BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

GÉPÉSZMÉRNÖKI KAR

MECHATRONIKA, OPTIKA ÉS GÉPÉSZETI INFORMATIKA TANSZÉK

Horváth Botond

TDK Dolgozat

Általánosított Cartesian 3D nyomtató szimulációja

Konzulens (Külső):

Dr.Galambos Péter

Technikai igazgató

Konzulens (Felelős):

Décsei-Paróczi Annamária

Tanársegéd

Budapest, 2023.

Tartalomjegyzék

Előszó

A technológia fejlődésével egyre inkább teret nyernek a kiberfizikai rendszerek. A virtuális valóság ma már nemcsak ipari alkalmazásokban, vagy a sokak által kedvelt videójátékokban, hanem az oktatásban is számos helyen megjelenik, ezzel segítve a különböző folyamatok mélyebb megismerését.

Ezen dolgozat témája az általános 3D nyomtató szimulációjának megvalósítása a MaxWhere nevű háromdimenziós szoftver segítségével, mellyel bármely Cartesian nyomtató működése bemutatható annak 3D modelljeinek és URDF leírásának megadásával, ezzel egy könnyen elérhető és kezelhető demonstrációs eszközt nyújtva az oktatás támogatásához.

~ ~ ~

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani a Budapesti Műszaki és Gazdaságtudományi Egyetem oktatóinak az átadott tudásért, a MaxWhere Solutions Zrt. munkatársainak a projekt feltételeinek biztosításáért, valamint az Európai Únió Erasmus+ programjának támogatásáért.
Külön köszönettel tartozom dr. Galambos Péternek a projekt irányításáért, Drexler Kristóf Kenéznek a projekt megalapozásáért, és Décsei-Paróczi Annamáriának az útmutatásért.

Budapest, 2023. november 5.

Horváth Botond

Jelölések jegyzéke

Jelölés	Megnevezés, megjegyzés, érték	Mértékegység
n	futó változó	–
P_n	Bézier-görbe n-edik pontja	–
$L_n(t)$	P_n-et P_{n+1}-vel összekötő szakasz	–
$Q_n(t)$	L_n és L_{n+1} szakaszok közti lineáris interpoláció	–
$B_n(t)$	n-ed fokú Bézier-görbe	–
R	x tengely körüli elfordulás $(Roll)$	rad
P	y tengely körüli elfordulás $(Pitch)$	rad
Y	z tengely körüli elfordulás (Yaw)	rad
$\hat{n}$	általános forgatás tengelye	–
n_x	$\hat{n}$ vektor x tengely irányú komponense	m
n_y	$\hat{n}$ vektor y tengely irányú komponense	m
n_z	$\hat{n}$ vektor z tengely irányú komponense	m
Θ	forgásszög	rad
w, x, y, z	kvaterniók	–

1. fejezet

Bevezetés

1.1. A dolgozat célkitűzése

Dolgozatom célja egy olyan általános virtuális szimuláció létrehozása, mellyel bemutatható bármely Cartesian 3D nyomtató működése. Ehhez nincs egyébre szükség, mint az adott nyomtató modelljeire, illetve annak mechanizmusát leíró URDF (Unified Robotics Description Format) fájlra, a program pedig ez alapján fel tudja építeni a gépet, és helyesen működteti is azt, ezzel lehetővé téve tetszőleges nyomtatópéldány prezentálását.

Egy ilyen alkalmazás nagy segítség lehet az oktatásban az additív gyártási folyamatok bemutatásához, mivel tetszőleges Cartesian nyomtatók korlátlanul megjeleníthetők vele egy virtuális tanteremben, nem szükséges ehhez valós labor fenntartása, ezzel sokkal egyszerűbbé, könnyebben hozzáférhetővé és költséghatékonyabbá téve a demonstrációt, nem beszélve a virtuális térben történő tanulás pozitív hatásairól. [20] [6]

1.2. Felhasznált szoftverek

A virtuális szimuláció a MaxWhere alkalmazásban készül, ez bárki számára ingyenesen elérhető [25]. A honlapon látható bemutató terekhez hasonló a felépítése a dolgozat során megvalósult programnak is.

A MaxWhere szimuláció programját *TypeScript*-ben, a vezérléshez használt weblapot *HTML*-ben *JavaScript*-tel kiegészítve, míg a projekt dokumentációját *MarkDown*

formátumban írom, mindegyik esetben a Visual Studio Code fejlesztőkörnyezet lesz a segítségemre.

Képekből 3D modellek extrudálása a Blender alkalmazásban könnyen megvalósítható. A nyomtatók 3D-s modelljeit is Blenderből lehet exportálni az arra kifejlesztett MaxWhere bővítménnyel, hogy meg lehessen jeleníteni azokat a MaxWhere-térben.

A saját nyomtatni kívánt modellek tervezéséhez az Autodesk Inventor CAD szoftvert használom. Logók modellezéséhez az Inkscape grafikus szoftver jó választásnak bizonyul. Ahhoz, hogy az elkészült modellekből nyomtatható G-kód szülessen, az Ultimaker Cura szeletelőszoftvert hívom segítségül.

1.3. Áttekintés

A *2. fejezet* irodalomkutatást tartalmaz a 3D nyomtatókról, annak fajtáiról és az általuk használt technológiákról, valamint oktatási célú virtuális demonstrációs lehetőségekről. Ezen témákat dolgozatomban mélyebben érintem, a kutatás során gyűjtött információk tudatában határozott elképzeléssel vághatok neki a program készítésének.

A *3. fejezet* az általános 3D nyomtató URDF modell alapján történő felépítését és megjelenítését mutatja be.

A *4. fejezet* a vezérlőpanel összeállításáról és működéséről szól, szó esik webfejlesztésről és IPC kommunikációról is.

A *5. fejezet* a nyomtató részeinek mozgatását, a hőmérséklet változásának szimulálását, majd annak fényében a ventilátor irányítását, valamint a nyomtatvány megjelenítését ismerteti.

A *6. fejezetben* saját modellek tervezése és nyomtatása valósul meg valós és virtuális környezetben, ezáltal bemutatható a modell sikeressége.

A *7. fejezet* foglalja magában az összegzést, az eredményeket, illetve a javaslattételt a szimuláció továbbfejlesztésével kapcsolatban.

2. fejezet

Irodalomkutatás

2.1. A 3D nyomtatás, annak fajtái és az általuk használt technológiák

2.1.1. Additív gyártási folyamatok

A különböző gyártási folyamatokban közös, hogy a termék egy 3D-s modell geometriája alapján készül el. A legtöbb *CAM* (Computer Aided Manufacturing) eljárás szubsztraktív, vagyis lebontó módon hozza létre a megmunkált darabot forgácsolással, marással, fúrással, csiszolással vagy vágással, de egyre elterjedtebbek az additív gyártási folyamatok is, ezeket egyszerűen 3D nyomtatásnak is nevezik, melyek esetén a modell a szó szoros értelmében felépül különböző technológiák felhasználásával.

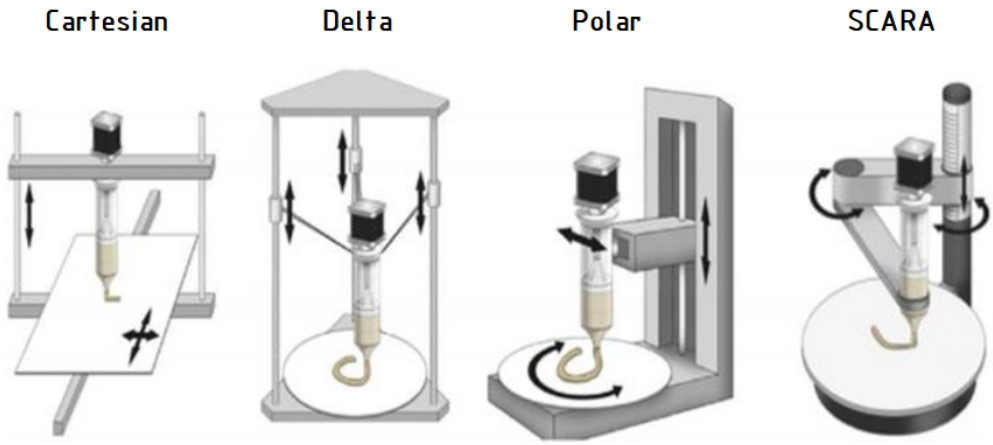
2.1.2. 3D nyomtatók fajtái

A SCARA (Selective Compliance Assembly Robotic Arm) nyomtató a hasonló elven működő robotról kapta a nevét. Egy vagy több, az emberi karhoz erősen hasonlító robotkar hajtja végre a mozgást az $X - Y$ síkban. Az asztal lehet kör- vagy téglalap alapú, a sebessége viszonylag gyors, és a minimalista váznak köszönhetően több hely jut a nyomtatásnak, azonban fontos megjegyezni, hogy ezen eszközök kalibrálása sok időt vehet igénybe, így nem annyira népszerűek. [34]

A polár nyomtatók a többivel ellentétben hengerkoordináta-rendszert használnak. A kör alakú asztal forgatásával történik az $X - Y$ síkban a pozicionálás, a nyomtatófej pedig a Z tengely mentén mozog. A kevés motor használata miatt a folyamat halk, ugyanakkor lassú, és alacsonyabb minőségű. [30]

A delta nyomtató a gyors *pick and place* műveletekhez kialakított delta robotok elvén működik, tehát a nyomtatófej három (vagy három pár) mozgó karra van erősíve, melyek egymástól függetlenül mozgatják a térben pozicionálás során. A munkaterület henger alakú, korlátozott alapterületű, viszont magas, így alkalmas vékony, de hosszú testek (pl.: tornyok, oszlopok) készítésére. A nyomtatófej kicsi és könnyed, így nagyon gyors és pontos, a mechanikájából adódóan pedig gördülékeny és látványos a nyomtatás, azonban meglehetősen drága. Különlegessége, hogy akár kis felületen felfekvő, döntött modellekhez is alkalmazható. [4]

A Cartesian nyomtató legnagyobb előnye a mechanizmus egyszerűségében rejlik, kiemelkedő népszerűségét és alacsony árát is ennek köszönheti, a legtöbb 3D nyomtató derékszögű koordinátarendszerben dolgozik. Ebben az esetben a modell minden egyes pontjához három érték (x, y, z) tartozik, mely meghatározza a térben elfoglalt helyét. A sok tengely és motor miatt nagyobb az esély ugyan pontatlanságra, de egyszerűen kalibrálhatók, így a nagyobb hibák kiküszöbölhetők.
Mivel ez a legelterjedtebb és legátláthatóbb mechanizmus, a dolgozatomban is ezzel foglalkozom részletesebben. [39]



2.1. ábra. 3D nyomtatók fajtái [16]

2.1.3. 3D nyomtatás technológiák

A nyomtatók működése nemcsak felépítésükben, de az általuk használt technológiában is különbözhetnek, az egyes technológiák más-más anyagokkal, sebességgel, felületi minőségggel vagy költséggel dolgoznak. Az ASTM Standard szerint hét csoportba sorolhatjuk a 3D nyomtatási technológiákat. [36]

A Binder Jetting (BJ) elsősorban fém 3D nyomtatási technológia, de polimerekkel és kerámiával is alkalmazzák. A por alapanyagot rétegről rétegre teríti, és folyékony kötőanyaggal szilárdítja meg végleges formájára, majd - hogy a termék megfelelő keménységű legyen - kemencébe kell helyezni. A maradék port el kell távolítani a felületről, és érdemes felületi utómunkát végezni.

A Directed Energy Deposition (DEP) technológiát főképp már meglévő tárgyak javítására vagy kiegészítésére használják. Az alapanyag – mely lehet por vagy huzal – hevítése és adagolása is a nyomtatófejben történik. Főként ezt is fémes anyagokkal használják, de szóba jöhetnek polimerek és kerámiák is.

A Materials Jetting (MJ) eredetileg polimer nyomtatási technológia, de kerámiák és kompozitok esetén is jól használható. Működési elve hasonló a tintasugaras 2D-s nyomtatókéhoz, miszerint a nyomtatófej UV fény hatására szilárduló anyag cseppjeinek injektálásával építi fel a rétegeket.

Vat Photopolymerization (VPP) esetén nyomtató fényre keményedő (*fotopolimer*) műgyanta alapanyagból hozza létre a rétegeket, jellemzően lézer vagy UV megvilágítással. A legismertebb VPP technológiák az SLA (Stereolithography) és a DLP (Digital Light Processing), előbbinél lézerfénnyel, utóbbinál ívlámpával történik a levilágítás. Az üreges, illetve kilógó részeket támaszanyaggal (support) együtt nyomtatják ki, ez később könnyen eltávolítható. A termék jellemzően fejjel lefelé kerül kinyomtatásra.

Powder Bed Fusion-nél (PBF) por állagú alapanyagot használnak, melynek megolvasztása elektronsugár vagy lézer segítségével történik. A legismertebb ilyen technológia az SLS (Selective Laser Sintering), de gyakori még az EBM (Electron Beam Melting) és az SHS (Selective Heat Sintering) is. Az anyag adagolásának köszönhetően sok esetben használhatóak nem nyomtatott alkatrészek javítására is, legyen szó polimerekről, fémekről, kerámiákról vagy kompozitokról.

Sheet Lamination (SL) során az alapanyag kivágott lemezek formájában kerül feldolgozásra, melyeket rétegenként összeillesztenek, és ragasztóval vagy ultrahangos hegesztéssel rögzítik. Ilyen eljárások a LOM (Laminated Object Manufacturing) és a UAM (Ultrasound Additive Manufacturing). Ez a technológia viszonylag olcsó, de a többi módszerhez képest kevésbé pontos.

A Material Extrusion (ME) a legolcsóbb, legelterjedtebb formája a 3D nyomtatásnak, és egyúttal a legkövethetőbb is, hiszen a lényege, hogy a megolvasztott alapanyagot – mely főként polimer, de lehet beton, élelmiszer, vagy élő szövetek – rétegről rétegre felviszi, így alakul ki a kívánt modell. Széles körben elterjedt otthoni felhasználásra is, tehát célszerű ezt a technológiát bemutatni, a dolgozatban is olyan nyomtatók kerültek szimulálásra, melyek ezt használják. Bizonyos esetekben megoldható akár többszínű, illetve többféle anyagból összetevődő termék készítése is, és ha szükséges, támaszanyagot is használhatnak. A legismertebb ilyen eljárások az FFF (Fused Filament Fabrication) és az annál jobb szilárdságú és felületi minőségű FDM (Fused Deposition Modeling).

2.2. Oktatási célú virtuális demonstrációs lehetőségek

Az oktatásban is igyekeznek kihasználni a technológia fejlődését, egyre gyakrabban alkalmaznak virtuális szimulációkat, melyek megkönnyíthetik az ismeretek átadását. A koronavírus járvány, és az akkor szükségessé vált távolléti oktatás fel is gyorsította azt a folyamatot, mely során a különböző virtuális demonstrációk megjelennek az iskolák és egyetemek eszköztárában. Ezeknek előnye, hogy bárhonnan, bármikor elérhetőek, és sokkal alacsonyabb költségvetésűek, mint a valós hardverek, kísérletek, vagy bemutatók. Ez sem utolsó szempont, már csak azért is, mert sok esetben gazdaságilag megterhelő jól felszerelt laborok beszerzése és fenntartása az intézményeknek. Mi több, az oktatásban résztvevőkre is pozitív hatással lehetnek ezek a szimulációk, az ausztráliai Deakin University felmérése szerint ezek a módszerek a diákok 93%-át motiválták arra, hogy optometrista váljon belőlük, és 77%-ukat ösztönözte, hogy a tananyagon túl is utánanézzenek az adott témának. [14] Ezek az eszközök lehetőséget biztosítanak arra, hogy a diákok jobban elmélyülhessenek az egyes demonstrációk tanulmányozásában, hiszen a tanórán kívül is használhatják azokat.

