

Óbudai Egyetem
Neumann János Informatikai Kar
Alkalmazott Informatikai Intézet



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

**NAGY FELBONTÁSÚ KÉPPEL
TEXTÚRÁZOTT 3D SZKENNELÉS RGBD
SZENZOROKKAL**

Szerzők: **Könnyű Zsolt**
mérnök informatikus BSc. szak, IV. évf.
Sütő Balázs
mérnök informatikus BSc. szak, IV. évf.

Konzulensek: **Dr. Kozlovsky Miklós**
egyetemi docens
Dr. Vámosy Zoltán
egyetemi docens

Budapest, 2014.

Tartalomjegyzék

1.	Bevezetés	5
2.	3D-s érzékelők és szenzorok.....	6
2.1.	RGBD kamerák	6
2.1.1.	Primesense.....	6
2.1.2.	Asus Xtion PRO	7
2.1.3.	Asus Xtion PRO live	8
2.1.4.	Kinect for Windows	8
2.1.5.	Összehasonlítás	10
2.2.	Alternatív 3D digitalizálási megoldások	12
2.2.1.	Két kamera	12
2.2.2.	Lézeres	13
2.2.3.	Strukturált fény.....	13
3.	Térbeli objektumok digitális ábrázolásának módszerei és adatszerkezetei	14
3.1.	A 3D modellezés	14
3.2.	3D modellek ábrázolási típusai.....	14
3.2.1.	Héjmodell	14
3.2.2.	Tömör modell.....	15
3.3.	3D modellek ábrázolási módszerei.....	15
3.3.1.	3D poligonháló modellezés	15
3.3.2.	NURBS modellezés.....	15
3.3.3.	Pontfelhő modellezés	16
3.4.	3D szkennelés	16
3.4.1.	A pontfelhők generálása RGBD adatokból	17
3.4.2.	A pontfelhő rendezése és tárolása	18
4.	Adatkeretek illesztése egységes modellé.....	19
4.1.	Jellemző pontokat kereső algoritmusok	19
4.2.	Illesztő algoritmus	20
5.	3D rekonstrukció.....	22
5.1.	Algoritmusok	22
5.1.1.	Powercrust	22
5.1.2.	Eigencrust.....	23
5.1.3.	On Fast Surface Reconstruction Methods for Large and Noisy Point Clouds...	24
5.1.4.	Marching cubes algoritmus	24
5.1.5.	Poisson felszín rekonstrukció.....	24

5.1.6.	Radiális bázisfüggvényeken alapuló rekonstrukció	24
5.1.7.	A Volumetric Method for Building Complex Models from Range Images	25
6.	3D modellek 2D leképezése és megjelenítése	26
6.1.	2D renderelés	26
6.2.	Térfogati sugárkövetés	27
6.3.	Textúra leképezés	27
7.	Az irodalomkutatás összegzése.....	28
8.	Rendszerterv	29
8.1.	A rendszerrel szemben támasztott követelmények.....	29
8.2.	A rendszer moduljai.....	30
8.3.	Interfész modulok	31
8.4.	A rendszer osztályai és azok kapcsolatai.....	32
8.4.1.	Keretrendszer interfészek	32
8.4.2.	Adatkeretek kezelése és integrációja.....	33
8.4.3.	Metaadat-osztályok	34
8.4.4.	Segítőosztályok	35
8.5.	A szkennelés folyamata	36
8.5.1.	A szkennelés lépései a Fusion keretrendszer használata esetén.....	36
8.5.2.	A szkennelés lépései a Point Cloud Library használata esetén	37
8.6.	Térfogatmodell létrehozása	37
8.7.	Kamerakövetés	37
8.8.	ICP (Iterative Closest Point) algoritmus.....	39
8.9.	Színes szkennelési módszerek	39
8.9.1.	Vertex színek használata	39
8.9.2.	UV leképezés használata	40
9.	A felhasznált és támogatott hardverek.....	42
9.1.	Forgózsámoly és vezérlés	42
9.2.	Arduino Uno	43
9.1.	Firmware és szoftver implementáció.....	44
10.	Exportált fájl formátumok	45
10.1.	PLY (Polygon File Format).....	45
10.2.	OBJ (3D Object Format)	45
10.3.	STL (STereoLithography)	45
11.	Tesztesetek.....	46
12.	Eredmények értékelése	48

13.	Konklúzió.....	49
14.	Mellékletek	50
15.	Irodalomjegyzék	55
16.	Ábrajegyzék	58

1. Bevezetés

A dolgozat témája egy háromdimenziós színes szkennerek alkalmazás megtervezése és elkészítése, amely felhasználható objektumok digitalizálására és a folyamat során elkészített 3D modellek megjelenítésére, valamint exportálására a népszerűbb 3D modelleket leíró fájlformátumokba. Ez a korábbi TDK dolgozatunk továbbfejlesztése.

