Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Mészáros József

BARLANGÁSZAT 3D-BEN

KONZULENS

Rajacsics Tamás

BUDAPEST, 2011

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Mészáros József**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2011.12.13.

.....

Mészáros József

1. Bevezetés

Egyetemi tanulmányaim alatt több olyan tevékenységet próbáltam végezni, amelyek színesebbé tették olykor szürke hétköznapjaimat. Az egyik ilyen a barlangászat volt, amely mára már a legfontosabb és legkedvesebb hobbijaim közé tartozik. Az első néhány hónapban még nem voltam annyira lelkes rajongója a barlangászatnak, de az utóbbi években egyre inkább magával ragadott.

A barátaimmal sorra jártuk a hazai barlangokat, amelyekről egyre többet és többet szerettem volna tudni. Szerencsére az egyik barlangász egyesület, a Szegedi Karszt- és Barlangkutató Egyesület éppen akkor indított egy tanfolyamot, amelyre lelkes amatőrök jelentkezését várták. A tanfolyam alatt egyre többet tudtam meg a barlangászat különböző területeiről, amelyek közül legjobban a barlangjárás technikai részletei és a térképezés érdekeltek. Betekintést nyerhettem a térképek és a térképezés alapjaiba, sőt a barlangokban meg kellett tanulom közlekedni A4-es lapra nyomtatott térképek alapján. A barlangokból megfáradtan hazatérve a barátaimnak ugyanazon a térképen tudtam csak megmutatni, hogy a barlangnak melyik részét jártam be. Ők persze nehezen tudták elképzelni az egészet, hiszen a kép, amelyet láttak a barlangról készült vetület volt.

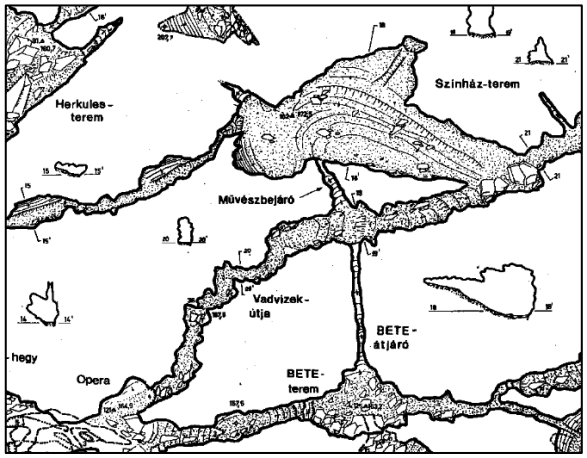
A tanfolyam elvégzése után azon kezdtem el gondolkodni, hogy mivel tehetném jobbá a hazai barlangászatot. Akkor kezdtem el gondolkodni barlangok háromdimenziós megjelenítésének lehetőségeiről és annak technikai részleteiről. Hamar kiderült számomra, hogy a témával nem igazán foglalkozott még senki hazánkban. Úgy döntöttem, hogy talán az első úttörőként megpróbálok valami használható módszert, vagy programot letenni a hazai és külföldi barlangász társadalom elé.

Elsőként a Barlangtani Intézethez fordultam, majd segítőkész barlangászokhoz, akik válaszoltak és kérdéseimre és lehetővé tették, hogy megtegyem az első lépéseket. A barlangok háromdimenziós megjelenítéséhez mindenképpen szükség volt alapjaiban új mérési módszerekre. A diplomámban ezeket a mérési módszereket mutatom be elsőként, majd az eredményeket feldolgozó algoritmusokat és végül az implementált grafikus programokat.

2. Barlangok felmérése régen és ma

A barlangok világa rendkívül érdekes és veszélyes világ, amelyet már sokan próbáltak meghódítani. A barlangászoknak abszolút nincs könnyű dolguk, amikor a föld alatt próbálnak meg eljutni az egyik pontból a másikba, hiszen a két pont között kialudhat a lámpájuk, a kiáltásuk pedig pár méter után a semmibe vész. Képzelje csak el a kedves Olvasó milyen végtelenül változatos és komplex geometriával rendelkezik egy barlang, amely szépsége ellenére éppoly kegyetlen lehet.

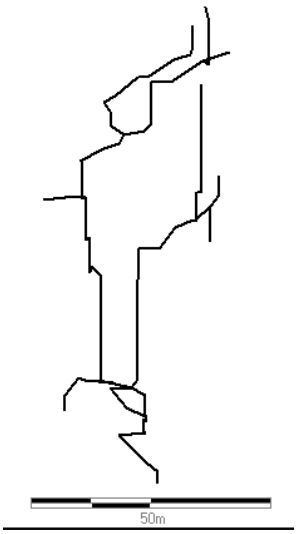
Azoknak a barlangászoknak, akik nem ismerik a barlang összes szikláját és túraútvonalát, szükségük van egy kis segítségre: egy térképre, amellyel már biztonságosan közlekedhetnek a járatokban. A térkép a barlang kétdimenziós ábrázolása, amely az üregrendszer térbeli jellegéről, annak szelvényeiről a legtöbb esetben jól áttekinthető információt nyújt. A következő ábrán az egyik leglátogatottabb budapesti barlang térképének látjuk egy részletét.



1. ábra Mátyás-hegyi barlang térképrészlete

A térkép elkészítéséhez pontosan fel kell mérnünk a barlangot, majd a mérési eredményeket felhasználva elkészíteni a kétdimenziós térképet. A barlangok bonyolult, szabálytalan térbeli felületekkel határolt üregrendszerek, amelyek felmérése és ábrázolása speciális feladatot jelent. A barlang méréstan vagy más néven szpeleometria a földalatti helymeghatározás tudományának barlangokra alkalmazott része. A barlangfelmérésben elsősorban a bánya méréstanból átvett módszereket használnak napjainkban is.

A felmérés lényege, hogy a barlangjáratok tengelyvonalában egy mért adatokkal meghatározott sokszögvonalat veszünk fel, amely a térképkészítés alapjául szolgál. Ez a sokszögvonal vagy más néven poligon lesz a térkép „váza", amire későbbi mérésekkel felvesszük a barlang kontúrvonalát és ábrázolandó járatkitöltő elemeit. A sokszögvonalat a járatok középvonalában vezetjük végig a legtöbb esetben. A sokszögvonal hosszal, irányszöggel és lejtszöggel megadott, egymáshoz láncszerűen kapcsolódó egyenes szakaszok sora. A legkönnyebben talán vektorok folyamatos összeadásával lehetne ezt a folyamatot szemléltetni. Az egyes szakaszokat sokszögoldalaknak (poligonoldal), a szakaszok töréspontjait és végpontjait pedig sokszögpontoknak (poligonpont, mérési pont) nevezzük. A 2. ábra a Bükkben található 100 méter mély Almási-zsomboly sokszögvonalát mutatja.



2. ábra Az Almási-zsomboly poligonja

Az eddig alkalmazott módszert a barlangászok huzagolásos módszernek nevezik. Huzagolásos módszerrel történő mérés esetén a járatok középvonalában poligonzsinórt feszítünk ki, amelyek a sokszögvonal oldalait fogják képezni. A sokszögvonal így létrejött töréspontjai lesznek a mérési pontok. A lejtszög és irányszög méréséhez pedig függőkompaszt használnak. A módszerről részletes leírás a függelékben található.

A huzagolásos módszernél egy szakaszon vétett hiba a sokszögvonal további elemeire is továbbterjed, így egy nagyobb barlang esetén az eredmény pontatlan lehet. Maga a mérési módszer nagyon idő- és emberigényes, hiszen egy zsinórt és szögmérőket használva kézzel vesszük fel és jegyezzük le egy sokszögvonal adatait. A mérés során könnyen hibákat véthetünk, amelyek aztán végiggyűrűznek egészen az

utolsó mérési pontig. Képzelje csak el a kedves Olvasó, hogy az első mérési pontnál elkövetett irányszög hiba minden többi mérési pontnál meg fog jelenni, amely egy nagyobb barlang esetén pontatlan sokszögvonalat eredményezhet.

Szerencsére az utóbbi években hazánkban is elterjedt a huzagolásos technika modern változata, amely a poligonzsinór és a lejtszögmérő helyett egy DistoX nevű digitális mérőeszközt használ. A DistoX egy lézeres távolságmérő továbbfejlesztett változata, amelyet egy svájci barlangász fejlesztett ki.

3. ábra DistoX és PocketTopo

A DistoX egy méréssel három adatot szolgáltat (irányszög, lejtszög, távolság) és ezeket továbbítja vezeték nélküli kapcsolaton keresztül. Létezik egy PocketTopo nevű Pocket PC-n futó alkalmazás, amely a DistoX vezeték nélküli kapcsolaton átküldött adatait tárolja és jeleníti meg. Az alkalmazásban az adatok a mérési jegyzőkönyv táblázatban jelennek meg, valamint a poligonvonal is megjeleníthető. A felmérés így papír nélkül jelentősen gyorsabban és pontosabban végezhető, mint a hagyományos „kézi” módszerrel.

A sokszögvonalról már tudjuk, hogy a barlang tengelyvonalában halad, de nem árul el semmit a felmért járat valós formájáról. Egy vonalra ránézve nem tudjuk eldönteni, hogy egy szűk járatról vagy egy hatalmas teremről van szó. Nem tudjuk elképzelni a hatalmas tereket, a szelvények formáját. A sokszögvonal nem árul el semmit a formakincsről és a jellemző kőzetekről sem.

A huzagolásos módszer hiányosságai miatt új mérési módszerek kidolgozása mellett döntöttem, amelyek lehetővé teszik, hogy a barlangokat 3D-ben jelenítsük meg. A mérési módszer eredményeinek megjelenítéséhez és feldolgozásához szükség volt egy programra is, amely a Tórusz nevet kapta.

2.1 Új mérési módszerek

Olyan mérési módszereket próbáltam kidolgozni, amelyek bárki számára gyorsan és könnyedén elvégezhetőek, nem igényelnek semmilyen matematikai háttértudást és a lehető legkevesebb mérési pontból a legjobb modellt állítják elő. Ha ezek nem teljesülnek, akkor az új módszerek egészen biztosan nem fognak elterjedni, hiszen senki sem szeret egy hideg és sötét barlangban órákat eltölteni egy lézeres távolságmérő társaságában.

Az első használható algoritmus ($4P^2$) kidolgozása után felkerestem a hazai barlangászat hivatalos szervezetét, a Magyar Karszt- és Barlangkutató Társulatot (a továbbiakban 'MKBT'). Miután bemutattam a Társulat részére előzetes eredményeimet kikértem a véleményüket, hogy a módszer számukra mennyire használható a gyakorlatban, illetve hogy szerintük mi lenne a leghatékonyabb mérés. További egyeztetéseket és megbeszéléseket követően újabb módszerek kerültek kidolgozásra (NP^2, SPCR), amelyeket később implementáltam és kipróbáltam az MKBT-től kapott DistoX-el a Pálvölgyi-Mátyás-hegyi barlangrendszerben[1], a Szemlő-hegyi barlangban és a Meta-barlangban.

Az új mérési módszerek mindegyikének az az alapja, hogy a poligonpontból a barlang kiterjedését több irányba is lemérjük, majd az így kapott pontfelhőt különböző módszerekkel egy háromszöghálóvá alakítjuk. Az eredmény egy háromdimenziós modell, amely élethűen adja vissza a felmért járat formáját.

A módszerek bemenetét a DistoX által szolgáltatott adatok képzik, amelyek hosszal, lejtszöggel és irányszöggel vannak megadva. Az algoritmusok bemenete Descartes-koordinátarendszerben megadott pontok, ezért a DistoX gömbi koordinátarendszeréről át kell térnünk. Egy olyan Descartes-koordinátarendszerre térünk át, amelynek *y* tengelye északi irányba, *x* tengelye kelet felé, a *z* tengelye pedig a Föld középpontjával ellentétes irányba mutat. Meg kell határoznunk egy kezdő mérési pontot, amelynek a koordinátái *x=0, y=0, z=0* lesznek. A mérési pontok koordinátáit az alábbiak szerint számolhatjuk ki:

$$x_n = x_k + \cos\delta * \cos\varphi * t$$

$$y_n = y_k + \sin\delta * \cos\varphi * t$$

$$z_n = z_k + \sin\delta * t$$

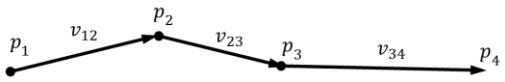
[1] 2011. december 11. óta már a Szépvölgyi-barlangrendszer, az ország leghosszabb barlangja.

ahol *n* az adott mérési pont száma, *k* a poligonoldal kezdőpontja, *t* a poligonoldal hossza, φ a lejtszög, míg δ az irányszög.
A következő fejezetekben az új mérési módszereket mutatom be.

2.1.1 4P^2

Az elsőként bemutatott módszer a 4 point passage vagy röviden 4P^2 nevet kapta, mivel a poligonpontból négy irányba mérjük le a barlang kiterjedését. Ezt a módszert mutattam be elsőként az MKBT-nek. A 4P^2 egy elég egyszerű algoritmusra épül, ami már használható eredményt ad, de még mindig nem alkalmas arra, hogy igazán formahű modelleket szolgáltasson. A külföldi barlangászok által használt programok, amelyek már 3D-s funkciókra épülnek (Survex, Compass, Therion) egytől egyig az itt bemutatott módszerre vagy annak egy egyszerűbb változatára épülnek.

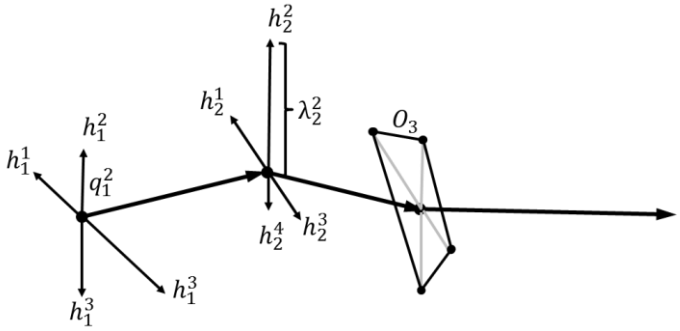
Az algoritmust legjobban talán egy giliszta segítségével lehet megérteni. Tegyük fel, hogy a feladatunk olyan nagyjából hengerformájú (gilisztára emlékeztető) folyosó háromdimenziós felmérése, amelyhez rendelkezésre áll a korábban felvett poligon is, amely a folyosó középvonalában halad. A poligont tekinthetjük a giliszta nem létező gerincének, a poligonpontokat pedig a csigolyáknak. A folyosó kiterjedését tetszőleges pontból nem tudjuk lemérni, kizárólag a poligonpontokat tudjuk használni erre a célra. Minden poligonpontból négy irányba lemérjük a folyosó falának és a poligonpontnak a távolságát, úgy hogy ezek a pontok a középvonalra merőlegesek legyenek. Ha ezt a négy pontot összekötjük, akkor a folyosónak egy közelítő keresztszelvényét kapjuk meg, amit tekinthetünk a giliszta egy gyűrűjének. A keresztszelvényekre egy háromszögekből álló köpenyt húzva kapjuk meg a már teljes gilisztát. A következő bekezdésekben található a fentiek formális leírása. Legyen adott a poligon $\{\mathrm{p_i} = (\mathrm{x_i}, \mathrm{y_i}, \mathrm{z_i})\}_{\mathrm{i=1}}^{\mathrm{m}}$ poligonpontokkal, mint ahogy azt a 4.ábra is mutatja. A poligonpontok legyenek rendezve, azaz p_i poligonpont után a p_{i+1} indexű poligon legyen a következő a sokszögvonalban. Legyen továbbá $v_{i,i+1} = p_{i+1} - p_i$ az i. poligonpontból a következő poligonpontba mutató vektor.



4. ábra: A kiindulási poligon

Minden egyes p_i poligonpontból felveszünk 4 további pontot, amelyek a poligonpont és a barlang falának távolságát mutatják az adott irányba. Nevezzük el

ezeket a pontokat határpontoknak és jelölje az *i.* ponthoz tartozó határpontokat $\{h_i^j = (u_i^j, v_i^j, w_i^j)\}_{j=1}^4$. Jelölje $\{(\lambda_i^j)\}_{j=1_i}^4$ a p_i poligonpont és a hozzá tartozó határpontok közötti távolságot, azaz $\lambda_i^j = |h_i^j - p_i|$. Nevezzük el továbbá profilvektornak egy határpont és a poligonpont különbségeként előálló vektort. A *j.* határpont és az *i.* poligonpont által meghatározott profilvektort jelölje $q_i^j = h_i^j - p_i$. A határpontok összeköttetéseként előálló Jordan-poligont (egyszerű poligont) nevezzük el szelvénynek. Jelölje az *i.* poligonpontból felmért határpontokhoz tartozó szelvényt $O_i = (h_i^1, h_i^2, h_i^3, h_i^4)$, amely 4P^2 esetén egy olyan négyszög, amely mindig az S_i síkon van.



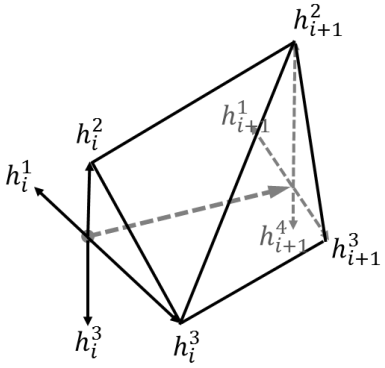
5. ábra A poligonpontok, határpontok és profilvektorok

A határpontokra a következő megkötéseket tesszük:

- A négy határpont mindig egy síkban van;
- A határpontok a síkon rendezettek, a sorrend minden sík esetén ugyanaz;
- A szomszédos határpontok által bezárt szög pontosan 90°.

Jelölje a p_i poligonponthoz tartozó határpontok által meghatározott síkot S_i, amelynek n_i legyen az egység hosszú normálvektora.

Az így definiált határpontokból nagyon könnyen tudunk háromszöghálót készíteni, hiszen csak a megfelelő indexű határpontokat kell összekötnünk.



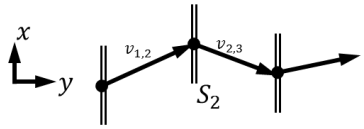
6. ábra Háromszögesítés

Vegyük példának p_i és a p_{i+1} pontok közötti poligonoldal háromszöghálójának elkészítését. Az első háromszöget a $(h_i^j, h_{i+1}^j, h_i^{j+1})$ hármas határozza meg, míg a következőt a $\left(h_{i+1}^j, h_i^{j+1}, h_{i+1}^{j+1}\right)$, majd a $\left(h_i^{j+1}, h_{i+1}^{j+1}, h_i^{j+2}\right)$ következik. A 6. ábrán is szemléltetett algoritmus a grafikában triangle strip néven ismert triangulációs eljárás, amelynek a határpontokat $h_i^j, h_{i+1}^j, h_{i+1}^j$ ... sorrendben adjuk át. A trianguláció végén 8 háromszöget kapunk. Természetesen ezt az eljárást az összes poligonoldalra el kell végeznünk.

A legegyszerűbb konstrukció, amely a határpontokra kimondott megkötéseknek eleget tesz, ha a határpontokat úgy értelmezzük, mint a poligonponttól fel, le, jobbra és balra lévő pontok. Tehát ha például

$$(u_i^1, v_i^1, w_i^1) = (x_i - \lambda_i^1, y_i, z_i)$$
$$(u_i^2, v_i^2, w_i^2) = (x_i, y_i, z_i + \lambda_i^2)$$
$$(u_i^3, v_i^3, w_i^3) = (x_i + \lambda_i^3, y_i, z_i)$$
$$(u_i^4, v_i^4, w_i^4) = (x_i, y_i, z_i - \lambda_i^4)$$

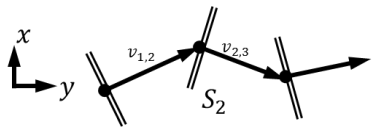
Ezzel a konstrukcióval az a baj, hogy az összes sík normálvektora ugyanaz lesz, a síkok ugyanúgy fognak állni, attól függetlenül, hogy a haladási irány folyton változik. Ezt szemlélteti a következő ábra, amely a z tengellyel ellentétes irányból, gyakorlatilag felülnézetből mutatja a poligonpontokat és a síkokat.



7. ábra: A haladási iránytól független síkok

Azt várnánk el, hogy S_i síkok orientációja változzon a haladási iránytól függően, például legyen haladási irányra merőleges. Ez azt jelenti, hogy az S_i normálvektora

párhuzamos a következő poligonszakasz vektorával, vagyis az n_i normálvektor párhuzamos a $v_{i,i+1}$ vektorral, amint azt a 8.ábra is szemlélteti.



8. ábra: A haladási iránytól függő síkok

A haladási irányra merőleges S_i vektoros egyenlete nagyon könnyen meghatározható, ha a normál vektorának a normalizált $\tilde{v}_{i,i+1}=\frac{v_{i,i+1}}{|v_{i,i+1}|}$ vektort választjuk:

$$\tilde{v}_{i,i+1} \cdot (X - p_i) = n_i$$

Az így definiált sík határpontjainak meghatározásához szükségünk van egy n_i vektorra merőleges egység hosszú $n_i{}^{\perp}$ vektorra. Az $n_i{}^{\perp}$ vektor felhasználásával a határpontok a következők:

$$h_i^1 = p_i + \lambda_i^1 n_i{}^{\perp}$$
$$h_i^2 = p_i + \lambda_i^2 n_i{}^{\natural}$$
$$h_i^3 = p_i - \lambda_i^3 n_i{}^{\perp}$$
$$h_i^4 = p_i - \lambda_i^4 n_i{}^{\natural}$$

ahol $n_i{}^{\natural}$ vektort az $n_i{}^{\perp}$ vektor n_i tengely körüli 90 fokos elforgatásával kapjuk, amelyhez a Rodrigues - formulát használjuk.

Az $n_i{}^{\perp}$ vektor n_i tengely körüli φ szögű elforgatásának eredménye az n_{rot} vektor lesz:

$$n_{rot} = n_i{}^{\perp} * cos\varphi + (n_i \times n_i{}^{\perp}) * \sin\varphi + n_i(n_i \cdot n_i{}^{\perp})(1 - \cos\varphi \cdot)$$

A fenti egyenletet megadhatjuk mátrixos formában is:

$$C = \cos\varphi, \mathrm{S} = \sin\varphi$$

$$A = \begin{bmatrix} C(1-{n_i}_x{}^2)+{n_i}_x & {n_i}_x{n_i}_y(1-C)+S{n_i}_z & {n_i}_x{n_i}_z(1-C)-S{n_i}_y \\ {n_i}_y{n_i}_x(1-C)-S{n_i}_z & C\left(1-{n_i}_y{}^2\right)+{n_i}_y{}^2 & {n_i}_x{n_i}_z(1-C)+S{n_i}_x \\ {n_i}_z{n_i}_x(1-C)+S{n_i}_y & {n_i}_z{n_i}_y(1-C)-S{n_i}_x & C\left(1-{n_i}_z{}^2\right)+{n_i}_z{}^2 \end{bmatrix}$$

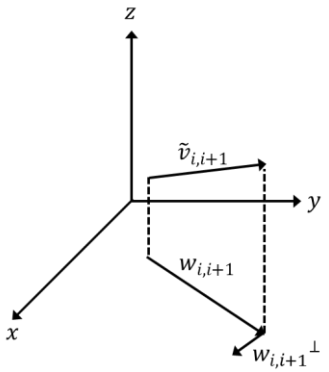
$$n_{rot} = n_i{}^{\perp} * A$$

Ezek alapján:

$$n_i{}^{\natural} = n_i{}^{\perp} * A, \varphi = 90°$$

Az egyetlen hátralévő feladat $n_i{}^{\perp}$ vektor meghatározása. Végtelen sok olyan vektor van, amely párhuzamos az n_i vektorra. Ezek közül kell választanunk egyet.