A 3D virtuális térben történő oktatás elősegíti a magasabb rendű gondolkodási készségek kibontakozását, mint az összefüggések észrevétele, ismeretek alkalmazása vagy elemzése. Lehetőség nyílik olyan környezet kialakítására, amit a valóságban sokszor nem állna módunkban biztosítani a tanulóknak, ilyen lehet egy legmodernebb technológiával felszerelt laboratórium, vagy egy ipari üzem. További előnye ezeknek az alkalmazásoknak, hogy a 3D-s környezetben történő mozgás egy új agyterületet, a paterális lebenyt aktivizálja, ezzel új utat nyitva az információ észlelésére és feldolgozására. Horváth Ildikó [20] alátámasztja, hogy a 3D-terekben tevékenykedő diákok munkájának hatékonysága 30%-kal, az emlékezés 50%-kal javul, a tananyag átlátása pedig 50%-kal gyorsabb, ezzel az ismeretszerzés egy új módja alakulhat ki.[6] [21] [18] [11] A továbbiakban az oktatásban is használt 3D virtuális demonstrációs eszközöket veszem górcső alá.

Nagy szerepe van a különböző szimulációknak a természettudományok elsajátításának sikerességében, és ezt egyre több iskolában előszeretettel használják: ma már a fizika, kémia, matematika, biológia és a földrajz területén is találkozhatunk hasznos és magas színvonalú anyagokkal, erre példa a PhET [8] és a 3jcn [2] oldala is.

Emellett egyre nagyobb szükség van a virtuális szimulációkra az orvostudományban is, ezzel felkészítve a hallgatókat a rájuk váró kihívásokra, különböző vizsgálatokat végezhetnek és klinikai problémákat oldhatnak meg a virtuális páciensen, akár többen is egyszerre valós időben együttműködve. Azzal, hogy mindez a virtuális térben történik, lehetőség van arra, hogy a hallgatók saját hibáikból tanuljanak, bátrabban próbálhatnak ki új dolgokat, nem lesznek súlyos következményei, ellentétben a valós kórházi helyzetekkel. Ilyen lehetőségeket kínál például a VMS [41] oldala is.

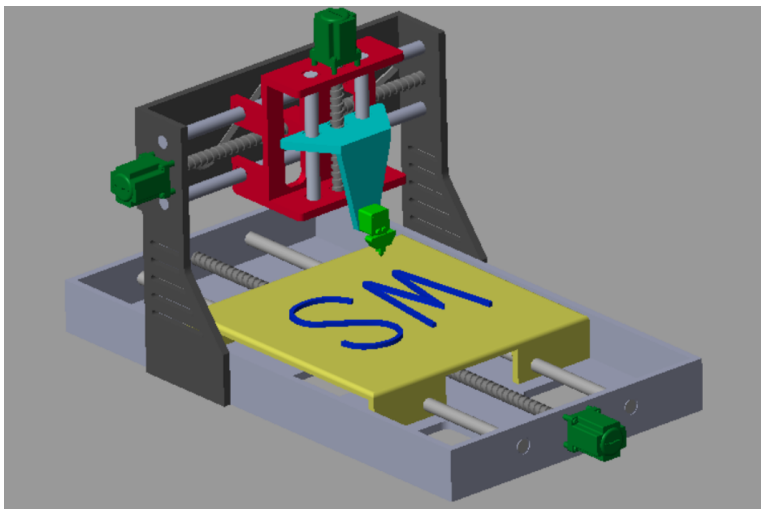
A járművezetés oktatására is vannak ma már jól használható szimulációs programok, mint például a VIRTUAL DRIVING SCHOOL [40]. Ezen szoftverek használatával a technikai tudást még ugyan kevésbé lehet elsajátítani, ám az az egyszerűbb része a tanulásnak, azonban sokszor ritka, de annál veszélyesebb helyzeteket szimulálva fejleszti a tudatos viselkedést, a döntéshozatali képességet, a kommunikációt és együttműködést a közlekedés többi résztvevőjével, illetve a stressz és fáradtság kezelését. Tehát ezeknek a szoftvereknek nagy szerepe lehet abban, hogy felkészítse a tanulókat a forgalomban potenciálisan megjelenő kritikus szituációkra, és annak megoldására, ezáltal a kezdő sofőrök nagyobb gyakorlattal kerülhetnek ki az utakra, ezzel növelve a mások és saját

biztonságukat is, ugyanis az köztudott, hogy a 25 év alatti sofőrök arányosan kétszer annyi közúti balesetet szenvednek, mint a náluk idősebb korosztály, és ennek jó része a tapasztalat hiányából fakad. [9] [33]

Ahol még különösen fontos az ehhez hasonló virtuális szimulációk használata, az a mérnöki és ipari ágazat, különösen a gépészet. A gépek, berendezések, alkatrészek, termékek viselkedésének modellezése rendkívül fontos, hiszen számos hiba kiszűrhető már a tervezési fázisban, ezzel rengeteg időt, energiát, és pénzt megtakarítva. Ilyenek például az egyetemeken is előszeretettel alkalmazott VEM (végeselem-módszer) programok, amelyekben szerkezeti szimuláció végezhető. Az ipari folyamatok tervezésével, tesztelésével, analizálásával, illetve optimalizásával, az ipari környezet kialakításával foglalkozó szimulációk is jó szolgálatot tesznek, hiszen még az éles bevetés előtt próbára tehetjük az új rendszert. Találkozhatunk ilyen szoftverekkel is a Gépészmérnöki Karon, mint például a Plant Simulation [28], vagy a Process Simulate [29].

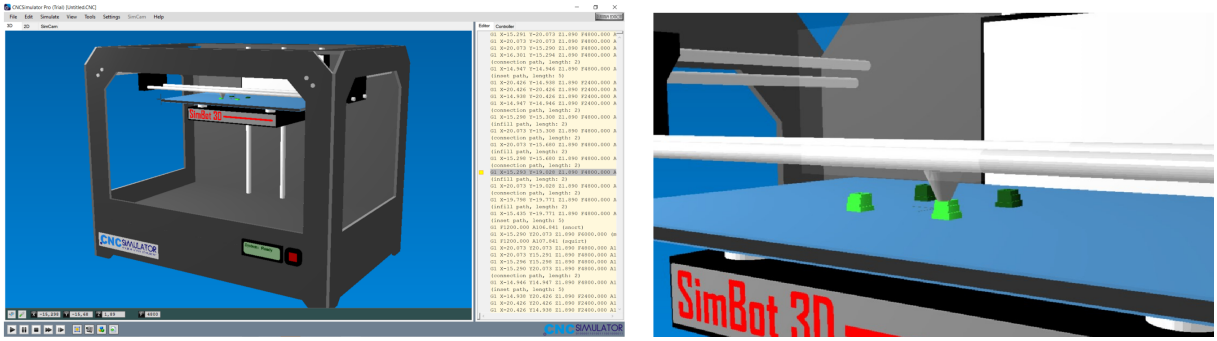
A gyártási folyamatokat is célszerű először szimulációs szoftver segítségével kipróbálni, ezzel elkerülve az esetleges selejtes darabok elkészítését, vagy a berendezések sérülését. A szubsztraktív gyártási folyamatokhoz kiválóan használható az NCT Szimulátor [27], melyet használtunk is maráshoz, illetve esztergáláshoz készült G-kód teszteléséhez több tantárgy keretein belül, de a CNCSimulator is jó választás lehet erre a célra.

Az additív gyártáshoz kevésbé áll rendelkezésre ilyen mélységű demonstrációs lehetőség, iyennel nem is találkoztam tanulmányaim során. Fellelhető ugyan a célnak megfelelő bemutató például a Matlab-ban, az `openExample('sm/Cartesian3DPrinterExample')` kód futtatásával előhívható a Cartesian 3D Printer szimuláció, amely valójában csak a 3D nyomtató egyes tengelyek mentén történő mozgásának vizsgálatára alkalmas egy előre definiált forma mentén, de mi magunk már nem tudnánk saját kódot feltölteni, vagy a nyomtatvány készülését követni. [10]



2.2. ábra. Matlab 3D nyomtató

A CNCSimulator [19], mely honlapja szerint a világon az első [5] 3D nyomtató szimuláció volt, már kielégítőbb megoldást ad a problémára, itt egy valósághűbb nyomtatót talál a felhasználó, maga adhatja meg a nyomtatni kívánt G-kódot, a vezérlőpanelen el lehet indítani, illetve meg lehet állítani a szimulációt, és a sebesség is változtatható. Míg fut a program, kijelzi a felület, hogy éppen melyik mondatot hajtja végre a G-kódból. Hiányzik azonban annak a lehetősége, hogy egyénileg válasszunk nyomtatót, illetve a nyomtatvány megjelenítése is kezdetleges, a nyomtató modellje pedig csak lebeg a virtuális térben, és nem igazán élethű a megjelenése, az alkatrészek sematizáltak, érződik, hogy ez csak egy szimulációs környezet, de tény, hogy a folyamat bemutatására, az elkészült kód futtatására alkalmas, ennél azonban látványosabb modellt szeretnék létrehozni.



2.3. ábra. CNCSimulator 3D nyomtató

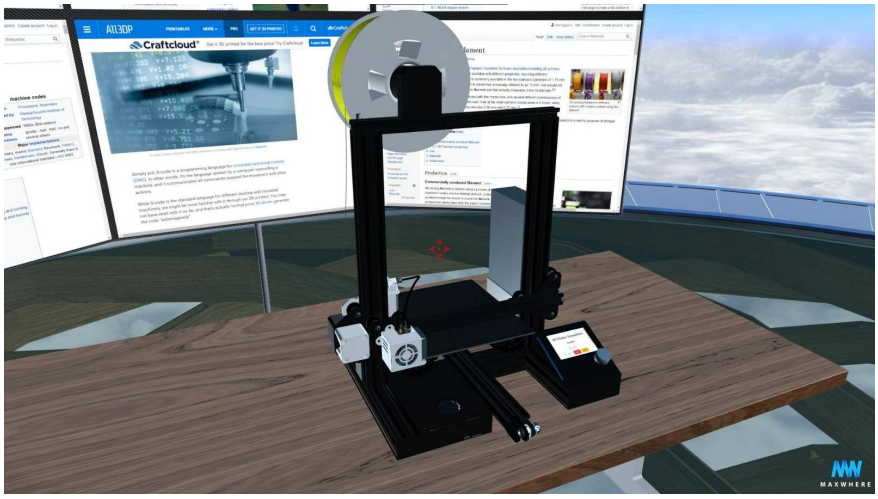
3. fejezet

Az általános Cartesian 3D nyomtató megjelenítése a MaxWhere-térben

3.1. Mi az a MaxWhere?

A virtuális teret a nyomtató szimulációjával a MaxWhere alkalmazásban valósult meg. Ez a 3 dimenziós szoftver lehetőséget nyűjt tetszőleges terek létreoözására, legyen szó iskolai, ipari, vagy akár otthoni felhasználásról. [25]

Maga a 3D nyomtatós tér egy működő komponenssel már elkészült mire elkezdtem ezzel a projekttel foglalkozni, ennek Drexler Kristóf Kenéz kollégám volt a szerzője, ez egy Ender nyomtató szimulációját valósította meg. [13]



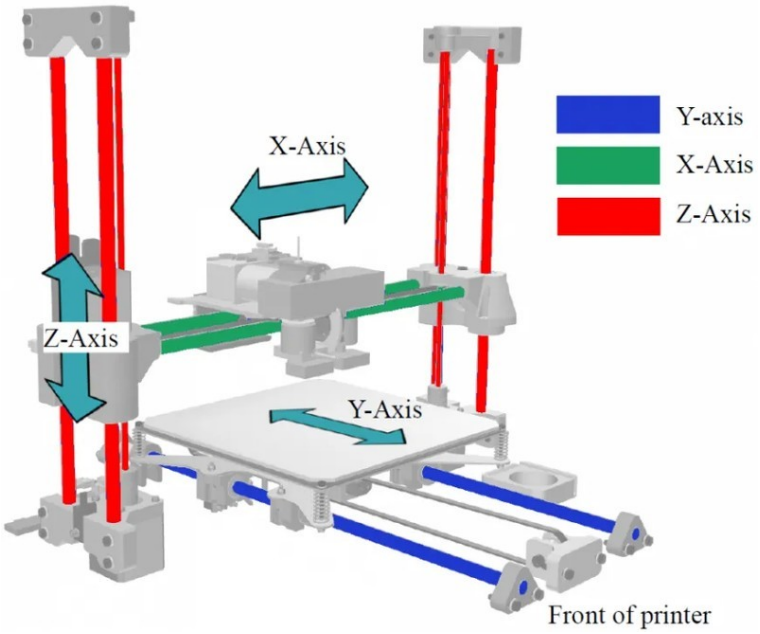
3.1. ábra. A nyomtató komponens kiindulási állapota [13]

Az én feladatom ezen program továbbfejlesztése, általánosítása volt, annak érdekében, hogy bármely Cartesian 3D nyomtatót, illetve azokból is bármennyit megjeleníthessünk a virtuális tantermünkben, ezzel lehetővé téve a különböző eszközök összehasonlítását, vagy hogy bárki megalkothassa saját nyomtatójának digitális mását, legyen szó magánszemélyről, vagy gyártóról. Szándékom volt továbbá bizonyos funkciókkal kiegészíteni a programot, ennek módját a későbbiekben bővebben kifejtem.

3.2. Nyomtató felépítése URDF modell alapján

A térben az általános 3D nyomtató megjelenítése URDF (Unified Robot Description Format) modell alapján megvalósíthatónak bizonyult. A cél az volt, hogy a felhasználó bármilyen Cartesian nyomtatót el tudjon helyezni, és az megfelelően működjön minden esetben. [24] [17] [38]

A Cartesian nyomtatók esetén tehát a három tengely (X, Y, Z) mentén lineáris mozgásokkal történik a nyomtatás, így az egyes elemeket - melyek gyakorlatilag egy-egy tengelyt jelentenek - transzlációs csuklók kötik össze egymással.



3.2. ábra. A Cartesian nyomtatók általános tengelyei [3]

3.2.1. Link-*ek felvétele*

A URDF modellben először is a `<robot>` *tag*-en belül fel kell venni a négy fő részegységet, ez pedig a gépalap, illetve a három tengelyt megvalósító alkatrész. Az egyes *link*-ekhez alapértelmezetten minden paramétert nullának tekintünk, és nem is szükséges semmi egyebet megadnunk, mint a *link* nevét, illetve a *mesh* fájljának nevét. A felhasználandó fájlokat a komponens egy előre megadott mappájába, a `resources_printer` mappába kell elhelyezni, hogy a komponens elérje azokat csupán a fájlnév alapján.

Szükség van további *link*-ekre a nyomtató élethű működéséhez. Ugyan a fő funkciók már így is jól reprezentálhatók, nem árt bizonyos dolgokra még figyelni, melyek minden nyomtatónál megjelennek.

Ki kell jelölni a megfelelő működés érdekében, hogy az asztal funkcióját melyik részegység tölti be, ehhez pedig fel kell venni `table` névvel egy *link*-et. Mivel ehhez nem tartozik újabb test, ezért fájlnév megadása felesleges.

