BME Fotogrammetria és Térinformatika Tanszék

Pontfelhők OpenCRG és járműdinamikai hasznosítási lehetőségei

Baranyai Dániel R66QGC

2021 tavaszi félév

Konzulensek: Dr. Lovas Tamás, egyetemi docens

Dr. Somogyi József Árpád, adjunktus

Absztrakt

Kulcsszavak: TLS, MLS, OpenCRG, OpenDRIVE

A lézerszkenneléses technológia segítségével előállított digitális útfelületeknek jelentős járműipari felhasználása lehet pl.: járműdinamikai szimulációk, önvezető szimulációk. Ilyen szimulációs környezetekben legelterjedtebb MATLAB modul az ASAM OpenCRG. Az OpenCRG úttengely menti ívelt szabályos rács (Curved Regular Grid) pontjaiban tárolja a magassági értékeket, amiket a lézerszkennelés során keletkezett pontfelhőből nyerünk ki.

Kutatómunkámban bemutatom azt az eljárási folyamatot, hogyan jutunk el a nyerspontfelhőtől az ívelt szabályos rács menti pontokig egyszerű és komplex útgeometria esetén. Továbbá vizsgálom két felmérési eljárás (földi és mobil lézerszkennelés) adatállományaiból létrehozott digitális útfelület közti különbségeket.

Abstract

Keywords: TLS, MLS, OpenCRG, OpenDRIVE

The digital road surface produced by laser scanning technology can have significant vehicle industrial applications e.g. vehicle dynamics simulations, autonomous driving simulations. In such simulation environments, the most common MATLAB module is ASAM OpenCRG. OpenCRG stores the elevation values in points of the Curved Regular Grid along the road axis, which are extracted from the point cloud generated during laser scanning.

In my research, I would like to show the process of how to get from the raw point cloud to the points on the Curved Regular Grid for simple and complex road geometries. Furthermore, I will investigate the differences between two surveying procedures i.e. terrestrial and mobile laser scanning-generated digital path surface.

Tartalomjegyzék:

1. OPENCRG – OPENDRIVE

Az Automatizálási és Méréstechnikai Rendszerek Szabványosítási Szövetsége (ASAM), amelynek tagjai elsősorban nemzetközi autógyártók, beszállítók és autóipari mérnöki szolgáltatók, műszaki szabványok kidolgozásának koordinálásáért és azok kiadásáért felelős szövetség. Az ASAM azt az elképzelést követi, hogy a fejlesztési folyamatlánc eszközei szabadon összekapcsolhatók legyenek, és lehetővé tegyék a zökkenőmentes adatcserét. A szövetség szorosan együttműködik más szervezetekkel, mint az ISO és az AUTOSAR. A szabványok adatmodelleket, fájlformátumokat, alkalmazás programozási felületeket határoznak meg [1]. Például az OpenSCENARIO, OpenDRIVE, OpenCRG,

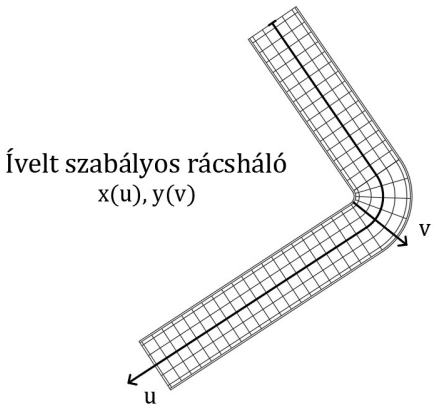
Az OpenSCENARIO használata olyan összetett, szinkronizált események leírása, amelyek járműveket, gyalogosokat és más közlekedésben résztvevőket tartalmaznak. Egy történet leírhatja egyetlen jármű vezetési manővereit, vagy több egymástól függő eseményben résztvevők dinamikus viselkedését. A történetek egymásra épülnek, egyes cselekmények akkor indulnak el, ha valamilyen feltétel teljesül pl.: meghatározott sebesség túllépése, biztonsági távolság átlépése. Vizsgálhatók továbbá azesemények hatásai, következményei is pl.: forgalmi dugók kialakulása. A szabványt az OpenDRIVE úthálózat geometriai leírásával együtt alkalmazzák.

Az OpenDRIVE lehetőséget biztosít utak, csomópontok definiálására, geometria és egyéb attribútum (pl.: út járhatósága, úttípus, útjelző objektumok) segítségével. A szabvány célja, hogy olyan úthálózati leírást biztosítson, ami kiszolgálja a járműszimulációs környezeteket. Az útleírások szabványosítása kikerüli az új fejlesztésű és tesztelésű célfájlok igényét, így csökkentve az iparág számára az ezekkel járó költségeket. Az útadatok létrehozhatók térképadatokból, úthálózat szerkesztőkből vagy valós utak feldogozott felméréseiből (pl. lézerszkennelésből). A leíró állomány .xml fájl struktúrájú, .xodr kiterjesztésű. A szabvány meghívhatja az OpenCRG által elkészített felületmodelleket [1][5][6].

Lézerszkenneléssel nagyfelbontású, nagypontosságú pontfelhők állíthatók elő az utak felületéről, amelyekből olyan felületmodellek vezethetők le, melyekre az út digitális ikermodelljeként (Digital Twin) tekinthetünk, és felhasználhatjuk azokat predikciós szimulációk részeként pl.: jármű avulás, gumiabroncs, vibráció és vezetés szimuláció.

Ennek egy lehetséges útja az ASAM OpenCRG, amely lehetőséget ad arra, hogy a rendelkezésre álló magassági adatokból felszínmodell vezessünk le. Ezen adatokat „.crg” formátumban tárolhatjuk, így felhasználható további szimulációs környezetekben pl.: IPG CarMaker [4].

Az Curved Regular Grid (CRG) egy út vagy sávszakasz referenciavonala mentén szabályos rácshálóban határozható meg (1.1 ábra). A rácsháló pontjaiban az út magassági értékeit tároljuk (z érték). Egy út/sáv esetén a rács sorai az út szélessége mentén felvett magassági pontok, oszlopai pedig az úttengelyen felvett pontok. Minden sorhoz tartozik egy irányszög, mely megadja a referenciavonal görbületét [1].



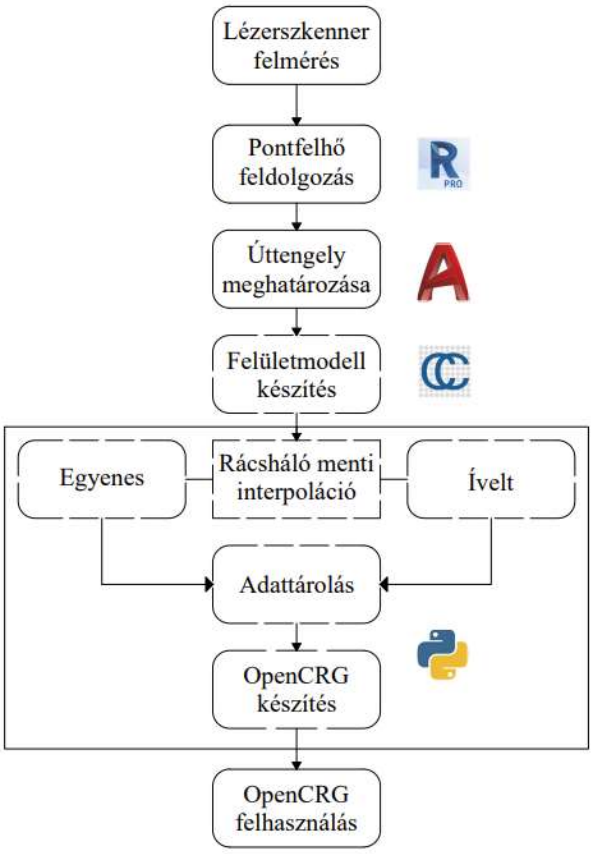
1.1. ábra: Referenciavonal mentén definiált szabályos rácsháló

(BME K épület melletti útszakasz)

A korábban említett három szabvány kiegészítési egymást, lefedve a járműszimulációs alkalmazások statikus és dinamikus tartalmait.