Első megoldásként vegyük a $\tilde{v}_{i,i+1}$ vektor $w_{i,i+1}$ kétdimenziós vetületét az xy síkon, határozzuk meg a vetületre merőleges ${w_{i,i+1}}^{\perp}$ vektort, amely $\tilde{v}_{i,i+1} = n_i$ vektorra is merőleges.



9. ábra $\tilde{v}_{i,i+1}$ vektor vetülete

Az egyik legismertebb külföldi barlangász program, a Compass első verziójában ezt a megoldást implementálták. A probléma ezzel az, hogy a vetület egy nulla hosszúságú vektor abban az esetben, ha az n_i vektor párhuzamos a z tengellyel.

Ennél egy sokkal egyszerűbb megoldás, ha veszünk egy tetszőleges r_i véletlen vektort, majd segítségével meghatározzuk az ${n_i}^{\perp}$ vektort:

$$ {n_i}^{\perp} = n_i \times r_i $$

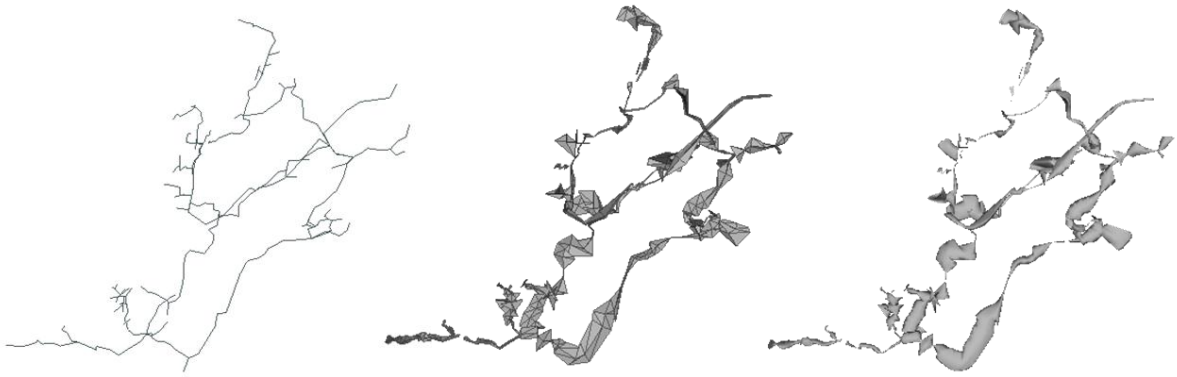
Arra kell csak figyelnünk, hogy a véletlen vektor ne legyen párhuzamos az n_i vektorral. Ha ez az eset áll fent, akkor generáljunk egy új véletlen vektort.

Most már a 4P^2 algoritmus minden lépését ismerjük.

2.1.1.1 Elemzés

A 4P^2 implementálása elkezdtem háromdimenziós mérési eredmények után kutatni az interneten, mivel akkor még nem végeztem saját méréseket. Azt tapasztaltam, hogy a külföldi programok nagy többsége a 4P^2-hez hasonló algoritmusokra épül.

Szerencsére találtam olyan méréseket is, amelyeknél meg volt adva a poligonponthoz tartozó négy határpont is. Az egyik ilyen mérés az észak-walesi Eglwys Faen nevű barlangban készült, amely a Llangattock hegység mészkőjében alakult ki. A 10. ábrán az eredeti poligont, a 4P^2 utáni eredményt illetve annak egy simított változatát láthatjuk.



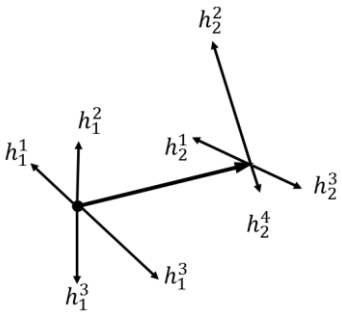
10. ábra Az eredeti poligon, a $4P^2$ és a simítás utáni eredmény

Ránézve a fenti ábrára a $4P^2$ már élvezhető képet ad a barlangról, de igazán élethű eredményt ez sem tud biztosítani.

A $4P^2$ egyik nagy hátránya, hogy a poligonpontból csak 4 irányba mérjük le a barlang kiterjedését. Ha élethű eredményt szeretnénk, akkor a 4 határpont túl kevés. Arra viszont jó, hogy egy járatot elnagyolva mutasson, de pontos és részletes méréseket nem végezhetünk vele.

Másik hátránya, hogy a határpontoknak egy síkban kell lenniük egymással 90° fokos szöget bezárva. Ez a megkötés gyakorlatilag alkalmatlanná teszi a $4P^2$ módszert arra, hogy barlangban használjuk. Drága és felesleges műszerek nélkül lehetetlen egyetlen DistoX-el a határpontokat egy síkba felvenni, ráadásul úgy, hogy ezek merőlegesek legyenek egymásra. Ha rendelkezésünkre állna egy olyan állvány, amellyel fel tudnánk venni egy síkba a határpontokat úgy, hogy azok merőlegesek legyenek egymásra, akkor is rendkívül hosszadalmas eljárásnak bizonyulna, hiszen ennek kivitelezése annyira lassítaná a mérést, hogy az első 3-4 poligonpontig jutnánk.

További hátránya, hogy a különböző síkok azonos indexű profilvektorainak azonos orientációval kell rendelkezniük. Erre azért van szükség, mert mindig az azonos indexű határpontokból és azok szomszédos pontjaiból készítünk háromszögeket. Ha ez nem így lenne, akkor minden esetben meg kellene határozni, hogy az S_i síkon lévő *k.* határpontnak az S_{i+1} síkon melyik a szemközti párja. Ha a profilvektorok orientációja nem egyezik, akkor egy csavart háromszöghálót kapunk, amelyben nagyobb eltérések esetén a háromszöget metszhetik egymást.



11. ábra Rosszul orientált profilvektorok

Vegyünk erre egy egyszerű példát. A 11. ábrán rosszul orientált profilvektorokat láthatunk. Ha most lefuttatnánk a triangulációt, akkor egy kissé csavart háromszöghálót kapnánk eredményül.

Az implementáció során ezt egy olyan referencia vektor segítségével oldottam meg, amelyet mindig csak az első poligonpontnál generáltam. A következő poligonpontok esetén ugyanazt a referencia vektort használtam. A referencia vektor és a $v_{i,i+1}$ vektor keresztszorzataként állítottam elő az első határpontot. Mivel a többi határpont az első határpont $v_{i,i+1}$ tengely körüli 90 fokos elforgatása, ezért azok orientációjával már nem kellett foglalkozni.

Összegezve a fentieket:

- látványos, de nem élethű modellt eredményez;
- nem alkalmazható barlangban;
- meglévő poligonok háromdimenziós kiegészítésére jó választás;
- gyors és nagyon egyszerű;
- pontos méréseket nem tesz lehetővé.

A következő fejezetben a 4P^2 továbbfejlesztett változatát szeretném bemutatni.

2.1.2 NP^2

A 4P^2 kezdeti sikerein felbuzdulva mutattam be a módszert az MKBT-nek, amely megnyerte tetszésüket, de továbbra is azon a véleményen voltak, hogy négy határponttal nem lehet pontos méréseket végezni. Azt nem állítom, hogy a 4P^2 teljesen használhatatlan, hiszen már meglévő poligonok háromdimenziós kiegészítéséhez egyszerű és gyors megoldást nyújt, de azt be kell látnunk, hogy szükség van annak továbbfejlesztésére. Az MKBT is hasonló álláspontón volt, így közösen

továbbfejlesztve elképzeléseinket, egy új módszert dolgoztunk ki, amely az N Point Passage, vagy röviden az NP^2 nevet kapta.

Már a neve is elárulja, hogy az NP^2 és az előző módszer közötti lényeges különbség, hogy egy poligonpontból az új módszert alkalmazva már tetszőlegesen sok határpontot vehetünk fel. Az előző fejezet giliszta hasonlatát ebben az esetben is használhatjuk annyi kiegészítéssel, hogy a közelítő keresztszelvényt már nem csak 4 pont határozza meg.

Három további fontos különbség van, amelyet mindenképpen szükséges kiemelni. Talán az egyik legfontosabb, hogy a határpontoknak nem kell egy síkba esniük. Arra figyelnünk kell, hogy a határpontok továbbra is egy szelvényt határoznak meg, így nagyon szétszórtan, véletlenszerűen nem helyezkedhetnek el a határpontok. Enyhítés a $4P^2$-hez képest, hogy a profilvektorok orientációjára semmilyen megkötést nem teszünk. Nem várjuk el, hogy ezek bármilyen relációban legyenek egymással. Arra sem kell a mérés során figyelemmel lenni, hogy a profilpontok egymás után sorrendezve kerüljenek felvételre.

Az előző fejezet első két különbsége pontosan azokat a megkötéseket oldja fel, amelyek megakadályozták, hogy a $4P^2$ algoritmust barlangban használjuk, hiszen a DistoX-el nem tudjuk a határpontokat úgy felvenni, hogy azok egy síkban legyenek és a profilvektorok egymásra merőlegesek legyenek. Az utolsóként említett különbség pedig még inkább alkalmassá teszi a módszert arra, hogy barlangban használják. Ha nem kell ügyelni a pontok sorrendjére, akkor gyorsabban halad a mérés, illetve fennáll annak lehetősége, hogy egy már felmért szakasz finomításához további határpontokat vegyünk fel utólag.

Az első használható implementáció után az MKBT-től kapott DistoX-el két mérést végeztem a Pálvölgyi-Mátyás-hegyi barlangrendszerben, amelyeknek az eredményét később mutatom be. Mindenesetre a barlang táróját elhagyva meggyőződtem arról, hogy az NP^2 valóban használható barlangban.

Az előző fejezetben bevezetett jelöléseket és elnevezéseket fogom használni. Nagyon fontos, hogy egy poligonpontból tetszőlegesen sok határpontot vehetek fel, így $\{h_i^j = (u_i^j, v_i^j, w_i^j)\}_{j=1}^{n}, O_i = (h_i^1, h_i^2, \dots, h_i^n)$, ahol n különböző érték lehet minden poligonpont esetén.

Az NP^2 egy olyan algoritmust használ, amelynek a bemenete a határpontok rendezett halmaza. Elsőként tehát a határpontokat és a hozzá tartozó profilvektorokat

kell rendezni. Ehhez mindenképpen szükség van egy síkra, ami alapján a pontokat rendezni lehet. Nem elég az, ha egy globális síkot alkalmazunk, hiszen a poligon haladási iránya folyton változik, ezért minden szelvénynél[2] egy új síkot veszünk fel, ami alapján a szelvény határpontjait rendezzük.

Ötletként felmerülhet, hogy használjunk erre egy olyan R_i síkot, amelynek normálvektora az egység hosszú $\tilde{v}_{i,i+1}=\frac{v_{i,i+1}}{|v_{i,i+1}|}$ vektor, egy pontja pedig a p_i poligonpont. Ez azért nem lesz minden esetben használható, mert elképzelhető hogy a határpontok a poligonponttól távol vannak vagy a határpontok által felmért szelvény közel sem merőleges a $\tilde{v}_{i,i+1}$ vektorra, sőt egy poligonpontból akár több szelvényt is felvehetünk. Fontos tehát, hogy az R_i síkot a szelvényhez igazítsuk és ne a poligonponthoz, illetve a $\tilde{v}_{i,i+1}$ vektorhoz. Feladatunk tehát a szelvény határpontjaihoz egy legjobban illeszkedő síkot találni, amire vetítve a határpontokon már elvégezhetjük a sorrendezést. Azért fontos, hogy a legjobban illeszkedő síkot használjuk, mert egy „megközelítőleg" jó sík rossz sorrendezéshez vezethet.

Azt gondolhatnánk, hogy akkor kapjuk a legjobb síkot, ha az átmegy a szelvény középpontján, azaz a határpontok átlagán. Jelölje az i. szelvény határpontjainak átlagát

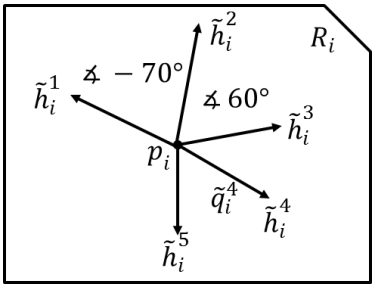
$$t_i = \frac{\sum_{k=1}^{n} h_i^k}{n}$$

Később igazolódik, hogy a fenti ötlet igenis helyénvaló.

Térjünk át egy kicsit arra, hogy egy sík milyen értelemben lehet a legjobb. Arra törekedünk, hogy egy olyan síkot találjunk, amely valamilyen hibafüggvény szerint minimális. Definiálhatjuk ezt a hibafüggvényt a sík pontjai és a határpontok közötti vertikális, z komponensű különbségként, de vehetjük hibának a sík és a határpontok közötti ortogonális távolságot is. Mindkét esetben egy regressziós problémához jutunk, amelyet könnyebb-nehezebb matematika apparátussal lehet megoldani. Ha az első hibafüggvényt használjuk, akkor a keresett sík a normál regressziós sík lesz, míg a második hibafüggvény minimalizálásával az ortogonális regressziós síkot kapjuk meg. Mindkét esetet a 3.1. fejezetben mutatom be részletesen, ahol a regressziós probléma definiálása és megoldása is bemutatásra kerül. A gyakorlatban inkább az ortogonális regressziós síkot szokták használni, amely esetünkben is használhatóbb síkot produkál.

[2] Szelvény alatt most határpontok összekötésével a képzett Jordan-poligont (egyszerű poligont) értem.

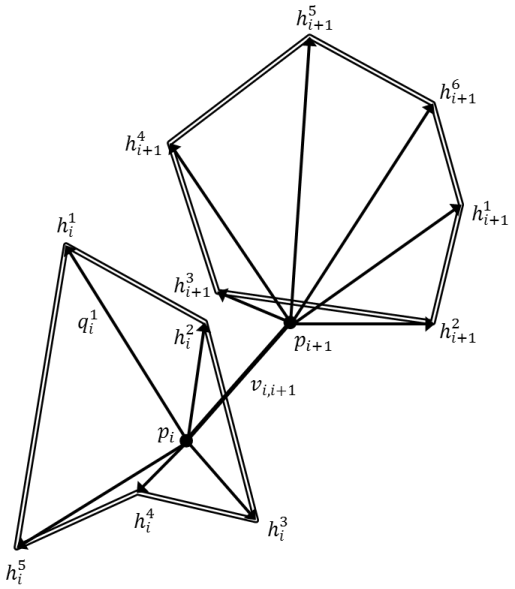
Miután meghatároztuk a regressziós R_i síkot a sorrendezésnek már csak a könnyebbik része maradt hátra. Jelölje $\tilde{h}_i^j$ az O_i szelvény h_i^j határpontjának vetületét az R_i síkra, továbbá legyen $\tilde{q}_i^j = \tilde{h}_i^j - p_i$. A bevezetett jelöléseket mutatja a 12. ábra is.



12. ábra Pontok sorrendezése a regressziós sík segítségével

A sorrendezéshez válasszunk ki egy tetszőleges $\tilde{q}_i^j$ profilvektor vetületet, majd határozzuk meg, hogy a kiválasztott vetület mekkora szöget zár be az összes többi vetülettel egy rögzített haladási irány szerint. Nagyon fontos kiemelni, hogy itt előjeles szögekre van szükség. Ha például a $\tilde{q}_i^2$ vetületet választjuk ki, akkor az óramutató járásával megegyező irányban haladva a $\tilde{q}_i^3$ 60 fokos szöget zár be, míg a $\tilde{q}_i^1$ vetület -70 fokos szöget. Az előjeles szögeket alakítsuk át pozitív szögekké (azaz a -70 fokból 290 fok lesz). A pozitív szögeket felhasználva már egyértelműen elvégezhetjük a sorrendezést, hiszen tudjuk, hogy a kiválasztott vetülete után a szelvényben az a vetület lesz a következő, amelyhez a legkisebb pozitív szög tartozik. A továbbiakban tegyük fel, hogy a határpontok és a profilvektorok az indexüknek megfelelően kerültek sorrendezésre.

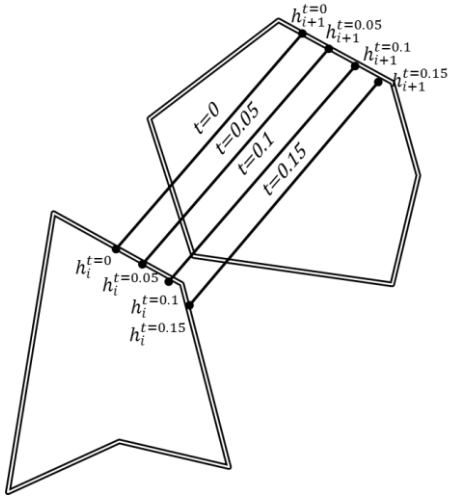
A határpontok sorrendezése után következő feladatunk az azonos poligonoldalhoz tartozó szomszédos szelvények összekötése. $4P^2$ esetén könnyű dolgunk volt, hiszen tudtuk, hogy minden esetben négy profilvektor van, amelyeknek azonos az orientációja. Ebben az esetben ennek sokkal nehezebb a meghatározása. Egyrészt a határpontok száma sem egyezik, másrészt a határpontok orientációja is tetszőleges.



13. ábra Az azonos poligonoldalhoz tartozó két szelvény

Elsőként az merült fel bennem, hogy a határpontokat felhasználva minden szelvényhez egy parametrikus görbét definiálunk, amelynek pontjait a $t \in [0,1]$ segédváltozó segítségével generálhatjuk. Ha megoldjuk azt, hogy a két szomszédos szelvény parametrikus görbéinek *t=0* pontjai a szelvények azonos pontjára essenek, akkor a szemközti szelvényeken minden esetben megfeleltethetünk egymásnak két pontot, amelyeket ugyanaz a segédváltozó érték generált. A parametrikus görbének mindenképpen olyan görbét kell választanunk, amely nem approximálja a pontokat, hiszen akkor a mérés eredménye kevésbé lenne pontos.

Ezt az ötletet szemlélteti a következő ábra is.



14. ábra Parametrikus görbe alapú háromszögesítés

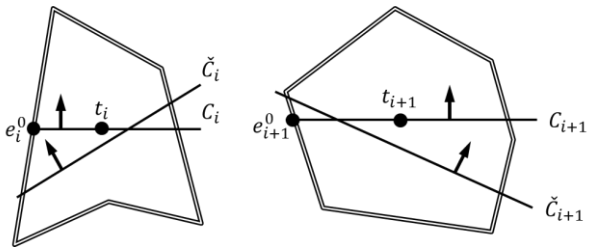
Mindezek alapján látható, hogy a háromszögesítéshez elegendő megtalálnunk a szelvények $h_i^{t=0..1}$ pontjait, amelyekből az előző fejezetben leírt triangle strip módszerrel készíthetünk háromszöghálót.

Ezt azért vetettem el, mert nagyon sűrűn kellene pontokat generálnunk a segédváltozó segítségével ahhoz, hogy pontos eredményt kapjunk. A 14. ábrán is látható, hogy a legtöbb határpont nem lesz a háromszögháló része, mert a segédváltozóba olyan értékeket helyettesítettünk, amelyek a határpont előtti és utáni pontokat generáltak.

A parametrikus görbék elvetése után az interneten kutattam a problémára már létező megoldások után. Mivel ilyet nem találtam egy saját algoritmust dolgoztam ki, amely sokkal egyszerűbb a parametrikus görbéknél. Az algoritmus bemenete a szelvények és a rendezett határpontok, kimenete pedig a szelvények párosított (e_i^k, e_{i+1}^k) pontjai, amelyekből elkészíthetjük a háromszöghálót.

Az algoritmus első lépése, hogy metszősíkok segítségével megkapjuk az első (e_i^0, e_{i+1}^0) pontpárt. Mielőtt az algoritmus következő lépésére ugranánk, vizsgáljuk meg jobban ezt a kérdést.

Mivel egy poligonpontból akár több szelvényt is fel tudunk venni illetve mivel a határpontok lehetnek egészen távol a poligonponttól érdemes azt az elvet követnünk, hogy a határpontokat használjuk a metszősíkok definiálásához és ne a poligonpontokat. Nagyon fontos, hogy mindkét szelvény esetén a metszősíkok orientációja azonos legyen. Ha nem lenne azonos, akkor a két kimetszett pont nem az első (e_i^0, e_{i+1}^0) páros lenne, hanem ezektől távolabb eső pontok.

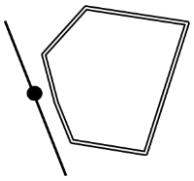


15. ábra Rosszul orientált metszősíkok

A 15. ábrára ránézve látható, hogy a rosszul orientált metszősíkok $(\check{C}_i, \check{C}_{i+1})$ nem a megfelelő pontokat metszik ki. Fontos tehát, hogy a metszősíkok normálvektorai párhuzamosak legyenek egymással. Ha ugyanazt a normálvektort használjuk minkét metszősík esetén, akkor ez biztosított.

Az előző fejezet referencia vektor generálását kis módosítással NP^2 esetén is tudjuk alkalmazni a metszősíkok normálvektorainak előállításához. Vegyünk egy

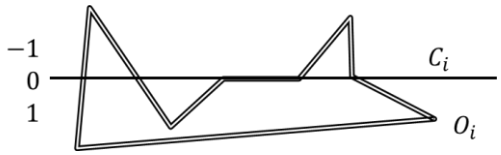
véletlen r_i vektort, valamint képezzük a t_{i+1} és t_i középpontok különbségét, amelyet jelöljön $d_{i,i+1} = t_{i+1} - t_i$. A metszősíkok normálvektora legyen az $r_i \times d_{i,i+1}$ keresztszorzat normalizált eredménye.



16. ábra "Céltalan" metszősík

Már csak az a kérdéses, hogy a metszősík mely ponton menjen át. Ha a metszősíkot a határpontok átlagával definiáljuk, akkor garantált, hogy a metszősík és a szelvény egyenesei legalább egy pontban metszik egymást és nem fordul elő az 16. ábrán látható eset. Ez pont abból adódik, hogy a határpontok átlagát használtuk.

Az előzőek értelmében biztosan lesz egy metszéspont, de koránt sem biztos, hogy maximum csak kettő lehet.



17. ábra Több mint két metszéspont

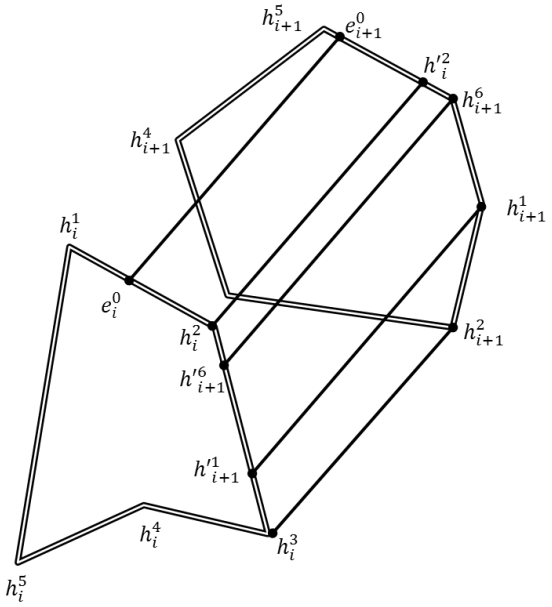
Nézzünk csak rá az 17. ábrán látható szelvényre és vágósíkra, amely kevésbé életszerű példát mutat, de orvosi lónak most tökéletesen alkalmas lesz. Most már remélem a kedves Olvasó számára is nyilvánvaló, hogy a szelvény oldalai közül kettőnél több is metszheti a C_i síkot. Az is elképzelhető, hogy a határpontok pontosan a metszősíkon fekszenek. Az NP[2] esetén egy nagyon egyszerű algoritmust alkalmaztam a metszéspontok meghatározásához, amely kicsit hasonlít Sutherland-Hodgeman [3] poligonmetsző algoritmusához.. Minden határpont esetén eldöntjük, hogy a metszősík melyik oldalán van, amelyet 0, 1 vagy -1 fog jelölni. Ha a határpont a síkon van, akkor ez az érték 0 lesz, minden esetben 1 vagy -1. Vesszük a szelvény első egyenesét. Ha az egyenes két határpontjához tartozó értékek szorzata 1, akkor tudjuk, hogy nem kell foglalkozni metszéssel, hiszen a határpontok a sík azonos oldalán vannak. Ha a szorzat nulla, akkor az egyenes egy, vagy mindkettő határpontja a síkon van. A kimenet ebben az esetben a megfelelő határpont lesz. Ha a szorzat -1, akkor tudjuk, hogy egy határpont

[3] http://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm

sincs a síkon, viszont az egyenest metszi a sík. A kimenet ekkor a metszéspont lesz. Ha ezt minden egyenes esetén elvégezzük, akkor megkapjuk az összes metszéspontot.

