindaddig, ameddig olyan távol nem kerültünk az oldaltól, hogy már egy másik oldal van a pont és az eredeti oldal között. Ez alapján vehetjük a vizsgált ponthoz a legközelebbi oldalt, és elég azt vizsgálni, hogy a pont pozitív vagy negatív szögben látja az oldalt (4.6. ábra), ez az állítás egy határesetétől eltekintve igaz, amit a következő pontban fogok tárgyalni. Pont és oldal távolsága alatt a következőt értjük: a pontból az oldal egyenesére húzott merőleges szakasz hossza, ha az így keletkezett metszéspont az oldalra esik, különben a pont és a közelebbi végpont távolsága.

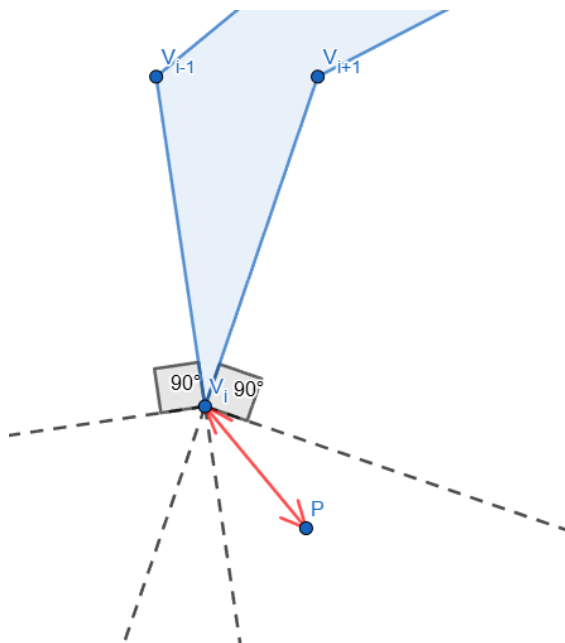


4.6. ábra. legközelebbi oldal forgásiránya

4.3.1. Hegyesszögek közelében lévő pontok lekezelése

Ha egy szakasz nem tartalmazza az egyenesének azon pontját, ahol merőlegest lehetne húzni a pontból, akkor a pont és az egyik végpont távolsága lesz az oldal és a pont távolsága. Mivel egy végpont közös a szomszédos oldalak esetében, ezért könnyen előfordulhat, hogy két oldaltól ugyanannyi a távolsága a pontnak, és így előfordulhat, hogy két oldal is a legközelebbi oldala a pontnak.

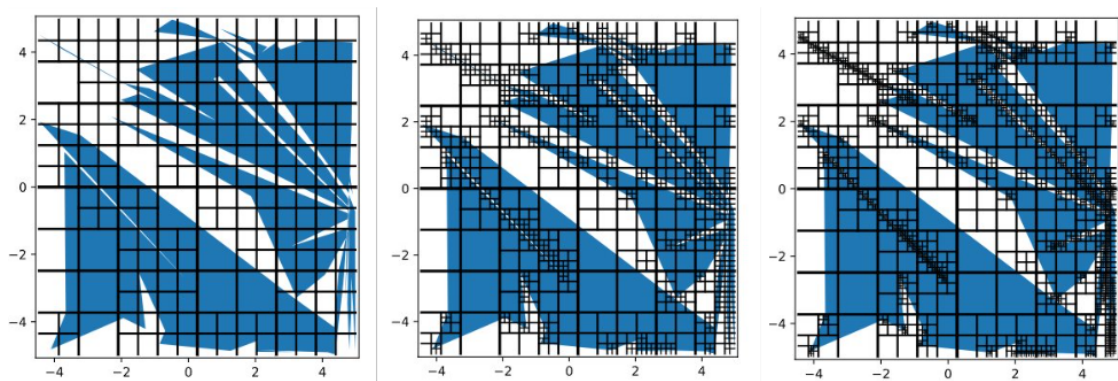
Ez a hegyes szögek esetén okozhat problémát ugyanis itt létezik egy tartomány, hogy ha abba esik a pont, akkor nem mindegy, hogy melyik oldal forgásirányát vizsgáljuk, mert két különböző eredményre juthatunk (4.7. ábra). Az ábrán látható pont ugyanolyan távolságra van mindkét oldaltól azonban a V_{i-1}, V_i oldal alapján a sokszögön belül van, a V_i, V_{i+1} oldal alapján viszont a sokszögön kívül. Erre az nyújt megoldást, ha ebben az esetben vesszünk a csúcstól egy epsilon távolságra lévő pontot az oldal irányában, majd ellenőrizzük, hogy a pontból az elmozdított csúcspontba tudunk-e egyenest húzni, hogy az a vizsgált oldalt metsze el először. Az előző példát tekintve ebben az esetben kiderül az algoritmus számára, hogy a V_i, V_{i+1} a legközelebbi oldal és így visszaadja a helyes eredményt, miszerint a pont a sokszögön kívül van.



4.7. ábra. Problémás határeset a legközelebbi oldal vizsgálatának

4.3.2. Térfelosztás

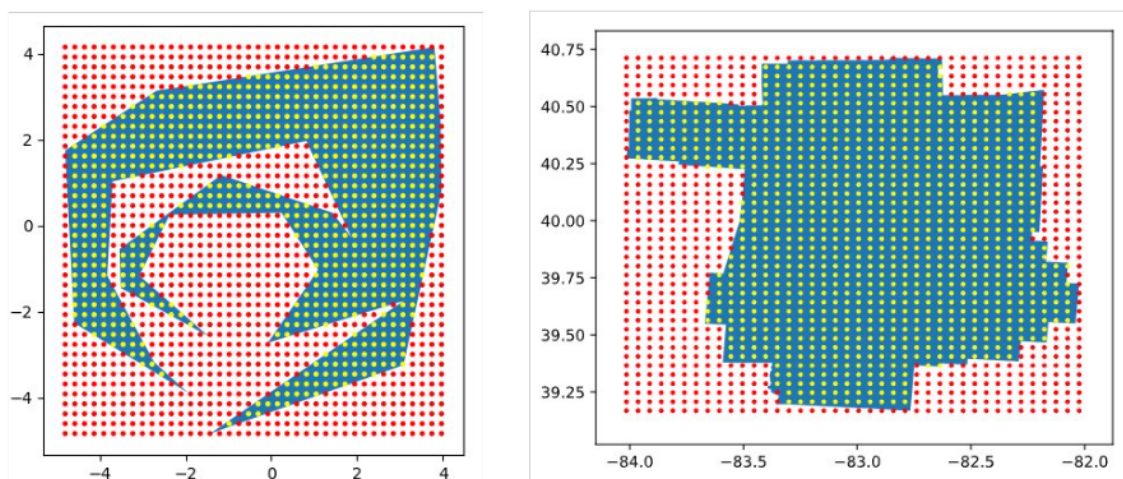
Az algoritmus hatékonyabb, ha hamar meg lehet határozni a legközelebbi oldalt. Ehhez a négyesfa nyújt segítséget. A sokszög határoló dobozát addig osztjuk a négyesfa algoritmussal, ameddig csak legfeljebb egy oldal van a cellában, vagy ameddig el nem érünk egy maximum mélységet (4.8. ábra). Mivel az oldalak jelentik a váltást a poligon külseje és belseje között, ezért ha a fa egy levelében nem marad oldal, akkor vagy az egész cella kívül van vagy az egész cella belül van, és ez kiszámolható az előfeldolgozás folyamán. Ha van a falevélen belül oldal akkor meghatározható a cellán belüli legközelebbi oldal. A fejezet bevezetőjében tárgyalt tulajdonság teljesüléséhez elég, ha egy olyan oldalt találunk, hogy a vizsgált pont és az oldal között ne legyen másik oldal, ami igaz a cellán belüli legközelebbi oldalra is akkor is, ha ez nem a legközelebbi oldal az egész poligonra nézve.