A cél a modellezők munkájának megkönnyítése a térbeli objektumok (tárgyak, személyek) minél pontosabb és gyorsabb digitalizálásának, megjelenítésének, valamint exportálásának lehetővé tételével (lásd **Hiba! A hivatkozási forrás nem található.** ábra). Fontos kikötés, hogy a beszkennelt modelleket a prototípusgyártáshoz és az egyedi 3D-nyomtatott termékek elkészítéséhez is lehessen használni. Az alkalmazás támogatja a népszerűbb 3D modell fájlformátumokat (OBJ, STL, PLY), valamint minimalizálja a későbbi felhasználás előtt szükséges modellezői utómunkát.

A szkennelést jelenleg RGBD szenzorokkal valósítja meg, mivel ezek manapság már megfizethetőek, a gyakorlatban jól alkalmazhatóak, és a technológiának köszönhetően a 3D rekonstrukcióhoz megfelelő minőségű szín és mélység adatokat biztosítanak

Az automatikus rögzítés elősegítése érdekében a pozicionálás számítógépről vezérelhető, melyhez külön opcionális hardvereket is felhasználhatóvá tesz. A szoftver jelenleg két különböző verziójú elektromos forgó zsámolyt támogat (lásd 2. ábra), amivel egy statikus szenzor használatával is elérhető a 360°-os látószög.

A szoftverrel szemben támasztott elvárás a könnyű kezelhetőség, hogy ne kelljen a felhasználónak mérnöki ismeretekkel rendelkeznie a program használatához. Ennek szellemében, jól átlátható grafikus felületet is biztosít.



1. ábra Egy modell, amit a saját szoftverünkkel készítettünk egy mankós emberről.



2. ábra Egy ember szkennelése

2. 3D-s érzékelők és szenzorok

2.1. RGBD kamerák

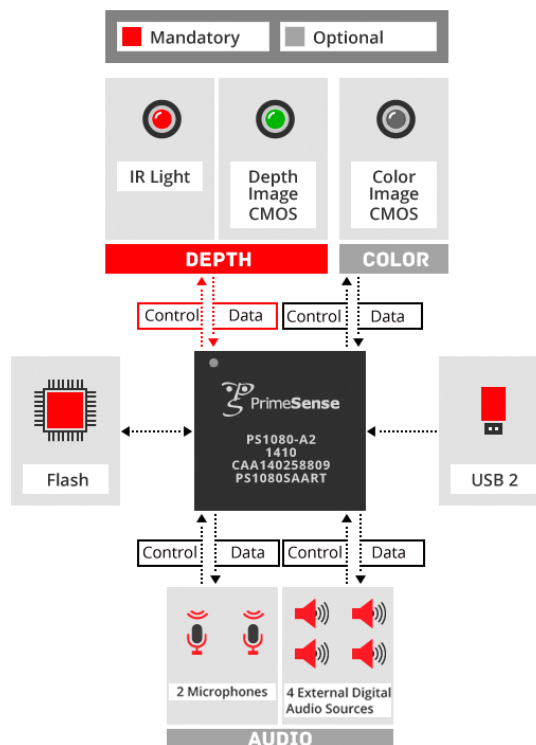
2.1.1. Primesense



3. ábra PrimeSense szenzor [1]

A PrimeSense technológiája [1] elérhetővé teszi a digitális eszközöknek az egyszerű háromdimenziós képfeldolgozást. Lefordítja és szinkronizálja a jelenetből kinyert mélység- és a szín adatfolyamokat. Ezeket az adathalmazokat felhasználhatjuk objektumok osztályozására, mint például a falak elhelyezkedésének érzékelése, vagy a testrészek felületének detektálása.

A “PrimeSense Light Coding™” egy „System on a Chip (SoC)” szabadalmaztatott technológiai megoldás, ami 3D mélység érzékelést valósít meg (lásd 4. ábra). Kódolt infravörös fényt bocsájt ki, amely láthatatlan az emberi szem számára. Ezt a fényt egy infra szűrővel ellátott szabványos CMOS képérzékelővel érzékeli, ebből állítja vissza a jelenet mélységi adatait. Ez az a folyamat, amely lehetővé teszi a mélységi adatok begyűjtését, és amittől PrimeSense megoldás annyira pontos.