A filamenttel feltöltött dob modelljét is érdemes megjeleníteni, ezt `Spool` névvel egységesen kell rögzítenünk, és fel kell tüntetni az ennek megfelelő fájl nevét is.

Mivel minden nyomtatóhoz saját vezérlőpanel tartozik, ennek is helye van a URDF struktúrában `controlPanel` néven. Ehhez sem tartozik *mesh*, ez csak az vezérlőpanel szerepét ellátó okostábla helyzetét jelöli majd ki.

Amennyiben szeretnénk a nyomtatófejbe hűtést, be kell tenni a modellbe egy ventilátort is `fan` néven a hozzá kapcsolódó *mesh* fájllal együtt.

Létre kell hozni továbbá segédpontokat a dobról lelógó filament, illetve a filamentet a fejhez vezető cső megjelenítéséhez. A megadott pontokra görbe illeszthető, a görbe mentén pedig létrejöhet a csőgeometria. Erre a célra a *Bézier-görbe* kézenfekvő választás lehet, ez egy a számítógépes grafikában gyakran használt parametrikus görbe. Tetszőleges fokszámú *Bézier-görbe* képezhető a célnak megfelelően. [37] [7]

Az elsőfokú esetben adott két pont (P_0, P_1), és az ezeket összekötő szakasz lesz a görbe, ez gyakorlatilag megegyezik a P_0 és P_1 pontokatn összekötő szakasszal a *3.1* egyenlet szerint, ezért ezt *lineáris Bézier-görbének* is szokták nevezni.

$$B_1(t) = L_0(t) = (1 - t) \cdot P_0 + t \cdot P_1, t \in [0, 1] \tag{3.1}$$

A másodfokú, *kvadratikus Bézier-görbe* esetén három pont adott (P_0, P_1, P_2), a görbe a P_0 pontból indul ki P_1 irányába, és P_1 irányából érkezik P_2-be. A görbe P_0 és P_2 pontban vett érintője is áthalad P_1-en. A *3.4* egyenlet alapján interpolálunk a *3.1* és a *3.2* egyenletekben meghatározott $L_0(t)$ és $L_1(t)$ görbék között.

$$L_1(t) = (1-t) \cdot P_1 + t \cdot P_2, t \in [0,1] \tag{3.2}$$

$$B_2(t) = Q_0(t) = (1-t) \cdot L_0(t) + t \cdot L_1(t), t \in [0,1] \tag{3.3}$$

$$B_2(t) = (1-t)^2 \cdot P_0 + 2 \cdot (t-t^2) \cdot P_1 + t^2 \cdot P_2, t \in [0,1] \tag{3.4}$$

A harmadfokú, *köbös* vagy *kubikus Bézier-görbe* előállításához négy pontra van szükség (P_0, P_1, P_2, P_3), a görbe a P_0 pontból indul ki P_1 irányába, és P_2 irányából érkezik P_3-ba. A *3.6* egyenlet alapján interpolálunk a *3.2* és a *3.5* egyenletekben meghatározott $L_1(t)$ és $L_2(t)$ görbék között, majd a *3.4* egyenletekben meghatározott $Q_1(t)$ és az imént számított $Q_2(t)$ görbék között is a *3.7* egyenlet alapján.

$$L_2(t) = (1-t) \cdot P_2 + t \cdot P_3, t \in [0,1] \tag{3.5}$$

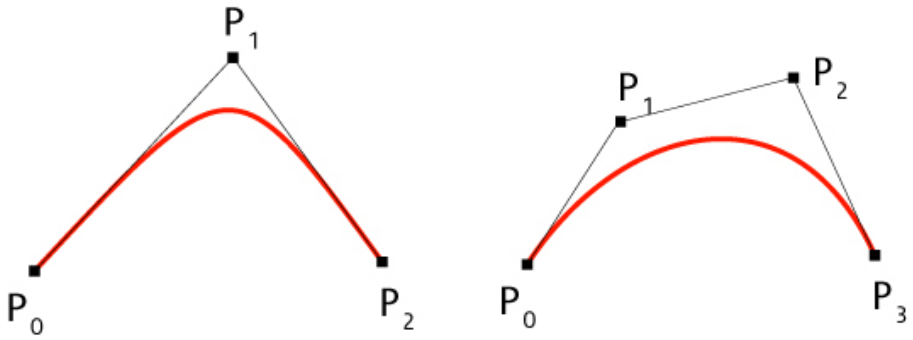
$$Q_1(t) = (1-t) \cdot L_1(t) + t \cdot L_2(t) = (1-t)^2 \cdot P_1 + 2 \cdot (t-t^2) \cdot P_2 + t^2 \cdot P_3, t \in [0,1] \tag{3.6}$$

$$B_3(t) = Q_1(t) = (1-t) \cdot Q_0(t) + t \cdot Q_1(t), t \in [0,1] \tag{3.7}$$

$$B_3(t) = (1-t)^3 \cdot P_0 + 3 \cdot (1-t)^2 t \cdot P_1 + 3 \cdot (1-t)^2 t^2 \cdot P_2 + t^3 \cdot P_3, t \in [0,1] \tag{3.8}$$

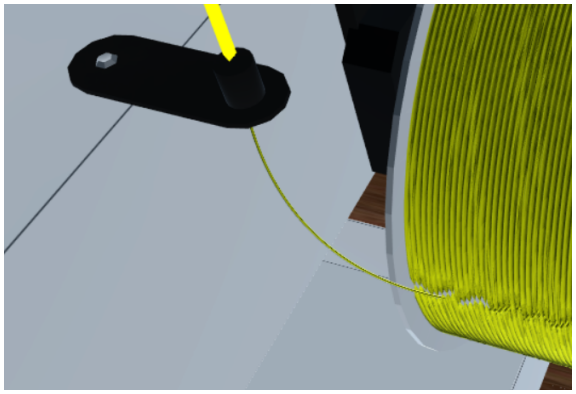
Az n-edfokú *Bézier-görbe* általános alakja felírható adott $n+1$ pont ($P_0, P_1, ..., P_n$) esetén a *3.9* képlettel.

$$B_n(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} \cdot t^i \cdot P_i, t \in [0,1] \tag{3.9}$$



3.3. ábra. A másod- és harmadfokú Bezier-görbe [1]

A harmadfokú formula már tökéletesen megfelel a célunknak, így felesleges magasabb fokú eseteket vizsgálni, a szimulációban így a *köbös Bézier-görbét* alkalmaztam a korábban említett alkotóelemek megjelenítésére a *3.4* és a *3.5* ábrákon látható módon, ehhez pedig négy-négy pont megadása szükséges.

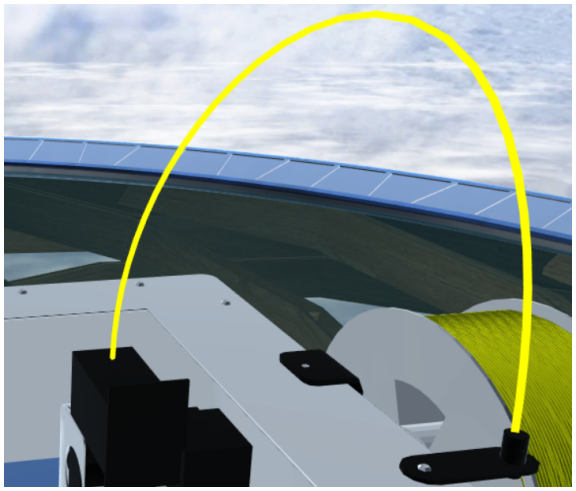


3.4. ábra. A lelógó filament

A lelógó filament kezdetét a dobon kell kijelölni, erre szolgál a `spoolDisp` pont (P_0). Annak irányát, hogy merre folytatódik a filament, a `spoolControl` pont (P_1) határozza meg. A filament végét a `zInDisp` ponttal (P_3) jelöljük ki, azt pedig, hogy milyen irányból fut be a filament, a `zInControl` pont (P_2) adja meg. Mind a négy pontot fel kell venni a megadott névvel a *link*-ek közé, *mesh file* nem tartozik hozzájuk, hiszen ezek csak segédpontként funkcionálnak.

A filament vezető cső esetén meg kell jelölni annak kezdetét, erre szolgál a `zOutDisp` pont (P_0). Annak irányát, hogy merre folytatódik a cső, a `zOutControl` pont (P_1) határozza meg. A cső becsatlakozási pontját a nyomtatófejbe a `xInDisp` ponttal (P_3) adjuk meg, hogy milyen irányból fut be a cső, azt a `xInControl` pont (P_2) adja meg. Mind a négy

pontot fel kell venni a megadott névvel a *link*-ek közé a filamenthez hasonló módon, *mesh file* ezekhez sem tartozik.



3.5. ábra. A filament cső

3.2.2. Joint-*ok felvétele*

Az alkotóelemek közötti kapcsolatot az őket összekötő csuklók határozzák meg, ezeket a *joint*-ok írják le a modellben. A gépalapot kivéve minden egyes *link*-et valamilyen csuklóval hozzá kell kapcsolnunk a géphez.

A tengelyeket vagy a gépalappal, vagy egymással kötik össze a csuklók, ez a felépítésből adódik. Mivel Cartesian nyomtatóról beszélünk, ezek prizmatikus csuklók lesznek. Fontos, hogy az irányukat úgy adjuk meg, hogy az adott tengely pozitív elmozdulásának irányát jelölje ki. Itt kell megadni azt is, hogy egymáshoz képest milyen a szülő és a gyermek elem helyzete.
A ventilátort rotációs csukló köti össze a nyomtatófejjel. Itt meg kell adni a pozitív forgás tengelyének irányát.
A többi *link* fix csuklóval van rögzítve a megfelelő szülő *link*-hez, mindig ahhoz kell kapcsolnunk, amellyel együtt kell mozognia. A `table` legyen ahhoz a tengelyhez parentálva, amely az asztal funkciót tölti be.

Az általam használt két nyomtató kész URDF modelljeit az *M.1.* és *M.3. melléklet* tartalmazza.

3.3. A URDF modell beolvasása

3.3.1. A URDF modell feldolgozása

A megadott URDF modell alapján a komponensben felépül a nyomtató szerkezete. A `Printer.ts` osztály (*M.5.melléklet*) az, ami létrehoz egy nyomtató példányt, ebben a fájlban történik a URDF feldolgozása.

Először az `XML`-fájlt beolvastam a `data` változóba a mappaszerkezetből, hogy aztán értelmezni tudjam annak tartalmát, ezt a *3.1 kódrészlet* mutatja. A `printerSource` határozza meg, hogy melyik URDF modellt használjuk fel az adott komponenshez.

```
1 // urdf modell beolvasása
2 import * as fs from 'fs';
3 const data=fs.readFileSync(resolve(__dirname,"../urdf_models", this.printerSource), 'utf8')
4 var datas= data;
```

3.1. kódrészlet. XML beolvasása

A beolvasott fájlt `json` objektumba átalakítva könnyen feldolgozhatóvá válik. Ehhez az `xml2js` [43] könyvtár `Parser()` függvényét használtam a *3.2 kódrészletnek* megfelelően. Először egy üres `string`be beolvastam a `json` formátumba átalakított fájlt, majd a `JSON.parse()` függvénnyel `json` objektumot kreáltam belőle.

```
1  // xml parser
2  import * as xml2js from 'xml2js';
3  var parser = new xml2js.Parser();
4  // xml to json
5  var printer_urdf="";
6  parser.parseString(data, function (err: any, result: String) {
7    console.log('Done');
8    printer_urdf = JSON.stringify(result);
9    return data;
10 });
11 var printer3d=JSON.parse(printer_urdf);
```

3.2. kódrészlet. XML beolvasása

A feldolgozott URDF modellből alkotott objektumon belül létrejön egy a *link*-eket és egy a *joint*-okat tartalmazó tömb. Az attribútumok a $ által jelzett objektumokba kerülnek.

Látva a `robot` objektum struktúráját, az egyes elemein végigiterálva kiszűrhetőek a szükséges információk.

Az Ender nyomtató feldolgozott URDF modelljét JSON formátumban az *M.4. melléklet*, a Craftbot nyomtatóét pedig az *M.2. melléklet* tartalmazza.

Az egyezményes nevek alapján kiválaszthatók a részegységek az objektumból egy iterálás során, ennek módját példaképp a *3.3 kódrészlet* mutatja, melynek esetén a `Spool` néven felvett dob kerül lekezelésre. Megvizsgáljuk, hogy van-e ilyen elem a *link*-ek között, ha van, elmentjük a *mesh* url-jét, majd végigiterálunk a *joint*-ok között is, és az ahhoz tartozó pozíciót, illetve orientációt is eltároljuk. Hasonlóan történik mindez a többi alkatrész esetén is a tengelyeket kivéve, annyi különbséggel, hogy a segédpontokhoz nem tartozik *mesh* fájl.

```
1   for(let i=0; i<printer3d.robot.joint.length;i++ ){
2     if(printer3d.robot.joint[i].child[0].$.link == "Spool"){
3       for(let j=0; j<printer3d.robot.link.length;j++ ){
4         if(printer3d.robot.link[j].$.name == "Spool"){
5           this.url_spool = printer3d.robot.link[j].visual[0].geometry[0].mesh[0].$.filename;
6           let origin=printer3d.robot.joint[i].origin[0].$.xyz;
7           let xyzArray = origin.split(" ");
8           let origin_x=parseFloat(xyzArray[0]);
9           let origin_y=parseFloat(xyzArray[1]);
10          let origin_z=parseFloat(xyzArray[2]);
11          this.spool_pos=new THREE.Vector3(origin_x, origin_z, -origin_y);
12          let rpy=printer3d.robot.joint[i].origin[0].$.rpy;
13          let rpyArray = rpy.split(" ");
14          let rpy_x=parseFloat(rpyArray[0]);
15          let rpy_y=parseFloat(rpyArray[1]);
16          let rpy_z=parseFloat(rpyArray[2]);
17          this.spool_rpy=new THREE.Vector3(rpy_x, rpy_z, -rpy_y);
18        }
19      }
20    }
```

3.3. kódrészlet. Base feldolgozása a URDF-ből

A tengelyeknél másképp kell eljárni, a *3.4 kódrészlet* szerint meg kell vizsgálni, hogy melyik alkatrész melyik tengely szerepét tölti be, és csak ezután lehet megfeleltetni a programban a testeket a tengelyeknek. Ez a URDF-ben eltárolt `axis` *tag* `xyz` attribútumának elemeit egy tömbbe kiszedve könnyen meghatározható, az egyes tengelyek mentén történő elmozdulások a tömb egy-egy elemének felelnek meg. Mivel Cartesian

nyomtatóról van szó, mindig egyetlen elem lesz 1, az összes többi 0. Mivel a tengelyeken kívül csak a ventilátor az, ami különállóan mozgó alkatrész, egy elágazásban le kell kezelni, hogy azt ne vegye figyelembe, így kizárólag a tengelyek lesznek feldolgozva a *3.3 kódrészlethez* hasonlóan.

```
1   if(printer3d.robot.joint[i].child[0].$.link != "fan"){
2     let xyz=printer3d.robot.joint[i].axis[0].$.xyz;
3     let myArray = xyz.split(" ");
4     let axis_x=parseInt(myArray[0]);
5     let axis_y=parseInt(myArray[1]);
6     let axis_z=parseInt(myArray[2]);
7     if(axis_x != 0){
8       ... }
9     if(axis_y != 0){
10      ...
11    }
12    if(axis_z != 0){
13      ... }
14  }
```

3.4. kódrészlet. Tengelyek kiválasztása a URDF-ből

3.3.2. Objektumok létrehozása

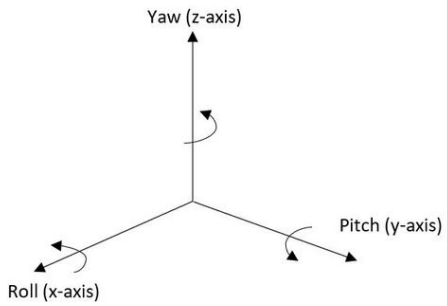
MaxWhere-térben testeket létrehozni a `wom.create()` függvénnyel lehet. Az előzőleg kinyert adatokat felhasználva már meg is lehet ezt tenni. Minden létrehozott objektum azonosítója tartalmazza a komponens azonosítóját is, ezzel elkerülve az esetleges összeakadásokat több komponens egy térben történő példányosításának esetén. A *3.5 kódrészlet* szemlélteti a gépalap létrehozását, de a többi részegység esetén is hasonlóképpen kell eljárni.

```
1   this.base = wom.create("mesh", {
2     id: `base${this.componentID}`,
3     url: this.url_base,
4     position: this.basePos.add(this.base_start_pos),
5     orientation: eulerToQuat(this.base_rpy),
6     scale: { x: this.baseScale.x, y: this.baseScale.y, z: this.baseScale.z },
7     autophysical: false
8   });
```

3.5. kódrészlet. Tengelyek kiválasztása a URDF-ből

Láthatjuk, hogy a skálázás rögzítésénél a `baseScale` objektumból lettek kiolvasva az értékek, ez csak a gépalap esetén van így, ez az adat a komponens hívásakor állítható be, és mivel a gépalap minden más részegység szülő *node*-ja, így ez adja meg gyakorlatilag a nyomtató skálázását. A tengelyekre még lehet opcionálisan skálázást megadni a komponens példányosításakor, ez azt az esetet hivatott megoldani, ha a *mesh* fájlok különböző skálázással lettek generálva.

