Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Javítással visszacsatolt automatikus mesh generálás speciális 3D szkenner pontfelhőjéből

TDK DOLGOZAT

2016

Készítette
Mezei Adrián

Konzulens
Dr. Kovács Tibor

Tartalomjegyzék

Kivonat

A körülöttünk levő világ megfigyelésére, elemzésére és fejlesztésére egyre nagyobb hangsúlyt és egyre több energiát fektetünk. A kétdimenziós képalkotás és képfeldolgozás mindennapjaink természetes része éppúgy, ahogy ezek nyomtatása és többszörösítése.

A közelmúltban megjelentek a háromdimenziós nyomtatók, valamint a tér szkennelésére alkalmas eszközök. A térszkennerek például pontfelhő készítésével tudják letapogatni a kívánt objektumot, melyből – például háromdimenziós nyomtatáshoz – háromszögháló generálására lehet szükség.

Jelen dolgozatban egy speciális lézerszkenner által generált pontfelhőből olyan háromszögháló készül, melyen hatékonyan végezhető hibaelemzés. Ennek segítségével a korábbi szkennelés által – például az árnyékhatás miatt – hiányosan feltérképezett felületdarabok egy újabb beállítással ismét szkennelésre kerülnek, majd az újabb pontfelhőből a hibás elemek javíthatók.

A szkenner beállítása automatikusan kerül meghatározásra a korábbi pontfelhőkből, ezért a szkennelés folyamatában emberi beavatkozásra nincs szükség. A szkennelés pontossága előre megadható különböző paraméterek segítségével. Nagyobb pontosság meghatározásával valószínűleg több szkennelésre lesz szükség, illetve elképzelhetők olyan részek is, melyek az adott hardverrel nem szkennelhetők. Az ilyen részek meghatározására is sor kerül, hiszen ha többszöri – a hibacsoport javításához beállított – szkennelés során sem keletkezik pontosabb pontfelhő, akkor az adott területet nem lehet így szkennelni. Ilyen módon igazolható a folyamat konvergenciája adott pontosság mellett.

A dolgozatban egy zárt mesh automatikus generálása, majd annak elemzése, hibadetektálása és hibacsoportosítása kerül bemutatásra. A kidolgozott folyamat algoritmusai kellően optimális komplexitással rendelkeznek ahhoz, hogy egy átlagos számítógép is néhány másodperc alatt el tudja végezni az említett feladatokat egy több millió ponttal rendelkező ponthalmazon. Emellett az elkészített háromszögháló továbbra is zárt, és a hibafoltok körvonalai folytonosak. Az algoritmusokhoz részletes komplexitás elemzés társul, melyeket mérési eredmények is alátámasztanak.

A további szkennelési beállítások ezekből az adatokból számíthatók szintén automatikusan, majd további algoritmusokkal a korábban meghatározott hibafoltok javíthatók.

Abstract

Automatic mesh generation from point cloud of a special 3D scanner with error correction feedback

In the last few decades we have dedicated a great deal of time to observe, analyse and develop our environment. The two dimensional imaging and image processing is now part of our life, as well as scanning, copying and printing.

A few years ago three dimensional printers and technologies that are able to scan 3D objects appeared, and they are becoming more and more popular. The 3D scanners can mostly generate point clouds from which – for 3D printing, for example – a triangle mesh generation might be required.

In this paper, a triangle mesh is created from a point cloud – generated by a special 3D scanner – on which error detection can be efficiently performed. This allows each deficient part of the previous scan – because of the shadow effect, for example – to be partially scanned again. The settings for each following scan can be calculated from the corresponding deficient section of the surface.

The settings of the scanner are automatically calculated from the previous point cloud so no human interaction is needed during the scanning process. The precision of the scan can be set up in advance by adjusting the provided parameters. A more precise scanning setup will probably need more scanning. Occasionally it is impossible to perform an adequate scanning with the provided hardware, so these cases must be recognized. These can be detected after a few scans during which no useful triangles could be formed. With this extension, it can be proved that the process is convergent, since a surface part can either be enhanced or it will be denoted as an unimprovable section.

In this paper, the automatic generation of a closed mesh, its analysis, error detection and organization is presented. The algorithms of the described process has optimal complexity, so an average computer can also carry out the mentioned tasks on a model consisting of a few million points in a few seconds. Furthermore, the created triangle mesh is still closed, and the borders of the calculated error groups are continuous. In this paper, I provide a detailed complexity analysis and corresponding measurement results that prove the theory and which serve as evidence of the suitability of the process.

Further scanning setups can automatically be calculated from the results of the described algorithms, and methods can be created to correct the error groups.

Bevezetés

A BME Automatizálási és Alkalmazott Informatikai Tanszéken régebben készült egy aktív triangulációs lézerszkenner (1. ábra) (Kovács, 2009, old.: 4-15), mely egy szabadságfokkal rendelkezett. A szkenner egy forgatható tárgyasztalból, egy kamerából és egy lézerből állt. Működése során a lézer egy függőleges vonalban világította meg a tárgyat, melyről a kamera egy másik irányból képet készített. Az így keletkezett képen a lézervonal elhajlása alapján egy vonalkövető algoritmus pontsorozatot generált a vonalból. Egy szkennelés egy rotáció eredményeként állt elő, melyben végül a pontsorozatok alkották a pontfelhőt.

1. ábra: Az egy szabadságfokkal rendelkező 3D szkenner

Ez azonban az évek múltán – a kialakítás nyújtotta korlátozások miatt – számos esetben nem felelt meg a szkennelési pontosságok követelményeinek. A csupán rotációs szabadsági fok kevésnek bizonyult, hiszen számos helyet nem lehetett vele szkennelni. Ez legtöbbször az árnyékhatásból – ilyen értelemben topológiai lyukakból – eredt, mely azt jelenti, hogy a tárgy egy kiálló része kitakarja a lézervonalat a kamera elől. További nehézségeket okoztak a textúrális tulajdonságok, melyeknek egy részén nem látszódott jól a lézervonal. Ezek hatására bizonyos részeket nem lehetett szkennelni.

Ennek továbbfejlesztéseként készült el néhány éve egy immár rotációs és transzlációs transzformációkra is képes szkenner (2. ábra). Ezzel a módosítással a szkennelhető felületek lényegesen bővültek, azonban a modellek megalkotásának komplexitása is megnőtt. Egy szkennelés így nem csupán egyetlen rotációból, hanem több egymást követő, különböző irányokból történő transzlációs és rotációs transzformációk együtteséből áll.

A szkenner tervezésekor mindig a pontosság volt az elsődleges szempont, mely most azt is jeleni, hogy ha egy szkennelésből nem lehet kellő pontosságot elérni, akkor a

szükséges részeket újból kell szkennelni egy másik beállítással. Ez mind automatizálási, mind algoritmikai szempontból bonyolítja a feladatokat. A szkenneléseket elemezni kell, hogy azok hol hiányosak vagy pontatlanok, a hibás részeket újra kell szkennelni, az elkészült szkenneléseket össze kell illeszteni és háromszögelni is kell.

2. ábra: A 3D szkenner, melyen látható a kamera, a lézer, a forgótányér és egy – különböző adatok megjelenítésére képes – LCD kijelző, a kontroller pedig a tálca alatt helyezkedik el

Az én feladatom egy olyan folyamat kidolgozása, mely segítségével a szkennelés során automatikusan is generálható egy háromszögháló, valamint több szkennelési folyamat összeilleszthető és a hibás részek újbóli szkenneléséhez a szkenner automatikusan konfigurálható. A folyamatnak olyannak kell lennie, hogy a javító szkennelések beillesztése után egy zárt háromszögháló keletkezzen, mely később például 3D nyomtatóval is kinyomtatható lesz.

3D szkenner

A szkenner ebben az esetben egy olyan eszköz ((Kovács, 2009, old.: 4-15) továbbfejlesztése), mely két szabadságfokkal rendelkezik. Egy transzlációra képes a Z tengely mentén, illetve teljes rotációra az Y tengely körül. Az algoritmusokat tehát ezek figyelembevételével kell kialakítani. A két transzformációhoz két léptetőmotor tartozik, melyet egy – a forgótálca alatt elhelyezkedő kontroller – irányít. A tálcán lévő objektumot egy – a pozitív X tengely mentén elhelyezkedő – lézer világítja meg egy függőleges vonalban. Az így keletkezett képet a kamera rögzíti, majd továbbítja a számítógépnek. A vonalkövető algoritmus (Kovács, 2009, old.: 95-109) a lézer vonalából pontsorozatot készít. Néhány szkennelés után, ezekből a pontsorozatokból egy pontfelhő keletkezik, melyet egy szoftver segítségével meg lehet jeleníteni. Ezt követően jutnak el a vonalak a mesh generátorhoz.

A háromszögháló generálása során fel kell készülni többszörös szkennelési pontfelhők összeillesztésére, melyekből egyetlen zárt háromszöghálót kell készíteni. A szkennelés teljes automatizálása a cél, így a további szkennelések beállítása, vagyis a transzlációs és rotációs adatok automatikus meghatározása is feladat. A szkennelési folyamatot nem hátráltathatja az algoritmusok lassú futása, tehát az algoritmusok komplexitásának olyannak kell lennie, hogy az egész folyamat néhány másodperc alatt elvégezhető legyen.

Triangulációs eljárások

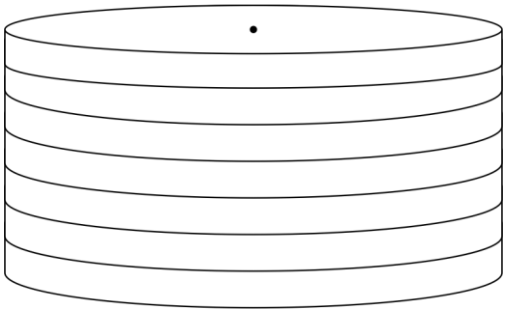
Számos felületrekonstrukciós módszer létezik, melyek előre adott pontfelhőből készítenek háromszöghálót. A legelterjedtebbek közé sorolható például a Poisson felületrekonstrukció (Kazhdan, Bolitho, & Hoppe, 2006). Görbületeket figyelembevevő eljárás, mely a felületet különböző módokon próbálja felosztani, majd az egyes felosztások között szavazással választják ki a legmegfelelőbbet (Schmidt & Simari, 2012), illetve hasonlóan (Zagorchev & Goshtasby, 2012).

A számítógépes geometriában leggyakrabban alkalmazott rekonstrukciós algoritmusok nagy része a Voronoi diagramra vagy a duálisára, a Delaunay háromszögelésre alapuló módszereket használnak. Ilyen algoritmusok részletes kidolgozásai például (Bernardini, Mittleman, Rushmeier, Silva, & Taubin, 1999), (Dey, 2006). Szintén ebben a témakörben elterjedt módszer az „Alpha shapes” módszer (Edelsbrunner & Mücke, 1994).

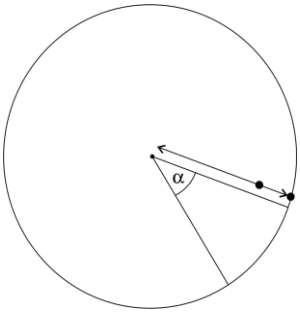
Implicit felületek approximációs módszereit is érdemes ismertetni, mindazonáltal a számítógépes grafikában a megjelenítéshez végül tesszellált felületekre van szükség.

Ilyen módszerek például (Blinn, 1982), (Jia, és mtsai., 2010), (Kazhdan, Bolitho, & Hoppe, 2006). Ezek mellett vannak paraméteres felületillesztést kialakító módszerek is (Goshtasby, 2004) és (Jackowski, Satter, & Goshtasby, 2003).

Fontos azonban, hogy végső soron olyan módszerre lesz szükség, mellyel egy zárt háromszögháló készíthető, tehát a mesh zártsága biztosan garantálható. További elvárás, hogy a generált hálón a navigálás és a csúcsokhoz tartozó háromszögek meghatározhatósága gyors és hatékony legyen. Egy ilyen megoldáshoz hasonlót alkalmaznak szintén 3D szkennerhez (Khan, Okuda, & Takahushi, 2004), melyben henger leképzést végeznek a szkennelt pontokból.

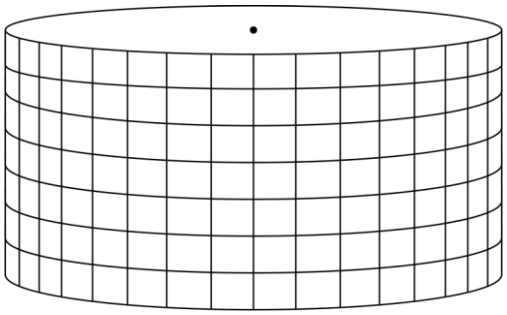


3. ábra: Az említett módszerben (Khan, Okuda, & Takahushi, 2004) a henger felosztása síkokra történik, hogy az egyes szkennelt pontokat ezeken lehessen ábrázolni

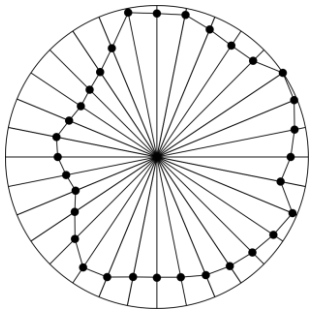


4. ábra: Az említett módszerben (Khan, Okuda, & Takahushi, 2004) a pont mozgatása a síkon, azonban az α szög nagysága tetszőleges (a szkennelt pont helyétől függ)

Itt azonban egyszeri szkennelési adatokat dolgoztak föl, illetve a szkennernek csak egy fix tengely körüli forgatási szabadságfoka volt. Ennek a módszernek a finomításával elérhetőek a kívánt kritériumok. Az említett módszerhez (Khan, Okuda, & Takahushi, 2004) képest a jelenlegi szkennerből (Kovács, 2009, old.: 4-15) származó pontfelhő adatok estén az is meghatározott, hogy egy pont melyik szkennelési oszlopban van. Ennek segítségével tehát a henger vízszintes felosztásának módszere kiegészíthető – a forgástengely körüli – függőleges felosztással.



5. ábra: Egyenletes felosztás síkokra, egyenletes függőleges felosztással kibővítve



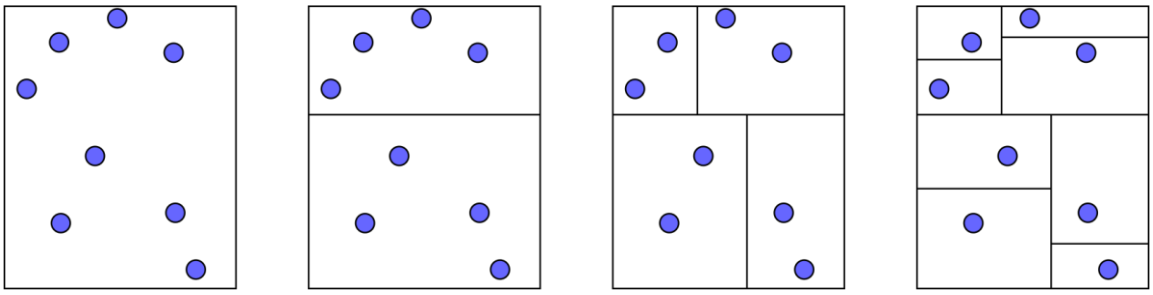
6. ábra: A szkennelési pontok mozgatása az egyenletesen felosztott palást mentén sugár irányba (a felosztott henger egy tetszőleges síkján)

A felosztás kezdetben teljesen egyenletes, mely a pontok mozgatása során bizonyos mértékben megváltozik. A szkennelt pontok sűrűsége azonban előre tudható, melyek függvényében a felosztások – mind a függőleges, mind a rotációs – akár dinamikusan is változtathatóak[1]. Ekkorra azonban ezek a módosítások a teljes struktúrára hatással lesznek.

