

Kutatási beszámoló

Építészeti és BIM folyamatok algoritmizálása DWG alapon

Mentor:

Név: Dr. Máder Patrik Márk

Titulus: Bimprove kft. cégvezető, BIM SKILL LABOR kutatócsoport tag

Mentorált:

Név: Ráczy Viktor Norbert

Kelt.: Pécs 2024.04.16.

Tartalom

Bevezetés	3
Köszönetnyilvánítás	3
1. BIM története	4
2. Módszertan	6
2.1 Kutatási célok meghatározása	6
2.2 DWG és PDF rajzok átalakítása és kezelése	7
2.3 Programozás implementálása az algoritmizálás folyamatába.....	7
3. Algoritmusok.....	8
3.0 Az algoritmusok elkészítéséhez használt szoftverek	8
3.1 DWG – PDF alapú modellezés	9
3.1.0 Módszertan	9
3.1.1 DWG – PDF állományok információ tartalma közötti különbség	15
3.1.2 DWG-ből kinyerhető adatok Revit-ben és Dynamo-ban	18
3.1.3 Szükséges információk meghatározása a modellezés és erőforrások optimalizálásához ...	20
3.1.4 Létrehozott algoritmus	23
3.2 Geodéziai felmérés - 2D DWG -> 3D terep	28
3.2.0 Módszertan	28
3.2.1 DWG import és felhasználás	29
3.2.2 Szövegek Pozíciójának vizsgálata a pontokhoz képest	31
3.2.3 Magasság olvasása, pontok emelése	34
4. Összegzés	38
5. Fogalomtár	40
6. Hivatkozások	42
7. Ábrajegyzék.....	44

Bevezetés

A CAD (Computer-Aided Design) szoftverek története a 20 század közepére nyúlik vissza. [1], [2] A megoldásra akkor DAC-ként (Design Automated by Computer) hivatkoztak. Majd 1963-ban Ivan Sutherland létrehozta az első CAD szoftvert, amelyet Sketchpad-nak nevezett el. A program Sutherland egyik Phd téziséhez készült az MIT-n és már grafikus felülettel is rendelkezett. Innovációja a grafikus bevitel lehetővé tétele volt, amelyet egy CRT monitor és egy fénytoll segítségével valósított meg. A következő években a CAD szoftverek iránti igény fokozódott és először a mérnöki területeken - például: autógyártás, repülőgépgyártás - indult rohamos fejlődésnek. Ehhez kapcsolódóan szükséges megemlíteni, hogy ebben az időszakban a CAD és CAM (Computer-Aided Manufacturing) szorosan összefonódott. A számítógéppel tervezett gyártás során (CAM), kihasználva a kézi rajzokhoz viszonyított magasabb precizitást, CAD szoftverekkel előállított műszaki tervek kezdtek alkalmazni. Az idő előre haladtával, a technológia fejlődésével pedig a CAD szoftverek elérhetővé váltak a közepes méretű vállalkozások számára is, valamint egyre több területen váltak képessé a műszaki tervezés hatékonyságának növelésére.

A következő nagy lépés a rajzok 3D drótvázza alakítása volt. Az első 3D modellező rendszert 1972-ben hozták létre. [2] A GLIDE egy adatbázisokat tartalmazó számítógépes nyelv, mely tartalmaz minden tervezéshez és nagyobb kivitelezési projektekhez szükséges információt. Alkalmazásával, a BIM felhasználójával modellekhez hasonlóan, többletinformáció közölhető a CAD állomány. Ebben az évben hozták létre a CATIA-t is, mely megtalálható a modern mérnöki tervezésben is.

Köszönetnyilvánítás

Jelen kutatás a “ Kárpát-medence Tehetségkutató Alapítvány – Kutatói ösztöndíj” program keretei között készült, valamint a BIM SKILLS LAB kutatócsoport támogatásával valósulhatott meg.

A “szabad”, matematikai függvényekkel leírható felületek létrehozására 1989-ben továbbfejlesztették a 3D CAD rendszereket, mely elsősorban csak a Silicon Graphics munkaállomásokon volt elérhető, majd 1993-ban a CAS Berlin megalkotta a számítógépen is elérhető NURBS modellezőt, a NöRBS-öt. [3] Ezt követően a 3D modellekben tárolt adatok könnyebb és gördülékenyebb kezelésére létrehozták a STEP fájlformátumot, mely napjainkra nemzetközi szabvánnyá [4]vált. Majd létre jött a mostanában is használt tervező és modellező szoftverek nagyrésze. Ide sorolhatóak például: Autodesk Revit [5], SolidWorks [6], SketchUp [7].

1. BIM története

A BIM módszertanok létrejöttének idejét nehéz pontosan meghatározni[8]. Kialakulását a CAD/CAM rendszerek folyamatos fejlesztése eredményezte. A CAD programokhoz hasonlóan, létrejöttének kiváltó oka a kézzel való rajzolás technológiai hátrányai voltak. A fő problémákat a rajzok minőségének folyamatos romlása, valamint azok elkészítési ideje jelentette. Utóbbi miatt például egy felújítás alkalmával nehéz és időigényes volt a felmerülő változások rajzban való lekövetése, továbbá az emberi hibafaktor is magas volt, a tervek nehéz ellenőrizhetősége miatt.

Az 1970-es évek közepén Charles M. Eastman publikálta a “Building Description System” (BDS) prototípusát. Ez egy olyan rendszer ötlete volt, mely parametrikus, magas minőségű 3D megjelenítést biztosít, integrált adatbázissal, melyben vizuális és mennyiségi számításokat, analíziseket lehet végezni. Ezt követte az automatikus modell ellenőrzés fogalma, mely során a “terv szabályosságát” vizsgálták. Az említett folyamat modern olvasatban a BIM módszertanhoz tartozik.[8] A BDS már tartalmazott grafikus felhasználói felületet, valamint orthografikus és perspektivikus nézeteket is, így ez volt az első projekt, amely létrehozta az első építési adatbázist. (building database)

A 20. század végén [9] a piac fő részvevői a CAD alapú szoftverek fejlesztésére fókuszáltak. Ezen fejlesztések során ötvözték a CAD és AEC technológiákat, így megalapozták a parametrikus 3D modellezést, majd az IM rendszer implementálásával megalkották a BIM módszertani igényeket kielégítő szoftvereket.

Témához való személyes kötődésem

Építészeti tanulmányaim 2. évében kezdtem el megismerkedni a BIM módszertanokkal. Ekkor volt egy 2 féléves tárgyunk, mely gyakorlati óra keretei között Autodesk Revitet tanultunk, az előadásokon pedig bevezettek minket a BIM módszertanokba, mely felkeltette az

érdeklődésem. A tárgy elvégzése után, jelentkeztem egy építész irodába gyakornokként, ahol BIM folyamatokkal foglalkoznak. Gyakornoki időm alatt már részt vettem valós piaci projektben is, majd a letelte után maradtam és jelenleg is itt dolgozom. Tanulmányaim és a munka során is végeztem ismétlődő, robotias folyamatokat, melyek kapcsán fogalmazódott meg bennem a kérdés, hogy létezik-e ezekre gyorsabb megoldás. Bizonyos esetekben voltak kezdeményezések, de ezek nem voltak hatékonyak.

Munkám során megismerkedtem a parametrizálással és algoritmizálással Autodesk Revitben. Utóbbihoz a Dynamo nevű bővítményét használtam, melyet autodidakta módon sajátítottam el. Kezdetben voltak próbálkozásaim a munkafolyatok gyorsítására, de ezek nem vezettek eredményre. Az akadályt az elemek kezelése jelentette. A Dynamo a számításokban részt vevő elemeket sorszámozott listákban kezeli, melyek lehetnek akár listába ágyazott listák is. Ezek kezelése és feldolgozása, tisztán alapelemekkel, egy bizonyos elemszám és listaszerkezet után nehéz, valamint nagy számítási kapacitást igényel. Az algoritmus gyorsítására segítségül lehet hívni a szöveges programozási nyelveket Python Script-ek formájában. A scriptek használatának szükségessége adta meg a kezdőlökést a Python programozási nyelv elsajátításához. Bekerültem a BIM SKILLS LAB kutatócsoportba, ahol a Tématerületi Kiválósági Program keretein belül kamatoztatni tudtam tudásomat. Létrehoztunk egy munkahelyi leesésvédelmi algoritmust, mely egy 3D modellt használva azonosítja a veszélyzónákat, majd különböző feltételek szerint biztonsági korlátokat szerkeszt. A jobb láthatóság érdekében ezen zónák színekkel is jelölve vannak. Jelenleg programozási tudásom szinten tartására és továbbfejlesztésére szabadidőmben programozási kihívásokat oldok meg, programozói versenyekre jelentkezem és felfedezek más programozási nyelveket is.

Az építészeti, legfőképp a BIM alapú modellkészítési eljárások során gyakran előfordul, hogy 2D digitális rajzok alapján (DWG, PDF) kell elkészíteni, valamint ellenőrizni a modellt. Jelenleg 2D DWG esetén Autodesk AutoCad-ben már van összehasonlító funkció, mely az elemek egyedi azonosítóját vizsgálja, majd jelöli az eltéréseket, az új és törölt elemeket. A módosítások átvitele a BIM modellbe továbbra is manuálisan történik, ugyanakkor ez már az első lépés lehet az automatizálás felé. Véleményem szerint programozáson keresztül a folyamat részben vagy teljes mértékben automatizálható. A probléma aktualitását alátámasztja, hogy egy épület tervezésében résztvevő szakágak által generált 2D állományok feldolgozása BIM modellben manuálisan történik. A modellezési folyamat során felmerülő hibákat, észrevételeket a modellezőnek dokumentálnia kell, majd esetlegesen az észrevétel pontos helyét visszavezetni a kapott szakági rajzra. A dokumentálási folyamat időigénye a hiba/észrevétel komplexitásától, valamint a dokumentálás módjától is függ, így indokolt lehet

egy olyan algoritmus létrehozása, mely a modellezés kezdete előtt képes felderíteni és dokumentálni a 2D állományokból származó hibákat. Már több kutatás is rávilágított az algoritmizálás nyújtotta lehetőségekre[10], [11], [12], ugyanakkor a DWG rajzok felhasználása összetett feladat.

2. Módszertan

2.1 Kutatási célok meghatározása

Kutatásom rövid távú célja, hogy a meghatározott módszertanok alapján létrehozzam az algoritmusokat, a kitűzött folyamatok elvégzésére. A DWG és PDF állományok felhasználását célzó algoritmusokkal szeretném javítani a szoftverek közötti átjárhatóságot, valamint az állományok használhatóságát. A kutatás végzésének idején, a használt szoftverekben sajnos csak kis mennyiségű információ nyerhető ki ezen állományokból, ezért szükséges meghatározni az állományokkal szemben támasztott követelményeket, mely egy későbbi pontban kerül kifejtésre.

További rövidtávú céljaim közé tartozik az emberi hibafaktor csökkentése, melyet a kutatásban létrehozott algoritmusokban automatikus modellezéssel valósítok meg, falburkolatok, valamint térbeli pontok modellezésével. A falburkolatot modellező algoritmusnál a hibafaktor a burkolatok falszerkezetekhez illesztéséből, a használandó falburkolat típusának kiválasztásából, a referenciavonal helyes meghatározásából, valamint a fal alsó és felső referenciájának beállításából áll. Jelen faktorok közül a létrehozott algoritmusban egyedül az alsó és felső referencia meghatározásához nem végeztem vizsgálatot, mert a teszt projekt fájlomban csak két szint volt meghatározva. A többi faktor megoldását a DWG állományból vagy a külső táblázatokból automatikusan határoztam meg. A felmérési pontokat térbe mozgó algoritmusnál az élőmunkaigényt szerettem volna csökkenteni a 2D felmérési pontok 3D ponttá alakításával. A felhasznált DWG állománynál a nehézséget az okozza, hogy a pontokhoz tartozó magassági adatot tartalmazó szövegdoboz nem azonos távolsággal és irányban került lehelyezésre, így az algoritmus határfoka nem érheti el a 100%-ot.

Jelen kutatás lezárása után további célom a létrehozott algoritmusok továbbfejlesztése, valamint olyan további algoritmusok létrehozása, melyek tovább tudják javítani az előző bekezdésekben meghatározott tényezőket. Fejlesztési területek lesznek a falburkolat modellező algoritmus esetében a referencia szintek, valamint lehelyezett födécek vizsgálata, ezzel elkerülve azt, hogy a lehelyezett burkolat esetlegesen belelógjon a felette vagy alatta

elhelyezett földem elembe. Továbbá szeretnék létrehozni egy modell ellenőrző algoritmust is, mely a már lehelyezett elemeket vizsgálja a modellező algoritmus bemeneti állományai alapján, így kiváltható a manuális modell ellenőrzés, ugyanakkor a helyes működéshez a falburkolatok változásait fel kell vezetni a használt külső táblázatokba is.

Hosszú távú célom, hogy rávilágítsak az algoritmizálás építészeti lehetőségeire, valamint későbbi doktori tanulmányaim alatt és után oktatni szeretném építész hallgatóknak a kutatásban használt szoftvereket.

2.2 DWG és PDF rajzok átalakítása és kezelése

DWG és PDF állományok is importálhatóak Autodesk Revit szoftverbe, ugyanakkor ezek különféle célú automatizált, algoritmussal támogatott felhasználásra való előkészítése bonyolultabb feladat. Ebből a szempontból vizsgálva a PDF formátumú állományok információtartalmát, megállapítható, hogy azok nehezebben kezelhetők és hiányosabbak, mint a DWG fájlok. Előbbi például elemekre bontásnál a szaggatott vonalakat rövid vonalakra bontja fel.

A PDF és DWG állományok beolvasás utáni átalakítására már léteznek különböző megoldások, azonban ezek egyedi paramétereket igényelnek. Az általam vizsgált átalakítók a rajzi elemekből olvasták az elhelyezendő elemek fajtáját (pl.: gipszkarton fal), majd ha a mért vastagságú fal nem állt rendelkezésre, akkor az előre megadott típusból hozott létre egy új típust és helyezte le azt.

2.3 Programozás implementálása az algoritmizálás folyamatába

Mind a Grasshopper, mind a Dynamo használatával felépített algoritmus, függetlenül attól, hogy használtunk-e Python Script Node-ot vagy sem, már önmagában is programozásnak számít, nemhiába nevezik az általuk biztosított teret vizuális programozási felületnek. Ugyanakkor egy algoritmus sebessége, ami tisztán a beépített Node-okat használja, növelhető Python Scriptek használatával. Mind a Dynamo, mind a Grasshopper, hozzáférnek a saját fő szoftverük (Revit, Rhinoceros) API-jához, melyek kibővítik a Scripteknél használható funkciók listáját. Azonban a Python programozási nyelv ilyen mértékű használatához már magasabb szintű programozási tudás szükséges, mert így a funkciók eléréséhez ismerni kell a Revit API -t is.

A kutatásban bemutatott algoritmusaimban, a szükséges tudás hiányában, ilyen szintű Scripteket nem alkalmaztam, azonban implementáltam a beépített Node-okat, így szövegesen is tudtam használni őket. Az Autodesk Revit „PyRevit[13]” bővítménye képest egy külső

szoftverben írt Python Script közvetlen futtatására. Úgy gondolom, hogy ez lehet az algoritmizálás következő szintje, melyhez már nem szükséges a vizuális programozási felület, azonban úgy gondolom, hogy egy ilyen algoritmus írásához átfogó programismeret is szükséges.

3. Algoritmusok

3.0 Az algoritmusok elkészítéséhez használt szoftverek

A PDF – DWG állomány alapú modellező algoritmushoz Autodesk Revit, valamint Dynamo szoftvereket, a geodéziai adatok feldolgozásához Rhinoceros 7, továbbá Grasshopper programokat használtam.

Az Autodesk Revit egy 3D modell készítő tervező szoftver, amely használható az építéset, épületgépészet, valamint építés iparágakban is. A szoftver alkalmas BIM módszertanokra alapuló 3D modell készítésére is, mely lehetőség miatt megfelelő alapot biztosít az algoritmusaim létrehozásához.

A Dynamo az Autodesk Revit egyik beépülő modulja. Vizuális programozási felületet biztosít, melyen elérhetőek a Revit projektben tárolt elemek, továbbá parametrikus módon akár létre is hozhatóak vagy módosíthatóak. Legkisebb elérhető egysége a Node, mely egy funkciót tartalmazó egység. Lehet több bemeneti és kimeneti portja, azonban előfordulhat az is, hogy csak kimeneti portjai vannak. A következő egységek a Code Block-ok, melyekbe szövegesen vihetőek be a parancsok, melyeket a programnak végre kell hajtania, ugyanakkor ezek programozása nehéz a használt programozási nyelv miatt. Az elemek következő szintje a Python Script Node-ok, melyekbe Python programozási nyelven írt kód vihetőbe és futtatható le. Ennek a Node-nak több bemenete is lehet, azonban csak egy kimeneti portja lehet. Az algoritmusok elkészítéséhez főként ezeket a Node-okat használtam. Lehetőség van egyedi és mások által készítette egyedi Node-ok és Node csomagok letöltésére is, azonban az algoritmusok elkészítéséhez próbáltam kerülni ezek használatát.

A Rhinoceros egy 3D modell készítő szoftver, melyel könnyen és gyorsan készíthetőek tetszőleges 3D modellek. Több verziója is létezik, ebből az algoritmus készítésének idejében a Rhinoceros 7 verziót használtam.