Használtam továbbá az `eulerToQuat()` függvényt, erre azért volt szükség, mert a URDF modellben Euler-szögeket szokás használni, a MaxWhere `wom.create()` függvénye azonban kvaterniókat vár.
Egy objektumot három egymás után következő, egymásra merőleges tengelyű forgatással tetszőleges térbeli helyzetbe hozhatjuk, ezt a három forgatást a három Euler-szög adja meg (`R, P, Y`), jelen esetben az `X`, `Y` és `Z` tengelyek mentén. Az Euler-szögeket szokás Roll-Pitch-Yaw (RPY) szögeknek is nevezni.



3.6. ábra. Euler-szögek [44]

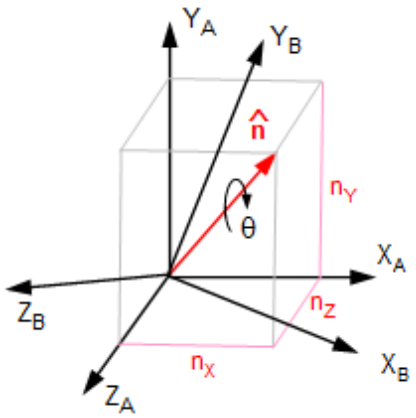
A kvaterniók a komplex számok nem kommutatív kiterjesztései négy dimenzióra, egy valós (`w`) és három imaginárius (`x, y, z`) része van. A *3.7 ábrán* látható `A` koordinátarendszert `B`-be egyetlen tengely körüli elforgatással átvihetjük, ehhez definiálni kell az $\hat{n}$ vektor n_x, n_y és n_z komponenseit, illetve a Θ forgásszöget. A kvaterniók a *3.10-3.13 egyenletek* alapján számíthatók. [26] [42] [31]

$$w = \cos\left(\frac{\Theta}{2}\right) \tag{3.10}$$

$$x = n_x \cdot \sin\left(\frac{\Theta}{2}\right) \tag{3.11}$$

$$y = n_y \cdot \sin\left(\frac{\Theta}{2}\right) \tag{3.12}$$

$$z = n_z \cdot \sin\left(\frac{\Theta}{2}\right) \tag{3.13}$$



3.7. ábra. Kvaterniók számítása [31]

Euler-szögekből van lehetőség áttérni kvaterniókra, ezt megtehetjük a *3.14-3.17 egyenletek* alapján. [32] Ezen összefüggések alapján felírható a *3.6 kódrészletben* található `eulerToQuat()` függvény.

$$w = \cos\left(\frac{R}{2}\right) \cdot \cos\left(\frac{P}{2}\right) \cdot \cos\left(\frac{Y}{2}\right) + \sin\left(\frac{R}{2}\right) \cdot \sin\left(\frac{P}{2}\right) \cdot \sin\left(\frac{Y}{2}\right) \tag{3.14}$$

$$x = \sin\left(\frac{R}{2}\right) \cdot \cos\left(\frac{P}{2}\right) \cdot \cos\left(\frac{Y}{2}\right) - \cos\left(\frac{R}{2}\right) \cdot \sin\left(\frac{P}{2}\right) \cdot \sin\left(\frac{Y}{2}\right) \tag{3.15}$$

$$y = \cos\left(\frac{R}{2}\right) \cdot \sin\left(\frac{P}{2}\right) \cdot \cos\left(\frac{Y}{2}\right) + \sin\left(\frac{R}{2}\right) \cdot \cos\left(\frac{P}{2}\right) \cdot \sin\left(\frac{Y}{2}\right) \tag{3.16}$$

$$z = \cos\left(\frac{R}{2}\right) \cdot \cos\left(\frac{P}{2}\right) \cdot \sin\left(\frac{Y}{2}\right) - \sin\left(\frac{R}{2}\right) \cdot \sin\left(\frac{P}{2}\right) \cdot \cos\left(\frac{Y}{2}\right) \tag{3.17}$$

```
function eulerToQuat(euler: THREE.Vector3):Quaternion{
  R=euler.x
  P=euler.y
  Y=euler.z;
  x=Math.sin(R/2)*Math.cos(P/2)*Math.cos(Y/2)-Math.cos(R/2)*Math.sin(P/2)*Math.sin(Y/2);
  y=Math.cos(R/2)*Math.sin(P/2)*Math.cos(Y/2)+Math.sin(R/2)*Math.cos(P/2)*Math.sin(Y/2);
  z=Math.cos(R/2)*Math.cos(P/2)*Math.sin(Y/2)-Math.sin(R/2)*Math.sin(P/2)*Math.cos(Y/2);
  return {
    w: w,
    x: x,
    y: y,
    z: z,
  };
}
```

3.6. kódrészlet. eulerToQuat() függvény

A vezérlőpanel méreteit és felbontását a komponens meghívásakor kell megadni. Ezeket, illetve a URDF modellből kinyert adatokat felhasználva létrehozható egy *billboard*, melyen tetszőleges tartalom betölthető. A *3.7 kódrészletben* látható módon az `url` attribútumban megadtam azon weblap elérési útvonalát a komponens mappáján belül, amit erre a célra készítettem, és ami ezen a táblán megjelenítésre kerül.

```
    this.controlPanel = wom.create("billboard", {
      id: `component-tutorial-wom-manipulation-billboard${options.componentID}`,
      url: resolve(__dirname, "../billboardResources/index.html"),
      nodeIntegration: true,
      width: this.controlPanelWidth,
      height: this.controlPanelHeight,
      scaleFactor: 1,
      "resolution-width": this.resolutionWidth,
      "resolution-height": this.resolutionHeight,
      position: this.controlPanel_pos,
      orientation: eulerToQuat(this.controlPanel_rpy),
      physical: { raycast: true },
    });
```

3.7. kódrészlet. controlPanel létrehozása

Itt kerülnek inicializálásra a már az eredeti programban is használt üres *node*-ok (`printBaseCube`, `printBaseTube`, `printCoordTracker`), melyek a mozgás során nyújtanak segítséget. Ebben az esetben elegendő csak egy azonosító megadása, minden más érték automatikusan nulla, és mivel a *node*-okhoz nem kötelező testet rendelni, és jelen

esetben megjelenést nem is szeretnénk hozzá társítani, még fájl elérési útvonalat sem kell megadni.

Ahhoz azonban, hogy görbét lehessen illeszteni a *Bézier-görbék* segédpontjaira, mindenképpen *mesh* fájlokat kellett létrehoznom, mivel az üres *node*-ok esetében erre nincsen lehetőség. Csináltam tehát minden ponthoz egy-egy egységkockát, melyek elenyészően kicsik, gyakorlatilag észrevehetetlenek, így a szimulációt nem zavarják meg, de ellátják a feladatukat.

3.3.3. Görbék generálása

A filament és a cső vizualizálásához görbe mentén csőgeometriát hoztam létre. Ehhez a már meglévő CurveMesh osztályt használtam fel. A `filament` és a `sleeve` az osztály egy-egy példánya, melyekhez kezdetben csak anyagot rendeltem a *3.8 kódrészletben látható módon*.

```
1  filament = makeCurveMesh(wom, {
2    materialName: FILAMENT_MATERIAL_NAME,
3  });
4  sleeve = makeCurveMesh(wom, {
5    materialName: SLEEVE_MATERIAL_NAME
6  });
```

3.8. kódrészlet. Filament és cső létrehozása

Ahhoz, hogy egy adott csőgeometriát bővíteni lehessen, szükség van radiális és axiális irányban is a szegmensek számára, illetve a cső keresztmetszetének sugarára, ezeket mind konstansként definiáltam. Meg kell adni továbbá a görbét, aminek mentén a geometriát kialakíthatja, ehhez a már létrehozott négy-négy segédpont koordinátáit kell felhasználni. A négy pontot felvettem egy tömbbe, majd ebből a tömbből generáltam görbét a THREE.js könyvtár `CubicBezierCurve3()` függvényével, ami négy pontra képes háromdimenziós Bézier-görbét illeszteni. [12] A filament (és ennek mintájára a cső) megrajzolásáért felelős függvény kódja a *3.9 kódrészleten* látható.

```
createFilament() {
  if(this.zInControl_created && this.zInDisp_created && this.spoolControl_created && this.spoolDisp_created){
    console.log("filament is being generated");
    this.spoolDisp = V3ToThree(this.spoolDisp_node.getPosition());
    this.spoolControl = V3ToThree(this.spoolControl_node.getPosition());
    this.zInControl = V3ToThree(this.zInControl_node.getPosition());
    this.zInDisp = V3ToThree(this.zInDisp_node.getPosition());
    this.filHandleArr = [
      this.spoolDisp, //, this.baseScale).add(this.basePos),
      this.spoolControl,//new THREE.Vector3().multiplyVectors(this.spoolControl, this.baseScale).add(this.basePos),
      this.zInControl,//new THREE.Vector3().multiplyVectors(this.zInControl, this.baseScale).add(this.basePos),
      this.zInDisp,//new THREE.Vector3().multiplyVectors(this.zInDisp, this.baseScale).add(this.basePos),
    ];

    this.filamentBezier = new THREE.CubicBezierCurve3(...this.filHandleArr);
    console.log("spooldisp: "+this.spoolDisp.x+"; "+this.spoolDisp.y+"; "+this.spoolDisp.z)
    console.log("spooldisp_node: "+this.spoolDisp_node.getPosition().x+"; "+this.spoolDisp_node.getPosition().y+"; "+this.spoolDisp_node.getPosition().z)

    this.filament.appendSection(
      this.filamentBezier,
      {
        tubularSegments: FILAMENT_TUBULAR_SEGMENTS,
        radius: FILAMENT_RADIUS_SCALE * this.baseScale.x,
        radialSegments: FILAMENT_RADIAL_SEGMENTS
      },
      this.FILAMENT_COLOR
    );
    this.spoolDisp_created=false;
    this.spoolControl_created=false;
    this.zInControl_created=false;
    this.zInDisp_created=false;
  }
}
```

3.9. kódrészlet. Filament kirajzolása

Maga a függvény csak akkor fut le, ha mind a négy segédpontot jelentő node már létezik. Ezt azzal oldottam meg, hogy bool változókat vettem fel az összes ponthoz, majd minden egyes *node* létrejöttére "akasztottam" egy eseményfigyelőt, és amennyiben már létrejött az adott node, meghívja a `createFilament()` függvényt. Az utolsó node

létrejöttekor a meghívott függvény le tud futni, és legenerálódik a görbe mentén a csőgeometria.

3.3.4. Szülő-gyermek elemek kapcsolata

A URDF modellben leírt csuklókapcsolatok alapján meghatározható, melyik elem melyiknek lesz a leszármazottja. Ehhez végigiteráltam a *joint*-okon, egyenként megvizsgálva a parent, illetve a child elemet is, majd elmentettem a `parentNode` és a `childNode` változókba. A kiválasztott elemeket a MaxWhere `appendChild()` függvényével egymáshoz kapcsoltam.

A nyomtatást segítő *node*-ok az eredeti programhoz hasonlóan maradtak egymáshoz parentálva. Az asztal esetében a `table` *node* segítségével meghatározzuk, minek a gyereke legyen a `cubeTracker` *node*. Két lehetőség van: mivel az `X` tengely mindig a nyomtatófej, ezért a másik két tengely közül kerül ki az asztal funkcióját betöltő elem. Amennyiben az az `Y` tengely, úgy ahhoz kell parentálni a `cubeTracker`-t, ha a `Z` tengely, akkor a gépalaphoz. A vezérlőpanel minden esetben a gépalapon található, így azt is ahhoz kell kapcsolni. A kapcsolatok kialakítása után - mivel minden egyes elem visszavezethető a gépalap, mint ősszülőhöz - elegendő csak a gépalapot kirenderelni, így a leszármazottak is kirenderelődnek.

3.8. ábra. Két nyomtató betöltése MaxWhere-be

4. fejezet

Kezelőfelület a vezérléshez

Ahhoz, hogy a nyomtatót használni tudjuk, egy kezelőfelületre van szükség. Erre a MaxWhere-ben több lehetőség is van, különböző okostáblákon bármilyen webes tartalom megjeleníthető, adta magát tehát, hogy egy weblapon keresztül történjen a program vezérlése.

4.1. Billboard

MaxWhere-ben a Canvas és a Webtable is alkalmas különböző oldalak megjelenítésére. A Canvas elsősorban akkor hasznos, ha az overlay-en szeretnénk elhelyezni tartalmat, ennek egy különleges fajtája a Webview, melyen webes tartalom is betölthető. A térben elhelyezett okostáblákhoz elsősorban Webtable-t érdemes használni, melynek egy speciális fajtája a Billboard, aminek esetén nem lehet átírni a felületen az oldal címét, tehát URL megváltoztatásával nem lehet onnan elnavigálni, a szoftveren belüli alkalmazások kezeléséhez ezt szokás használni. Mivel itt egy állandó felületről van szó, ami helyett nem ajánlott más oldalt megnyitni, a Billboard tűnt ideális választásnak.