Ennek a módszernek továbbá előnye lesz, hogy az egyes csúcsoknak konkrét indexe van. Ez ugyanígy igaz a háromszögekre is, hiszen ez a felosztás a palástón automatikusan egy négyzetháló, melyeket két-két háromszögből össze lehet rakni. A felső és alsó lapokat pedig úgy lehet háromszögelni, hogy a felső és alsó síkok pontjait összekötjük a hozzájuk tartozó felső vagy alsó középponttal.

Adatstruktúra lehetőségek

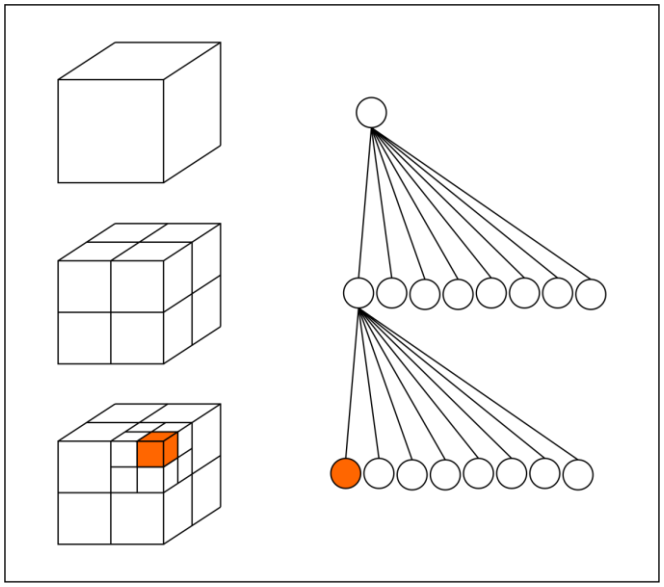
A háromdimenziós struktúrák tárolásának számos lehetséges megoldása van. A két legfontosabb része a csúcspontok és a háromszögek – tehát, hogy mely csúcsok alkotnak háromszöget – tárolása. Számítógépes grafikában gyakran használt struktúra például a k-d fa, mely hatékony megoldást ad térbeli keresésre, egy bizonyos csúcs körüli csúcsok keresésére (Friedman, Bentley, & Finkel, 1977) (Muja & Lowe, 2009).



7. ábra: A k-d fa kialakításának lépései

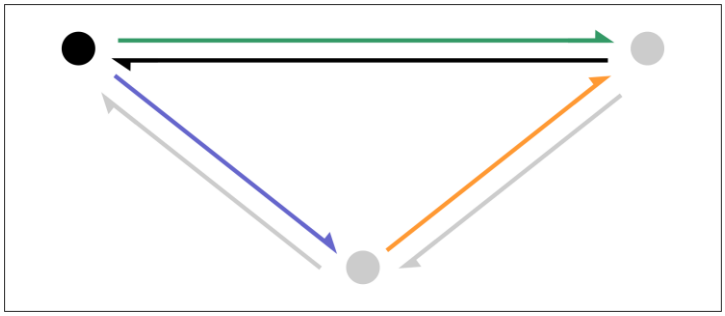
Egy másik megoldás az oktális fa, mely segítségével egy hierarchikus struktúra építhető a pontfelhőből. Az oktális fában a tér nyolc diszjunkt – a teret teljesen lefedő – kockára van bontva, majd ezek rekurzívan mindig további nyolc-nyolc kisebb kockára, egészen addig, ameddig a felosztás finomsága ezt megköveteli. Pontfelhők kezelésére teljes függvénykönyvtárak is készültek, melyek hatékonyan implementálnak különböző pontfelhő elemző módszereket (Rusu & Cousins, 2011).

[1] Például meg lehet azt tenni, hogy a függőleges felosztás középen sűrűbb, a szélekhez közeledve pedig egyre ritkább, de az ilyen módosítások a henger teljes kerületére fognak vonatkozni.



8. ábra: Az oktális fa térfelosztása és hierarchikus tárolásának struktúrája

Szintén gyakran használt adatstruktúra az úgy nevezett „half-edge” struktúra, melynek technikájára a neve is utal, hiszen fél élek képzik a tárolás alapelemeit. Itt minden pontpár által képzett él egyszer vagy kétszer szerepel, annak függvényében, hogy mindék oldalán van-e háromszög. Egy half-edge-hez tartozik egy csúcs, a háromszögének körüljárási irányában következő half-edge, a half-edge-ek által meghatározott háromszög (vagy sokszög) valamint a másik half-edge, mely ugyanezek között a pontok között megy csak a másik irányba (ez a half-edge az él másik oldalán levő háromszöghöz tartozik) (Botsch, Steinberg, Bischoff, & Kobbelt, 2002).



9. ábra: Fekete színnel egy half-edge, a hozzá tartozó csúcs szintén feketével, kékkel a következő half-edge, zölddel a hozzá tartozó ellentétes irányú half-edge, a lapot pedig a három (fekete, kék, sárga) half-edge valamelyike határozza meg

Szkenner feldolgozási struktúrája

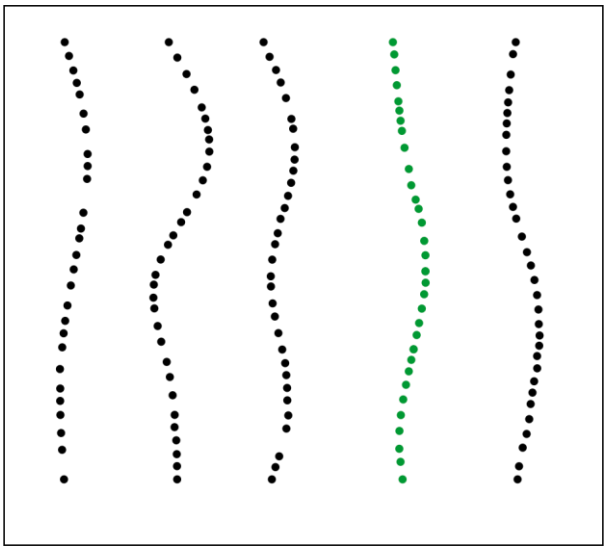
A 3D szkenner már egy definiált adatstruktúrával dolgozik. Ennek legkisebb építőeleme egy csúcspont (továbbiakban `Vertex`), mely egyetlen csúcspontról tartalmaz információkat. Tartalmazza például a csúcs pozíciójának x, y és z koordinátáját,

tartalmazza a csúcs színét r, g, b[1] értékekkel, továbbá később egyéb – a csúcspontra vonatkozó – adatokkal bővíthető. Ilyenek lehetnek például a csúcspont normálvektorai, textúra koordináták valamint a pontfelhők elemzéséből számított különböző statisztikai információk.

A következő építőelem a struktúrában egy vonal (továbbiakban `Stripe`), mely a vonalkövető algoritmus által elkészített sorrendezett Vertex-ek sokasága.

Az egymást sorrendben követő Stripe-ok egy vonalhalmazt állítanak elő (továbbiakban `Batch`). Egy Batch tartalmazza azon Stripe-okat, melyek egyetlen szkennelési folyamatból keletkeztek.

A legbővebb struktúra pedig a felhő (továbbiakban `Cloud`), mely az egymás utáni szkennelésekből keletkezett Batch-eket tartalmazza.



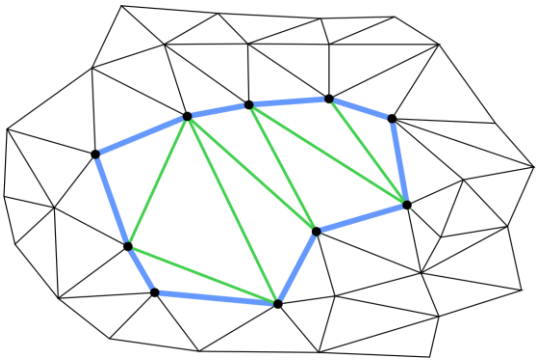
10. ábra: A szkenner adatainak struktúrája, ahol a pontok a Vertex-ek, az egymást követő Vertex-ek alkotják a Stripe-okat (például a zöld Vertex-ek), az öt Stripe pedig alkotja a Batch-et, és jelen esetben ez az egyetlen batch alkotja a Cloud-ot

Javítási módszerek

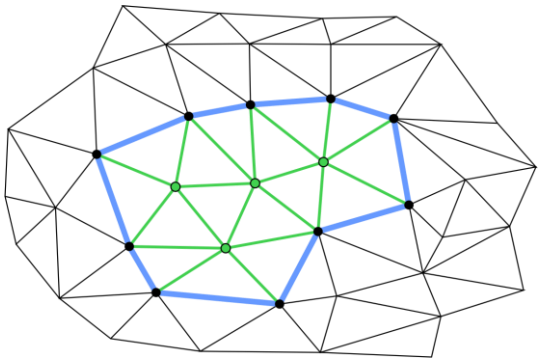
Az egyes szkennelések utáni zárt háromszöghálóban egy ideig maradni fognak hibás részek, melyek javításra szorulnak. Ezek a hibák különböző okokból adódhatnak, például a kamera nem minden beállításból látja a felület minden részét, vagy a textúra alkalmatlan lézeres szkenneléshez. A háromszögháló hibáit – és azok javítási módszereit – különböző módon lehet kategorizálni (Attene, Campen, & Kobbelt, Polygon mesh repairing: An application perspective, 2013), melyek közül most leginkább a lyukjavítás lesz az érdekes. Ilyen lyukak javítására alapvetően kétféle módszer van. Egyik esetben

[1] Ebben a pillanatban nincsenek tényleges textúrális adatok (később azonban lehetnek). Jelenleg különböző skalárhibák hőtérképes vizualizációjához tartalmaznak színkódokat (pl. 21. ábra).

új pontok hozzáadásával (Zhao, Gao, & Lin, 2007), a másik esetben pedig a meglévő pontok megfelelő összekötésével (Barequet & Sharir, 1995) kerül befoltozásra a hiányzó rész.



11. ábra: Új pont nélküli befoltozás



12. ábra: Befoltozás új pontok hozzáadásával

A szkenner esetében ez javítási módszertől függ, azonban ha nem sikerül egy részt szkennelni, akkor végül új pontokkal kiegészítve[1] érdemes megoldani a feladatot (Zhao, Gao, & Lin, 2007). A cél azonban az, hogy további szkennelések által be legyenek foltozva a hiányos részek. Ehhez arra lesz szükség, hogy az újabb szkennelésből be lehessen illeszteni pontokat a hibás foltokba.

Végül a javító foltokat össze kell kötni a korábbi helyes háromszögekkel, mely megvalósítható ez olyan módszernek a minimális módosításával, ami újabb pontok nélkül foltozza be egy háromszögháló lyukas részeit (Barequet & Sharir, 1995), vagy annak egy továbbfejlesztésével (Attene & Falcidieno, ReMESH: An Interactive Environment to Edit and Repair Triangle Meshes, 2006).

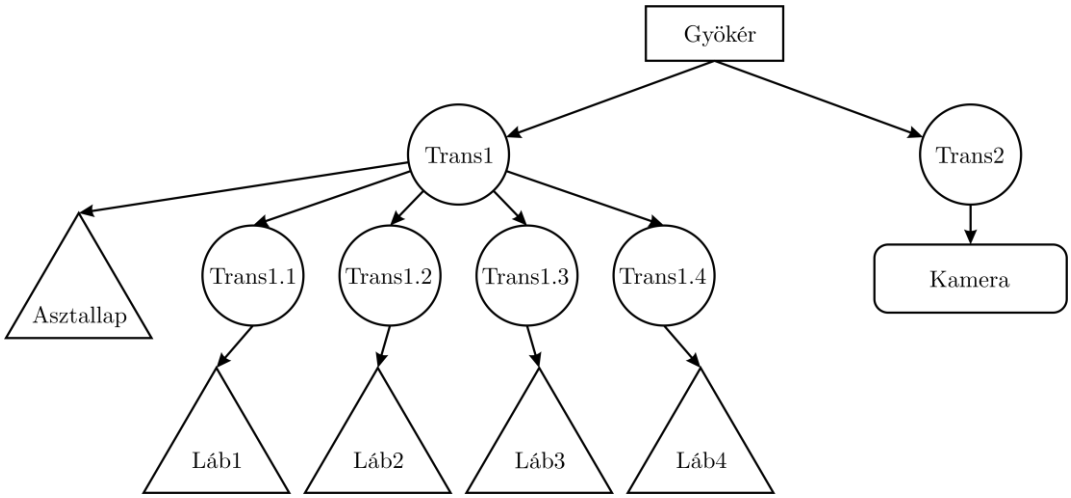
Megjelenítési technika

A virtuális világ kezeléséhez szükség van egy hierarchikus adatszerkezetre, melyben az objektumok egyes részeit és a rájuk vonatkozó tulajdonságokat (geometria, transzformációk, anyagjellemzők, hibainformációk, viselkedési minták) lehet tárolni. Erre a feladatra a *színtérgráf* (Szirmay-Kalos, Antal, & Csonka, 2003, old.: 129-138) egy megfelelő megoldás, melyben a megjelenítendő objektumokon kívül a különböző egyéb – például fényviszonyok – is egy alá- és fölérendeltségi viszonyban vannak. Az így kapott gráf irányított és körmentes, más néven DAG.

Ennek egy egyszerű példáját a 13. ábra mutatja. Itt a gráf egy asztalt és egy kamerát tartalmaz, melyeknek külön egy saját transzformációjuk van (az ábrán Trans1 és Trans2), mely a világban való elhelyezkedésüket határozza meg. Az asztal esetében az

[1] Ha egy részt nem lehet szkennelni, akkor a meglévő pontokból lehet interpolálni néhány új pontot a hiányos részre.

asztallap elhelyezkedése meghatározott (az ábrán a Trans1 által), a lábakat pedig ehhez képes négy különböző transzformáció fogja megadni (az ábrán Trans1.1, Trans1.2, Trans1.3 és Trans1.4). Az egyes komponensek világban való elhelyezkedését tehát a színtérgráf gyökerétől a komponensig vezető út mentén meglátogatott transzformációk szorzata határozza meg.



13. ábra: Színtérgráf egyszerű példája

Tervezés és heurisztikák áttekintése

Tervezési szempontok

Olyan módszert kell kidolgozni, melynek segítségével több szkennelési pontfelhőt is össze lehet illeszteni. Fontos figyelembe venni, hogy a szkennernek csak két szabadsági foka van, így bizonyos alakzatoknak bizonyos területeit képtelenség vele szkennelni.