4. ábra PrimeSense SoC kapcsolatai [2]

PrimeSense System on a Chip (SoC)

CMOS szenzort használ a PrimeSense SoC [2] Carmine (PS1080) és a következő generációs SoC Capri (PS1200) is, amik kifinomult párhuzamos algoritmusokkal számítják ki a VGA méretű kép mélységi adatait. A SoC USB 2.0-es interfészen keresztül kommunikál az adatokat fogadó géppel, aminek nincs szüksége nagy CPU teljesítményre, miután a számításigényes mélységi információkat a SoC saját maga állítja elő.

Carmine (PS1080)

Carmine egy több érzékelős rendszer, amely szinkronizált mélységi-, színes kép és hangadatokat szolgáltat. Ez a leggyakrabban használt 3D SoC a világon, több mint 20 millió felhasználóval.

Capri (PS1200)

Capri – a következő generációs SoC mélység számításra. Javított hatékonyságú algoritmusok, multi-modális 3D érzékelő technológiák felhasználásával készül, amelyek lehetővé teszik a kiváló felbontási teljesítményt és a kisebb méretet a csökkentett költségek mellett.

	2 m távolságból	0,4 m távolságból
Térbeli (X / Y) felbontása	3,4 mm	0,9 mm
Mélység (Z) felbontása	12 mm	10 mm

1. táblázat Feltételezett felbontás értékek a PrimeSense referencia modell alapján

A technológiát a PrimeSense fejlesztette ki és tette licenszelhetővé. A szenzorok az OpenNI nyílt forráskódú keretrendszerét használják, PrimeSense NiTE néven, ami az OpenNI.org weboldaltól érhető el.

2.1.2. Asus Xtion PRO



5. ábra Asus Xtion Pro [3]

Az Xtion PRO [3] az infravörös- és az adaptív mélység érzékelő technológiát használja, hogy a felhasználók valós idejű testmozgását tudja figyelni. Több algoritmust is kínál a fejlesztők számára, amivel a fejlesztés folyamatát igyekszik egyszerűsíteni.

2.1.3. Asus Xtion PRO live



6. ábra ASUS Xtion Pro Live [4]

A legnagyobb különbség [4] az Asus Xtion PRO-hoz képest, hogy itt már megjelent a második kamera, amivel az RGB adatokat is rögzíteni tudjuk, ezáltal a szkennelt modellre szín információkat tudunk rögzíteni.

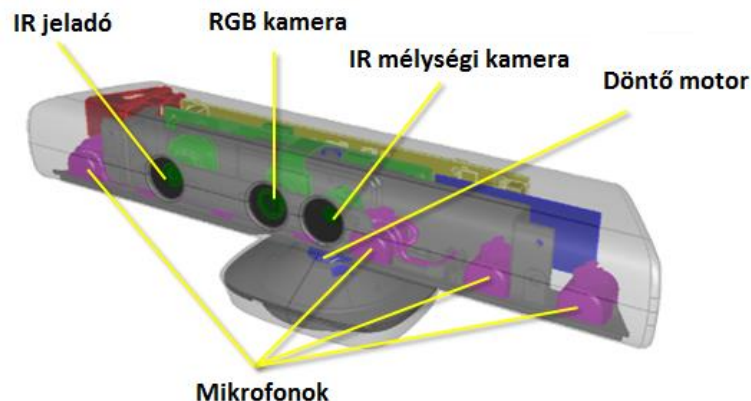
2.1.4. Kinect for Windows



7. ábra Kinect for Windows [5]

Az alábbi szenzorok kaptak benne helyet [5] [6]:

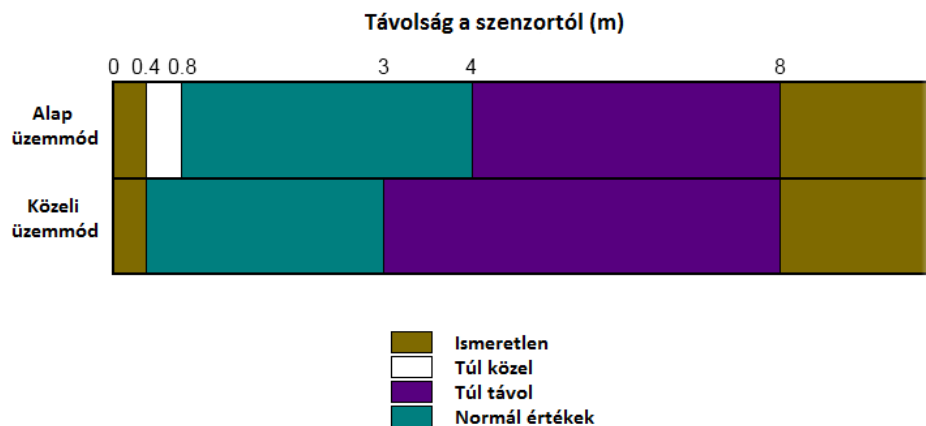
- RGB kamera 1280×960-as felbontással, ami a színes képet szolgáltatja
- Egy infravörös (IR) adó és egy IR mélységi szenzor. Az adó kivetít egy „hálót” infravörös tartományban, amit a tárgyakról visszatükröződve a mélységi szenzor beolvas. A visszavert pontokat átalakítják mélységi információvá, ami méri a távolságot egy tárgy és az érzékelő között. Ebből származik a mélységi információnk.
- 3-tengelyű gyorsulásmérő, amit 2 G-s tartományra állítottak, ahol a G a gravitációs gyorsulás. Ez az érzékelő a Kinect aktuális tájolását határozza meg.



8. ábra Kinect szerkezeti felépítése [5]

- Apróbb pozicionálási segítségként a döntőmotor vezérlését Implementáltuk a programunkba. -27 és +27° között lehet vezérelni, de ez nem befolyásolja a szkennelési folyamatot.
- Négy darab mikrofon számunkra lényegtelen, így ezek részletezésével nem foglalkozunk.

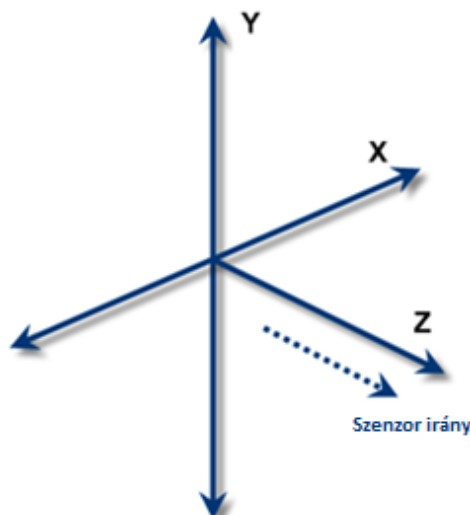
A „túl közeli” vagy „túl messzi” érték azt jelenti, hogy egy tárgyat érzékelt ugyan az eszköz, de nem tudott megfelelően pontos eredményt adni annak távolságáról.



9. ábra Kinect érzékelési tartományok [5]

Megbízható távolsági adatokat 80 cm-től 4 m-es távolságig szolgáltat az eszköz, amit a „közeli mód” bekapcsolásával 40 cm-re csökkenthetünk. Ekkor azonban már csak 3 m-ig „lát el” az érzékelő, valamint a feldolgozás szélessége 20%-os visszaesést mutat.

Jobb kezes koordináta rendszerben kapjuk meg az adatokat, ahol a Z tengely pozitív iránya amerre a Kinect néz, az Y tengely a függőleges irány, az X tengely pozitív iránya pedig balra (lásd 10. ábra).



10. ábra Kinect koordináta rendszere [5]

2.1.5. Összehasonlítás

Az eszközök összehasonlíthatósága [7] érdekében mellékeljük a részletes specifikációkat összefoglaló táblázatot (lásd 2. táblázat).