4.8. ábra. Négyesfa különböző maximum mélységekkel

4.4. Az implementáció

A vizsgálatokhoz a programot C++ nyelven írtam meg. A program egy paraméter fájl alapján generál egy adott típusú és adott oldalszámú sokszöget, de meg lehet neki adni egy konkrét fájlt is ami egy poligon csúcsait sorolja fel, például a 4.9. ábra jobb oldalán egy ilyen módon megadott sokszög látható földrajzi koordinátákkal. Ezután a program elvégzi az előzőekben tárgyalt algoritmusokat (Sugármetszés, Körülfordulási szám, fejlesztett Körülfordulási szám, legközelebbi oldal vizsgálata különböző maximum mélységekkel) és leírja a futásidőket. A grafikus megjelenítésekhez egy Python kódot használtam, ami a program által írt szöveg fájlokból dolgozik (4.9. ábra).



4.9. ábra. Futáseredmények megjelenítése. A piros pontok a testen kívül vannak, a sárga pontok a testen belül

5. fejezet

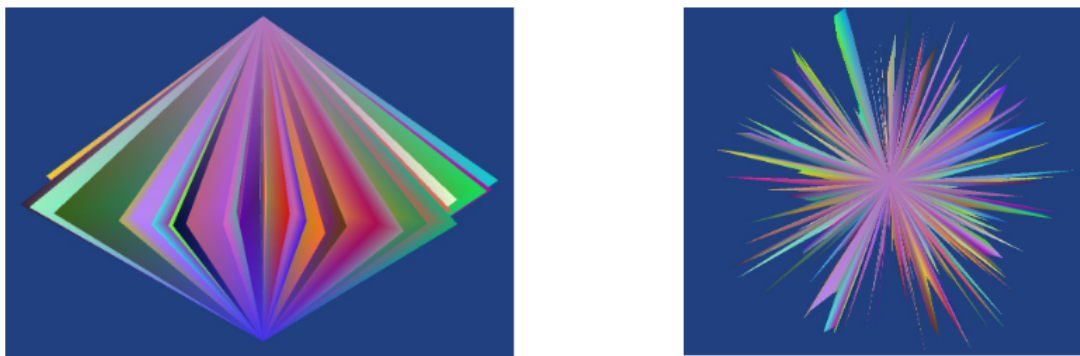
Pont a poliéderben

5.1. Poliéder generálás

A robusztus mérésekhez tetszőleges csúcsszámú véletlen poliéderek generálása volt szükséges. Az önmetszéseket itt is kerültem hasonló okból mint a kétdimenziós esetben. Ha megengedjük, hogy a lapok egyenesszöggel metsszék egymást, akkor elegendő a háromszög lapú testekkel foglalkozni, hiszen azonos síkon lévő szomszédos háromszögek le tudják írni a bonyolultabb lapokat is.

Ezek után a test tárolását a következő módon lehet megoldani hatékonyan: a csúcsokat felsoroljuk egy tömbben tetszőleges sorrendben, majd egy index tömbben felsoroljuk hármásával a lapok csúcsainak az indexeit. Itt ügyelünk arra, hogy ha a lapokat a testen kívülről nézzük, akkor a csúcsok pozitív forgásirányban vannak felsorolva. Ezt a megadást használják az obj fájlok is [16].

A tetszőleges csúcsszámú poliéder generálást úgy oldottam meg, hogy készítettem egy két dimenzióban tárgyalt csillag poligont, majd ezzel az alappal készítettem egy kétoldalú piramist, másnéven bipiramist (5.1. ábra). A poliéder csúcsszáma (5 fölött) tetszőlegesen skálázható, továbbá az alakzat kellően bonyolult a számításokhoz.



5.1. ábra. véletlen csillag alapú bipiramis oldalról és felülről

5.2. Sugármetszés határesetei

A Sugármetszés során létezik három határeset, amikor az egyszerű algoritmus, ami csak a lap metszéseket vizsgálja és az így kapott szám paritásából dönti el, hogy a pont kívül vagy belül helyezkedik el, rossz eredményt adhat vissza.

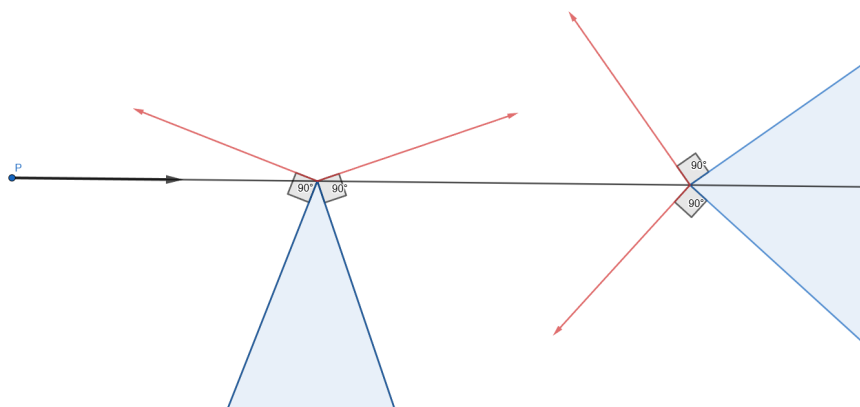
5.2.1. A sugár és a lap egy síkon van

Ha a sugár elmetshi ezt az oldalt, akkor a kettő metszete egy szakasz lesz egy pont helyett. Ebben az esetben a helyes számításhoz figyelmen kívül kell hagyni az oldallal való metszést, hiszen itt csak érintjük a testet és az nem fog változni, hogy kívül vagy belül helyezkedünk el, ez csak akkor fog változni, amikor a lap végére ér a sugár és elmetshi a következő lapot, de azt elég annál a lapnál vizsgálni.

5.2.2. A sugár egy élen megy át

Ebben az esetben a sugárnak két oldallal is lesz metszéspontja, de attól fog függeni, hogy az algoritmusnak kell-e egyet hozzáadnia a metszéspontok számához, hogy a sugár az élt „érinti” vagy „átmegy rajta”. Az élmetszés típusát úgy lehet ellenőrizni, hogy a lapokból állított merőlegeseket megnézzük, az él és sugár által meghatározott sík melyik oldalára esnek. Ha mindkettő egy oldalon van, akkor érintjük a síkot, szóval nem kell megnövelni a metszéspontok számát tároló változót, ha különböző oldalon vannak, akkor átmegy az élen, tehát egyet hozzá kell adni az eredmény változóhoz (5.2. ábra). Ha a sugár az előző pontban tárgyalt párhuzamos oldallapot metszi, természetesen itt is lesz egy élmetszés, azonban egyenértékű számítást lehet végezni akkor, ha a párhuzamos lapot figyelmen kívül hagyjuk és az előtte és utá-

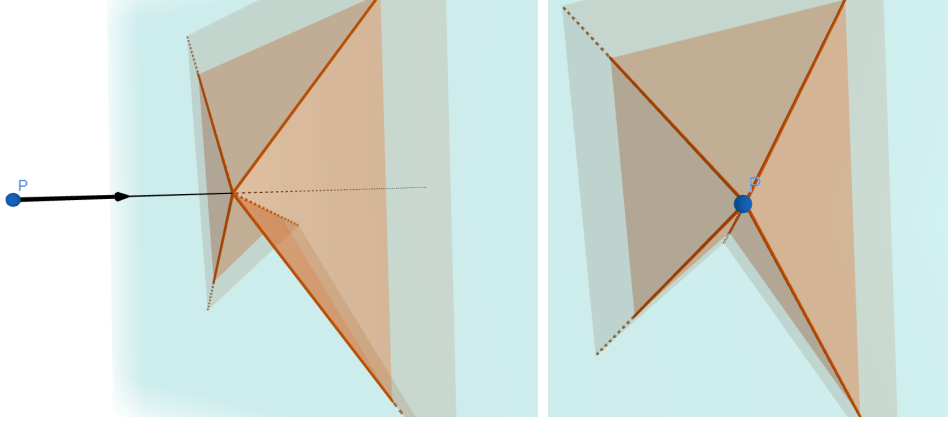
na lévő lapot úgy vesszük, mintha egymást metszenék, és ezeknek a merőlegeseit vizsgáljuk.