A Grasshopper a Rhinoceros egyik beépülő modulja, mely a Dynamo-hoz képest vizuális programozási felületet biztosít, ugyanakkor a kezelőfelület és a használható funkciók nagy mértékben különböznek. A Grasshopper a Rhino-val való élő kapcsolat mellett képes a

Graphisoft Archicad, valamint a Revit szoftverekkel is élő kapcsolat létrehozására. A Dynamohoz hasonlóan lehetőség van Python Script-ek használatára, azonban a Geodéziai adatokat feldolgozó algoritmushoz főként a beépített Node-okat használtam.

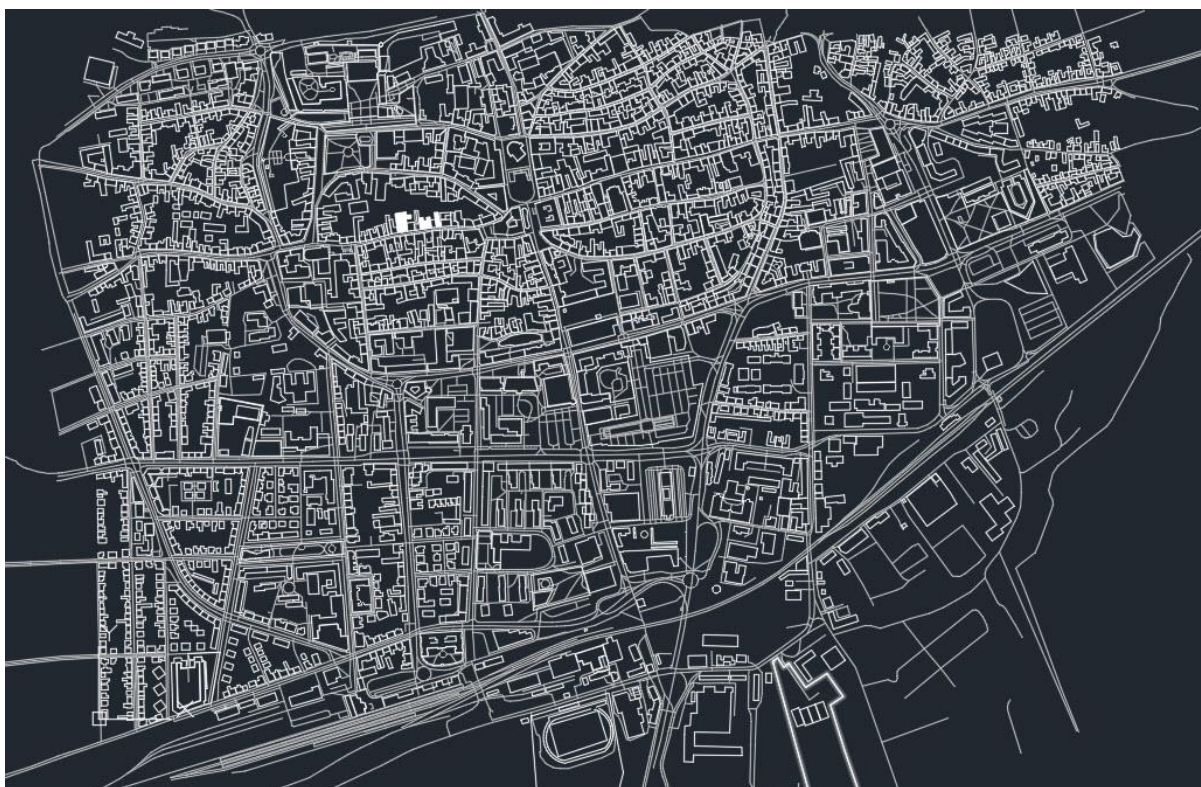
3.1 DWG – PDF alapú modellezés

Alapvetően két féle módon modellezhetünk DWG vagy PDF állományból: manuálisan, a tervezőszoftverbe való importálása nélkül, a méretek olvasásával, majd követésével, vagy szintén manuálisan, de az állományok importálásával. Utóbbi esetben jobban kell figyelnünk a modellelemek elhelyezésére, mert az importált állományokban található pontatlanságok a modellben is pontatlanságokat, összemetsződési hibákat szülhetnek.

Az algoritmus készítése során importáltam a DWG állományt Autodesk Revit szoftverbe, a tárolt adatok kinyerése végett. Az importálásra két lehetőség áll rendelkezésre, importálás és linkelés. A különbség a két módszer között, hogy az importálás egyirányú, ami azt jelenti, hogy ha változás történik az importált állományban, akkor az nem lesz látható a tervezőszoftverben, mert ott nem frissíthető. Linkelésnél az állomány megtartja a fájl elérési útvonalát, így ha változás történik, az importált állomány frissíthető. Az algoritmus szempontjából nincs különbség a linkelt és importált állomány között. A teszteléshez szükséges volt a DWG állomány módosítása, majd frissítése, így a linkelést választottam.

3.1.0 Módszertan

A teszt állományom egy Pécsről készített DWG. Ebben jelen vannak az épületek és az úthálózat. Jelen kutatásban az épületek leképezésével foglalkozom. Az állományban ezek a „Polyline” eszközzel készültek, így egy épület teljes kontúrja meghatározható egy elemmel. A teszteléshez Autodesk AutoCAD program segítségével készítettem úgynevezett blokkokat is. Ezek a blokkok, több épület csoportosításából alakultak ki. A blokkoknál lehetőség van egyedi paraméterek hozzáadására, melyekhez készítettem egy „Test01” nevű paramétert, ami „test” szöveggel került feltöltésre. Az így kialakult DWG állományt egy üres Revit projektbe importáltam.



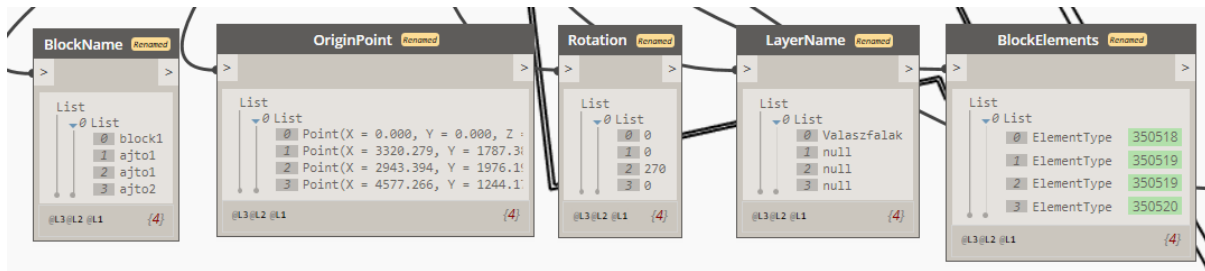
1. ábra - Pécs várostérkép részlet Autocad-ben (saját ábra)

Az 1. ábrán látható a első kiindulási állomány, mellyel elsősorban az adatátvitelt teszteltem.

Az importált DWG objektumain Reviten belül módosítást nem végeztem. A Dynamo nevű kiegészítő segítségével megkerestem az importált DWG-t, majd megjelenítettem a benne tárolt paramétereket és geometriát.

Az állomány geometriákra bontásával a tárolt információk elvesznek, így információ átvitel szempontjából ez a módszer nem alkalmazható. A DWG-ből kiolvasott paraméterek között sem volt megtalálható a „Test01” paraméter, így kijelenthető, hogy importálás után, a blokkokban tárolt információ nem jelenik meg a DWG állományban.

A DWG további bontásához egy fejlesztő által készített egyedi csomagot használtam (GeniusLocí). A csomagban található több egyedi Node is, melyekből az egyik ki tudja emelni az importált DWG állományból a blokkokat, visszaadva azok tárolt főliáját, nevét, illesztési pontját és szögét, ha elforgatásra került. Ezt a Node-ot dolgoztam tovább, hogy ki tudjam emelni a teljes blokk elemet, de sajnos a kiemelés után a tárolt paraméterek közül továbbra sem nyerhető ki a „Test01” paraméter és „test” paraméterérték.



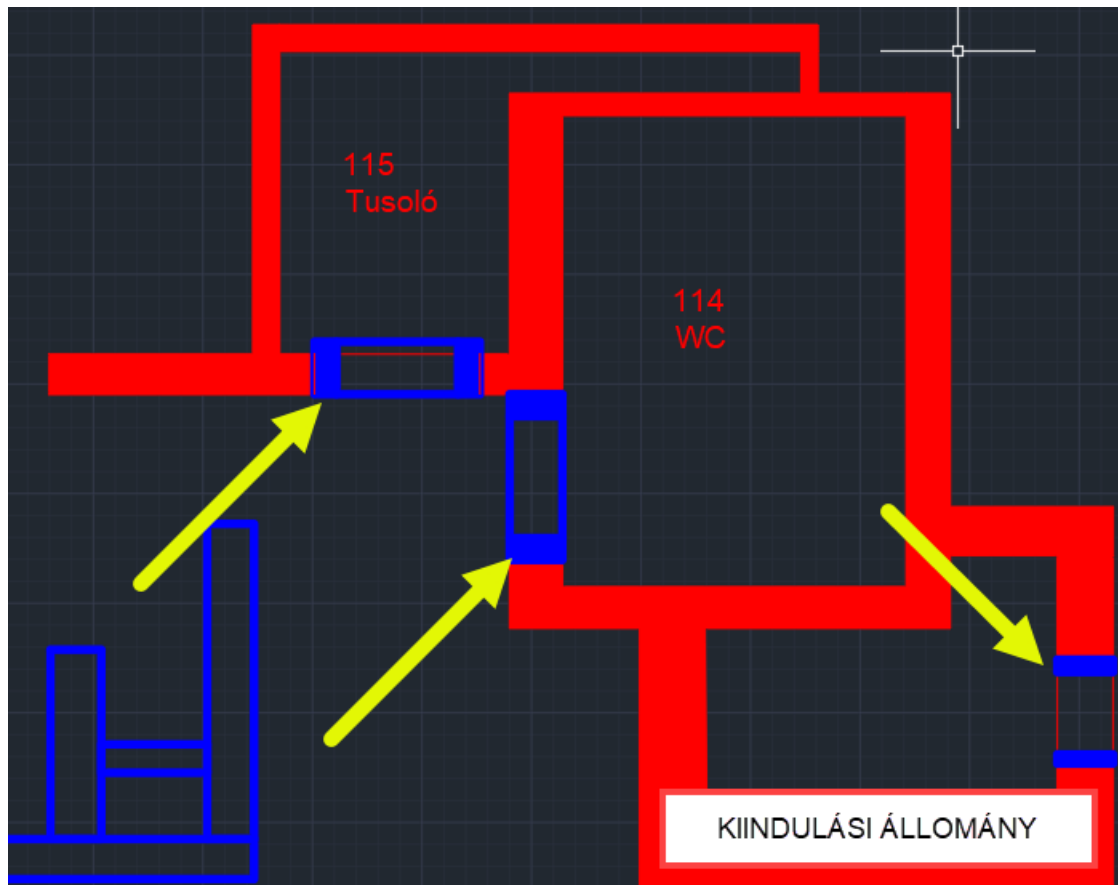
2. ábra - Dynamo-ban kinyert adatok a DWG import után (saját ábra)

A Revit API releváns fejezeteinek átnézése után megállapítottam, hogy a DWG-ben szereplő blokkokat szimbólumként értelmezi a program, ezért sajnos ebből az elemből csak a neve, az azonosítója és a geometriája nyerhető ki, a hozzáadott egyéb paraméterek nem.

A fent említettek alapján, valamint a 2. ábrán látható információk alapján kijelenthető, hogy tisztán DWG-ből nem olvasható ki elegendő adat a modell generáláshoz és a modell ellenőrzéshez. Kiegészítő megoldás szükséges, jelen esetben egy külső adatbázis formájában, amelyből egy azonosítón keresztül össze tudja kapcsolni az elemeket a szükséges információval. Az Autocad „DATAEXTRACTION” parancsa lehetővé teszi, hogy a fájlból kimentsük a fájlban lévő elemekből az adatokat. Először kiválaszthatjuk, hogy egy új kigyűjtést szeretnénk-e létrehozni, vagy egy meglévőt szerkeszteni, emellett az új létrehozásánál lehetőség van egy meglévő kigyűjtést mintának használni. Ezt követően kiválaszthatjuk, hogy mely rajzokból szeretnénk adatot kigyűjteni. Itt lehetőség van a jelenlegi rajz elemeit is kiválasztani, amennyiben csak egy részét szeretnénk használni. Ezt követően kiválasztjuk a kimentendő elemek típusát és az exportálandó adatokat, végül pedig a mentési formátumot. A formátumok közül a .csv és a .xls használható Dynamo-ban adatok importálására.

Mint azt már előzőleg megállapítottam, Autocad-ben csak a blokkokhoz lehet egyedi paramétert csatolni, így ezek készítése és exportálása jelenti majd a hidat a két program között. A blokkok létrehozása egyszerű, de időigényes, kiváltképp ha egy épület alaprajzáról beszélünk. Egy blokk létrehozásához először meg kell rajzolnunk az elemeket, melyek a blokkot fogják alkotni, majd a blokk létrehozó paranccsal kiválasztani őket, meghatározni az elhelyezési pontot, majd kiválasztani, hogy törölni szeretnénk-e a megrajzolt elemeket a blokk lehelyezésével. Ha az elhelyezési pontot nem határozzuk meg, akkor alapértelmezetten a program koordináta rendszerének origóját használja ennek a pontnak. A későbbiekben ez azért jelent problémát, mert ha újból használni szeretnénk az adott blokkot, akkor az elhelyezési pontjánál fogja tudjuk lehelyezni a rajzba, ezért kell ennek a helyét előre pontosan meghatározni. Jelen algoritmushoz az ajtókat reprezentáló blokkokat hoztam létre, az

elhelyezési pont az ajtók egyik oldali középpontja volt. Két különböző ajtó blokkot hoztam létre és helyeztem le a falakba, összesen három helyre.



3. ábra - Az elhelyezett ajtó blokkok Autocad-ben (saját ábra)

A 3. ábrán láthatóak a lehelyezett ajtó blokkok az Autocad programban.

A gyári parancsok között nem található olyan, ami automatikusan létrehozná ezeket a blokkokat. Egyéni parancsot programozással lehet létrehozni. Ehhez három programozási nyelv áll rendelkezésre: AutoLisp, Visual Basic, .NET.

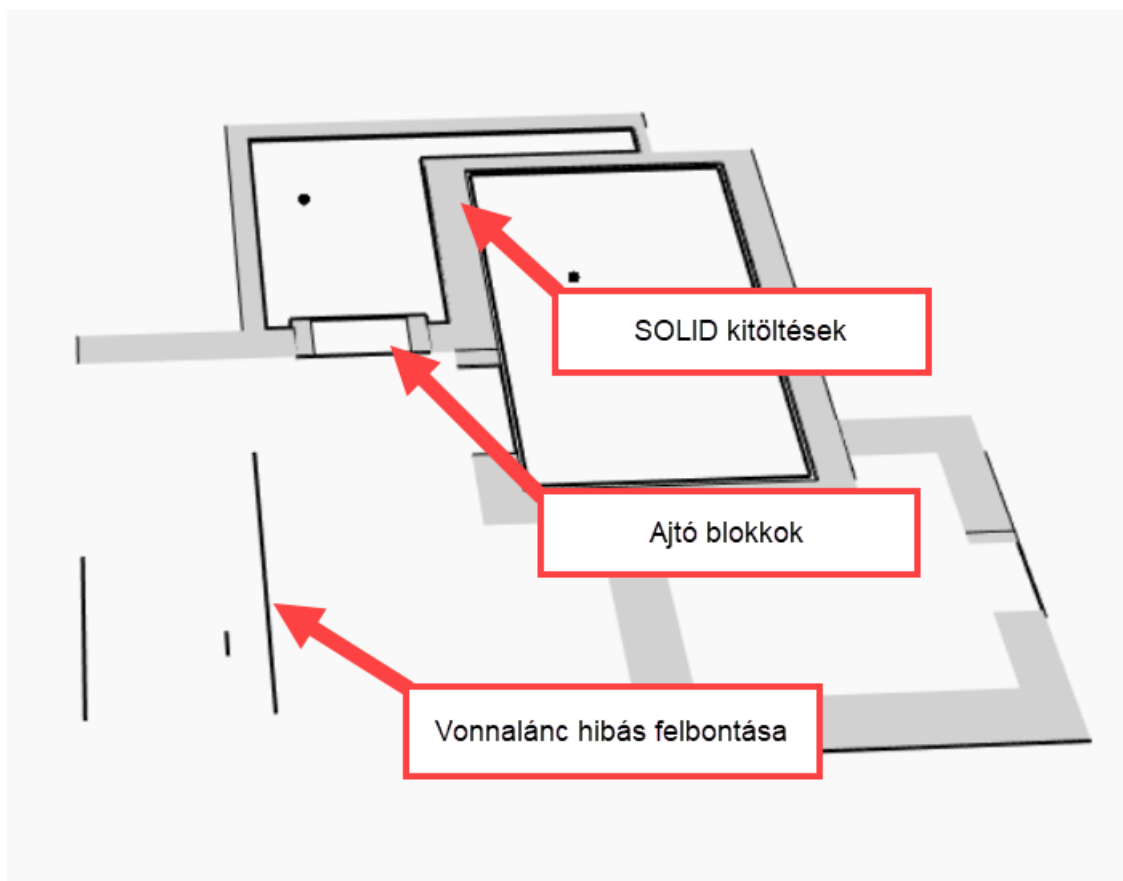
Az AutoLisp volt az első nyelv, mellyel automatizálni lehetett a munkafolyamatokat. A megalkotásánál a fejlesztők célja a könnyű használhatóság volt, hogy használata ne igényeljen komoly programozói tapasztalatokat.