A metszéspontok közül már csak ki kell választanunk a megfelelőt. Itt dönthetünk, de vagy azt az a metszéspontot választjuk, amely a metszősík „bal szélén" van, vagy azt, amelyiket a jobbon. A lényeg, hogy minden szelvény esetén ugyanazt válasszuk. Szomszédos szelvények esetén a kiválasztott metszéspontok adják a (e_i^0, e_{i+1}^0) párost.

Térjünk vissza a szelvények összekötésével foglalkozó algoritmus második lépéséhez. Most már adott az első (e_i^0, e_{i+1}^0) pontpáros. A feladatunk ezen pontokból kiindulva egy háromszöghálót feszíteni az *i.* és az *i+1.* szelvényekre, amelyhez elegendő a triangle strip bemeneteként szolgáló pontokat megadnunk.



18. ábra Parametrikus görbe alapú összekötés

Az algoritmus az e_i^0 *és* e_{i+1}^0 pontokból indul. Mielőtt bármilyen kimenetet adnánk, meghatározzuk az *i.* és az *i+1.* szelvények kerületét, amelyet k_i illetve k_{i+1} jelöl. Meghatározunk továbbá egy haladási irányt, amely az 18. ábrán az óramutató járásával megegyező. Felveszünk négy speciális mutatót, amelyek mindig a szelvények valamely pontjára fognak mutatni: $act_i, act_{i+1}, next_i, next_{i+1}$. Kezdetben legyen

$$act_i = e_i^0$$

$$act_{i+1} = e_{i+1}^0$$

Vegyük a e_i^0 és e_{i+1}^0 pontok után haladási irány szerint következő két határpontot, a $next_i, next_{i+1}$ mutatók erre a két pontra mutassanak. A 18. ábra alapján ezek

$$next_i = h_i^2$$

$$next_{i+1} = h_{i+1}^6$$

Adjuk a kimenetre az $(act_i, act_{i+1}) = (e_i^0, e_{i+1}^0)$ pontpárost.

A. Határozzuk meg a következő értékeket:

$$l_i = \frac{next_i - act_i}{k_i}$$

$$l_{i+1} = \frac{next_{i+1} - act_{i+1}}{k_{i+1}}$$

l_i és l_{i+1} gyakorlatilag az aktuális és a következő pontok közötti különbséget adja meg. Azért kell még leosztani a kerülettel is, mert a szelvények kerülete nem feltétlenül egyenlő.

B. Vegyük l_i és l_{i+1} közül a kisebbet, illetve a megfelelő indexű aktuális pontot. Ha l_i kisebb, akkor ez act_i, ellenkező esetben act_{i+1}. Egyenlőség esetén külön járunk el, ugrás az F pontra.

Az ábra alapján az első menetben h_i^2 lesz a kiválasztott pont, mert a kerületek arányát figyelembe véve közelebb van kiindulási ponthoz.

C. A kiválasztott ponthoz vegyünk fel a szemközti szelvényen egy új pontot arányosítva a kerületekkel.

Ez az új pont első alkalommal h'^2_i lesz.

D. Adjuk a kimenetre a kiválasztott és az új pontot.

E. Frissítsük a mutatókat.

A kiválasztott pontnak megfelelő indexű aktuális mutatót állítsuk a *next* által mutatott pontra, a *next* mutató pedig mutasson a haladási iránynak megfelelő következő határpontra. A szemközti szelvény aktuális mutatója mutasson az újonnan felvett pontra, a *next* mutató maradjon változatlan.

Mivel a kiválasztott pont h_i^2 volt, ezért a mutatók a következők lesznek:

$$act_i = h_i^2$$

$$act_{i+1} = h'^2_i$$

$$next_i = h_i^3$$

$$next_{i+1} = h_{i+1}^6$$

F. Ha l_i és l_{i+1} egyenlő, akkor adjuk a kimenetre az aktuális pontokat, majd frissítsük a mutatókat. Mindkét aktuális mutató mutasson a *next* által mutatott pontra, a *next* mutatók pedig a haladási iránynak megfelelő következő határpontokra.

A következő lépésben ugyanúgy járunk el az *A* lépéstől kezdve. A kiválasztott pont ebben az esetben a h_{i+1}^6 lesz, így a szemközti szelvényen fel kell vennünk egy h'^6_{i+1} pontot. A kimenetre ez a két pont kerül. A mutatók pedig a következők lesznek:

$$act_i = h'^6_{i+1}$$

$$act_{i+1} = h_{i+1}^6$$

$$next_i = h_i^3$$

$$next_{i+1} = h_{i+1}^1$$

Az *A*-*F* lépéseket addig végezzük gyakorlatilag egy ciklusban, amíg a *next* mutatók a kiindulási e_i^0 és e_{i+1}^0 pontokra nem mutatnak.

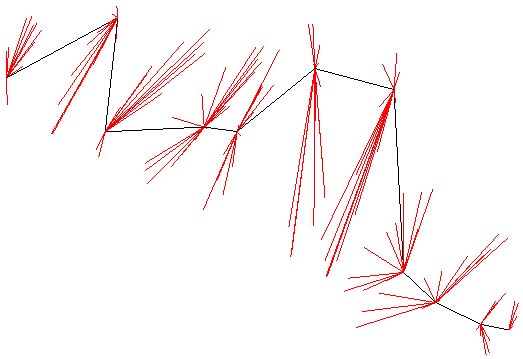
Az algoritmus tehát e_i^0 és e_{i+1}^0 pontokból indulva minden határponthoz felvesz a szemközti szelvényen egy új pontot (kivéve, ha a szelvényen ez az új pont egy már létező határpont), majd ezeket a kimenetre adja. Ezzel gyakorlatilag az NP^2 minden lépését ismerjük, következhet a módszer elemzése.

2.1.2.1 Elemzés

Az NP^2 nagy előnye, hogy egy poligonpontból tetszőlegesen sok határpontot tudunk felvenni. Ha csak egy elnagyolt, de viszonylag pontos képet szeretnénk kapni egy járatról, akkor szelvényenként felveszünk 5-10 határpontot, amelyek már használható eredményt adnak. Mindig az elérni kívánt részletességnek megfelelő számú határpontot vegyünk csak fel. Mivel a szelvény egyeneseinek nem kell egy síkba esniük, ezért a mérés könnyedén elvégezhető egy DistoX-el is.

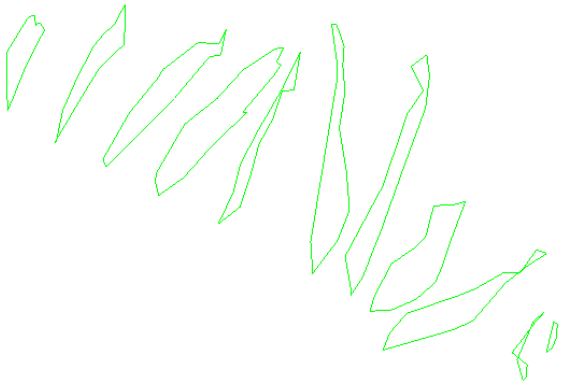
A módszert még inkább használhatóbbá teszi az a szabadság, hogy egy poligonpontból akár több szelvényt is felvehetünk. A barlangban végzett mérések során volt erre példa, amikor egy járat kanyarodását egyetlen pontból mértem fel.

A NP^2 első implementációját a Pálvölgyi-Mátyás-hegyi barlangrendszerben próbáltam ki. Az első mérést a Színház-teremben végeztem, amely egy körülbelül 20 méter hosszú, átlagosan 5-6 méter magas terem. A DistoX-el 141 pontot vettem fel, szelvényenként átlagosan 11 határponttal. Az következő ábrán fekete vonal jelöli a poligont, míg a piros vonalak a profilvektorokat.



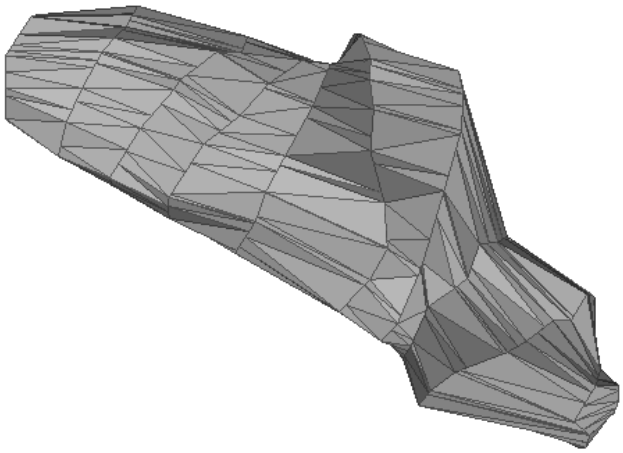
19. ábra A poligon és a profilvektorok

A következő ábrán a határpontokból készített szelvényeket láthatjuk zöld színnel.



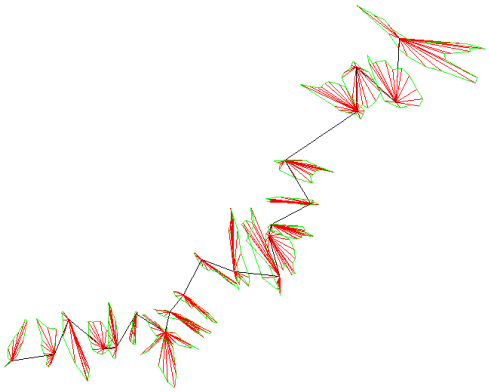
20. ábra A szelvények

Következzen a szelvényekre feszített háromszögháló.



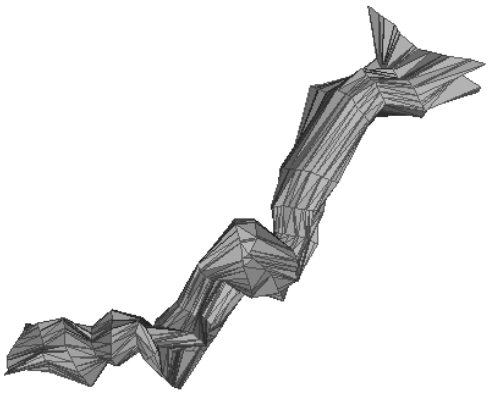
21. ábra A háromszögháló

A fenti eredmény már pontos és élethű, amelyet körülbelül egy óra munka árán kaptam meg úgy, hogy az NP^2 módszert először használtam a gyakorlatban. Az 21. ábrára tekintve remélem a kedves Olvasót is sikerült meggyőznöm arról, hogy az NP^2 alkalmas arra, hogy barlangban használjuk. Az első mérés után már tudtam, hogy termek felméréséhez használható, de mindenképpen szerettem volna egy meanderező[4] járatot is felmérni. A második alkalommal a Pálvölgyi-Mátyás-hegyi barlangrendszerben végeztem lézeres mérést a Vadvizek-útján, amely körülbelül egy 20 méter hosszú kacskaringós folyosó.



22. ábra Vadvizek útja

A 22. ábrán most egyben látható a poligon, a profilvektorok és a szelvények. A fenti szelvényekre háromszöghálót feszítve a következő ábrán látható eredményt kaptam.



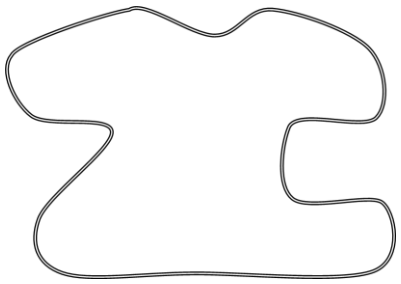
23. ábra Vadvizek útja - háromszögháló

[4] Kanyargós

Az 24. ábra barlang 1980-as években készített térképének azt a részletét mutatja, amelyen a Vadvizek-útja látható. A háromszögháló és a kétdimenziós térkép közötti eltérés szemmel látható.

24. ábra Vadvizek-útja 30 évvel ezelőtt

A második mérés alátámasztotta, hogy az NP^2 „egyszerű” járatok felmérésére is alkalmas, de nem alkalmazható minden helyzetben. Egyszerű járat alatt egy olyan járatot értek, amelynek pontjai elérhetőek egy poligonpontból. Hazánkban is több olyan barlang van, amelynek járatai több szintesek. Ezen járatok általában úgy alakultak ki, hogy az üregeket formáló víz mennyisége folyamatosan változott. Például egy szárazabb időszakban a víz keskenyebb járatot eredményezett, míg egy olvadásos korszakban a rengeteg víz szélesebb utat tört magának. Többszintes járatok kialakulhatnak eltérő kőzetekben is, hiszen egy keményebb kőzet jobban ellenáll a víz erodáló hatásának. Ilyen esetekben az 25. ábrán is látható színlők[5] alakulnak ki, amelyek a járatokat több szintre osztják.



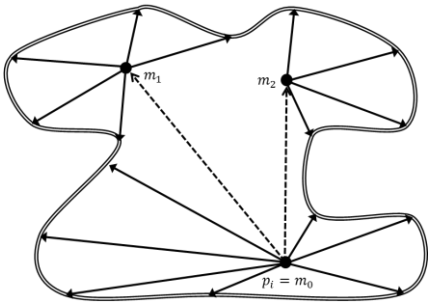
25. ábra Többszintes járat

Többszintes járatoknál nem találunk olyan poligonpontot, amellyel a fent látható szelvény minden pontját fel tudnánk mérni, ezért az NP^2 továbbfejlesztésére van szükség.

[5] párkány

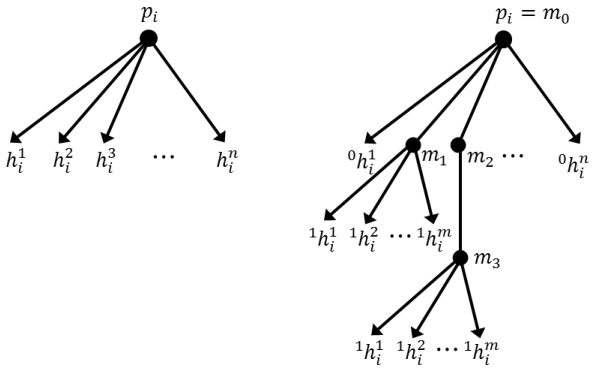
2.1.3 Kiterjesztett NP^2

Az előző pontban kifejtettek alapján látható, hogy az NP^2 nem alkalmas többszintű járatok felmérésére. A korlátozás abból fakad, hogy a határpontokat mindig csak egy poligonpontból vesszük fel, amely egy kétszintes fát eredményez. Amennyiben megengedjük olyan köztes pontok felvételét, amelyekből szintén vehetünk fel határpontokat, akkor tetszőleges profilú szelvényt fel tudunk mérni.



26. ábra Kiterjesztett NP2

A 26. ábra többszintes járatát például elkezdhetjük felmérni a poligonpontból, azokat a részeket pedig, amelyeket nem érünk el a poligonpontból két köztes pont segítségével (m_1, m_2) tudjuk felmérni. Elképzelhető, hogy a szelvény bizonyos részét csak egy olyan köztes pontból tudnánk felmérni, amely a poligonpontból nem érhető el a lézersugárral. Engedjük hát meg azt is, hogy a köztes pontokból további köztes pontokat tudjunk felvenni. Ezzel gyakorlatilag a poligonpontok és a köztes pontok között semmilyen különbséget nem teszünk, tekintsük hát a poligonpontot a 0. indexű köztes pontnak.



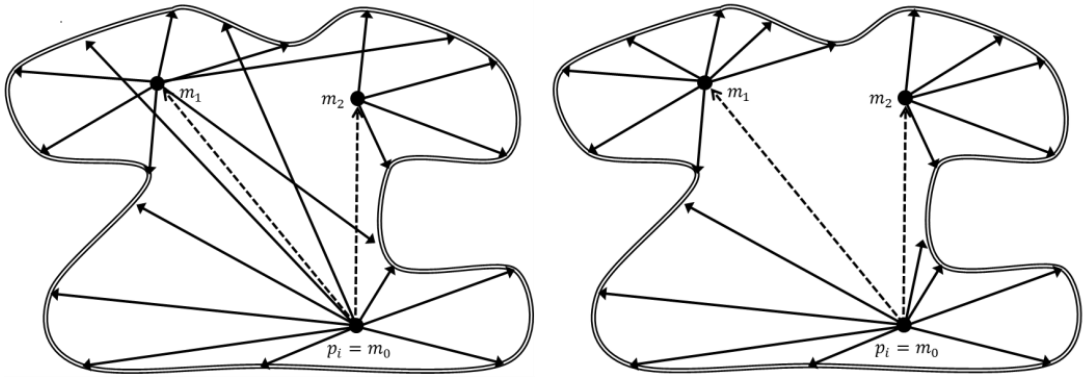
27. ábra Többszintes fa

A fenti kiegészítések egy többszintes fát eredményeznek, amelynek a határpontok lesznek a levelei, a fa gyökere és belső csúcsai pedig a köztes pontok. A határpontokat ki kell egészítenünk annak a köztes pontnak az indexével, amelyből a

határpontot felmérték. Jelölje tehát az *i.* szelvény *j.* köztes pontjából felmért *k.* határpontot ${}^{j}h_i^k$.

Az egyetlen feladat, amit meg kell oldanunk a határpontok globális sorrendezése. A sorrendezés után már csak át kell adnunk a határpontokat az előző fejezetben leírt algoritmusnak, amely a szelvényekre feszít egy háromszöghálót. NP[2] esetén könnyű dolgunk volt, hiszen ott minden határpont ugyanabból a poligonpontból lett felmérve. Esetünkben nem használhatjuk a poligonpontot, vagy a határpontok átlagát a sorrendezéshez, így egy új algoritmus kidolgozása szükséges.

Az első lépésben vegyük az összes határpontnak megfelelő legjobb R_i ortogonális regressziós síkot és vegyük a profilvektorok, határpontok és köztes pontok vetületét erre a síkra. A pontokat az R_i sík alapján fogjuk rendezni, de előtte még szüntessük meg a profilvektorok vetületeinek keresztezését.

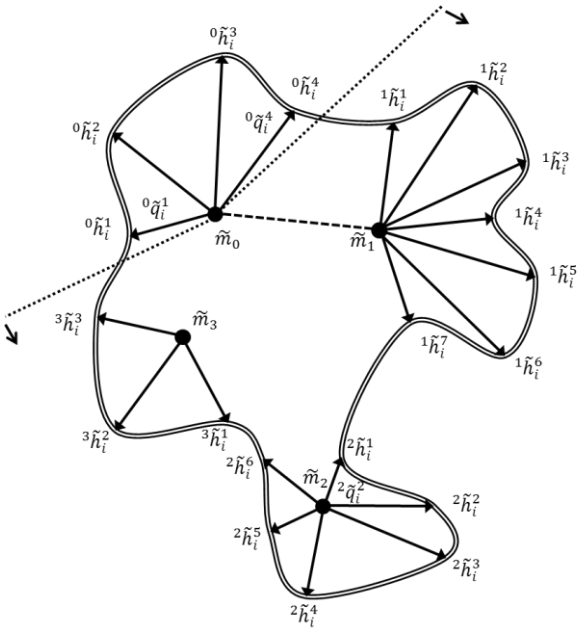


28. ábra Keresztezések megszüntetése

A 28. ábrán látható keresztezéseket a határpontok átrendezésével tudjuk megszüntetni, miután előáll a jobb oldali eredmény. Vegyük a határpontok vetületeit egyenként és kössük ahhoz a köztes ponthoz, amelyhez a legközelebb van. A legtöbb esetben ez az eredeti köztes pont lesz, de elképzelhető, hogy egy új köztes ponthoz kerül. A fenti ábrán az m_1 köztes pont például két határpontot veszít el, de m_0 köztes ponttól kettő köztes pontot közben örököl.

A keresztezések megszüntetése után kezdődhet a sorrendezés, amelyhez a vetített ${}^{j}\tilde{h}_i^k$ határpontokat és a szintén a regressziós síkon lévő ${}^{j}\tilde{q}_i^k$ profilvektorokat és $\tilde{m}_i$ köztes pontokat fogjuk használni. Az első lépés a határpontok lokális sorrendezése. Minden köztes pont esetén rendezzük a határpontokat az előző fejezetben leírt módszer szerint úgy, mintha a köztes pont lenne a poligonpont. Az egyetlen különbség, hogy ebben az esetben nem kell egy új regressziós síkot keresnünk a köztes pont

határpontjainak, hanem használhatjuk a már meglévő síkot is. Fontos, hogy minden lokális rendezést az óramutató járásával egyező vagy ellentétes irányba végezzünk.



29. ábra Globális sorrendezés

A lokális rendezések után az 29. ábrához hasonló eredmény kapunk, amelyen határpontok és köztes pontok különálló gócait látjuk. A globális sorrendezés utolsó lépése a gócok összekötése. Ehhez vizsgáljuk meg minden köztes pont esetén, hogy melyek azok a tartományok, amelybe további idegen határpontok esnek. Két határpontot nevezzünk idegennek, ha azok nem ugyanahhoz a köztes ponthoz tartoznak. Az 29. ábrán például az $\tilde{m}_0$ köztes pont $^0\tilde{h}_i^4$ és $^0\tilde{h}_i^1$ határpontjai közötti tartományban vannak idegen határpontok, míg ez a tartomány az $\tilde{m}_2$ köztes pont esetén $^2\tilde{h}_i^5$ és $^2\tilde{h}_i^2$ által meghatározott. Induljunk el egy tetszőleges gócból, mondjuk $\tilde{m}_0$köztes ponthoz tartozó gócból, és képzeletben hosszabbítsuk meg azokat a profilvektorokat, amelyek az idegen tartomány határain vannak. Ez a két profilvektor $^0\tilde{q}_i^4$ és $^0\tilde{q}_i^1$ lesznek. Vegyük az első ilyen profilvektort és közelítsük a másik profilvektor felé, de közben figyeljünk arra, hogy metszeni fogja-e valamelyik másik góc határpontját. Abban biztosak lehetünk, hogy ilyen idegen határpont biztosan lesz, az idegen tartomány definíciója miatt. Vegyük annak a gócnak a köztes pontját, amellyel ez a képzeletbeli vonal találkozott és kössük össze az induló gócpont köztes pontjával. Az ábrán ezt egy szaggatott vonal $\tilde{m}_0$ és $\tilde{m}_1$ köztes pontok között. Kössük össze azt a két határpontot, amelyek a legkisebb szöget zárják be ezzel az egyenessel. Ezzel gyakorlatilag a két gócpontot összekötöttük. Az előző lépéseket végezzük addig, amíg különálló gócpont létezik.

A gócpontok összekötése után előáll határpontok globális sorrendezése, amely a szelvényeket beburkoló algoritmus bemenete. A kiterjesztett NP^2 már tetszőleges formájú szelvények felmérését lehetővé teszi, ami miatt a három eddig bemutatott módszer a közül a kiterjesztett NP^2 a leginkább alkalmazható.

Sajnos a kiterjesztett NP^2 –nek is vannak hátrányai, amelyek abból adódnak, hogy az algoritmus szelvényekből építkezik. Nem minden barlang mérhető fel könnyedén a szelvényes módszerrel, bizony vannak olyan esetek, amikor a szelvényes technika inkább megszorításnak tűnik. Képzeljen el a kedves olvasó például egy olyan járatot, amelyben egy éles kanyar van. Ha élethű modellt szeretnénk kapni, akkor legalább 3-4 szelvényt fel kell vennünk a kanyarhoz, ami nagyon sokáig tarthat. Ügyelnünk kell arra is, hogy a szelvények ne keresztezzék egymást, különben egy furcsa háromszöghálót kapunk. A barlangban végzett méréseket lassabban tudjuk csak végezni, ha folyamatosan fejben kell tartanunk, hogy egy terem melyik része van már felmérve és melyik nem.