5.2. ábra. Élmetszés az él irányából nézve

5.2.3. A sugár egy csúcson megy át

A sugár ebben az esetben a csúcshoz tartozó lapokat metszi el, de az előző esethez hasonlóan két eset különböztethető meg: a sugár vagy „érinti” vagy „átmegy” a csúcson. Ezt a következő módon lehet eldönteni: egy sugárral merőleges síkot húzunk a metszéspont mögött, majd megvizsgáljuk a sík és a csúcshoz tartozó élekkel párhuzamos félegyenesek metszéspontjait (5.3. ábra). Ha a metszéspontokból keletkezik egy sokszög, akkor megoldható rá a pont a sokszögben feladat, és attól függően, hogy a sugár síkra vetítése belül vagy kívül van a poligonon függ, hogy átmegyünk vagy érintettük a csúcst. Ha nem keletkezik sokszög, mert 3-nál kevesebb metszéspontja van a síknak a félegyenesekkel, akkor a sugár mindenképpen csak érintette a csúcst. Ahhoz, hogy a csúcshoz tartozó éleket gyorsan le lehessen kérdezni, célszerű a fél-él adatszerkezetet alkalmazni, hiszen itt végig lehet menni ezeken az éleken a következő- és pár-él egymás utáni lekérdezésével, ameddig körbe nem érünk.

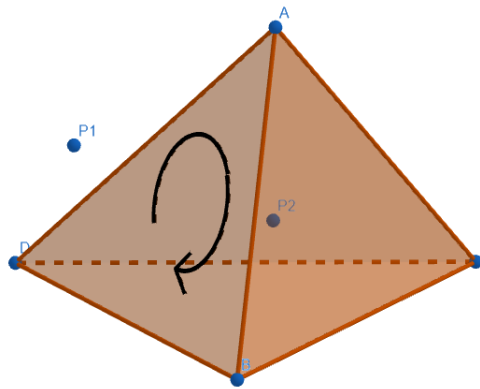


5.3. ábra. Csúcsmetszés oldalról és a sugár irányából

5.3. Legközelebbi lap vizsgálata

Mivel a poliéder úgy van megadva, hogy a lapok csúcsai kívülről nézve, pozitív forgásirányban szerepelnek, ezért a lap közvetlen környezetére igaz, hogy a pont-tartalmazások eldöntésére elég azt vizsgálni, hogy a legközelebbi lap csúcsai milyen forgásirányban látszanak a vizsgált pontból (5.4. ábra). A pont és háromszög távolságok számításához egy más által megírt implementációt használtam [17].

A két dimenzióban tárgyalt határeset itt is előfordulhat az élek és csúcsok mentén, amire szintén az nyújt megoldást, ha eltoljuk a legközelebbi pontot, és ellenőrizzük, hogy az első lap, amit elmetsz, a pontból húzott egyenes-e. Annak érdekében, hogy az eltolás legközelebbi pont a lapon maradjon, az eltolást a lap középpontja felé végzem el.



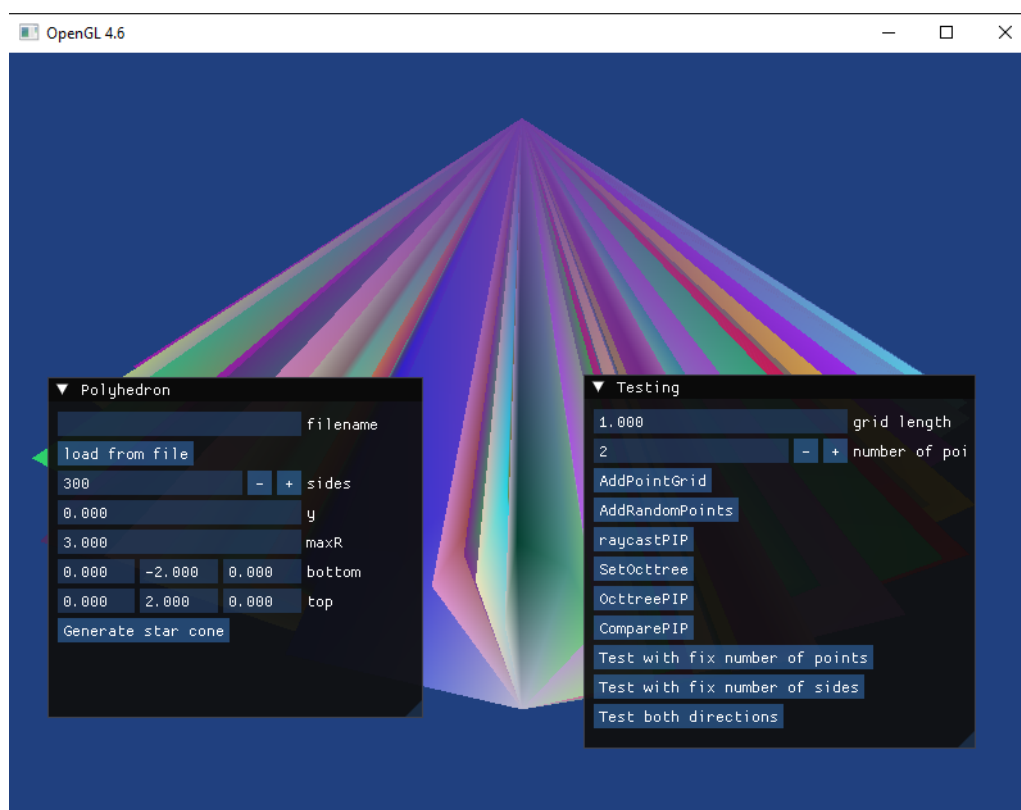
5.4. ábra. Legközelebbi lap forgásirányának a vizsgálata

5.3.1. Térfelosztás

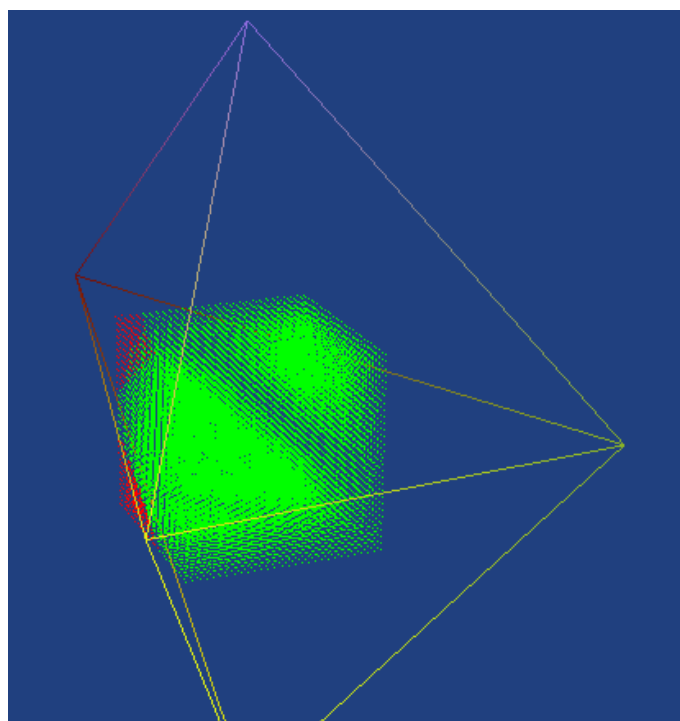
A két dimenzióhoz hasonlóan a térfelosztás itt is gyorsítja a vizsgálatokat, mert logaritmikus időben el tudjuk érni a ponthoz közeli lapokat a számításakor. Továbbá ha keletkezik egy levél, amiben nincsen lap, akkor ebben az esetben is előfeldolgozási időben el tudjuk dönteni az egész celláról, hogy kívül vagy belül van a testhez képest. A térfelosztáshoz a négyesfa térbeli megfelelőjét, a nyolcasfát használom. A szükséges cella és háromszög metszésekhez egy más által megvalósított implementációt vettem igénybe [18].