Továbbá szem előtt kell tartani, hogy az egyes szkennelések időigényesek lehetnek, hasonlóképpen a szkennelt pontfelhők feldolgozása, elemzése illetve javítása is. Ennek megfelelően minél automatizáltabb szkennelési mechanizmust szükséges kidolgozni, mely akár képes az objektumok teljesen automatikus szkennelésére.

Az egyes szkennelések esetén két – alapvetően különböző – mechanizmust lehet megkülönböztetni. Egyik esetben a cél a minél pontosabb modell elkészítése, ahol a szkennelések száma és a ráfordított idő és erőforrás kevésbé érdekes, mint a pontosság. A másik esetben a cél az, hogy minél kevesebb szkenneléssel, minél rövidebb idő alatt készüljön el a modell. Az első esetben tehát a pontosság, a második esetben pedig a sebesség az elsődleges. A megtervezendő mechanizmus segítségével minkét módszer megvalósítható. Azt, hogy a sebesség vagy a pontosság az elsődleges, azt elsősorban a javítási folyamat technikája fogja befolyásolni.

Szkennelés közbeni mesh

Szükség van egy olyan háromszögelésre is, mely a szkennelés közben automatikusan – például minden szkennelési vonal beérkezésekor – megjeleníti a szkennelt pontokból készített háromszöghálót (*szkennelés közbeni mesh*). Ez egy hasznos visszacsatolás a szkennelés folyamán, mely akár azonnal terminálható észlelt problémák esetén. Különböző információkat lehet így vizualizálni, mely például hibás szkennerbeállításokra is felhívhatja a figyelmet. A csúcspont és strukturális adatok tárolására a szkenner Vertex, Stripe, Cloud struktúráját megfelelően lehet használni. A háromszögelési adatokat tárolása – az algoritmikailag komplex műveletek hiányában – egy listában is megfelelő.

Hengeres mesh

Az irodalomkutatás során legmegfelelőbbnek talált kiindulási pont, miszerint egy szabályosan felosztott hengerből indul a mesh kialakítása azt is jelenti, hogy a szkennelt pontfelhő újra mintavételezésre kerül (*hengeres mesh*). A tároláshoz így nem lesz

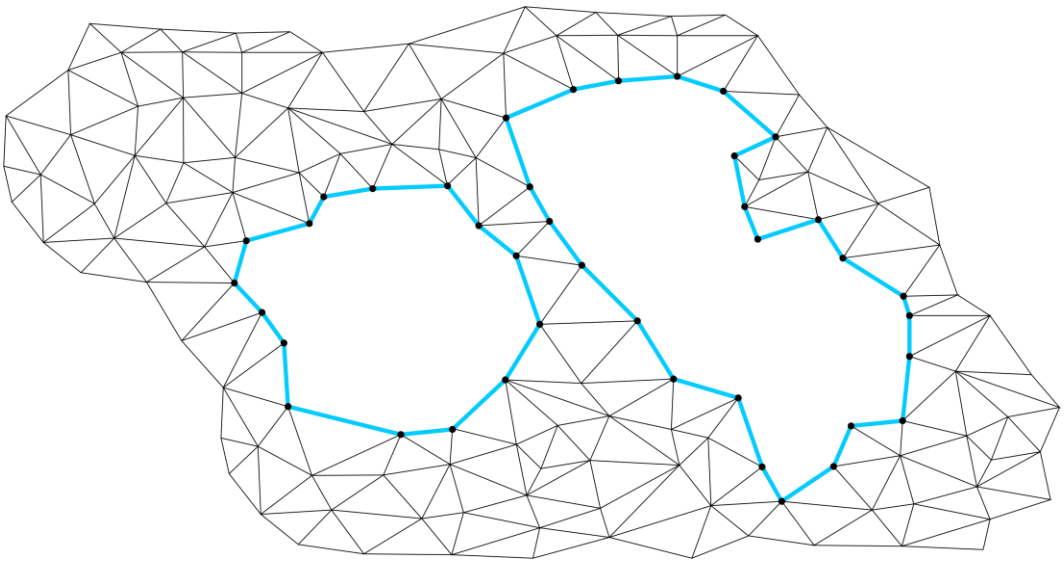
szükség speciális struktúrákra, elég a csúcspontok egy listáját tárolni azzal a kiegészítéssel, hogy tárolják a saját indexüket is, hiszen ez alapján lehet tudni, hogy hol helyezkednek el.

Heurisztikák a szkennelés közbeni mesh generáláshoz

A szkennelés Stripe-ok sorozataként történik, így tehát minden egyes beérkező új Stripe esetén ki lehet generálni az új háromszögeket. Ezek valós idejű visszacsatolással segíthetik a felhasználót. A Stripe-okat tehát „össze kell varrni". Ilyen megoldásokra vannak példák (Barequet & Sharir, 1995), azonban a szkenner adottságait kihasználva egyszerűsíthetőek a módszerek. A két Stripe átlagos meredekségéből kiindulva, mindig olyan élű háromszögeket kell bevenni, amelyeknek a számított Stripe-ok meredekségére leginkább merőleges a két Stripe között menő éle.

Heurisztikák a hengeres mesh generáláshoz

A hengeres mesh generálása során a korábban ismertetett technika (Khan, Okuda, & Takahushi, 2004) kerül továbbfejlesztésre. Az egyenletesen felosztott henger a palástja mentén az első 360°-os rotációs szkenneléskor kerül mintavételezésre. Az így kapott mesh egy bizonyos helyeken helyes, más helyeken hibás zárt felületet biztosít, hiszen a szkenner nem biztos, hogy minden helyről elegendő helyes adatot kapott. Az így kapott felületet tehát javítani kell. Ehhez először a hibás háromszögeket kell azonosítani. Ezt követően a hibás háromszögeket csoportosítani kell, hogy hibás háromszögek helyett hibacsoportok keletkezzenek. Ezek a hibacsoportok javíthatóak egy hozzájuk igazított szkenneléssel, melyekből a szükséges csúcspontokkal a korábbi hálóba beilleszthetőek.



14. ábra: Hengeres mesh generálása és a hibaperemek (az ábrán kékkel) meghatározás után az elvárt eredmény

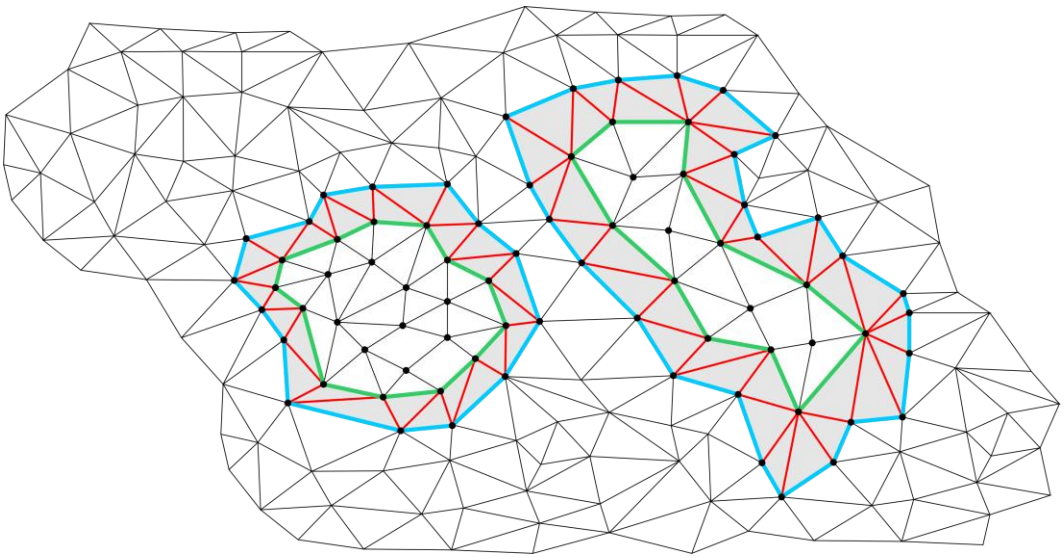
Heurisztika a javításhoz

A javítási és beillesztési mechanizmus részletes kidolgozása nem része ennek a dolgozatnak, azonban kidolgozandó megvalósítási ötletet tartalmaz. Ennek a lépésnek a megvalósításakor rendelkezésre áll egy megfelelőnek ítélt mesh, illetve egy hibaperem lista, mely az egymást nem metsző, diszjunkt hibaperemeket tartalmazza. Az egyes hibaperemek, pontok rendezett sorozataként állnak elő, mely sorozat első eleme szomszédos az utolsóval, illetve minden egymást követő pont egy létező él a háromszögelésben.

Az ilyen hibaperemekhez egyesével történik meg egy szkennerbeállítás, mely alapján egy újabb mesh keletkezik. Ezen először egy hibadetektálást kell végezni, hogy csak azok a háromszögek álljanak rendelkezésre, melyek megfelelnek a pontossági feltételeknek.

Ezt követően az adott hibaperem javításra kerül. A perem mentén végezhető javítás, mellyel az adott hibás terület mérete csökkenthető, vagy több kisebb részre osztható. Fontos, hogy az eredeti hibafoltból is meg kell hagyni a maradék hibás terület befoltozásához szükséges részeket. Ezzel biztosítható a mesh zártsága.

A folyamat konvergenciája is igazolható, hiszen ebben az esetben a javítás sikeressége esetén a hibás terület csökken, azonban ha a javítás nem lehetséges – mert például a hibadetektálási folyamat túl kevés helyes háromszöget hagyott meg a javító szkennelésekből – akkor a mesh az adott hardverrel nem javítható tovább.



15. ábra: Heurisztikus javítás után elvárt eredmény, a hibaperem kékkel, a javítási perem zölddel, a javítási perem és a hibaperem összevarrása szürke háttérrel, a varrás élei pirossal

Szkennelés közbeni mesh generálás

Tudjuk, hogy a bemeneti Vertex tömb Stripe folytonos, valamint a Stripe-ok Vertex folytonosak. Valós időben megoldható lehet, hogy az egymást követő Stripe-ok között generálunk háromszögeket. Ezzel a Batch-ből való mesh generálása visszavezethető a két Stripe közötti mesh generálásra. A feladat tehát két Stripe között valós időben legenerálni a háromszögeket.

Algoritmus lépései

Az alábbi lépésekben meghatározásra kerülnek azok a csúcshármasok, melyek az egyes háromszögeket alkotják. Az kialakított háromszög elfogadhatóságáról a hibakezelő algoritmus (Hibakezelés módja) dönt.

1. Első háromszög meghatározása

   Amennyiben mindkét Stripe tartalmaz legalább egy Vertex-et és legalább az egyik tartalmaz két Vertex-et, akkor mindkét Stripe-ból az első Vertex-et kell kiválasztani. Ellenkező esetben nem alakítható ki hasznos háromszög.

   Ezekhez a kiválasztott Vertex-ekhez kell tehát egy harmadikat választani. Ez potenciálisan valamelyik Stripe-ból a második Vertex. Annak eldöntésére, hogy a két lehetséges háromszögből melyik a megfelelőbb, a háromszög kiválasztó algoritmust kell használni.

2. További háromszögek kiválasztása

   Ha csak az egyik Stripe tartalmaz további Vertex-et, akkor abból kell a következőt kiválasztani. Amennyiben mindkettő Stripe tartalmaz további Vertex-eket, akkor egy kiválasztó függvény fog dönteni arról, hogy melyik Stripe-ból vegye be a következő Vertex-et.

   Két háromszög közül ki kell tehát választani azt, amelyik megfelelőbb a mesh kialakításának szempontjából. Ezt a feladatot a háromszög kiválasztó algoritmus fogja elvégezni.

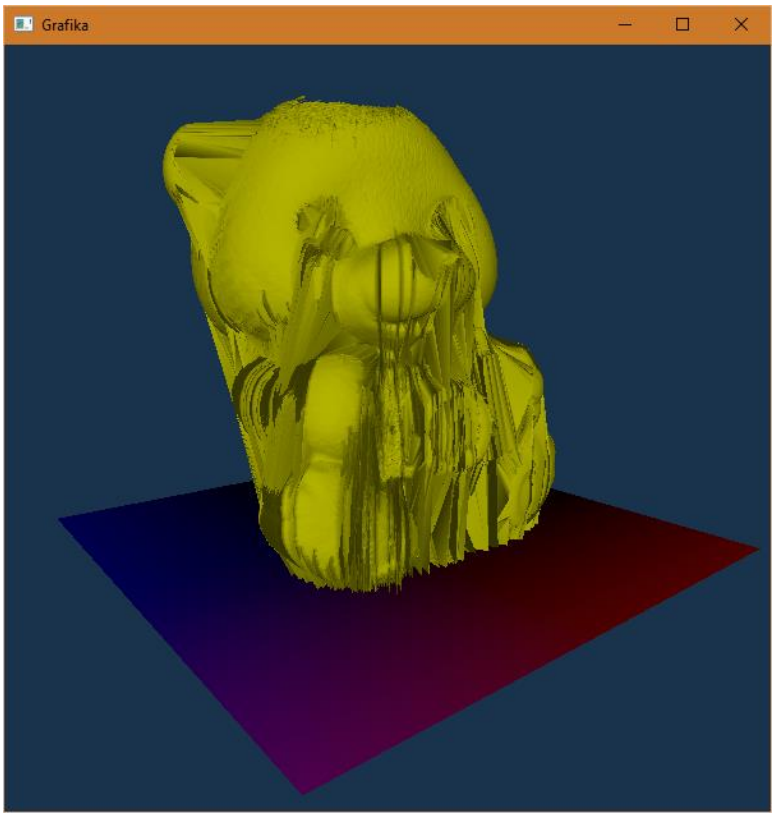
3. Iteráció folytatása, amíg van szabad Vertex

   A kiválasztott három Vertex alkot egy háromszöget, így ezt rögzítjük a kimeneti listában. Ezt követően kezdődik az egész algoritmus elölről az alábbi módon. A következő iterációban a kezdeti pontoknak mindkét Stripe-ból azt a Vertex-et választjuk, amelyik a legnagyobb indexű az adott Stripe-on belül azok közül a Vertex-ek közül, melyek már be lettek választva egy háromszögbe. Az

iterációkat egészen addig hajtjuk végre, amíg létezik olyan Vertex, mely még nem része a háromszögelésnek.

Háromszög kiválasztás

Számos faktort figyelembe lehet venni annak érdekében, hogy minél szebb, esztétikusabb háromszögeket készítsünk. Ezek közül talán a legelterjedtebb az a módszer, ahol a legkisebb szög maximalizálása a cél. (Ehhez hasonló módszer például a Delaunay trianguláció.) Ilyenkor tehát az a háromszög lesz a megfelelőbb, amelyiknek nagyobb a legkisebb szöge. Ez azonban itt nem alkalmazható, mert a háromszögelendő Cloud struktúrája kötött. Emiatt a háromszögelés folyamán be kell tartani néhány szabályt. Ezek miatt a legkisebb szöget maximalizáló algoritmus működésképtelen lesz olyan szempontból, hogy a generált háromszögháló messze nem lesz ideális.