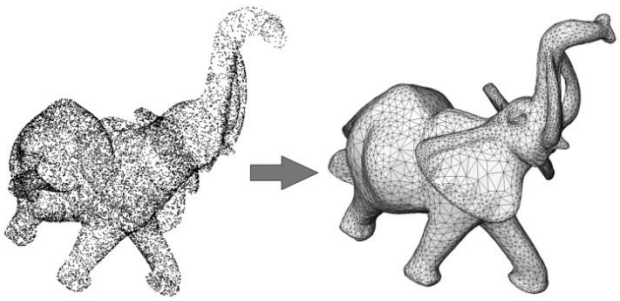
A fentiek ellenére azt gondolom, hogy egy olyan gyors mérési módszert sikerült kidolgoznom, amely az esetek többségében használható és eredményül egy élethű modellt szolgáltat. Miután újra jelentkeztem az MKBT-nél és jeleztem nekik az újabb implementációt felajánlották, hogy a Pálvölgyi kőfejtőben megrendezett több mint 100 fős barlangász kutatótáborban rövid előadást tarthatok az eddigi eredményeimről. Az előadás megtartása után lehetőséget kaptam, hogy felmérjem a világ legnagyobb hévizes termét az ország egyik legszebb és jobbal féltve őrzött barlangjában, a József-hegyi barlangban.

30. ábra Kinizsi-pályaudvar

A József-hegyi barlang a Rózsadomb fokozottan védett barlangja, amelynek legnagyobb terme a Kinizsi-pályaudvar. A felajánlás után eldöntöttem, hogy megpróbálok egy nagyon pontos háromszöghálót előállítani, amelyen aztán utómunkálatokat végzek más grafikai programokkal. Az 30. ábrára ránézve látszik, hogy ehhez nagyon sűrűn kellene felvenni a szelvényeket, ezért a szelvényes technikát elvetettem. Következhetett egy új mérési módszer, amely mindenféle struktúra nélkül a lehető legkevesebb pontból, gyakorlatilag egy ritka pontfelhőből próbálja rekonstruálni a felületet. Az előadás megtartása után több barlangásszal is beszélgettem, akik szintén ezt a továbbfejlesztést javasolták.

2.1.4 SPCR

Az előző fejezetekben olyan mérési módszerekkel foglalkoztunk, amelyek szelvények felvételével és azok összekötésével foglalkoztak. Ebben a fejezetben elrugaszkodunk a $4P^2$ módszernél bevezetett giliszta hasonlattól, és a mérési pontokból egy egészen mást technika, a felület rekonstrukció segítségével állítjuk elő a modellt. A felület rekonstrukció pontok halmazából állít elő egy felületi modellt, amely vagy egy approximációs vagy egy interpolációs eljárás eredményeként születik meg. Mivel a bemenet pontok halmaza, így a felmért tárgy, alakzat, vagy forma a struktúrájáról semmilyen információnk nincs.



31. ábra A felületrekonstrukció lényege: pontfelhőből 3D modell

Felület rekonstrukciós algoritmusok már viszonylag régóta léteznek, de sokáig elkerülték a kutatással foglakozó szakemberek figyelmét és érdeklődését. Az áttörést a lézerszkennerek megjelenése hozta meg, hiszen szükség volt olyan eljárásokra, amelyekkel a szkennerek adatait fel lehetett dolgozni. Mára már a rekonstrukciós algoritmusok ipari és tudományos alkalmazása is jelentős.

A lézerszkenner egy létező fizikai objektumot pásztáz végig lézersugár segítségével, melynek eredménye egy nagyméretű pontfelhő lesz. Manapság már olyan

hihetetlenül gyors szkennerek léteznek, amelyek másodpercenként több tízezer pontba végeznek mérést. A lézerszkennereket leginkább az iparban alkalmazzák, de használták már barlangok felmérésére is. Magyarországon körülbelül 5 éve végezték az első mérést a Baradla-barlang egy rövid szakaszán, melynek eredményét volt szerencsém megtekinteni a Barlangtani Intézetben még évekkel ezelőtt. A mérés eredménye egy több millió pontból álló nagyon pontos és sűrű pontfelhő volt, amelyet egy külön megvásárolt programmal lehetett kezelni. Már akkor biztos voltam abban, hogy lézerszkennerek barlangi használata kivitelezhetetlen, mert nagyon drágák, nem teszik lehetővé a gyors mérést, valamint szűk, vizes illetve kötéltechnikás barlangokban rendkívül komplikált és kockázatos a szállítása. Helyette használhatjuk a DistoX-et, ami bár nem olyan költséges, de hátránya, hogy nagyságrendekkel kevesebb pontot tudunk vele felmérni, mint egy modern lézerszkennerrel. A Pálvölgyi-Mátyás-hegyi barlangrendszerben végzett mérés során kevesebb, mint 400 pontot vettem fel egy 20 méter hosszú járatról. A lézerszkenner a szakasz felénél elérte volna a milliós nagyságrendet. A probléma tehát, amellyel ebben a fejezetben foglalkozunk: Hogyan tudunk egy ritka pontfelhőből felület rekonstrukciós algoritmusokat felhasználva előállítani egy helyes felületi modellt. A problémára egy olyan módszer nyújtja a megoldást, amely az SPCR (Spare Point Cloud Reconstruction) nevet kapta.

A probléma vizsgálatakor több alkalmazott algoritmust vizsgáltam meg, amelyet a következő bekezdésekben szeretnék bemutatni a kedves Olvasónak.

Elsőként a Poisson felület rekonstrukciós algoritmust vizsgáltam meg, amely az egyik legelterjedtebb módszer[6]. A részletes matematikai háttér, illetve az algoritmus elemzése a 3.2.1 fejezetben található. Nagyon röviden a Poisson algoritmus egy Poisson egyenlet megoldásával állítja elő a bemeneti $\vec{V}$ vektortérből (orientált mérési pontok halmaza) az approximált felületi modellt. A további elemzések megértéséhez mindenképpen javaslom a 3.2.1 fejezet végigolvasását.

A vizsgálatok után arra az eredményre jutottam, hogy a Poisson rekonstrukciós algoritmus jó választás lehet egy lézerszkenner pontfelhőjének felület rekonstrukciójához, de sajnos barlangban készített ritka pontfelhőknél sokszor használhatatlan eredményt ad. A pontok sűrűségére nem tudunk egy egzakt, globális minimum korlátot adni, amely biztosítaná a megfelelő minőségű felületi modellt. Sajnos ez a szám legtöbbször függ az adott modelltől, és így leginkább csak empirikus úton

[6] A CGAL és VCG library, illetve a MeshLab program is ezt használja

mondhatunk egy minimum értéket. Sajnos az algoritmus az esetek többségében nem birkózik meg a ritka vagy változó sűrűségű pontfelhőkkel.

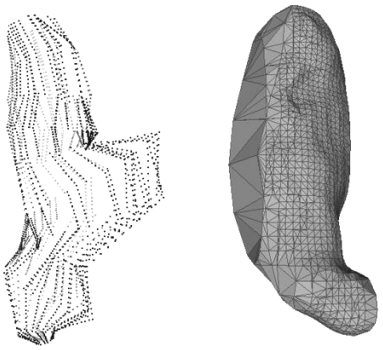
32. ábra Neptun szigonya

A 32.ábra Neptun szigonyán mintavételezett 1000 pontot mutatja. Láthatjuk, hogy az algoritmus azokon a helyeken ad szép rekonstrukciót, ahol a pontok viszonylag sűrűk, míg a ritka sűrűségű pontok közelében már nem generál felületelemeket.

A másik probléma, hogy az algoritmus nem követeli meg, hogy a mért pontok a rekonstruált $\partial\tilde{M}$ felületen vagy annak közelében legyenek. Kényszereket sem adhatunk, amelyek biztosítanák, hogy a rekonstruált $\partial\tilde{M}$ felület és a mért pontok közötti távolság négyzetes hibája egy adott határon belül maradjon. Egy olyan approximált $\tilde{\chi}$ indikátor függvényt használunk, amely minimális a következő értelemben:

$$\min\left\|\nabla\tilde{\chi} - \vec{V}\right\|$$

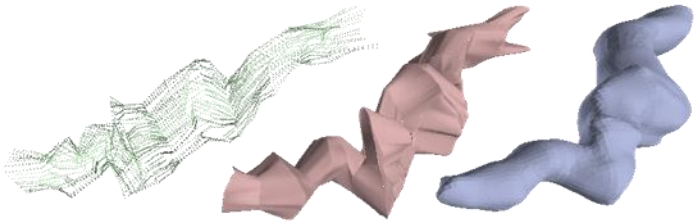
A minimum értékek esetében nem várhatjuk el a 0 értéket, hiszen a mérési adatokból egy közelítő modellt próbálunk előállítani. Sajnos vannak olyan esetek, amikor a minimum érték nagyon nagyra nő, így az indikátor függvény gradiense által képzett vektortér nagyon eltér a mérési eredményeket tartalmazó vektortértől, amely aztán felismerhetetlen felületi modellt eredményez.



33. ábra Az eredeti ponthalmaz és a rekonstruált modell

Az 33.ábra egy 3168 méretű ponthalmazt és annak rekonstruált változatát láthatjuk. A jobb oldali modell nem közelíti meg a pontfelhőhöz tartozó ideális modell felületét.

A harmadik probléma, hogy a Poisson algoritmus érzékeny az éles ugrásokra és az olyan pontfelhőre, melyben a háromszögek által meghatározott síkok közötti szög kicsi. Egy barlang természeténél fogva tartalmaz kiugró cseppköveket, képződményeket, éles sarkokat, amelyeket a Poisson rekonstrukció meg sem közelít. Ez annak köszönhető, hogy a Poisson algoritmus egy simított felületet generál, nem pedig egy durva háromszöghálót.

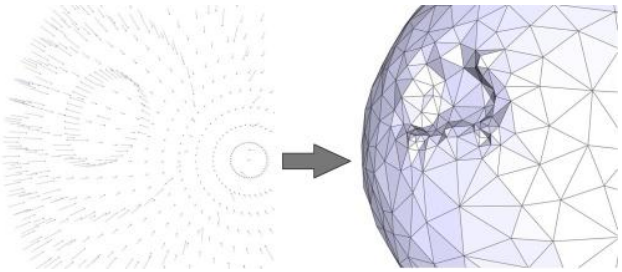


34. ábra Az eredeti pontfelhő, a valós "éles" modell, és a rekonstrukció eredménye

A 34. ábra bal oldalán látható a mintavételezett ponthalmaz, középen a barlang realisztikus modellje, jobb oldalon pedig a Poisson algoritmus eredménye. A rekonstruált felület valóban egy sima felület, mely a durva kiugrásokat nem követi.

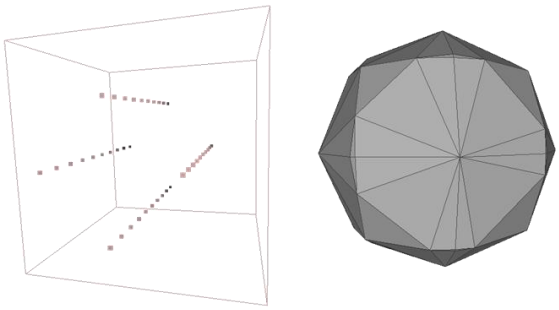
A negyedik probléma, hogy a Poisson eredménye egy zárt felület lesz, abban az esetben is, ha az eredeti mérés mondjuk egy minkét végén nyitott folyosóról készült. Nem áll rendelkezésre az eredeti pontok és a rekonstruált felület pontjai között egy megfeleltetés, így nehezen tudjuk azt is meghatározni, hogy a zárt felületet hol és milyen profillal vágjuk el.

Az ötödik probléma, hogy a Poisson algoritmus érzékeny a rosszul orientált normálvektorok csoportjára. A 3.2.1 fejezetben foglalkoztunk azzal, hogy a Poisson jól kezeli a zajos adatokat, a rosszul megadott normál vektorokat, de ha ezekből képzett csoportok is vannak, akkor a rekonstruált $\partial\widetilde{M}$ felület nem lesz tökéletes. A következő ábrán látható, hogy egy rosszul orientált normálvektor csoport az ideális gömbfelülethez képest egy bemélyedést tartalmaz.



35. ábra Rosszul orientált normálvektorok, és az eredmény

A hatodik probléma, hogy a Poisson algoritmus nagyon érzékeny a megfelelő paraméterezésre. Nem megfelelő paraméterezés esetén rendkívül pontatlan eredményeket kaphatunk.



36. ábra Rosszul paraméterezett futás eredménye

Az ábra bal oldalán egy téglatest mintavételezett pontjait látjuk, amelyből a Poisson algoritmus hibásan egy gömbfelületet rekonstruált. Rossz paraméterezés révén a Poisson algoritmus egy 1000 pontos minta rekonstrukciójához több mint 6GB memóriát használt fel.

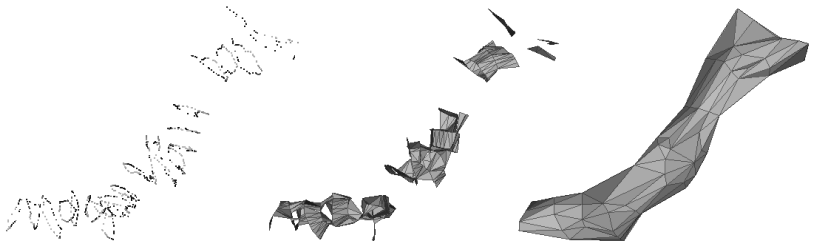
Mindezt figyelembe véve, úgy vélem, a Poisson algoritmus nem használható barlangban végzett mérések rekonstrukciójára.

A Poisson módszernél azt is láthattuk, hogy egy approximációs eljárás mennyire pontatlan eredményeket adhat, éppen ezért következőként egy interpolációs felület rekonstrukciós algoritmust kerestem. Az első ilyen algoritmus az IBM által propagált ball-pivoting algorithm, vagy röviden BPA volt.

A ball-pivoting algorithm (BPA) egy olyan interpolációs felület rekonstrukciós algoritmus, melynek az elve nagyon egyszerű: Egy δ sugarú labdát görgetünk végig a pontfelhőn, egészen addig, amíg nem találunk olyan pontot, amelyet nem érintett még a labda. A BPA részletes leírását a 3.2.2 fejezetben találja a kedves Olvasó. A BPA nagy előnye a Poissonnal szemben az egyszerűség, ami az algoritmus futási idejében is megmutatkozik. A Poisson algoritmussal ellentétben a generált felületi modell az

eredeti pontfelhő pontjaiból áll, így nem eredményez pontatlan modelleket. Az algoritmusnak két olyan jellemzője van, amely alkalmatlanná teszi barlangokban felvett ritka pontfelhők rekonstrukciójához.

Az egyik probléma, hogy nem tudunk minden esetben megfelelő δ értéket adni. A következő ábrán ugyanannak a pontfelhőnek a rekonstrukcióját láthatjuk különböző δ értékekkel. Ha a δ túl kicsi, akkor a felületi modell több komponensre esik szét, ha pedig δ értékét túl nagynak választjuk, akkor a felületi modell egyre inkább a konvex burokhoz közelít. A fenti problémára egy olyan adaptív kiegészítés lenne a megoldás, amely lokális δ értéket használna a globális helyett. Egy olyan környezetben nagyobb δ értéket használna, ahol a pontok viszonylag ritkák, míg egy sűrű ponthalmaz esetén egy kisebb δ értéket alkalmazna.



37. ábra Az eredeti pontfelhő illetve felületrekonstrukció δ=6 és δ=13 értékekkel

A másik probléma, hogy az algoritmus első lépése, a forrás háromszög kiválasztása és a *ball_pivot* lépésben az $e_{(i,j)}$ él kiválasztása a véletlen alapján történik. Ez azt eredményezi, hogy ugyanannak a pontfelhőnek rögzített δ érték mellett több felületi modellje van, amelyek teljesen eltérhetnek egymástól. Képzelje csak el a kedves Olvasó, hogy mennyire megdöbbenne, ha a korábban látott kedvenc barlang részlete a következő alkalommal teljesen máshogy nézne ki.

Mindent egybevetve azt találtam, hogy a BPA nem alkalmazható barlangban készült mérések rekonstruálásához.

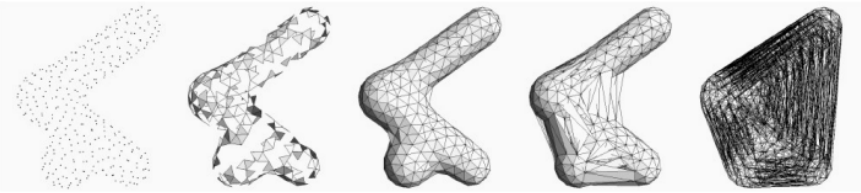
Az utolsó megvizsgált rekonstrukciós algoritmus az α-shape, amelynek minden fontos részletét a 3.2.3 fejezetben találjuk meg. Mivel az SPCR az α-shape algoritmusra épül, ezért nagyon fontos, hogy tisztában legyünk az ehhez kapcsolódó fogalmakkal.

Az α-shape algoritmus pontok halmazához rendel egy formát, amelynek a részletezettségét az α paraméter szabályozza. A következő értelmezés egy informális definíciót ad az α-shapre. Képzeljük el a pontok $\mathbb{R}^3$ terét, mint egy hatalmas

habszivacsot, melyben a pontok kis, szilárd kövek formájában vannak beékelődve különböző helyeken. Képzeljünk el egy gömb formájú α sugarú radírt. A radírral a habszivacs minden olyan részét kivágjuk, melyek elérhetőek anélkül, hogy szilárd pontokba ütköznénk. A habszivacs megmaradt része alkotja a ponthalmazhoz tartozó α-shapet.

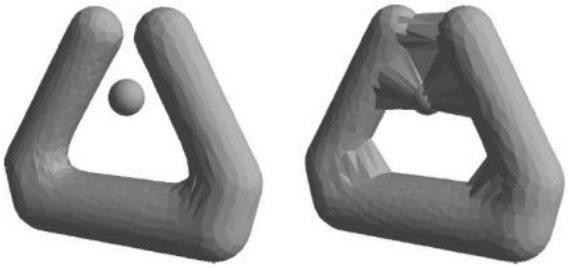
Az algoritmus elemzésénél kiderül, hogy az α-shape $O(n^2)$ lépésszámot garantál, amely 10000 pont esetén már lassú futást eredményez. Szerencsére a barlangokban végzett mérések során ennél jóval kevesebb pontot veszünk fel, így az algoritmus garantáltan lefut néhány másodperc alatt, és nem érezteti a négyzetes lépésszámot.

Az 3.2.3 fejezet végigolvasása után biztosan az Olvasóban is felmerül a kérdés, hogy miért egy olyan algoritmus következett a BPA után, amely szintén egy fix paramétert felhasználva készíti el a felületi modellt. A BPA rekonstrukciót azért vetettük el, mert nem tudtunk egy olyan δ értéket adni, ami nem azonos sűrűségű ponthalmaz esetén is megfelelő eredményt adott volna. Az algoritmus futtatása után olyan háromszöghálót kaptunk, amely több komponensből állt, sőt a komponensekben még lyukak is voltak. Joggal merül fel a kérdés, hogy az α-shape esetén nem ugyanez a probléma áll-e fent?



38. ábra Honnan tudjuk a megfelelő α értéket?

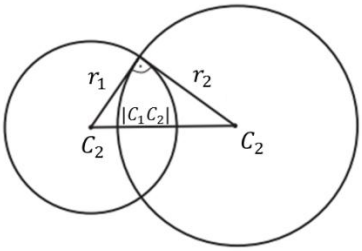
Az előző ábrán ugyanazzal a problémával találkozunk, mint BPA esetén. Az optimálisnál kisebb α egy hiányos felületi modellt generál, míg a túl nagy α pedig a konvex burokhoz hasonló háromszöghálót eredményez. A fenti ábrán látható pontfelhő különleges abban az értelemben, hogy létezik hozzá egy optimális α érték, amely a kívánt felületi modellt eredményezi. Vannak azonban olyan ponthalmazok, amelyekhez nem létezik optimális α érték. Az 39. ábrán egy ilyen ponthalmazra látunk példát.



39. ábra Nem minden esetben létezik optimális α érték

A fenti két problémára az α-shape algoritmus két kiegészítése, a súlyozás bevezetése, illetve a lokális α használata nyújt megoldást. Az így módosított algoritmust nevezzük kiterjesztett α-shape algoritmusnak.

A súlyozott α-shape bemenete a pontok halmaza és a pontokhoz rendelt pozitív súlyok. Intuitívan a nagy súly előnyben részesíti, a kis súly pedig próbálja elkerülni a kapcsolatot a szomszédos pontokkal. Ha minden ponthoz 0 súlyt rendelünk, akkor a hagyományos algoritmust kapjuk. A hagyományos α-shapehez képest súlyozott esetben reguláris háromszögesítést kell használnunk. Két gömb vagy két súlyozott pont C_1, C_2 középpontokkal és r_1, r_2 sugarakkal (súlyokkal) ortogonális, ha $|C_1C_2|^2 = {r_1}^2 + {r_2}^2$ és szubortogonális, ha $|C_1C_2|^2 < {r_1}^2 + {r_2}^2$.



40. ábra Ortogonális pontok

Adott α érték esetén a súlyozott α-komplex egy reguláris triangulációs eljárás szimplexeiből áll. A kétdimenziós reguláris trianguláció olyan háromszögeket eredményez, melyek köré írható kör reguláris a háromszöget alkotó súlyozott pontokkal és minden más ponttal szubortogonális. Háromdimenziós esetben a tetraéder pontjait határoló gömb ortogonális a tetraéder pontjaival és szubortogonális az összes többi ponttal. Az α-shapet ugyanúgy a súlyozott α-komplex szimplexei által meghatározott tartományként kapjuk meg.

A második kiegészítés lehetővé teszi lokális $\tilde{\alpha}$ értékek használatát a globális α helyett. A módosított algoritmus a konform α-shape nevet kapta.

41. ábra Eredeti ponthalmaz, hagyományos α-shape és a konform α-shape

A fenti két kiegészítés adaptív felület rekonstrukciót tesz lehetővé, amely az egyetlen alkalmazható módszer nem egyenletes ponthalmazok esetén. Mivel a barlangban végzett mérések során egy ritka és nem egyenletes ponthalmazt kapunk, ezért a kiterjesztett α-shape az egyetlen megvizsgált algoritmus, amely alkalmas barlangok mérési eredményeit megfelelően feldolgozni. A kiterjesztett α-shape algoritmus az α-shape szimplexeit négy osztályba sorolaja:

- reguláris
- szinguláris
- belső
- határoló

Egy k-szimplex akkor reguláris, ha az α-komplex határoló szimplexei közé tartozik, és az α-komplex egy másik k+1-szimplexének a lapja. Ellenkező esetben a szimplex szinguláris. Gondoljunk úgy a szinguláris szimplexekre, mint különálló szimplexekre, a reguláris szimplexre pedig mint egy összetartozó csoportban lévő szimplexre. A k-szimplex határoló, ha az α-komplex határoló szimplexei közé tartozik, ellenkező esetben belső. Az α-shape azon változatát, amely csak reguláris szimplexeket tartalmaz reguláris α-shapenek nevezzük.