Az eredeti verzióban a Billboard külön osztályba ki volt szervezve, azonban az általánosított nyomtató esetén jobb megoldásnak tűnt, ha ez is a Printer osztályban kerül létrehozásra, mivel ennek pozícióját is az adott nyomtató URDF modellje adja meg. A Billboard méreteit és felbontását az *index.jsx* fájlban adjuk meg a nyomtató példányosításakor. Ezeken felül szükség van egy azonosító megadására, annak érdekében, hogy a megfelelő panel a megfelelő komponenst vezérelje, illetve a weblap

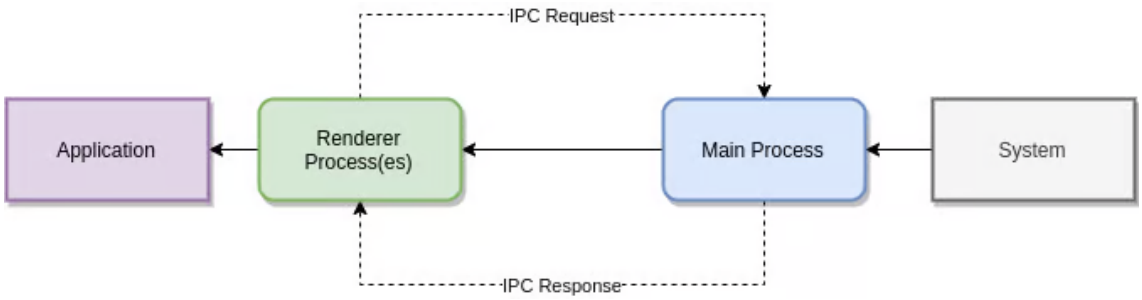
elérési útvonalára, mellyel így a tartalom is meghatározásra kerül (*4.1* kódrészlet). Mindemellett a `nodeIntegration` paramétert igazra kell állítani annak érdekében, hogy az IPC kommunikáció működhessen a Billboard és a szimuláció között, ennek módja a következőkben részletesen is kifejtésre kerül.

```
this.controlPanel = wom.create("billboard", {
  id: `component-tutorial-wom-manipulation-billboard${options.componentID}`,
  url: resolve(__dirname, "../billboardResources/index.html"),
  nodeIntegration: true, // IPC communication
  width: this.controlPanelWidth,
  height: this.controlPanelHeight,
  scaleFactor: 1,
  "resolution-width": this.resolutionWidth,
  "resolution-height": this.resolutionHeight,
  position: this.controlPanel_pos,
  orientation: eulerToQuat(this.controlPanel_rpy),
  physical: { raycast: true }, // orbitable
});
```

4.1. kódrészlet. Billboard létrehozása

4.2. IPC kommunikáció

Az IPC kommunikáció (Inter-Process Communication) az ElectronJS framework része, ennek segítségével asztali alkalmazások és weboldalak közti adatátvitel valósítható meg egyszerűen. Az ipcMain() függvény a Main Process-től küld adatot a Renderer Process-nek, míg az ipcRenderer() pont fordítva. Előbbi a weblap készítésénél, utóbbi pedig a főprogramban kerül alkalmazásra, hiszen a nyomtató panelről vezérelhető, ugyanakkor a nyomtató programjának fogadnia kell a szükséges utasításokat. [22]



4.1. ábra. IPC kommunikáció folyamata [15]

A Printer osztályban az ipcMain.on() függvénnyel már a kommunikációs csatornára való feliratkozás az eredeti programban is megtörtént, azonban ennek módja kiegészítésre szorult. Ahhoz, hogy a megfelelő panelről érkező adatokat vegye figyelembe a program, először meg kell a *4.2 kódrészlet* alapján vizsgálni, hogy az adott komponens példány vezérlőpaneljének azonosítója megegyezik-e a küldő fél azonosítójával, amennyiben igen, úgy a releváns információk érkeznek, és ezek kerülnek lekezelésre, és nem akad össze a különböző komponensek működése, amennyiben több is van egyszerre a térben.

```
1 // IPC kommunikáció a weblappal
2 ipcMain.on("compdev-tutorial-wom-manipulation" , (event, obj) => {
3   if (this.controlPanel.webview.browserWindow.webContents == event.sender) {
4     ...
5   }
6 });
```

4.2. kódrészlet. ipcMaimn.on() föggvény

Az IPC-n érkező objektum `cmd` attribútuma az, amely megadja, hogy milyen funkcióhoz érkezik a parancs. A `start`, `stop`, `reset` és `browse` funkciók már eleve működtek, ezek esetén meghívódik a `Printer` osztály megfelelő metódusa.
Szükség volt azonban új esetekre is. A `setTimescale` üzenet esetén az objektum `value` attribútuma adja meg a nyomtatás sebességét szabályzó arányt.
Színválasztásnál attól függően, hogy melyik színkomponens értékében történt változtatás, `red`, `green` vagy `blue` üzenet érkezik, az objektum ahhoz a színhez tartozó értéke kerül kiolvasásra, majd a filament színe frissül.
A mechanizmus fastruktúrájának lekérdezésekor a `tree` üzenet érkezik, ekkor válaszként négy különböző csatornára a négy fő elem azonosítóját küldi vissza a program, melyek aztán a megfelelő elrendezésben a képernyőn megjelennek. Az IPC üzenetek feldolgozását a *4.3 kódrészlet* mutatja.

```
1 switch (obj.cmd) {
2   case "start":
3     this.start();
4     break;
5   ...
6   case "setTimescale":
7     this.timeScale = obj.value;
```

```
8       break;
9     case "red":
10      this.red = parseFloat(obj.r);
11      this.FILAMENT_COLOR={r: this.red, g: this.green, b:this.blue};
12      break;
13    case "green":
14      this.green = parseFloat(obj.g);
15      this.FILAMENT_COLOR={r: this.red, g: this.green, b:this.blue};
16      break;
17    case "blue":
18      this.blue = parseFloat(obj.b);
19      this.FILAMENT_COLOR={r: this.red, g: this.green, b:this.blue};
20      break;
21    case "tree":
22      event.reply('ipc-tree1', this.tree0);
23      event.reply('ipc-tree2', this.tree1);
24      event.reply('ipc-tree3', this.tree2);
25      event.reply('ipc-tree4', this.tree3);
26      break;
27  }
```

4.3. kódrészlet. Az IPC üzenetek feldolgozása

4.3. Vezérlés

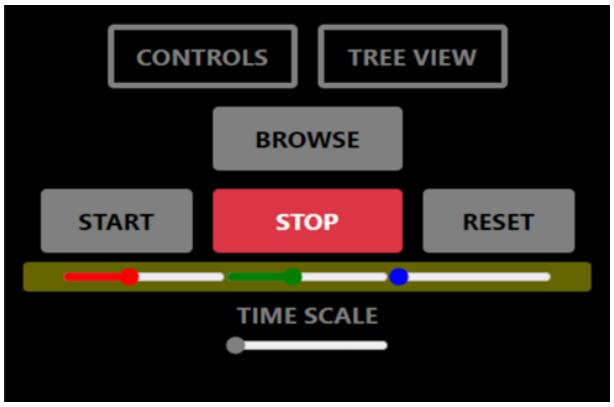
A nyomtató vezérlése eredetileg is az *index.html* segítségével történt a négy alap gombbal, ezt a fájlt egészítettem ki az új funkciókkal, illetve dizájnoltam újra saját ízlésem szerint a *4.2 ábrán* látható módon, ebben segítségemre volt a Bootstrap [23] könyvtár, mely egy egyszerűen használható előre összeállított frontendkészlet.

Először a két másik oldalra mutató gombot helyeztem el egy sorban középre igazítva, ezeknek `href` attribútumában megadtam a *control.html*, illetve a *tree.html* relatív elérési útvonalat. Ezek alá tettem a négy alap gombot, melyekkel a vezérlés történik, ezek mindegyikének `onclick` eseménye meghívja a megfelelő függvényt, amely IPC-n kiküldi az adott parancsot a főprogramnak, ahogy ezt a *4.4 kódrészlet* szemlélteti. A teljes *index.html* fájl megtalálható az *M.6. mellékletben*.

```
1   const { ipcRenderer } = require("electron");
2   const sendIpcMsg = (msg) => {
3     ipcRenderer.send("compdev-tutorial-wom-manipulation", msg);};
4   const reset = () => {
```

```
5     sendIpcMsg({ cmd: "reset" });};
6   const start = () => {
7     sendIpcMsg({ cmd: "start" });};
8   const stop = () => {
9     sendIpcMsg({ cmd: "stop" });};
10  const browse = () => {
11    sendIpcMsg({ cmd: "browse" });};
```

4.4. kódrészlet. Vezérlő függvények



4.2. ábra. index.html előnézete

A színválasztást az RGB kód három komponensének egyenkénti állíthatóságával oldottam meg, a három csúszka különböző színekkel a három szín arányát reprezentálja, mindegyik 0 és 255 közti értéket vehet fel.

Amennyiben valamelyik csúszka új értéket vesz fel, meghívódik a *4.5 kódrészletben* látható függvény, melynek segítségével IPC-n elküldi a főprogramnak az új értékeket, így real time változtatható akár nyomtatáson belül is a filament színe, emellett a csúszkák körül lévő terület is ezt a színt veszi fel, így az ellenőrizhető átállítás közben.

```
1   function changeRGB() {
2       var col, r, g, b;
3       r = document.getElementById("slideRed").value;
4       g = document.getElementById("slideGreen").value;
5       b = document.getElementById("slideBlue").value;
6       document.getElementById('rgb').style.backgroundColor = "rgb("+r*255+","+g*255+","+b*255+")";
7       sendIpcMsg({ cmd: "red", r });
8       sendIpcMsg({ cmd: "green", g });
9       sendIpcMsg({ cmd: "blue", b });
10    }
```

4.5. kódrészlet. changeRGB() függvény

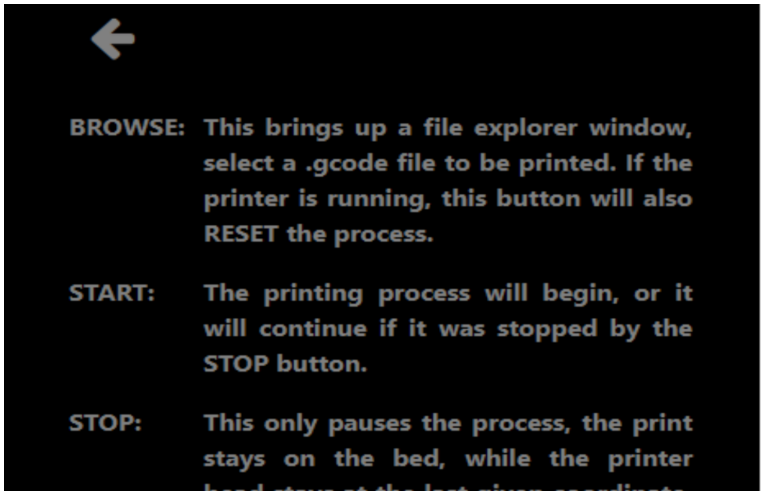
A nyomtatás sebességét egy állítható csúszkával 1 és 20 között lehet skálázni. A valós sebességtől indul a skála, és akár 20-szorosára felgyorsítható. Amint változás történik a csúszka állapotában, meghívódik a `setTimescale()` függvény, mely az új értéket elküldi a szimulációnak a *4.6 kódrészlet* szerint.

```
1     const setTimescale = (event) => {
2       const value = parseFloat(event.target.value);
3       sendIpcMsg({ cmd: "setTimescale", value });
4     };
```

4.6. kódrészlet. setTimescale() függvény

4.4. Kezelési útmutató

Annak érdekében, hogy senkinek ne okozzon gondot a nyomtató irányítása, a *controls.html* fájlban leírást adtam meg minden egyes funkcióhoz (*M.7 melléklet*). Ezek tanulmányozása után a vissza gomb újra a vezérlőpanelre navigál.



4.3. ábra. Kezelési útmutató

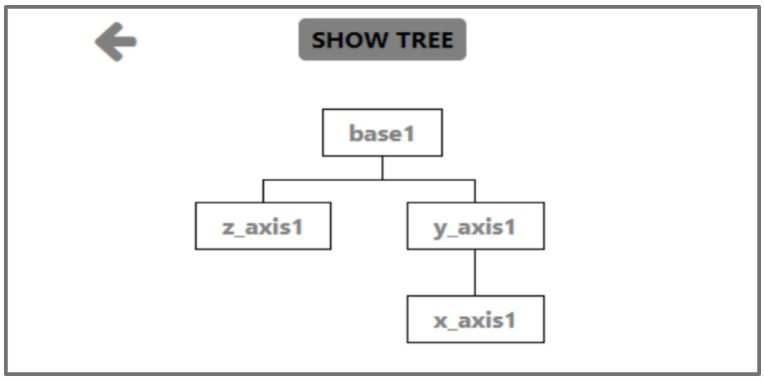
4.5. Kinematikai lánc

A *tree.html* az adott nyomtató kinematikai láncának fastruktúrában történő megjelenítéséért felel. Ehhez először egy gombot helyeztem el, ami a *4.7 kódrészletben* látható `tree()` függvényt hívja meg. A függvény IPC-n kiküldi a megfelelő üzenetet, a főprogram pedig visszaküldi a megjelenítendő azonosítókat a megfelelő sorrendben.

```
1     const tree = () => {
2       sendIpcMsg({ cmd: "tree" });
3       const treestyle=document.querySelector('#tree-id');
4       treestyle.setAttribute('style', 'content-visibility: visible !important;');
5     };
```

4.7. kódrészlet. tree() függvény

A fastruktúra megjelenítéséhez a *treeflex* könyvtárat [35] használtam, előre felvettem benne négy pontot, hiszen minden Cartesian nyomtató vázát az alapja és a három tengelye alkotja, itt igazából csak a közöttük lévő kapcsolat a kérdés, amit a Printer osztály konstruktorában határoztam meg a URDF modell alapján. Minden esetben a gépalap a legfelső szint, a *joint*-ok alapján pedig amelyik tengelynek nem az alap a szülője, az lesz a legalsó szinten, a szülője a csuklóból adódik, a negyedik elem pedig az alap azon leszármazottja, amelyhez több elem nem kapcsolódik. Ez a gráf statikusan megjeleníti a kapcsolaotokat akkor is, ha üresek az egyes pontok, így amíg nincs adat, láthatatlanná tettem, a `tree()` függvény változtatja újra láthatóvá. A *tree.html* kódja megtalálható az *M.8. mellékletben*.A lekérdezett kinematikai lánc az egyik nyomtatóra a *4.4 ábrán* látható.



4.4. ábra. Kinematikai lánc

5. fejezet

Nyomtató mozgatása

5.1. Tengelyek mozgatása

Az általános nyomtató működtetéséhez a mozgásokat újra kellett konstruálnom, mivel az eredeti csak a Crealty Ender 3 nyomtató felépítéséhez lett kitalálva.

A G-kód feldolgozását a `GcodeReader` oszttállyal valósította meg kollégám. [13] Ebben segítségére volt a `gcode-parser` könyvtár, melynek `parseStringSync()` függvényével a konstruktorban soronként szét lehet szedni a kódot, és szavanként elkülöníteni az utasításokat, melyeket egy JSON objektumban ad vissza. Minden egyes mondatból kinyerhető így az X, Y, és Z koordináta, illetve az előtolás és az extrúzió. Mivel a kódból csak azokra a sorokra van szükség mozgás szempontjából, amik G1 vagy G0 lineáris mozgást valósítanak meg, az ezekre szűrt eredmény el lett tárolva a `filteredArray` tömbben. Ennek felhasználásához írtam egy általános függvényt, mely JSON objektumokat vár a node nevével és a cél koordinátáival, így mozgatja a különböző tengelyeket.