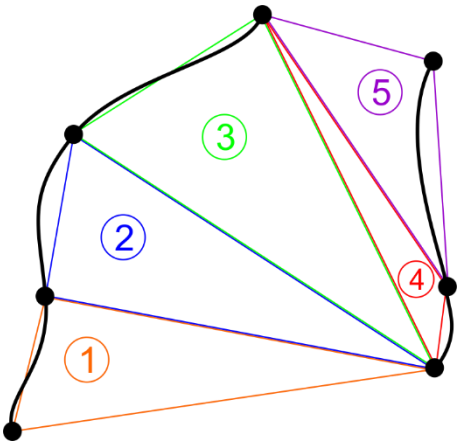


16. ábra: A maximalizált legkisebb szöggel való háromszögelés konkrét hibája

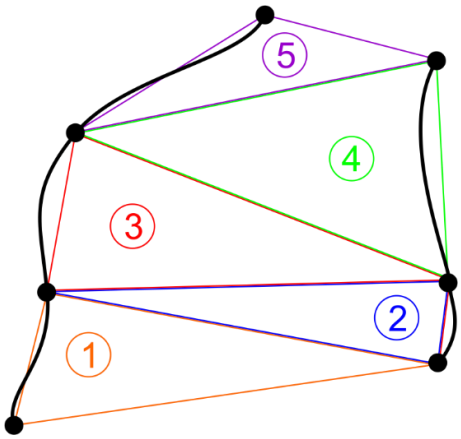
A probléma leggyakrabban olyankor fordul elő, amikor az egyik Stripe pontsűrűsége egy adott szakaszon sokkal nagyobb, mint a másik Stripe hasonló „magasságban[1]" levő pontjainak sűrűsége. A probléma részletesebben egy ábrával kerül bemutatásra, mely

[1] Itt függőleges rotációval készült szkennelésről van szó, ezért a magasság az y tengely mentén értelmezendő.

egy rossz (17. ábra) háromszögelést mutat be két Stripe között. Mellette látható a jó háromszögelése (18. ábra), így megfigyelhető a jelenség.



17. ábra: A maximalizált legkisebb szöggel való háromszögelés egy egyszerű példája



18. ábra: A vízszinteshez legközelebbi él kiválasztása a háromszögeléshez

Ehelyett inkább úgy történik a két háromszögből a megfelelő kiválasztása, hogy a háromszög új élei (ebből pontosan kettő van, hiszen az egyik éle az előző lépésben egy háromszöget alkotott) kerülnek összehasonlításra a Stripe-ok átlagos meredekségével.

Háromszög kiválasztó algoritmus

1. Stripe-ok meredekségeinek kiszámítása

   A két Stripe-ra vonatkozó átlagos meredekség kiszámításához, a két Stripe átlagos meredekségéből származó normalizált vektorokat össze kell adni. Ez által a vektor által meghatározott irány használható a továbbiakban.

   Ezt a számítást az adott szkenner legtöbb beállítása esetében lehet közelíteni a $(0; 1; 0)$ vektorral, hiszen az eszköz függőleges tengely körül képes rotációs transzformációt végezni (ehhez az is szükséges, hogy a lézer úgy legyen beállítva, hogy a lézervonal függőleges síkra való vetülete függőleges legyen).

2. Kiválasztás a Stripe meredekségétől való eltérés alapján

   Mindkét háromszög esetében meg kell határozni, hogy az új oldalak (amelyek még nem szerepelnek korábbi háromszögelésben) mekkora szöget zárnak be az 1. lépésben meghatározott irányra való merőlegessel A két háromszög közül az lesz a megfelelőbb, amelyiknek valamelyik oldala a legkisebb szöget zárja be ezzel.

Szélsőséges esetek

Figyelembe kell venni, hogy számos szélsőséges eset is előfordulhat. Ilyenek például, hogy egy Stripe egy része nem látszódik, hiszen például egy konkáv alakzat bizonyos részei nem láthatóak minden irányból.

Hasonló módon előfordulhat például az is, hogy kisebb nagyobb megszakításokkal, mint egy „szaggatott vonal" szerepelnek a Vertex-ek a Stripe-ban.

Lehetséges, hogy az egyik Stripe sokkal rövidebb, mint a másik, hiszen egy transzlációs szkenneléskor a szkennelendő tárgy széléhez ért a lézer, vagy egyszerűen csak lyukas a tárgy, illetve számos további ok miatt.

Fontos gondolni arra is, hogy sokszor olyan pontok is bekerülhetnek a pontfelhőbe, melyek valójában nem részei a szkennelt objektumnak, csak például valamilyen háttérvilágítást a lézervonal részének érzékel a vonalkövető.

Szélsőséges esetek kezelése

A hibakezelés számára egy külön modul kell, hogy készüljön, azonban néhány heurisztika definiálásával elkerülhető rengeteg hibás háromszög készítése. Ilyenek például a háromszög kerülete, illetve területe. Felhasználhatóak például a szkennelési beállításokat a heurisztikák felállításához. Amennyiben tudni lehet, hogy milyen távol van egymástól a két szkennelt Stripe, illetve, hogy a Stripe-on belül milyen messze vannak egymástól a pontok, akkor ki lehet számítani egy átlagos háromszög kerületét és területét. Ez alapján felső korlátot lehet adni erre a két metrikára, melyek fölött már nagyon valószínűtlen, hogy helyes háromszögről van szó.

Ennek értelmében további metrikák definiálása is lehetséges, melyekkel tetszőlegesen bővíthető a hibakezelésnek ez a része. Ezekhez minden új metrika esetében szükséges egy metrika számítási függvényt elkészíteni, valamint egy hozzá tartozó szélsőértéket, mely alapján eldönthető, hogy megfelelő-e.

Hibakezelés módja

Minden metrika esetében kiszámítható egy valószínűség is, mely megmondja, az adott háromszög helyességének valószínűségét. Ezen valószínűségekhez minden metrika esetén súlyokat kell rendelni, majd ezen adatok alapján egy véső valószínűséget kell számolni.

Végül pedig a döntést ennek a számított valószínűségnek függvényében lehet meghozni. Természetesen azt is érdemes figyelembe venni, hogy amennyiben valamelyik metrika

nagyon szélsőségesen gyenge értéket ad, akkor hiába jó a többi metrikából számított valószínűség, talán érdemes mégis hibásnak ítélni a háromszöget.

Hibakezelő algoritmus

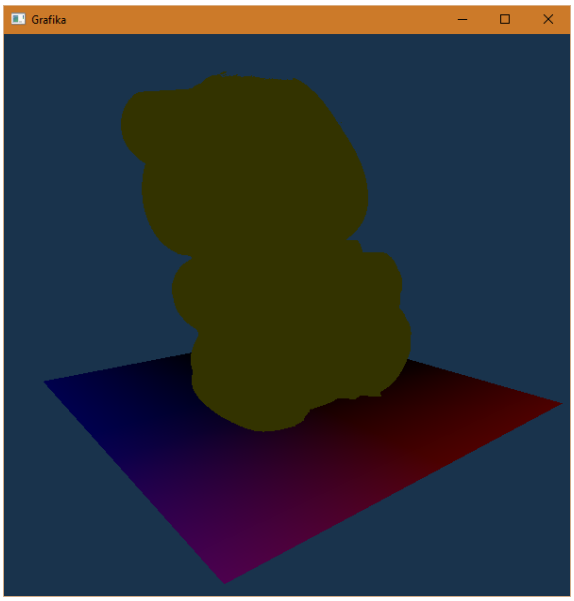
1. Minden metrika kiértékelése.
2. A metrikákhoz rendelt küszöbértékek alapján mindegyik esetében eldönthető, hogy a háromszög elfogadható-e.
3. Amennyiben van olyan metrika, amelyik alapján a háromszög nem fogadható el, akkor a háromszög nem lehet része a kimeneti háromszögek listájának. Ellenkező esetben (tehát abban az esetben, ha minden metrika alapján helyes a háromszög) a háromszög megfelelő.

Vizuális információk, további funkciók

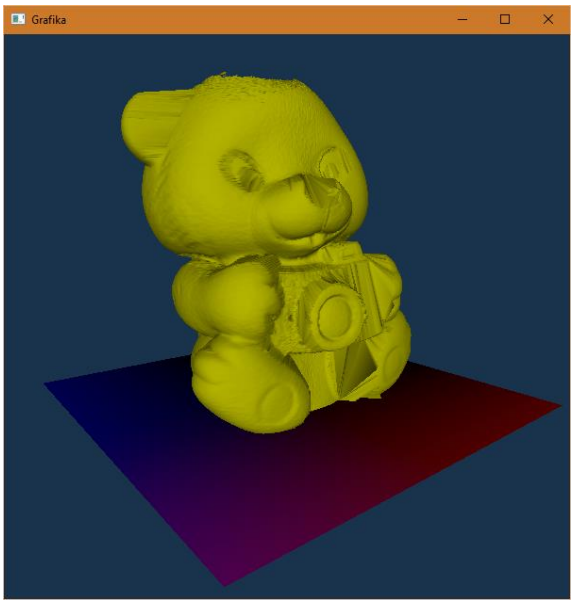
Közelítő normálvektorok kiszámítása

Az árnyaláshoz szükség lesz a csúcspontok normálvektorára, így ezeknek egy egyszerű közelítésének hatékony kiszámítását is el lehet itt végezni. Amikor elkészül egy új háromszög, akkor annak három Vertex-éhez hozzá kell adni az általuk meghatározott sík egység hosszú normálvektorát. Ez úgy kell megvalósítani a Vertex-ben, hogy számon kell tartani a már hozzáadott normálvektorok darabszámát. Így minden további hozzáadott normálvektor esetén a korábbi normálvektorok átlagát a korábbi vektorok darabszámával súlyozva kell az új normálvektorhoz adni, majd ezt normalizálni.

Ehhez azonban további információra is szükség van, hiszen a normálvektor egy sík esetében két irányba állhat. Amennyiben a transzlációs és rotációs információk elérhetőek, akkor ezt a kétértelműséget fel lehet oldani.



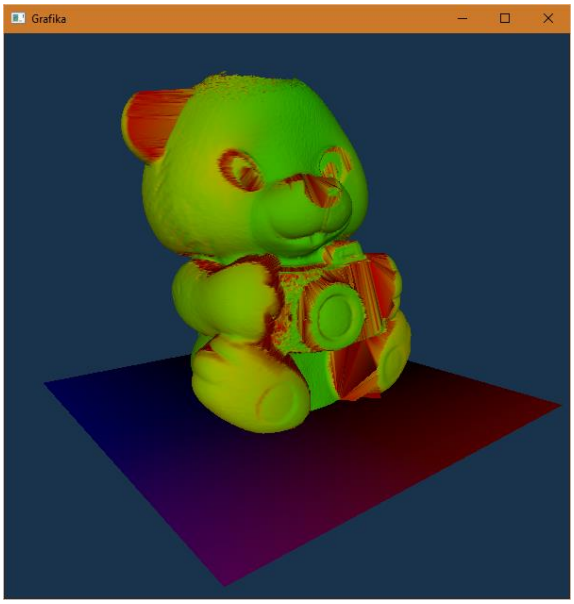
19. ábra: Normálvektorok nélküli háromszögháló megjelenítése



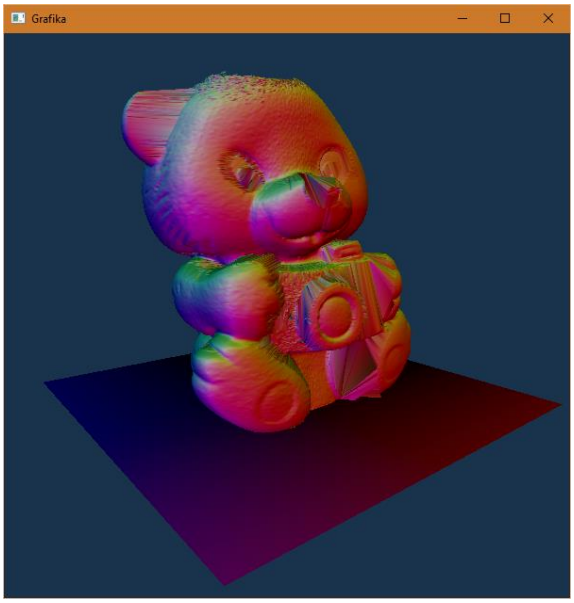
20. ábra: Háromszögháló megjelenítése a normálvektorok kiszámításával

Hibák vizualizációja

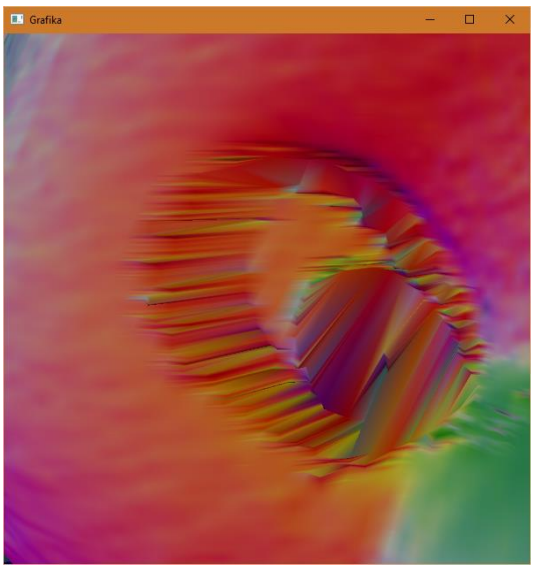
Lokális hibainformációt is számíthatunk a Vertex-ekhez, a korábban már említett metrikák alapján. Ezeket a normálvektorhoz hasonlóan érdemes tárolni a Vertex-ben, hogy újabb háromszög esetén könnyű legyen hozzáadni az új hibainformációt. Ez alapján például a csúcspont színét is állíthatjuk, mellyel egy hőtérképhez hasonlóan rendkívül látványos információt ad a különféle hibákról.