Vannak a kiterjesztett α-shape algoritmusnak olyan változata, amely lehetővé teszi, hogy egy olyan α-shapet generáljunk, amely kielégíti a következő feltételeket:

- minden pont egy reguláris α-shape belsejében, vagy annak felületén van
- a komponensek száma egyenlő, vagy kisebb egy meghatározott éréknél

A továbbiakban kiterjesztett α-shape algoritmus alatt ezt a változatot értjük. Ha a komponensek felső korlátját 1-nek választjuk, akkor a felületi modell nem szakad több részre, és nem fog lyukakat tartalmazni. Nem meglepő ezek után, hogy az SPCR a kiterjesztett α-shapere épül. Azonban a kiterjesztett α-shape algoritmust ki kell egészíteni, hogy minden szimplex határoló szimplex legyen. A barlangban végzett mérések során csak olyan pontokat veszünk fel, amely egy járat, vagy terem felületének mintapontjai. A ponthalmaz így csak felületi pontokat tartalmaz, belső pontokat nem.

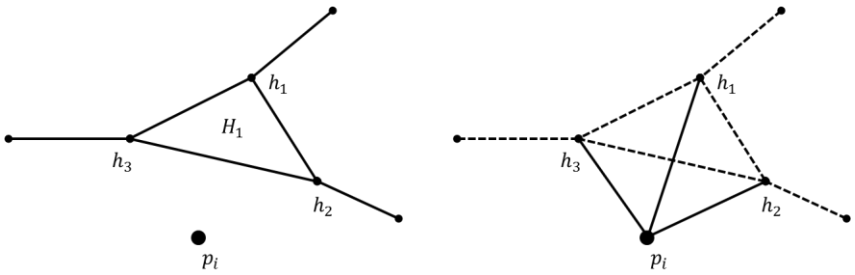
Azt várjuk el, hogy a rekonstrukció háromszöghálója a ponthalmaz minden elemét tartalmazza, ellenkező esetben maradnának olyan pontok, amelyek belső pontnak számítanak a rekonstrukciós algoritmus szerint.

Az így módosított algoritmust nevezzük SPCR algoritmusnak, mely a következő lépésekből áll:

1. Futtassuk a kiterjesztett α-shape algoritmust, amely egy olyan α-shapet generál, amely:
    - reguláris
    - konform
    - minden pont az α-shape belsejében vagy annak felületén van
    - pontosan egy komponensből áll
2. Vegyük az összes belső pontot, és kapcsoljuk őket a rekonstrukció háromszöghálójához. Ennek részletei a következők:

Ha nem találunk belső pontot, akkor a kiterjesztett α-shape felületi modelljét fogadjuk el. Ellenkező esetben építsünk a belső pontokból egy oktális fát, amely a háromdimenziós teret felosztja. A fát felhasználva vizsgáljuk meg, hogy egy belső pont környezetében vannak–e még más belső pontok is. Ha találunk ilyen pontokat, akkor képezzük ezek csoportját. A csoportokat rendre jelölje $\mathcal{G}_i, 0 \leq i \leq n$. Ha nem találunk csoportokat, akkor az azt jelenti, hogy minden belső pont különálló pont.

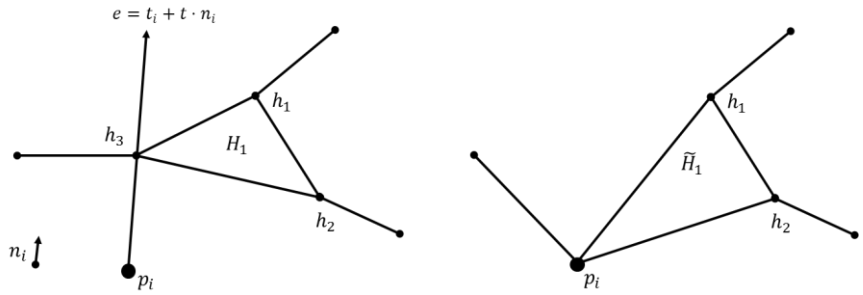
Különálló pontok esetén keressük meg a ponthoz legközelebbi háromszöget. A különálló p_i pont legközelebbi háromszögét jelölje H_1. A legközelebbi háromszöget távolítsuk el a háromszöghálóból és a háromszög pontjait, illetve a különálló pontot felhasználva adjuk a hálóhoz a következő háromszögeket: . $(h_1, h_2, p_i), (h_2, h_3, p_i), (h_3, h_1, p_i)$.



42. ábra Különálló pont hálóhoz csatolása

Az utolsó feladatunk a $\mathcal{G}_i$ csoport pontjainak a hálóhoz kötése. Vegyük ehhez a csoport pontjainak megfelelő legjobb R_i ortogonális regressziós síkot és ennek n_i

normál vektorát. Vegyük az $e = p_i + t \cdot n_i$, $-\infty \leq t \leq \infty$ egyenest, és keressük meg azt a két háromszöget, amelyeket metsz az egyenes. Ilyen háromszögek biztosan lesznek, mert a csoport pontjai belső pontok. Vegyük azt a háromszöget, amely a p_i ponthoz közelebb van, és az előző bekezdésben bemutatott művelettel iteratívan kapcsoljuk a hálóhoz az összes belső pontot. Abban az esetben, ha az e egyenes nem egy háromszög lapját metszi, hanem annak egy pontját, akkor a 43. ábrának megfelelően járjunk el, azaz a metszett ponthoz tartozó lapokat kössük a p_i ponthoz.



43. ábra Az egyenes egy pontot metsz

Ezzel az SPCR minden lépését ismerjük.

2.1.4.1 Elemzés

Az SCPR egy olyan adaptív felület rekonstrukciós algoritmus, amely semmilyen megkötést nem tesz a felmért pontok számáról, struktúrájáról. Lehetővé teszi, hogy ritka és nem egyenletes pontfelhőből egy olyan háromszöghálót generáljunk, amely nem tartalmaz lyukakat és az eredeti mérés összes poligonpontját tartalmazza.

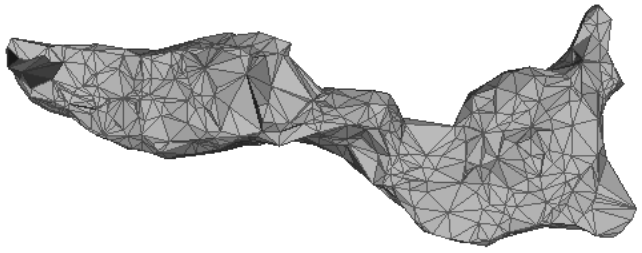
Az első olyan mérést, amely már nem a szelvényes technika alapján készült egy olyan barlangban végeztem, amelyet közel másfél hónapja a szerző és társai fedeztek fel a Pál-völgyi kőfejtőben. Az új barlang végül a Meta-barlang nevet kapta, amely az utóbbi napokban nagyobb figyelmet kapott, hiszen ez a barlang hozta meg a Pál-völgyi kőfejtő két nagy barlangrendszere közötti összekötést 2011. december 11-én. Az összekötött barlangrendszert Szépvölgyi-barlangrendszernek nevezték el, amely jelenleg Magyarország leghosszabb barlangja.

44. ábra A Meta-barlang bejárata a felfedezés estéjén és egy héttel később

A mérést körülbelül 2 óra alatt fejeztem be, amelynek az eredménye a következő ábrán látható 1500 elemű ponthalmaz lett. A mérés során hatalmas szabadságot adott, hogy bárhonnan bármelyik pontot fel tudtam mérni.

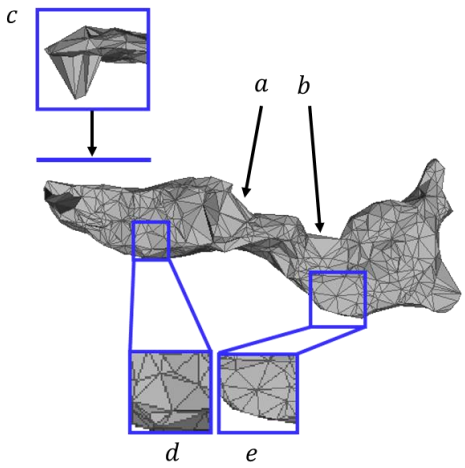
45. ábra Meta-barlang ponthalmaza

A kiterjesztett α-shape algoritmus eredménye a következő háromszögháló lett.



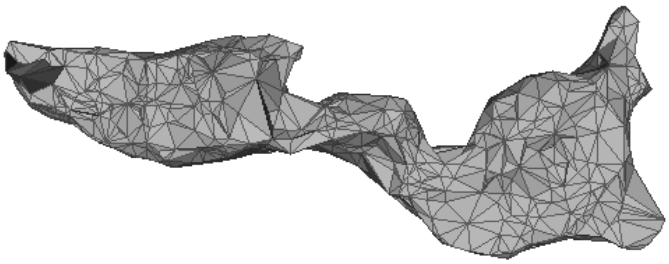
46. ábra Az α-shape eredménye

Vizsgáljuk meg kicsit jobban ezt a hálót. A 47. ábra *c* pontja a felülnézet egy részletet mutatja, amely azt igazolja, hogy az algoritmus nem generál szinguláris háromszögeket. A *d* és *e* pontok a háló különböző részletét mutatják kinagyítva, amelyen tisztán látható, hogy a kiterjesztett α-shape eltérő lokális α értékeket használ. Az *a* és *b* pontok pedig azokat a háromszögeket mutatják, amelyek miatt belső pontok vannak. Összehasonlítva az eredeti ponthalmazzal az *a* és *b* pontok helyén nem szabadna háromszögeknek lennie.



47. ábra

Az *a* és *b* pontoknál megjelent felesleges háromszögeket, és ezzel együtt a belső pontokat az SPCR fogja eltávolítani, amelynek eredményét a következő ábra mutatja.



48. ábra Az SPCR eredménye

Hasonlítsa össze a kedves Olvasó az eredeti ponthalmazzal. Véleményem szerint az SPCR jól teljesített az első mérési adatokon, a fenti képek azt bizonyítják, hogy tökéletesen alkalmazható barlangokban.

3. Algoritmusok

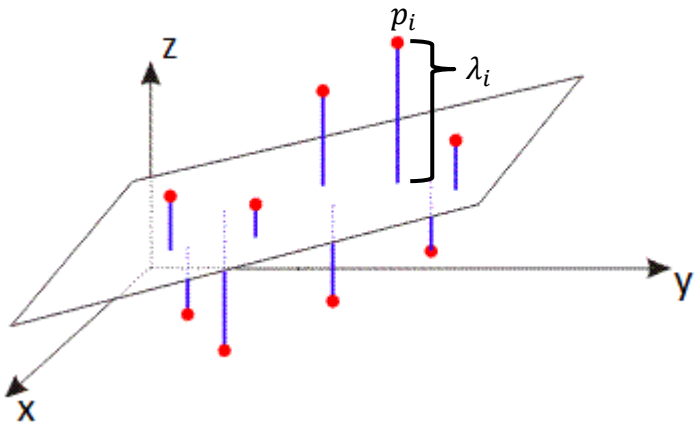
3.1 Regressziós sík

Legyen adott egy m méretű ponthalmaz $\{(x_i, y_i, z_i)\}_{i=1}^{m}$ alakban, és tételezzük fel, hogy a *z* komponens funkcionálisan függ az *x* és az *y* komponenstől. A feladat, hogy egy olyan síkot találjunk, amely valamilyen értelemben a legjobb a ponthalmazhoz.

A következőkben két regressziós problémát vizsgálunk meg: az első a normál regressziós síkot találja meg, a második pedig az ortogonális regressziós síkot.

3.1.1 Normál regressziós sík

Adott a pontok halmaza $\{p_i = (x_i, y_i, z_i)\}_{i=1}^{m}$ alakban. A feladat, hogy megtaláljuk azon *A,B,C* paramétereket, amelyekkel alkotott $z = Ax + By + C$ sík minimális abban az értelemben, hogy a z_i komponens és a sík paraméterei által meghatározott $Ax_i + By_i + C$ érték közötti hiba négyzetes összege a legkisebb. Tehát a regressziós síktól számított vertikális eltérést próbáljuk minimalizálni, amint azt a 49. ábrán is láthatjuk.



49. ábra Normál regressziós sík

Definiáljuk a hibafüggvényt a következő alakban:

$$E(A,B,C) = \sum_{i=1}^{m} [(Ax_i + By_i + C) - z_i]^2 = \sum_{i=1}^{m} {\lambda_i}^2$$

Ez egy nem negatív, hiperparabólikus függvény, amely minimuma esetén a gradiens minden komponense nulla, azaz $\nabla \mathrm{E} = (0,0,0)$. A minimum megkeresése tehát

egy lineáris egyenletrendszer megoldásához vezet, melyben *A,B* és *C* paraméterek az ismeretlenek.

$$(0,0,0) = \nabla \mathrm{E} = 2\sum_{i=1}^{m}[(Ax_i + By_i + C) - z_i]\,(x_i, y_i, 1)$$

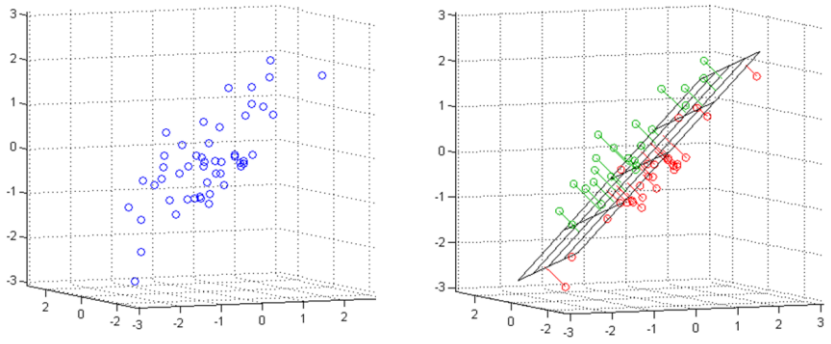
A fenti egyenletrendszert könnyedén megoldhatjuk valamely eliminációs módszerrel, majd az *A,B,C* paramétereket felhasználva felírhatjuk a regressziós síkot $z = Ax + By + C$ alakban.

Az egyenletrendszer kifejtve mátrixos alakban:

$$\begin{bmatrix} \sum_{i=1}^{m} x_i{}^2 & \sum_{i=1}^{m} x_i y_i & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i y_i & \sum_{i=1}^{m} y_i{}^2 & \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} y_i & \sum_{i=1}^{m} 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_i z_i \\ \sum_{i=1}^{m} y_i z_i \\ \sum_{i=1}^{m} z_i \end{bmatrix}$$

3.1.2 Ortogonális regressziós sík

Legyen adott a ponthalmaz $\{X_i = (x_i, y_i, z_i)\}_{i=1}^{m}$ alakban, és határozzuk meg azt a síkot, amely nem a pont és a regressziós sík közötti vertikális eltérést próbálja minimalizálni, hanem a pont és a sík közötti ortogonális távolságot. Ezt a síkot nevezzük a ponthalmazhoz tartozó ortogonális regressziós síknak. A következő ábra bal oldalán a ponthalmaz elemei láthatóak, jobbra pedig a legjobb ortogonális sík, amely esetén a pontok sík közötti ortogonális távolság négyzete minimális.

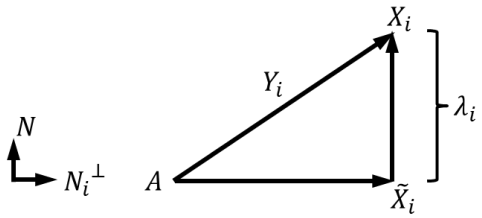


50. ábra Ortogonális regressziós sík

Legyen a sík vektoros egyenlete $N \cdot (X - A)$ alakú, ahol N jelöli a sík egység hosszú normálvektorát, A pedig a sík egy pontját. Vegyük észre, hogy a ponthalmaz összes elemét felírhatjuk a következő alakban:

$$X_i = A + \lambda_i N + p_i N_i^{\perp}$$

ahol $\lambda_i = N \cdot (X_i - A)$ és $N_i^{\perp}$ pedig az N vektorra merőleges normalizált vektor p_i. együtthatóval. Ne foglalkozzunk egyelőre azzal a ténnyel, hogy végtelen sok ilyen vektor van, hiszen $N_i^{\perp}$ nem fog megjelenni az egyenletekben, p_i-vel együtt csak a megértést segítik elő.



51. ábra Minden pont felírható $A + \lambda_i N + p_i N_i^{\perp}$alakban

Az 51. ábrán a regressziós sík merőleges a képernyőre. A sík normálvektorát N jelöli, az X_i pont vetületét a regressziós síkra pedig $\tilde{X}_i$. A vetület és a pont közötti távolságot λ_i jelöli.

Definiáljuk még az Y_i vektort az $X_i$és az A vektor különbségeként, azaz $Y_i = X_i - A$. Ekkor az X_i és $\tilde{X}_i$ pont távolsága $\lambda_i = N \cdot Y_i$ lesz.

Írjuk fel a minimalizálandó hibafüggvényt a fentieket felhasználva:

$$E(A, N) = \sum_{i=1}^{m} {\lambda_i}^2 = \sum_{i=1}^{m} (N \cdot Y_i)^2$$

Mivel a skaláris szorzás kommutatív, és felírható vektoros alakban is ($a \cdot b = a^T b$) az előző függvénynek két további alakját tudjuk megadni:

$$E(A, N) = \sum_{i=1}^{m} \left({Y_i}^T [N N^T] Y_i \right)$$

$$E(A, N) = N^T \left(\sum_{i=1}^{m} Y_i {Y_i}^T \right) N = N^T M(A) N$$

Az A paramétert az első, az N paramétert a második alak segítségével fogjuk megadni.

Bizonyítható, hogy a regressziós sík minden esetben tartalmazza a pontok által meghatározott középpontot. Az A meghatározásához vegyük az első alak A szerinti parciális deriváltját:

$$\frac{\partial E}{\partial A} = -2[NN^T]\sum_{i=1}^{m} Y_i$$

A parciális derivált pontosan akkor lesz 0, ha $\sum_{i=1}^{m} Y_i = 0$, azaz ha $A = \frac{1}{m}\sum_{i=1}^{m} X_i$. Tehát megkaptuk a kívánt eredményt, a regressziós sík tartalmazni fogja a centroidot.

Adott A mellett határozzuk meg az $N^T M(A) N$ miminumát. Az *M(A)* együttható mátrix minden eleme ismert, hiszen $\sum_{i=1}^{m} Y_i {Y_i}^T = M(A)$ és az $Y_i = X_i - A$ vektor komponenseit az $X_i$és A vektorok egyértelműen meghatározzák.

A minimum meghatározásához a legkézenfekvőbb megoldást a Rayleigh-kvóciens adja, ami egy hermetikus *M* mátrix és egy nem nulla *x* vektor esetén felírható a következő alakban:

$$R(M, x) = \frac{x^T M x}{x^T x}$$

Egy négyzetes mátrix hermetikus, ha megegyezik a transzponáltjának konjugáltjával. Mivel a valós tér felett dolgozunk, ez azzal ekvivalens, hogy a mátrix szimmetrikus. Az triviális, hogy az *M* mátrix négyzetes.

Bizonyítható, hogy a Rayleigh-kvóciens minimuma az *M* mátrix legkisebb λ_{min} sajátértékével lesz egyenlő abban az esetben, ha az *x* vektor a legkisebb sajátértékhez tartozó sajátvektor. (Ehhez az is kell, hogy a mátrix pozitív szemidefinit legyen). A Rayleight-kvócienst felhasználva a regressziós probléma gyakorlatilag egy sajátérték problémára redukálódott. Az *M* mátrix legkisebb λ_{min} sajátértéke lesz a hibafüggvény minimuma, a λ_{min} sajátértékhez tartozó sajátvektor pedig a regressziós sík normálvektora.

Már csak azt kell bizonyítanunk, hogy a $\sum_{i=1}^{m} Y_i {Y_i}^T = M(A)$ mátrix szimmetrikus. Azt tudjuk, hogy szimmetrikus mátrixok összege is szimmetrikus mátrixot eredményez, ezért elegendő megvizsgálnunk, hogy az $Y_i {Y_i}^T$ mátrix szimmetrikus –e, azaz $(Y_i {Y_i}^T)^T = Y_i {Y_i}^T$. A transzponálás $(AB)^T = B^T A^T$ tulajdonságát felhasználva már ez is egyértelmű, hiszen $(Y_i {Y_i}^T)^T = Y_i {Y_i}^T$.

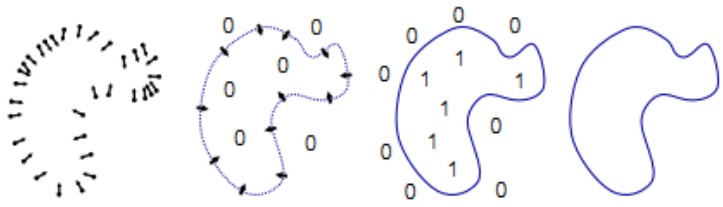
A probléma zárásaként még annyit fűznék hozzá, hogy a Rayleigh-kvóciens helyett használhattunk volna SVD dekompozíciót, illetve az ortogonális sík kereséséhez főkomponens analízist, amely szintén egy sajátérték problémára vezet.

3.2 Felületrekonstrukció

3.2.1 Poisson felületrekonstrukció

A Poisson felületrekonstrukció teljesen új szemléletet követve egy Poisson egyenlet megoldásával állítja elő a mérési adatokból a modellt. A módszer egy olyan λ indikátor függvényt keres, amelynek a modellen belül 1 az értéke, mindenhol máshol pedig 0. A rekonstruált felületet egy iso felület adja az indikátor függvényt felhasználva.

Az algoritmus kulcsa a modell (*M*) indikátor függvénye és a modell orientált felületi pontjai közötti kapcsolat. Vegyük észre, hogy az indikátor függvény gradiense egy olyan vektorteret eredményez, amely mindenhol nulla, kivéve a felület közelében, ahol a felületi normálisok ellentett vektorával egyenlő. Éppen ezért az orientált pontokat tekinthetjük a modellhez tartozó indikátor függvény gradiensének elemeiként. A megérést segíti a következő ábra, amely 2D-ben szemlélteti a fentieket.



52. ábra Az orientált pontok, a gradiens mező, az indikátor függvény és a modell

Az indikátor függvény megtalálása innentől kezdve a gradiens operátor invertálására redukálódik, azaz találjunk egy olyan λ skalárfüggvényt, amelynek a gradiense legjobban közelíti a mérési eredményeket tartalmazó $\vec{V}$ vektorteret, azaz a kettő közötti különbség éppen λ esetén éri el a minimumot:

$$\min\|\nabla\chi - \vec{V}\|$$

Ideális esetben, ha a modellt nem a mérési eredmények ismeretében kellene elkészítetnünk, hanem az már a mérés előtt rendelkezésre állna, akkor a fenti minimum 0 lenne, azaz:

$$\nabla\chi = \vec{V}$$

A fenti egyenlőséget akkor tudjuk a legegyszerűbben megoldani, ha alkalmazzuk a divergencia operátort, így az egész probléma egy Poisson egyenletté alakul. A Poisson egyenlethez már léteznek hatékony megoldók (Green függvénye, relaxációs technika), amelyek egy olyan λ skalárfüggvényt eredményeznek, amelyre alkalmazva a Laplace (Δ) operátort a $\vec{V}$ vektortér divergenciáját kapjuk:

$$\Delta\chi \equiv \nabla \cdot \nabla\chi \equiv \operatorname{div}\operatorname{grad}(\chi) = \nabla\vec{V} \equiv \operatorname{grad}(\vec{V})$$

A felületrekonstrukciót Poisson problémaként formalizálva további előnyöket tapasztalhatunk:

- Nem kell az adatokat különböző heurisztikák alapján kisebb szegmensekre bontanunk, lokális becsléseket és rekonstrukciókat készítenünk majd illesztenünk. Az algoritmust elég a teljes adathalmazon egyszer lefuttatni
- Zajos adatok esetén is elfogadható eredményt kaphatunk

3.2.1.1 Megoldás lépésről lépésre

A bemeneti S adathalmaz mérési eredmények egy olyan $s \in S$ halmaza, amelynek minden eleme egy $s.p$ pontból és egy olyan befele mutató $s.\vec{N}$ normálvektorból áll, amely az ismeretlen M modell ∂M felületén, vagy annak közelében fekszik. A célunk a modell indikátor függvényének approximálásával a modell felületének egy elfogadható, közelítő rekonstrukcióját adni.