5.4. A program

A programot C++ nyelven valósítottam meg és a `glm` könyvtárat használtam, hogy hasonló legyen a grafikus megjelenítéshez és a geometriai számításokhoz használt kódforma. Az egyik lenyíló ablakkal be lehet tölteni, vagy le lehet generálni különböző poligonokat, a másikkal meg be lehet tölteni egy pontfelhőt, és különböző méréseket és összehasonlításokat lehet végezni (5.5. ábra). A sokszöget a program meg tudja jeleníteni úgy is, hogy csak az éleket színezi ki, és így könnyebben megtekinthetők az eredmények (5.6. ábra). A ComparePIP gombot megnyomva egy összehasonlítás is látható a különböző algoritmusokról, mind futásidőben, mind eredményben (5.6. ábra).



5.5. ábra. Program felhasználói felülete



5.6. ábra. Futáseredmények megjelenítése. A piros pontok a testen kívül vannak, a zöld pontok a testen belül. Ebben az esetben a sugármetszés 5 ms , egy nyolcasfa-alapú módszer 1 ms alatt végzett.

6. fejezet

Eredmények

A következőkben összehasonlítjuk az előző fejezetekben részletezett algoritmusoknak a futásidejét mind a kettő-, mind a háromdimenziós esetben. Az összes futásidő ms mértékegységgel szerepel

6.1. 2D

A pont a poligonban probléma vizsgálatánál összevetettem a sugármetszés és körülfordulási szám algoritmusokat, a 4. fejezetben tárgyalt fejlesztett körülfordulási szám algoritmussal, és a legközelebbi oldalon alapuló algoritmussal. A négyesfa készítés akkor állt le egy ágon, ha legfeljebb egy oldal maradt a cellában, vagy ha a grafikonon feltüntetett maximum mélységet el érte.

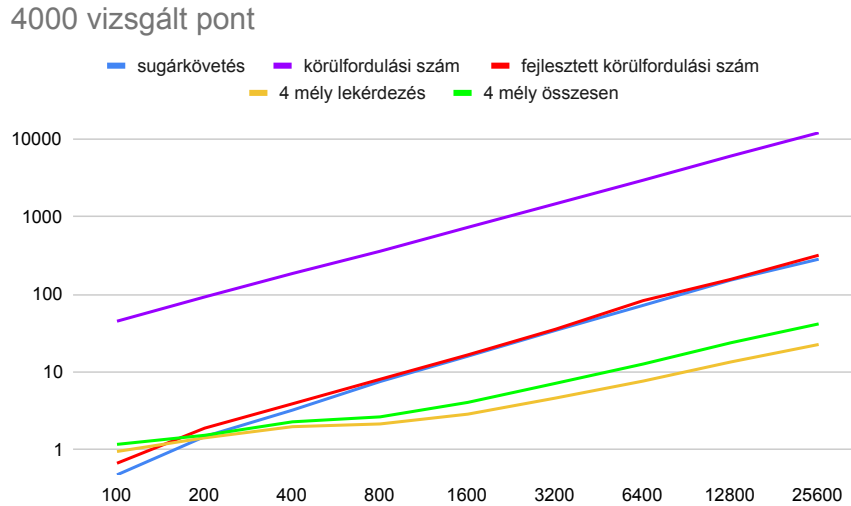
6.1.1. Változó oldalszám

A vizsgált pontok számának a rögzítésével azt lehet nyomon követni, hogy a különböző algoritmusok futásideje, hogyan változik bonyolultabb poligonokra (6.1. ábra). Mind a Sugármetszésről, mind a Körülfordulási szám algoritmusról ismert, hogy lineárisan függenek az oldalszámtól, hiszen a futásuk során végigmennek az összes oldalon, és egy konstans idejű számítást hajtanak rajtuk végre.

A fejlesztett körülfordulási szám algoritmus, szintén lineáris, hiszen csak az oldalankénti számítás módja változik, viszont az látszódik az ábrán, hogy a futásidő jelentősen csökken így, hogy nem végzünk el bonyolult trigonometrikus műveleteket.

A legközelebbi oldalon alapuló algoritmus szintén lineáris az oldalszám függvényében, hiszen minél több oldalú a poligon, annál több oldal kerül a négyes fa cel-

láiba, tehát annál több oldal közül kell kiválasztani a legközelebbit. Az ábrán azért választottam a maximum 4 mélységet a négyesfának, mert ennyi pont vizsgálatára ez az optimális mélység amivel a leggyorsabb futást lehet elérni.



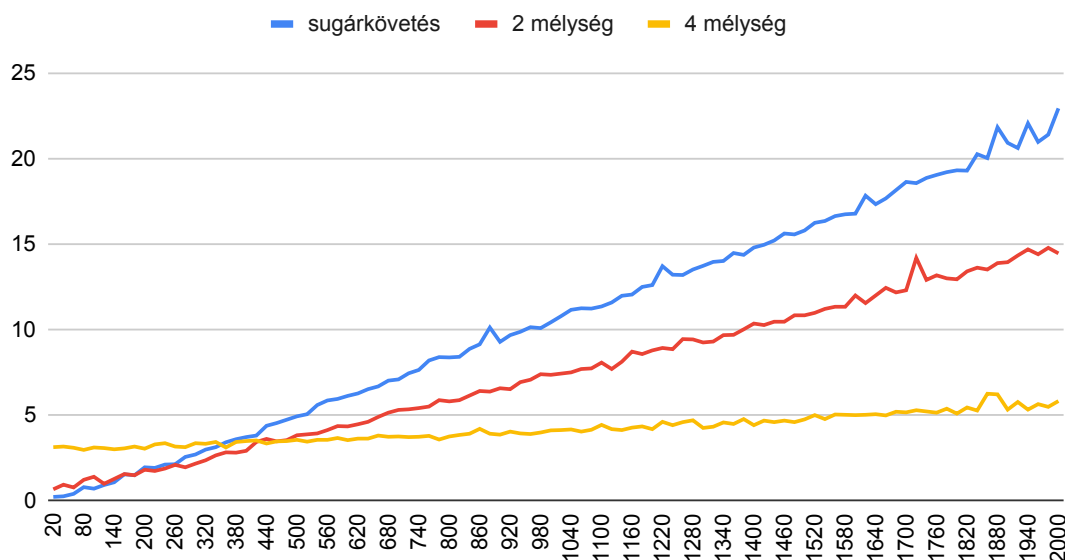
6.1. ábra. Futásidők rögzített vizsgált pont számmal a poligon lapszám függvényében logaritmus tengelyekkel

6.1.2. Változó pontszám

A Sugármetszés algoritmus nem végez előfeldolgozást és minden pontra közel ugyanolyan gyorsan fut le. Ezért a grafikonja lineáris és az origóból indul. A különböző maximum mélységű és leállási feltételű négyesfán alapuló megoldásoknak van egy előfeldolgozási idejük, ami exponenciálisan függ a fa maximum mélységétől, és van egy lekérdezési futásidő, ami annál gyorsabb minél részletesebb a fa, hiszen annál kevesebb oldalra kell meghatározni a pont-oldal távolságot (6.5. ábra).