A Visual Basic (VBA) később jelent meg, beépülő modulként letölthető jelenleg is az Autocadhez. A modulban létrehozhatóak felugró ablakok, melyek programozásához a bővítmény egy külön felületet biztosít. A VBA-t a Microsoft úgy fejlesztette, hogy hozzáférjen az alkalmazások objektum könyvtárához, ezáltal csak a használandó objektumok nevét kell tudni a funkcióik közvetlen eléréséhez. A VBA használható több Microsoft termékhez is, pl.: Word, Excel...

Az utolsó programozási nyelv a .NET. A többivel ellentétben, használatához egy külön fejlesztői környezet felállítása szükséges, valamint a C++ és a C# nyelvek ismerete is.

Egy jövőbeni fejlesztési irány lehet az elérhető nyelvek mélyebb vizsgálata, majd egy bővítmény fejlesztése a blokkok automatikus elkészítésére.

A módszertan teszteléséhez készítettem fal és falburkolati elemeket. Két fóliát használtam, egyet a válaszfal szerkezeti részének, a másikat pedig a falburkolat elemeinek. Az elemeket megrajzoltam, majd kitöltést helyeztem el. A különböző anyagoknak különböző kitöltést adtam. Az így elkészült elemekből az adatok exportálása után kiolvasható a szerkezeti besorolás, mint falburkolat és válaszfal, továbbá a kitöltés típusa is, így tudjuk az adott réteg anyagát is. Ebben az esetben nem használtam blokkokat, az azonosítást az elem fóliája és kitöltésének neve adja. Egy előre elkészített táblázatban meghatároztam az épületben használt fal rétegrendeket, emellett azt is, hogy melyik anyagnak melyik fólián kell lennie, ezáltal biztosítható egy rajzi ellenőrzés is, mert külön gyűjthetők azon elemek, melyeknél valamelyik adat hibás.



4. ábra - DWG betörése Revitbe, majd az elemek megjelenítése Dynamo-ban (saját ábra)

A probléma ezzel az azonosítási móddal az, hogy a kitöltések közül csak a SOLID kitöltést tudja megjeleníteni a Revit és a Dynamo is, melyet a 4. ábra szemléltet. Ez abban nyilvánul

meg, hogy a geometria kinyerésénél Dynamo-ban a kitöltések nem jelennek meg, a SOLID-ot kivéve, mert azt egy felület elemmel ábrázolja. A DWG rajzi elemekké történő szétbontása sem nyújt megoldást, mert akkor a program egyedi kitöltéseket hoz létre, melyeknek neve a fájl neve lesz.

Eddigi eredményeim alapján a legsokoldalúbban és hatékonyabban működő megoldásnak a blokkok használata tűnik. Ebben az esetben Dynamo-ban ki tudjuk olvasni a blokk nevét és a referencia pontját, ami blokkonként egyedileg megadható. A blokk neve magában elegendő adat az azonosításhoz, mert a név blokkonként egyedi. A referencia pont azért fontos, mert ha a blokkból kinyerjük a geometriai adatokat, akkor látható, hogy ezt a pontot a koordináta rendszer origójába helyezi. Így, ha a referenciapontot az origóban határozzuk meg, akkor az elemek elhelyezési pozíciójában tudjuk az elemek geometriáját is, mint ahogy a 2. ábra is mutatja.

A módszertan tesztelése után megállapítottam, hogy az összetartozó elemek kiszűrése sok vizsgálatot igényel, ezáltal a program lefutása lassúvá és erőforrás igényessé válik.

Jobban megvizsgálva az építészeti rajzokat, visszatérő elem a helyiségpecsét. A helyiségpecsét, egy helyiségen belül elhelyezett szövegdoboz, melyben szerepel a helyiség neve, a helyiség száma és általában a helyiség területe, mint kiegészítő adat. Autocad-ben nem lehet helyiségpecsétet elhelyezni, de szövegdobozt igen, ami ebben az esetben ugyan azt a funkciót fogja betölteni. A szövegdoboz adatainak exportálásával kinyerhető a helyiségben tárolt szöveg, valamint a szövegdoboz elhelyezési pontja is. Tudjuk, hogy minden helyiségben található helyiségpecsét, a kinyert koordinátával meghatározható a helyiség pontos helye és belőle egy egyszerű vizsgálattal megállapíthatóak a helyiség körvonalai is. A vizsgálat gyorsítása érdekében érdemes lehet Autocad-ben egy plusz vonallal körberajzolni a helyiségek belső oldalait. Az így kapott új vonalak egy külön fóliára helyezzük és blokkosítjuk, akkor Dynamo-ban csak ezeket a vonalakat kell vizsgálnunk a helyiség körvonalainak megállapításához. A vonal meghatározza nekünk a lehelyezendő falburkolati rétegek referenciavonalát is. A modellezéshez szükséges tudnunk, hogy a helyiségekben milyen falburkolat található. Ehhez egy külön táblázatot hoztam létre, ahova helyiségenként bevittem a falburkolat típusát. A teszteléshez egy helyiségben két falburkolatot is alkalmaztam úgy, hogy ezek horizontálisan vannak elvágva, így az egyik burkolatnak lábazati szerepe van. Az építészek gyakran használják ezt a burkolati megoldást, ezért fontosnak láttam vizsgálni ezt a módot is.

A modellben nem szeretnénk rétegenként modellezni a falakat, mert az épület alapterületének növelésével exponenciális jelleggel nő az elemek száma, ezáltal a fájl mérete is. A fájl

méretének növekedése miatt pedig csökken a modell kezelhetősége, ezáltal a használt program összeomlását is eredményezheti. Ennek elkerülése érdekében a Revit projekt fájlban, melyben az épületünk is lesz, létrehoztam a táblázat alapján réteges szerkezeteket. Ezek megnevezése a táblázatban feltüntetettekkel azonos.

Összegzés a módszertanhoz:

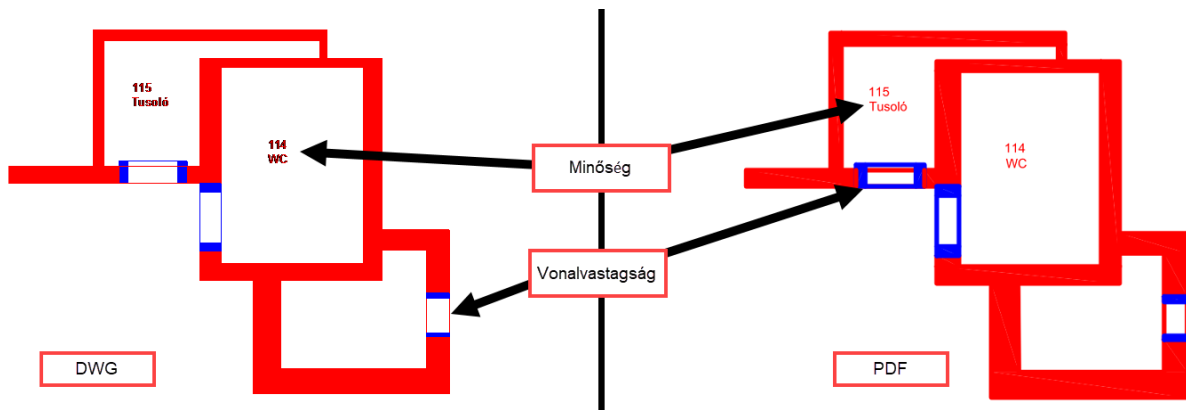
Szükségünk van egy DWG-re, melyben saját fóliája van a válaszfalak tartószerkezetének. A külön fólia azért fontos, mert így a többi elemtől függetlenül meg tudjuk modellezni ezeket a szerkezeteket, továbbá felhasználhatóak a későbbi számításokhoz.

Szükségünk lesz szövegdobozokra a helyiségekben, melyek tartalmazzák a helyiség számát, valamint ezen dobozoknak a helyiségen belül kell elhelyezkedniük, melyhez tartoznak. A tartalom tekintetében a többi információ, ami a helyiségpecsétben van számunkra nem fontos, mert a helyiség száma biztosan egyedi lesz, míg a helyiség neve megegyezhet két különböző helyiségnél. Az elhelyezés a határoló falak meghatározása miatt fontos.

Szükségünk lesz két különálló táblázatra. Az egyik táblázatnak tartalmaznia kell a helyiségek számát, valamint a helyiségben használt falburkolatot. Amennyiben két falburkolat található a helyiségben, akkor azokat is fel kell tüntetni, továbbá ehhez szükséges egy magassági adat is, ahol a falburkolat rétegrendet vált. A másik táblázatnak tartalmazni kell a használandó falburkolati rétegrendeket. Ezt az automatizálásnál nem fogjuk használni, a Revites elemek létrehozásához van rá szükség.

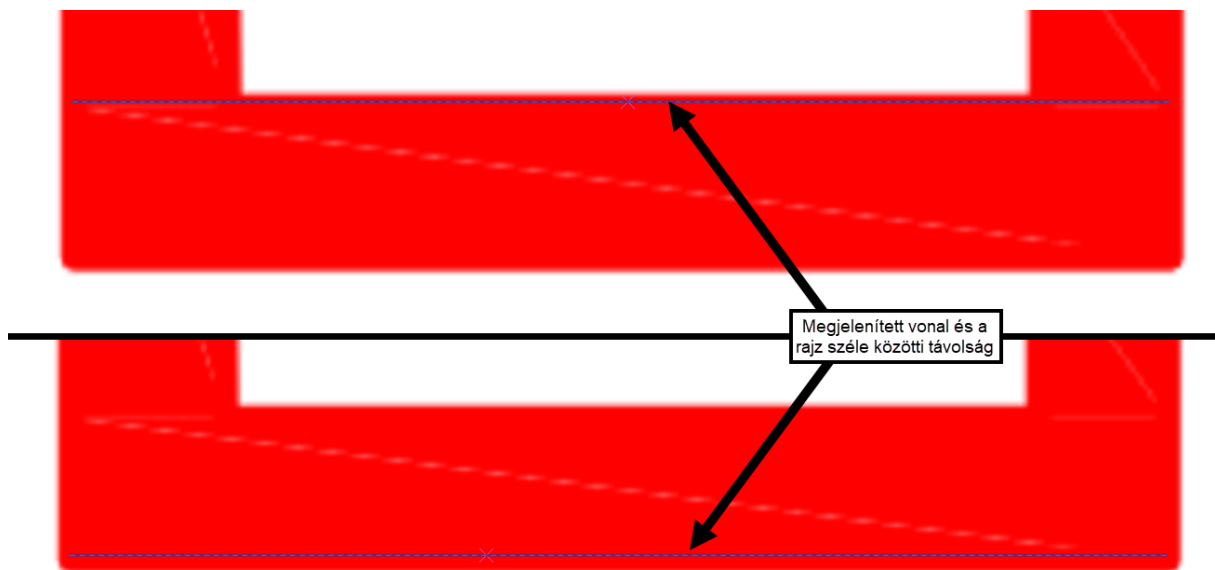
3.1.1 DWG – PDF állományok információ tartalma közötti különbség

Az állományok információtartalma közötti vizsgálatot az algoritmizálás megkezdése előtt vizsgáltam, az optimális eredmény elérése érdekében. Importáltam ugyan azt a rajzot PDF és DWG állományként is Revit szoftverbe, már az import után is látható a különbség.



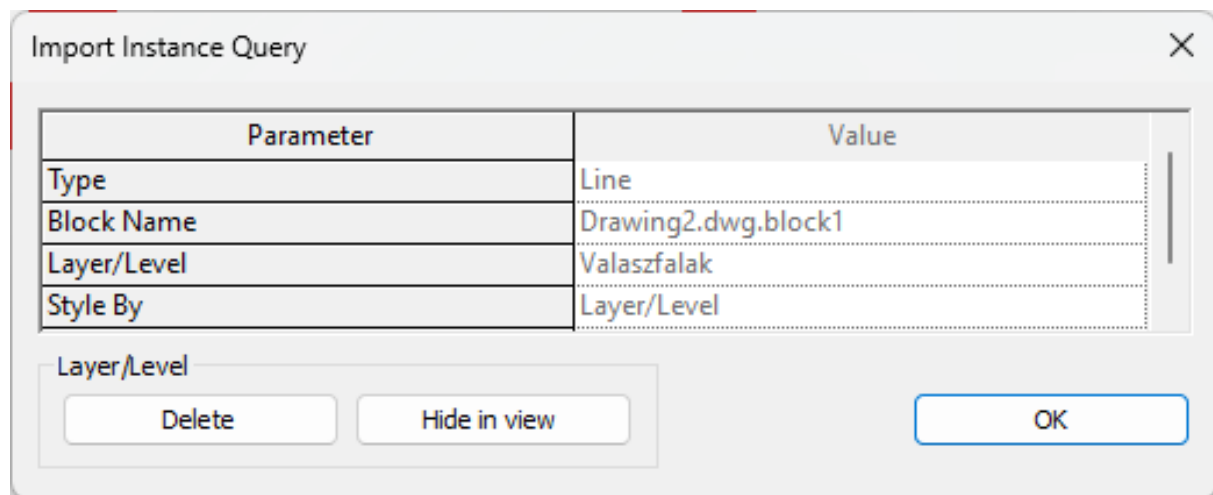
5. ábra - Állományok megjelenésének összehasonlítása (saját ábra)

Mint az 5. ábrán látható, vizuális különbségek is észrevehetőek a két állomány között. A PDF állomány az Autocad-ben beállított vonalvastagságokat használja, míg a DWG állomány vonalvastagságai főként szerkeszthetőek a Revit grafikai beállításain belül. Előbbi nagy mértékben megnehezíti a manuális modellezést, mert nem helyezhetőek le a vonalak metszéspontjaiba az elemek, valamint pontatlanságot is szül. Ugyanakkor meg kell említeni, hogy vektoros PDF állomány esetén bekapcsolható a SNAP mód, mely hatására kattinthatóvá válnak a vonalak, valamint metszéspontjaik. Ez azt jelenti, hogy egy elem lehelyezésénél megjeleníti nekünk a vonalakat, valamint metszéspontjaikat, ha elég közel visszük hozzájuk a kurzort. Azonban ezzel a móddal sem lehet precízem dolgozni az állománnyal, mert a vonalakat alacsony pontossággal, hozzávetőleges pozícióban jeleníti meg, továbbá egy egyenes vonalat is több részre bont.



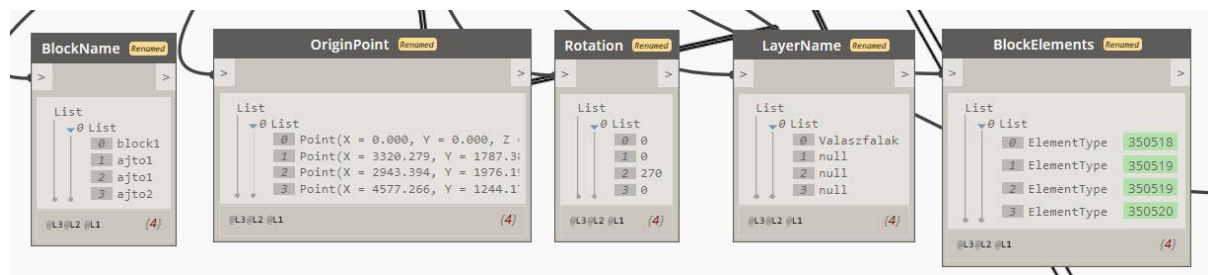
6. ábra - SNAP funkcióval a vonal érzékelése (saját ábra)

Az 6. ábrán látható, hogy bekapcsolt SNAP funkció mellett különböző távolságokra helyezi le a vonalat a rajz szélétől, így a közöttük lévő távolság sem egyezik meg az Autocad-ben mért távolsággal. Az eredeti távolság 136.77 mm, a Revitben mért távolság pedig 139.7 mm. Az értékek között levő 3mm eltérés ebben a kis léptékben még nem okoz gondot, de így az állomány nem használható precíz modellezésre. Az importált DWG állományból mért távolság az ábrán látható helyen megegyezik az Autocad-ben mért távolsággal.



7. ábra - DWG állományból kinyerhető adatok (saját ábra)

A következő lényeges eltérés a két állomány között, hogy DWG állomány esetén lehetőség van a fájlban tárolt kiemelt tulajdonságainak vizsgálatára. Ezek a tulajdonságok az elem típusa; blokk neve, amennyiben az elem nem része egy blokknak, a paraméter értéke N/A; Fólia/Szint neve; az elem stílusa. Ezzel szemben a PDF állomány esetében nincs lehetőség a fájlban tárolt elemek vizsgálatára, így nem nyerhető ki belőle információ, csak vizuálisan. Az 7. ábra a DWG állomány egy elemének adatait mutatja. A kijelölt elem a „block1” nevű blokk része, a „Block name” sorban látható, hogy a blokk neve a rajz neve, majd a kiterjesztése, végül pedig a blokk neve ponttal elválasztva. A Dynamo-ban végzett adat kigyűjtésnél is így jelenik meg a tárolt név, azonban a folyamatot végző script úgy került megírásra, hogy a visszaadott névből törölje a fájl nevét és kiterjesztését, így megkönnyítve a későbbi folyamatokat.