2. CRG LÉTREHOZÁSA EGYSZERŰ ÉS KOMPLEX GEOMETRIA ESETÉN

Útfelületek nagypontosságú és nagyfelbontású modellezéséhez olyan magassági információkra van szükség, amely hatékonyan lézerszkennelés alapú adatnyerési eljárással állítható elő. Korlátozott terület, illetve nagy felbontás és pontosság igény esetén jellemzően a földi lézerszkennelés biztosít hatékony megoldást, amikor rögzített álláspontokból mérjük fel az utat (és környezetét). Nagy kiterjedésű, hosszabb útszakaszok esetén gazdaságosabb megoldást biztosít a mobil térképezés, pl. járműre szerelt lézerszkennerekből és kamerákból álló szenzorcsoport alkalmazása. Mindkét technológia kimenete a pontfelhő és mindkét technológia igényelheti kiegészítő geodéziai mérések végzését a pontosság növelése, illetve georeferálás céljából. A lézerszkenneléssel gyűjtött pontok jellemzően nem az OpenCRG-nek megfelelően az út tengelyére rendeződve állnak elő, emiatt a pontok újra-rendezésére van szükség. A 2.1. ábra mutatja a munkafolyamatot a geometriai adatok gyűjtésétől az OpenCRG modellezésig. Az ábrán szaggatottal jelöltem a jelenleg részben vagy teljesen automatizált folyamatokat.



2.1. ábra: CRG feldolgozás folyamatábrája

2.1. Pontfelhő előfeldolgozás

Cél olyan pontok legyűjtése a pontfelhőből, amelyek az ívelt rácsháló pontjait határozzák meg. Első lépésként szükséges az út tengelyét meghatározni - amennyiben nem áll rendelkezésre más forrásból.

Az 1.1. ábrán ábrázolt *u* vektor jelöli az út tengelyét, erre az *u* tengelyre merőleges egyenesek alkotják a *v* vektort. Az utakról előállított pontfelhő alapján CAD rendszerben manuálisan levezethető azok tengelye. A tengelyt célszerű egyszerűbb objektumokkal, egyenesek és körívek segítségével meghatározni, hogy a későbbi számításokat megkönnyítsük. A pontfelhő szakaszokra bontása nem feltétlen szükséges, dolgozhatunk a teljes pontfelhővel, csak a szakaszok elkülönítése fontos. Nagyobb méretű pontfelhők esetén a gyorsabb interpoláció érdekében célszerű az állomány kisebb méretű egységekre bontása.

2.1.1. Felületmodell készítés

A rendezett adatpontstruktúra kialakításához CloudCompare-ben mesh modellt (felületmodellt) vezettem le. Ennek első lépéseként ki kellett számítani a pontokhoz tartozó normálisokat, amelyeket a CloudCompare algoritmusa a szomszédos pontokra illesztett háromszögháló alapján határoz meg. Ennek során megadható a számításba bevont szomszédos pontok száma, illetve a ponttól mért maximális távolság. A helyi felületmodellezés történhet sík illesztéssel, 2D háromszögeléssel vagy másodfokú görbült felületekkel.

A síkillesztés funkció a megadott szomszédos pontok, illetve azoktól mért maximális távolság függvényében meghatározott pontokra egy síkot illeszt és meghatározza a sík normálvektorát (alapértelmezés szerint pozitív Z koordinátával). A síkillesztés jó eredményeket ad zajos mérések esetén, viszont gyengén teljesít az élek és sarkok kezelésében.

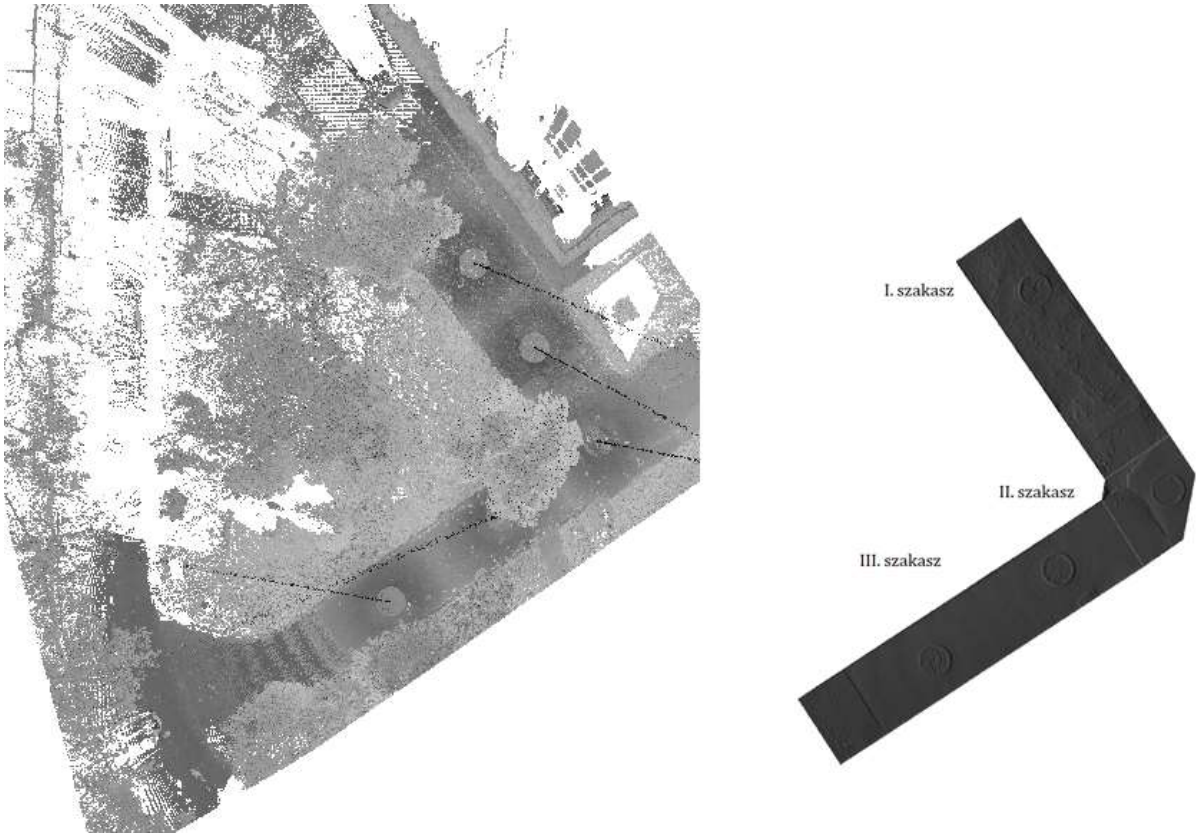
Normálérték számítás során, ha a helyi felületmodellezés 2D háromszögeléssel történik, akkor egymást nem fedő háromszög síkok normálvektorát határozzuk meg. A szomszédok száma és a keresési távolság megválasztása függ a pontfelhő sűrűségétől; ezt az értéket automatikus módon határozza meg a program. A 2D háromszögelés érzékeny a zajra, viszont jól visszaadja a sarkokat, éleket és a görbe felületeket.

Továbbá a normálértéket számíthatjuk másodfokú görbült felület funkcióval, ilyenkor a szomszédos pontok, illetve azoktól mért maximális távolság függvényében meghatározott

pontokra az eljárás egy másodfokú felületet illeszt, majd ennek a síknak határozza meg a normálvektorát.

Mivel egy útszakaszról készültek a felmérések, és cél a kisméretű felületi egyenetlenségek kimutatása, a 2D háromszögelést választottam a normálisok meghatározásához.

A CloudCompare Misha Kazhdan által elkészített Poisson modellezéssel készíti el a mesh modellt, ennek bemenete a normálértékekkel rendelkező pontfelhő [7][2]. Alapértelmezett felhasználása zárt 3D alakzatokra érvényes, azonban utómunkával nyitott mesh modellek létrehozására is alkalmazható. Az nyolcasfa mélység növelésével egyre finomabb modellt kaphatunk, azonban ezzel együtt a számítási idő is növekszik. Futtatás után, egy zárt mesh modell készült el, amelyet egy AutoCAD-ből importált poligon segítségével vágtam át, eredményül pedig megkaptam a lehatárolt mesh modellt (2.2. ábra).



2.2. ábra: Mesh modell

2.2. Interpoláció egyszerű geometria esetén

Egyenes útszakaszok esetén az ívelt szabályos rácsháló a következőképpen kerül meghatározásra. Mivel egyenes útszakasz görbületi értéke 0, így az ívelt rácsháló egy egyenes szabályos rácshálónak feleltethető meg. A rácsháló létrehozásához az egyenes kezdő és végkoordinátájára, illetve a sávszélességre van szükség. Továbbá szükséges még két paramétert meghatározunk, amelyek a rácsvonalak távolságát adják meg, az OpenCRG alapértelmezett értékei: u_{inc} = 0.01 m, v_{inc} = 0.01 m. Tehát az úttengellyel párhuzamosan *„u_{inc}”* távolsággal, a teljes sávszélességre kiosztva, illetve az úttengelyre merőlegesen *„v_{inc}”* távolsággal és a teljes útszakasz hosszára kiosztva határozzuk meg a rácshálót. Ennek meghatározása a következő: legyen „x” olyan sorvektor mely 0-tól az úttengely hosszáig tart *„u_{inc}*” lépésközzel, illetve „y” olyan sorvektor, ami a sávszélesség felének -1-szeresétől az úttengelyre merőleges szélesség feléig tart *„v_{inc}”* lépésközzel.

$$x_i = u_{inc} \times i \times \frac{|u|}{u_{inc}}, \qquad \bar{x} = (x_1, x_2, \dots x_i) \tag{2.1}$$

$$y_j = -\frac{|v|}{2} + v_{inc} \times j \times \frac{|v|}{v_{inc}}, \qquad \bar{y} = (y_1, y_2, \dots y_i) \tag{2.2}$$

ahol |u| az úttengely hossza, |v| az u vektorra merőleges egyenes hossza (sávszélesség).