Az olvasható le a grafikonokról, hogy egy konkrét poligonra hány pont vizsgálatánál éri meg egy bizonyos maximum mélységű fát használni. Kevés pont esetén nem éri meg mélyre menni a fában, mert túl hosszú lesz az előfeldolgozási idő, sok pont vizsgálatára pedig nem éri meg az előfeldolgozáson spórolni, mert túl hosszú lesz a pontszámítási idő.

4000 oldalú sokszög



6.2. ábra. Futásidők rögzített 4000 oldalú sokszögre a vizsgált pontok számának függvényében

6.2. 3D

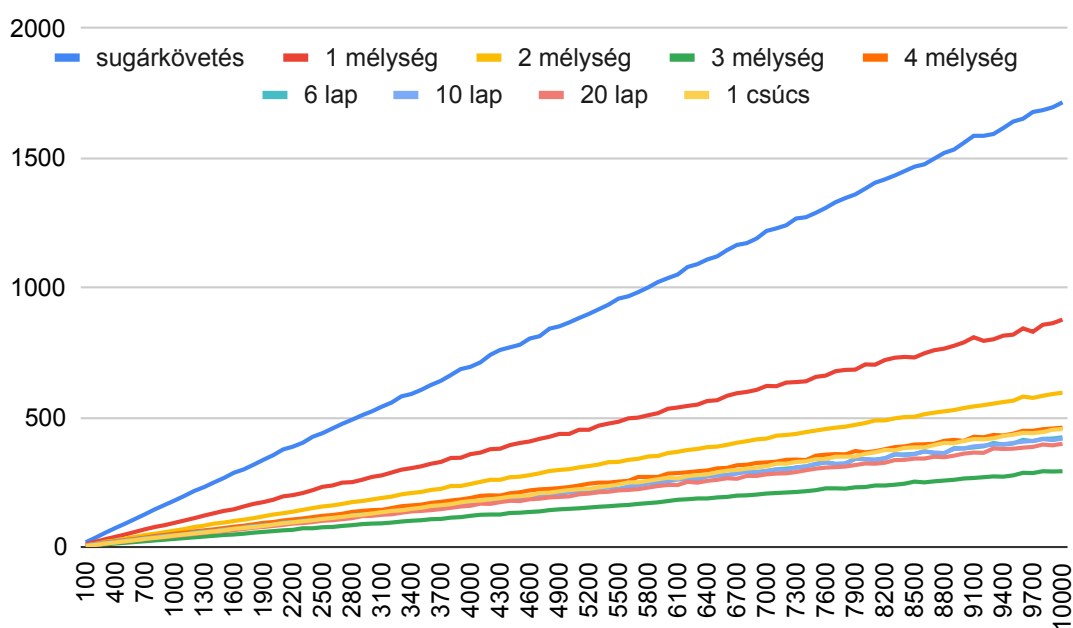
A pont a poliéderben probléma vizsgálatánál összevettem a sugármetszés algoritmust a 5. fejezetben tárgyalt legközelebbi oldalon alapuló algoritmussal. A nyolcasfa készítésnek különböző leállási feltételeit is összevettem. Alapesetben a fa készítés akkor áll le egy ágon, ha legfeljebb 3 lap marad egy cellában, vagy ha a grafikonon feltüntetett maximum mélységet elérte. Ezen kívül a 4 maximum mélységű esetet vizsgáltam úgy is, hogy akkor is le tudjon állni, ha már csak 6, 10 vagy 20 lap marad egy cellában. Továbbá mértem a futásidőt úgy is, hogy az algoritmus a lapok száma helyett a csúcsok számát vizsgálja a cellákban, és csak akkor áll le, ha legfeljebb egy csúcs marad egy cellában, ebben az esetben nem is kell maximális mélységet beállítani, mert csak véges sok diszkrét ponthoz kell eljutni.

6.2.1. Változó lapszám

A vizsgált pontok számának a rögzítésével azt lehet nyomon követni, hogy a különböző algoritmusok futásideje, hogyan változik bonyolultabb poliéderekre (6.3. ábra). Mivel a sugármetszés az összes lappal szembeni viszonyt vizsgálja, ezért várható, hogy a futásidő lineárisan függ a lapszámtól.

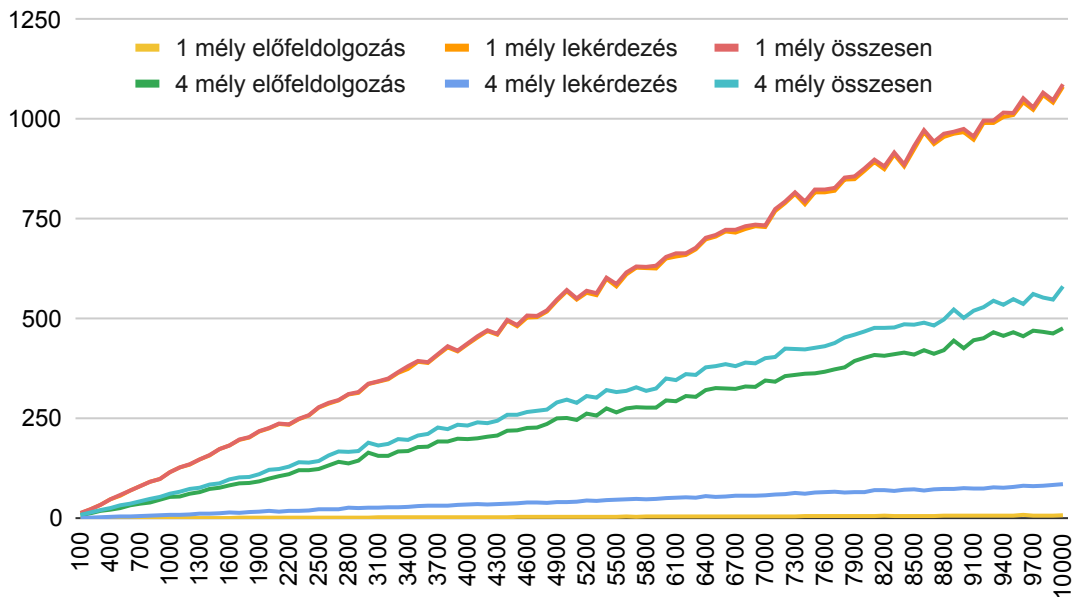
A legközelebbi lap alapú algoritmusnak mind az előfeldolgozási ideje, mind a pont lekérdezési ideje lineárisan függ a lapszámtól (6.4). Az előfeldolgozás során minden cella létrejötténél az algoritmus vizsgálja, hogy hány lap metszi a cella oldalait, ami lineárisan függ a poliéder lapszámától. Ha a maximum mélység nem változik a bonyolultabb testekre, akkor a fa levelekben arányosan annyival több lap fog maradni, mint amennyivel több lapja van a poliédernek, tehát a lekérdezési idő is lineárisan nő.

4000 vizsgált pont



6.3. ábra. Futásidők rögzített vizsgált pont számmal a poliéder lapszám függvényében

Előfeldolgozási idők



6.4. ábra. Legközelebbi lap alapú algoritmus futásideje szétszedve előfeldolgozási és lekérdezési futásidőre

6.2.2. Változó pontszám

Ugyanúgy mint a kétdimenziós esetnél a sugármetszés futásideje egy origóból induló lineáris függvény, a különböző leállási feltételű nyolcas fa alapú algoritmusok pedig olyan lineáris függvények, amik az y tengelyt egy pozitív értéknél metszik. Mélyebb fák esetén az y tengelymetszet nagyobb a meredekség pedig kisebb (6.5. ábra).