8. ábra - DWG állományból kinyerhető adatok Dynamo-ban (saját ábra)

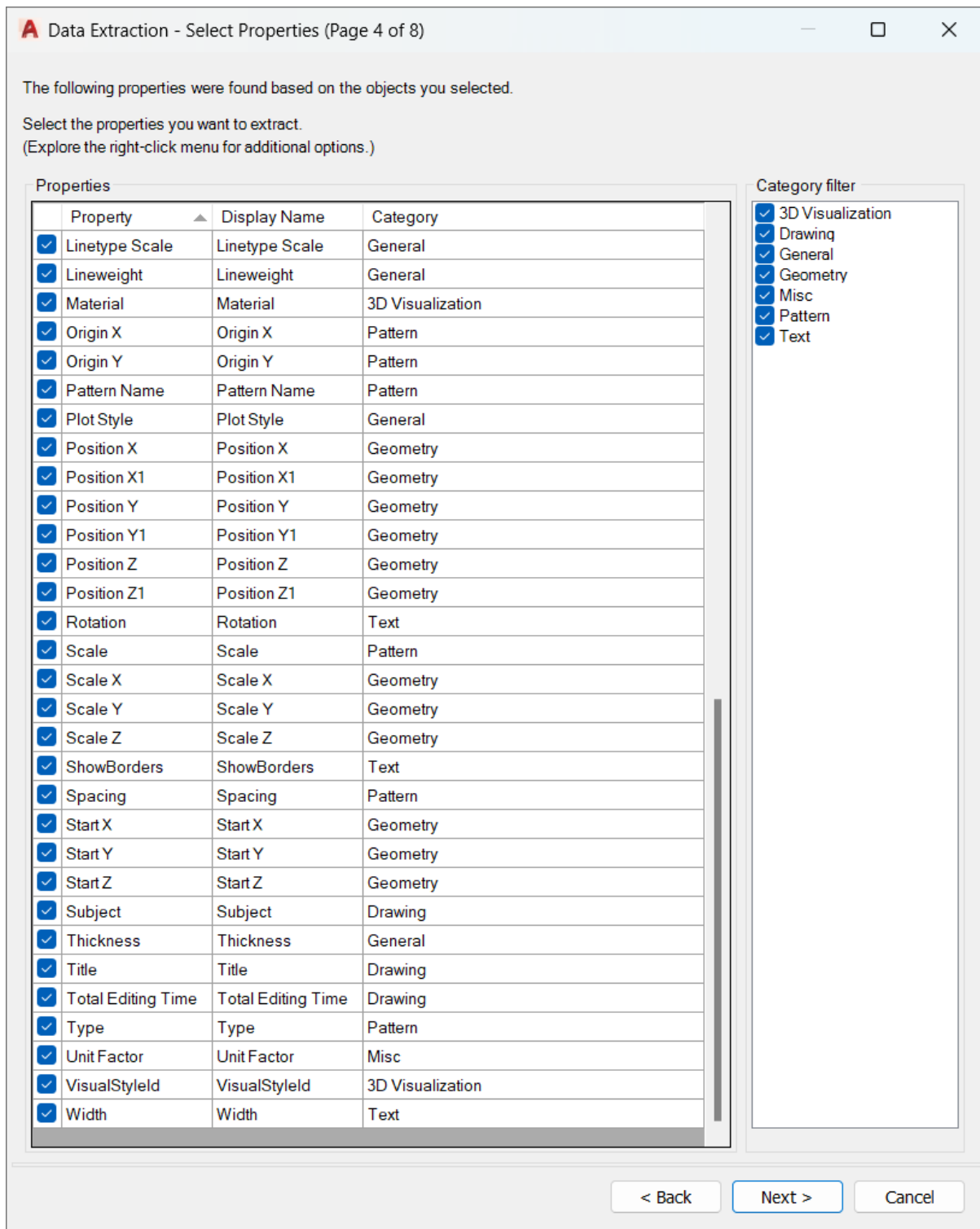
Az 8. ábrán látható a Dynamo-ban végzett adatkigyűjtés eredménye. A folyamatot végző Python Script a blokkokban tárolt adatokat gyűjti ki, azonban ezen megegyeznek az előbbi ábrán látható adatokkal. Az első, „BlockName” Node-ban láthatóak a fájlban található blokkok nevei, melyből látható, hogy csak a blokkok nevei kerülnek kigyűjtésre. Ezt követően láthatóak a blokkok elhelyezési pontjai, mint Dynamo „Point” elemek, a blokkok elforgatási szöge, majd az elem fóliájának neve és maga a blokk elem. A fólianevek kigyűjtésénél látható, hogy vannak olyan sorok, melyek értéke „null”. Ez egy érdekes jelenség, többszöri tesztelés után sem tudtam meghatározni, hogy miért nem képes kiolvasni a blokkból a tárolt fóliát. A következő pontban jobban kifejtem, hogy miért nem használtam a fóliák neveit az elemazonosítás vagy a további műveletek során. A fejezetben kiemeltem, hogy a tárolt fóliák neveit nem használom, melynek az ábrán látható jelenség az oka.

A blokk elemek kigyűjtése itt azért fontos, mert a szobák körvonalainak feltérképezéséhez ki kell töltenünk az ajtónyílásokat. Abban az esetben, ha kigyűjtés nélkül, a DWG állomány geometriáját használom és az ajtók ajtólapjai nem tartalmazzák a módszertan részben említett „SOLID” típusú kitöltést, akkor a falban marad egy üres nyílás, mely problémát okoz a szoba feltérképezésénél. Az blokk elemek kigyűjtésével meghatározható a határoló dobozuk, amely adatai alapján készíthető egy felület, mely kitölti a falban található ajtónyílást. Jelenleg az algoritmus nincs felkészítve arra az esetre, ha az ajtótok nagyobb mint a falszerkezet. A tesztelés során olyan ajtó blokkokat készítettem, melyek nem generálnak ilyen hibát, azonban ez egy későbbi továbbfejlesztési lehetőség. Emellett további fejlesztési lehetőség, a DWG állomány geometriájának kigyűjtése után a keletkezett elemek szűrése, mert ezek tartalmazhatják az ajtó blokkokból kinyerhető geometriát is. Ez a geometria gondot okozhat, amennyiben az előbb említett vastagság közötti különbség is fennáll.

3.1.2 DWG-ből kinyerhető adatok Revit-ben és Dynamo-ban

Az előző pontban már említettem, hogy milyen információt lehet kinyerni az importált DWG állományból Revitben. A linkelt és az importált állományok között ilyen szempontból nincs különbség. Sajnos Dynamo-ban is csak az előző pontban felsorolt adatok gyűjthetők ki az állományból, azonban itt lehetőség van külső adatbázisok olvasására is, amellyel bővíthetők a rendelkezésre álló információk. Jelen kutatásban ezt a módszert alkalmaztam. Az importált DWG állományból kinyertem a geometriát, valamint a blokkok neveit, fóliát, elhelyezési pontjukat és az elforgatási szögüket is. Az így ki nyert adatokból csak a fóliák neveit nem tudtam felhasználni. Az importált állományban a voltak szövegdobozok is, melyek a helyiség nevét és számát tartalmazták. A szöveg dobozokból sajnos natívan nem tudtam adatot kinyerni,

így Autocad-ből „DATAEXTRACTION” parancssal kinyertem belőlük a helyiségek neveit és számait, továbbá a szövegdozók elhelyezési pontjait is. Utóbbi azért volt fontos, mert fel tudtam használni a helyiségek körvonalainak meghatározásánál.



9. ábra - Autocad DATAEXTRACTION parancssal kinyerhető adatok (saját ábra)

Az 9. ábrán láthatóak a „DATAEXTRACTION” parancssal Autocadben kinyerhető adatok. Megállapítható, hogy a natívan kinyerhető adatok mennyisége jelentősen meghaladja az

importálás után kinyerhető adatok mennyiségét, továbbá jelen módszerrel a DWG állományban tárolt minden elem minden adata kigyűjthető, ezért döntöttem a használata mellett.

	A
1	Contents,Position X,Position Y,Position Z
2	114\PWC,3579.9849,2214.4204,0.0000
3	115\PTusoló,2827.8153,2524.6609,0.0000
4	

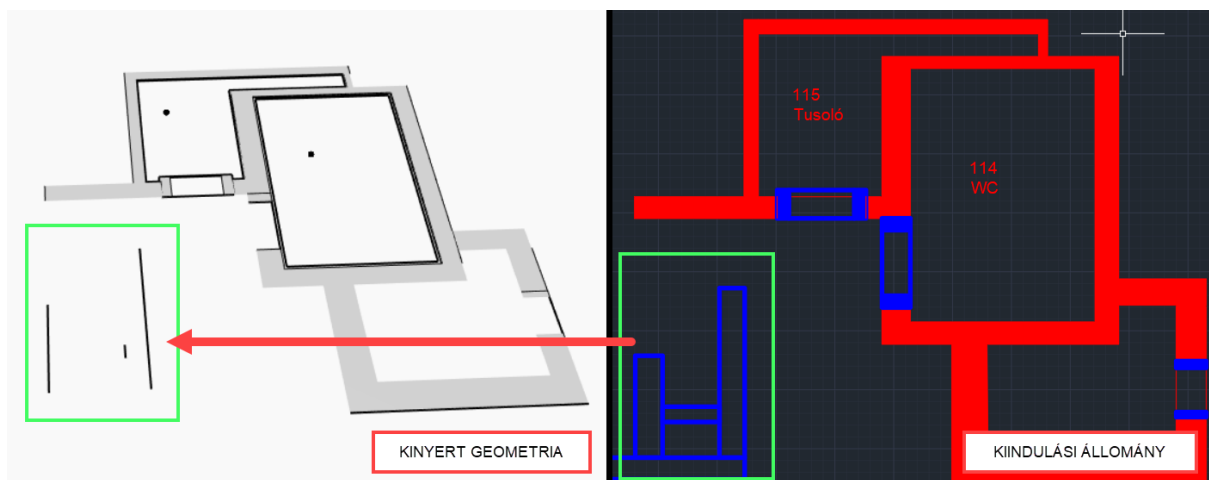
10. ábra - DATAEXTRACTION parancssal kinyert adatok (saját ábra)

Az 10. ábrán láthatóak az általam kigyűjtött adatok. Ezeket az adatokat CSV állományba exportáltam, mert kevesebb emberi hibafaktort tartalmaz a használata, mint XLS állomány esetén. Utóbbi állomány esetén a Dynamo-ba való importálás során meg kell adni az importálandó munkalap nevét is, mely véleményem szerint változhat abban az esetben, ha módosítást végzünk az állományon importálás előtt. CSV import esetén el tudjuk dönteni, hogy transzponálni szeretnénk-e az állományt, a műveletet végző Node-nak más információra nincs szüksége.

3.1.3 Szükséges információk meghatározása a modellezés és erőforrások optimalizálásához

A tesztelés során kipróbáltam többféle megközelítést is, így meg tudtam határozni a feltétlenül szükséges adatokat az algoritmus használatához és létrehozásához. Ezeket az adatokat már az előző pontban említettem, mint például geometria, blokkok neve, elhelyezési, elforgatási szög, helyiségek neve és száma. A tesztelés során használtam a fóliák neveit is, azonban nem találtam olyan felhasználási módot, ahol érdemlegesen használni tudtam volna őket. Ez abból fakad, hogy egy fólián több különböző elem is használatos, így nehéz volt a szükséges elemek szűrése, anélkül, hogy sok fóliát létre ne hoztam volna. Hatékonyabb megoldást nyújtott a blokkok nevének használata, mert minden blokk neve egyedi, így mind az előző pontban említett paranccsal, mind Dynamo-n belül, mind az előző pontban említett paranccsal könnyen szűrhetők ezek elhelyezési pontjai és forgatási szögei. ez a szűrhetőség különösen az ajtó blokkok esetében fontos, mert a fal megszakad az ajtó pozíciójában, így a helyiségek körvonalának meghatározásánál kellemetlenséget okozhatnak. Az elhelyezési pontok és az ajtók geometriájának adataival meghatározható az ajtó pozíciója, valamint befoglaló méretei, így a falban keletkezett nyílás kitölthető.

A tesztelés kezdetleges fázisaiban elsősorban vonalakat használtam, melyeket blokkosítottam. Ezek használata nem hozta meg a várt eredményt, mert a geometria kinyerése során a vonalakat, függetlenül attól, hogy vonal vagy vonallánc eszközzel kerültek létrehozásra, a Dynamo vonalláncokra bontotta fel őket, az össze metsző vonalláncok pedig hibát generáltak, így a program törölte őket. Ezt követően a tesztelés következő fázisában a létrehozott, zárt vonalláncokba tömör kitöltést helyeztem. A kitöltés típusát is többször teszteltem, a Dynamo csak a tömör kitöltésből tudta kinyerni a geometriát, a többi vonalas kitöltésből semmilyen információ kinyerésére nem képes. A megállapítás után csak tömör kitöltéseket, a kitöltéseket meghatározó vonalláncokat, valamint szövegdobozokat vonalláncokat használtam.



11. ábra - Geometria kinyerése DWG állományból Dynamoval (saját ábra)

Az 11. ábrán látható, hogy a DWG állományban tárolt különböző elemekből a Dynamo milyen geometriákat tud kinyerni. A kiindulási állományban található vonallánc, kitöltés, szövegdoboz és blokk elemek. Megfigyelhető, hogy a „SOLID”, vagyis tömör kitöltésekből felületeket készít, ezeket jelölik a bal oldali képen látható szürke sávok. A szövegdobozokból nem tud geometriát leképezni, a zölddel keretezett vonalláncokból pedig csak pár vonalat képes létrehozni. Emiatt a konverzió miatt nem lehet csak vonalakat vagy vonalláncokat használni a falszerkezetek, szobák és blokkok meghatározásához.

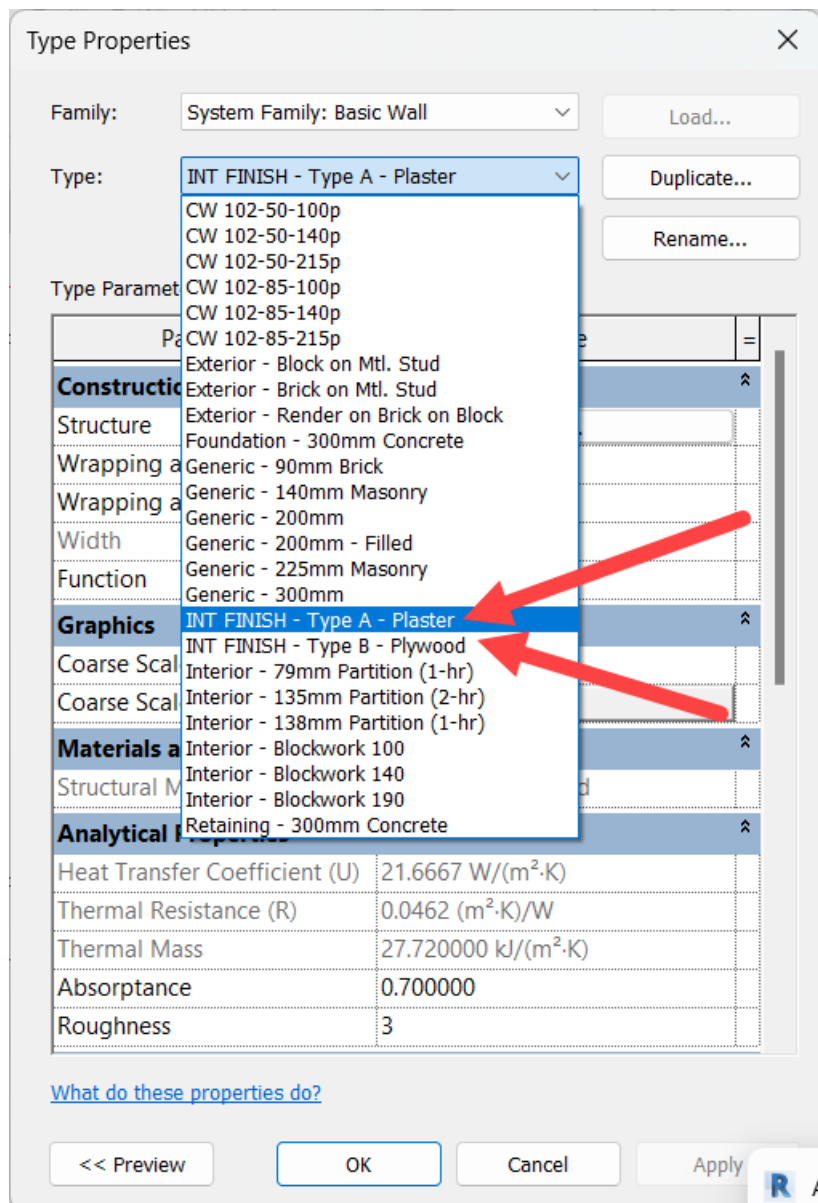
A DWG állomány tesztelési fázisának végén az állományom tömör kitöltésekből, blokkokból, továbbá a helyiségeket meghatározó szövegdobozból álltak. A helyiségek geometriájának szándékosan nem egyszerű négyszöget használtam, hogy általánosabban tudjam az algoritmus helyiség geometria meghatározó részét létrehozni. A DWG állomány linkelésén kívül létrehoztam egy Excel táblázatot is, mely tartalmazza a DWG állományban található helyiségek számait, neveit, valamint, a helyiségben használt falburkolatot is. A táblázatot azért hoztam létre, mert egy épület tervezési folyamata során szintén készítenek ilyet, így biztosan rendelkezésre fog állni. Azonban a helyiségben jellemző falburkolat megnevezése kinyerhető

lehetne a DWG állományban található szövegdobozból is, ugyanakkor abba ezt az információt nem szokás beleírni.