	Asus Xtion PRO	Asus Xtion PRO Live / PrimeSense Carmine	Microsoft Kinect	Microsoft Kinect for Windows
Áramfogyasztás	2,5 W - USB	2,5 W - USB	USB + 12 V adapter	USB + 12 V adapter
Érzékelő távolság	0,8 – 3,5 m	0,8 – 3,5 m	0,8 – 4 m	near mode: 4 - 3 m normal: 0,8 - 4 m
Látószög	vízszintes: 58°, függőleges: 45°, átlós: 70°	vízszintes: 58°, függőleges: 45°, átlós: 70°	vízszintes: 57°, függőleges: 43°	vízszintes: 57°, függőleges: 43°
Függőleges dőléstartomány	-	-	±27°	±27°
Szenzor típusa	mélyégi	mélyégi, RGB, Mikrofon × 2	mélyégi, RGB, Mikrofon × 4	mélyégi, RGB, Mikrofon × 4
Mélyégi képméret (IR)	VGA (640×480): 30 fps, QVGA (320×240): 60 fps	VGA (640×480): 30 fps, QVGA (320×240): 60 fps	QVGA (320×240): 30 fps	VGA (640×480): 30 fps
Színes képméret	-	SXGA (1280×1024): 30 fps	SXGA (1280×1024): 30 fps	SXGA (1280×1024): 30 fps
Platform	Intel X86 & AMD	Intel X86 & AMD & AMR	Intel X86 & AMD	Intel X86 & AMD
OS támogatás	Win 32/64: XP/Vista/7	Win 32/64: XP/Vista/7	Win 32/64: XP/Vista/7/8	Win 32/64: XP/Vista/7/8
	Linux Ubuntu 10.10: X86, 32/64bit	Linux Ubuntu 10.10: X86, 32/64bit	Linux 32/64bit	Linux 32/64bit
	Android	Android	-	-
Csatoló	USB 2.0	USB 2.0	USB 2.0	USB 2.0
Szoftver	Fejlesztői eszközkészlet (OPEN NI SDK mellékelve)	Fejlesztői eszközkészlet (OPEN NI SDK mellékelve)	Kinect SDK	Kinect SDK

Programozási nyelv támogatás	C++/C# (Windows), C++(Linux), JAVA	C++/C# (Windows), C++(Linux), JAVA	C++ / C#	C++ / C#
Működési környezet	beltéri	beltéri	beltéri	beltéri

2. táblázat RGBD szenzorok specifikációi

A PrimeSense Carmine 1,08 szenzor lényegében megegyezik az ASUS Xtion Live-al, csak a márkanév különbözik. Az ASUS Xtion Live-ot vagy a PrimeSense Carmine-t ajánlják a fejlesztőknek, mert magukban foglalják a színérzékelőt is. Az érzékelők a színes képet jelenleg nem használják a jellemző pontok keresésére, de folyamatban van ennek fejlesztése, hogy az is segítsen a rendszer működésében. A PrimeSense Carmine-nek jobb az USB 3.0 kompatibilitása, mint az eredeti ASUS Xtion Live-é, bár az Asusnak van egy firmware frissítése, ami egy szintre hozza őket.

Megjegyzés: Az ASUS Xtion és az ASUS Xtion Live is kapható PRO módosítással (a fejlesztőknek). Bár a hardver ugyanaz, az egyetlen különbség a mellékelt szoftver, amivel nem kell használni az IPI Motion Capture rendszert.

A Microsoft Kinect és ASUS Xtion (Live) / PrimeSense Carmine szenzorok alapját ugyanaz a PrimeSense infravörös technológia alkotja. Tehát az alapvető jellemzőik hasonlóak, de vannak bizonyos eltérések, amiket figyelembe kell venni.

A Kinect a PrimeSense technológiáját használja, így nagy a hasonlóság köztük, főleg a mélységi térkép felbontásában. A PrimeSense Sensor sokkal kisebb és USB meghajtású. Ezen kívül, mivel másik piaci rést céloznak meg vele, ezért speciális igényeknek kell megfelelnie. Például az érzékelők száma és típusa is tesztre szabható, azaz az RGB és a mélységi érzékelőt is le lehet cserélni a beágyazott moduljuknak köszönhetően, ezáltal megrendelő igényeire lehet szabni a szenzort. PrimeSense szinte minden platformmal kompatibilis.

Eszköz	Előnyök	Hátrányok
Microsoft Kinect	Jó minőségű, könnyen használható driver Stabil működés különböző hardver elemekkel Döntő motorral lehet távvezérléssel pozícionálni az eszközt	Nagyobb méretű (12 "× 3" × 2.5 ") Nagyobb tömeg (1,3 kg) Szükséges a 220 V-ról üzemelő tápegység Magasabb interferencia keletkezik több Kinect együttes használat közben Alacsony RGB képminőség kivéve MS Kinect for Windows

ASUS Xtion / PrimeSense Carmine	Kisebb méret (7 "× 2" × 1.5 ")	Kevésbé népszerű eszköz
	Kisebb súlyú (0,23 kg)	Alacsony driver minőség
	Nem igényel külső tápegységet az USB-t leszámítva	Nem működik néhány USB vezérlővel (különösen a USB 3.0-t nem támogatja)
	Kisebb interferencia egy másik ASUS Xtion / PrimeSense Carmine érzékelővel	Nincs motor, csak kézi pozicionálás lehetséges
	Jobb RGB képminőség	

3. táblázat Eszközök előnyei, hátrányai

2.2. Alternatív 3D digitalizálási megoldások

Passzív és aktív megközelítés létezik. A passzív a környezetből és a tárgyról visszaverődő információk alapján számol, míg az aktív kibocsájt magából valamilyen jelet, általában adott hullámhosszú fényt, és annak a torzulásából számítja a távolságot.

