Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

3D nyomtatáshoz külső alátámasztás generálása

Témalaboratórium beszámoló

Készítette
Erős Pál

Konzulens
Dr. Salvi Péter

2023. december 8.

Tartalomjegyzék

Bevezető

A 3D nyomtatás világa mindig is vonzott. Izgalmas dolog, amikor akár otthoni körülmények között is szinte bármilyen háromdimenziós alakzatot létre lehet hozni. Szerettem volna olyan témát választani, ami nem csak egy elméleti dolog, hanem a gyakorlatban is alkalmazható és akár valamilyen szinten látványos is. Így esett a választásom erre a projektre, amely a 3D nyomtatáshoz alátámasztásokkal foglalkozik. A 3D nyomtatás során a nyomtatók nem tudnak levegőben nyomtatni, illetve bizonyos meredekség esetén sem tudja megtartani magát az alakzat a nyomtatás folyamán. Ennek a problémának az orvoslására szolgálnak az alátámasztások (1. ábra), amelyek szintén nyomtatott alakzatok és a szerepük az, hogy a nyomtatás során az új rétegnek legyen mire támaszkodnia. A feladat során nem volt cél, hogy egy teljesen új algoritmust dolgozzak ki, vagy hogy hibátlanul működjön a gyakorlatban bármilyen modellen. Ez azért jóval komolyabb feladat lett volna. Azt szerettük volna elérni, hogy egy 3D modellre a program kiszámolja azokat a pontokat, amiket aká kellene támaszani egy éles nyomtatás során.

1. ábra. *Egy 3D nyomtatás eredménye alátámasztással [1]*

1. fejezet

Szakmai áttekintés

1.1. Általános probléma

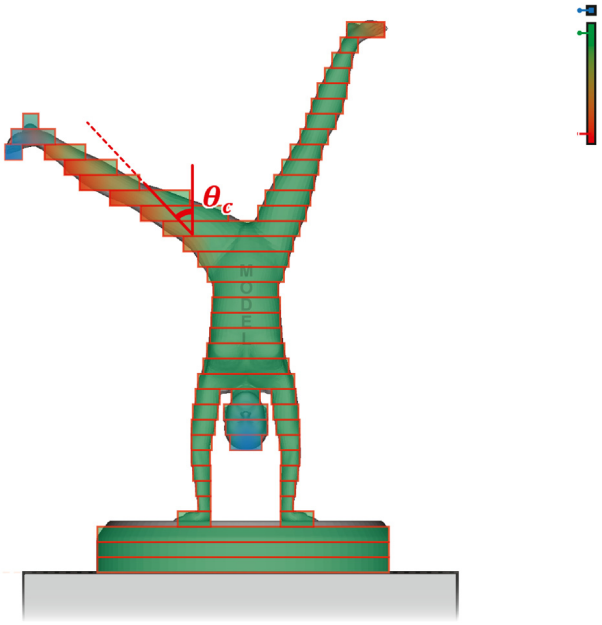
A 3D nyomtatás manapság kezd egyre inkább elterjedni. Már egy ideje nem csak az iparban, hanem hobbi szinten is egyre többen használják. Rengeteg, bárki számára elérhető ezköz van a piacon. A műanyagot használó nyomtatók a legelterjedtebbek, de vannak olyanok is, amik fémeket vagy akár betont is képesek nyomtatni. A mi esetünkben elsősorban a műanyag alapú nyomtatókra fogunk koncentrálni.
A 3D nyomtatási technológiák közül kettő terjedt el főként. A gyantás és filamentes nyomtatók. A mi feladataunk szempontjából nem kell különbséget tennünk a típusok között. Az alap probléma, amire megoldást kerestünk a következő. A nyomtatás folyamán az alakzatot rétegekből építjük fel. Az új réteg mindig a korábbi rétegekre támaszkodik. Vannak azonban olyan esetek, amikor ezt nem tudja megtenni. Például, ha az alakzatnak vannak bizonyos „lebegő" részei, amik csak egy fentebbi réteggel fognak kapcsolódni az alakzatunkhoz. De az is előfordul, hogy a modellben van olyan rész, ami túlzottan lapos vagy akár vízszintes. Ebben az esetben az alakzatban az új réteg nem tud kellően megtapadni a korábbi részeken (1.1. ábra), ezért leeshet. A nyomtatás közben az is lehet probléma, hogy – a még lágy anyag – a saját súlya alatt lehajlik vagy letörik. Ezeket a problémákat szeretnénk elkerülni.