A	B	C	D	E
Szoba neve	Szoba szár	Falburkola	Falburkolat neve	
Szoba	113	Type B	OSB	
WC	114	Type A	Falfesték gipszkartonon	
Tusoló	115	Type B	OSB	
Előtér	116	Type A	Falfesték gipszkartonon	

12. ábra - Falburkolatokat tartalmazó CSV (saját ábra)

Az 12.ábrán látható szobák falburkolatainak adatait tartalmazó EXCEL táblázat. A szoba nevét és számát szándékosan írtam külön oszlopba, mert eddigi tapasztalataim alapján, a tervezők is külön szokták megadni ezeket az adatokat. A szövegdobozok pozícióival való összefésülés során az ábrán látható táblázatból a szoba szám szolgál a keresés alapjául. Ennek oka, hogy a szobaszám minden helyiségre nézve egyedi, míg a szoba neve ismétlődhet abban az esetben, ha például az épületben több fürdőszoba vagy hálósoba található és a tervező „Hálósoba” vagy „Fürdőszoba” elnevezést ad a helyiségeknek, nem pedig „Hálósoba 1”, „Hálósoba 2”. A vizsgálathoz szükséges következő fontos adat a falburkolat típusa. Ez látható a harmadik oszlopban, „Type A” és „Type B” típusokkal megadva.



13. ábra - Revit-ben létrehozott falburkolatok (saját ábra)

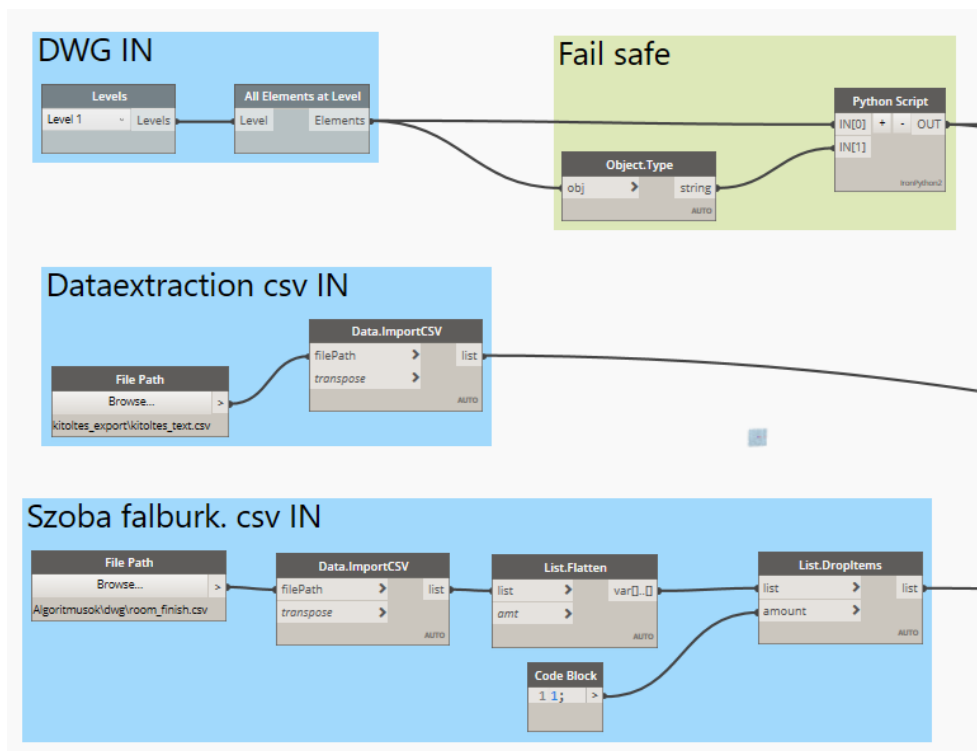
Az 13. ábrán látható a Revit-ben létrehozott falburkolatok nevei. Az elnevezésnél, eddigi tapasztalataim alapján, először megadtam az elem típusát, ezt jelenti az „INT FINISH”, mint beltéri falburkolat (Interior Finish), ezt követően megadtam a falburkolat típusát, „Type A”, majd a külső burkolat anyagát, mint „Plaster” vagy „Plywood” (vakolat, rétegelt lemez).

3.1.4 Létrehozott algoritmus

3.1.4.1. Kiindulási állomány

Az előző pontban meghatározott szükséges információk alapján a kiindulási állományom egy DWG állomány, valamint két táblázat. Előbbi tartalmazza a belső válaszfalakat, ajtókat, továbbá a helyiségek nevét és számait szövegdoboz formájában, melyek a megfelelő

helyiségekben vannak elhelyezve. Az egyik táblázat tartalmazza a helyiség nevét, számát és jellemző falburkolatát, a másik táblázat Autocad-ből került kinyerése, tartalmazza a dobozok pozícióját és tartalmát.



14. ábra - DWG és CSV állományokat importálásának folyamata (saját ábra)

Az 14. ábrán látható a bemeneti állományok feldolgozására szolgáló Node-ok csoportosítva. A DWG állomány feldolgozásához a Revit-ben egy szintre lehelyezett elemeket vizsgáljuk, majd ebből kiválasztjuk azokat melyek típusának nevében benne van az „ImportInstance” kifejezés. Ez a kifejezés utal az importált állományra.

A CSV állományok importálásához a fájlok elérési útvonalát kellett megadni, majd a Node-ot mely az állomány olvasását végzi. Ezt követően a falburkolatokat tartalmazó állományból kiszűröm azt a sort, mely az oszlopok neveit tartalmazza, mert erre a továbbiakban nincs szükség. Fejlesztési irány lehet ezen adatok kezelése is arra az esetre, ha később ki kellene bővíteni a CSV állományokat.

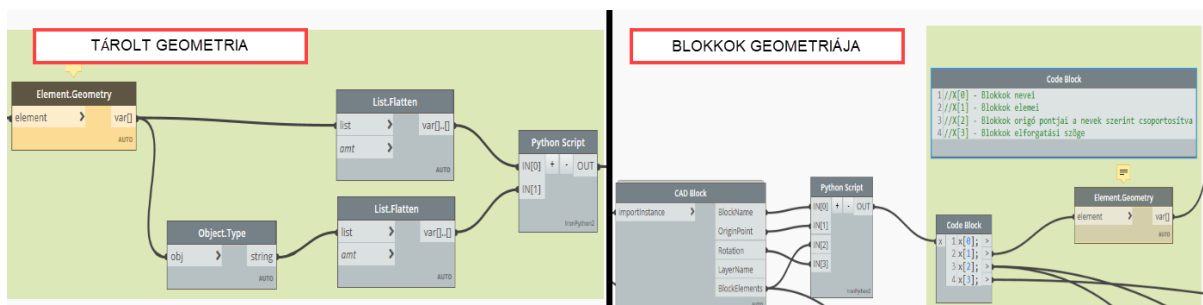
3.1.4.2 Táblázatok átalakítása

A táblázatokat CSV formátumban importáltam Dynamo-ba. A fájlban tárolt adatok soronként lettek listába rendezve, így az 1. sor, amely az oszlopok megnevezését tartalmazta, törlése egyszerű volt. Az autó ki nyert táblázatot ezután újra rendeztem helyiségek szerint, majd a másik táblázattal összevetve kigyűjtöttem a szövegdobozok elhelyezési pontjait, valamint, a helyiségekben használandó falburkolatok típusait. A pontoknak létrehoztam Dynamo pontokat

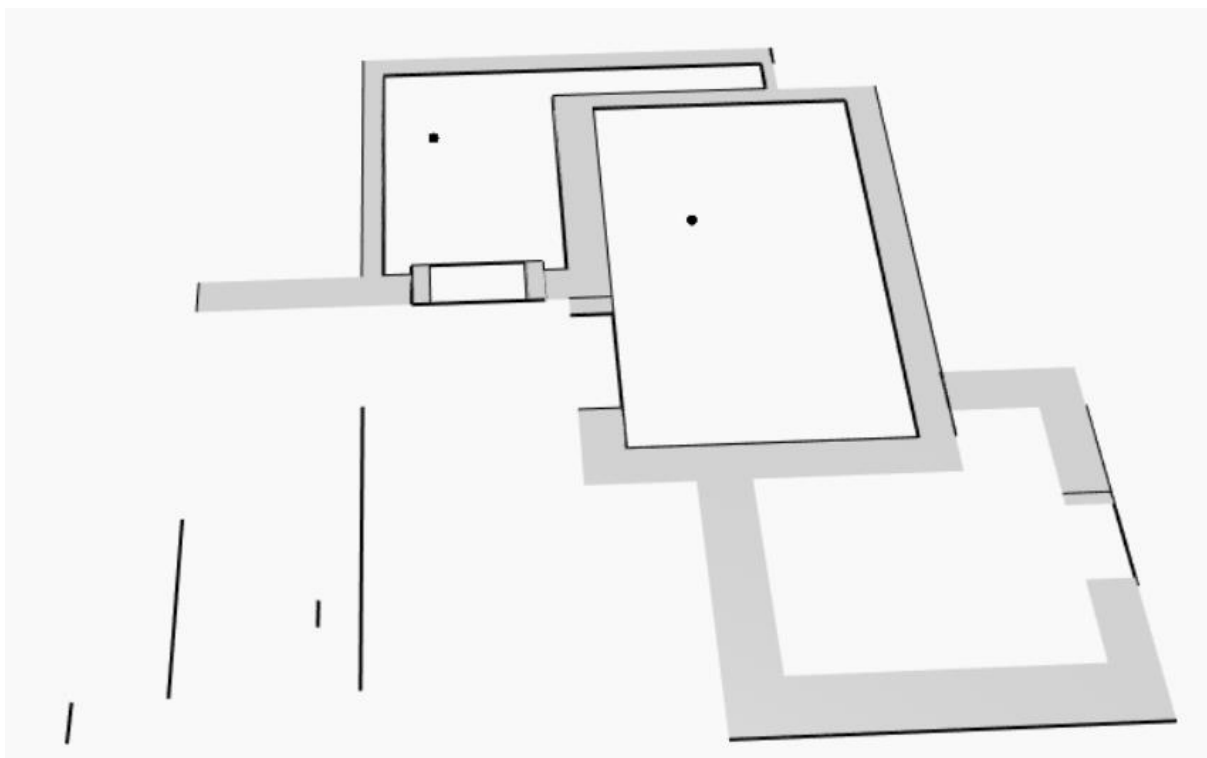
is a későbbi műveletekhez. Az így létrehozott elemeket és ki nyert burkolatokat a helyiségek geometriájának meghatározásánál valamint a falburkolatok elhelyezésénél használtam.

3.1.4.3 DWG átalakítás

A DWG állomány több féle módon is importálható Dynamo-ba, én egy olyan módszert választottam, ahol esetlegesen több DWG is használható egyszerre. Az állomány adatainak kinyerése előtt beépítettem egy biztonsági funkciót, menj kiszűr minden elemet, ami nem importált. Ezt követően két vonalon indultam, az egyik a tárolt blokkok adatainak gyűjtése, melyből meg tudom határozni az ajtók pozícióját valamint geometriáját, a másik a belső falak meghatározása.



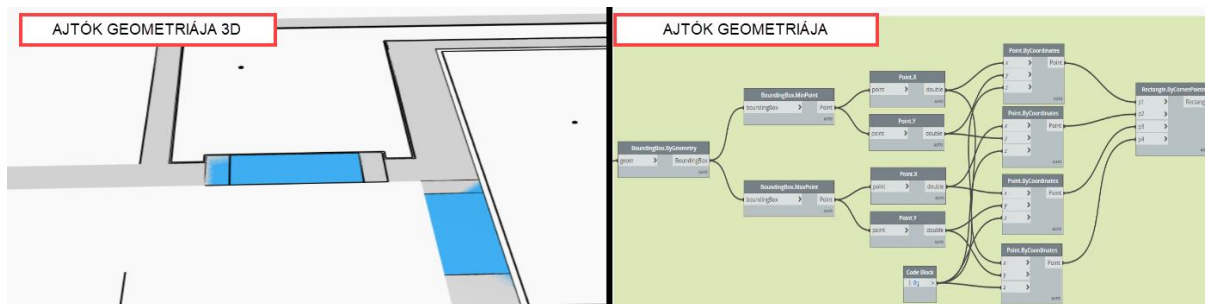
15. ábra - Geometria kinyerését végző Node-ok (saját ábra)



16. ábra - DWG állományból kinyert geometria (saját ábra)

A 15. ábrán látható a blokkok adatainak és a DWG állományban tárolt geometriák kinyerését végző node-ok. Megfigyelhető, hogy az „Element.Geometry” nevű Node színe más a többihez képest. A jelenség oka, hogy bizonyos elemek esetén a Node hibát generál. A hibát a kitöltések meghatározásához használt vonallánc elemek okozzák, melyet a 3.1.3. pontban már kifejtettem. A 16. ábrán látható a kinyert geometria, melyen megfigyelhető, hogy egy ajtó szélesebb, mint a falszerkezet, melyből a 17. és 18. ábrán látható hiba keletkezik.

Az ajtók gyűjtésének egyedül a fólia nevet nem használtam. Írtam egy Python Scriptet ami a fájlban tárolt blokkok közül ki gyűjti azokat, melyek nevében szerepel az ajtó szó, majd az így meghatározott lista alapján sorba rendezi a kigyűjtött ajtókhoz tartozó pontokat és geometriákat. A geometriából meghatározza a befoglaló dobozt melyből egy felületet hoz létre. Az ajtók elhelyezési pontjait az ajtó egyik oldali központjaként határoztam meg, azonban a blokkokban tárolt geometria kinyerése akkor az elhelyezési pont a program origójába kerül, így ezeket el kell mozgatni a kigyűjtött elhelyezési pontokba, valamint el kell forgatni a tárolt szöggel.

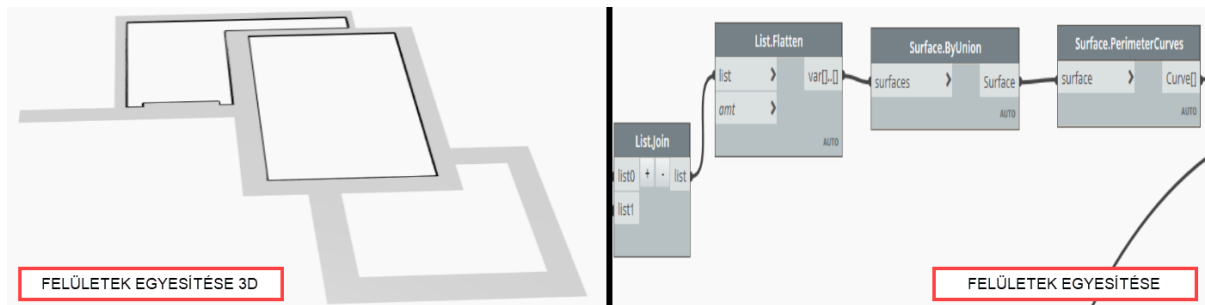


17. ábra - Ajtók határoló dobozának meghatározása (saját ábra)

A 17. ábrán látható az ajtó blokkok geometriájából készített határoló doboz kékkel jelölt felülete. Megfigyelhető, hogy a folyamatot végző Node lánc sikeresen kitöltötte a falban található üres ajtónyílást. Emelett látható, hogy mindkét ajtó esetében hiányosnak tűnik a kék felület. A jelenség kiváltó oka, hogy ugyan ebben a pozícióban, a DWG állományból kinyert geometria tartalmazza az ajtótokokat, melyek itt metsződnek és az ábrán látható módon jelennek meg. A DWG geometriák szűrésével ez a jelenség megszüntethető, ezt a folyamatot meghatároztam mint lehetséges fejlesztési irány.

A belső falak meghatározásához kinyertem az állományban tárolt geometriát, melyet egyesítettem az ajtók előbbiekben leírt műveleteinek eredményével, így kaptam egy nagy felületet, mely meghatározza a belső válaszfalakat ajtónyílás nélkül. Ez a módszer nagy mértékben Megkönnyíti a helyiségek geometriájának kinyerését, mert így mindenképpen helyiség körvonala zárt. A tesztelés során próbáltam az ajtók helyettesítése nélkül megtalálni a

helyiségek kontúrjait, azonban ez egy hosszadalmas, erőforrás igényes folyamat volt, mely sok hibalehetőséget tartalmazott.

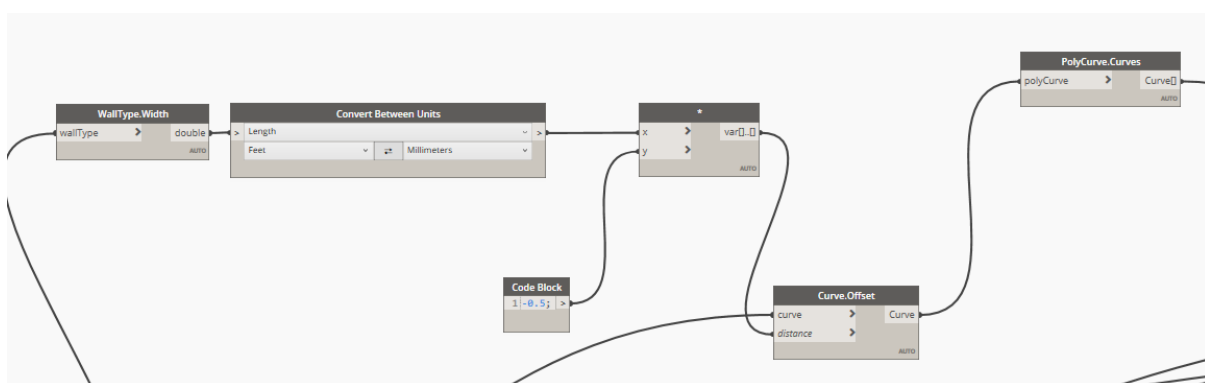


18. ábra - Kinyert és létrehozott geometriák egyesítése (saját ábra)

A 18. ábrán láthatóak a kinyert felületek egyesített felülete, valamint a szövegdobozok pontjai alapján kinyert helyiségek kontúrvonalai is (fekete vonalak). Az ábrán megfigyelhető, hogy abban a pozícióban, ahol szélesebb volt az ajtótok mint a fal, az algoritmus lekövette a geometriát, így ez pontatlan megoldáshoz vezet. Ennek a jelenségnek a kiküszöbölését is kitűztem egy jövőbeni fejlesztési iránynak.