A két vektor segítségével, „X” és „Y” mátrixokat hozunk létre, ahol „X” egy olyan mátrix, aminek minden sora másolata az „$\bar{x}$”-nak, és az „Y” pedig egy olyan mátrix, aminek minden oszlopa másolata a „$\bar{y}$” vektornak. Ahhoz, hogy a rácshálónk „$\bar{x}$” sorai párhuzamosak legyenek az úttengellyel, forgatást kell alkalmaznunk. A forgatási érték az „u" vektornak az x tengellyel bezárt szöge lesz. A vektorok origója az úttengely kezdőpontja. Végül a tömböt eltoljuk az úttengely kezdőpontjával.

$$\propto = atan2(y_1 - y_0, x_1 - x_0) \tag{2.3}$$

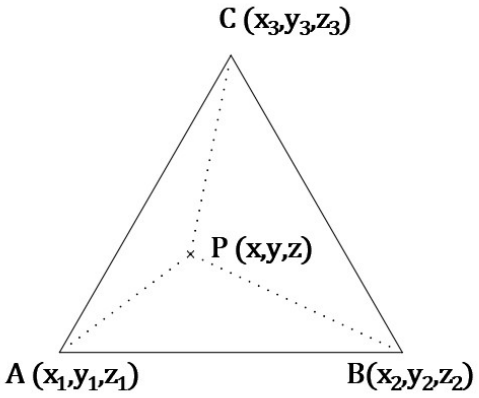
$$X' = X \times \cos \propto - Y \times \sin \propto \tag{2.4}$$

$$Y' = X \times \sin \propto + Y \times \cos \propto \tag{2.5}$$

ahol a tengely kezdőpontja x_0, y_0, végpontja x_1, y_1.

A betöltött mesh és a korábban létrehozott rácsháló segítségével interpolációt hajthatunk végre a rácsháló pontjaira. Lehetőségünk van lineáris, legközelebbi szomszéd alapú vagy köbös

interpolációra. A pontok magassági értékei lineáris baricentrikus interpoláció segítségével lettek meghatározva. Legyen „P” a pontunk, az „ABC” pedig a vizsgált mesh modell által definiált háromszög egyike (2.3. ábra).



2.3. ábra: Magassági érték interpolációja egy P pontra

Mivel az ABC háromszög csúcspontjaiban ismert a „z” magassági érték, így a P pont magassági értékét interpolálhatjuk a háromszög segítségével. Súlyponti koordináták segítségével számíthatjuk a „P” pont „z” magassági értékét (2.6 képlet).

$$w_1 = \frac{(y_2-y_3)(x_p-x_3)+(x_3-x_2)(y_p-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}$$

$$w_2 = \frac{(y_3-y_1)(x_p-x_3)+(x_1-x_3)(y_p-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}$$

$$w_3 = 1 - w_1 - w_2$$

$$z = z_1 \times w_1 + z_2 \times w_2 + z_3 \times w_3 \qquad (2.6)$$

ahol w_1, w_2, w_3 a súlyozás értéke.

A súlyponti koordináták számítása során egy ellenőrzést is végzünk, ugyanis, ha valamelyik súly értéke negatív, akkor a pont a háromszögön kívül helyezkedik el. Minden keresett pontra elvégezve megkapjuk a keresett magassági értékét. A számítási folyamatot Python programozási nyelven írtam meg, ami a mellékletben „straight_road” funkció néven található meg. Bementi értéke sorban, a felület modell elérési útvonala, kezdőpont, végpont, útszélesség, fájl mentési útvonala, illetve alapértelmezett értékekként az u_{inc} = 0.01 m, v_{inc} = 0.01 m.

2.3. Interpoláció körív esetén

Korábban meghatároztuk CAD környezetben az út/sáv körívének sugarát és középpontját. Az ívelt rácsháló elkészítéséhez meg kell határozni a kezdőpont és a végpont közti szögértékét, amit számíthatunk az úttengelyből (2.7 képlet). A rácsháló meghatározását érdemes polárkoordinátákkal felírni. A rácsháló origója legyen a körív origója, majd következő lépésként határozzuk meg azt a „φ" szögértéket mely az úttengely mentén „u_{inc}" körív hosszat ad (2.8. számú képlet).

$$\Theta_{min} = \cos^{-1}\left(\frac{y_{p1}-y_o}{R}\right), \quad \Theta_{max} = \cos^{-1}\left(\frac{y_{p2}-y_o}{R}\right) \tag{2.7}$$

ahol Θ_{min} minimális szögértéke [°], Θ_{max} maximális szögértéke [°] az ívelt úttengely origójára

$$\varphi = \frac{u_{inc}}{R} \tag{2.8}$$

ahol φ szögérték [°] mely az úttengely sugár mentén „R" [m], „u_{inc}" körkerületet ad [m]

A „φ" szögérték és annak többszörösének segítségével meghatározhatjuk az úttengelyen „u_{inc}" távolságra lévő szakaszokat. Ezután meghatározhatjuk az úttengellyel párhuzamos szakaszokat, melyek „v_{inc}" távolságra és annak többszörösére vannak „s" sávszélesség terjedelmében. A pontok polárkoordináta formában a következő mátrix segítségével írhatók fel.

$$r_i = R - \frac{s}{2} + v_{inc} \times i, \qquad i = 0,1, \ldots \left[\frac{s}{v_{inc}}\right], \tag{2.9}$$

$$\Theta_j = \Theta_{pmin} + \varphi \times j, \qquad j = 0,1, \ldots \left[\frac{\Theta_{pmax}-\Theta_{pmin}}{\varphi}\right], \tag{2.10}$$

$$A = \begin{bmatrix} r_{11}, \Theta_{11} & \cdots & r_{i1,}\Theta_{i1} \\ \vdots & \ddots & \vdots \\ r_{1j}, \Theta_{1j} & \cdots & r_{ij,}\Theta_{ij} \end{bmatrix}. \tag{2.11}$$

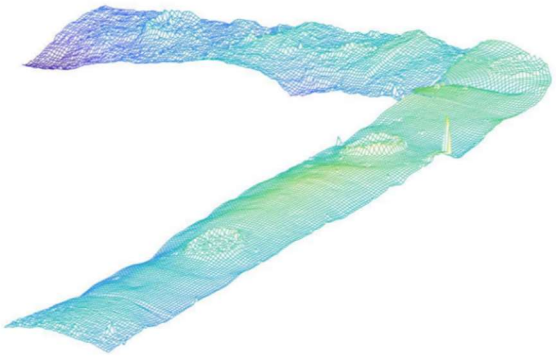
akkor, ha „r" maradék nélkül osztható „v_{inc}" értékkel, illetve, ha „Θ" maradék nélkül osztható „φ" értékkel.

Polárkoordináta-rendszerben előállt pontjainkat adjuk meg derékszögű koordináta-rendszerben.

$$x = r \times \cos\Theta\,, \; y = r \times \sin\Theta, \tag{2.12}$$

$$A = \begin{bmatrix} x_{11}, y_{11} & \cdots & x_{i1}, y_{i1} \\ \vdots & \ddots & \vdots \\ x_{1j}, y_{1j} & \cdots & x_{ij}, y_{ij} \end{bmatrix}. \tag{2.13}$$

Innentől az interpoláció azonos a korábbiakban bemutatott móddal. A mesh modellen (2.4. ábra) lineáris baricentrikus interpoláció segítségével számítható az x, y pontok z magassági értéke. A számítási folyamatot Python programozási nyelven írtam meg, ami a mellékletben „curved_road" funkció néven található meg. Bementi értéke sorban, a felület modell elérési útvonala, kezdőpont, végpont, a körív origója, útszélesség, fájl mentési útvonala, illetve alapértelmezett értékekként az u_{inc} = 0.01 m, v_{inc} = 0.01 m.