1. A gradiens mező definiálása

Mivel az indikátor függvény egy szakaszosan konstans függvény, nem vehetjük azonnal a gradiensét, hiszen határtalan értékeket kapnánk. Azért, hogy ezt elkerüljük egy simító függvényt alkalmazunk, majd ennek vesszük a gradiensét. A következő lemma formalizálja a kapcsolatot a simított indikátor függvény és a felület normálvektorai között.

Lemma: Egy ∂M felülettel rendelkező térbeli M modell esetén legyen λ_M a modell indikátor függvénye, $\vec{N}_{\partial M}(p)$ a felület $p \in \partial M$ pontja esetén a felületi normális ellentett vektora, $F(q)$ a simító függvény, $F_p(q-p)$ a simító függvény eltolása a p pontba. Ekkor a következő egyenlőség áll fent:

$$\nabla(\lambda_M * \mathrm{F})(q_0) = \int_{\partial M} F_p(q_0)\,\vec{N}_{\partial M}(p)\,dp$$

Bizonyítás:

$$\frac{\partial}{\partial x}\Big|_{q_0}(\lambda_M * \mathrm{F}) = \frac{\partial}{\partial x}\Big|_{q_0}\int_M F(q-p)\,dp = \int_M \left(-\frac{\partial}{\partial x}F(q_0-p)\right)dp$$

2. A gradiens mező közelítése

Mivel nem tudjuk a felület pontos geometriáját, ezért az előző lemmában írt felületi integrált nem tudjuk kiszámolni. Szerencsére a bemeneti pontok elég információt adnak ahhoz, hogy ezt az integrált egy diszkrét összegzéssel közelítsük. Az *S* ponthalmazt felhasználva particionáljuk a ∂M felületet diszjunkt $\aleph_s \in \partial M$foltokra**.** Az integrált most már közelíthetjük a $\aleph_s$ foltok segítségével, felhasználva az *s.p* pont érékét és a folt területét:

$$\nabla(\lambda_M * \mathrm{F})(\mathrm{q}) = \sum_{s\in S}\int_{\aleph_s} F_p(\mathrm{q})\,\vec{N}_{\partial M}(p)dp \approx \sum_{s\in S}|\aleph_s|F_{s.p}(q)s.\vec{N}(p)dp \equiv \vec{V}(q)$$

3. Poisson probléma megoldása

A gradiens mező ismeretében most már meghatározhatjuk a $\tilde{\chi}$ indikátorfüggvényt.

4. Isofelület generálás

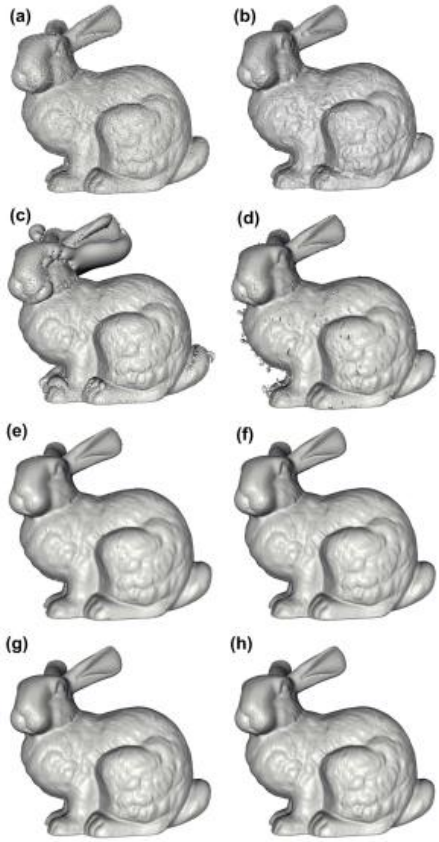
Annak érdekében, hogy elkészítsük a $\partial\tilde{M}$ rekonstruált felületünket találnunk kell egy isoértéket az indikátor függvény felhasználásával, amelyre aztán egy iso felületet illesztünk.

Az iso értéknek az indikátor függvény mérési pontokban vett érékének átlagát vesszük:

$$\partial\tilde{M} = \{q \in R^3 |\, \tilde{\chi}\,(q) = \gamma\}, ahol\ \gamma = \frac{1}{|S|}\sum_{s\in S}\tilde{\chi}(s.p)$$

3.2.1.2 Elemzés

A Poisson algoritmust alapvetően lézerszkennerek által produkált pontfelhők háromszögesítéséhez dolgozták ki. A publikált rekonstrukciós algoritmusok általában a Stanford Egyetem által készített 3D Scanning Repository pontfelhőit használják. Ennek egyik előnye, hogy a pontfelhők viszonylag jó minőségűek, hiszen viszonylag sok pontból állnak és nem tartalmaznak nagy lyukakat, vagy éles ugrásokat. Másik nagy előnye, hogy bárki számára ingyenesen hozzáférhető.



53. ábra Standford bunny rekonstrukciója

Az 53. ábra több rekonstrukciós algoritmust hasonlít össze, amelyek a Standford Bunny közel 360000 pontból álló pontfelhőjét dolgozták fel. A megfelelő képhez tartozó algoritmusok:

a) Power Crust
b) Robust Cocone
c) Fast RBF
d) MPU
e) Hoppe
f) VRIP
g) FFT-alapú rekonstrukció
h) Poisson rekonstrukció

Az 53. ábrán talán nehezen kivehető, de a Poisson algoritmus kiemelkedően szép eredményt ad a többi algoritmushoz képest.

Algoritmus	Idő (s)	Memória (MB)	#Háromszögek
Power Crust	380	2653	554 332
Robust Cocone	892	544	272 662
Fast RBF	4919	796	1 798 154
MPU	28	260	925 240
Hoppe	70	330	950 562
VRIP	86	186	1 038 055
FFT	125	1684	910 320
Poisson	263	310	911 390

1. táblázat: Poisson futási eredményei

Ha figyelembe vesszük az előző táblázatban megadott teljesítmény adatokat is, akkor a Poisson algoritmus egy nagyon jó választásnak tűnik. Azok az algoritmusok, amelyek gyorsabban futnak (MPU, Hoppe, VRIP) zajos háromszöghálót eredményeztek.

3.2.2 BPA

A ball-pivoting algorithm (BPA) egy olyan interpolációs felület rekonstrukciós algoritmus, melynek az elve nagyon egyszerű: egy δ sugarú labdát görgetünk végig a pontfelhőn, egészen addig, amíg nem találunk olyan pontot, amelyet nem érintett még a labda. A következő bekezdésben kicsit formálisabb definíciót adunk.

Legyen M egy háromdimenziós objektum felülete, S pedig a felület mintapontjainak halmaza. Tételezzük fel, hogy S elég sűrű ahhoz, hogy egy δ sugarú gömb, továbbiakban δ labda ne tudjon áthatolni a felületen mintapont metszése nélkül. Vegyük ezt a pontot, és mozgassuk úgy a δ labdát, hogy az három mintapontra illeszkedjen, köztük a kezdőpontra is úgy, hogy a labda más pontot ne tartalmazzon. A három mintapontból képzett háromszög a háló első eleme. A labdát elkezdjük forgatni úgy, hogy az előző ponthármas közül két pontot továbbra is érintsen. A forgatást egészen addig végezzük, míg a labda egy harmadik pontot nem metsz. Az új ponthármas lesz a háló második eleme. Az eljárás eredményeként a háló folyamatosan növekszik egészen addig, amíg a labda forgatása új pontot nem talál. Arra kell figyelnünk, hogy a forgatást mindig a háló határoló élein végezzük, így felesleges lépéseket spórolunk meg.

Vizsgáljuk meg kicsit jobban az algoritmust. A BPA bemenete o_i mintapontok halmaza, amelyekhez az n_i normálvektor tartozik, valamint a d sugár. Az első lépés egy forrás háromszög keresése, amelyeknek csúcsai rendre o_i, o_j, o_k. A pontok csak akkor

felelnek meg, ha azok érintik a *d* sugarú gömböt, és a gömb más pontot nem tartalmaz. Ha elsőre olyan hármast találunk, amelyek ezt a kritériumot nem teljesíti, akkor keressünk új forrás háromszöget.

Legyen *F* élek láncolt listája, amelyet a forrás háromszög csúcsai által alkotott hurokkal inicializálunk. Jelölje $e_{(i,j)}$ o_i, és o_j pontok közötti élt, o_o a háromszög harmadik pontját, illetve $c_{i,j,o}$ a három pontot érintő gömb középpontját. Osszuk fel az éleket aktív és határoló élekre. Az aktív élek azok, amelyeket még fel fogunk használni forgatásra. Azokat az éleket nevezzük határoló éleknek, amelyek körül sikertelen volt a forgatás.

A BPA algoritmusa bemenete a mintapontok *S* halmaza és a gömb *d* sugara.

```
while (true) do
       while (e(i,j) = get_active_edge(F)) do
              if (ok = ball_pivot(e(i,j)) && (not_used(ok) || on_front(ok)))
                     output_triangle(oi, oj, ok)
                     join(e(i,j), ok, F)
                     if (contains(F, e(k,i))) glue(e(i,k), e(k,i),F)
                     if (contains(F ,e(j,k))) glue(e(k,j), e(j,k),F)
              else
                     mark_as_boundary(e(i,j))
       done

       if ((oi, oj, ok) = find_seed_triangle())
              output_triangle(oi, oj, ok)
              insert_edge(e(i,j),F)
              insert_edge(e(j,k),F)
              insert_edge(e(k,i),F)
       else
              return
done
```

A fenti kódban a következő metódusokat használjuk:

- *get_active_edge(F)* – az F listából kiválaszt egy aktív élt
- *ball_pivot($e_{(i,j)}$)* – a labdát forgatja az $e_{(i,j)}$ él körül, eredményül pedig a forgatás közben metszett o_k pontot adja
- *join($e_{(i,j)}$, o_k, F)* – a hálóhoz adja a két új élet
- *glue($e_{(i,k)}$, $e_{(k,i)}$,F)* – ellentmondások szüntet meg
- *find_seed_triangle()* – egy megfelelő forrás háromszöget keres

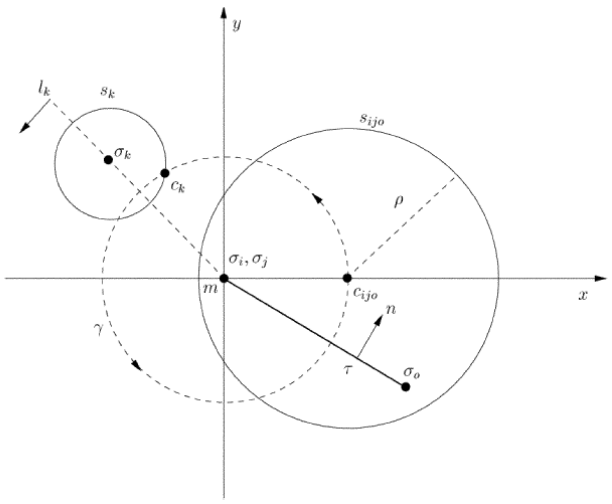
Az *find_seed_triangle* eljárás lépései a következők:

1. Válasszuk *S*-nek egy olyan *o* elemét, amelyet eddig nem használt az algoritmus
2. Vegyük a *o* ponthoz két legközelebbi szomszédját, ezek legyenek o_a és o_b
3. Készítsünk egy potenciális forrás háromszöget az *o*, o_a, o_b ponthármasból

4. Próbáljuk egy *d* sugarú gömböt feszíteni a három pont köré úgy, hogy a gömb más pontot ne tartalmazzon
5. Ha ez sikerült, akkor a háromszög egy megfelelő forrás háromszög, ellenkező esetben keressünk egy újabb *o* pontot és ismételjük meg az egész algoritmust

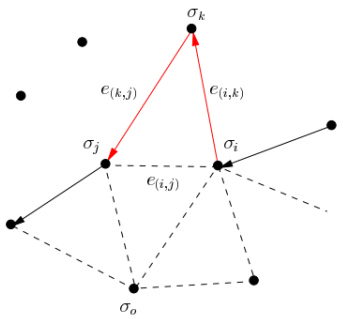
A forgató *ball_pivot* metódus bemeneti paraméterei egy $T=(o_i, o_j, o_o)$ háromszög és az ezeket a pontokat érintő gömb (labda) *d* sugara. Tételezzük fel, hogy $e_{(i,j)}$ él körül kell elvégeznünk a forgatást. A $c_{i,j,o}$ középpontú labda kezdeti helyzetében nem tartalmaz a három ponton kívül más pontot, mivel *T* vagy egy forrás háromszög, vagy egy előző forgatás eredményeként kapott háromszög. A forgatás a labda él körüli mozgatása, miközben a labda továbbra is érinti az $e_{(i,j)}$ él két végpontját. A labda $c_{i,j,o}$ középpontja egy forgatás során egy $\gamma = \|c_{i,j,o} - m\|$ sugarú kört ír le, amely az $e_{(i,j)}$ élre merőleges síkon fekszik, a középpontja pedig az $e_{(i,j)}$ él *m* felezőpontja, ahol $m = \frac{1}{2(o_i + o_j)}$. A labda forgatása közben metszhet új pontot, legyen ez pont o_k. Ha a forgatás végeztével nem találtunk új pontot, akkor az élet határoló élként jelöljük meg, ellenkező esetben az (o_i, o_j, o_k) egy megfelelő háromszög, amely nem tartalmaz belső pontot.

Az o_k pontot a következő módon keressük meg. Vesszük az *m* pont *2d* sugarú környezetében lévő összes o_x pontot. Minden o_x pontra kiszámoljuk az (o_i, o_j, o_x) pontokra illeszkedő gömb c_x középpontját, ha egyáltalán létezik ilyen gömb. Minden c_x középpont az *m* pont körüli γ körön fekszik. A c_x középpont kiszámolásához vegyünk egy *d* sugarú gömböt az o_x pont körül, és vegyük a gömb és a γ kör metszéspontját. A c_x pontok közül azt választjuk ki, amely a γ körön az óramutató járásával ellentétes irányban körbejárva a $c_{i,j,o}$ ponthoz a legközelebbi.



54. ábra A labda forgatása

A join műveletet akkor használjuk, amikor az $e_{(i,j)}$ él körüli forgatás eredményeként egy olyan o_k pontot metsz a labda, amely az eddigi hálónak még nem volt eleme. Ebben az esetben eredményként kiadjuk az (o_i, o_j, o_k) háromszöget, majd módosítjuk az F listát: eltávolítjuk az $e_{(i,j)}$ élet és helyette az $e_{(i,k)}$, $e_{(k,j)}$ éleket adjuk hozzá.



55. ábra Join művelet

A glue művelet az ellentmondásos éleket távolítja el a listából. Például amikor az $e_{(i,k)}$ élt a listához adjuk a join művelettel, ellenőrizni kell, hogy az $e_{(k,i)}$ él eleme–e a listának, azaz hogy egy hurok alakul-e ki. Ha igen, akkor a két ellentmondásos élt egy éllel helyettesítjük.

3.2.2.1 Elemzés

A BPA a Poissonnál sokkal egyszerűbb rekonstrukciós algoritmus, amely lineáris mind a lépésszám, mind a felhasznált tár tekintetében. Az első C++ implementáció nem haladta meg a 4000 sort. A legtöbb lépése $O(1)$ ellenőrzésekből és

struktúrafrissítésekből áll. A két legköltségesebb műveletre, a *ball_pivot* és a *find_seed_triangle* eljárásokra is $O(n)$ korlátot tudunk mondani. Azt várjuk tehát, hogy az algoritmus gyorsabban fog futni, mint a Poisson rekonstrukció. A következő táblázat a BPA eredményeit mutatja a Standford Egyetem 3D Scanning Repository különböző pontfelhőin.

Pontfelhő	#Pontok	δ	Idő(s)	Memória(MB)	#Háromszögek
Bunny	361 K	0.3	7	86	714 560
Dragon	2.0 M	0.5	30	228	3 459 200
Buddha	3.3 M	0.2	74	325	5 230 655

2. táblázat: BPA futási eredményei

A táblázat igazolja az elvárásainkat. A BPA a Standford Bunny pontfelhőből 7 másodperc alatt készítette el a felületi modellt, míg a Poisson algoritmusnak ehhez 263 másodpercre volt szüksége. A következő ábrán láthatjuk az eredményt is.

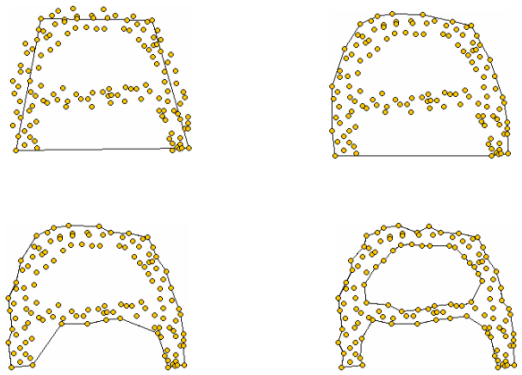
56. ábra Standford bunny

Sűrű pontfelhők esetén, amelynek a sűrűsége állandónak mondható a BPA szép eredményt ad, és ami még nagyon fontos, hogy mindezt $O(n)$ korláttal teszi.

3.2.3 Alpha-shapes

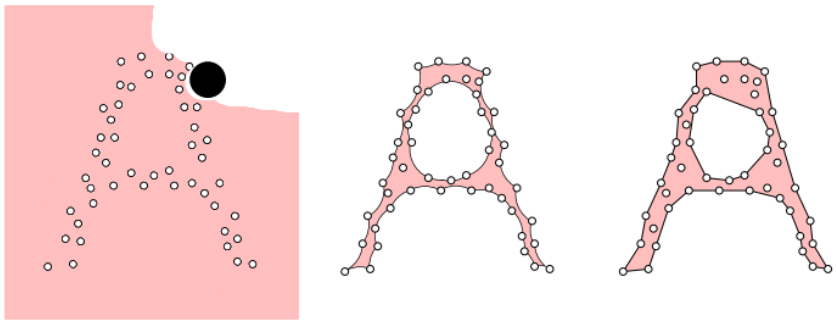
Tételezzük fel, hogy adott háromdimenziós pontok S halmaza, és meg szeretnénk határozni azt a formát, amelyet a pontok alkotnak. Az „forma” egy elég bizonytalan fogalom, amely éles kontrasztban áll a már megszokott térfogat, távolság fogalmakkal. A „forma” fogalomnak több interpretációja lehet, melyek közül az egyik az alpha-shape, vagy röviden α-shape.

Az α-shape algoritmus az α paramétertől függően különböző formát rendel az S ponthalmazhoz. A következő ábrán az algoritmus eredményét láthatjuk különböző α paraméterezéssel.



57. ábra A ponthalmaz különböző formái

A következő értelmezése egy informális definíciót ad az α-shapre. Gondoljunk a pontok $\mathbb{R}^3$ terére, mint egy hatalmas habszivacsra, amelyben a pontok kis szilárd kövek formájában vannak beékelődve különböző helyeken. Képzeljünk el egy gömb formájú α sugarú radírt. A radírral a habszivacs minden olyan részét kivágjuk, amelyek elérhetőek anélkül, hogy szilárd pontokba ütköznénk. Az eredményt, amelyet kaptunk nevezzük el α-buroknak. Cseréljük le az íves éleket egyenes élekre, a radír gömb formájú behatolásait pedig háromszögekre. Ezzel megkaptunk az α-shapet. A következő ábrán egy kétdimenziós példát látunk. Az ábra bal oldalán a habszivacs radírozásának első néhány lépése látható, középen az α-burok, jobb oldalon pedig az α-shape.



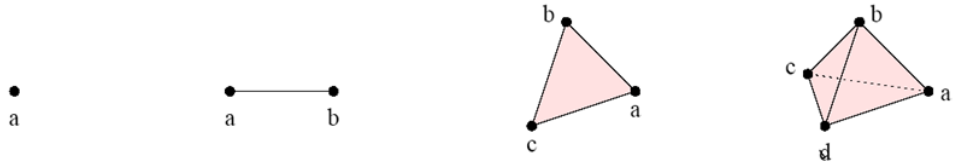
58. ábra Ponthalmaz, α-burok és α-shape

Következzen a formális definíció. Legyen S az $\mathbb{R}^3$ tér pontjainak egy véges halmaza, az α valós szám pedig az algoritmus bemenete, amelyre teljesül a következő: $0 \leq \alpha \leq \infty$. Az S ponthalmazhoz tartozó α-shape egy olyan politóp[7], amely nem feltétlenül konvex és nem feltétlenül csak egy komponensből áll. $\alpha = \infty$ esetében az α-

[7] Sima felületekkel rendelkező geometriai objektum. A poligon egy kétdimenziós, a poliéder pedig egy háromdimenziós politóp.

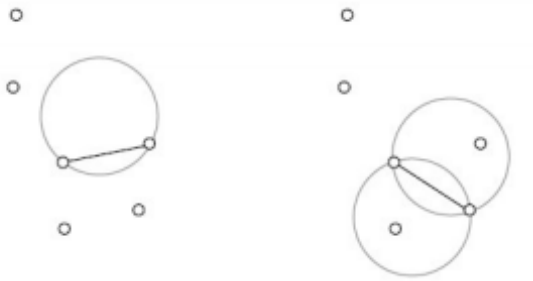
shape a konvex burokkal azonos. Tételezzük fel, hogy a pontok általános helyzetűek, azaz nincs olyan 4 pont, amelyek közös síkon lennének és nincs olyan 5 pont, amelyek egy közös gömb felületén. Definiáljuk az α-labádt, mint egy $0 < \alpha < \infty$ sugarú gömböt, amelynek jelölje a felületét ∂b. A 0-labda a definíció alapján egy pont, a ∞-labda pedig egy nyitott fél tér. A b α-laba üres, ha $b \cap S = \emptyset$. S minden $k+1$ méretű $T \subseteq S$ részhalmaza $0 \leq k \leq 3$ korláttal egy σ_T k-szimplexet határoz meg, amely a T halmaz konvex burka, amelyet jelöljön $conv(T)$.

A szimplex a matematikában a háromszög, illetve a tetraéder általánosítása végesdimenziós vektortérre. *n*-dimenziós vektortérben *n+1* pont konvex burkaként fogalmazható meg. A következő ábrán különböző szimplexeket láthatunk.



59. ábra 0-szimplex, 1-szimplex (él), 2-szimplex (háromszög), 3-szimplex (tetraéder)

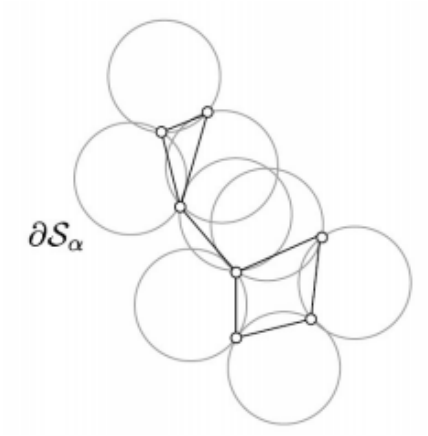
Az általános helyzetű pontok miatt garantált, hogy a k-szimplex valóban k dimenziós. $0 \leq k \leq 2$ esetén az σ_T k-szimplexet nevezzük α-védtelennek, ha létezik üres b α-labda a $T = \partial b \cap S$ halmazzal.



60. ábra α-védtelen és nem α-védtelen 1-szimplexek (élek)

Egy fix α érték esetén az S ponthalmazhoz tartozó α-shapet jelölje S_α. Az α-shape ∂S_α határa nem más, mint az összes α-védtelen k-szimplex összessége.

$$\partial S_\alpha = \{\sigma_T \mid T \subseteq S, 1 \leq |T| \leq 3 \text{ és } \sigma_T \text{ } \alpha - \text{védtelen}\}$$



61. ábra Az α-shape határa

A korábban tett állításaink igazak lesznek:

$$\lim_{\alpha \to 0} S_\alpha = S$$

$$\lim_{\alpha \to \infty} S_\alpha = conv(S)$$

A következő részekben az α-shape meghatározásával foglalkozunk, amelyhez Delaunay triangulációs eljárást fogjuk alkalmazni.