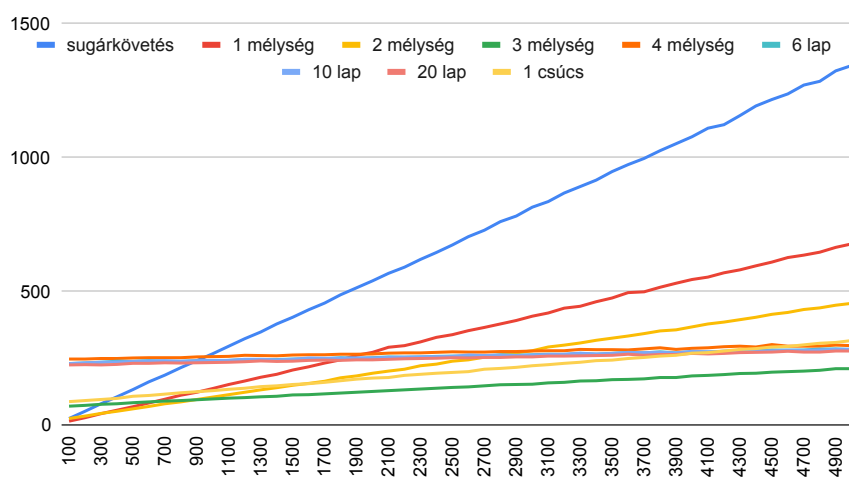
Leolvasható a grafikonokról, hogy egy konkrét poliéderre hány pont vizsgálatánál éri meg egy bizonyos maximum mélységű fát használni. Kevés pont esetén nem éri meg mélyre menni a fában, mert túl hosszú lesz az előfeldolgozási idő, sok pont vizsgálata esetén pedig nem éri meg az előfeldolgozáson spórolni, mert túl hosszú lesz a pontszámítási idő.

A két ábrát összehasonlítva az is leolvasható, hogy a betöltött poliéder esetén (6.5b. ábra) több időt lehet megtakarítani a nyolcas fával, mint a generált, bonyolultabb poliédernél (6.5a. ábra), mert az élek nem halmozódnak fel annyira egy helyen, ezért kevesebb lap kerül a cellákba.

Az ábrákról továbbá az is leolvasható, hogy a lapszám leállási feltétel változtatása javít valamennyit de közel se annyit, mint a helyes mélység megválasztása.

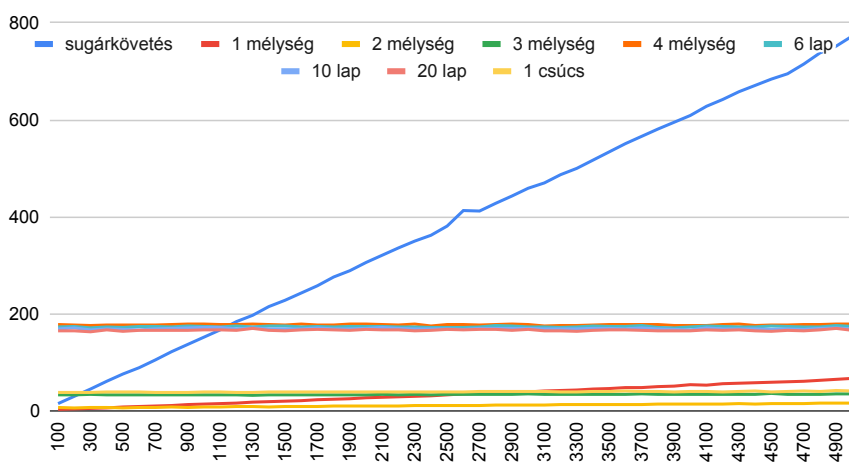
A 6.5a. ábráról leolvasható, hogy a 3 mélységű fa teljesít a legjobban ennyi pont leolvasása esetén. Azonban, ha a függvényt kibővítjük és megnézzük, hogy több pont vizsgálata esetén mi a futásidő (6.6. ábra), akkor látszódik, hogy nagyjából 10000 pont körül a 4 mélység megelőzi a 3-mat. Mivel az egyre nagyobb mélységű fán alapuló algoritmusok futásidő meredeksége egyre jobban csökken, mindig lesz egy pont, ahol érdemes átváltani egy nagyobb mélységű fára.

5000 oldalú csillag alapú bipiramis



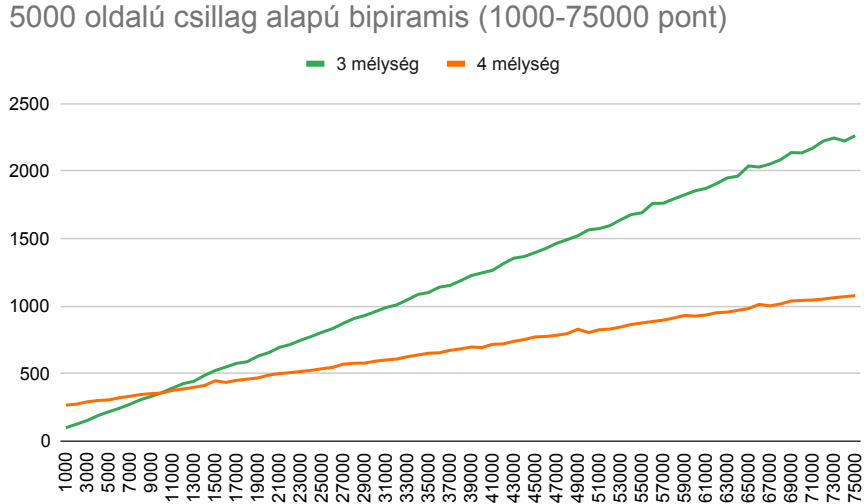
(a) Csillag-bipiramis

5000 oldalú tehén obj



(b) Tehén obj fájl

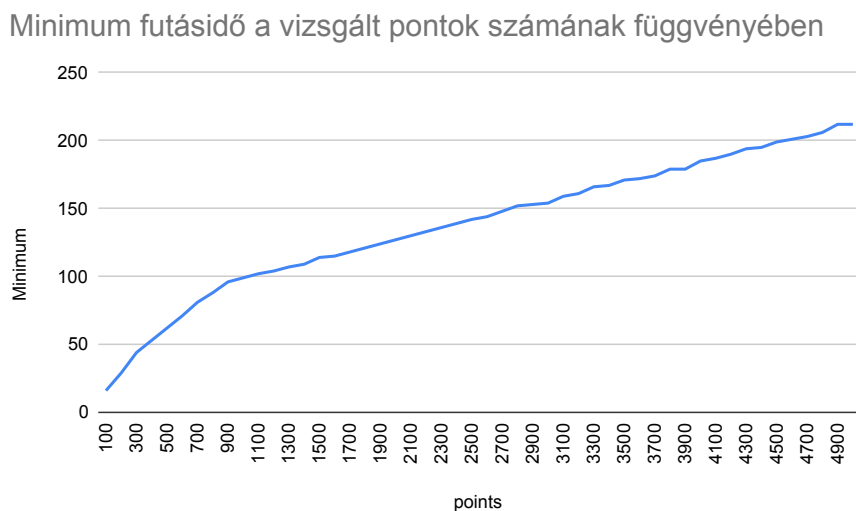
6.5. ábra. Futásidők rögzített testekre a vizsgált pontok számának függvényében (100-5000)