2.2.1. Két kamera

A sztereo kamera [8] működésének alapelve az emberi szem mélységérzékelésének elvével megegyezik. A két szemgolyó egymástól fix távolságra helyezkedik el. Amire fókuszálunk, azt ezáltal két különböző szögből is látjuk. A két szem képe közötti különbség alapján az agy számolja ki a távolságot. Ennek köszönhetően érzékeljük a mélységet. [9]

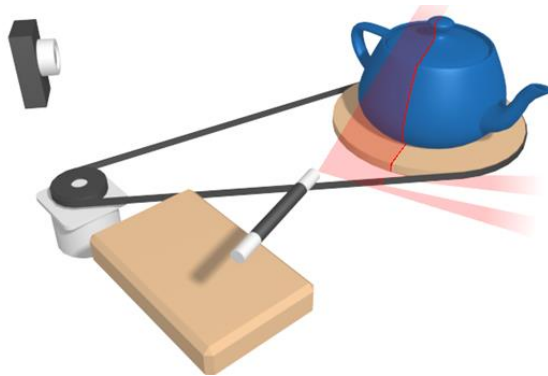
Robosztus, lokális jellemző pontokat keresünk a képen. Jellemzően él, sarok detektálással, például a közkezdvelt a Harris féle algoritmus. Ezeket a pontokat a másik kamera képén is megkeressük, majd interpoláljuk a teljes felületet. Az epipoláris egyenesre [10] felfekvő pontjainkat háromszögeléssel kit tudjuk számolni, hogy a térben hol helyezkednek el. A párosítás során sok képjavító, közelítő számítást kell végeznünk, hogy a tükröződést, és egyéb hibákból kialakuló veszteségeinket minimalizálni tudjuk.

Ha nem jellemzőkkel szeretnénk párosítani: Mass-Poggio (1979 → fekete-fehér) Összetartozás térképet készítünk. Sorokat alkotunk (1D), majd iteratíván számoljuk az összetartozás tömböt

Korreláció alapú sztereo: A mélység a jellemzőknél kerül kiszámolásra és a többi pixelnél interpolálás történik. Diszparitás térkép készül a mért korreláció alapján, majd a diszparitás értékéből a mélység is kiszámolható.

Energia alapú sztereo: A párosítás jóságát energiaként fogalmazzuk meg, ahol az “adattag” bünteti a rossz párosítást, és a “szomszédosság tag” a térbeli simaságot növeli.

2.2.2. Lézeres

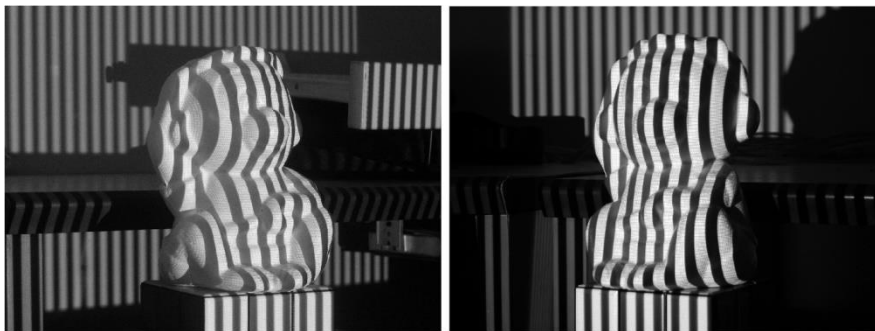


11. ábra Lézer szkennelés működése [11]

A lézerszkennelés elve is a szögeltérésre épül, csak nem két kamerával, hanem egy kamerával és egy lézervonal kivetítővel oldja meg a mélység-meghatározást a 3D szkennelés (lásd 11. ábra). Ha a kamerát és a lézervonal kivetítőt a lézer kivetítés síkjában helyeznénk el, akkor a kamera mindig egy egyenesnek látná a lézercsíkot, függetlenül attól, hogy milyen felületre vetítjük. Ha eltávolítjuk ebből a síkból a kamerát, akkor a felülettől függően törések és szakadások lesznek a kivetített lézercsíkjában. Adott szögeltérés esetén a lézercsík törései és szakadásai alapján egyértelműen visszaszámolhatóak az adott pontok térbeli helyzete. [9]