2.4. ábra: 0.1×0.1 m-es rácsháló felület modellje

2.4. CRG készítés

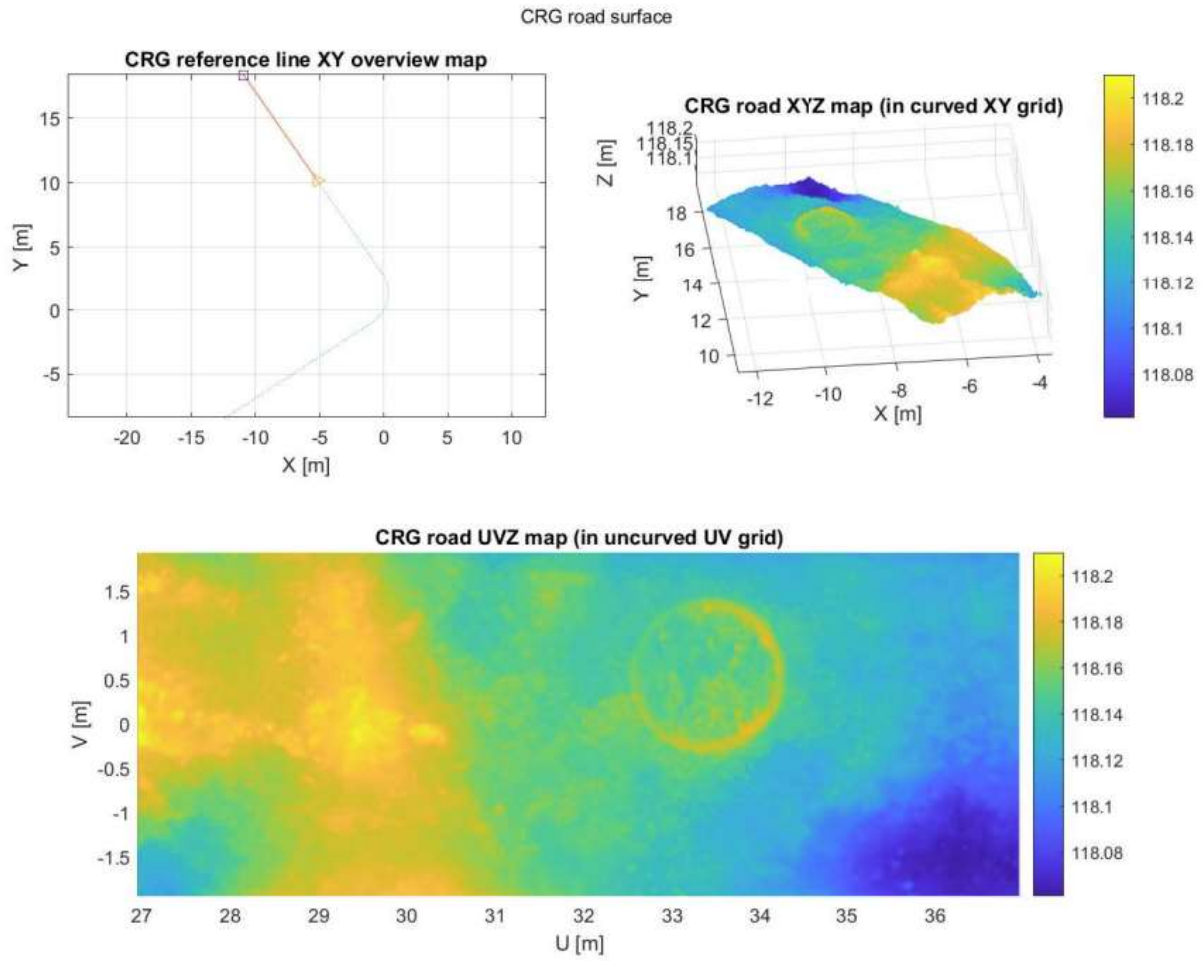
A CRG modellt elkészíthetjük valós mérésekből vagy akár generált adatokból is, az alábbiakban mindkettőre mutatok példát.

2.4.1. Modellezés valós adatokból

A CRG előállításához szükséges bizonyos bemeneti paramétereket megadni, ilyenek: a referencia vonal, görbület nagysága (1/r), az u és v vektorok hossza (sávszélesség fele), a korábban legyűjtött magassági értékek, illetve egy szöveges kiegészítő állomány, amely leírja, hogy mi alapján lett elkészítve a modell. Továbbá megadható a referencia vonal kezdő és végponjtának valamely vetületi rendszerben lévő koordinátái, ezzel georeferálva a modellt.

Az íveket úgy határozhatjuk meg, hogy minden sorhoz rendelünk egy görbületi értéket. A hivatalos OpenCRG görbületi meghatározás, a pozitív görbület az óramutató járásával

ellentétes, a negatív görbület az óramutató járásával megegyező irányú. Továbbá a görbületi értéket mindig az előző sor görbületi értékéhez képest határozzuk meg. Tehát ha a görbületi érték azonos az előzővel, akkor egyenest, ha pedig több vagy kevesebb akkor ellentétes irányú görbületet, vagy megegyező irányú görbületet kapunk. Egy referencia vonal görbületi értéke [-π, + π] között lehet. A 2.5. ábra mutatja egy útszakasz referencia vonalát és CRG modelljét.

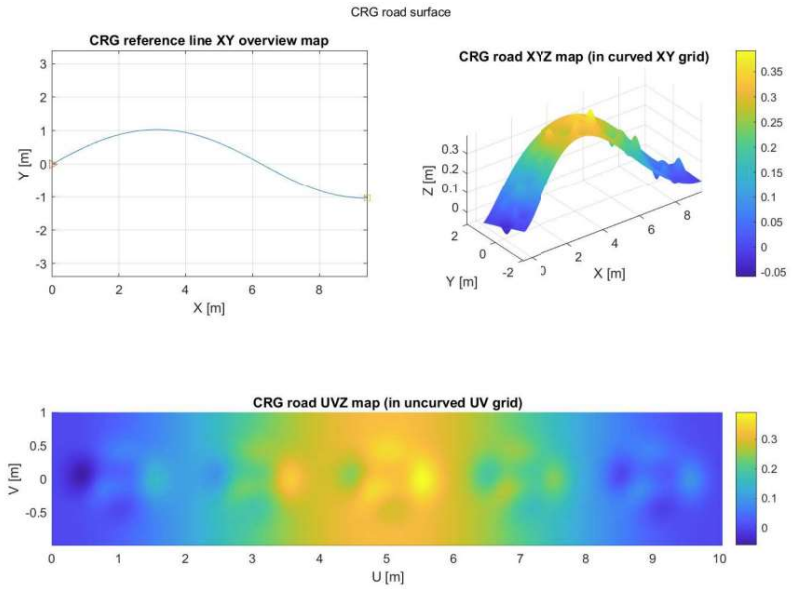


2.5. ábra: III. szakasz CRG modellje

2.4.2. Szintetikus utak

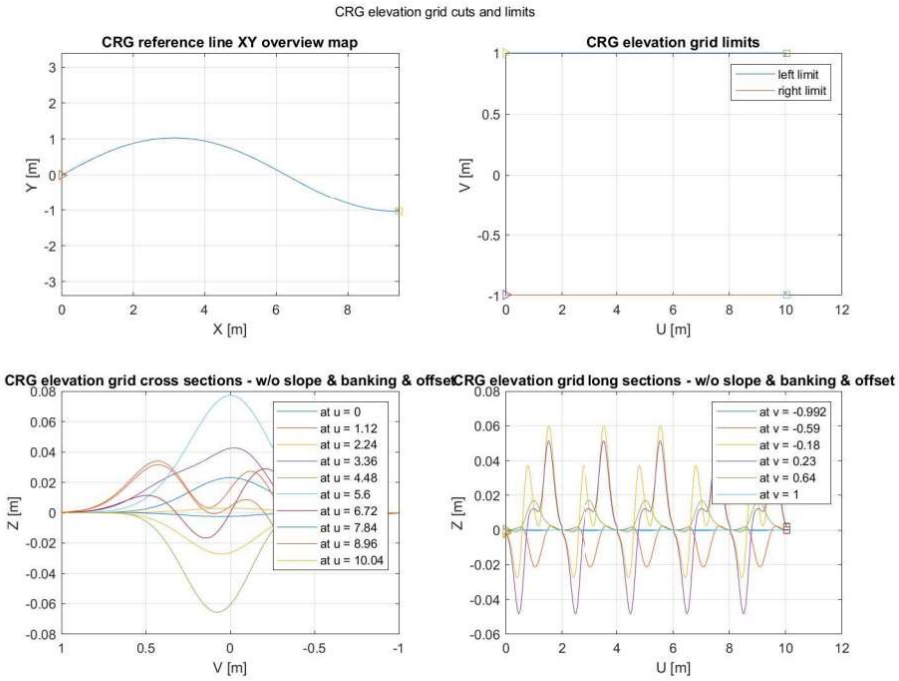
Az OpenCRG sok alapfunkciót ad a szintetikus utak generálására. Van lehetőség esés, oldalesés, görbület vagy akár a „z” tömb manipulálásával tervezett keresztmetszetek alapján utak létrehozására. A szintetikus utak is szerepet kaphatnak, társítva a valós adatokból modellezett utakkal pl.: hiányos adatfelmérés kiegészítéseként, nem járható utak generálására.