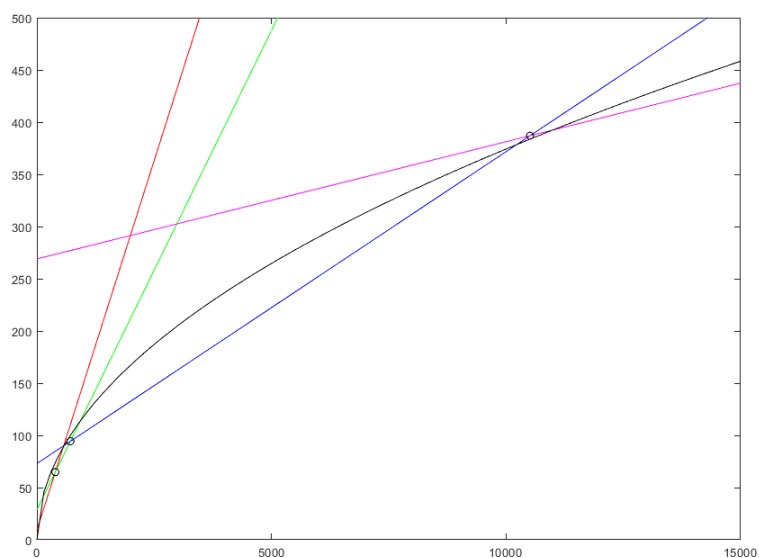
6.6. ábra. Futásidők rögzített testekre a vizsgált pontok számának függvényében (1000-75000)

Felmerülhet a kérdés, hogy miben jobb térfelosztós megoldás, ha ugyanúgy lineáris, mint a sugármetszés. Mivel tetszőlegesen megválasztható a nyolcas fa maximális mélysége, ezért inkább azt érdemes vizsgálni, hogy mennyi a minimum futásidő, ha a vizsgált pontok számához viszonyított optimális mélységet választjuk (6.7. ábra). Így máris egy gyökfüggvénynek tűnő függvényt kapunk, aminek csökken a meredeksége minden alkalommal, amikor egy nagyobb mélységű fa lesz a legjobb megoldás. Ha egy szinttel mélyebbre megyünk a fában, akkor az előfeldolgozási idő egy valós számmal fog megszorozódni, mert a cellák száma exponenciálisan növekszik. Továbbá, ha a cellák száma valahányszorosára nőtt, akkor az egy levélre jutó lapok száma ezzel arányosan fog csökkenni. Ezek alapján, amikor a fa mélység eggyel nő, akkor a futásidő grafikon függvényértéke nagyjából a -szorosára nő és a meredekség a -ad részére csökken ($a \in \mathbb{R}$), és ez a tulajdonság igaz a gyökfüggvényre is (6.8).

A csillag-bipiramis méréseinél 1-ről 2 mélységre változásnál az előfeldolgozási idő nagyjából az 1.5-szeresére nőtt és a pont lekérdezési idő az 1.5-öd részére csökkent. Amikor a mélység 2-ről 3-ra, valamint amikor 3-ról 4-re változott, akkor az előfeldolgozási idő a nagyjából háromszorosára nőtt a pont lekérdezési idő pedig nagyjából a harmadára csökkent.



6.7. ábra. Futásidő rögzített 5000 oldalú poliéderre optimális mélységválasztással

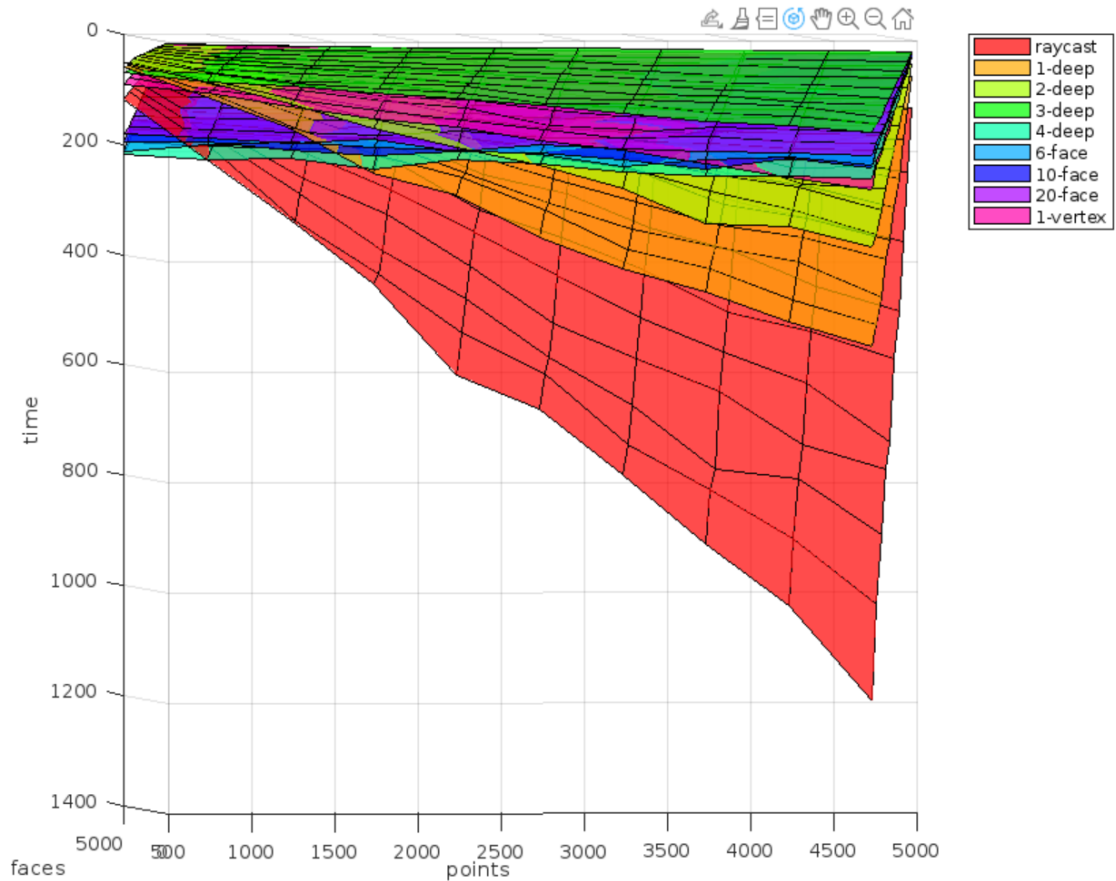


6.8. ábra. Különböző mélységű térfelosztásos algoritmusok színessel, és az általuk közelített gyökfüggvény feketével

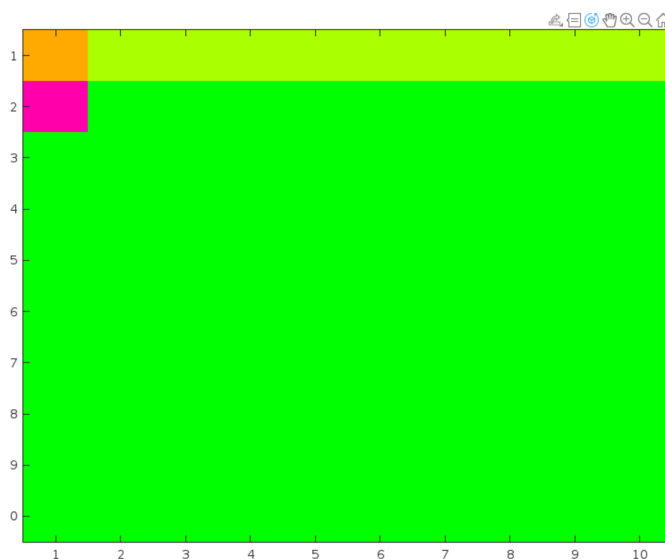
6.2.3. Változó lapszám és pontszám

Az előző két pontban tárgyaltakat lehet együtt is vizsgálni és ezáltal három dimenziós grafikon készíteni a lapok számának és a vizsgált pontok számának függvényében (6.9. ábra). Itt megmutatkozik az, ami sejthető volt az alapján hogy lineárisak voltak a függvények mindkettő dimenzió szerint, miszerint a háromdimenziós függvény egy bilineáris kapcsolatot ír le az összes algoritmusra.