A függvényben a mozgatáshoz a MaxWhere `Node.animate()` metódusát használtam (*5.1 kódrészlet*). Ennek attribútumában megadtam, hogy pozíciót (`position`) szeretnék animálni, de van lehetőség orientáció (`orientation`) és méret (`scale`) animálására is. A referencia (`reference`) paramétert absolute-ra állítottam, a célt (`to`) és az időtartamot (`duration`) a tömb elemeiből nyeri ki a metódus. A görbe típusát (`cubic`), mely mentén a mozgatás történik lineárisnak választottam (`linear`), de lenne lehetőség többek között körpálya vagy Bézier-görbe megadására is.

```
animateMulti(animations: Array<{
  node: Node,
  to: Vector3,
}>, cbComplete: () => void) {
  const promises = new Array<Promise<void>>();
  for (const anim of animations) {
    const promise = new Promise<void>(resolve => anim.node.animate('position', {
      reference: 'absolute',
      to: anim.to,
      duration: this.dur,
      loop: false,
      cubic: 'linear',
    }, resolve));
    promises.push(promise);
  }
  Promise.all(promises).then(cbComplete);
}
```

5.1. kódrészlet. animateMulti() függvény

Az előző függvényt az `animate()` metódusban használom fel, mely először a G-kódból kiolvassa az új koordinákat, majd megállapítja az extrúzió mértékéből, hogy van-e nyomtatás, ha nőtt az extrúzió, az azt jelenti, hogy a filament fogy, a nyomtatás zajlik. Az egyes mozgatatandó testeknek le kell kérni a jelenlegi pozícióját. Ezeket, illetve a gépalap és a tengelyek kezdeti pozícióját felhasználva a `headPos` vektorban meghatározható a nyomtatófej relatív pozíciója az asztalhoz képest, mivel ott megjelenik mindhárom tengely irányában történt elmozdulás, ennek számítását a *5.2 kódrészlet* mutatja.

```
let headPos = new THREE.Vector3(
  this.x_direction*(x_axisPos.x - this.basePos.x-this.x_start_pos.x*this.baseScale.x) / this.baseScale.x,
  -this.y_direction*(y_axisPos.z - this.basePos.z-this.y_start_pos.z*this.baseScale.y) / this.baseScale.y,
  this.z_direction*(z_axisPos.y - this.basePos.y-this.z_start_pos.y*this.baseScale.z) / this.baseScale.z
).multiplyScalar(10);
```

5.2. kódrészlet. Nyomtatófej relatív pozíciója

A `coordIn` vektor tartalmazza a következő parancsból az új koordinátákat. A `diff` változóban meghatároztam az ebben a lépésben végrehajtandó relatív mozgást, a `distance` mutatja az elmozdulás hosszát. A mozgás időtartamát erre a távolságra lehet számítani a *5.3 kódrészlet* alapján. Az előtolással le kell osztani, és mivel annak

mértékegysége [mm/min], 60-nal meg kell szorozni, hogy másodpercben megkapjuk az időt. Mivel az idő skálázható, annak értékével még le kell osztani a kapott időtartamot, így alakul ki, milyen hosszú lesz az adott lépésben az animáció.

```
let diff: THREE.Vector3 = coordIn.clone().sub(headPos);
let distance = diff.length();
this.dur = (distance / this.command.feedRate) * 60 / this.timeScale;
```

5.3. kódrészlet. Animáció időtartama

A parancsban érkező koordináták alapján a *5.4 kódrészlet* szerint meghatározhatóak az egyes tengelyek új pozíciói, ahova el kell őket mozgatni, itt a tengelyek kiindulási pozíciójához kell előjelhelyesen hozzáadni az új koordinátákat, ebben a URDF modellből kiolvasott mozgási irányvektorok segítenek.

```
const targetXAxis = new THREE.Vector3(this.x_start_pos.x+this.command.coords.x*this.axis_x[0]/10, 0, 0);
const targetYAxis = new THREE.Vector3(0, 0, this.y_start_pos.z-this.command.coords.y*this.axis_y[1]/ 10);
const targetZAxis = new THREE.Vector3(0, this.z_start_pos.y+this.command.coords.z*this.axis_z[2]/10, 0);
```

5.4. kódrészlet. Tengelyek új pozíciói

A `targetPrintTracker` vektor jelöli a *5.5 kódrészletben* látható módon `printCoordTracker` node új pozícióját a gépi koordinátarendszerben, ennek a pontnak a mozgását monitorozva történik majd a nyomtatás is.

```
const targetPrintTracker = new THREE.Vector3(
  this.x_start_pos.x-this.command.coords.x*this.axis_x[0]/10,
  this.z_start_pos.z-this.command.coords.z*this.axis_z[2]/10,
  this.y_start_pos.y-this.command.coords.y*this.axis_y[1]/10
);
```

5.5. kódrészlet. printCoordTracker új pozíciója

Miután minden célkoordináta meghatározásra került, az egyes tengelyek és a `printCoordTracker` mozgatása a korábban említett `animateMulti()` függvénnyel történik az *5.6 kódrészlet* szerint.

```
this.animateMulti([
  {
    node: this.x_axis,
    to: V3ToMw(targetXAxis),
  },
  {
    node: this.y_axis,
    to: V3ToMw(targetYAxis),
  },
  {
    node: this.z_axis,
    to: V3ToMw(targetZAxis),
  },
  {
    node: this.printCoordTracker,
    to: V3ToMw(targetPrintTracker),
  },
], cbComplete);
```

5.6. kódrészlet. Tengelyek mozgatása

Ennek mintájára működik az `animateToStart()` metódus is, amely akkor hívódik meg, ha a nyomtatót visszaállítjuk a kiindulási állapotára. Ekkor csupán mozgatás történik, már nincs szükség nyomtatásra, így az extrudálást és a nyomtatást is ki kell kapcsolni. A nyomtatás visszaállításakor minden tengely mentén 0 érték érkezik, a `targetPrintTracker` ezeket kapja meg, a tengelyeket pedig a kiindulási állapotra kell pozicionálni.

A `denimate()` metódussal az animált mozgások megszakításra kerülnek mind a négy node-ra a `deanimate()` függvény felhasználásával a *5.7 kódrészlet* szerint.

```
denimate() {
  this.x_axis.deanimate("position");
  this.y_axis.deanimate("position");
  this.z_axis.deanimate("position");
  this.printCoordTracker.deanimate("position");
}
```

5.7. kódrészlet. denimate() metódus

5.2. Filament és cső mozgatása

A filament és a cső helyzete a nyomtató mozgása során folyamatosan változik, azonban minden időpillanatban nem lehetne újrarajzolni őket, mert ez rendkívül erőforrásigényes lenne. Erre adott megoldást kollégám azzal, hogy csupán a tengelyek bizonyos mértékű elmozdulása esetén kelljen újragenerálni a csőgeometriákat. Kellően kicsi elmozdulás beállítása esetén a szimuláció a szem számára nem veszít folytonosságából, számítási igénye viszont sokkal kevesebb lesz. Külön kell vizsgálni minden tengelyt, hiszen a cső az X, a filament a Z tengely menti mozgás esetén változik meg, így a megfelelő ütemben újragenerálódnak a csőgeometriák az általam írt metódusok futtatásának köszönhetően a *5.8 kódrészletnek* megfelelően.

```
1  // lelógó filament újrarajzolása
2  if (Math.abs(this.zMoved) > this.eps * this.baseScale.x) {
3    this.redrawFilament();
4  }
5  // cső újrarajzolása
6  if (Math.abs(this.xMoved) > this.eps * this.baseScale.x) {
7    this.xMoved = 0;
8    this.redrawSleeve();
9  }
```

5.8. kódrészlet. Csőgeometriák újrarajzolása

A `redrawFilament()` és a `redrawSleeve()` metódusok a *3.9 kódrészletben* bemutatott `createFilament()` metódushoz hasonlóan görbét illeszt a *node*-ok pillanatnyi pozícióiból összeállított tömb pontjaira, és legenerálja a csőgeometriát. Ebben az esetben nem kell megvizsgálni, hogy létrejöttek-e a *node*-ok, mivel már a mozgás zajlik. Az `appendSection()` helyett a `changeSection()` metódust alkalmaztam, mivel nem az előző geometriához hozzáillesztve, hanem ahelyett kell legenerálni.

5.3. Ventilátor mozgatása

További újításként hoztam be a hűtés szemléltetéséhez a ventilátor mozgatását, ehhez a `Fan` osztályt (*M.9. melléklet*) hoztam létre, mivel ezt összetettebb folyamatként kezeltem, mint egy kétállapotú gép, mely vagy forog, vagy nem. Ehelyett három állapotot határoztam meg a `FanState` típusban: álló, szabadon pörgő, és táppal pörgetett. Utóbbi kettőnél a sebesség eltárolásának is van értelme, hiszen az jellemzi az adott állapotot, a sebesség az idő függvényében változhat.

A nyomtatófej melegedését is szerettem volna leszimulálni egy egyszerű modell alapján. Ehhez a `TemperatureSimulator` osztályt hoztam létre. Ez paraméterként az adott nyomtatót, ventilátort, illetve az egységnyi idő alatt bekövetkező hőmérsékletnövekedést és csökkenést kapja meg példányosításkor.

Az osztály konstruktorában meghatároztam a szükséges adattagokat, ezek a minimum és maximum hőmérsléklet, a forgás tengelye, a forgatni kívánt test, valamint a nyomtató példány, amiben szerepel. A ventilátor alaphelyzetben áll, és nincs hozzá hőmérsékletszenzor csatlakoztatva. A szenzort a `TemperatureSimulator` osztály valósítja meg, ennek működéséről szó esik majd a következő alfejezetben.

```
1    constructor(private minTemp: number, private maxTemp: number, private axis: THREE.Vector3, private node
     : Node, private printer: IPrinter) {
2        this.state = {
3            status: 'idle',
4        };
5        this.sensor = null;
6    }
```

5.9. kódrészlet. Fan osztály konstruktora

Az `attachSensor()` metódus egy `TemperatureSimulator` osztály példányt hoz létre, míg a `detachSensor()` metódus meghívásával ki lehet törölni azt. Létrehoztam továbbá a `speed()` és `maxSpeed()` *getter*-eket a fordulatszámok lekérdezéséhez.

A folytonos megjelenítés érdekében a forgást 90°-onként animáltam. Ezt a `startAnimation()` metódus `cbBeginAnimCycle()` függvényének folytonos újrahívásával valósítottam meg. Amennyiben a ventilátor áll, úgy az animáció megáll, ellenkező esetben viszont új értékeket kell kiszámolni a forgatáshoz. Az aktuális fordulatszámból

számítható az az időtartam, ami egy negyedfordulat megtételéhez szükséges. A forgatást leíró tengely-szög reprezentációt a `rotation` objektumban tároltam el, ide az a tengely kerül behelyetteítésre, amelyet az osztály példányosításakor megadtunk. Az `animate()` függvénybe a meghatározott értékeket beillesztve a *5.10 kódrészlet* alapján megtörténik az animálás, ennek leállításáért pedig a `stopAnimation()` metódus felel, melynek semmi más dolga nincs, minthogy meghívja a `deanimate()` függvényt.

```
startAnimation() {
    const cbBeginAnimCycle = () => {
        if (this.state.status === 'idle') {
            // Fan is idle, stop animation
            return;
        }

        const speed = Math.max(this.state.speed, 1);
        const qSpeed = 4 * speed;
        let duration = 60 / qSpeed;

        const rotation = {
            axis: { x: this.axis.x, y: this.axis.y, z: this.axis.z },
            angle: 90,
        };

        this.node.animate("orientation", {
            reference: "relative",
            to: rotation,
            duration: duration,
            loop: false,
            cubic: "linear",
        }, cbBeginAnimCycle);
    };
    cbBeginAnimCycle();
}
```

5.10. kódrészlet. Animálás

A *5.10 kódrészletben* fellelhető `simulate()` metóduson belül dől el, hogy az animálás milyen sebességgel történjen, illetve hogy történjen-e egyáltalán. Itt a szenzor által becsült aktuális hőmérséklet is felhasználásra kerül.

Ha nyomtatás van folyamatban, de a vetnilátor mégis áll, vagy csupán szabadon pörög, akkor mindenképpen pörgetni kell maximális sebességgel, hogy beinduljon a hűtés folyamata *5.11 kódrészlet*.

```
simulate(deltaTime: number) {
    const currentTemp = (this.sensor !== null) ? this.sensor.temperature : -273.15;

    if (this.printer.isPrinting) {
        if (this.state.status === 'idle' || this.state.status === 'rotating') {
            // Fan is not powered; turn it on
            this.state = {
                status: 'powered',
                speed: FAN_MAX_SPEED_RPM,
            };
            this.startAnimation();
        }
    }
    ...
}
```

5.11. kódrészlet. simulate() metódus nyomtatás alatt

Abban az esetben is szükség lehet hűtésre, ha éppen nincs extrudálás. Be kell indítani a ventilátort, ha az nem volt táplálva, és eközben a nyomtatófej hőmérséklete elérte a felső limitet. Ekkor a minimális sebességgel kezd pörögni, aztán egyre gyorsul az előre megadott mértékben egységnyi időközönként mindaddig, amíg el nem érte a maximális fordulatszámot *5.12 kódrészlet*.

```
    if (currentTemp > this.maxTemp) {
        if (this.state.status === 'idle' || this.state.status === 'rotating') {
            // Fan is not powered; turn it on
            this.state = {
                status: 'powered',
                speed: FAN_MIN_SPEED_RPM,
            };
            this.startAnimation();
        } else if (this.state.status === 'powered') {
            // Fan is already powered; keep powered and increase fan speed until it reaches max speed
            this.state = {
                status: 'powered',
                speed: Math.min(this.state.speed + FAN_ACCELERATION * deltaTime, FAN_MAX_SPEED_RPM),
            };
        }
    }
```

5.12. kódrészlet. simulate() metódus hűtés alatt

Amennyiben a nyomtatófej hőmérséklete a minimum hőmérséklet alá csökkent (*5.13 kódrészlet*), és forgatva volt a ventilátor, azt hagyni kell csak szabadon pörögni,

ezután pedig egységnyi időközönként egyenletesen csökkenteni a sebességét, mígnem az a minimum sebesség alá csökken, ekkor már teljesen megáll a hűtés.

```
    else if (currentTemp < this.minTemp) {
        if (this.state.status === 'powered') {
            // Turn off the power to the fan
            this.state = {
                status: 'rotating',
                speed: this.state.speed,
            };
        } else if (this.state.status === 'rotating') {
            // Fan is not powered but still has momentum; keep decelerating it
            this.state = {
                status: 'rotating',
                speed: Math.max(FAN_MIN_SPEED_RPM, this.state.speed - FAN_DECELERATION * deltaTime),
            };

            if (this.state.speed <= FAN_MIN_SPEED_RPM) {
                // The fan has come to a stop
                this.state = {
                    status: 'idle',
                };
            }
        }
    }
```

5.13. kódrészlet. simulate() metódus leállítás közben

5.4. Hőmérséklet érzékelése

A nyomtatófej melegedését is szerettem volna leszimulálni egy egyszerű modell alapján. Ehhez a `TemperatureSimulator` osztályt (*M.10. melléklet*) hoztam létre. Ez paraméterként az adott nyomtatót, ventilátort, illetve az egységnyi idő alatt bekövetkező hőmérsékletnövekedést és csökkenést kapja meg példányosításkor.