Az OpenCRG segítségével tetszőleges geometriájú útszakasz mintaállománya generálható (2.6. ábra).



2.6. ábra: CRG demo9.crg [forrás: CRG minta fájl]

A 2.7. ábra a mintaállomány hossz- és keresztszelvényeit mutatja.



2.7. ábra: CRG kereszt és hossz-szelvényei demo9.crg [forrás: CRG minta fájl]

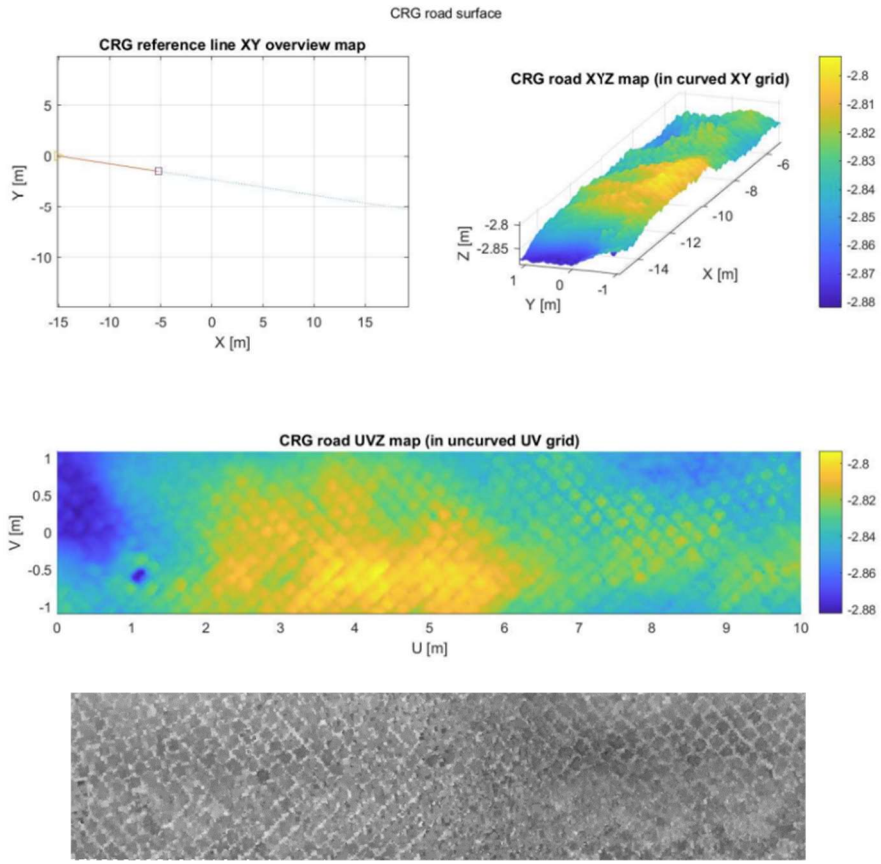
3. MÓDSZER ALKALMAZÁSA KÜLÖNBÖZŐ PONTFELHŐKRE

A korábban meghatározott módszert alkalmaztam különböző út és felmérés típusokra. A Fotogrammetria és Térinformatika tanszék számos mérést végzett az elmúlt években az egyetem területén és környékén, ezeket a felméréseket használtam fel.

3.1. Csíky utca TLS, egyenes geometria

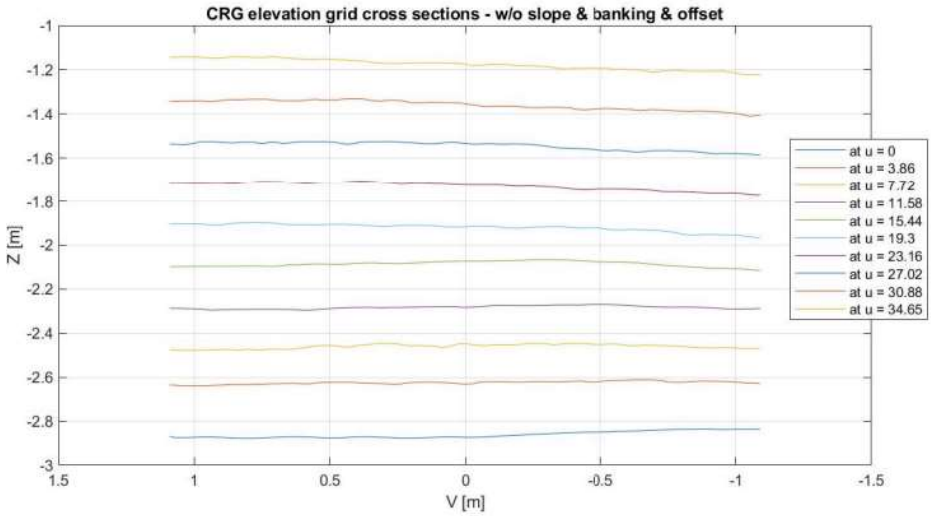
A BME K épülethez közeli Csíky utca macskakővel burkolt útszakasz. A pontfelhőt földi lézerszkenner segítségével állították elő. Mivel ez egy rövidebb szakasz ezért alkalmazhattam a pontfelhő normál értékeinek számítása során nagyobb szomszéd távolságot (~1-1.5 m) anélkül, hogy a futási idő jelentősen növekedett volna. Célszerű a mesh modellünket lehatárolni, amit pedig AutoCAD-ből .dxf formájában tudunk CloudCompare-ba importálni. Fontos, hogy ilyenkor adjunk meg a lehatárolásnak egy megnövelt környezetet pl.: 0.20 méterrel nagyobb keretet. Erre azért van szükség, mert ha azonos a vágott mesh és az interpolálni kívánt pontok területe akkor előfordulhat, hogy a mesh háromszögei nem fedik le teljesen és ilyenkor NaN értékekkel töltődik fel az interpolált pontok magassági értéke. AutoCAD rendszerbe a pontfelhőt úgy hívhatjuk be, ha előtte ReCap segítségével .rcp formátumba alakítjuk. Ilyenkor ritkíthatjuk a pontfelhőt, hogy kezelhetőbb adatmennyiséget kapjunk, ugyanis AutoCAD-ben csak a tengelyeket és a lehatárolást határozzuk meg, ami lehetséges kisebb pontsűrűségű pontfelhőből is.

Mivel a pontfelhőn sok felesleges objektum található, ami kitakarta az útszakaszt (parkoló autók), ezért az útfelület látható tartományára szűkítettem, ami a mintaállományban hosszirányban 34,71m keresztirányban pedig 2.20 méterre adódott. A 3.1 ábra mutatja az állomány referenciavonalát, az OpenCRG modellt és összehasonlításul, a pontfelhő intenzitásképét, melyet jól kivehetők a macskakövek.



3.1. ábra: Csíky utca CRG modellje

Ezen a szakaszon az út kezdő és végpontja között ~1.9 méter magasság különbség van (3.2. ábra).