1.1. ábra. *Bal oldalt alátámasztás nálkül, jobb oltalt pedig alátámasztással nyomtatták ki ugyanazt az alakzatot. Jól látszik a különbség [1]*

Ennek következtében úgynevezett alátámasztásokat szoktunk alkalmazni. A lényegük az, hogy egy ideiglenes támaszt is kinyomtatunk az alakzat mellé, ami a nyomtatás során megtámasztja az alakzatot. Ehhez a segéd alakzathoz már tudnak tapadni az „lebegő" rétegek is. Ezeket az alátámasztásokat úgy szokták csinálni, hogy a nyomtatás végeztével könnyedén – lehetőleg nyom nélkül – el lehessen távolítani. Az alátámasztásokból is sok típus létezik. Van ami egy egyszerű oszlop, de vannak olyanok is amelyek egy fa alakját veszik fel. Vannak olyan modellek, ahol egyik vagy másik típus használata előnyosebb, illetve az anyagköltség szempontjából is lehet különbség.

1.2. Kiindulópont

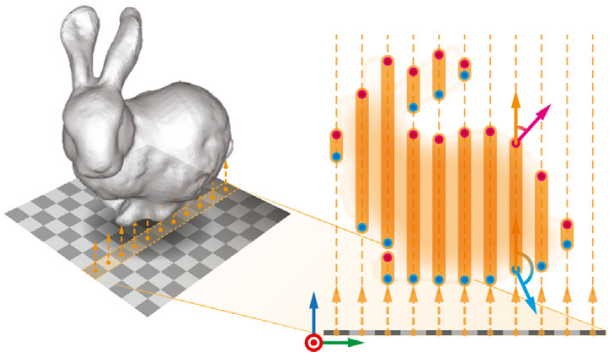
A feladat megldása során *Seongje Jang, Byungjin Moon és Kunwoo Lee* által 2020-ban publikált *Free-Floating Support Structure Generation* [1] cikk alapján dolgoztam.
A cikk elején általánosságban bemutatják a 3D nyomtatást és az alátámasztás szükségességét. Ezt követően kezdik el részletezni a folyamatot. Mint korábban én is írtam, két féle eset van, amikor szükséges alátámasztás. Az egyik az úgynevezett „island", azaz sziget. [3] Ebben az esetben az adott rétegen belül nem kapcsolódik másik, korábbi réteghez az alakzat aktuális része, azaz „lebeg". A másik eset az „overhang", azaz a túlnyúlás. Ebben az esetben az alakzat túl lapos, vagy túlságosan hosszan nyúlik ki a levegőbe. Így az alakzat a nyomtatás során nem tudja saját magát megtartani. Ezekben az esetekben van szükség az alátámasztásokra.



1.2. ábra. *A képen pirossal a túlnyúlás, kékkel a sziget, zölddel pedig az alátámasztott részek vannak jelölve [2].*

A cikkben a támaszok kiszámításának folyamata négy elkülöníthető részre bomlik. Az első részben a modellnek azokat a pontjait keresik ki, amikhez potenciálisan alátámasztás kellhet. A módszer a következő. A modell vízszintes síkjában egy rácsot készítünk. A rácsnak a kereszteződéseiből függőleges sugarakat indítunk az alakzaton keresztül. Minden

sugár esetében megkeressük azokat a szakaszokat, ahol a sugár egyenese az alakzaton belül fut (1.3. ábra). Egy sugár esetében ez lehet egy vagy több szakasz is, de az is előfordulhat, hogy nem metszi vagy csak egyetlen pontban az alakzatot. A feladat során úgy döntöttem, hogy figyelmen kívül hagyom ezeket a szélsőséges eseteket. Az alakzaton belül futó szakaszoknak a két végpontjára lesz szükségünk. A függőlegesen alacsonyabban elhelyezkedő pontok lesznek a bemeneti metszéspontok, míg a szakaszok magasabban lévő végpontjai lesznek a kimeneti metszéspontok. Értelemszerűen a bemeneti pontok lesznek azok, amikhez lehet, hogy szükséges alátámasztás.