2.2.3. Strukturált fény

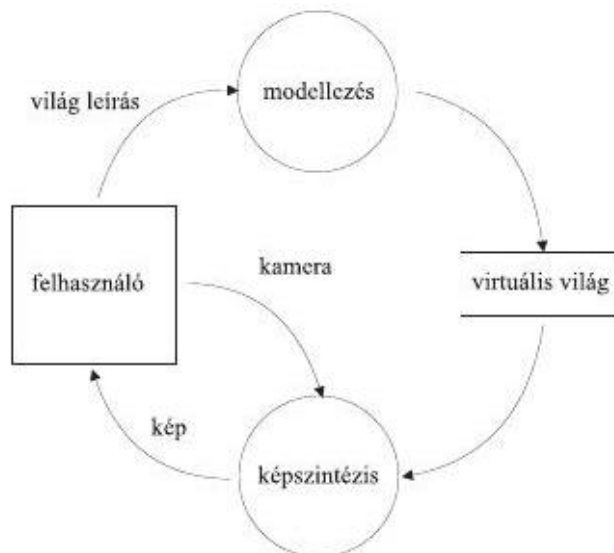
Az elve nagyon hasonlít a lézerrel történő digitalizáláshoz, csak itt nem csak egy csíkban lévő lézervonalat detektálunk és annak a mentén számolunk, hanem az általunk kivetített (általában projektorral) változtatható méretű és elhelyezkedésű csíkok deformálódása alapján lehet kiszámolni a távolságokat (lásd 12. ábra). Miután egyszerre több „vonalunk” látszódik a képen, ezért egyszerre több információt tudunk feldolgozni.



12. ábra Strukturált fényvel történő sztereó feldolgozás [12]

3. Térbeli objektumok digitális ábrázolásának módszerei és adatszerkezetei

3.1. A 3D modellezés



13. ábra A számítógépes grafika adatfolyam-modellje [13]

A háromdimenziós objektumok digitális ábrázolása 3D modellek segítségével történik. A folyamatot, aminek során a térbeli objektumok alapján egy személy vagy algoritmus matematikai modelleket készít, 3D modellezésnek nevezzük. A modellezés során a külvilágot leíró adatokból kiindulva alakítjuk ki az absztrakt virtuális világot, valamint a benne ábrázolni kívánt modellünket (lásd 13. ábra). Az így kapott 3D modelleket felhasználhatjuk különböző számításokhoz, azonban egy 2D kijelzőn teljes egészükben nem jeleníthetők meg. Megadott kameraállásokból 2D projekciókat hozunk létre a modelltől, amiket képek formájában már ki tudunk rajzolni a képernyőre. Ezt képszintézisnek vagy 2D renderelésnek nevezzük.

3.2. 3D modellek ábrázolási típusai

Attól függően, hogy mit várunk el a 3D modellünktől, több modell típust és ábrázolási módszert alkalmazhatunk. Az ábrázolási módszer kiválasztása során érdemes megvizsgálni, hogy milyen alakú testet szeretnénk modellezni, valamint ezzel a testtel milyen jellegű számításokat kívánunk végezni. A használt modell jellege határozza meg a számítási és tárolási lehetőségeinket.

3.2.1. Héjmodell

A héjmodellek csak az objektum felszínét ábrázolják, az objektum belsejében található dolgokkal nem foglalkoznak. Csúcsokból és az őket összekötő élekből épülnek fel. Az egyszerűségük miatt ezeket a modelleket alkalmazzák a számítógépes játékokban és a 3D hatásokat tartalmazó filmekben.