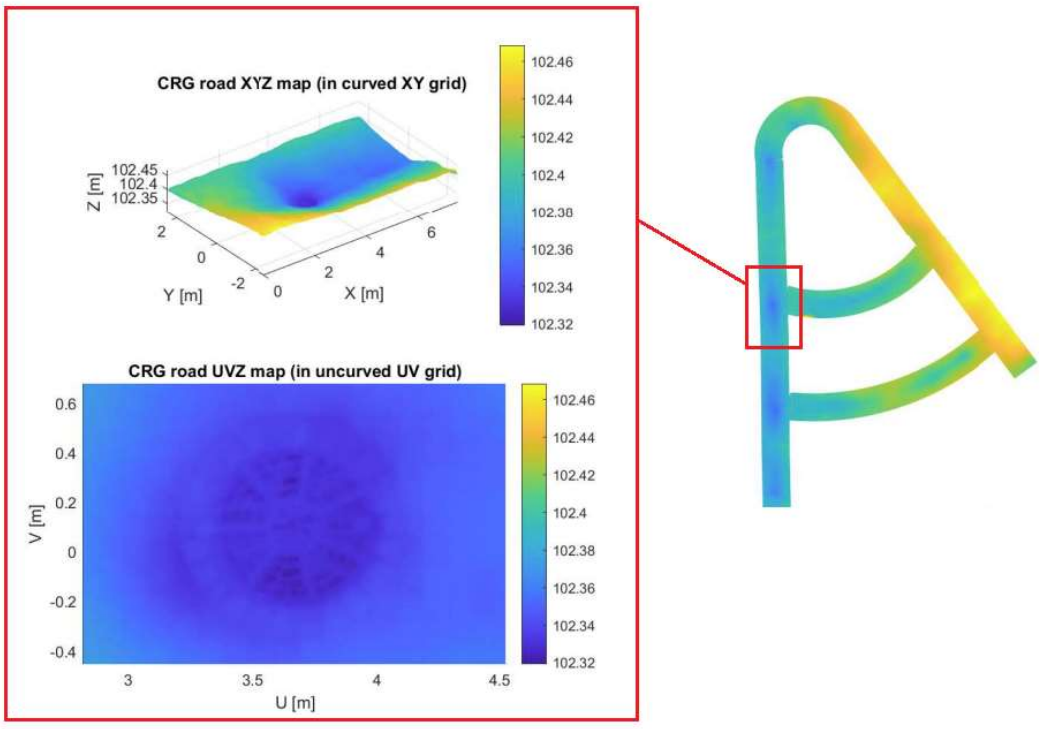


3.2. ábra: Csíky utca keresztszelvényei

A megjelenítés érdekében, hogy látható legyen a modellen a macskakő által okozott felületi egyenetlenség, ezt a magasságkülönbséget kiegyenlítettem. A crg modell jól visszaadja a felület struktúráját, mintázatát; járműdinamikai szempontokból érdekes lehet a macskakővel burkolt útszakaszokon végrehajtott szimuláció támogatása nagyfelbontású és nagypontosságú crg modellekkel (3.1. ábra).

3.2. BME ST épület parkoló TLS, ívelt geometria

Az ST épület környékét is földi lézerszkennerrel mérték fel. Mivel itt már nagyobb az érintett útszakasz, így a pontfelhő normál értékének számítása során ~1 méteres szomszéd távolságot alkalmaztam, a Poisson rekonstrukció során pedig 12-es nyolcasfa szintet. Mivel az út egyenesekből és ívekből áll, így az interpolálandó pontokat külön tömbökben kezeltem, majd egyesítettem és végrehajtottam az interpolációt a mesh modellen. Ennek a megoldásnak előnye, hogy a mesh modell egyszer kerül csak betöltésre. A 3.3. ábra egy csatorna fedlap környezetét mutatja a CRG modellen, illetve az áttekintő ábrán a további csatornafedlapok is kivehetők.

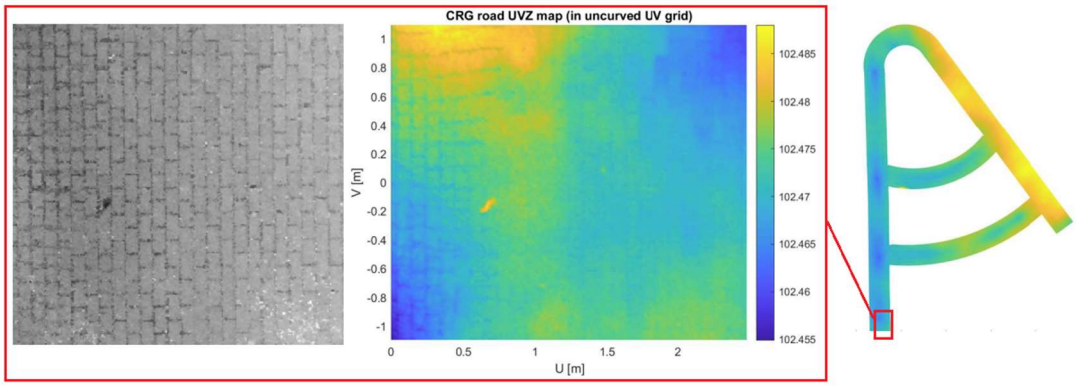


3.3. ábra: BME ST épület melletti parkoló egyik csatorna fedele

Fontos lehet egy-egy CRG készítés során utólagosan ellenőrizni, hogy az objektumok kellően részletesek-e, mert ha nem, érdemes újabb, pontosabb mesh számítás futtatása (ha lehetséges a felbontás növelése). Ennek a területnek a feldolgozása is igazolta, hogy a szimulációs környezetek részére is készíthető olyan állomány, mely visszaadja a felületi egyenetlenségeket és így akár járműdinamikai szimulációkat is támogathat.

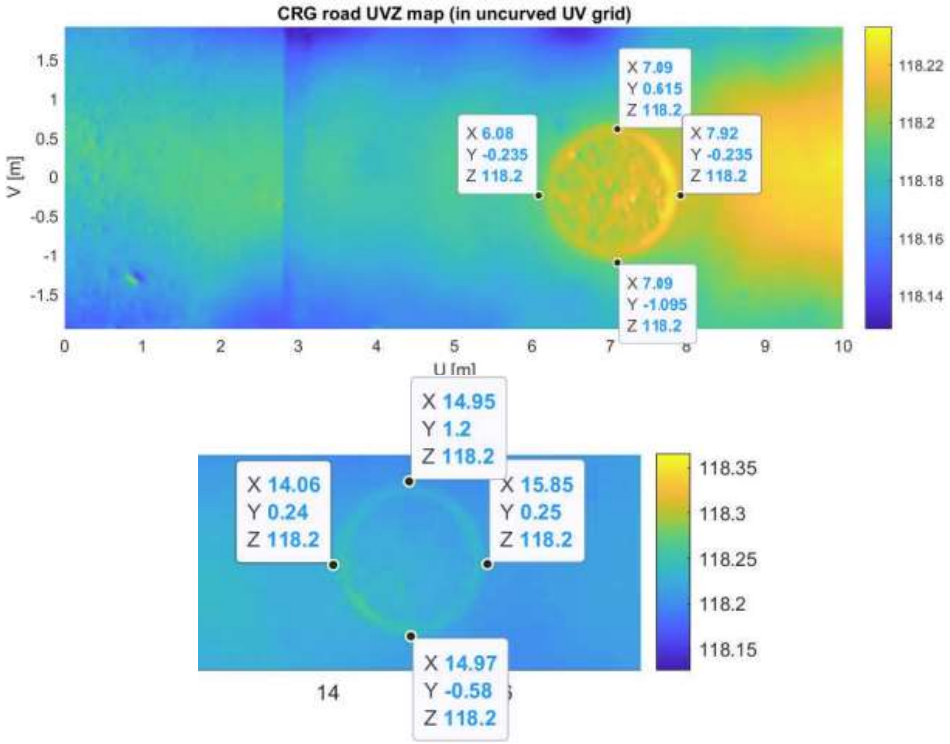
3.3. Észrevételek TLS felmérés esetén

A földi lézerszkennelés segítségével a pontfelhő kellően sűrű ahhoz, hogy apróbb változások is észrevehetők legyenek pl.: egy 10 × 20 cm-eres térkővel lerakott útszakaszon a térkövek is felismerhetők. A 3.4. ábrán látható kimetszés 2.2 × 2.5 méteres, a pontfelhő vágat ~45.000 pontot tartalmaz, erre a területre eső pontsűrűség 0,81 pont/cm2, a területet fedő mesh modell ~115.000 háromszögből áll.



3.4. ábra: BME ST parkoló egyrészlete

A földi lézerszkennelés során jellemzően több álláspontból készül a felmérés. A CRG modellek esetén azt tapasztaltam, hogy ahol nem állt rendelkezésre elegendő hely és emiatt az álláspont az út felületén került elhelyezésre, ott a modellen körök formájában megjelent az álláspont helye (korábban bemutatott 2.4. ábrán is megjelenik). A jelölt pontok segítségével a kör átmérője meghatározható ~1.75 méter, a színezés alapján pedig jól kivehető, hogy a kör az útfelülethez képest jelentős magasság különbséget mutat. Észrevehető az is, hogy ez a magasságkülönbség nem csak a körvonalon, de a körön belül is megfigyelhető (3.5. ábra). Valószínűleg a szkenner ilyen szélsőséges magassági szögnél már zajos pontfelhőt készít, ennek vizsgálata jelen kutatásnak nem tárgya, cél viszont a zajos állományrészek elhagyása.