A felületek egyesítése után a szövegdobozok pontjaihoz képest megkerestem a legközelebbi a zárt kontúrt, így megkaptam a helyiség körvonalát. Revit-ben létrehoztam fal burkolatokat, melyeknek nevében megtalálható a helyiségekhez rendelt típus. Miután megvolt a helyiséghez tartozó körvonal valamint burkolat típus már csak létre kellett hozni a falburkolatok elemeit.

Az elemeket először a helyiség kontúr, mint referencia vonal mentén hoztam létre, azonban alapértelmezetten ez a vonal a fal középvonalának felelt meg, így a falakat el kellett tolni, hogy ne ütközzenek a válaszfal szerkezetével. Az eltolásnál érdekes módon hiába adtam meg neki az eltolás mértékét, az elemek nem a megfelelő helyre érkeztek.

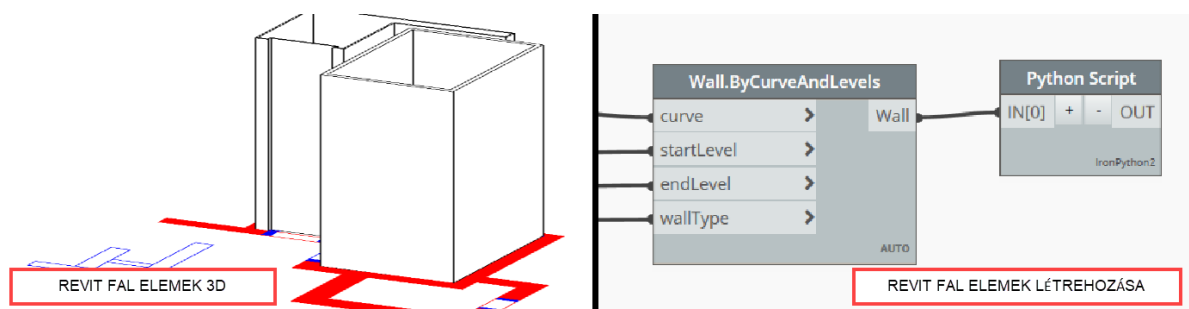


19. ábra - Falak referencia vonalának eltolása és mértékegység váltás (saját ábra)

A 19. ábrán látható mértékegység váltó, majd referencia vonalat eltoló folyamatot végző Node lánc. A mért értékek milliméterre váltása után megszorozzuk -0.5-el. A negatív előjel miatt a szoba középpontja felé toljuk a referenciavonalat, valamint a falvastagság felével kell végeznünk a tolást. A felezés oka, hogy a fal létrehozását végző Node alapértelmezetten a fal

középvonalát tekinti referenciavonalnak, valamint ezt csak a elhelyezés után lehet megváltoztatni.

Kiseb kutatás után megállapítottam, hogy a Revit projekt és a Dynamo által használt mértékegységek különböznek. A Dynamo mértékegysége alapvetően milliméter, melyet nem lehet megváltoztatni, azonban a Revit projekt angolszász mértékegységeket használ, annak ellenére, hogy a beállított projekt mértékegység milliméter. Ez abból fakad, hogy a Revit alapvetően lábban és hüvelykben számol, a projekt mértékegységének változtatásával csak átalakítja az így kapott méreteket milliméterben. Emiatt az algoritmusba egy mértékegység átváltót is alkalmazni kellett. Az eltolást követően létrehoztam a falakat, melyekben utolsó lépésként megváltoztattam az elemek referencia vonalának helyét, hogy a fal középvonala helyett a falszerkezet felőli oldalon legyen. Erre azért volt szükség, mert így ha változik a falburkolat, akkor a vastagsága a helyiség felőli oldalon fog változni, így gyorsabban elvégezhető a burkolatok cserélje a modellben.



20. ábra - Falakat létrehozó Node-ok és a létrehozott falak 3D nézetben (saját ábra)

3.2 Geodéziai felmérés - 2D DWG -> 3D terep

Az előző algoritmussal ellentétben itt nem importálás után közvetlenül, hanem az állomány rajzi elemekre bontása után, a pontokat meghatározó vonalakból és szövegdobozokból nyerjük ki az információt. Ehhez két manuális Node-ot használtam, ami az 20. ábrán látható. Az egyik Node a vonalakat gyűjti, a másik a szövegdobozok azonosítóját, majd utóbbiból egy Python Scripttel kinyerjük a tárolt szöveget és a szövegdoboz bal alsó sarkát. A szöveget a pontok magassági koordinátájaként, a sarokpontot pedig a szövegdobozok elhelyezkedésének vizsgálatához használtam.

3.2.0 Módszertan

Jelen kutatásban vizsgáltam egy geodéziai felmérés során készített DWG állomány algoritmizálhatóságát is. A geodéziai felmérés során a földmérő felméri a telken található tereptárgyak elhelyezkedését, a telek méretét és lejtésviszonyait, igény esetén a szomszédos

épületek ereszmagasságait is felmérheti. A felmért állománnyal további műveleteket végez CAD szoftverben, végül az eredmény lehet egy szintvonalas térkép vagy tervezési térkép is, mely a lejtésviszonyokat ábrázolja. A vizsgált állomány egy szintvonalas térkép volt, mely tartalmazta a felmérési pontokat a hozzájuk tartozó magassággal együtt.

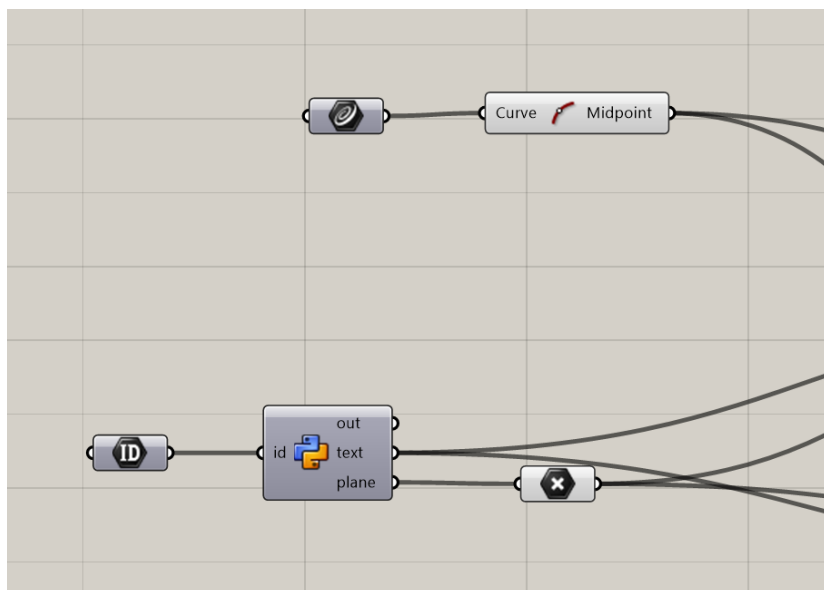
A kiindulási állomány egy 2D DWG, amelyben megtalálhatóak a mérési pontok a mért magassággal, X-el jelölve. A feldolgozáshoz használt szoftver Rhinoceros 7, továbbá a Grasshopper nevű bővítménye. A DWG állományt Rhino-ba importálása után rajzi elemekre bontottam a könnyebb használhatóság miatt. Ezt követően Grasshopperrel végeztem a további műveleteket.

3.2.1 DWG import és felhasználás

Első lépésként importáltam a felmért pontokból létrehozott DWG állományt, majd rajzi elemekre bontottam. Az állományból sajnos natívan nem tudtam adatokat kinyerni, ezért volt szükség a rajzi elemekre bontásra. A Rhino-ban további műveletet nem végeztem, a Grasshopper nevű bővítményben lehelyeztem két Node-ot. Az egyik elem fogadta a pontokat jelölő vonalakat, másnéven „curve” elemeket, a másik pedig a szövegdobozok adatait tartalmazta. Ezek sajnos nem tudták automatikusan kinyerni a megfelelő elemeket a Rhino munkamenetéből, így manuálisan kellett őket hozzáadni. Ezt követően a két elemtípussal egymástól függetlenül kezdtem el műveleteket végezni, majd az egymáshoz viszonyított pozíciók vizsgálatánál használtam őket egyszerre.

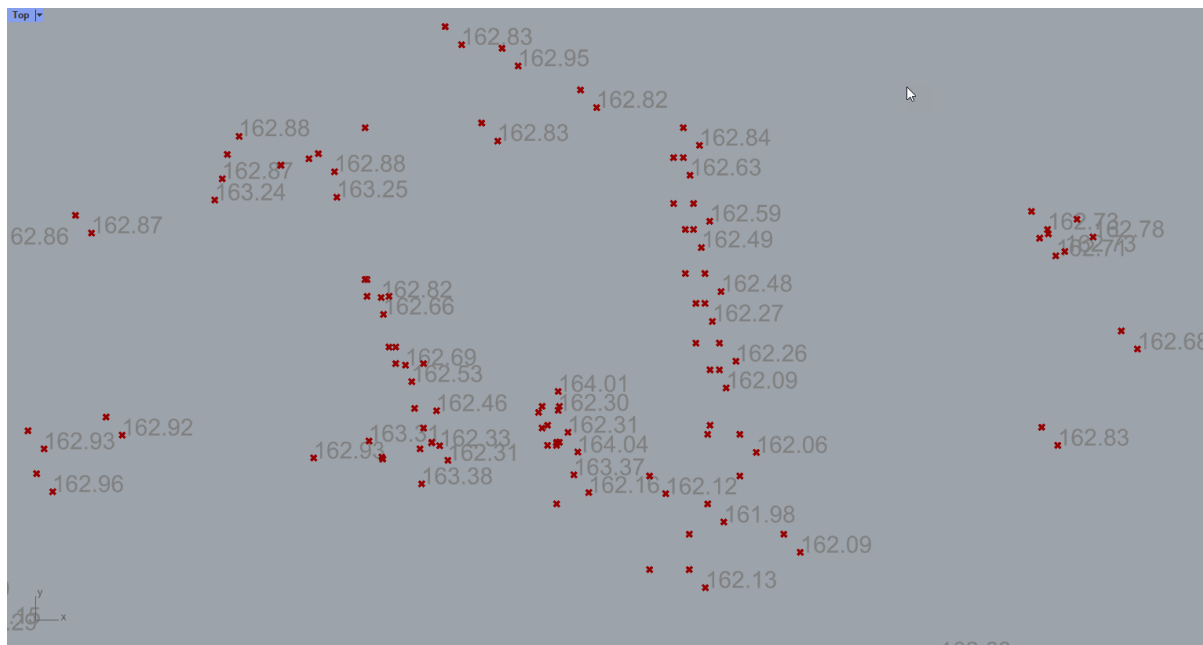
A vonalakból ki kellett nyernem az általuk meghatározott pont X és Y koordinátáját. Ehhez a művelethez egyszerűen a vonalak középpontját kerestem, melyhez egy beépített Node-ot használtam.

A szövegdobozoknak az egyedi azonosítóját kellett tárolnom, mert így kinyerhettem belőlük a sarokpont mellett a tárolt szöveget is. Csak a geometria tárolásánál a tárolt információ elveszett volna. Ezt követően egy Python Script segítségével kinyertem a bennük tárolt magassági adatot, továbbá a szövegdobozok elhelyezési referencia síkot. Az elhelyezési referencia sík jelenti azt a síkot melyre a szövegdoboz le lett helyezve. Ez a sík meghatároz egy új koordináta rendszert, amely az elemhez tartozik, origó pontja a szövegdoboz bal alsó sarka. Ezekből a típusú elemekből más geometriai adatot nem tudtam kinyerni, így a kinyert sík origóját használtam a pozíció vizsgálatához. Ehhez egy „Point” Node-ot használtam, mely kinyeri a sík origóját.



21. ábra - DWG állományból kinyert rajzi elemek és szövegdobozok importja Grasshopper-be (saját ábra)

Az 21. ábrán a leírt munkafolyamat látható. A kép felső részén látható a vonalakat tartalmazó, valamint a központjukat kinyerő Nodeok, az alsó részen az „ID” felirattal ellátott Node tartalmazza a szövegdobozok azonosítóját. A következő a Python Scriptet tartalmazó Node, továbbá a síkok középpontját kinyerő elem.



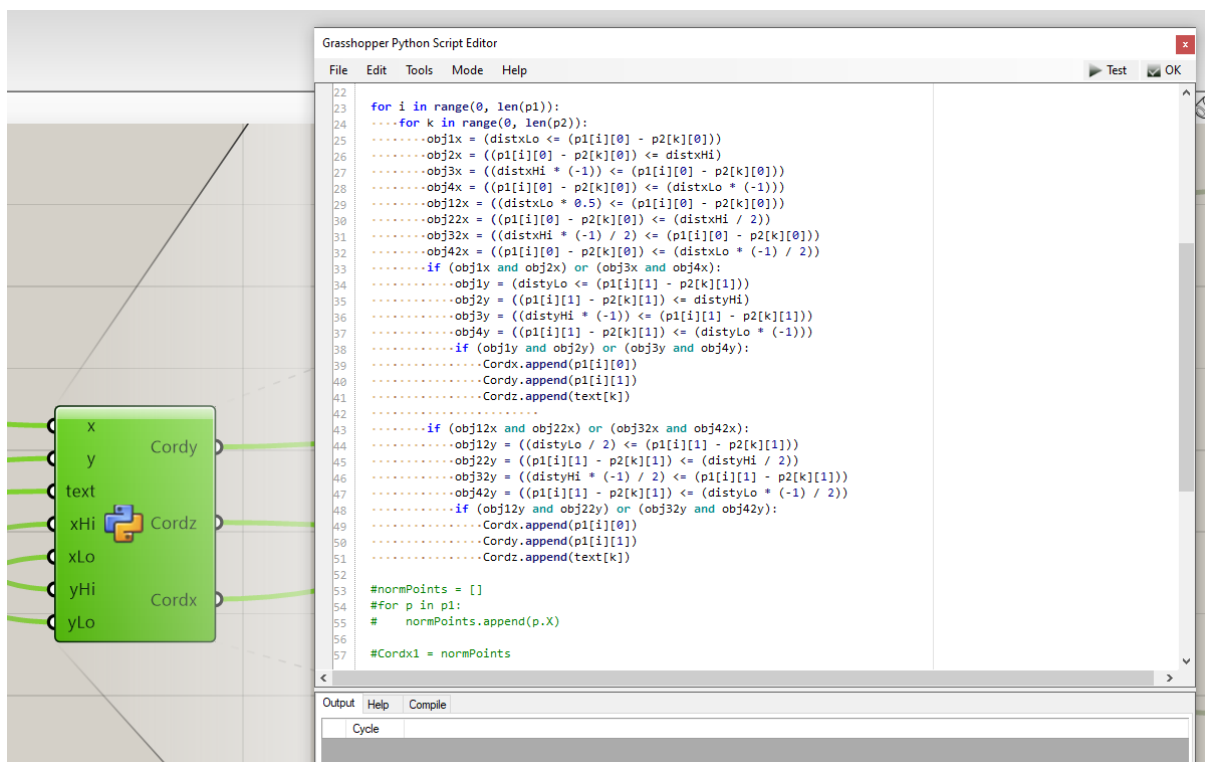
22. ábra - Rajzi elemekből létrehozott pontok, a pontokhoz tartozó magassági adatokkal (saját ábra)

Az 22. ábrán látható a kinyerés eredménye, a számok mutatják a szövegdobozokat, a piros x-ek pedig a kinyert pontokat. A pontok között megtalálhatóak mind a szövegdoboz azokból kinyert pontok, mind a vonalak által meghatározott felmérési pontok. Az ábráról leolvasható,

hogy vannak az állományban olyan helyek, ahol nagy a pontsűrűség, az ábra jobb oldalán és felső részén megfigyelhető, hogy a pontokhoz képest a szövegdozok nagyjából azonos távolságra és irányban helyezkednek el. n ez az észrevétel fog kiindulási alapként szolgálni a pontokhoz tartozó szövegdozok pozíciójának vizsgálatánál.

3.2.2 Szövegek Pozíciójának vizsgálata a pontokhoz képest

Mint már az előző pontban említettem, megfigyelhető, hogy a szövegdozok pozíciója a pontokhoz képest konzekvensen a ponttól jobbra, lefelé átlósan található. Több méréssel megállapítottam, hogy a megfigyelt távolság és irány bizonyos értékek között változik. Az így megfigyelt eltérések oka, hogy a szövegdozok manuálisan lettek le helyezve a jelölt magassághoz tartozó pont vagy pontok mellé, így nem határozható meg egy vektorral a távolság és az irány.