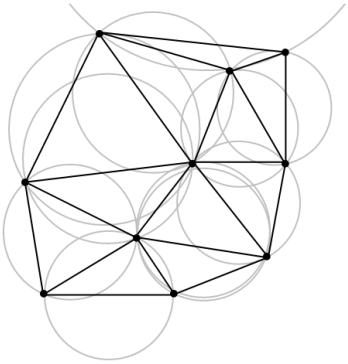
A triangulációs eljárások geometriai objektumokat bontanak szimplexekre. Kétdimenziós esetben pontokból készít háromszögeket, háromdimenziós térben pedig tetraédereket. Az $S \subset \mathbb{R}^d$ pontok *DT(S)* Delaunay triangulációja egy olyan szimpliciális komplex, amely a következő szimplexet tartalmazza:

- Minden olyan d-szimplexet, amelynek határoló gömbje nem tartalmaz más szimplexet.
- Minden olyan d-szimplexet, amely *DT(S)* szimplexnek lapja

A geometriában a szimplexek összességét nevezzük komplexnek. Egy $\mathcal{K}$ komplex szimpliciális, ha teljesíti a következő feltételeket:

- Egy $\mathcal{K}$-ból vett szimplex bármely lapja is $\mathcal{K}$-ban van
- Bármely $\tau_1, \tau_2 \in \mathcal{K}$ szimplexek esetén a metszet mindkét szimplex lapjához tartozik

Kétdimenziós pontok Delaunay triangulációja olyan háromszögeket eredményez, amelyek köré írt kör nem tartalmaz a három ponton kívül más pontot. Ezt szemlélteti a következő ábra is.



62. ábra Delaunay háromszögesítés

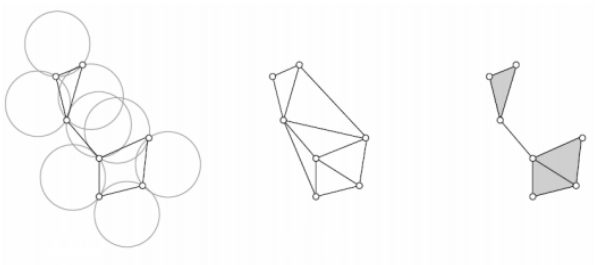
A Delaunay triangulációra építve a következő állítások igazak:

- Ha σ_T az S ponthalmaz α-védtelen szimplexe, akkor $\sigma_T \in DT(S)$
- Minden $0 \leq \alpha \leq \infty$ érték esetén $S_\alpha \in DT(S)$

A fenti állításokat felhasználva a következő egyszerű algoritmussal meghatározhatjuk az összes 2- szimplexet.

```
for each σT 2-szimplex in DT(S)
      if (σT valamely α sugarú körülhatároló gömbje üres)
            σT α-védtelen
```

A 0-szimplexek (pontok) és 1-szimplexek (élek) meghatározásához egy újabb fogalomra, az α-kompexre lesz szükségünk. Az S ponthalmaz $C_\alpha(S)$ α-kompexe a Delaunay háromszögesítés eredményeként kapott *DT(S)* rész komplexe. A $\sigma_T \in DT(S)$ akkor lesz $C_\alpha(S)$ eleme, ha a szimplexet körülhatároló gömb üres, és a sugara kisebb, mint α, vagy σ_T szimplex a $C_\alpha(S)$ komplex bármely szimplexének lapja. Az alpha shape ezek után nem más, mint a $C_\alpha(S)$ szimplexei által meghatározott tartomány, amelyet a következő bekezdésben bizonyítani fogunk.



63. ábra $\partial S_\alpha - az$ α-shape határa, DT(S) a trianguláció eredménye és a $C_\alpha(S)$ α-kompex

Bizonyítható, hogy a következő állítások igazak:

$$\sigma_T \in \partial S_\alpha(S) \Rightarrow \sigma_T \in C_\alpha(S)$$

$$\sigma_T \in \partial S_\alpha(S) \Rightarrow \sigma_T \in \partial C_\alpha(S)$$

$$\sigma_T \in \partial C_\alpha(S) \Rightarrow \sigma_T \in \partial S_\alpha(S)$$

Az első két állítás szerint az α-shapet határoló szimplex az α-komplexnek eleme, sőt határoló szimplexe lesz Az utolsó állítás szerint az α-komplex határoló szimplexe az α-shape határoló szimplexe lesz. Kimondhatjuk tehát a követő tételt:

$$\partial C_\alpha(S) = \partial S_\alpha(S)$$

Az α-shape algoritmusa ezek után a következő:

1. Vegyük az *S* pontjainak Delaunay *DT(S)* háromszögesítését
2. Vegyük az α értékének megfelelő $C_\alpha(S)$ α-komplexet, amelyet *DT(S)* szimplexeiből építünk fel
3. $C_\alpha(S)$ minden szimplexe $S_\alpha(S)$ belső szimplexe lesz, míg $\partial C_\alpha(S)$ szimplexei $S_\alpha(S)$ határoló szimplexei lesznek

3.2.3.1 Elemzés

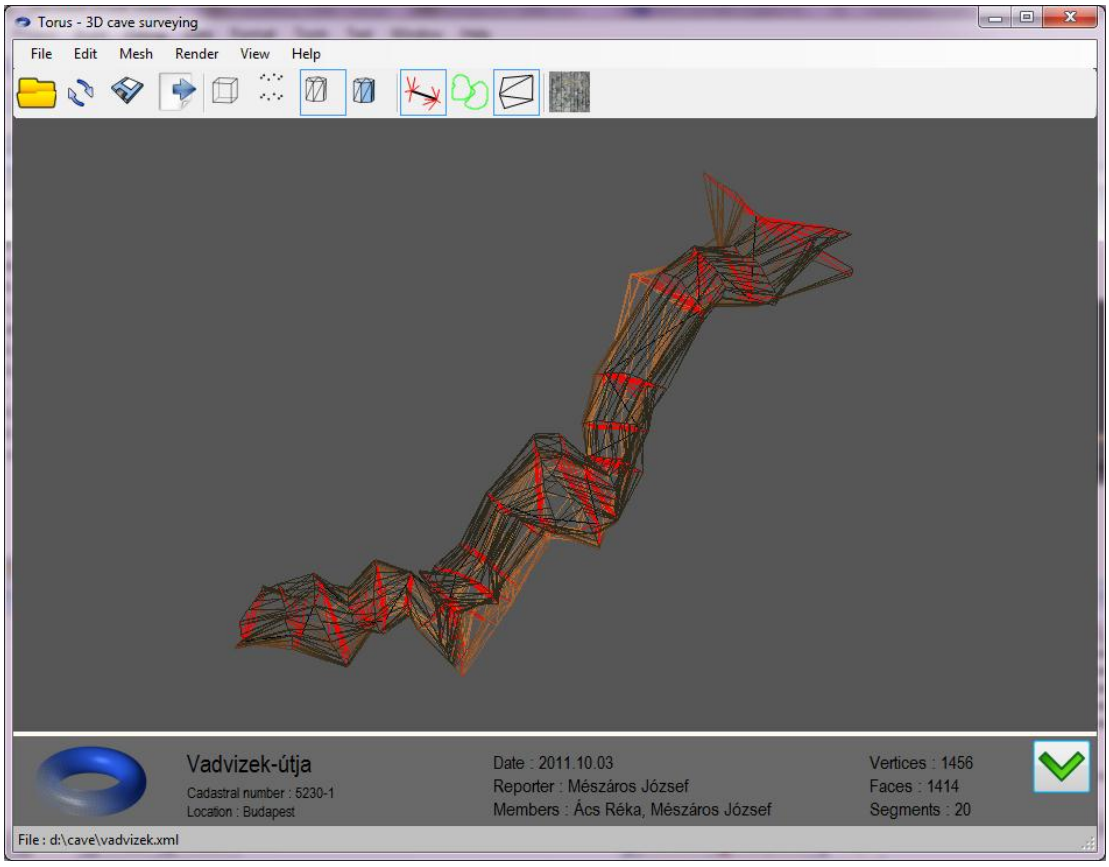
Az α-shape algoritmus legrosszabb esetben is $O(n^2)$ lépésszámot igényel, a felhasznált tár tekintetében pedig lineáris. A következő táblázatban látható, hogy a négyzetes lépésszám 10.000 pont esetén már viszonylag lassú futást eredményez. A legköltségesebb művelet, amelyet nem tudunk kihagyni a Delaunay háromszögesítés. A táblázatban egy olyan algoritmus eredményeit látjuk, amely a Delaunay trianguláció inkrementális flipping változatát használja, amely már önmagában egy $O(n^2)$ -es algoritmus. Léteznek más Delaunay változatok, például az „Oszd meg és uralkodj”, amelyek $O(n\ logn)$ lépésszámot garantálnak.

#Pontok	Idő(s)	Memória(MB)
318	2	0.3
1200	7	2.3
9600	870	67.8

3. táblázat: Az α -shape futási eredményei

4. Tórusz és Toroid

A mérési eredmények felvételéhez, feldolgozásához és a modellek megjelenítéséhez kifejlesztettem egy szoftvert, amely a Tórusz nevet kapta, illetve. készült egy Toroid nevű Androidos program is, amely csak a modellek megjelenítését teszi lehetővé. A következő ábrán a Tórusz program látható.



64. ábra A Tórusz program

A Tórusz funkciói a következők:

- mérési adatok lekérdezése a DistoX-től bluetooth kapcsolaton keresztül
- mérési adatokból $4P^2$ és NP^2 módszerekkel háromszögháló generálása
- importálás
- exportálás
- textúrázás
- felületsimítás

A Tórusz alapvetően a mérési eredmények tartalmazó XML fájl feldolgozásához és az abból készült felületi modell megjelenítéséhez készült. A saját formátumán kívül lehetővé teszi a következő barlangász programok mérési eredményeinek importálását:

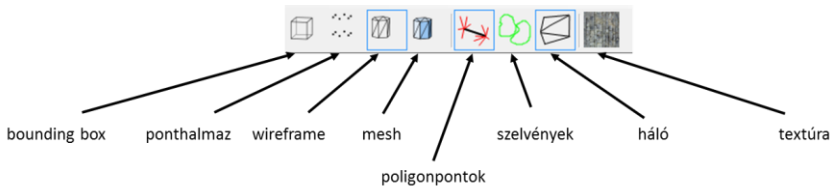
- A magyar Polygon program (.cave)
- Compass (.plt)
- Therion (.th)
- Survex (.svx, .3d)

Importálhatjuk továbbá a következő formátumokat, amelyek nem mérési adatokat tartalmaznak, hanem egy poligonháló elemeit:

- .ply (Standford Triangle Format)
- .pts (CGAL formátum)
- .nrpe

Exportálni jelenleg csak ply formátumba tudunk.

A háromszögháló megjelenítését a menü alatt található ikonokkal tudjuk befolyásolni.

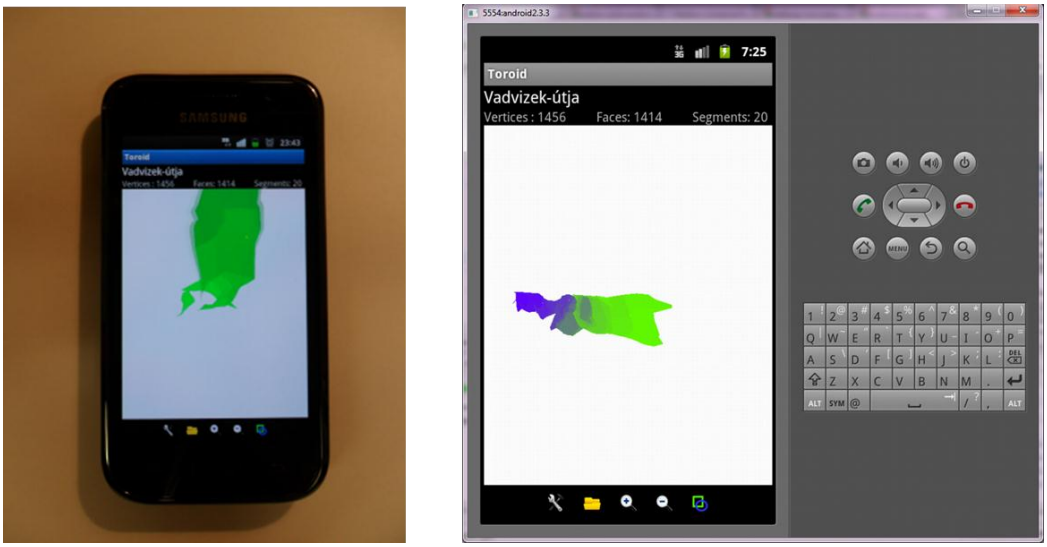


65. ábra A megjelenítés változtatása

Választhatunk, hogy a háromszöghálót határoló téglatest (bounding box), a háló pontjai, a háló váza (wireframe), vagy a teljes háromszögháló jelenjen meg. A szelvényes technikáknál bemutatott poligon- és határpontokat, a szelvényeket és a generált felületi modellt szintén ki-be kapcsolhatjuk.

Az elterjedt modellező programokhoz hasonlóan a Tórusz is lehetővé teszi a háromszögháló forgatását (jobb egérgomb), mozgatását (nyilak) illetve nagyíthatjuk vagy kicsinyíthetjük a képet.

A Tórusz alapvetően az otthoni számítógépekre szánt alkalmazás, amely minden olyan funkciót tartalmaz, amely a DistoX adatainak felületi modellé konvertálásához és megjelenítéséhez kell. Szükség van azonban egy olyan csökkentett funkcionalitású programra is, amelyet a barlangban tudunk használni a már felvett pontok ellenőrzéséhez és megjelenítéséhez. A Tórusz barlangba szánt változata a Toroid nevet kapta, amely egy Androidos alkalmazás.

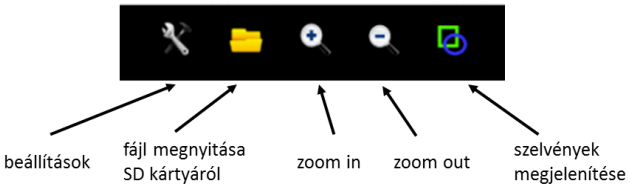


66. ábra A Toroid program egy Samsung okostelefonon és az emulátorban

A Toroid a következő funkciókat tartalmazza:

- háromszögháló megjelenítése
- adatok beolvasása SD kártyáról
- háromszögháló forgatása
- zoomolás
- szelvények megjelenítése

Az alkalmazás alsó menüjének magyarázatát a következő ábra tartalmazza.

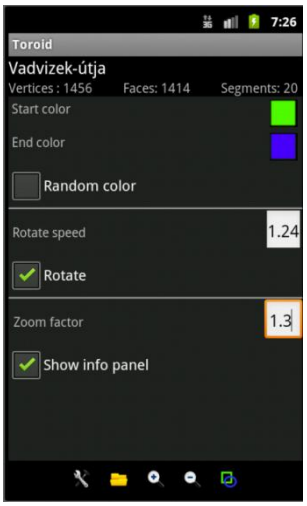


67. ábra Az alsó menü

A Toroid a következő beállításokat teszi lehetővé:

- megjelenítése színek
- forgatás
- forgatás sebessége
- zoomolás mértéke
- infó panel elrejtése

A beállításokat az első gomb megnyomásával hozhatjuk be miután a következő ábrán látható képet kapjuk.



68. ábra A Toroid beállításai

4.1 Technikai részletek

A Tórusz tervezésekor az volt a célom, hogy egy olyan egyszerű program szülessen, amelyet bárki könnyedén használhat a legelterjedtebb operációs rendszereken. Éppen ezért az implementáció során csak olyan komponenseket használtam fel, amelyek több operációs rendszeren is működnek. További fontos szempont volt, hogy a Tórusz mindenféle plusz framework, program vagy library telepítése nélkül futtatható legyen.

A Tórusz jelentős része C++-ban készült, amelynek legfőbb oka, hogy a felhasznált algoritmusok is ezen a nyelven íródtak. Persze bizonyos esetekben léteznek más nyelven írt kevésbé hatékony implementációk, de koránt sem annyi, mint C++-ban. A C++-ot egy rendkívül gazdag és robosztus programozási nyelvnek tartom, amely nagyon hatékony eszköz tud lenni, ha megfelelően bánnak vele.

A nyelv kiválasztása után olyan komponenseket kerestem és használtam fel, amelyek garantálják az első bekezdésben leírt követelményeket.

A felhasználói felület a wxWidget-re épülve készült el, amely egy olyan elterjedt C++ library, amely grafikus alkalmazások fejlesztését teszi lehetővé Windows, Linux és Os X rendszerekre. A wxWidget egyik előnye a többi GUI library-val szemben, hogy az alkalmazások megjelenítéséhez az operációs rendszer natív API-ját használja.

A háromdimenziós megjelenítést az OGRE (Object Oriented Graphics Rendering Engine) végzi, amely egy cross-platform 3D engine. Az Ogre teljesen elfedi az operációs rendszer grafikus könyvtárának (DirectX vagy OpenGL) sajátosságait, amely gyors és kényelmes fejlesztése tesz lehetővé. Biztos felmerül a kedves

Olvasóban, hogy egy wrapper megoldás mindig lassít valamennyit a rendszeren, de az Ogre annyira sok és használható szolgáltatást nyújt (amelyet a Tórusz több moduljában is felhasználok), hogy feledteti ezt a plusz réteget. Az Ogre definiált például egy saját nyelvet, amellyel materialokat adhatunk meg, melyeket később felhasználhatunk a programunkban. A következő részlet egy rövid példát mutat arra, hogyan határozhatjuk meg például egy barlang járatának megjelenítését.

```
material Cave/Brown
{
	technique
	{
		pass
		{
			ambient 0.7 0.1 0.1
			point_size 3.0
			diffuse 0.54 0.27 0.14 1.0
			cull_hardware none
		}
	}
}
```

A Tórusz két elterjedt algoritmus gyűjteményt használt, a CGAL-t és a VCG-t.

A CGAL (Computational Geometry Algorithms Library) hatékony és robosztus geometriai algoritmusok gyűjteménye, amely a legfontosabb algoritmusok implementációját tartalmazza. Ezek közül csak a legfontosabbakat sorolom fel:

- 2D és 3D háromszögesítés
- Geometriai feldolgozás
- Konvex burok
- α-shape
- Felület felosztás

A Tórusz implementálása után a feladatom egy olyan program fejlesztése volt, amelyet a barlangban is lehet használni. Az MKBT tagjaival folytatott beszélgetések során arra jutottam, hogy a leghasználhatóbb egy tableten futó alkalmazás lenne. A két legfontosabb indok:

- A tabletek képernyője már elég nagy ahhoz, hogy a háromszöghálót megfelelő méretben megjelenítse
- A mérete és formája lehetővé teszi, hogy bárhova levigyük a barlangba

Az első tableteket még a Microsoft árulta, de ma már a kereskedelmi forgalomban kapható tabletek jelentős része az Android operációs rendszert használ. Vannak persze Windowsos tabletek, de ezek egyrészt drágábbak, mint az Andoidos

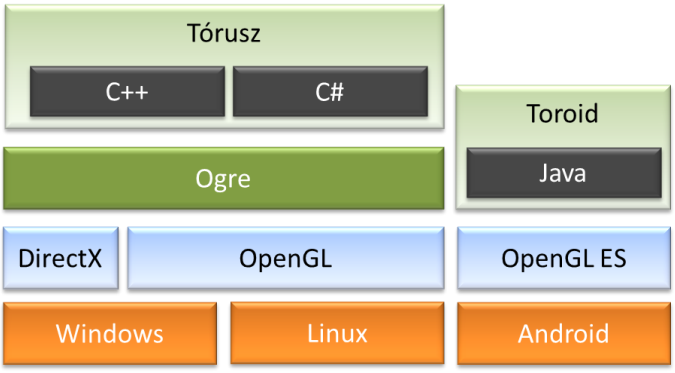
tabletek, másrészt a használatuk nincs az érintőképernyőre optimalizálva. Mivel egy mindenki számára elérhető és könnyen használható programot szerettem volna, azért az Android mellett döntöttem, amely már a világ legelterjedtebb mobil operációs rendszere. Ez persze azzal is járt, hogy az algoritmusok egy részét át kellett írni Java nyelvre.

A legtöbb mobil operációs rendszer, többek között az Android is az OpenGL ES API-ját használja 2D és 3D megjelenítésre. Az OpenGL ES (OpenGL for Embedded Systems) az OpenGL részhalmaza, amely mobil eszközökre lett optimalizálva.

A tervem az volt, hogy az Androidos programban is az Ogre-t fogom használni, de mivel az Ogre portolása még nem teljesen megoldott, ezért ezt az ötletet elvettem. Az Ogre után a jMonkey és a jPCT-AE open source library-kat vizsgáltam meg, de mindkettő használata nehézkesnek bizonyult, illetve nem adtak akkor szabadságot, mint az OpenGL ES API. A fenti két library elvetése után úgy döntöttem, hogy az Android SDK-t használva, plusz réteg beiktatása nélkül implementálom a 3D megjelenítést.

Az Androidos változat a Toroid nevet kapta, amelyet az előző részekben már részletesen bemutattam.

A Tórusz és a Toroid architektúráját mutatja a következő ábra.



69. ábra Tórusz és Toroid architektúrája

4.2 Cave Surveying Markup Language

A Cave Surveying Markup Language (CSML) a mérési eredményeket tartalmazó XML fájl, amelyet a Tórusz és a Toroid is felhasznál. Az XML 3 részből áll:

- leírás
- metaadatok listája
- mérési adatok

A leírás tartalmaz kötelező elemeket, mint például a mérés dátuma, a felmért szakasz neve vagy a mérő csapat tagjai. Ha olyan információt szeretnénk itt megadni, amely nem egy kötelező elem, akkor erre a <*properties*> listát használhatjuk, amely névvel és értékkel rendelkező property-k listáját tartalmazza.

```
<description>
    <name>Vadvizek-útja</name>
    <date>2011.10.03</date>
    <reporter>Joe</reporter>
    <member>Ács Réka</member>
    <member>Mészáros József</member>
    <properties>
        <property name="Location" value="Budapest"/>
        <property name="Duration" value="1h"/>
    </properties>
</description>
```

A metaadatok listája teszi lehetővé, hogy a mérés pontjaihoz információkat csatoljunk. Jelenleg csak szöveges tartalmat kapcsolhatunk egy ponthoz, de ezt később mindenképpen szeretném kiegészíteni képekkel és tudományos információkkal (mint például geológiai adatok).

```
<metaCollection>
    <string name="meta_01">Entrance of the cave</string>
    <string name="meta_02">Top of the pit</string>
</metaCollection>
```

Az utolsó rész, amely a mérési adatok tartalmazza, két részre oszlik:

NP^2 alapú mérési adatok és $4P^2$ alapú mérési adatok. Utóbbi esetben csak poligonpontokat (<*polygonPoint*>) és határpontokat (<*boundaryPoint*>) tudunk megadni, amely NP^2 esetén még a köztes pontokkal (<*middlePoint*>) egészül ki. Minden pont tartalmaz egy <*vertex*> gyerek elemet, amely a pont koordinátái adja meg, továbbá tartalmazhat egy <*metaRef name=”id”*> elemet is, amely a metaCollection-ben definiált meta adatra hivatkozik. Ez a megoldás teszi lehetővé, hogy a metaCollection-ben definiált információkat mérési ponthoz kapcsoljuk. A poligonpont tartalmazhat köztes pontot és határpontot. A köztes pont tartalmazhat további köztes pontokat és határpontokat. A következő oldalon egy NP^2 alapú mérés adatainak egy részletét láthatjuk.

```
<np>
    <polygonPoint>
        <metaRef name="meta_01"/>
        <vertex>
            <x>-0.207832</x>
            <y>-0.410725</y>
            <z>0.398737</z>
        </vertex>

        <boundaryPoint>
            <vertex>
                <x>0.00116916</x>
                <y>0.131228</y>
                <z>0.00582139</z>
            </vertex>
        </boundaryPoint>
        <middlePoint>
            <vertex>
                <x>0.0065365</x>
                <y>0.0469448</y>
                <z>0.3469448</z>
            </vertex>
            <boundaryPoint>
                <vertex>
                    <x>0.0292903</x>
                    <y>0.0123436</y>
                    <z>0.0763478</z>
                </vertex>
            </boundaryPoint>
        </middlePoint>
    </polygonPoint>
</np>
```

5. Továbbfejlesztési lehetőségek

Mivel a barlangászathoz a térképezésen kívül nagyon sok már terület is kapcsolódik, ezért nagyon sok új feladat van még.