A modell lényege, hogy amikor nyomtatás zajlik, az a `generatedHeat` változóban megadott hőmérsékletnövekedést okoz, mely 40 millisecundumonként következik be. Ezen időközönként a maximális sebességgel pörgetett ventilátor ugyanekkora mértékű hőmérsékletcsökkenést eredményez. Amennyiben a nyomtatás áll, a nyomtatófej hőmérséklete `lostHeat`-tel csökken a `simulate` metódus szerint (*5.14 kódrészlet*).

```
simulate(deltaTime: number) {
    const DELTA_TIME = 40 / 1000;
    if (this.printer.isPrinting) {
        this.headTemperature += this.generatedHeat * DELTA_TIME;
    } else {
        this.headTemperature -= this.lostHeat * DELTA_TIME;
    }
    if (this.fan.state.status === 'powered') {
        this.headTemperature -= this.generatedHeat * DELTA_TIME * this.fan.speed / this.fan.maxSpeed;
    }
    this.headTemperature = Math.max(ROOM_TEMPERATURE, this.headTemperature);
    this.fan.simulate(deltaTime);
}
```

5.14. kódrészlet. Nyomtatófej hőmérsékletének változása

5.5. Nyomtatvány megjelenítése

A nyomtatvány kirajzolásához a nyomtatófejen található fúvóka pozícióit használta fel kollégám, és ezen az úton indultam el én is. Ha a nyomtató elmozdul, egy vektorba felveszem a fúvóka végének koordinátáit a *5.15 kódrészlet* alapján. Az x koordináta minden esetben a nyomtatófej pozíciójának x komponense. A másik két esetben már számít, hogy melyik tengely az, amelyik asztalként is funkcionál, hiszen ezek elmozdulását adott esetben kompenzálni kell annak érdekében, hogy torzítás nélkül kaphassuk meg a nyomtatott formát.

```
  const headPosRelY = new THREE.Vector3(
    this.x_axisPos.x + (this.headOffset.x * this.baseScale.x),
    this.table_id ==this.y_axis.id ? this.z_axisPos.y + (this.headOffset.y * this.baseScale.y) : this.
     z_axisPos.y + (this.headOffset.y * this.baseScale.y)-(this.z_axisPos.y-this.z_start_pos.y*this.
     baseScale.y-this.basePos.y)*2,
    this.table_id ==this.y_axis.id ? this.x_axisPos.z + (this.headOffset.z * this.baseScale.z)- this.
     y_axisPos.z : this.x_axisPos.z + (this.headOffset.z * this.baseScale.z)
  );
```

5.15. kódrészlet. Tubegeometry pontjai

A mozgás során célszerű vizsgálni az elmozdulás mértékét, és csak abban az esetben kell eltárolni az előbb említett koordinátákat, amennyiben már értelmezhető pozíció-különbség van az előző ponthoz képest, ezzel meggátolható a túlzott mennyiségű adat feldolgozása. Ez a térbeli Pitagorasz-tétel segítségével könnyen meghatározható.

Ha a nyomtató elmozdult, és a nyomtatás is folyamatban volt, úgy az adott pont bekerül a test geometriáját leíró tömbbe, illetve ebbe a pontba egy kiskocka helyezendő, ezzel a folytonos nyomtatást reprezentálva a *5.16 kódrészlet* alapján. Az eredeti verzióban ez sima *mesh*-ként volt generálva, azonban mivel ugyanabból a kockából rengetek kerül lehelyezésre a nyomtatás során, és egyedül a helyzetük különbözik, a MaxWhere *instancedmesh* formátumának használatával optimalizálható a folyamat. Ezzel jóval kevesebb erőforrást igényel egy azon test renderelése újra és újra.

```
1   this.tubePointsArray.push(
2     headPosRelY
3   );
4   let buildSegment = wom.create("instancedmesh", {
5     id: `buildSegment${this.componentID}${Math.random() * 10000}${this.tubePointsArray.length}`,
6     url: "myCube.mesh",
7     position: {
8       x: (this.x_axisPos.x - this.basePos.x) / this.baseScale.x + this.headOffset.x,
9       y: (this.x_axisPos.y - this.basePos.y) / this.baseScale.y + this.headOffset.y,
10      z: (this.z_direction*this.basePos.z - this.z_direction*this.y_axisPos.z) / this.baseScale.y +
    this.headOffset.z
11    },
12    orientation: { x: 0, y: 0, z: 0, w: 1 },
13    scale: {
14      x: this.cubeWidth, y: this.cubeWidth, z: this.cubeWidth,
15    },
16    autophysical: false,
17  });
18  wom.render(buildSegment);
19  this.printBaseCube.appendChild(buildSegment);
```

5.16. kódrészlet. Pontok felvétele és nyomtatása

Mikor a tömbbe felvett pontok száma eléri az előre meghatározott limitet, a tömb pontjaira a `CatmullRomCurve3()` függvénnyel görbét illesztve, majd ezen görbe mentén egy csőgeometria generálásával az eddig lehelyezett kis kockák helyettesíthetők, így azok törölhetőek, ennek pedig legegyszerűbb módja, ha a `printBaseCube` *node*-ot töröljük, hiszen ennek összes gyermeke is megsemmisül. A törlés után a `printBaseCube`

node-ot újra létre kell hozni, hogy a következő fázisban is ehhez lehessen parentálni az egységkockákat. A pontokat tároló tömb tartalmát is törölni kell, majd az első elemeként a legutolsó felvett pontot belepusholni, ahogy azt az *5.17 kódrészlet* mutatja.

```
if (this.tubePointsArray.length === this.maxBuild) {
  const curve = new THREE.CatmullRomCurve3(this.tubePointsArray);
  this.buildTube.appendSection(curve, curveOpt, color);
  this.printBaseCube.clear();
  this.printBaseCube = wom.create("node", {
    id: `printBaseCube${this.componentID}`,
    position: {
      x: 0,
      y: 0,
      z: 0,
    },
  });
  this.cubeTracker.appendChild(this.printBaseCube);
  this.tubePointsArray.length = 0;
  this.tubePointsArray.push(headPosRelY.clone());
}
```

5.17. kódrészlet. Nyomtatvány kirajzolása

Amennyiben még nem érte el a limitet a felvett pontok száma, viszont a nyomtatás megszakad, vagyis az extrudálás abbamarad, akkor szintén ugyanezt a folyamatot kell végigjátszani, csak kevesebb pontra történik a görbeillesztés.

Magát a nyomtatványt is mozgatni kell, hogy lekövesse az asztal helyváltoztatását, ehhez attól függően, hogy melyik tengely tölti be az asztal funkcióját, azon tengely mentén a nyomtatvány elmozdítása is szükséges (*5.18 kódrészlet*).

```
if(this.table_id == this.y_axis.id){
  this.buildTube.setPosition({ x: 0, y: 0, z: this.y_axisPos.z-this.y_start_pos.z})}
if(this.table_id == this.z_axis.id){
  this.buildTube.setPosition({ x: 0, y: this.z_axisPos.y-this.z_start_pos.y*this.baseScale.y-this.basePos
   .y, z: 0});}
```

5.18. kódrészlet. Nyomtatvány elmozdítása

6. fejezet

Tervezés és nyomtatás

Az előzőekben elkészített általánosított nyomtató szimulációjának működését természetesen tesztelni kell. Két különböző felépítésű gépet helyeztem el a virtuális tanteremben, így célszerű mindkettőt kipróbálni az eredményesség ellenőrzése érdekében. A következő fejezetben különböző tárgyak tervezése és nyomtatása kerül bemutatásra mind a valós, mind a digitális térben.

6.1. Telefon állvány tervezése

Gyakran szembesülhet az ember olyan esetekkel, mikor fel kell töltenie a telefonját, de a konnektor az adott helyiségben olyan helyre van téve, ahol nincs a közelben egy asztal vagy polc (pl.: öltöző, fürdőszoba), amire biztonságosan le lehetne azt helyezni. Erre a problémára kerestem a megoldást egy olyan állvány kialakításával, melyet a telefon töltőjének adapterére lehet felszerelni.

Az interneten utánanézve sok ilyen eszközt találtam, de azokat legtöbb esetben túl nagynak, vagy hiányosnak ítéltem, némelyiket pedig a falra kellett volna szerelni, ezt pedig el akartam kerülni, hiszen egy olyan mobilis eszköz volt a cél, amit bárhol lehet használni, ahol éppen szükség van rá. Szempont volt továbbá, hogy a telefont ne vízszintesen, hanem betámasztva lehessen elhelyezni, ezzel lehetővé téve a videónézést például hajszárítás közben.

Mivel a családban mindenkinek Samsung töltője van, célszerű volt ezzel kompatibilis állványt terveznem. Először is letöltöttem a standard Samsung töltő adapterének

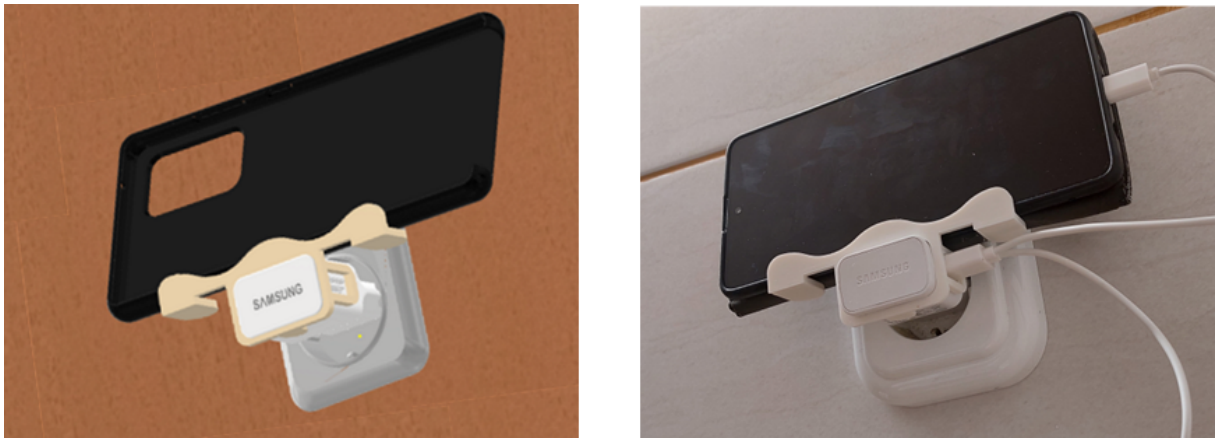
modelljét, és felvettem a méreteit, az állvány méreteit pedig ennek megfelelően alakítottam ki.

A támasztó részt hullámosra terveztem a mutatósabb dizájn érdekében. A megfelelő görbületi sugarak kiszámítása elengedhetetlen az egyenletes hullám kialakításában. Ezt úgy valósítottam meg, hogy az összeérő köríveknek a közös pontban ugyanaz legyen az érintője, ehhez ki kellett számolni a megfelelő görbületi sugarat az egyes köríveknek.

Az állványt a minél kevesebb anyaghasználat céljából úgy alakítottam ki, hogy a tartó funkciót maga az adapter lássa el az azzal egy szintben elhelyezett két oldalsó füllel együtt.

Annak érdekében, hogy az állvány semmiképp ne csúszhasson le az adapterről, egy fület terveztem az USB aljzat köré, melyet így a csatlakoztatott kábel helyhez köt. Azt pedig, hogy az állvány előredőljön a telefon terhétől, egy alulra elhelyezett kiálló felület akadályozza meg.

A Cad modellt Inventorban készítettem, amit aztán *STL* formátumba exportáltam a további felhasználás érdekében.



6.1. ábra. Tervezett telefon állvány

6.2. Logók készítése

A MOGI (Mechatronika, Optika és Gépészeti Informatika) Tanszék logóját felesleges lett volna magam megtervezni, erre találtam sokkal egyszerúbb megoldást. A Blender-ben lehetőség van 2D képből 3D modellt kreálni. Ehhez nem kell egyebet tenni, mint az adott

képet *SVG* formátumba exportálni, ezt az Inkscape szoftverrel oldottam meg. Ezt követően a Blender-ben a kép kívánt méreteinek beállítása után a megfelelő vastagságú testté lehet extrudálni. A kész modellt *STL* formátumba érdemes kiexportálni. Hasonlóképpen jártam el a MaxWhere logó esetében is.

6.2. ábra. PNG-ből SVG, majd STL

6.3. Szeletelés

A modellek szeletelését az Ultimaker Cura szoftverével végeztem. A kitöltés mértékének és a rétegvastagságnak beállítása után a szeletelés eredményeképp megkapjuk az idő- és anyagigényét a nyomtatásnak, valamint a 3D felületen a nyomtatás végtermékét is. Az így generált G-kódokat betöltve a programba már indulhat is a nyomtatás.

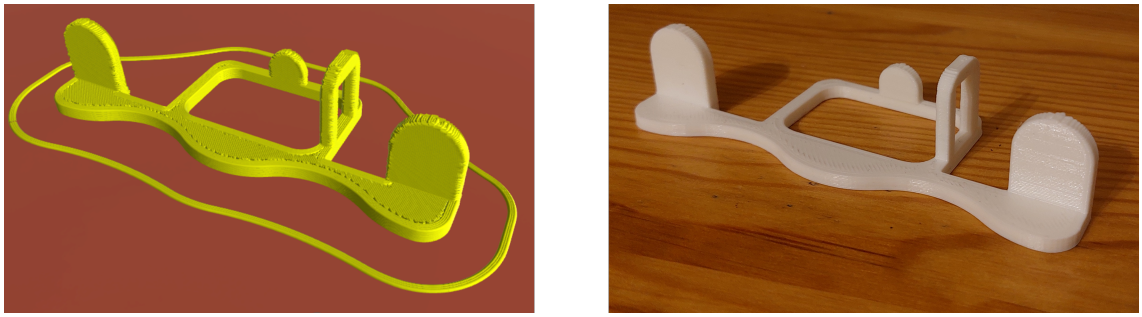
6.4. Nyomtatás

A szeletelés során létrehozott G-kódok alapján a szimulációban több nyomtatót felhasználva, illetve a Mechatronika Szakosztály Ender 3-as nyomtatójával is elkészítettem az egyes modelleket, melyek eredményei a *6.3*, *6.4* és *6.5 ábrákon* láthatóak.

Megfigyelhető, hogy ahogyan a valóságban, úgy a szimulációban is rétegről rétegre épülnek fel a testek, az így kapott termékek egy az egyben megegyeznek az általam kinyomtatottakkal.

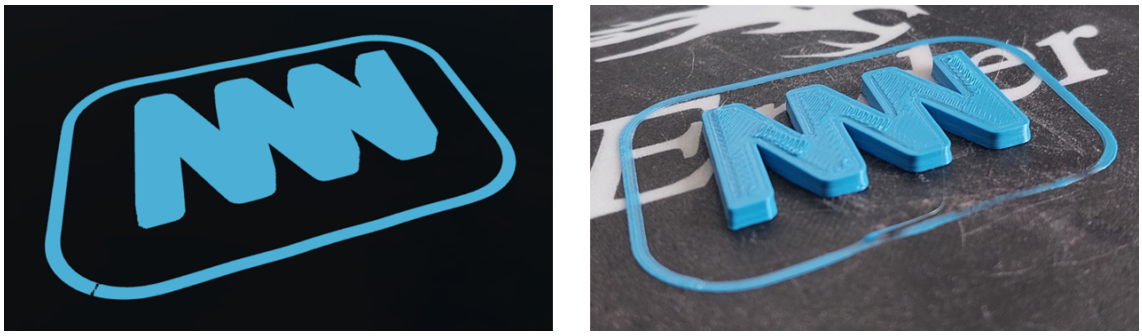
Sajnos a dinamikusan változtatható színű modelleknek nincsen dinamikus árnyéka a térben, így ezek megjelenése kevésvbé látványos (*6.4 ábra*), előre felanyagozott filament betöltése esetén azonban teljesen élethűek a fények is (*6.3* és *6.5 ábra*).