23. ábra - A pontok és szövegdozok kinyerését végző Python kód (saját ábra)

Az 23. ábrán az a Python Script látható, mely a szövegdozok pozícióját vizsgálja. A Node-nak hét bemeneti paramétere van, melyek a Node bal oldalán láthatóak. Az x és y paraméterek a felmérési pont X és y koordinátái, a „text” paraméter a szövegdozban tárolt magasság, a többi bemeneti változó pedig a vizsgált x és y koordináták alsó és felső tűréshatárát határozzák meg. A jobb oldalon látható maga a Script, amely a vizsgálatot végzi. Az állományban 2754 darab felmérési pont található, ugyanakkor csak 2732 darab szövegdoz. Ebből az a következtetés vonható le, hogy előfordul olyan eset, amikor egy magassági érték több ponthoz

is tartozik. Ezt sajnos a Script nem tudja szűrni, mert ezeknél a különleges eseteknél, nem fedezhető fel olyan általános logika, adj a ponthoz képest milyen irányban és távolságban kell keresni a hozzátartozó szövegdobozt. Az algoritmus jelen állása szerint, ahhoz hogy megtalálja a szövegdobozhoz képest meghatározott tartományban található pontot, át kell vizsgálnia az állományban található összes pontot, minden pontra nézve, ez körülbelül 7 millió vizsgálatot jelent ennél a pontmennyiségnél. A talált pontokat végül három listába rendezi, mert külön kerülnek tárolásra az x, y és z koordináták.

0	17090.417461	
1	14161.417461	
2	14663.417461	
3	13891.417461	
4	-190.582539	
5	-4653.582539	
6	-5950.582539	
7	-6321.582539	
8	-6285.582539	
9	-5852.582539	
10	-4096.582539	
11	500.417461	

		{0;0}
0	13314.291626	
1	14053.291626	
2	15508.291626	
3	17354.291626	
4	29907.291626	
5	35000.291626	
6	35910.291626	
7	36513.291626	
8	37278.291626	
9	37647.291626	
10	37747.291626	
11	36590.291626	
12	34916.291626	
13	22608.291626	
14	22823.291626	
15	22211.291626	
16	21910.291626	
17	25571.291626	

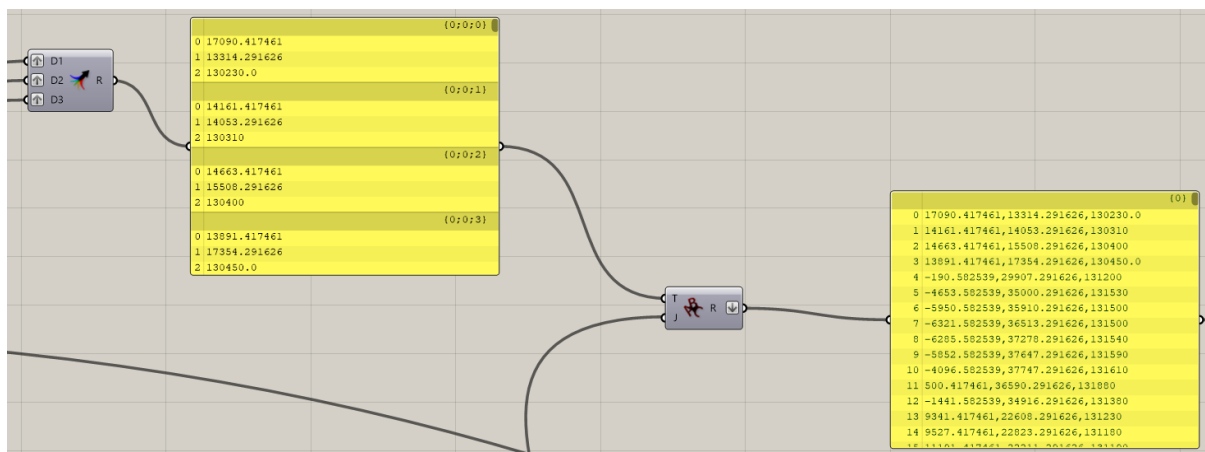
		{0;0}
0	130.23	
1	130.31	
2	130.40	
3	130.45	
4	131.20	
5	131.53	
6	131.50	
7	131.50	
8	131.54	
9	131.59	
10	131.61	
11	131.88	
12	131.38	
13	131.23	
14	131.18	
15	131.19	
16	130.73	
17	131.86	

24. ábra - A kinyert adatok (saját ábra)

Az 24. ábrán látható a vizsgálat eredménye, a három kimeneti lista, fentről lefelé haladva az x, y majd z koordináták.

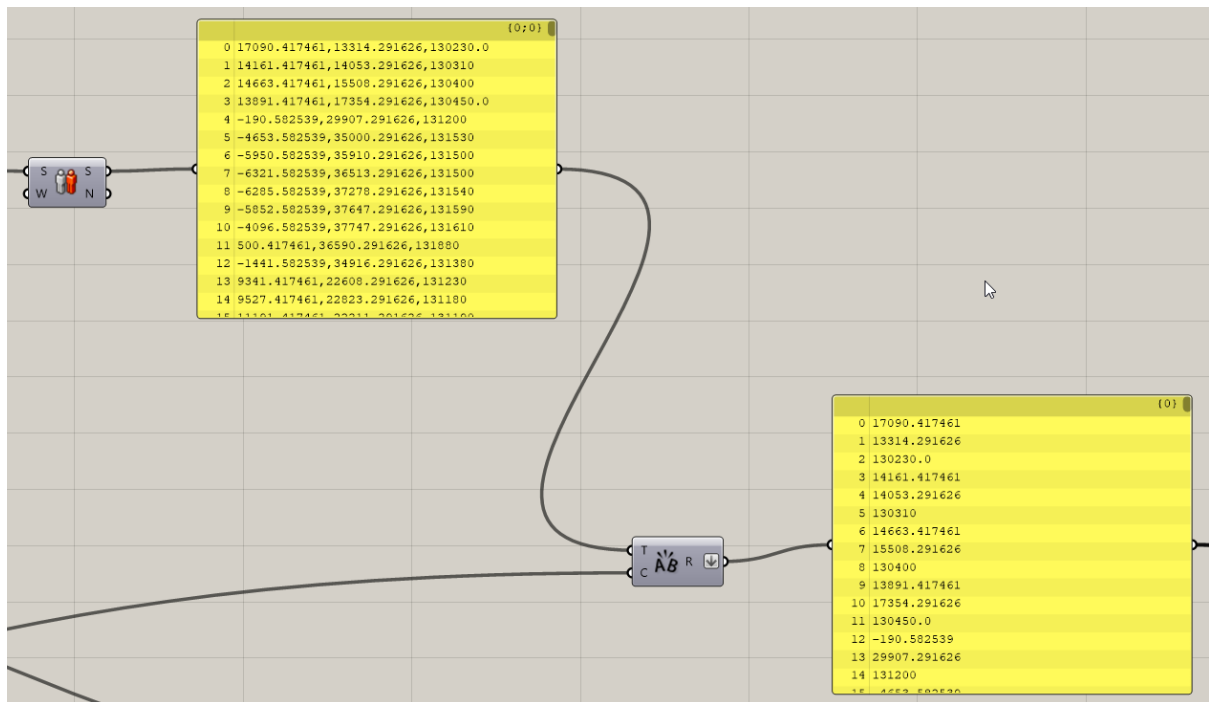
3.2.3 Magasság olvasása, pontok emelése

A következő lépés a pontok létrehozása, az 24. ábrán látható adatokból. Először ezeket az adatokat egy olyan Node-ba továbbítottam, melyik képes a koordinátákon pontokat létrehozni. A Node lehelyezése után teszteltem az algoritmust, arra lettem figyelmes, hogy nagy mértékben megnőtt a futási idő. Miután folyamat lejátszódott, látható volt, hogy valamilyen logika alapján a pontokat létrehozó Node összekeverte a bevitt adatokat, a 2732 pont helyett több mint 1 000 000 pontot hozott létre.



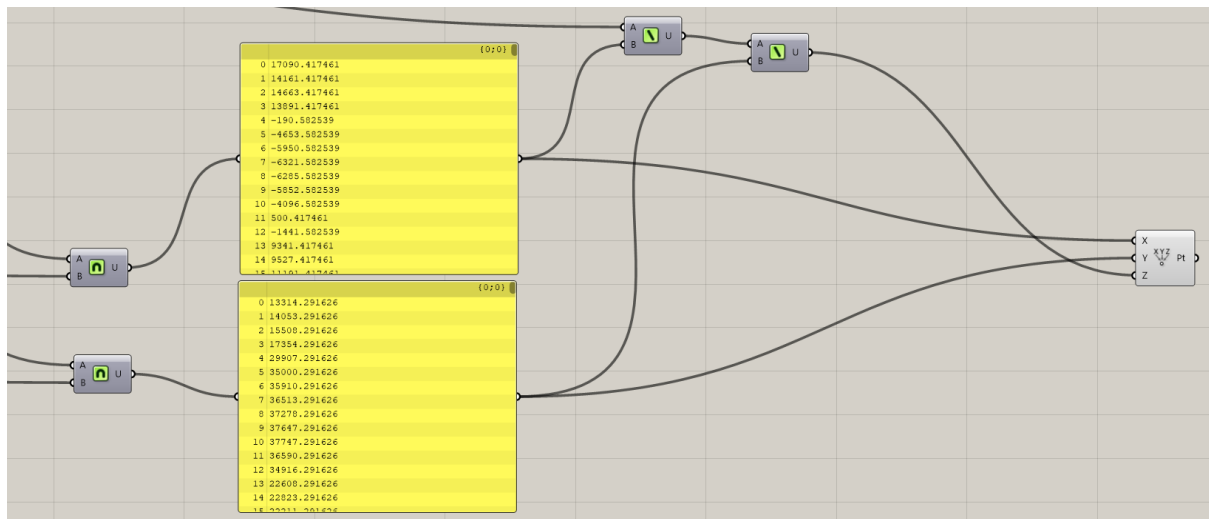
25. ábra - Koordináták csoportosítása, szűrése és rendezése (saját ábra)

Miután nem találtam meg a hiba okát, új megközelítést alkalmaztam. Megállapítottam, hogy a hiba egyik kiváltó oka az lehetett, hogy a magassági pontok nem milliméterben, hanem tengerszint feletti magasságban, méterben kerültek megadásra. Ezt követően átváltottam a magassági méretek mértékegységét, majd a koordináták listáit egyesítettem úgy, hogy egy listát alkossanak, ugyanakkor megmaradjon listák sorrendisége. Ehhez egy olyan Node-ot használtam, mely a bemeneti listákat egyesíti, de ugyanakkor hármasságokba rendezi őket, így a lista egy eleme tartalmazta a ponthoz tartozó minden koordinátát. Erre a műveletre azért volt szükség, mert a szövegdobozok és felmérési pontok száma eltért, ha így nem minden ponthoz tartozott magassági adat. Az eltérést az okozta, hogy volt olyan eset, ahol a szövegdobozoktól mért tartományokban a Script több pontot is talált, így minden talált pontot a kimeneti listához adtam, ugyanakkor csak egyszer adtam hozzá a magassági értéket. A rendezés után egy sorba rendeztem a koordinátákat, vesszővel elválasztva egymástól, ahogy az 25. ábra jobb oldalán látható.



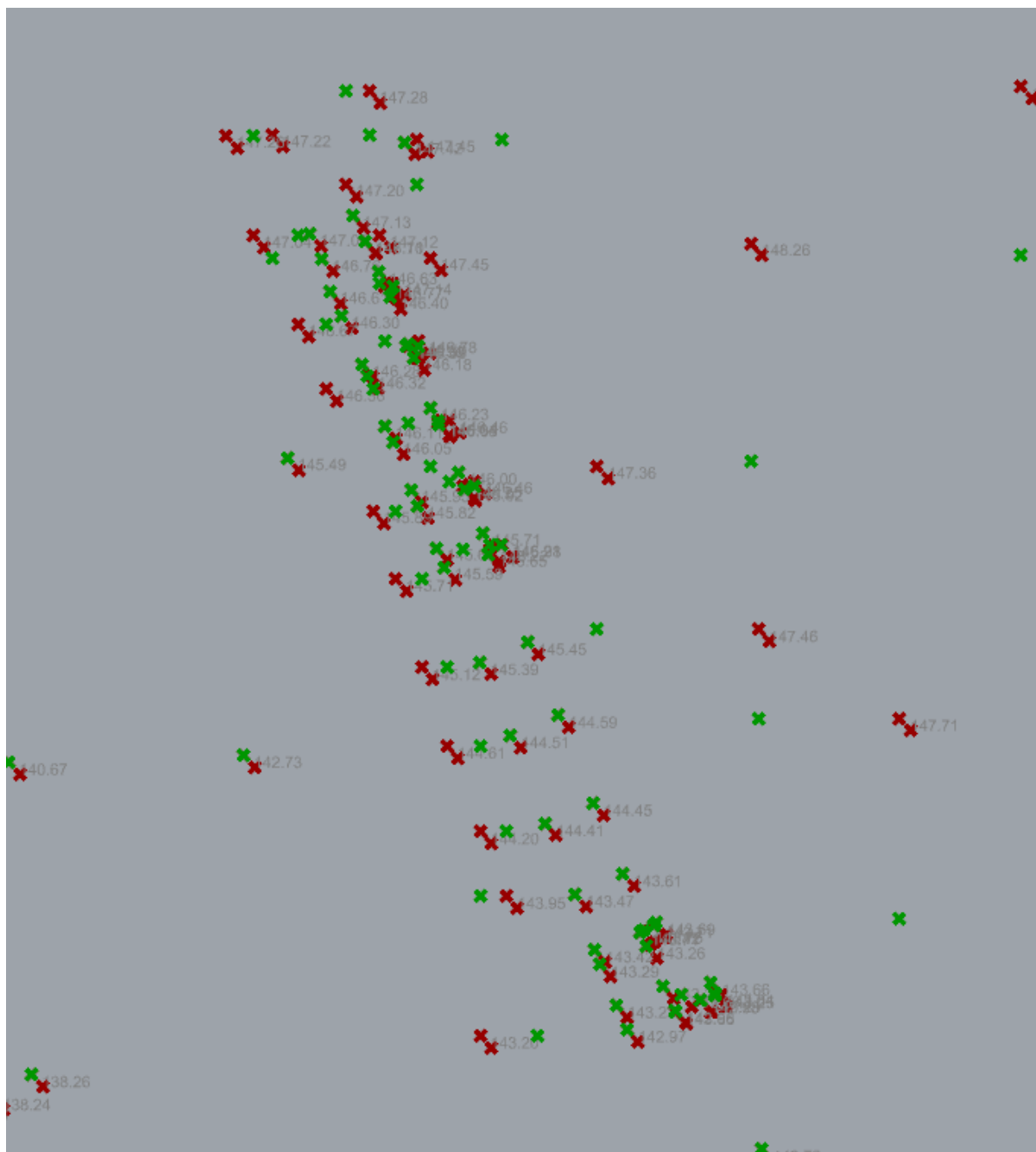
26. ábra - Szűrés után a koordináták listájának darabolása (saját ábra)

A rendezés után futtattam egy biztonsági vizsgálatot, mely a duplikátum szűrését és törlését tartalmazta. Az így kapott állományt a vesszők szerint szétbontottam, majd halmaz műveletekkel újra három listát alkottam belőlük. Az 26. ábra bal oldalán látható a biztonsági szűrés, majd a kapott állomány, ezt követően pedig a lista sorainak vágása a vesszők helyén. A vágás után kapott állomány látható az ábra jobb oldalán.



27. ábra - Listák vágása és a megemelt pontok létrehozása (saját ábra)

Az 27. ábrán láthatóak a végrehajtott halmazműveletek, valamint a műveletekből keletkező állományok is. Miután újra létrejött a három lista, már csak létre kellett hozni a pontokat.



28. ábra - A megemelt pontok (zöld) és kinyert pontok (piros) (saját ábra)

Az 28. ábrán látható a kapott végeredmény. Az ábrán láthatóak a magasságot tartalmazó szövegdozok is, azok sarokpontjai, valamint zöld x szel jelölve a megemelt mérési pontok is. Az ábrán látható, hogy nem sikerült minden pontot megemelni. Ugyanakkor meg kell említeni, hogy amennyiben ezt a műveletet egy embernek manuálisan kellett volna elvégeznie, az körülbelül 2-3 heti munkát vett volna igénybe, míg az algoritmus elkészítése és tesztelése a csupán pár órát vett igénybe. Emellett meg kell említeni azt is, hogy egy ilyen állomány feldolgozása építészeti feladat, az algoritmus elkészítéséhez pedig programozási ismeretek is szükségesek. A kutatás elkészítésének időpontjában a legjobb tudásom szerint

Magyarországon nem található olyan építész képzés, amely magában foglalná az algoritmusban használt Python Scriptek elkészítéséhez szükséges programozói tudás átadását.



29. ábra - A megemelt és kiindulási pontok 3D nézete (saját ábra)

Az 29. ábrán látható egy Rhino-ból kiemelt előnézeti kép, amelyen jól látható, hogy a kiindulási állomány egy 2D elemeket tartalmazó állomány volt, az algoritmus pedig 3D elemeket generált belőlük.

4. Összegzés

A szakirodalmi kutatás készítésekor áttekintésre került több a DWG felhasználására irányuló cikk, kutatás is, azonban ezek többsége elméleti szinten foglalkozik a témával, és nem írnak gyakorlati megvalósításról. Ugyanakkor találtam olyan kisebb, ázsiai vállalkozásokat, amelyek algoritmusok készítésével foglalkoznak, azonban profiljukat vizsgálva nem a szoftverek és állományok közötti átjárhatóság javítása a céljuk. Kutatásom során feltártam az átjárhatóság javítását szolgáló, a kutatási cél elérését segítő lehetőségeket, azonban ezek használata programozói tudást, valamint a programok széleskörű ismeretét igényelték. A kutatásom során használt szoftvereket a magyarországi építész képzésekben, legjobb tudásom szerint nem oktatják, akár kis mennyiségű programozói tudással is több időigényes, manuális munkafolyamat algoritmizálható, automatizálható.