1.3. ábra. *Az alakzatnak egy keresztmetszeti szeletén jól látszanak a kékkel jelölt bemeneti és a pirossal jelölt kimeneti pontok. A sárga nyilak a sugarakat jelzik [2]*

A második részben a bementi pontoknak a szomszédait számoljuk ki. Minden egyes bemeneti ponton végigmegyünk. Megnézzük, hogy a pont körül van-e másik bemeneti pont. A rácspontok mentén húzott sugarak miatt minden pontnak maximum nyolc szomszédos pontja lehet. A szomszédság viszont az esetünkben csak akkor áll fenn, ha bizonyos kritériumok teljesülnek. X pontnak Y csak akkor szomszédja, ha X és Y pontok felülnézetből (azaz X és Y koordináták mentén) egy-egy szomszédos rácsponton helyezkednek el. De ennek a feltételnek a teljesülése még nem elégséges. Fontos hogy X pont egy síkban vagy alacsonyabban helyezkedjen el, mint Y pont. Ez azért fontos, mert a modellt lentről felfelé nyomtatjuk, ezért csak a korábban kinyomtatott réteghez tud tapadni az új. Ez alapján rájöhetünk arra is, hogy ezek a szomszédsági kapcsolatok nem feltétlenül kétirányúak. Elképzelhető, hogy X pontnak szomszédja Y, de Y pontnak nem szomszédja X pont. Ezekből a szomszédsági, irányított élekből építhetünk egy hálót. Minél kisebbre vesszük a sugarak közti távolságot, annál sűrűbb lesz a háló és annál jobban közelíti a modell alakját. Minden szomszédsági élre számolunk egy súlyt, ami a két pontot összekötő szakasz meredeksége lesz. $\theta = \tan^{-1}\left(\frac{d}{h}\right)$ képlettel számolható ki a meredekség, azaz a súly, ahol d a két pont közötti távolság, h pedig a két pont magasságbeli különbsége. Ez a θ szög lesz az, amit az él a függőleges tengellyel bezár. A nyomtatótól és a nyomtatott anyag tulajdonságaitól függően létezik egy küszöbérték, aminél kisebb θ szög esetén az alakzat meg tudja magát tartani, ha nagyobb, akkor pedig nem, vagy csak bizonyos távolságig.
A harmadik részben a szomszédsági háló alapján kiszámoljuk azokat a pontokat, amelyeket alá kell támasztani. A cikkben leírt algoritmus és az általam megvalósított változat minimálisan eltér. A végeredmény és a logika ugyanaz, csak az implementációm sajátosságai miatt

kicsit változtattam a folyamaton. Ezt majd később részletezem. Most tekintsük át röviden a cikkben leírt algoritmust. Először sorba rendezi a bemeneti pontokat a magasságuk szerint. Ezután kiválasztja a legalsó pontot, amit alátámasztásnak jelöl. Ezután a legalsó pont szomszédjaira számol súlyt, amely az aktuális pont súlya plusz a két pont közötti él súlya. Ekkor minden szomszédos pontra kiszámolja a súlyt. Majd minden szomszédos pontra lép és kezdi előlről, lényegében rekurzívan a folyamatot. Így, mint egy fában végigjárja az összes bemeneti pontot. Amennyiben egy pontra a kiszámított súly nagyobb, mint a határérték, akkor az a pont is alátámasztandó lesz. Bekerül az alátámasztandó pontok listájába és ezzel együtt a súlyát is nullázzuk, hogy a pont szomszédjai már innen számolódjanak. Így lényegében kiszámoljuk az összes olyan pontot, amit alá kell támasztani, illetve azt is, hogy melyik alátámasztott pont, melyik pontokat tartja meg.

A cikk negyedik része magát az alátámasztás generálását taglalja. Az alátámasztás típusokból itt a legegyszerűbb, függőleges oszlopokat használják. Minden alátámasztandó bemeneti pontból egy oszlopot húznak lefelé, az alakzat legalsó pontjának a síkjáig. Abban az esetben, ha az alátámasztandó pont és a legalsó pont síkja között van a modellünknek egy része, akkor a támasztó oszlop nem a legalsó pont síkjáig megy, hanem csak a modell azon kimeneti pontjáig, ami az aktuális, alátámasztandó pont alatt van. Ezzel elkerüljük azt, hogy az alátámasztások az alakzaton belül fussanak. A cikkben az alátámasztásokat függőleges hengerekkel valósítják meg. A henger azon végei, ahol a modellükkel érintkezik, el vannak keskenyítve. Erre azért van szükség, hogy a nyomtatás végeztével könnyedén el lehessen távolítani a támaszokat. A hengereknek az a vége, amely a tárgyasztallal érintkezik, enyhén ki van szélesítve, hogy a nyomtatás során jobban tapadjon és merevebben álljon az asztalon. A stabilitás növelésének érdekében a szomszédos hengerek között átlós, merevítéseket is alkalmazhatunk. Így az alátámasztás végeredménye egy rácsos tartó lesz. A cikk ezen részét, tehát a tényleges alátámasztó struktúrát már nem teljesen valósítottam meg a feladat során. Ezt a későbbiekben részletezem.