Az első ilyen feladat, a mérési adatoknál megadható metaadatok kiegészítése, amely lehetővé tenné, hogy ne csak szövegeket adjunk meg, hanem képeket, vagy akár más jellegű információkat. A kiegészítés lehetővé tenné, hogy a Tóruszt tudományos munkára is használni tudják a geológusok, hidrológusok, biológusok. Az összes ismertebb budapesti barlangban végeznek jelenleg is tudományos méréseket, amelyeknek eredményét így a barlang adott pontjához lehetne kötni.

Nagyon fontosnak tartanám, hogy a Tórusz a jól megszokott projekt szemléletet követve ne csak egy modell megjelenítését tegye lehetővé. Egy projekthez tartozhatna akár több mérési eredmény is, amely mondjuk egy kutatási terület összes barlangját tartalmazná. A barlangok megjelenítése mellett sokat lendítene a látványon, ha különböző objektumokat is a nézethez tudnánk adni. Egy járat omladékos részére köveket helyezhetnénk, vagy egy felmászáshoz egy létrát.

A Tórusz széles körű elterjedéséhez elengedhetetlen olyan modulok írása, amely a legelterjedtebb modellező programok formátumába tudná menteni az adatokat. Így azoknak a barlangászoknak, vagy kutatóknak nem kellene leváltani a már jól megszokott programjaikat. A Tórusz soha nem fog olyan sok és részletes funkciót biztosítani a megjelenítés módosítására, mint mondjuk a Maya vagy a 3D Studio Max, viszont egy exportáló modullal mindazon módosításokat, amelyek nem a Tórusz feladatai megoldhatnánk 3D Studio Max-ban.

Nagyon hasznosnak tartanám, ha a Tórusz eredményeit egy központi helyen tarthatnánk, amelyet bárki elérhetne egy webes böngészővel. A Tórusz telepítése nélkül nézhetnénk meg a legismertebb hazai barlangokról készített háromdimenziós modelleket.

6. Összefoglaló

Több hónapos munka után elmondhatom, hogy barlangok háromdimenziós felmérésével és megjelenítésével foglalkozó területen az országban elsőként használható eredményeket értem el.

Sikerült olyan alapjaiban új mérési módszereket kidolgoznom, amelyek lehetővé teszik, hogy a barlangokról pontos és élethű modellt kapjunk. Az első mérési módszer, a $4P^2$ leginkább a már meglévő térképek háromdimenziós kiegészítésére alkalmas. A következő módszer az NP^2, amely tekinthető az $4P^2$ kiterjesztésének, egy olyan szelvényekből építkező technika, amely a barlang járatainak és termeinek gyors és homogén felmérését teszi lehetővé. Az utolsó módszer, az SPCR egy olyan adaptív felület rekonstrukciós algoritmus, amely semmilyen megkötést nem tesz a felmért pontok számára vagy struktúrájára vonatkozóan. Lehetővé teszi, hogy egy ritka és nem egyenletes pontfelhőből olyan reguláris és konform felületi modellt generáljunk, amely nem tartalmaz lyukakat, de tartalmazza az eredeti mérés összes pontját.

A mérési módszerek feldolgozásához és a generált felületi modell megjelenítéséhez két szoftvert fejlesztettem ki, amelyek a Tórusz és a Toroid nevet kapták. A Tórusz alapvetően az otthoni számítógépekre szánt alkalmazás, míg a Toroid ennek barlangban használható verziója. A Toroid egy olyan Androidos alkalmazás, amelyet a barlangban tablet PC-n vagy okostelefonon használhatunk.

Az új mérési módszereket több hazai barlangban is kipróbáltam (Pálvölgyi-Mátyás-hegyi barlangrendszer, Szemlő-hegyi barlang, Meta-barlang). A mérések igazolták, hogy a kidolgozott algoritmusok és programok a gyakorlatban is használhatóak.

Az új eredményeket felhasználva olyan emberek is betekintést nyerhetnek a barlangok világába, akik soha nem jutnak le egyik hazai barlangba sem. Remélhetőleg így az emberek többsége ráébred arra, hogy a barlangok micsoda kincset jelentenek, amelyeket óvni és tisztelni kell.

7. Abstract

After a hard research & develop activity it could be stated as the pioneer of this interesting field I reached usable and applicable results in connection with the 3D measurement and visualization of the caves.

I successfully developed basically new measurement methods which give accurate and realistic models. The first method, called $4P^2$ is an easy way of extending previous cave maps with 3 dimensional visualization. The following method, the NP^2, which is the extension of $4P^2$, is a segment based technique which makes it possible to measure the passages of the caves quickly and homogeneous. The last method, SPCR is an adaptive surface reconstruction algorithm, which does not limit the number and the structure of the measured points. It allows creating regular and conformal surface model from a sparse and non-uniform point cloud which does not contain any holes and all the measured points are represented in it.

For the processing of the measured data and for the visualization of the generated surface model I developed two softwares named Torus and Toroid. Torus is developed for the desktop computers while Toroid is the version of Torus, which can be used in caves. Toroid is an Android application which permits of the usage of an Android based tablet or smartphone in caves.

I got the opportunity to test the new methods in Hungarian caves (Pálvölgyi-Mátyás-hegyi cave system, Szemlő-hegyi cave, Meta-cave). It was proved after the tests that the new 3D methods developed by me are applicable in real environments.

As a result of my work people who haven' t got the opportunity to visit any caves, get the possibility to view the beautiful and unknown world of the caves. If we publish these new methods and softwares to everybody the human kind can realize that the caves are the nature's beautiful pearls which must be respected and protected.

8. Irodalomjegyzék

[1] Pierre Alliez – Computational Geometry Algorithm

[2] Frederic Cazals, Joachim Giesen – Conformal alpha-shapes

[3] Frederic Cazals, Joachim Giesen, Mark Pauly, Afra Zomorodian – The conformal alpha shape filtration

[4] Kaspar Fisher – Introduction to alpha shapes

[5] Fausto Bernardini, Joshua Mittleman, Gabriel Taubin – The Ball-pivoting Algorithm for Surface Reconstruction

[6] Michal Zamek – Regular Triangulation in 3D and Its Applications

[7] Nataraj Akkiraju, Herbert Edelsbrunner, Michael Facello, Ernst P. Mücke – Alpha –shapes: Definitions and Software

[8] Herbert Edelsbrunner – Delaunay and regular triangulation

[9] Herbert Edelsbrunner, D.G. Kirkpatrick, R. Seidel – On the shape of a set of points in the plane

[10] Herbert Edelsbrunner – The union of balls and its dual shape

[11] Celikik Marjan – Alpha –shapes

[12] Michael Kazhdan, Matthew Bolithot, Hugues Hoppen – Poisson surface reconstruction

[13] Mark de Berg, Otfried Cheong, Marc van Kreveld – Computational geometry

[14] Jean-Daniel Boissonnat, Mariette Yvinec – Algorithmic geometry

[15] Kurt Mehlhorn – Data structures and efficient algorithms 3

[16] Herbert Edelsbrunner – Algorithms in combinatorial geometry

[17] Monique Teillaud – Computational Geometry Algorithms Library

[18] Julian Smart, Kevin Hock, Stefan Csomort – Cross-platform GUI programming with wxWidgets

[19] Eero Simoncelli – Least squares optimization

9. Függelék

9.1 A barlangászatról röviden

A barlangászat egyidős az emberrel, csak akkor még szállásként, menedékként, kultikus és vallási helyként használták a barlangokat őseink. A bronzkortól már csak alkalmi szálláshelyként használták a barlangokat, s a kisebb üregek elé kunyhókat emeltek. A középkorban háborúk esetén búvóhelyként jártak a barlangokban, illetve mindenféle legendákkal és misztikus sárkányokkal népesítették be őket. A barlangok tudományos megismerése és feltárása a 17. században, a kötelekkel és technikai eszközökkel való felfedezésük a 19. század utolsó harmadában kezdődött. A 20. század elején már turisztikai célból is lejártak emberek a barlangokba, majd a század második felétől kezdődően a rohamos technikai fejlődés egyre jobb eszközöket adott a barlangászok kezébe. Világszerte nagyszámú közösség és klub alakul a barlangok kutatása és bejárása céljára.

A barlangászat két fő területre oszlik fel, ezek a barlangkutatás és a barlangtúrázás. E kettő nem különül el élesen egymástól.

- A barlangkutatás során felszíni karsztjelenségek alapján keresnek és találnak egy eddig ismeretlen barlangot, vagy már egy ismert barlang további járatait próbálják a föld alatt felkutatni a szpeleológia tudományát használva. Ezekhez kitartás, akár évekig tartó munka szükséges.
- A barlangtúrázás során már egy ismert barlangot járnak be térkép segítségével, a barlang szépségére, képződményeire figyelve, és megküzdenek a nehézségekkel, földalatti tavak, patakok, mély aknák vagy vízesések képében.

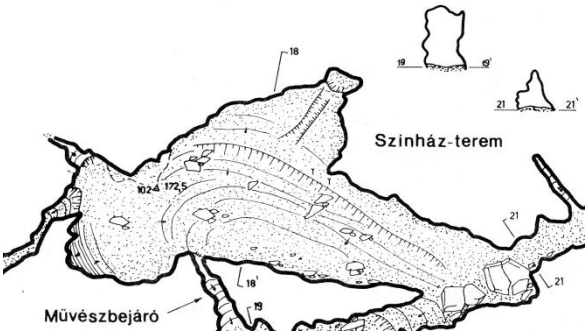
Ezek a lent tartózkodások akár több napig is eltarthatnak, ami azt jelenti, hogy nagy mennyiségű felszerelést, ételt-italt, tartalék ruhát, akkumulátort kell leszállítani a föld alá. A barlangban alkalmi pihenő- és alvóhelyeket kell létre hozni, ezeket „bivak”-nak hívják.

9.2 Térképek és használhatóságuk

A barlangokról készülő dokumentáció egyik legalapvetőbb eleme a megbízható térkép, mely az üregrendszer térbeli koordinációjáról és jellegéről jól áttekinthető

információt nyújt. A kutatási eredmények dokumentálása mellett a térképek alapul szolgálnak a feltáró munkák, expedíciók tervezéséhez, különböző vizsgálatokhoz. adatgyűjtésekhez, műszaki tervezésekhez valamint a barlangot bemutató előadásokhoz, kiadványokhoz. A felmérések, ill. a térképek különböző pontossággal, részletességgel készülnek a kívánt céltól, és a barlang jelentőségétől, jellegétől függően.

A hazai barlangok térképezése mintegy kétszáz éves múltra tekint vissza, kezdetben a barlangábrázolások nem műszeres felméréssel készültek, csak művészi kivitelezésű vázlatoknak tekinthetjük őket (1. melléklet). Pontos. műszerekkel történő barlangfelmérések a XIX. sz. elejétől készültek (2. melléklet). Ezek már a barlangok nyomvonalát és hosszmetszetét pontosan ábrázolták, de alaprajzuk még inkább a művészi kivitelezés.. mint a pontosság felé hajlott, emiatt torzított volt. Az alaprajzi nézet mérethelyes megjelenése csak a XX. században kezdődött meg, de ekkor is a pontos geodéziai méréseket követően a barlang alaprajzát emlékezetből rajzolták meg, és a járatok formakincsét egyáltalán nem vagy nagyon elnagyoltan ábrázolták. A XX. század második felében jelentek meg a barlangok térbeli jellegét is visszaadó, mérethelyes alaprajzi hosszmetszetet és keresztszelvényeket is tartalmazó térképek, melyek már ábrázolták a barlang formakincsét is. A XX. század végén az Aggteleki-karszt barlangjainak Világörökség listára kerülésével, valamint az Országos Barlangnyilvántartás törvényben (1996 évi LIII. tv. 49. §.) előírt kötelezettségeinek megfogalmazásával felmerült az igény a korábbi barlangtérképeknél minőségileg újabb, sokkal szigorúbb feltételeknek megfelelő. nagylátképű (1:100) barlangtérképek felvételére.



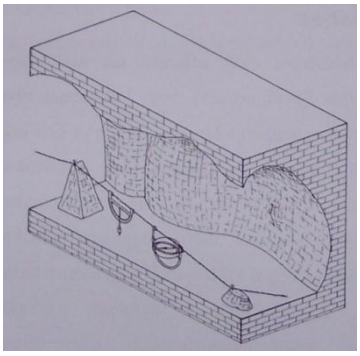
70. ábra Mátyáshegyi-barlang térképrészlete

Mutatnék egy térkép részletet a 70. ábrán a kedves Olvasónak egy budapesti barlang terméről. A barlangászok ilyen kétdimenziós térképeket használnak tájékozódásra, amely a legtöbb esetben elegendő. De képzeljünk el egy olyan barlangot,

amely többszintes és a különböző színitek járatai fedik egymást. Sajnos a kétdimenziós térképen ilyenkor alig tudjuk használni, hiszen a fedések miatt kevés információt olvashatunk le.

A következő fejezetben arról lesz szó, hogy a térképeket hogy készítik el a szakemberek és előtte hogyan mérik fel a barlangot különböző módszerekkel.

9.3 A sokszögvonal felvétele huzagolásos módszerrel



71. ábra Huzagolásos technika

Huzagolásos módszerrel történő mérés esetén a járatok középvonalában poligonzsinórt feszítünk ki, melyek a sokszögvonal oldalait fogják képezni (71. ábra). A sokszögvonal kijelölésénél ügyelnünk kell arra, hogy a hogy a töréspontok között zavartalan összelátás legyen (a kifeszített zsinór ne törjön meg, ne érjen a töréspontokon kívül semmihez), valamint hogy a töréspontok könnyen hozzáférhető helyre kerüljenek. A sokszögvonal így létrejött töréspontjai lesznek a mérési pontok.

9.3.1 Függőzés

Meredek letöréseken, függőleges aknák esetében, valamint ha 30°-nál meredekebb poligonszakaszt szeretnénk kiváltani, függőzést kell beiktatni. A függőzés során a kezdő mérési pontunkból a poligonzsinór végére akasztott függőón segítségével jelöljük ki a következő mérési pontot, mely a kezdőpont alatt lesz pontosan függőlegesen (a kezdőpont függőleges levetítése).

A függőzéssel kijelölt szakaszok irányszöge nem értelmezhető, lejtszöge a mérés irányától függően + vagy -90°. A mérés során ezeken a szakaszokon csak távolságmérést kell elvégezni.

9.3.2 Mérések a sokszögvonalon

A mérés során a szomszédos pontok által meghatározott szakaszok térbeli hosszát, vízszintes síkkal bezárt szögét, valamint a mágneses északi iránytól való eltérését mérjük.

A mérési pontok elhelyezését követően, a pontok között kifeszített poligonzsinór segítségével kezdjük meg az egyes szakaszok felvételét. A mérések során műszereinket a poligonzsinórra akasztjuk fel. A mérés pontosságát jelentősen befolyásolhatja, ha a poligonzsinór nem feszes, hanem belóg, ezért nagyon fontos a megfelelő zsinór alkalmazása, és megfeszítése. A zsinór egyik végén hurkot képzünk, így az az egyik mérési pontra felakasztható, másik végét kézi erővel meghúzva rátekerhetjük a másik mérési pontra (csavarra). Az alkalmazott műszerekről a következő fejezetben lesz szó

9.3.2.1 A távolságmérés és eszközei:

Mérőszalag

A két szomszédos mérési pont közötti távolságot mérőszalaggal kellő megbízhatósággal megmérhetjük. Használhatunk acél és műanyag mérőszalagot, ezek hossza általában 20, 30, vagy 50 méter. A mérőszalag pontatlansága miatt mindig két mérést kell végeznünk.

Lézeres távolságmérő

A technika fejlődésével a geodéziában megjelentek a lézeres távolságmérők, amelyek a felszínes gyors, pontos távolságmérést tesznek lehetővé. Már 1975-ben kísérleteztek hasonló elven működő lézeres távolságmérő műszerrel a barlangtérképezésben, ekkor még csak 20 méteres hatótávolságot és mintegy ± 10 cm-es pontosságot sikerült elérni.

A mai távolságmérők mérési tartományuk típustól függően 0,005 métertől 60-200 méterig terjed, mérési pontosságuk 1,5-3 mm között van. Barlangi körülmények között a mérések sikerességét befolyásolja a páratartalom.

Amennyiben nem követelmény a nagyfokú pontosság, használható a sokszögvonal távolságmérésére is.

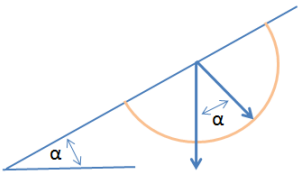
Ultrahangos távolságmérő

Hasonló a lézeres távolságmérőhöz, működési elve: a készülék olyan magas frekvenciájú hanghullámokat (ultrahangot) bocsát ki, amelyek a falról visszaverődnek. A kisugárzás és a visszavert jel beérkezése között eltelt időből a készülék kiszámolja a pontos távolságot. A célpontra való pontos célzást működés alatt kibocsátott lézer irányfény segíti. Mérési tartományuk típustól függően 0,6 métertől 15-20 méterig terjed (felszíni körülmények között), mérési pontosságuk ±2% körüli, tehát jóval kevésbé alkalmasak barlangi mérésekre, mint a lézeres távolságmérők. Elképzelhető, hogy a működés közben kibocsátott ultrahang a barlangban élő denevéreket zavarja. Alkalmazhatósági területe a barlangi mérések során kizárólag kiegészítő mérésekre korlátozódik.

9.3.2.2 A lejtszögmérés és eszközei

A lejtszög a sokszögvonal egyes szakaszain két pont között a poligonszakasz vízszintes síktól való eltérése. Jele: φ, mértékegysége: fok. A lejtszöget többféle műszerrel is mérhetjük, melyekkel különböző pontosságot érhetünk el.

Fokív



72. ábra Lejtszögmérés fokívvel

A fokív egy 15-20 cm sugarú, fokbeosztással ellátott félkör, két végén két kampóval, mellyel a kifeszített poligonzsinórra felakasztható. A műszer fokbeosztása középtől mindkét irányba 90°-ig terjed, osztásköze 0,5° illetve 1/3°. A műszer félkörének középpontjában egy furat található, melyen átvezetett vékony szálon egy függőón található. Méréskor a kifeszített poligonzsinórra felakasztjuk a fokívet, és megvárjuk, amíg a függőón lengése lecsillapodik. Ezt követően leolvassuk azt az értéket, ahol a szál elmetszi a fokskálát. Amennyiben a fokív „0" pontjától a mérés iránya felé metszi a szál a skálát, negatív előjelet kap a leolvasott érték (lejtés), amennyiben a mérés kezdőpontja felé esik a szál a „0" ponttól a skálán, a leolvasott

érték pozitív előjelű (emelkedés). A pontos mérés elengedhetetlen feltétele a poligonzsinór megfelelő megfeszítése.

Az irányszög mérés eszközei

A poligonoldal irányszöge alatt az északi iránnyal bezárt szögét értjük, az óramutató járásával megegyezően, 0°-tól 360°-ig. Jele:δ, mértékegysége: fok.

Az irányszöget többféle műszerrel is megmérhetjük, melyekkel különböző pontosságot érhetünk el. Fontos megemlítenünk, hogy a műszereinkkel csak a mágnese északi iránnyal bezárt szöget mérhetjük, a valós északi irány ettől eltér (a Föld mágneses északi pólusa nem egyezik meg a hosszúsági körök által kimetszett északi sarkkal), ez az eltérés a mágneses deklináció, mely földrajzi helytől és időtől függően változik. Magyarország területére vonatkoztatott mágneses deklináció (D) értéke a földrajzi koordináták másodrendű függvényeként határozható meg:

D['] = 99,04 + 0,00469(j-2730') + 0,2196(l-960') + 0,00027(j-2730')2 + 0,000010(j-2730')(l-960) - 0,00001(l-960')2

A fenti formula 1995,0 epochára vonatkozik. A mágneses deklináció átlagos éves változás normál értéke (d) szintén a pont földrajzi koordinátáiból számítható:

d['év] = 4.41 + 0,00033(j-2730') - 0,00276(l- 960')

ahol D = a mágneses deklináció percekben, j = a pont földrajzi szélessége percekben, l = a pont Földrajzi hosszúsága percekben, d = a mágneses deklináció átlagos éves változása percekben.

Emiatt a barlangban történt méréseinket követően a kapott értékeket a felszínen a mágneses deklináció értékével korrigálni kell. Szintén fontos, hogy a mágneses elven alapuló iránymméréseink során csak mágneses zavartól mentes környezetben kapunk megfelelő értéket. Nem alkalmazhatóak ezek a mérések olyan helyen, ahol a barlangban elektromos vezetékek, vasjárdák, korlátok, létrák vannak beépítve.

Az irányszögmérésnél kapott értékeket mindig korrigálni kell az alábbiak szerint:

$\delta_{valós} = \delta_{mért} - D$

Függőkompasz

A függőkompasz 1633 óta igen elterjedt a bányamérések területén. Két részből áll: egy külső keretből és egy belső iránytűből. A külső keretet mindig a poligonzsinórra helyezzük, a belső iránytűt vízszintes helyzetbe állítjuk, majd leolvassuk az értéket. Fontos, hogy függőlegesen nézzük rá az iránytűre, különben mérési hibát kapunk.

9.3.3 Mérési munkálatok létszámigénye

A fentiekből következik, hogy a barlangi sokszögvonalak mérései munkálatai során minimálisan 2 fő szükséges, amennyiben minden pont csavarral ellátott, amihez a kifeszített poligonzsinórt rögzíteni tudjuk. Ekkor egy fő a műszerek leolvasását végzi, a másik fő vezeti a mérési jegyzőkönyvet, és távolságmérésnél a mérőszalag egyik végét pontra illeszti. Kényelmesebb azonban három fővel végezni a mérést, ekkor a jegyzőkönyvvezető tiszta kézzel tud dokumentálni.

9.3.4 A huzagolásos mérési módszer pontossága

A mágneses eszközökkel (függőkompasz, geológus kompasz) végzett felmérések során minden oldal irányszögét az előzőétől függetlenül határozzuk meg (a mágneses északi irányhoz), ezért az egyes oldalak ±μ*s irányhibája az előző oldalak irányhibájától független (ahol a ±μ az irányszögmérés középhibája, az s az átlagos poligonoldalhossz). Így az elcsavarodási részhibák (irányhibák) a hiba továbbterjedés szabályai szerint összeadódnak a poligonvonal végpontján: pl. n oldalú poligon esetén az elcsavarodási középhiba a végponton: $\mu_{össz}=\mu_1*s_1+ \mu_2*s_2+\dots \mu_n*s_n$.

A középhibák négyzeteinek összege így:

$$\mu_{össz}=\pm\sqrt{n * \mu^2 * s^2}$$

Ha L a poligonvonal összhossza, akkor:

$$\mu_{össz}=\pm\frac{u^2}{g^2} * L * \sqrt{\frac{1}{n}}$$

Ebből a képletből az következik, hogy ugyanazon hosszúságú poligonvonal esetében a poligonvonal elcsavarodási középhibája annál kisebb, minél rövidebb az oldalak hosszúsága. Emiatt mágneses eszközökkel végzett méréseinknél célszerű a hosszú poligonszakaszok felvételét elkerülni.