Az első, jelen kutatásban bemutatott algoritmusom bemeneti paraméterei, mint az ajtó blokkok nevei vagy falburkolatok típusainak elnevezései jelenleg nem kerültek kivezetésre változtatható paraméterként. A módosítás elvégzésével javul az algoritmus újrahasználatossága különböző projektek között, mert nem a Python Node-ba írt kódot kell módosítani, hanem csak a bemeneti paramétert, így megszüntetünk egy nagy hibafaktort az algoritmus használatában. Emellett a használatával jelentős munkaidő takarítható meg, egy nagy léptékű épület modellezésénél, mert így nem falanként, manuálisan kell a falburkolat elemeit modellezni, hanem gyakorlatilag egy gombnyomásra elhelyezésre kerülnek a falburkolati elemek. Fontos megemlíteni, hogy az algoritmussal elkerülhető az emberi hiba faktor is, mely szintén jelentős időmegtakarítást jelent, mert így a falburkolatok típusának helyességét sem kell ellenőrizni.

A második létrehozott algoritmus egy ritkábban előforduló problémára nyújt megoldást, azonban, mint arról már írtam az előző pontban, jelentősen csökkenti a modellezésre fordítandó idő mennyiségét. Az algoritmus eredményeként kialakult pontfelhő a jelenleg forgalomban lévő 3D modellező szoftverek többségével már használható, így leképezhető a háromdimenziós felmérési állomány.

A kutatási időszak alatt a létrehozott algoritmusokhoz számos fejlesztési irányt határoztam meg, a pályázat lezárását követően folytatni fogok az algoritmusok fejlesztését, valamint egyéb, a DWG állományok szélesebb körű felhasználását célzó algoritmusok létrehozását. Úgy gondolom, hogy a kutatás rávilágít a PDF és DWG állományok, valamint a tervező és modellező szoftverek közötti információ átadás hiányosságaira. Az első két algoritmusnál meghatározott módszertanok alapján az információátadásra jelenleg kiegészítő állományok

létrehozására van szükség, mely egy bonyolult épület esetén exponenciálisan növelheti az algoritmusok futási idejét és erőforrásigényét. A kutatás zárásaként a kutatási eredmények cikk formájában publikálásra kerülnek.

5. Fogalomtár

API – „Application Programming Interface” rövidítése, az adott szoftver utasításkészletét tartalmazza, mely megkönnyíti a fejlesztők munkáját.

CAD – Computer-Aided Design: Számítógéppel támogatott tervezés.

CAM – Computer-Aided Manufacturing: Számítógéppel támogatott gyártás

Renderelés – Renderelésnek hívjuk azt a folyamatot, mely során a modellező szoftver 2D vagy 3D leképezést készít például egy 3D modellről. Renderelésnek hívjuk azt a folyamatot is, mely során egy videószerkesztő szoftver véglegesíti a vágott videó beállításait és létrehozza a végleges terméket.

GLIDE – Egy integrált tervezői adatbázis, mely tartalmazza egy mérnöki projekt tervezéséhez és kivitelezéséhez szükséges információkat. Egy adatbázist tartalmazó számítógépes nyelv, magában foglalja modellek geometriai, grafikai és utasításfeldolgozási eszközeit is.

BIM – „Building Information Modeling”, Épületinformációs modellezés, a BIM olyan CAD-alapú tervezésmódszertani folyamatok és irányelvek alkalmazásának összessége, amely lehetővé teszi az építmények létrehozásában és üzemeltetésében érdekelt szereplők (építetők, tervezők, kivitelezők, üzemeltetők) számára a valóságnak megfelelő virtuális térben történő együttműködést és információátadást, illetve a releváns adatok gyors és hatékony megjelenítését.

PDF – „Portable Document Format”, fájlformátum. Az Adobe fejlesztette ki a dokumentumok rendszerektől (operációs rendszer, eszközök, szoftverek) független megjelenítés céljából. A formátum lényege, hogy a dokumentum minden platformon ugyan úgy jelenjen meg.

DWG – fájlformátum, egy szabadalmaztatott bináris fájlformátum, amely két és háromdimenziós tervezési adatok és metaadatok tárolására szolgál.

NURBS – „Non-uniform rational basis spline”, egy matematikai modell, mellyel ábrázolható egy tetszőleges 3D modell vagy görbe.

AEC – „Architectural, Engineering and Construction industry”, az építészet, gépészet és kivitelezés iparágak külföldi szaklapokban használt összefoglaló neve.

Parametria – Egy olyan folyamat, melynek kimenetelét különböző bemeneti paraméterekkel változtathatjuk. Például egy négyszög parametrikus, ha valamelyik oldalának hosszát, vagy közbezárt szögeit dinamikusan változtatni tudjuk egy vagy több változón keresztül. A kutatásomban használt Dynamo és Grasshopper parametrikus tervezési bővítmények.

IM – „Information Management”, információk kezelése

Python – Egy általános célú és magas szintű programozási nyelv, mely könnyíti a kód olvashatóságát.

Python Script – Scripteknek hívják azokat a rövid kódokat, melyek az algoritmusban egy kisebb folyamat elvégzéséért felelősek. Általában olyan műveleteket végeznek, amelyeket Node-okkal időigényes és bonyolult lenne elvégezni.

Node – A Revit Dynamo modulját, valamint a Rhinoceros Grasshopper modulját alkotó legkisebb egységek. Egy Node egy funkció ellátására képes, meghatározza a tárolt művelet elvégzéséhez szükséges paramétereket, tartalmazza az általa végzett folyamat rövid leírását, majd visszaadja a folyamat végeredményét, amely akár egy vagy több kimeneti pont is lehet.

Polyline – Pontokkal tördelt, több vonalból álló, összefüggő vonallánc. A tárolt vonalat lehetnek egyenesek és ívesek is.

Határoló doboz – „Bounding Box”, egy tárgy határoló dobozának nevezzük azt a téglatestet, melyből a tárgy nem lóg ki, valamint a téglatest mérete pontosan megfelel a tárgy legnagyobb magasságának, hosszának és szélességének.

Transzponálás – „Transpose”, röviden pozíció cserét jelent. Az algoritmusba importált CSV állomány esetében a bemeneti állományt oszloponként külön listába rendezi a Node, transzponálás esetén pedig soronként végzi el ugyan ezt a műveletet.

6. Hivatkozások

- [1] C. M. Eastman és M. Henrion, „The GLIDE Language for CAD”, *Journal of the Technical Councils of ASCE*, köt. 106, sz. 1, o. 171–184, aug. 1980, doi: 10.1061/JTCAD9.0000065.
- [2] devsupport, „The History of CAD/CAM - Computer Aided Design & Manufacturing”, Technical Foam Services. Elérés: 2022. december 14. [Online]. Elérhető: <https://technicalfoamservices.co.uk/blog/blog-history-of-cad-cam/>
- [3] „Non-uniform rational B-spline”, *Wikipedia*. 2022. november 19. Elérés: 2022. december 14. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=Non-uniform_rational_B-spline&oldid=1122800985
- [4] 14:00-17:00, „ISO 10303-21:2016”, ISO. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.iso.org/standard/63141.html>
- [5] „Revit szoftver | Revit 2023 hivatalos verzió, árszabása és megvásárlása | Autodesk”. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.autodesk.hu/products/revit/overview>
- [6] „Solutions”, SOLIDWORKS. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.solidworks.com/solutions>
- [7] „3D Design Software | 3D Modeling on the Web”, SketchUp. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.sketchup.com/page/homepage>
- [8] mokki, „The History of Building Information Modelling”, DiConnex. Elérés: 2022. december 14. [Online]. Elérhető: <https://diconnex.com/en/blog/2020/05/05/the-history-of-building-information-modelling/>
- [9] M. Wierzbicki, „BIM – History and Trends”, okt. 2011.
- [10] P. M. Máder, O. Rák, N. Bakai, J. Etlinger, M. Zagorác, és I. E. Háber, „Use of algorithms in BIM-based audit processes”, *Pollack Periodica*, köt. 17, sz. 2, o. 20–25, dec. 2021, doi: 10.1556/606.2021.00485.
- [11] K. S. Ravshanovich, „REVIT+DYNAMO”, *Pind. Jour. C. Lite. ELT*, köt. 2, sz. 1, Art. sz. 1, jan. 2022.
- [12] „1404-1.pdf”. Elérés: 2022. december 14. [Online]. Elérhető: https://alfabuild.spbstu.ru/userfiles/files/AlfaBuild/AlfaBuild_2020_14/1404-1.pdf
- [13] E. Iran-Nejad, „eirannejad/pyRevit”. 2024. április 6. Elérés: 2024. április 10. [Online]. Elérhető: <https://github.com/eirannejad/pyRevit>
- [14] X. Yin, P. Wonka, és A. Razdan, „Generating 3D Building Models from Architectural Drawings: A Survey”, *IEEE Computer Graphics and Applications*, köt. 29, sz. 1, o. 20–30, 2009, doi: 10.1109/MCG.2009.9.
- [15] Márk M. P., „Algoritmussal támogatott BIM alapú tevékenységek az építőipar megvalósulási és megvalósítási folyamataiban”, febr. 2022, Elérés: 2022. december 14. [Online]. Elérhető: <http://pea.lib.pte.hu/handle/pea/33826>
- [16] P. Wilson, „A Short History of CAD Data Transfer Standards”, *IEEE Computer Graphics and Applications*, köt. 7, sz. 06, o. 64–67, jún. 1987, doi: 10.1109/MCG.1987.276899.
- [17] A. Beck, „60 Years of CAD Infographic: The History of CAD since 1957”, CADENAS PARTsolutions. Elérés: 2022. december 14. [Online]. Elérhető: <https://partsolutions.com/60-years-of-cad-infographic-the-history-of-cad-since-1957/>
- [18] „History of CAD software”, *Wikipedia*. 2022. december 5. Elérés: 2022. december 14. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=History_of_CAD_software&oldid=1125773702

- [19] „Computer-aided design/History, Present and Future - Wikiversity”. Elérés: 2022. december 14. [Online]. Elérhető: https://en.wikiversity.org/wiki/Computer-aided_design/History,_Present_and_Future
- [20] „AutoCAD Software | Get Prices & Buy Official AutoCAD 2023 | Autodesk”. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.autodesk.com/products/autocad/overview>
- [21] R. M. & Associates, „What are NURBS?”, www.rhino3d.com. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.rhino3d.com/features/nurbs/>
- [22] „What is nonuniform rational B-spline (NURBS)? | Definition from TechTarget”, WhatIs.com. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.techtarget.com/whatis/definition/nonuniform-rational-B-spline-NURBS>
- [23] „A Brief History of BIM”, ArchDaily. Elérés: 2022. december 14. [Online]. Elérhető: <https://www.archdaily.com/302490/a-brief-history-of-bim>
- [24] „lechner-tudaskozpont-bim-kezikonyv-1-kotet-1-kiadas.pdf”. Elérés: 2022. december 14. [Online]. Elérhető: <https://lechnerkozpont.hu/doc/terbeli-szolgáltatások-attekintes/lechner-tudaskozpont-bim-kezikonyv-1-kotet-1-kiadas.pdf>
- [25] „Take Dynamo Further ”. Elérés: 2022. december 14. [Online]. Elérhető: <https://dynamopythonprimer.gitbook.io/dynamo-python-primer/>
- [26] „ApiDocs.co · Revit · GeometryInstance Members”. Elérés: 2022. december 14. [Online]. Elérhető: <https://apidocs.co/apps/revit/2021/56d06b4e-8695-9343-6b74-ae9ff9e27152.htm>
- [27] „ironpython-stubs/release/stubs.min at master · gtalarico/ironpython-stubs”, GitHub. Elérés: 2022. december 14. [Online]. Elérhető: <https://github.com/gtalarico/ironpython-stubs>
- [28] A. de Chasteigner, „albandechasteigner/GeniusLociForDynamo”. 2022. november 26. Elérés: 2022. december 14. [Online]. Elérhető: <https://github.com/albandechasteigner/GeniusLociForDynamo>
- [29] „Geometry Overview | The Dynamo Primer”. Elérés: 2022. december 14. [Online]. Elérhető: https://primer.dynamobim.org/05_Geometry-for-Computational-Design/5-1_geometry-overview.html
- [30] „IronPython.net / Documentation / .NET Integration”. Elérés: 2022. december 14. [Online]. Elérhető: <https://ironpython.net/documentation/dotnet/>
- [31] „Python 0.6.3 to 0.7.x Migration · DynamoDS/Dynamo Wiki”, GitHub. Elérés: 2022. december 14. [Online]. Elérhető: <https://github.com/DynamoDS/Dynamo>

7. Ábrajegyzék

1. ÁBRA - PÉCS VÁROSTÉRKÉP RÉSZLET AUTOCAD-BEN (SAJÁT ÁBRA)	10
2. ÁBRA - DYNAMO-BAN KINYERT ADATOK A DWG IMPORT UTÁN (SAJÁT ÁBRA)	11
3. ÁBRA - AZ ELHELYEZETT AJTÓ BLOKKOK AUTOCAD-BEN (SAJÁT ÁBRA)	12
4. ÁBRA - DWG BETÖRÉSE REVITBE, MAJD AZ ELEMÉK MEGJELENÍTÉSE DYNAMO-BAN (SAJÁT ÁBRA)	13
5. ÁBRA - ÁLLOMÁNYOK MEGJELENÉSÉNEK ÖSSZEHASONLÍTÁSA (SAJÁT ÁBRA)	16
6. ÁBRA - SNAP FUNKCIÓVAL A VONAL ÉRZÉKELÉSE (SAJÁT ÁBRA)	16
7. ÁBRA - DWG ÁLLOMÁNYBÓL KINYERHETŐ ADATOK (SAJÁT ÁBRA)	17
8. ÁBRA - DWG ÁLLOMÁNYBÓL KINYERHETŐ ADATOK DYNAMO-BAN (SAJÁT ÁBRA).....	17
9. ÁBRA - AUTOCAD DATAEXTRACTION PARANCCSAL KINYERHETŐ ADATOK (SAJÁT ÁBRA)	19
10. ÁBRA - DATAEXTRACTION PARANCCSAL KINYERT ADATOK (SAJÁT ÁBRA).....	20
11. ÁBRA - GEOMETRIA KINYERÉSE DWG ÁLLOMÁNYBÓL DYNAMOVAL (SAJÁT ÁBRA)	21
12. ÁBRA - FALBURKOLATOKAT TARTALMAZÓ CSV (SAJÁT ÁBRA)	22
13. ÁBRA - REVIT-BEN LÉTREHOZOTT FALBURKOLATOK (SAJÁT ÁBRA)	23
14. ÁBRA - DWG ÉS CSV ÁLLOMÁNYOKAT IMPORTÁLÁSÁNAK FOLYAMATA (SAJÁT ÁBRA)	24
15. ÁBRA - GEOMETRIA KINYERÉSÉT VÉGZŐ NODE-OK (SAJÁT ÁBRA)	25
16. ÁBRA - DWG ÁLLOMÁNYBÓL KINYERT GEOMETRIA (SAJÁT ÁBRA)	25
17. ÁBRA - AJTÓK HATÁROLÓ DOBOZÁNAK MEGHATÁROZÁSA (SAJÁT ÁBRA).....	26
18. ÁBRA - KINYERT ÉS LÉTREHOZOTT GEOMETRIÁK EGYESÍTÉSE (SAJÁT ÁBRA)	27
19. ÁBRA - FALAK REFERENCIA VONALÁNAK ELTOLÁSA ÉS MÉRTÉKEGYSÉG VÁLTÁS (SAJÁT ÁBRA)	27
20. ÁBRA - FALAKAT LÉTREHOZÓ NODE-OK ÉS A LÉTREHOZOTT FALAK 3D NÉZETBEN (SAJÁT ÁBRA).....	28
21. ÁBRA - DWG ÁLLOMÁNYBÓL KINYERT RAJZI ELEMÉK ÉS SZÖVEGDOBOZOK IMPORTJA GRASSHOPPER-BE (SAJÁT ÁBRA)	30
22. ÁBRA - RAJZI ELEMÉKBŐL LÉTREHOZOTT PONTOK, A PONTOKHOZ TARTOZÓ MAGASSÁGI ADATOKKAL (SAJÁT ÁBRA)	30
23. ÁBRA - A PONTOK ÉS SZÖVEGDOBOZOK KINYERÉSÉT VÉGZŐ PYTHON KÓD (SAJÁT ÁBRA).....	31
24. ÁBRA - A KINYERT ADATOK (SAJÁT ÁBRA)	33
25. ÁBRA - KOORDINÁTÁK CSOPORTOSÍTÁSA, SZŰRÉSE ÉS RENDEZÉSE (SAJÁT ÁBRA)	34
26. ÁBRA - SZŰRÉS UTÁN A KOORDINÁTÁK LISTÁJÁNAK DARABOLÁSA (SAJÁT ÁBRA)	35
27. ÁBRA - LISTÁK VÁGÁSA ÉS A MEGEMELT PONTOK LÉTREHOZÁSA (SAJÁT ÁBRA)	35
28. ÁBRA - A MEGEMELT PONTOK (ZÖLD) ÉS KINYERT PONTOK (PIROS) (SAJÁT ÁBRA)	36
29. ÁBRA - A MEGEMELT ÉS KIINDULÁSI PONTOK 3D NÉZETE (SAJÁT ÁBRA)	37