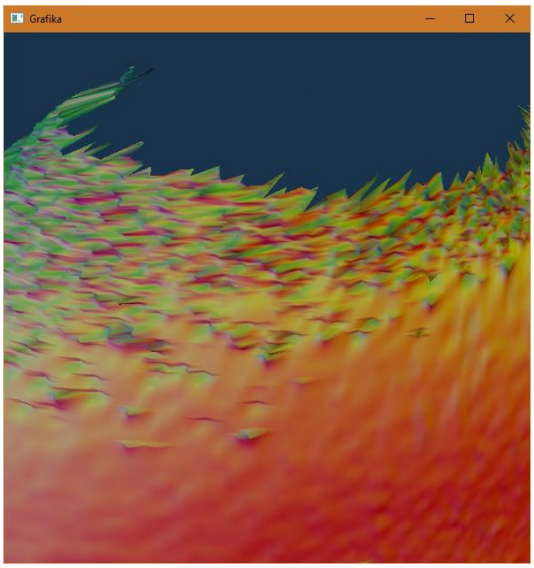
21. ábra: Hibák hőtérképes vizualizációja a generált háromszögek területe alapján



22. ábra: Hibák hőtérképes vizualizációja a háromszögek normálvektorainak iránya alapján



23. ábra: A maci szeménél látszik, hogy az egyes háromszögek csúcspontjainak nagyon eltérő irányba mutatnak a normálvektorai, ezért csíkosnak látszik a felület



24. ábra: A maci fejének tetejénél látszik, hogy egyes hibás pontok kilógnak a környezetükből

Hengeres mesh

Mesh generálás

1. Henger generálása

   A henger generálásához szükség van egy teljes 360°-os rotációs szkennelésre, melyből a szükséges paramétereket ki lehet nyerni. A vízszintes felosztása a szkennelés finomságától függ, ahány függőleges Stripe került szkennelésre, annyifelé kell osztani a hengert. A függőleges irányú felosztás tetszőleges lehet, azonban ez automatikusan is kiszámítható az első rotációs szkennelésből. A henger függőleges tengelyű, hiszen a szkenner csak ilyen irányban képes rotációs transzformációra. A tengely felső illetve alsó csúcspontja pedig a szkennelt pontfelhő végpontjainak átlagában (vagy a teteje esetében maximumában az alja estében a minimumában) helyezkedik el. A hengert alkotó Vertex-ek az oszlopokban fentről lefelé, az oszlopok pedig jobbról balra helyezkednek el. A háromszögek ugyanebben a sorrendben, az átlós élek északnyugat – délkelet irányúak, a háromszögek körüljárási iránya az óra járásának megfelelő irányú.

2. Henger alakítása

   A henger egyes pontjai sugár irányban való mozgatása, hogy azok a megfelelő helyre kerüljenek.

   Mivel a generált henger vízszintesen ugyanannyi részre van osztva, mint ahány szkennelési Stripe volt, így a henger minden Vertex oszlopához pontosan egy Stripe tartozik. Az egyes pontok mozgatásának kiszámításához tehát elegendő a hozzájuk tartozó Stripe.

   A henger függőleges felosztása azonban lehet finomabb, mint maga a szkennelés. Ebből kifolyólag az egyes generált pontokat, a Stripe-ban fölötte és alatta elhelyezkedő Vertex-ek távolságának függvényében lineárisan interpolálva kell elhelyezni.

Hibacsoportosítás

1. Hibás háromszögek azonosítása

   A hibás háromszögek azonosítása többféle algoritmus alapján történhet, ezért egy általános módszer szükséges erre, hogy ez később tetszőlegesen bővíthető legyen további algoritmusokkal.

   Az interfész annyit határoz meg, hogy az algoritmus egy igaz vagy hamis értékkel nyilatkozzon minden háromszögről, hogy szerinte megfelelő-e, vagy sem.

   Egy háromszög jóságát különböző metrikák alapján lehet eldönteni, majd az eredményeket összevetve (például többségi alapon) kell megállapítani, hogy ténylegesen jó-e a háromszög.

   - A háromszögek kerülete és területe alapján is eldönthető, hogy jók-e. Amennyiben az említett metrikák túl nagyok, akkor a háromszög rossz.
   - Ilyen lehet például az eredeti Vertex-ektől túl messzire interpolált pontok alkotta háromszögeket hibásnak ítélő algoritmus.

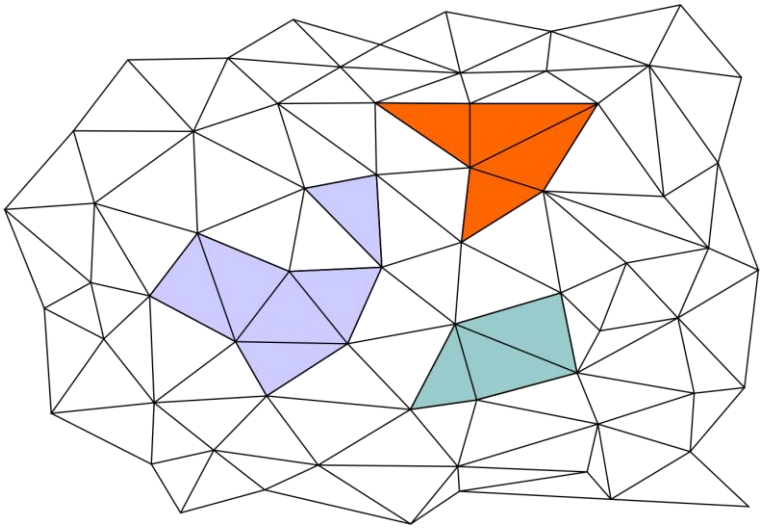
   A háromszögeket az esztétikusságuk alapján megítélő algoritmusokat is figyelembe lehet venni. Ezek különböző arányok alapján ítélik meg a háromszögeket:

   - a legkisebb szög hegyessége
   - a körülírt kör és a belső érintőkör sugarának az aránya
   - a leghosszabb oldal és a hozzátartozó magasság aránya
   - a legrövidebb oldal és a körülírható kör sugarának aránya

2. Hibás háromszögek csoportosítása

   A korábban hibásnak ítélt háromszögeket csoportokra kell osztani, melyek alapul szolgálhatnak a további szkennelések beállításaihoz, valamint a rendelkezése álló szkennelésekből való javításokhoz.

   Két háromszög akkor tartozik ugyanabba a hibacsoportba, ha a hibás háromszögek csúcsai mentén létezik köztük olyan út, melynek minden éle egy hibás háromszög éle.



25. ábra: Három hibacsoport három különböző színnel

A henger struktúrának köszönhetően azonban a háromszögek közti navigálás könnyen megvalósítható, hiszen néhány elemi művelet sorozataként a kívánt szomszédok indexi meghatározhatóak. Ennek segítségével egy rossz háromszög környező háromszögeit kell meghatározni, majd azok környező háromszögeit, és ezt folytatva addig, amíg van még rossz háromszög. Minden lépésben figyelni kell arra, hogy a korábban már megtalált háromszögek és a duplikációk ki legyenek szűrve, így a folyamat ne kerülhessen végtelen ciklusba. Minden lépésben le kell ellenőrizni, hogy a megtalált szomszédos háromszögek ténylegesen rosszak-e. Amennyiben ezzel a módszerrel nem lehet tovább bővíteni egy csoportot, akkor az a csoport készen van, a csoport háromszögei eltávolíthatóak a rossz háromszögekből. Ezt addig kell folytatni, ameddig el nem fogynak a rossz háromszögek.

3. Hibacsoportok bekerítése

Erre azért van szükség, mert ezekhez a határokhoz lesznek hozzáillesztve az új, javító háromszöghálók. A határok az óra járásának megfelelő sorrendben kerülnek előállításra, tehát a körüljárás irányában haladva a hibás Vertex-ek a jobb oldalon helyezkednek el.

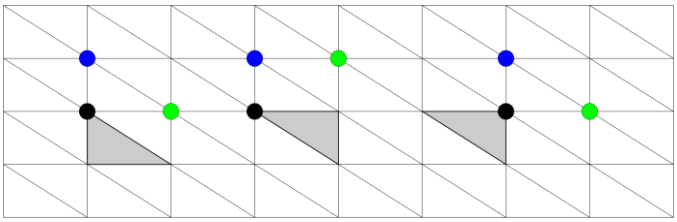
Először két kezdőpontot kell kiválasztani a hibacsoportokból.

a. Amennyiben a csoport nem tartalmazza a felső vagy az alsó csúcsot, akkor az alábbi módon kell eljárni:

   i. A hibacsoport legmagasabb pontját kell megkeresni, hiszen az efölötti Vertex biztosan jó, ez lesz az első kiindulási pont. (A hibás Vertex-ek legfelső Vertex-e fölött kell, hogy legyen jó Vertex, hiszen ez a hibacsoport nem tartalmaz olyan háromszöget,

melynek része a legfelső csúcs, tehát a felső gyűrűből sem tartalmaz Vertex-et.)

ii. Az első kiindulási pont a háromszögek elhelyezkedésétől függően háromféleképpen helyezkedhet el. Az alábbi ábrán fekete pont jelöli a hibacsoport legmagasabb csúcspontját, fölötte kék pont az első kiindulási pont, zöld pedig a második kiindulási pont. A második kiindulási pont meghatározásához tehát ezeket az eseteket kell megkülönböztetni.



26. ábra: Kiindulási pontok (első kék, második zöld) lehetséges elhelyezkedése a hibacsoport legmagasabb (fekete) pontjához képest, a szürke háromszög a hibacsoport egyik legmagasabban levő háromszöge

b. Amennyiben a hibacsoport tartalmazza a felső vagy az alsó csúcsot, akkor az alábbi módon kell eljárni:

i. Mivel valamelyik szélső Vertex (a felső vagy az alsó) benne van, biztos, hogy valamelyik Stripe-nak is benne van legalább egy Vertex-e (hiszen hibás háromszögként került be az alsó vagy a felső Vertex). Keresni kell egy ilyen Stripe-ot.

ii. Ezen a Stripe-on meg kell keresni (a hibás szélső Vertex irányából indulva) az utolsó hibás Vertex-et, majd az ezt követő Vertex lesz az első kiindulási pont. Ez után a Stripe irányához képes a megtalált Vertex körül jobbról az első nem hibás Vertex lesz a második kiindulási pont.

Megvan tehát a két kiindulási pont, melyek meghatároznak egy irányt. Mivel tudjuk, hogy ez az előre, megkülönböztethetjük a nyolc irányt a menetirányhoz képest (jobbra, jobbra-előre, előre, balra-előre, balra, balra-hátra, hátra, jobbra-hátra). A menetirányhoz képest kell jobbról indulva a következő jó Vertex-et megkeresni. Figyelni kell azonban arra, hogy a bekerítés ne kerülhessen zsákutcába, ezért a megfelelő irányokba való továbbhaladáshoz az alábbi irányokban is jó Vertex-nek kell állnia:

a. Jobbra esetén: jobbra, jobbra-előre, előre, balra-előre

b. Jobbra-előre esetén: jobbra-előre, előre, balra-előre

c. Előre esetén: előre, balra-előre

d. Balra előre esetén: balra-előre, balra

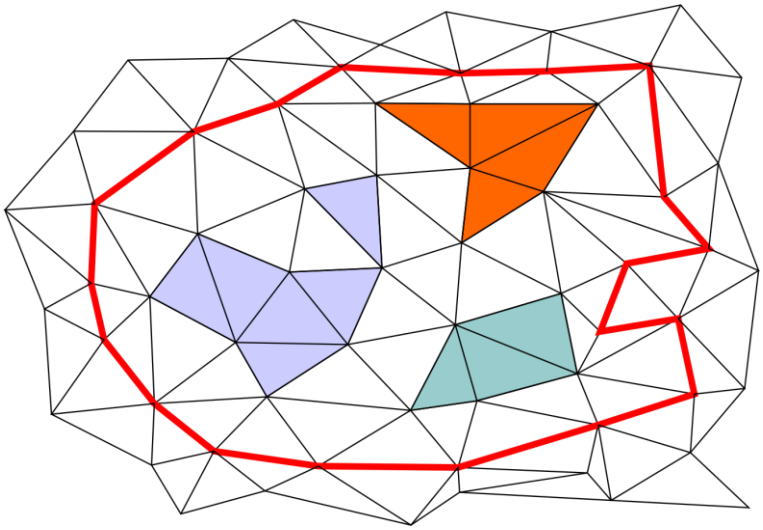
e. Balra esetén: balra

f. Balra-hátra esetén: balra-hátra

Ezeken felül figyelni kell, hogy a függőlegeshez viszonyított északkelet–délnyugat irányba nem futnak élek, így abba az irányba nem lehet haladni.

A bekerítést végül addig kell folytatni, amíg vissza nem érünk egy olyan ponthoz, ami már az aktuális bekerítés része. (Ez azonban nem biztos, hogy a bekerítés eleje.)

Előfordulhat azonban, hogy nem a bekerítés elejéhez érünk vissza, mert az egyes hibacsoportok között már nincs hely a bekerítésnek. Ilyen lehet például, ha egyetlen sávban jó háromszögek helyezkednek el két hibacsoport között, akkor ott minden Vertex hibás. Az ilyen hibacsoportok automatikusan összevonásra kerülnek, és a peremből el kell hagyni azokat a Vertex-eket, amelyek a kezdőpont és a találkozási pont között szerepelnek. A bekerítésnél tehát figyelni kell a csoportösszevonásokra, hogy ne generálódjon egy perem többször.



27. ábra: Hibacsoportok összevonása a bekerítés által

Ez tehát egy olyan Vertex sorozat lesz, melynek elemei sorban követik egymást a generált háromszöghálóban, és egy zárt hurkot alkotnak. Ezek a Vertex-ek már nem részei a hibásnak ítélt háromszögeknek, hiszen azokon kívül helyezkednek el.

Az így elkészült határok között azonban olyan háromszögek is lehetnek, mely a hibacsoportban még nincsenek benne, valamint újabb Vertex-ek is kerülhetnek a hibaperemen belülre, például ha a hibás háromszögek körülzárnak jó Vertex-

eket. Szükséges tehát a hibaperemeken belüli – a hibás háromszögektől immár független – Vertex-ek meghatározása.

4. Hibaperemeken belüli Vertex-ek megkeresése

   Ezen háromszögek megkereséséhez a hibás háromszögek csoportosításánál alkalmazotthoz nagyon hasonlót lehet alkalmazni. A kiinduláshoz egy Vertex szükséges (például a korábbi hibás háromszögekből valamelyiknek az egyik csúcsa). Ezt követően ennek a szomszédjait, majd azoknak a szomszédjait kell meghatározni, egészen a határig. Figyelni kell, hogy a függőlegeshez viszonyított északkeleti és délnyugati irányokba nem szabad navigálni, mert ezekben az irányokban nincsenek élek.

Előnyök

A kidolgozott módszer segítségével gyorsan készíthető el a háromszögháló, valamint a háromszögháló egyenletessége a függőleges felosztás állításával manuálisan, de akár automatikusan is optimalizálható.

A módszer nagy előnye, hogy a mesh generáláskor zárt a háromszögháló, majd ezzel lehet tovább dolgozni. Később ez fontos, hiszen a végeredmény előállításához egyszer csak mindenképpen zárt mesh-t kellene előállítani.

Továbbá a hibás háromszögcsoportok eltávolítása után a hiányzó háromszögháló darabok peremei is adottak. Ez azt jeleni, hogy javító mesh-sek esetén csupán azok széleit kell összevarrni a korábban generált hibacsoportok peremeivel.

Amennyiben nem sikerül javítani az adott hibacsoportot, akkor is rendelkezésre áll az első szkennelésből hibásnak ítélt felületdarab, mely ugyan nem tesz eleget a pontossági követelményeknek, – hiszen ezért lett hibásnak ítélve – de a háromszögháló mégis zárt. Ez a rész értelemszerűen az adott hardverrel nem javítható, ezért ezen a részen a pontossági követelményeknek nem is lehet eleget tenni.

Kompromisszumok

Fontos megemlíteni, hogy a generált pontfelhő vertikálisan tetszőleges finom felosztású lehet, de ettől függetlenül a pontok nem pontosan ott lesznek, ahol a szkennelések történtek. Ebből azonban a legnagyobb tényleges hiba, ami keletkezhet, az a leghosszabb generált háromszög élének a fele (hiszen lineárisan interpolálva vannak a pontok).