Ha azt vizsgáljuk, hogy melyik esetben melyik algoritmus a legrövidebb, akkor a 3 mélységű fa elsőprő sikert aratott ezekre a mérési szempontokra (6.10. ábra). Az előző alfejezetben tárgyaltaknak alapján a 4 mélység leelőzi a 3 mélységet elég pont vizsgálata esetén, de az kimondható, hogy 500-5000 lapú testekre, 500-5000 pont vizsgálata esetén egy 3 mélységű fával érdemes alkalmazni a legközelebbi lap alapú algoritmust.



6.9. ábra. Futásidők a lapszám (500-5000) és a vizsgált pontok számának (500-5000) függvényében



6.10. ábra. Leggyorsabb algoritmus különböző esetekben. 500 lap és 500 csúcs esetén 1 mély, 500 lap és 500+ csúcs esetén 2 mély, 1000 lap és 500 csúcs esetén 1 csúcs, minden más esetben 3 mély fa nyer

7. fejezet

Összegzés

A pont a sokszögben feladatra megmutattuk egy fejlesztését a körülfordulási szám algoritmusnak, továbbá létrehoztunk egy új, térfelosztáson alapuló algoritmust, ami hatszor olyan gyorsan működik, mint az eddigi sugárkövetés 5000 pont vizsgálata esetén.

Különböző poligon és poliéder generálási módszereket fejlesztettünk tetszőlegesen sok pontra, amivel egyszerűen lehet tesztelni és összehasonlítani a különböző algoritmusokat.

A háromdimenziós esetben a sugárkövetés határesetének a kezelésére mutatunk megoldást, valamint itt is létrehoztunk egy térfelosztáson alapuló algoritmust, ami 5000 pont vizsgálata esetén az eddigi sugárkövetéses módszernél 6.5-ször olyan gyorsan működik bonyolult generált poliéderekre, és 48.3-szor olyan gyorsan működik a természetesebb betöltött poliéderekre. Megmutattuk, hogy az optimális mélység megválasztásával közel gyökös időben lehet feldolgozni akárhány pontot, valamint, hogy az esetek túlnyomó többségében három az optimális mélység. Vizsgáltunk egyéb nyolcas fa leállási feltételt is, de kiderült, hogy a legnagyobb változtatást a jó maximum mélység megválasztása adja.

7.1. Továbbfejlesztési lehetőségek

Egy lehetséges fejlesztési iránya az algoritmusnak egy másik térfelosztó módszerre áttérni. A négyes és nyolcas fa az egyik legegyszerűbb térfelosztás, de nem feltétlenül a legjobb. A felezéseknél könnyen előfordulhat, hogy nem ugyanannyi csúcs/lap kerül a különböző oldalakra, ezért lehet, hogy mélyebbre kell menni, mint

egy optimálisabb módszernél. Erre jobb megoldás lehet, ha felezések helyett úgy osztanám fel a testet, hogy hasonló mennyiségű csúcs vagy lap kerüljön egy oldalra, így működik az úgynevezett KD-fa [19].

Irodalomjegyzék

- [1] Jianfei Liu és tsai. „A new point containment test algorithm based on pre-processing and determining triangles”. *Computer-Aided Design* 42.12 (2010), 1143–1150. old. ISSN: 0010-4485. DOI: <https://doi.org/10.1016/j.cad.2010.08.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0010448510001429>.
- [2] M. Shimrat. „Algorithm 112: Position of point relative to polygon”. *Commun. ACM* 5.8 (1962. aug.), 434. old. ISSN: 0001-0782. DOI: 10.1145/368637.368653. URL: <https://doi.org/10.1145/368637.368653>.
- [3] Kai Hormann és Alexander Agathos. „The point in polygon problem for arbitrary polygons”. *Computational Geometry* 20.3 (2001), 131–144. old. ISSN: 0925-7721. DOI: [https://doi.org/10.1016/S0925-7721\(01\)00012-8](https://doi.org/10.1016/S0925-7721(01)00012-8). URL: <https://www.sciencedirect.com/science/article/pii/S0925772101000128>.
- [4] Da Yan, Zhou Zhao és Wee Keong Ng. „Monochromatic and Bichromatic Reverse Nearest Neighbor Queries on Land Surfaces”. 2012. okt. DOI: 10.1145/2396761.2396880.
- [5] Lei Liu és tsai. „Efficient Construction of Voxel Models for Ore Bodies Using an Improved Winding Number Algorithm and CUDA Parallel Computing”. *ISPRS International Journal of Geo-Information* 12 (2023. nov.), 473. old. DOI: 10.3390/ijgi12120473.
- [6] Raphael Finkel és Jon Bentley. „Quad Trees: A Data Structure for Retrieval on Composite Keys.” *Acta Inf.* 4 (1974. márc.), 1–9. old. DOI: 10.1007/BF00288933.

- [7] Elmar Langetepe és Gabriel Zachmann. *Geometric Data Structures for Computer Graphics*. 2006. jan. ISBN: 9780367803735. DOI: 10 . 1201 / 9780367803735.
- [8] Tomas Möller és Ben Trumbore. „Fast, minimum storage ray/triangle intersection”. *ACM SIGGRAPH 2005 Courses*. SIGGRAPH '05. Los Angeles, California: Association for Computing Machinery, 2005, 7-es. ISBN: 9781450378338. DOI: 10.1145/1198555.1198746. URL: <https://doi.org/10.1145/1198555.1198746>.
- [9] Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. 1980. okt.
- [10] *Recursive subdivision of a cube into octants*. <https://en.wikipedia.org/wiki/Octree>. Accessed: 2024-12-01.
- [11] D.E. Muller és F.P. Preparata. „Finding the intersection of two convex polyhedra”. *Theoretical Computer Science* 7.2 (1978), 217–236. old. ISSN: 0304-3975. DOI: [https://doi.org/10.1016/0304-3975\(78\)90051-8](https://doi.org/10.1016/0304-3975(78)90051-8). URL: <https://www.sciencedirect.com/science/article/pii/0304397578900518>.
- [12] *An Introduction to the Half-Edge Data Structure*. <https://cs184.eecs.berkeley.edu/sp24/docs/half-edge-intro>. Accessed: 2024-11-29.
- [13] Ky Fan. *Convex Sets and Their Applications. Summer Lectures*. 48. oldal. Argon national laboratory, 1959.
- [14] *Convex Hull Algorithm in C++*. <https://www.geeksforgeeks.org/convex-hull-algorithm-in-cpp/>. Accessed: 2024-11-29.
- [15] Richard G. Brown. *Advanced Mathematics: Precalculus with Discrete Mathematics and Data Analysis*. Szerk. Andrew M. Gleason. Evanston, Illinois, USA: McDougal Littell, 1997. ISBN: 0-395-77114-5.
- [16] *Media subtype name: obj*. <https://www.iana.org/assignments/media-types/model/obj>. Accessed: 2024-12-01.
- [17] *Closest point in triangle in C++*. <https://stackoverflow.com/questions/2924795/fastest-way-to-compute-point-to-triangle-distance-in-3d>. Accessed: 2024-11-29.

- [18] *AABB box and triangle intersection in C++*. <https://gamedev.stackexchange.com/questions/142918/intersecting-triangle-and-boundingbox-in-3d-environment>. Accessed: 2024-11-29.
- [19] *Introduction to K-D Trees*. <https://www.baeldung.com/cs/k-d-trees>. Accessed: 2024-12-01.