2. fejezet

Implementáció

2.1. Általános áttekintés

Az implementáció során törekedtem mindent a lehető leginkábban a cikkben leírtak alapján megvalósítani. Van pár olyan részlet, amit másképpen csináltam, de ezeket a különbségeket a későbbiekben bemutatom. A program alapértelmezetten olyan `.obj` fájlokat olvas be amelyek függőleges tengelye az y. Ha a modellünk nem így van tájolva, akkor megadhatjuk, hogy a program forgassa el a modellt. A program gyakran számol double értékekkel, amelyeknél a számítások során kisebb eltérések léphetnek fel. Ezeket a minimális eltéréseket a progrem kezeli minden esetben. A feladat részletes implementációja a következőkben olvasható.

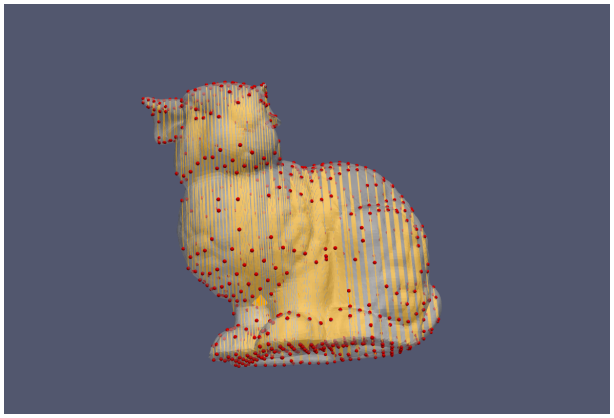
2.2. OpenMesh

A feladat megvalósításának alapjául az OpenMesh keretrendszert használtam. Ez a szoftver két- és háromdimenziós hálók létrehozására, kezelésére használható. c++ nyelven íródott, így a feladat megoldása során én is azt használtam. Az OpenMesh-t a feladat elején használtam főként. Számomra a legelőnyösebb funkciója az volt, hogy egy beépített függvénnyel könnyedén beolvas egy `.obj` fájlt és ezt eltárolja egy háromszög hálóban. Ebből a háromszög hálóból tudtam később kinyerni a szükséges adatokat, csúcsokat, éleket és a háromszögeket. Az OpenMesh-ben még rengeteg funkció és lehetőség van, viszont ezek nagy részét én nem tudtam kihasználni.

2.3. A program logikája

A fájl elején különböző direktívákkal létrehoztam demókat, amelyek egy-egy alakzatot és a hozzá kapcsolódó paramétereket definiálják. Erre nem volt kimondottan szükség, de így jelentősen gyorsult a folyamata annak, ha egy másik alakzattal szeretném kipróbálni a programot.
Ezt követően létrehoztam egy (`meshObject:MyMesh`) változót, amibe az alakzatot be is olvastam. Pár másik változóba pedig a fix értékeket tároltam el, az egyik a sugarak közti távolság (`l:double`), a másik pedig az alátámasztó struktúra oszlopainak vastagsága