A kezelhető objektumok irányában megkötés, hogy egy függőleges hengerből kialakíthatónak kell lennie, tehát például egy tórusz pontfelhőjéből nem lehet tórusz háromszöghálót generálni ezekkel az algoritmusokkal.

Algoritmikai komplexitás

Jelölések

p_{szkenn} – a szkennelt pontok száma

p_{gen} – a generált pontok száma

p_{stripe} – a Stripe mérete (hogy hány Vertex-et tartalmaz)

f – a generált henger oszlopainak felosztottsága (a bennük található Vertex-ek száma)

a – a hibás háromszögeket azonosító algoritmusok száma

h_h – a hibás háromszögek száma

h_{cs} – a hibacsoportok száma

Mesh generálás

1. Henger generálása

   A generálás során létre kell hozni a csúcspontokat és a háromszögeket, melyekhez néhány adatot ki kell nyerni a pontfelhőből. Ezek az adatok a felső és az alsó csúcs helye, valamint esetleg a felosztás mértéke. Minden metrikához legfeljebb egyszer kell bejárni az egész pontfelhőt, így ezek komplexitása $O(p_{szkenn})$. A pontok és háromszögek létrehozása $O(1)$-ben történik, így a teljes folyamat komplexitása $O(p_{szkenn}) + p_{gen} \cdot O(1) = O(p_{szkenn} + p_{gen})$.

2. Henger alakítása

   A Stripe – melynek pontjaihoz a generált Vertex-eket alakítjuk – kiválasztása $O(1)$-ben történik, hiszen pontosan annyiadik Stripe kell, ahányadik generált Vertex oszlopon vagyunk. A generáláskor a függőleges felosztás mértéke adott. Minden generált ponthoz (p_{gen}) meg kell határozni a fölötte és alatta levő két pontot a Stripe-on az interpoláláshoz, mely a Stripe pontjainak rendezettsége miatt $O\left[\log(p_{stripe})\right]$. Így a teljes folyamat komplexitása $p_{gen} \cdot O\left[\log(p_{stripe})\right]$.

Hibacsoportosítás

1. Hibás háromszögek azonosítása

   A hibás háromszögek azonosítására jelenleg implementált algoritmusok a pontok közötti távolságokat és a háromszögek méreteit elemzik. Ezekhez minden pontot és háromszöget algoritmusonként legfeljebb egyszer kell bejárni. Jelenleg öt ilyen algoritmus van, de az algoritmusok számával is számolva a teljes komplexitás $a \cdot O(p_{gen})$.

2. Hibás háromszögek csoportosítása

   A hibás háromszögek halmazát először egyszer szükséges rendezni az indexük szerint, hogy később hatékonyan lehessen vele dolgozni. Ezt $O[h_h \cdot \log(h_h)]$-ban lehet megtenni. Ezt követően ahány hibacsoport van (h_{cs}), annyiszor el kell végezni azoknak a felderítését. A felderítés során a csoport mindig egy réteggel – mint a hagyma rétegei – bővül. A szükséges rétegnövelések száma csoportonként $O\left(\sqrt{h_h/h_{cs}}\right)$. (A perem elemszámát a teljes hibacsoport gyökével közelítem. Szélsőséges esetben természetesen elképzelhető, hogy a csoport összes eleme része a peremnek.) A rétegnövelés lépései az alábbiak:

   a. Az aktuális réteg szomszédos háromszögeinek meghatározása: $O(1) \cdot O\left(\sqrt{\frac{h_h}{h_{cs}}}\right)$

   b. Egy ilyen csoport elemszáma nagyságrendileg legfeljebb $\sqrt{\frac{h_h}{h_{cs}}}$, melyeket a későbbi algoritmusokhoz rendezni kel: $O\left(\sqrt{\frac{h_h}{h_{cs}}} \cdot \log \sqrt{\frac{h_h}{h_{cs}}}\right)$

   c. Rendezés után az ismétlődéseket el kell távolítani: $O\left(\sqrt{\frac{h_h}{h_{cs}}}\right)$

   d. Ezek közül el kell távolítani a már korábbi iterációkban a csoportba bevett háromszögeket (rendezett halmazok különbségének képzése): $O\left(\sqrt{\frac{h_h}{h_{cs}}} + \frac{h_h}{h_{cs}}\right)$

   e. Az így kapott háromszöggyűrű közül ki kell válogatni azokat, amelyek ténylegesen rosszak, vagyis a metszetét kell képezni a háromszöggyűrűnek és az összes rossz háromszögnek. Mindkét halmaz rendezve van, tehát: $O\left(\sqrt{\frac{h_h}{h_{cs}}} + h_h\right)$

Végül pedig, ha elkészül egy csoport – tehát tovább nem bővíthető – akkor el kell távolítani a hibás háromszögek közül a csoport háromszögeit: $O\left(\sqrt{\frac{h_h}{h_{cs}}} + h_h\right)$

Ezeket összegezve az alábbiakat kapjuk:

$$O[h_h \cdot \log(h_h)] + h_{cs} \cdot \left\{O\left(\sqrt{\frac{h_h}{h_{cs}}}\right) \cdot \left[O(1) \cdot O\left(\sqrt{\frac{h_h}{h_{cs}}}\right) + O\left(\sqrt{\frac{h_h}{h_{cs}}} \cdot \log\left(\sqrt{\frac{h_h}{h_{cs}}}\right)\right) + \right.\right.$$

$$\left.\left. + O\left(\sqrt{\frac{h_h}{h_{cs}}}\right) + O\left(\sqrt{\frac{h_h}{h_{cs}}} + \frac{h_h}{h_{cs}}\right) + O\left(\sqrt{\frac{h_h}{h_{cs}}} + h_h\right)\right] + O\left(\sqrt{\frac{h_h}{h_{cs}}} + h_h\right)\right\} =$$

$$O\left\{h_h \cdot \log(h_h) + h_{cs} \cdot \left[\sqrt{\frac{h_h}{h_{cs}}} \cdot \left(\sqrt{\frac{h_h}{h_{cs}}} \cdot \log\left(\sqrt{\frac{h_h}{h_{cs}}}\right) + \sqrt{\frac{h_h}{h_{cs}}} + h_h\right) + \sqrt{\frac{h_h}{h_{cs}}} + h_h\right]\right\} =$$

$$O\left\{h_h \cdot \log(h_h) + h_{cs} \cdot \left[\frac{h_h}{h_{cs}} \cdot \log\left(\sqrt{\frac{h_h}{h_{cs}}}\right) + \sqrt{\frac{h_h}{h_{cs}}} \cdot h_h\right]\right\} =$$

$$O\left[h_h \cdot \log(h_h) + h_h \cdot \log\left(\sqrt{\frac{h_h}{h_{cs}}}\right) + \sqrt{\frac{h_h}{h_{cs}}} \cdot h_h \cdot h_{cs}\right] =$$

$$O\left[h_h \cdot \log(h_h) + \sqrt{\frac{h_h}{h_{cs}}} \cdot h_h \cdot h_{cs}\right] = O\left({h_h}^{1,5} \cdot {h_{cs}}^{0,5}\right)$$

Ez azt jelenti tehát, hogy a hibás háromszögek száma befolyásolja legerősebben ennek az algoritmusnak a futásidejét. A végső komplexitás tehát $O\left({h_h}^{1,5} \cdot {h_{cs}}^{0,5}\right)$.

3. Hibacsoportok bekerítése

Itt az egyes bekerítésekhez szükséges kiindulási pontok megkeresése $O(h_h/h_{cs})$-ben megvalósítható (szokásos szélsőérték keresés). A felső vagy alsó végpontokat tartalmazó csoportok esetén $O(p_{stripe})$, melyek skálázódás szempontjából kevésbé lényegesek, hiszen ilyen csoportból mindig legfeljebb kettő lesz. A teljesesség kedvéért a komplexitás tehát $O\left(h_h/h_{cs} + p_{stripe}\right)$ lenne, melyet $O(h_h/h_{cs})$-vel közelítek.

A kiindulási pontokból a folytatás minden lépése $O(1)$-ben tehető meg, a perem hossza pedig $\sqrt{h_h/h_{cs}}$. (A korábbi pontban erre a közelítésre tett megjegyzés természetesen itt is igaz.)

Az első néhány hibás pont levágása $O(\sqrt{h_h/h_{cs}})$-ben elvégezhető.

Így végül a h_{cs} darab hibacsoport bekerítésének teljes komplexitása: $h_{cs} \cdot \left(O(h_h/h_{cs}) + O\left(\sqrt{h_h/h_{cs}}\right) \cdot O(1) + O(\sqrt{h_h/h_{cs}})\right) = O(h_h)$.

4. Hibaperemeken belüli Vertex-ek megkeresése

   Ennek az algoritmusnak a komplexitása megegyezik a hibás háromszögek csoportosításánál használt algoritmuséval: $O\left({h_h}^{1,5} \cdot {h_{cs}}^{0,5}\right)$.

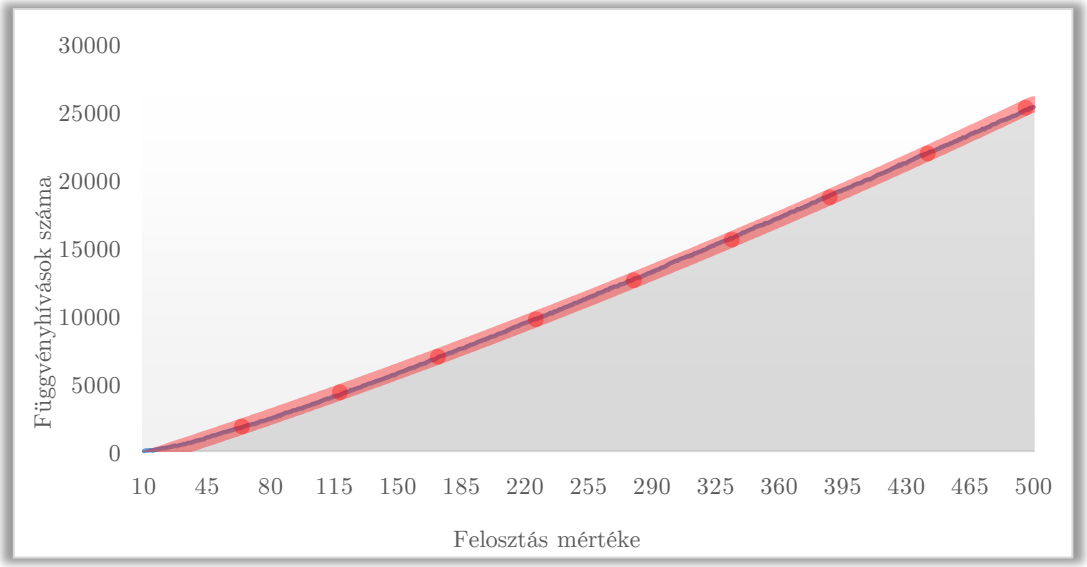
Mérési igazolások

Számos mérés elvégzése kerül bemutatásra a háromszög környezetében található háromszögek megtalálását megvalósító algoritmusról. Ez az algoritmus az alapja a hibás háromszögek csoportosításának, illetve a hibás háromszögek azonosítása esetén is fel lehet használni azoknak egy megadott sugarú környezetében levő háromszögek meghatározására is.

Ez az algoritmus paraméterül kap egy háromszöget illetve egy mélységet, majd ez alapján kiszámítja, hogy az adott sugarú környezetben mely háromszögek vannak. Első lépésben azokat a háromszögeket határozza meg, melyekkel van közös csúcsa. Második lépésben azokat, amelyekkel az elsőző lépésben kapott háromszögeknek van közös csúcsa és így tovább.

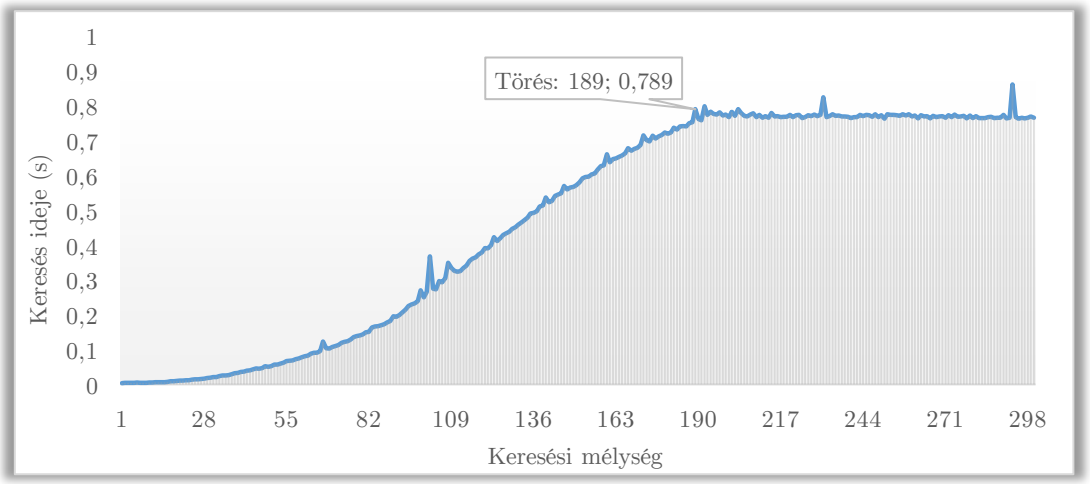
Nagyon fontos volt, hogy ez az algoritmus hatékonyan fusson, hiszen teljesítmény szempontokból egy kulcsfontosságú pont.

A következő diagramon azt lehet látni, hogy a generált háromszögháló függőleges felosztását növelve – ezzel mind a Vertex-ek, mind pedig a háromszögek számát növelve – hogyan változik az algoritmus hívásainak száma a teljes generálási folyamat során. Jól megfigyelhető, hogy milyen sokszor fut a függvény, ezért ennek hatékonysága kulcsfontosságú.



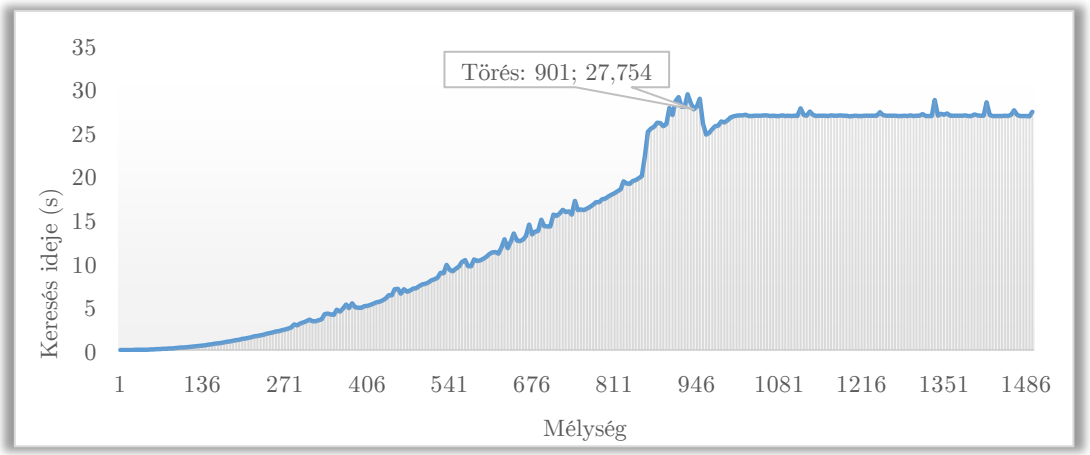
28. ábra: Háromszög körüli háromszögeket meghatározó függvény hívásainak száma a teljes generálási folyamat során a felosztás függvényében, másodfokú polinomiális trendvonallal

Az alábbi diagram az algoritmus futását mutatja egy olyan háromszögelés esetén, ahol a henger palástjának közepétől indul a szomszédossági keresés. Körülbelül a felénél eljut a henger tetejéhez és aljához, ahol a legnagyobb szomszédossággal rendelkező háromszögek vannak. Ezt követően a végén – a diagramon töréspontal jelölt részhez jutva – az összes háromszög be lett véve, így ezt követően hiába nő a mélység, továbbra sem tud több háromszöget bevenni, mert körbeért a palástón.