3.5. ábra: BME K épület melletti szakaszon az álláspontok megjelenése körformájában

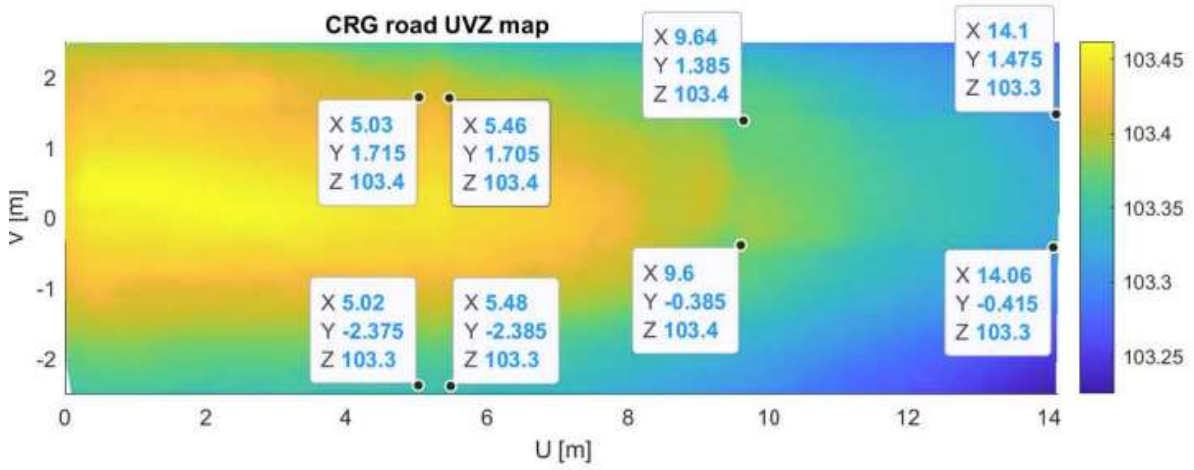
A hiba kiküszöbölésére lehetőség, hogy még a nyers pontfelhőadatok összeillesztése elött kivágjuk a körbe eső pontokat, majd ezt követően illesztjük össze a pontfelhőket. Egy befoglaló négyzet vágás segítségével meghatározható a környezetében lévő minimum, és ahhoz képest a maximális magasság, ami az álláspont körüli körben mérhető.

3.4. MLS felmérés

Rendelkezésemre álltak mobil lézerszkenner rendszer (MLS) által készített felmérések is a BME környékéről (3.6. ábra). Ezen felmérések esetén is kerestem olyan szakaszt, ahol megjelenhetnek az útegyenletlenségek. A Bertalan Lajos utca esetén találtam útfoltozásokat (a felmérés óta már javították), ebből egyet kiragadtam és modelleztem.

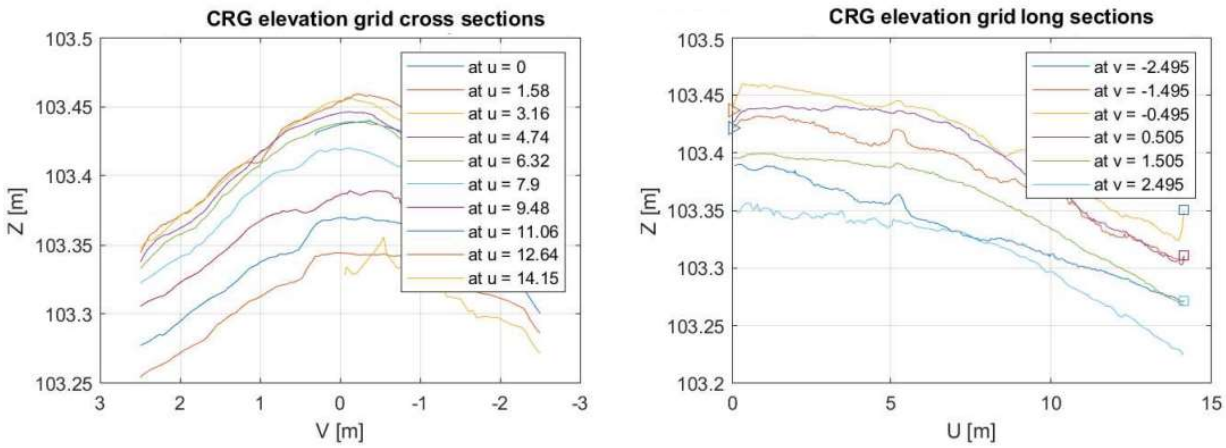
3.6. ábra: MLS felmérésből származó pontfelhő

A pontfelhő intenzitásképén láthatók az útjavítások, foltozások; ezek az út kivágott CRG modelljén is megjelennek. A 3.7. ábrán a jobboldali 4 darab pont jelöli a pontfelhőn látható úttengelyre merőleges felület változást, a baloldali 4 darab pont pedig a pontfelhőn látható világos színű foltozástól jobbra lévő útjavítást.



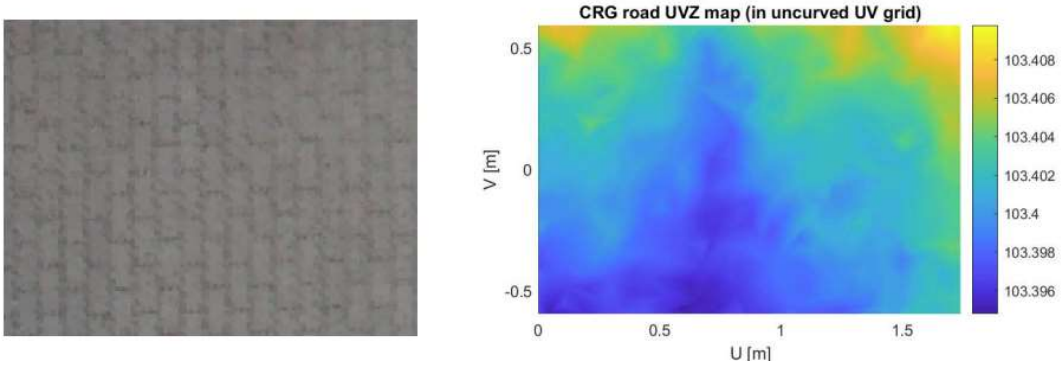
3.7. ábra: MLS felmérésből származó CRG modellrészlet

A pontfelhő intenzitásképén nem látszik, de CRG modell jól mutatja az úttengelyben tapasztalható kiemelkedést (bakhát). A 3.8. számú ábrán láthatók a CRG modell által generált hossz és keresztszelvények.



3.8. ábra: CRG generált hossz- és keresztszelvények

A keresztszelvényeken jól látható a tetőszelvény, illetve a felület egyenletlensége. Mivel ezen az útszakaszon a keresett útfelület javítás kevésbé látható ezért kerestem egy olyan szakaszt, ahol a felület változása jobban észlelhető. Cél, hogy ellenőrizzem, MLS esetén is elegendő-e a pontfelhő sűrűsége ahhoz, hogy apróbb felületi változások megjelenjenek a CRG modellben, a feldolgozás után.



3.9. ábra: MLS térkő CRG

Az 3.9. számú ábrán látható, hogy a pontfelhő intenzitásértékei visszaadja a térkő mintázatát (különböző anyagtulajdonság, visszaverődés), de ez már a CRG modellről nem mondható el. Azonban ennek a részletnek jól látható az egész felület magasság változása (menetirány szerinti oldalesés változása balról-jobbra ~10mm). Ezalapján lehet következtetni arra, hogy maga az út geometriája jól kivehető MLS felmérésekből is, azonban a felület változása (pl.: jó minőségű aszfalt burkolat és jó minőségű térkőburkolat között) nem feltétlen.

4. KONKLÚZIÓ

Tanulmányoztam a hivatalos ASAM OpenCRG dokumentációt, amely egy útszakaszt leíró fájl készítéséhez szükséges adatformátumot ismerteti. Felmértük a BME K épület melletti útszakaszt földi lézerszkenner segítségével, ennek pontfelhőjéből készítettem egy olyan felületmodellt, amely a lehető legjobb felbontást adja vissza; az általam használt számítógép (i5-8300H, 16 GB memória) kapacitásait kihasználva. Nyilvános könyvtárakat felhasználva Python programozási nyelvben írtam egy felületmodellen történő interpolációs eljárást. Ezt az eljárást úgy határoztam meg, hogy felhasználható legyen különböző utak/sávok leírására. Eredményül egy félautomatikus megoldást kaptam, amelynek bemenetei a pontfelhő, a felületmodell, illetve a vizsgált sáv referencia vonalát leíró adatok. Mértem a számítások lefutási idejét, egy 30 méter hosszú, 5 méter széles útszakaszt tartalmazó felületmodellen való centiméteres rácsháló menti interpoláció ideje modell betöltéssel együtt ~20-30 másodpercre adódott a vizsgálatok során használt számítógépemen.

MLS pontfelhőből keletkezett OpenCRG modell esetén az alacsonyabb (TLS-hez viszonyítva) pontsűrűség miatt részletességvesztést tapasztaltam. Apróbb eltérést adó útburkolatok (pl.: térkövek) nem feltétlen jelennek meg a CRG modellben. Azonban az út magassági egyenletlensége, dőlése, hosszesés változása a modellen jól érzékelhető. Az MLS technológiával rövid alatt nagy mennyiségű adat gyűjthető, akár a legnagyobb felbontást és pontosságot biztosító felmérési beállítások és körülmények között is.