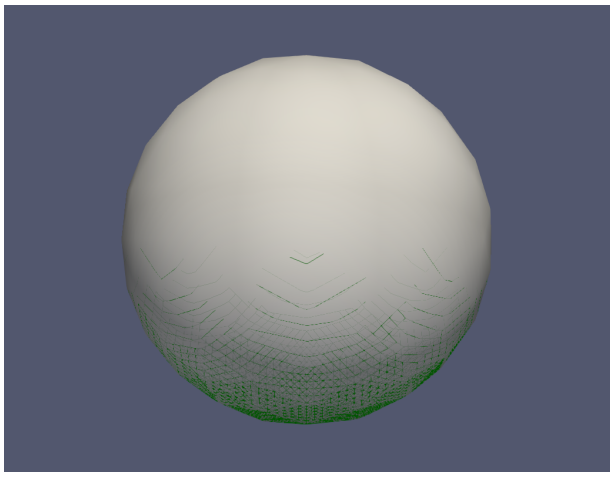
(`epsilon:double`). A `maxWeight:double` válzozó a maximalis súlyt tárolja, amit egy támasztás elbír. Az első komolyabb for-ciklus kiszámolja a sugarak metszéspontjait az alakzattal. A külső ciklusban végigmegy a mesh-ben tárolt háromszögeken. Minden alkalommal kiszámolja az aktuális háromszög csúcspontjait, majd kiszámolja a minimális és maximális x és z koordinátákat. Egy újabb ciklusban végigmegyünk azokon a rácspontokon, melyek a minimális és maximális x és z koordináták között helyezkednek el. Minden rácspont esetében kiszámoljuk a rácsponthoz tartozó egyenes és az adott háromszög metszéspontját. Ezt a következőképpen csináltam: a háromszög csúcspontjai és a metszéspont x és z koordinátái alapján meghatározott pontok közötti, kétdimenziós háromszögek területei alapján meg lehet határozni, hogy a háromszögön belül van-e a pont, azaz metszi-e a sugár a háromszöget. Ha metszi, akkor ezt hozzáadjuk a metszéspontok listájához (`intersectPoints:std::vector<Point>`), ezt a listát kezdetben két-, majd háromdimenziós tömbként kezeltem, de ebben az esetben már annyira összetett és bonyolult volt az alkalmazása, hogy inkább lecseréltem egy `std::vector<>` típusra. Így az indexelés ugyan nem volt olyan egyszerű, de minden más kezelés és használat nagyságrendekkel egyszerűsödött, és sokkal átláthatóbb lett, ami nálam fontos szempont volt. A fentebb leírt ciklusokat és számításokat megpróbáltam külön fájlban elhelyezni, de ha egy kisebb szakaszt áttettem egy másik fájlba, akkor a futási idő a normális pár másodpercről 10–15 percre nőtt. Ez persze lehet csak a saját hibámból kövezkezett. Ezért inkább itt hagytam a `main()` függvényben. A programom következő eleme az volt, hogy a metszéspontok közül kiszűrjem azokat, amelyeknek nincsen párja, azaz nem alkotnak bemenet–kimenet párt. Végezetül pedig kiírtam az `output1.obj` fájlba azokat a sugár-szakaszokat, amelyek az alakzaton belül futnak.



2.1. ábra. *A modellen belül futó szakaszok sárgával jelölve, a be- és kimeneti pontok pedig pirossal.*

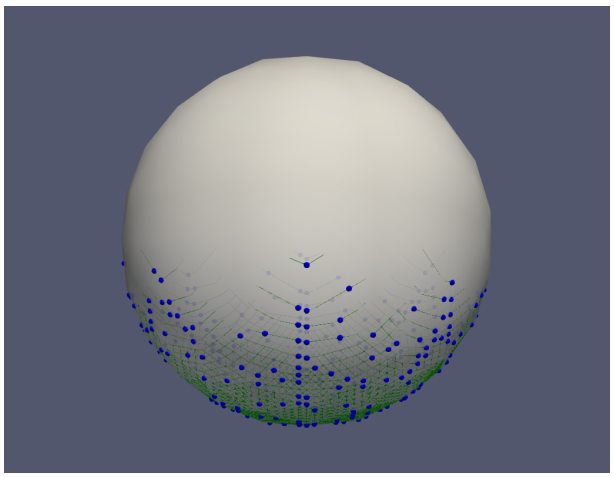
A második fázisban a bemeneti pontok közötti éleket számoltam ki és tároltam el. Először is végigmentem az összes bemeneti metszésponton. Ekkor megkerestem mindegyik szomszédját, abban az esetben, ha létezett. Ahogy korábban is említettem, itt jött elő az `std::vector<>` típus használatának a hátránya a háromdimenziós tömbbel szemben. Ugyanis a tömbök esetében egy egyszerű indexeléssel már meg is találtuk volna a szomszédokat, míg ebben az esetben kicsit hosszabb a folyamat. De ezzel a kis macerával együtt is jobban

megérte nekem így tárolni az adatokat. Ha létezik az adott szomszéd, akkor számolunk rá egy súlyt és elmentjük az élek közé. Végül pedig kiírjuk az `output2.obj` fájlba az élek által meghatározott hálót.