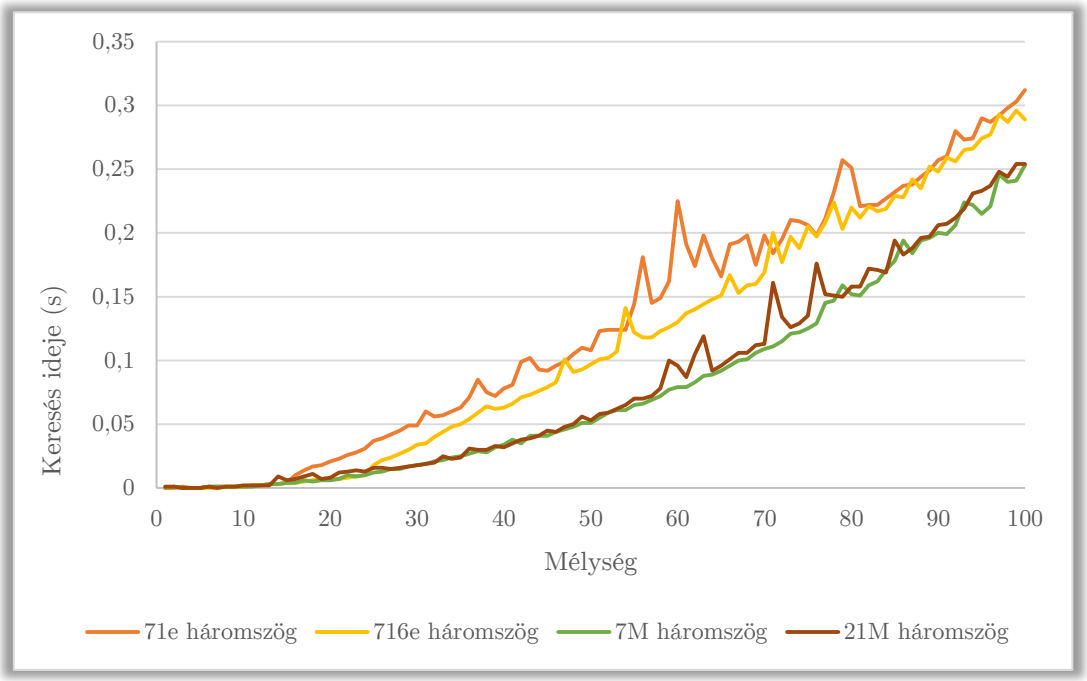
29. ábra: Háromszög körüli háromszögeket meghatározó függvény futási ideje a szomszédossági mélység függvényében 143.000 háromszögnél

Egy nagyobb háromszögszám esetén jól megfigyelhető a szomszédosság keresésének ideje a mélység függvényében. Itt a geometria szintén beleszól, azonban itt a futás végén. Ez azzal magyarázható, hogy a függőleges felosztás növelésével az algoritmus hamar körbeér vízszintes irányba, majd a töréspontohoz közel érve a geometriában eljut a henger aljához és tetejéhez. Ezen a két helyen az algoritmusnak természetesen sokkal több dolga van, hiszen itt a háromszögeknek több száz szomszédjuk van.



30. ábra: Háromszög körüli háromszögeket meghatározó függvény futási ideje a szomszédossági mélység függvényében 1.430.000 háromszögnél

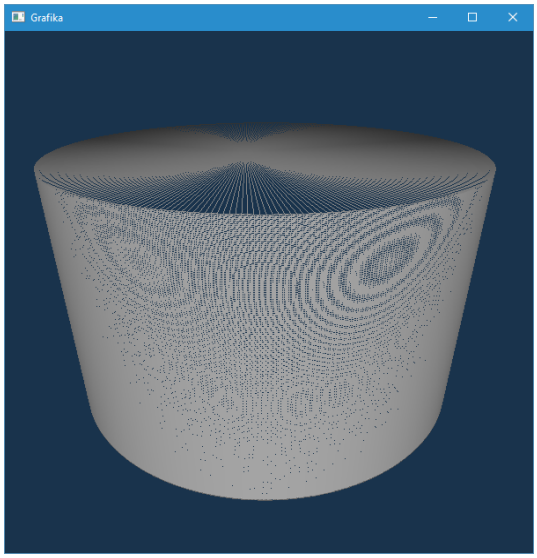
Amit érdemes még kihangsúlyozni, hogy különböző háromszögszámok esetén sem romlik az algoritmus futási teljesítménye. A függvénynek a menete az érdekes, azonban megfigyelhető, hogy nagyobb háromszögszám esetén jobban teljesít. Ez azzal indokolható, hogy a négy teszteset (a különböző háromszögszámok) ugyanazon a modellen készültek, azonban a függőleges irányba való nagyobb felosztás esetén a szomszédosságok máshogy alakulnak. Míg kisebb háromszögszám esetén majdnem az egész háromszögháló bejárásra kerül, nagyobb háromszögszám esetén csak a palást egy kisebb középső része. Ebből kifolyólag a kisebb háromszögszámok esetén, sok helyen kell halmazok metszetét, különbségét számítani, melyek a nagyobb háromszögszám estén az üres halmazokkal gyorsan lefutnak.



31. ábra: Háromszög körüli háromszögeket meghatározó függvény futási ideje a szomszédossági mélység függvényében, különböző háromszögszámok esetén

Eredmények

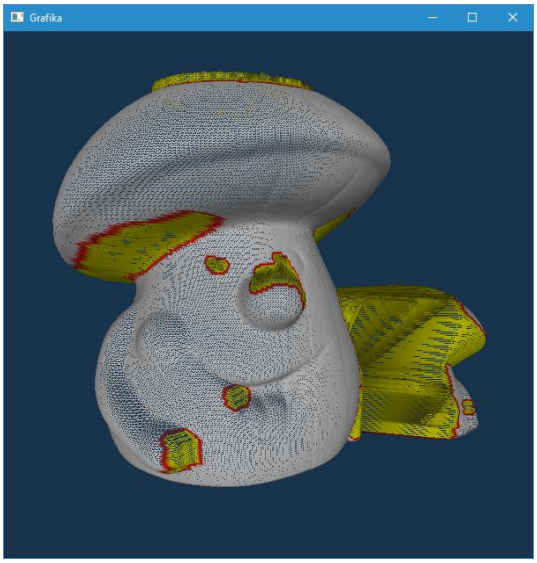
A munkamenet implementálásra került egy általam írt keretrendszerbe. Az algoritmusok eredményeit az alábbi képek szemléltetik.



32. ábra: A kezdetben generált henger háromszöghálója, melynek a sugara teljesen tetszőleges, hiszen a következő lépésben ezek lesznek a szkenneléshez illesztve

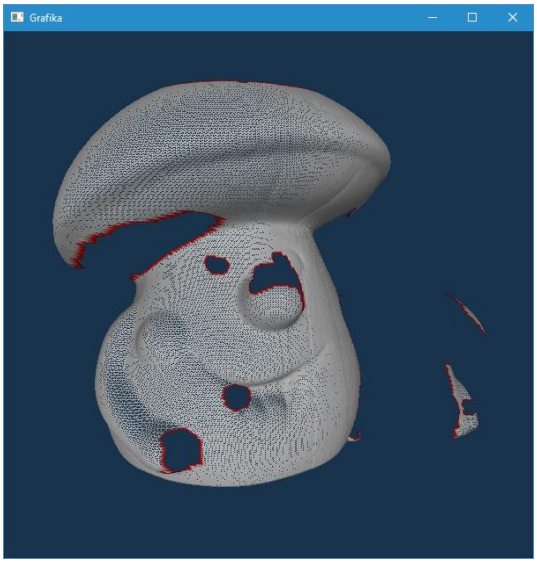
33. ábra: A henger háromszöghálója a szkenneléshez illesztés után, ahol a generált Vertex-ek kizárólag sugár irányba vannak mozgatva

34. ábra: A hibás háromszögeket detektáló különböző algoritmusok által hibásnak ítélt háromszögek, jelenleg még egyetlen háromszöghalmazként



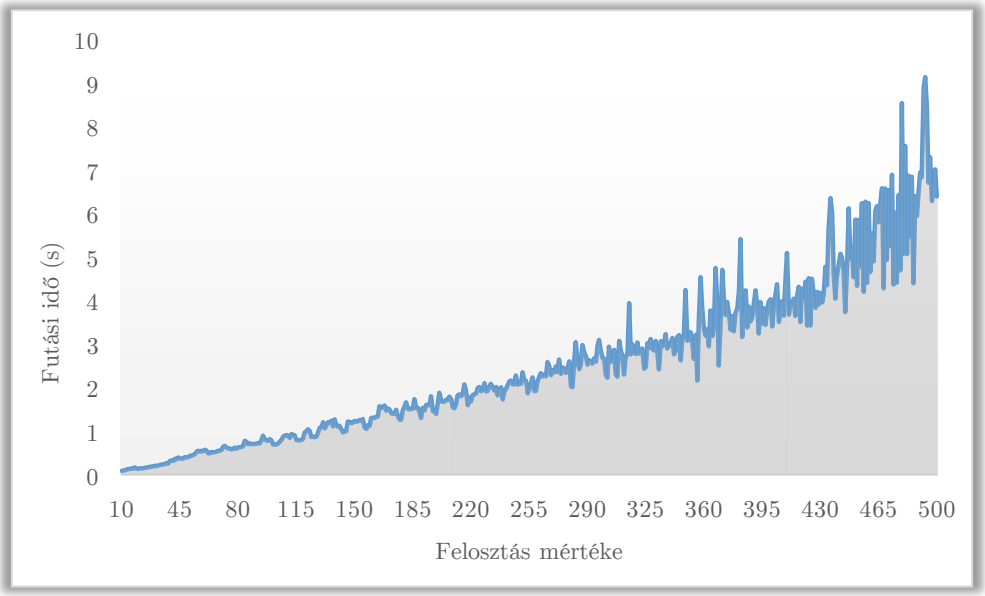
35. ábra: A hibás háromszöghalmaz részeinek külön bekerítése, melyben a hibacsoportok határai készülnek el, de az egyes hibacsoportok még nincsenek külön azonosítva

36. ábra: A csoportok azonosítása, a duplikációk kiszűrésével; fontos, hogy itt már minden csoportperem rendezve van, a közrezárt Vertex halmazok pedig diszjunktak



37. ábra: A köztes hibás háromszögek és Vertex-ek eltávolítása után előáll a javítandó mesh, minden csoportperem rendelkezésre áll Vertex folytonosan

A paraméterek beállításai 200-as felosztáshoz lettek igazítva, így nagyobb felosztások esetén ezeket újra be kell állítani. Ettől függetlenül a kisebb felosztás esetében a lineáris skálázódás jól látszódik. A függőlegesen 500 Vertex-re osztott háromszöghálón – mely közel 358 ezer háromszöget tartalmaz – mindössze 7 másodperc alatt fut le a teljes folyamat. A szélsőséges, kiugró – illetve a végén emelkedő – értékek a hibás paraméterek okozta nagyon sok kisméretű hibacsoport okozza.

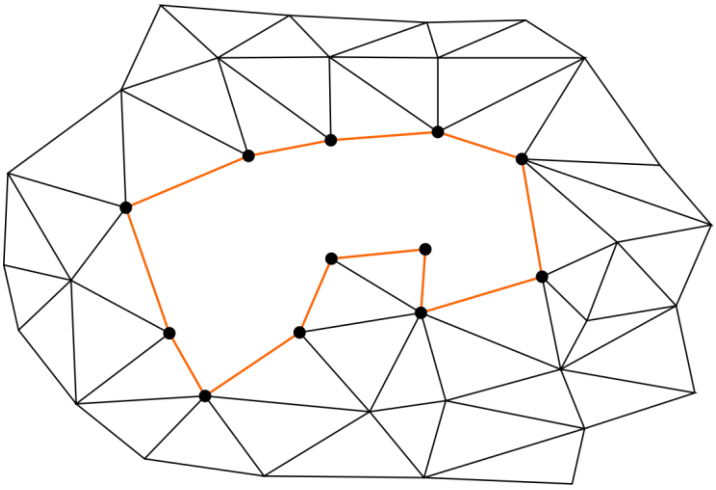


38. ábra: Teljes generálás futási ideje, statisztikai paraméterek változtatása nélkül (A fixen beállított paraméterek 200-as felosztáshoz vannak igazítva, ezért nagyobb felosztás esetén ezeket érdemes hangolni, vagy számításukat lehet automatizálni is.)

Szkennelés javítási lépései

Rendelkezésre álló mesh

Ebben a lépésben a szkennelt háromszöghálónak bizonyos hibafoltjait kell javítani. A hibafoltok az őket határoló pontokkal adottak oly módon, hogy minden egymást követő pont része egy élnek a háromszögelésben. Továbbá a javítandó területen belül nincsenek olyan szigetek, melyek helyesek lennének.



39. ábra: Egy javítandó terület, mely pontok sorozataként adott

Szkenner konfigurálása a javításokhoz

A javításokhoz két – alapvetően különböző – módszert lehet elkülöníteni. Az egyik a lehető legpontosabb javítás, a másik a lehető leggyorsabb. Jelen szkennernél általában cél a pontosságot előtérbe helyezni, így erre egy lehetséges megoldás, hogy minden hibafolthoz külön szkennelési (javítási) beállítást kell készíteni.