Igazoltam, hogy nagyfelbontású és nagy pontosságú lézerszkennelt pontfelhőkből olyan CRG állományok vezethetők le, melyek nemcsak önvezető szimulációs, de akár járműdinamikai szimulációs alkalmazásokat is kiszolgálhatnak, hiszen visszaadják az útfelület egyenetlenségeit.

TLS pontfelhőből készíttetett OpenCRG modell esetén fontos tanulság, hogy az álláspontok környezetében tapasztalható magassági anomáliákat a CRG modell előállítása során kezelni kell. Ebből arra következtetésre juthatunk, hogy útszakaszok felmérése során amennyiben lehetőség adódik rá, a földi lézerszkennelés álláspontjait olyan helyekre helyezzük, amelyek nem értinik a felmérendő útszakaszt. Ezzel elkerülve az OpenCRG során az álláspontok helyén keletkezett körmenti magasság eltérést.

4.1. Kitekintés a kutatás folytatására

Érdemes olyan járműdinamikai szimulációk vizsgálata, mely ugyanazon a szakaszon több felméréstípusból eredeztetett CRG modellt használ fel. Ebből lehetne következtetni arra, hogy a modellt használó szoftverek milyen pontfelhő sűrűségi igényekkel rendelkeznek.

Későbbiekben az interpolációs módszert is bővíteném az úttervezésben gyakran használt klotoid átmenetiívvel, illetve a kötöttpályás közlekedéstervezésben gyakran használt koszinusz átmenetiívvel.

Jelenleg a felületmodell készítés, interpoláció, illetve az CRG fájl készítése automatikus vagy félautomatikus. Későbbiekben vizsgálnám a sávok és sávtengelyek automatikus detektálásának lehetőségeit a pontfelhő segítségével. Ezen felül érdemes lehet a felületmodell készítését például Python környezetben megvalósítani (erre lehetséges könyvtár az Open3D) a folyamat egyszerűsítése érdekében. Így végeredményül elhagyva más programok használatának igényét.

IRODALOMJEGYZÉK

[1] https://www.asam.net/standards/detail/opendrive (2021.10.03)

[2] https://www.cloudcompare.org (2021.10.03)

[3] Schmeitz, A., Versteden, W., & Eguchi, T. (2011). Road load simulation using the MF-Swift tire and OpenCRG road model (No. 2011-01-0190). SAE Technical Paper.

[4] Ormándi, T., Varga, B., & Tettamanti, T. (2021). Estimating Vehicle Suspension Characteristics for Digital Twin Creation with Genetic Algorithm. Periodica Polytechnica Transportation Engineering, 49(3), 231-241.

[5] Potó, V., Csepinszky, A., & Barsi, Á. (2018). REPRESENTING ROAD RELATED LASERSCANNED DATA IN CURVED REGULAR GRID: A SUPPORT TO AUTONOMOUS VEHICLES. International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, 42(2).

[6] Barsi, Á., Potó, V., Lógó, J. M., & Krausz, N. (2020). Creating an OpenDRIVE Model of the Campus of the Budapest University of Technology and Economics for Automotive Simulations. Periodica Polytechnica Civil Engineering, 64(4), 1269-1274.

[7] Kazhdan, M., Bolitho, M., & Hoppe, H. (2006, June). Poisson surface reconstruction. In Proceedings of the fourth Eurographics symposium on Geometry processing (Vol. 7).

MELLÉKLETEK

POINT CLOUD TO OPENCRG

Requirements:

- Point Cloud as mesh modell of the road.
  `Calculate Mesh from point cloud. Programs: CloudCompare, Meshlab, open3d python module etc..`
- Road properties like alignment start-end position, road width, origo (optional)
- Used 3.7.0 Python packages:
  ```
  numpy version == 1.18.5
  trimesh version == 3.9.34
  scipy version == 1.7.0
  ```

Import packages.

In [2]:
```
import numpy as np
import trimesh
from scipy.interpolate import griddata
```

Calculate the interpolation with straight road alignment.

In [3]:
```
def straight_road(mesh_path, P1, P2, road_width, name, uinc=0.01,
                  vinc=0.01):
    """ The function create a straight grid with the length of
        the road and road width.
        The grid spaces defined by uinc, vinc.
        Interpolate over mesh with the created grid.

        Parameters:
            mesh_path (str): File direction to the triangulated mesh
            P1 (tuple): Road alignment start position as a tuple x,y
            P2 (tuple): Road alignment end position as a tuple x,y
            road_width (float): Road width
            name (str): File name
            uinc (float): Space between "v" vectors, default 0.01
            vinc (float): Space between "u" vectors, default 0.01

        Returns:
            x,y,z (list): Returns numpy.array of "x", "y" and "z".
            Where "x" and "y" define the grid, "z" define the
            elevation datas at the grid points.
    """
    # Load the mesh
    tri_mesh = trimesh.load(mesh_path)

    x1, y1 = P1
    x2, y2 = P2


    # Calculate the "u" vector length
    len_u = np.sqrt((x1-x2)**2 +(y1-y2)**2)
```

```
    # Calculate the angle of the "u" vector
    angle = np.arctan2((y1-y2),(x1-x2))-np.radians(180)

    # Define the x and y vectors
    x_array = np.arange(0, len_u, uinc)
    y_array = np.arange(-road_width/2, road_width/2, vinc)

    # Create the grid with x and y vector
    X, Y = np.meshgrid(x_array, y_array)

    # Rotate the calculated grid X and Y values with the "u" vector angle
    X_ = np.cos(angle)*(X) - np.sin(angle)*(Y)
    Y_ = np.sin(angle)*(X) + np.cos(angle)*(Y)


    # The start position was 0,-road_width/2
    # Push the rotated meshgrid to the road alignment start position
    x = X_ + x1
    y = Y_ + y1

    # Calculate the interpolation with the meshgrid over the loaded triangular mesh
    z = griddata((tri_mesh.vertices[:, 0], tri_mesh.vertices[:, 1]),
                 tri_mesh.vertices[:, 2], (x, y))

    # Save the x,y and z data with precision 0.0001
    np.savetxt(f"/{name}_x.txt", x, fmt="%1.4f")
    np.savetxt(f"/{name}_y.txt", y, fmt="%1.4f")
    np.savetxt(f"/{name}_z.txt", z, fmt="%1.4f")

    return x, y, z
```

Calculate the interpolation with curved road alignment.

In [4]:

```
def curved_road(mesh_path, P1, P2, origo, road_width, name, uinc=0.01, vinc=0.01):
    """ The function create a curved grid with the length of
        the road and road width.
        The grid spaces defined by uinc, vinc.
        Interpolate over mesh with the created grid.

        Parameters:
            mesh_path (str): File direction to the triangulated mesh
            P1 (tuple): Road alignment start position as a tuple x,y
            P2 (tuple): Road alignment end position as a tuple x,y
            origo (tuple): Road alignment radius origo as tuple x,y
            road_width (float): Road width
            name (str): File name
            uinc (float): Space between "v" vectors, default 0.01
            vinc (float): Space between "u" vectors, default 0.01

        Returns:
            x,y,z (list): Returns numpy.array of "x", "y" and "z".
            Where "x" and "y" define the grid, "z" define the
            elevation datas at the grid points.
    """
    # Load the mesh
    tri_mesh = trimesh.load(mesh_path)

    # Calculate the road alignment radius
    R = np.sqrt((origo[0]-P1[0])**2 + (origo[1]-P1[1])**2)

    # Calculate the start and end theta values [rad]
    theta_start = np.arcsin((P1[1] - origo[1])/R)
    theta_end = np.arcsin((P2[1] - origo[1])/R)
```

```
    # Calculate the value of phi which gives an arc length uinc on
    #an arc of radius R
    phi = uinc/R

    # Define the r and theta vectors
    r_array = np.arange(R-road_width/2, R+road_width/2, vinc)
    theta_array = np.arange(theta_start, theta_end, phi)

    # Create grid with r and theta vector
    r, theta = np.meshgrid(r_array, theta_array)

    # Transform from polarcoordinate system into Cartesian coordinate system
    x = np.cos(theta) * r + origo[0]
    y = np.sin(theta) * r + origo[1]

    # Calculate the interpolation with the meshgrid over the loaded triangular mesh
    z = griddata((tri_mesh.vertices[:, 0], tri_mesh.vertices[:, 1]),
                 tri_mesh.vertices[:, 2], (x, y))

    # Calculate curvature for each rows
    curvature = [i/R for i in range(len(z))]

    # Save the x,y,z and curvature data with precision 0.0001
    np.savetxt(f"{name}_x.txt", x, fmt="%1.4f")
    np.savetxt(f"{name}_y.txt", y, fmt="%1.4f")
    np.savetxt(f"{name}_z.txt", z, fmt="%1.4f")
    np.savetxt(f"{name}_curve.txt", curvature, fmt="%1.4f")

    return x, y, z
```