2.2. ábra. *A bemeneti pontok közötti élek, a gömb enyhén átlátszó, ezért a másik oldal is látszik.*

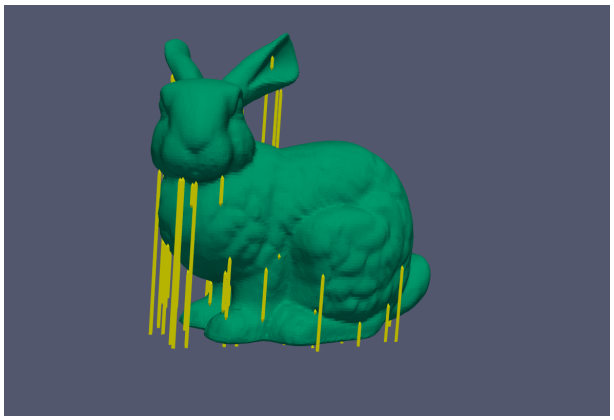
A harmadik fázis az alátámasztandó pontok kiszámításáról szól. Ez az a pont, ahol az alapul vett cikktől kicsit eltértem a megvalósítás során. Úgy csináltam, hogy először sorbarendeztem az éleket a kezdőpontjuk szerint magasság majd x és z szerint. Ezután végigmegyünk az éleken. Mindegyik végpontra kiszámolunk egy súlyt. Amennyiben az adott pont csak kezdő és nem végpont, akkor azt mindenképpen alá kell támasztani. Ha a pontot alátámasztottuk, akkor azokat az éleket töröljük is a listából amelyeket megtart a pont. Ha az adott pont nem csak kezdő, hanem végpont is a szakaszoknál, akkor viszont mindig összeadjuk az aktuális súlyt az él súlyával. Ha ez a súly nagyobb lesz a maximumnál, akkor azt a pontot már nem bírja el az alátámasztás. Fontos még arra is figyelni, hogy ha egy pontot több másik pont is alátámaszthat, akkor a kisebb súly lesz érvényben. Ha ez a ciklus lefutott, akkor



2.3. ábra. *A gömb alátámasztandó pontjait jelöltem kékkel. Ezek a pontok tartják a zölddel jelölt hálót.*

már töröltük azokat az éleket a listából, amelyeket alá kell támasztanunk az első körben.

Ekkor az a pontok súlyát (fontos hogy az élekét nem) nullázzuk és újra végigfutunk a cikluson. Ezt addig folyatjuk, ameddig el nem fogy az összes él, azaz az összes olyan pont alá van támasztva, aminek kell a támasztás. Végül pedig kiírjuk az `output3.obj` fájlba az alátámasztandó pontokat.
A negyedik fázis az előzőekkel ellentétben már jóval egyszerűbb. Itt már csak az alátámasztandó pontokból függőleges szakaszokat húzunk a tárgyasztalig, azaz a modell legalsó pontjának a síkjáig. Ezt annyiban módosítjuk, hogy abban az esetben, ha ez a szakasz elmetszi valahol az alakzatot, akkor a szakasz csak odáig tart, vagyis az alátámasztást jelképező szakasz nem haladhat az alakzaton belül. Ezeket az egyeneseket is kiírjuk az `output4.obj` fájlba.
Végezetül az ötödik és egyben utolsó fázis során generálunk minimális alátámasztó objektumokat a modellhez. Ezek nem teljesen jók, mert van ahol beleérnek kicsit az alakzatba, illetve a talpuk sem biztos, hogy elég stabil. De az elsődleges szerepük az, hogy vizuálisan is bemutassák a program végeredményét. Ezek az alátámasztó objektumok háromszög alapú oszlopok, amelyeknek a modellel érintkező végei tetraéder szerűen el vannak vékonyítva. (2.5. ábra) Ezeket az objektumokat az `output5.obj` fájlba írjuk ki.