Síkkal közelítés

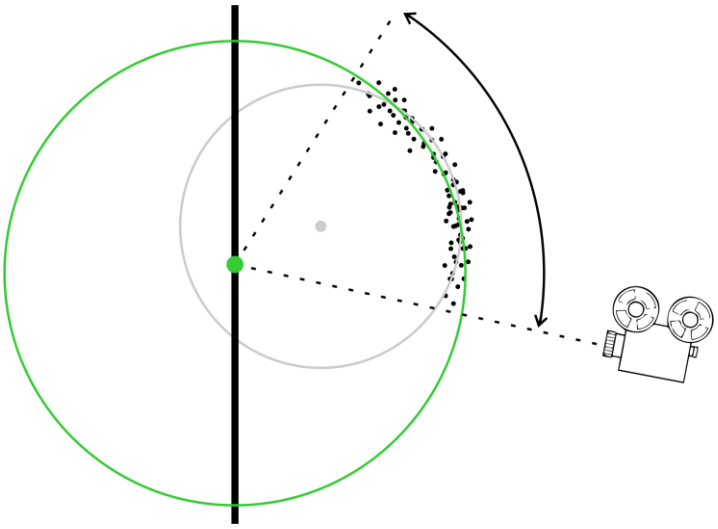
A hibaperem pontjainak (normalizált) normálvektorainak összegeként egy – a hibafoltot közelítő – sík normálvektorát lehet előállítani. A szkennert ez alapján egy sík szkenneléshez lehet felkonfigurálni. Ehhez kezdetben a kiszámított normálvektor alapján a szkenner megfelelő irányba forgatja, majd a hibaperemnek a közelítő síkra (a közelítő normálvektor és a pontok átlagaként kapott pontból előállítható ez a sík) vett merőleges vetületének a széléhez transzformálja a tárgyat. Ezt követően következik a transzlációs szkennelés, mely a normálvektorra merőleges irányba a hibaperemnek a közelítő síkra vett vetületének másik széléig tart.

Az adott két szabadságfokú szkenner esetében a közelítő sík normálvektorának a vízszintes vetületét kell venni, a transzláció iránya pedig adott.

Hengerrel közelítés

Amennyiben a közelítő sík túl durván közelít (például a négyzetes hiba (Heckbert & Garland, 1999) vagy hibaperem pontok síktól vett távolságainak szórása nagy), akkor hengeres szkennelésre is fel lehet konfigurálni az eszközt.

Az adott két szabadságfokú szkenner esetében ez úgy érhető el, hogy a hibaperem pontjainak vízszintes vetületére kell illeszteni egy olyan kört, melynek középpontja a szkenner transzlációs tengelyén fekszik. A szkennert kezdetben a transzlációs tengelyre illesztett kör középpontjába kell transzlálni, majd a legszélső ponthoz kell forgatni. A szkennelés ebben az esetben csak rotációs transzformációt végez, egészen az utolsó pontig.



40. ábra: Szkennerbeállítás hengeresen szkennelt javításhoz, vastag fekete vonal jelzi a transzlációs tengelyt, szürkével a pontok tényleges közelítő köre, zölddel pedig a pontok transzlációs tengely középpontú közelítő köre

A javítás folyamata

A korábban meghatározott hibacsoportoknak a pereme mentén kell javítani. Az egyes hibacsoportok javításához beállított szkennelésnek kétféle eredménye lehet. Az egyik eshetőség, hogy túl sok hibás részt tartalmaz, ezért a hozzá tartozó hibacsoport tovább nem javítható. A másik lehetőség, hogy a hibacsoport pereme mentén lehetséges javítani, ezáltal a hibás terület csökken, esetleg több kisebb része esik szét. Ezzel a két lehetőséggel eldönthető tehát, hogy érdemes-e tovább szkennelni, vagy már korábban beállított javító szkenneléssel sem sikerült a javítás, tehát az adott rész nem javítható tovább az adott hardverrel.

Ebben a lépésben fontos észben tartani, hogy a korábban generált mesh zárt, és ez a továbbiakban is megkövetelendő.

Amennyiben a javítás valamilyen okból nem sikerül az adott hibacsoportnál, ott akkor is megmarad a már korábban hibásnak ítélt mesh. Ez minden esetben biztosítani fogja a zártságot, de ezen a területen pontosabb szkennelésre nincs lehetőség.

Amennyiben sikerül valamilyen részt javítani, úgy a hibás terület mérete csökkenthető. Ehhez három dolgot szükséges megtenni:

1. A helyes mesh kiegészítendő a szkennelt javító mesh-sel. Ehhez egy összevarrási háromszögelést kell végezni, mely a helyes mesh és a szkennelt javító mesh között meghatározza a háromszögeket.
2. Ezt követően a hibaperem módosítandó:
    - Amennyiben a hibaperem egyetlen összefüggő helyen lett javítva, akkor a javított peremdarab javított része helyettesítendő a javító mesh megfelelő csúcspontsorozatával.
    - A másik eshetőség, hogy több részre esik szét a hibacsoport. Ekkor szükség van az újabb hibaperemek meghatározására.
3. Végül pedig a hibafoltok meghatározandók. Ez minden esetben azt jelenti, hogy a javítás után is legyen egy alternatív mesh a hibacsoport befoltozására (arra az esetre, ha a fennmaradó hibacsoportokat nem sikerül tovább javítani). Ezek az alternatív hibafoltok kialakíthatók a javítás előtti hibafoltból (annak egy részeként).

Összefoglalás

Az eddig elért eredmények nagyon jól működnek, azonban még számos részt lehet automatizálni. A hibás háromszögeket meghatározó résznél például matematikai statisztikával minden modell esetében ki lehet számítani, hogy pontosan milyen értékek fölött ne legyen elfogadva egy adott háromszög.

A kiindulási henger függőleges illetve vízszintes felosztásának sűrűségét az első szkennelés után dinamikusan lehetne kialakítani, hogy az objektum bizonyos részeinél sűrűbb, máshol pedig ritkább legyen.

A hibás háromszögek azonosításánál felhasznált metrikákat pontosítani lehetne a henger középpontjától való távolság figyelembevételével (pl. terület, kerület metrikák).

Az algoritmus javítási részét még szükséges részletesen kidolgozni, hogy a folyamat teljes legyen, és egy szkennelés az elejétől a végéig automatikusan végighaladhasson. Szintén a javításoknál kell lehetőséget biztosítani a pontos és a gyors szkennelések közötti választásra, hiszen ez kulcsfontosságú lehet az alkalmazási területeknél.

Fontos továbbá, hogy az algoritmusokat implementálásuk után a szkenner projektjébe is integrálni kell, illetve grafikus felületet is készíteni kell a megfelelő módok és paraméterek beállításához.

Fel lehet készíteni az eljárást olyan szkennerhez is, mely több lézerrel vagy több kamerával tud működni. Ezek esetében elsősorban a hibajavítás fázisban szükséges megfontolni a lépéseket, hiszen adott esetben párhuzamosíthatóak is lehetnek a szkennelések.

Különböző interpolációs módszerek segítségével a szkennelés után lehet finomítani azokon a területeken, ahova a szkenner nem látott be, vagy a textúra miatt nem volt szkennelhető az objektum. Ezekhez nem csupán a hibaperemeket, hanem a körülöttük levő háromszögeket is fel kellhet használni.

Szintén szkennelés utáni feldolgozási lépésnek egy mesh optimalizáló algoritmust is érdemes készíteni. Ez például a pontossági kritériumokat figyelembe véve egyszerűsítheti a háromszöghálót, hogy kevesebb helyet foglaljon.

Végül pedig 3D nyomtatóra exportálhatóvá kell tenni az adatokat, hogy a szkennelt objektum végül többszörösíthető is legyen.

Köszönetnyilvánítás

Ezúton szeretném megköszönni konzulensemnek, Dr. Kovács Tibornak a segítőkészségét és elhivatottságát, illetve a szkennerhasználat lehetőségét, mely nagyban megkönnyítette a munkámat.

Az emberi erőforrások minisztériuma únkp-16-1-i. kódszámú új nemzeti kiválóság programjának támogatásával készült.

Ábrajegyzék

Irodalomjegyzék

Attene, M., & Falcidieno, B. (2006). ReMESH: An Interactive Environment to Edit and Repair Triangle Meshes. *IEEE International Conference on Shape Modeling and Applications*. doi:10.1109/SMI.2006.29

Attene, M., Campen, M., & Kobbelt, L. (2013). Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)*, 1-33, Volume 45, Issue 2, Article No. 15. doi:10.1145/2431211.2431214

Barequet, G., & Sharir, M. (1995). Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 207-229, Volume 12, Issue 2. doi:10.1016/0167-8396(94)00011-G

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., & Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 349-359, Volume 5, Issue 4. doi:10.1109/2945.817351

Blinn, J. F. (1982). A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 235-256, Volume 1, Issue 3.

Botsch, M., Steinberg, S., Bischoff, S., & Kobbelt, L. (2002). OpenMesh - a generic and efficient polygon mesh data structure. *OpenSG Symposium*.

Dey, T. K. (2006). *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis.* Cambridge Monographs on Applied and Computational Math.

Edelsbrunner, H., & Mücke, E. P. (1994). Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 43-72, Volume 13, Issue 1. doi:10.1145/174462.156635

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 209-226, Volume 3, Issue 3.

Goshtasby, A. A. (2004). A weighted mean approach to smooth parametric representation of polygon meshes. *The Visual Computer*, 344-359, Volume 20, Issue 5. doi:10.1007/s00371-004-0247-1

Heckbert, P. S., & Garland, M. (1999). Optimal triangulation and quadric-based surface simplification. *Computational Geometry*, 49-65, Volume 14, Issues 1-3.

Jackowski, M., Satter, M., & Goshtasby, A. (2003). Approximating digital 3D shapes by rational Gaussian surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 56-69, Volume 9, Issue 1.

Jia, Y., Ni, X., Lorimer, E., Mullan, M., Whitaker, R., & Hart, J. C. (2010). RBF Dipole Surface Evolution. *Shape Modeling International Conference*, 143-150. doi:10.1109/SMI.2010.41

Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson Surface Reconstruction. *Eurographics Symposium on Geometry Processing*, 61-70.

Khan, I. R., Okuda, M., & Takahushi, S.-i. (2004). Regular 3D mesh reconstruction based on cylindrical mapping. *IEEE International Conference on Multimedia and Expo*. doi:10.1109/ICME.2004.1394143

Kovács, T. (2009). Vonalkereső algoritmus vizsgálata zajos környezetben. 4-15, 95-109. Budapest, Magyarország.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *International Conference on Computer Vision Theory and Application*, 331–340.

Rusu, R. B., & Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation*. doi:10.1109/ICRA.2011.5980567

Schmidt, R., & Simari, P. (2012). Consensus meshing. *Computers & Graphics*, 488-497, Volume 36, Issue 5. doi:10.1016/j.cag.2012.03.030

Szirmay-Kalos, L., Antal, G., & Csonka, F. (2003). Háromdimenziós grafika, animáció és játékfejlesztés. Budapest: Computer Books.

Zagorchev, L., & Goshtasby, A. (2012). A Curvature-Adaptive Implicit Surface Reconstruction for Irregularly Spaced Points. *IEEE Transactions on Visualization and Computer Graphics*, 1460-1473, Volume 18, Issue 9. doi:10.1109/TVCG.2011.276

Zhao, W., Gao, S., & Lin, H. (2007). A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 987–997, Volume 23, Issue 12. doi:10.1007/s00371-007-0167-y

Függelék

Mérési adatok a 31. ábrához

	Keresési idő (s)			
Mélység	71e háromszög	716e háromszög	7M háromszög	21M háromszög
1	0	0	0	0,001
2	0	0,001	0,001	0,001
3	0,001	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0,001	0	0,001	0,001
7	0,001	0	0,001	0
8	0,001	0,001	0,001	0,001
9	0,001	0,001	0,001	0,001
10	0,001	0,001	0,001	0,002
11	0,002	0,002	0,001	0,002
12	0,002	0,002	0,002	0,002
13	0,003	0,003	0,003	0,002
14	0,003	0,003	0,003	0,009
15	0,004	0,004	0,004	0,006
16	0,01	0,004	0,004	0,007
17	0,014	0,005	0,006	0,009
18	0,017	0,006	0,005	0,011
19	0,018	0,007	0,006	0,007
20	0,021	0,007	0,006	0,008
21	0,023	0,007	0,007	0,012
22	0,026	0,008	0,01	0,013
23	0,028	0,009	0,009	0,014
24	0,031	0,01	0,01	0,013
25	0,037	0,018	0,012	0,016
26	0,039	0,022	0,013	0,016
27	0,042	0,024	0,015	0,015
28	0,045	0,027	0,015	0,016
29	0,049	0,03	0,017	0,017
30	0,049	0,034	0,018	0,018
31	0,06	0,035	0,019	0,019
32	0,056	0,04	0,021	0,02
33	0,057	0,044	0,022	0,025
34	0,06	0,048	0,024	0,023
35	0,063	0,05	0,025	0,024

36	0,071	0,054	0,027	0,031
37	0,085	0,059	0,029	0,03
38	0,075	0,064	0,028	0,03
39	0,072	0,062	0,032	0,033
40	0,078	0,063	0,034	0,032
41	0,081	0,066	0,038	0,035
42	0,099	0,071	0,035	0,038
43	0,102	0,073	0,041	0,039
44	0,093	0,076	0,041	0,041
45	0,092	0,079	0,041	0,045
46	0,096	0,083	0,044	0,044
47	0,099	0,101	0,046	0,048
48	0,105	0,091	0,048	0,05
49	0,11	0,093	0,051	0,056
50	0,108	0,097	0,051	0,053
51	0,123	0,101	0,055	0,058
52	0,124	0,102	0,059	0,059
53	0,124	0,107	0,061	0,062
54	0,124	0,141	0,061	0,065
55	0,144	0,122	0,065	0,07
56	0,181	0,118	0,066	0,07
57	0,145	0,118	0,069	0,072
58	0,149	0,123	0,072	0,078
59	0,162	0,126	0,077	0,1
60	0,225	0,13	0,079	0,096
61	0,191	0,137	0,079	0,087
62	0,174	0,14	0,083	0,105
63	0,198	0,144	0,088	0,119
64	0,18	0,148	0,089	0,092
65	0,166	0,151	0,092	0,096
66	0,191	0,167	0,096	0,101
67	0,193	0,153	0,1	0,106
68	0,198	0,159	0,101	0,106
69	0,175	0,16	0,106	0,112
70	0,198	0,169	0,109	0,113
71	0,184	0,2	0,111	0,161
72	0,195	0,177	0,115	0,134
73	0,21	0,197	0,121	0,126
74	0,209	0,188	0,122	0,129
75	0,206	0,205	0,125	0,135
76	0,198	0,197	0,129	0,176
77	0,211	0,208	0,145	0,152
78	0,232	0,224	0,147	0,151
79	0,257	0,203	0,159	0,15

80	0,251	0,22	0,152	0,158
81	0,221	0,212	0,151	0,158
82	0,222	0,221	0,159	0,172
83	0,222	0,217	0,162	0,171
84	0,227	0,219	0,171	0,169
85	0,232	0,229	0,178	0,194
86	0,237	0,228	0,194	0,183
87	0,238	0,242	0,184	0,188
88	0,244	0,235	0,194	0,196
89	0,249	0,252	0,196	0,197
90	0,257	0,248	0,2	0,206
91	0,26	0,259	0,199	0,207
92	0,28	0,256	0,206	0,212
93	0,273	0,265	0,224	0,219
94	0,274	0,266	0,222	0,231
95	0,29	0,274	0,215	0,233
96	0,287	0,277	0,221	0,237
97	0,292	0,293	0,246	0,248
98	0,298	0,287	0,24	0,244
99	0,303	0,296	0,241	0,254
100	0,312	0,289	0,253	0,254