6.4.1. Telefon állvány



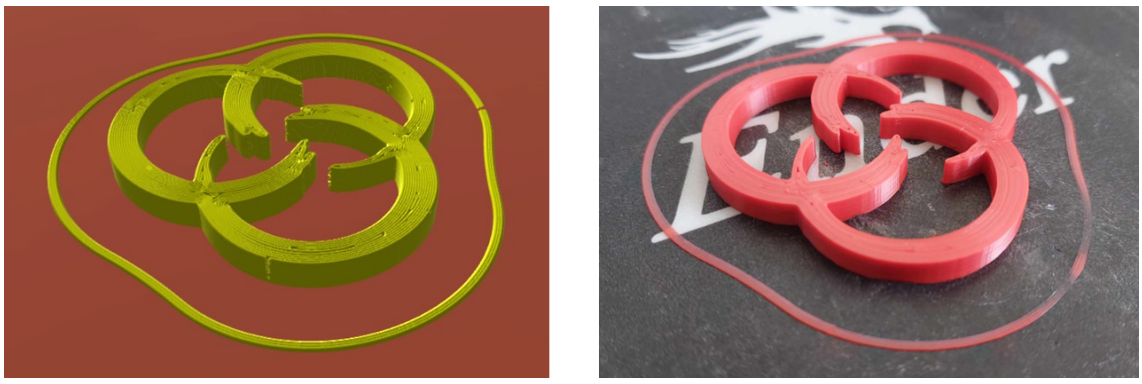
6.3. ábra. Telefon állvány kinyomtatva virtuálisan (Craftbot) és valósan

6.4.2. MaxWhere logó



6.4. ábra. MaxWhere logó kinyomtatva virtuálisan (Ender) és valósan

6.4.3. MOGI logó



6.5. ábra. MOGI logó kinyomtatva virtuálisan (Craftbot) és valósan

7. fejezet

Összefoglalás

7.1. Összegzés

A feladat megoldásához először rövid irodalomkutatásra volt szükség különböző témakörökben. A 3D nyomtatók fajtáit, illletve a nyomtatási technológiákat vizsgálva jutottam arra a döntésre, hogy az extrúzióval működő Cartesian nyomtatókra általánosítom a modellemet. Piackutatást is végeztem az oktatási célú virtuális demonstrációs eszközök között, hogy felmérjem mire is lehet igény, miben lehetne újat mutatni. A nyomtató felépítéséhez a URDF robotleírásnak, a nyomtatás értelmezéséhez pedig a G-kódnak néztem utána.

Ezt követően a már létező Ender nyomtató virtuális szimulációját bővítettem, azt általánosítottam, hogy URDF formátum alapján bármely Cartesian nyomtatót fel tudja magának építeni, illetve mozgatni azt az adott tengelyek mentén. Megújítottam a kezelőfelületet, illetve optimalizáltam a nyomtatás erőforrásigényét, és általánosítottam a nyomtatvány mozgatását is. A hűtés szemléltetése érdekében leszimuláltam a nyomtatófej hőmérsékletváltozását, a ventilátor mozgását pedig ennek megfelelően programoztam be. Egy telefonállványt terveztem, melyet több más modellel együtt kinyomtattam a virtuális nyomtatókkal és egy valós géppel is, hogy a szimuláció pontosságát bizonyítsam. A projekt angol nyelvű bemutatására statikus weboldalt hoztam létre MkDocs formátumban.

7.2. Eredmények

A dolgozatban bemutatott szoftvert egy Ender és egy Craftbot nyomtatóra is teszteltem, ezeknek különböző a tengelykiosztása, és kijelenthető, hogy a célt maradéktalanul sikerült elérni, a program képes volt lekezelni mindkét felépítésű gépet, a szerkezetek megjelenítése, illetve az alkatrészek megfelelő mozgatása sem okozott gondot. Kijelenthető tehát, hogy a Cartesian nyomtatók általánosítása sikeres, és egy gép modelljének, illetve szerkezetének URDF leírásának felhasználásával létrehozható annak digitális mása. A virtuális térben tetszőleges nyomtatók működtethetők, az elhelyezett okostáblákon pedig az adott célnak megfelelő tartalmak tölthetők be, ezzel még speciálisabbá téve a tanórát.

A sebesség növelése lehetővé teszi a nyomtatás felgyorsítását, a térben történő szabad mozgással pedig közelről megfigyelhetőek a folyamatok, ezek növelik az oktatásban szemléltetőeszközként használt szoftver értékét.

A valós, és a digitális nyomtató által létrehozott nyomtatványok összehasonlítása után megállapítható, hogy azok gyakorlatilag megegyeznek, így a nyomtatás minőségében is sikerült megfelelő modellt alkotni. Sajnos felgyorsított nyomtatás esetén a nyomtatvány kevésbé lesz részletes, mivel rövidebb idő alatt kevesebb pontot tud felvenni a program a nyomtatófej pozícióiból.

Ez a szimuláció a STEAM Upgrade Erasmus+ projekt egyik eredményterméke, melynek segítségével a tanulók megérthetik az FDM nyomtatás alapelvét. Több országban használják általános iskolák és egyéb oktatási intézmények. A 3D Printing Workshop bemutatásához, illetve annak használatához angol nyelvű weboldalt készítettem, mely a MaxWhere által publikált https://tools.maxwhere.com/3d-printing-workshop/index.html linken keresztül bárki számára elérhető.

7.3. Javaslatok

A G-kód részletesebb feldolgozását javasolnám a még élethűbb szemléltetés érdekében, ilyenre példa a melegedés függvényében működő hűtés mellett egy második ventilátor kódból történő vezérlése.
Érdemes lenne továbbá más nyomtatótípusokra is kiterjeszteni a modellt, hogy a ritkábban előforduló mechanizmusokkal (pl.: polár nyomtató) is megismerkedhessenek a diákok.
A URDF mellett érdemes lenne USD (Universal Scene Description) leíró formátum kezelése is, így kibővülnének a megjelenítés lehetőségei.
Amennyiben sikerülne kapcsolatot létesíteni egy fizikailag is létező nyomtatóval, úgy lehetőség nyílna digitális árnyékként a virtuális térben is valós időben megjeleníteni annak működését. Ennél messzebbre is tekinthetünk, kétirányú kommunikáció létesítése esetén digitális ikerpár is alkotható a szimulációból.

Budapest, 2023. november 5.

Horváth Botond

Irodalomjegyzék

[1] 1.6.2. Quadratic and Cubic Bézier Curves. URL http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/Curves_learningObject2.html.

[2] 3D INTERACTIVE PHYSICS SIMULATION. URL https://www.new3jcn.com/simulation.html.

[3] 3D Printer Bed: Types and Calibration | Top 3D Shop.
URL https://top3dshop.com/blog/3d-printer-bed-types-and-calibration.

[4] 3dparade: Mi az a Delta technológia a 3D nyomtatók világában? - 3dparade, 2020. december. URL https://3dparade.com/2020/12/08/mi-az-a-delta-technologia-a-3d-nyomtatok-vilagaban/.

[5] admin: The world's first 3D Printer Simulator | CNC Simulator Blog, 2012. december.
URL https://cncsimulator.com/blog/?p=224.

[6] Péter Baranyi – Ádám Csapó: Definition and Synergies of Cognitive Infocommunications. *Acta Polytechnica Hungarica*, 9. évf. (2012) 1. sz.

[7] Senay Baydas – Bulent Karakas: Defining a curve as a Bezier curve. *Journal of Taibah University for Science*, 13. évf. (2019. december) 1. sz., 522–528. p. ISSN null.
URL https://doi.org/10.1080/16583655.2019.1601913. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/16583655.2019.1601913.

[8] Browse - PhET Interactive Simulations.
URL https://phet.colorado.edu/en/simulations/browse.

[9] J-M. Burkhardt – V. Corneloup – C. Garbay – Y. Bourrier – F. Jambon – V. Luengo – A. Job – Ph. Cabon – A. Benabbou – D. Lourdeaux: Simulation and virtual reality-based learning of non-technical skills in driving: critical situations, diagnostic and adaptation. *IFAC-PapersOnLine*, 49. évf. (2016. január) 32. sz., 66–71. p. ISSN 2405-8963.
URL https://www.sciencedirect.com/science/article/pii/S2405896316328634.

[10] Cartesian 3-D Printer - MATLAB & Simulink. URL https://www.mathworks.com/help/sm/ug/cartesian_3d_printer.html;jsessionid=bffe7b655f674ed450a37d1117a4.

[11] Chris Christou: Virtual Reality in Education. In *Affective, Interactive and Cognitive Methods for E-Learning Design: Creating an Optimal Education Experience*. 2010, IGI Global, 228–243. p. ISBN 978-1-60566-940-3. URL https://www.igi-global.com/chapter/virtual-reality-education/www.igi-global.com/chapter/virtual-reality-education/40560.

[12] CubicBezierCurve3 – three.js docs.
URL https://threejs.org/docs/#api/en/extras/curves/CubicBezierCurve3.

[13] Kristóf Kenéz Drexler: 3d nyomtatás szimuláció maxwhere környezetben. 2021. december.

[14] Amanda K. Edgar – Susie Macfarlane – Elissa J. Kiddell – James A. Armitage – Ryan J. Wood-Bradley: The perceived value and impact of virtual simulation-based education on students' learning: a mixed methods study. *BMC Medical Education*, 22. évf. (2022. november), 823. p. ISSN 1472-6920. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9709374/.

[15] Electron IPC Response/Request architecture with TypeScript - LogRocket Blog. URL https://blog.logrocket.com/electron-ipc-response-request-architecture-with-typescript/.

[16] Figure 2. Structural configurations of major 3D food printers. (A)...
URL https://www.researchgate.net/figure/Structural-configurations-of-major-3D-food-printers-A-Cartesian-B-Delta-C-Polar_fig2_351468532.

[17] Lill Maria Gjerde Johannessen – Mathias Hauan Arbo – Jan Tommy Gravdahl: Robot Dynamics with URDF & CasADi. In *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)* (konferenciaanyag). 2019, 1–6. p.
URL https://ieeexplore.ieee.org/abstract/document/8988702/citations#citations.

[18] Sandra Helsel: Virtual Reality and Education. *Educational Technology*, 32. évf. (1992) 5. sz., 38–42. p. ISSN 0013-1962.
URL https://www.jstor.org/stable/44425644. Publisher: Educational Technology Publications, Inc.

[19] Home. URL https://cncsimulator.com/h/Home.html.

[20] Ildiko Horvath: Innovative engineering education in the cooperative VR environment. In *2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)* (konferenciaanyag). Wroclaw, Poland, 2016. október, IEEE, 000359–000364. p. ISBN 978-1-5090-2645-6.
URL http://ieeexplore.ieee.org/document/7804576/.

[21] Mustafa Hussein – Carl Nätterdal: The Benefits of Virtual Reality in Education- A comparision Study. 2015. július. URL https://gupea.ub.gu.se/handle/2077/39977. Accepted: 2015-07-20T13:38:15Z.

[22] Interprocess Communication Using Message Passing. In Weijia Jia – Wanlei Zhou (szerk.): *Distributed Network Systems: From Concepts to Implementations*. Network Theory and Applications sorozat. Boston,

MA, 2005, Springer US, 79–103. p. ISBN 978-0-387-23840-1.
URL https://doi.org/10.1007/0-387-23840-9_5.

[23] Mark Otto Jacob Thornton: Bootstrap. URL https://getbootstrap.com/.

[24] Lars Kunze – Tobias Roehm – Michael Beetz: Towards semantic robot description languages. In *2011 IEEE International Conference on Robotics and Automation* (konferenciaanyag). 2011, 5589–5595. p. URL https://ieeexplore.ieee.org/abstract/document/5980170. ISSN: 1050-4729.

[25] MaxWhere - Virtual spaces with the benefits of reality. URL https://www.maxwhere.com/.

[26] Ramakrishnan Mukundan: Quaternions. In Ramakrishnan Mukundan (szerk.): *Advanced Methods in Computer Graphics: With examples in OpenGL*. London, 2012, Springer, 77–112. p. ISBN 978-1-4471-2340-8. URL https://doi.org/10.1007/978-1-4471-2340-8_5.

[27] NCT Ipari Elektronikai Zrt. URL https://www.nct.hu/letoltes2_.php?menu=nct201.

[28] Plant simulation software | Siemens Software. URL https://plm.sw.siemens.com/en-US/tecnomatix/products/plant-simulation-software/.

[29] Process Simulate Digital Industries Software.
URL https://webinars.sw.siemens.com/de-DE/process-simulate/.

[30] Richard Baguley published: Polar 3D — 3D Printer Review, 2015. november.
URL https://www.tomsguide.com/us/polar-3d-printer,review-3206.html.

[31] Quaternions. URL https://www.tobynorris.com/work/prog/csharp/quatview/help/orientations_and_quaternions.htm.

[32] Quaternions.
URL https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html.

[33] John Peter Rothe: *The Safety of Elderly Drivers: Yesterday's Young in Today's Traffic*. Transaction Publishers. ISBN 978-1-4128-3882-5. Google-Books-ID: zkImIpFBTBEC.

[34] SCARA 3D Printer: All You Need to Know, 2021. május.
URL https://all3dp.com/2/scara-3d-printer-guide/.

[35] C. S. S. Script: Semantic Hierarchy Tree In Pure CSS - Treeflex, 2018. szeptember.
URL https://www.cssscript.com/semantic-hierarchy-tree-treeflex/.

[36] N. Shahrubudin – T. C. Lee – R. Ramlan: An Overview on 3D Printing Technology: Technological, Materials, and Applications. *Procedia Manufacturing*, 35. évf. (2019. január), 1286–1296. p. ISSN 2351-9789. URL https://www.sciencedirect.com/science/article/pii/S2351978919308169.

[37] Lejun Shao – Hao Zhou: Curve Fitting with Bézier Cubics. *Graphical Models and Image Processing*, 58. évf. (1996. május) 3. sz., 223–232. p. ISSN 1077-3169.
URL https://www.sciencedirect.com/science/article/pii/S1077316996900192.

[38] Martin Theobald – Mauro Sozio – Fabian Suchanek – Ndapandula Nakashole: URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules. 2010.
URL https://pure.mpg.de/pubman/faces/ViewItemOverviewPage.jsp?itemId=item_1323727.
Publisher: Max-Planck-Institut für Informatik.

[39] The Types of FDM 3D Printers: Cartesian, CoreXY & More, 2023. július.
URL https://all3dp.com/2/cartesian-3d-printer-delta-scara-belt-corexy-polar/.

[40] VIRTUAL DRIVING SCHOOL | Simulation.
URL https://www.cgasimulation.com/virtual-driving-school/.

[41] Virtual Medical Simulation - Virtual Patient - Practical Training.
URL https://virtualmedicalsimulation.com/.

[42] Roger A. Wehage: Quaternions and Euler Parameters — A Brief Exposition. In Edward J. Haug (szerk.): *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, NATO ASI Series konferenciasorozat. Berlin, Heidelberg, 1984, Springer, 147–180. p. ISBN 978-3-642-52465-3.

[43] xml2js, 2023. július. URL https://www.npmjs.com/package/xml2js.

[44] Yaw, Pitch, and Roll Diagrams Using 2D Coordinate Systems – Automatic Addison, 2020. március. URL https://automaticaddison.com/yaw-pitch-and-roll-diagrams-using-2d-coordinate-systems/.