2.4. ábra. *A modell és a generált alátámasztás.*

2.5. ábra. *Az alátámasztásokat közelebbről az alábbi képen tudjuk megnézni. Jól látszanak az elkeskenyedő végek. Fontos azonban azt megjegyezni, hogy abban az esetben, ha a támasz oszlop magassága kisebb, mint a támasz végén lévő tetraéder magassága, akkor a tetraéder nem jelenik meg.*

2.4. outputs mappa

Ebben a mappába írja ki a fájlokat a program. A ki- és bemeneti fájlok is `.obj` formátumúak. Ezek a fáljok 3D alakzatok tárolására alkalmasak. A fájok v vertexeket, l éleket és f oldalakat tartalmaznak.
A fájlok nevei a következők:

- `output1.obj` – A modell belsejében futó sugár-szakaszokat tartalmazza.
- `output2.obj` – A bemeneti pontok közötti éleket tartalmazza.
- `output3.obj` – Az alátámasztandó pontokat tartalmazza.
- `output4.obj` – Az alátámasztó struktúra oszlopainak helyén lévő egyeneseket tartalmazza.
- `output5.obj` – Az alátámasztó struktúra oszlopait tartalmazza.

2.5. modells mappa

Itt vannak a tárolt modellek, amiket be tudunk olvasni a programba. Ezek a modellek `.obj` formátumúak. A programban alapértelmezettként meg van valósítva pár direktíva, amik egy egyszerű `#ifdef` után már be is állítanak egy-egy alakzatot demónak. Direktíva meg van írva egy nyúl, egy gyémánt, egy gömb és két szobor beolvasására.

2.6. Dokumentáció

A feladat során minden egyes függvényhez és struktúrához kommenteltem a leírását, paramétereket és a funkciókat Doxygen kompatibilitással. Ugyan nem volt folyamatosan karbantartva, de a projekt repository-ja elérhető az alábbi linken: `https://github.com/paleros/Temalaboratorium-OpenMesh.git`

3. fejezet

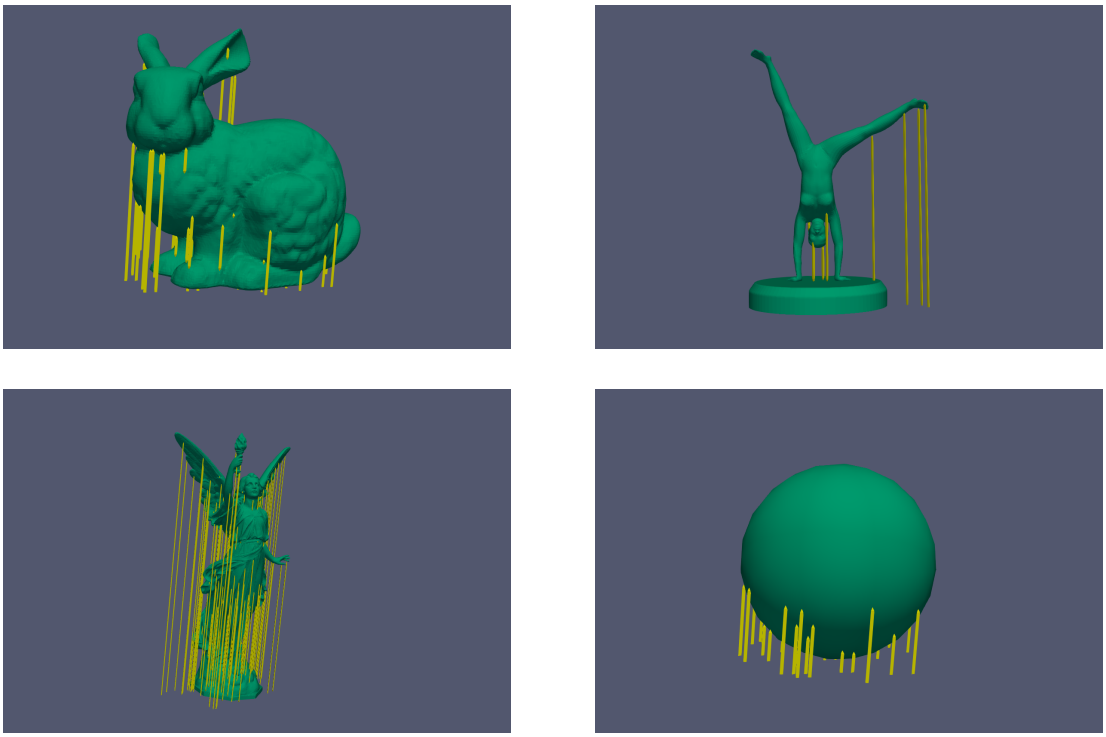
Eredmények

3.1. Kimeneti fájlok

Egy ilyen feladat esetén nehéz ellenőrizni, hogy a programunk helyesen működik-e. Az egyik lehetőség az, hogy a kimeneti fájlokat megjelenítjük egy 3D modell nézőben. Majd vizuálisan ellenőrizzük az előre felállított feltételeink mentén. Feltételeknek én a következőket írtam össze:

- A sugarak egységes távolságban helyezkednek el.
- A sugarak egymással párhuzamosan helyezkednek el.
- A sugarak a modellre merőlegesen helyezkednek el.
- A sugarak kiválasztott szakaszai csak az alakzaton belül haladnak.
- A bemeneti pontok csak a modell alsó felületein helyezkednek el.
- A bemeneti pontok között a szomszédsági kapcsolatok helyesek.
- Az alátámasztási pontok arányosan helyezkednek el.
- Az alátámasztási egyenesek a modell alsó felületén, függőlegesen helyezkednek el.
- Az alátámasztások a modell alsó felületének a síkjáig érnek, vagy az alakzatig.
- Az alátámasztás elemei az alátámasztási pontokba futnak.
- Az alátámasztás elemei merőlegesek a tárgyasztalra.
- Az alátámasztás elemei háromszög alapú oszlopok.
- Az alátámasztás elemei a modellel érintkező végpontjain elkeskenyedik.

Ezeket a feltételeket vizuálisan ellenőriztem. És ebből azt a következtetést vontam le, hogy a programom hozza a várt eredményt.



3.1. ábra. *A különböző modelleken különböző paraméterekkel generált alátámasztások.*

3.2. Nyomtatás

A másik lehetőség az lenne, hogy a gyakorlatban kipróbáljuk. Vagyis a modellt a generált alátámasztással együtt kinyomtatjuk egy nyomtatóval. Ez a módszer viszont jelenleg nem biztos, hogy sikeres lenne minden alakzat esetében, ugyanis az alátámasztó objektumnak több hiányossága is van egy valódi alátámasztással szemben. Ezeket korábban már részleteztem. Ennek ellenére szerettem volna egy egyszerűbb modellel kipróbálni, de sajnos pont a héten ment tönkre a nyomtató, amivel nyomtattam volna. Később lehet, hogy megpróbálom azért. A modellt viszont megnéztem az *UltiMaker Cura* [4] programban, és nyomtathatónak ítélte meg.

4. fejezet

Összegzés

Összességében én elégedett vagyok a munkámmal. A feladatot szerintem sikerült teljesítenem. Az előre kitűzött célokat megvalósítottam. A program ugyan nem tökéletes, de a feladatot elvégzi. Lehetne tovább fejleszteni. Egyrészt az optimalizáltságán is lehetne javítani, hogy a futási idő csökkenjen, bár most sem túl hosszú. Az alátámasztó objektumok esetében érdemes lehetne a hibákat kijavítani. Itt gondolok az alakzatba beleérő oszlopokra és a túl kicsi alátámasztásokra. Érdekes lehet esetleg egy más típusú alátámasztó objektumot is kipróbálni. Ezek mellett nekem kimondottan tetszett ez a feladat. Volt benne valamennyi elmélet, de nem túl sok. A gyakorlati alkalmazáson volt a hangsúly, ami hozzám közelebb áll. A haladás és a végeredmény is mindig látványos volt. Szerintem a tudásom is gyarapodott a feladat során. Én élveztem a munkát és örülök, hogy ezt a feladatot választottam.

Irodalomjegyzék

[1] Hubs a protolabs company. What are supports in 3d printing? when and why do you need them? https://www.hubs.com/knowledge-base/supports-3d-printing-technology-overview/, 2021.

[2] Seongje Jang, Byungjin Moon, and Kunwoo Lee. Free-floating support structure generation. *Computer-Aided Design*, 128(102908), 2020.

[3] Marco Livesu, Stefano Ellero, Jonàs Martínez, Sylvain Lefebvre, and Marco Attene. From 3d models to 3d prints: an overview of the processing pipeline. *STAR – State of The Art Report*, 36(2), 2017.

[4] UltiMaker. Ultimaker cura 5.0.0. https://ultimaker.com/software/ultimaker-cura/, 2022.