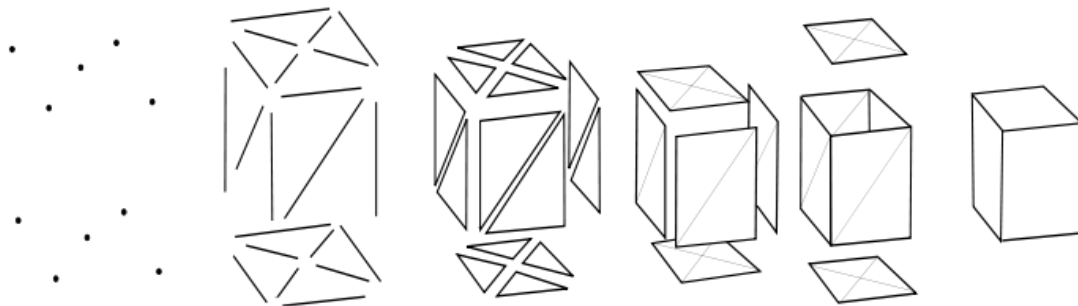
3.2.2. Tömör modell

A tömör modell az adott objektumot annak térfogatával és belső tartalmával együtt írja le. Ezt a modell típust elsősorban olyan helyeken használják, ahol az objektum pontos mértani ábrázolására kell fektetni a hangsúlyt, vagy komoly tudományos számításokat kell végezni az ábrázolt objektummal. Nehezebb létrehozni, és dolgozni vele, mint a hémmodellel.

3.3. 3D modellek ábrázolási módszerei

3.3.1. 3D poligonháló modellezés

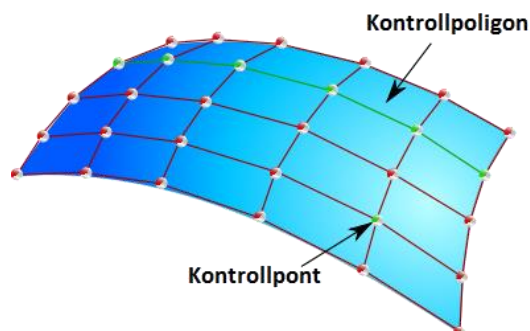
A 3D térben elhelyezett csúcsokból és az őket összekötő élekből felépülő hálós struktúra. A csúcsokat szokás vertexeknek is nevezni. A pontok és az élek együtt 2D sokszögeket, vagyis poligonokat alkotnak, melyek éleik mentén kapcsolódnak egymáshoz a térben egy összefüggő 3D felületet adva (lásd 14. ábra). A gyakorlatban leginkább háromszög vagy négyszög alapú hálókat használnak, amiket a megfelelő három-szögelő vagy négy-szögelő algoritmusok segítségével állítanak elő. Ez a módszer azért terjedt el, mert a használata a többi megoldáshoz képest egyszerű, a számítási igényei pedig viszonylag alacsonyak. Az egyszerűség ára az, hogy a görbéket ezzel a módszerrel nem tudjuk hatékonyan ábrázolni, mivel a sokszögekkel csak közelíteni tudjuk őket.



14. ábra A háromszög hálók felépítését bemutató ábra. A csúcsok és élek oldalakat alkotnak, az oldalak poligonokat, a poligonok felszíneket. Ezekből áll össze a 3D objektum. [14]

3.3.2. NURBS modellezés

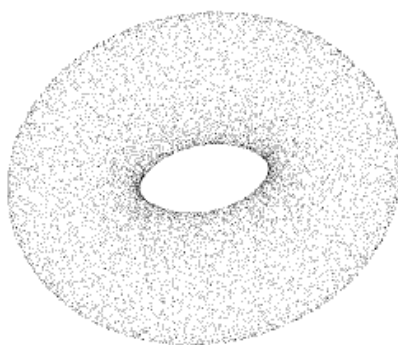
A Nem-egységes Racionális B-Szplájn (NURBS) modellezés a poligon hálóhoz hasonlóan elterjedt módszer. A modelleket görbék segítségével írja le, kontroll pontokat és kontroll poligonokat definiál. A transzformációk elvégzése során elég azokat a kontroll pontokra elvégezni. A NURBS modellek pontosabban írják le a térbeli testeket, mint a háló alapú modellek, itt nem jelentkeznek a térbeli görbék ábrázolási nehézségei. A görbék és felületek hatékonyabb ábrázolása azonban magával hozza a modell bonyolultságának és számítási igényének növekedését. Elsősorban CAD programoknál alkalmazzák őket, ahol nagyméretű, tervezési szempontból pontos modellekre van szükség.



15. ábra NURBS modell szerkezeti ábra [15]

3.3.3. Pontfelhő modellezés

A pontfelhő csúcsok halmaza egy háromdimenziós koordináta-rendszerben ábrázolva. Amennyiben eltároljuk a pontokhoz tartozó színadatokat is, minden csúcs rendelkezik egy vörös, egy zöld és egy kék színintenzitással, valamint x, y és z koordinátákkal, amelyek leírják az adott csúcs helyzetét a felhőhöz tartozó lokális koordináta-rendszerben. A színek tárolása természetesen nem kötelező, készíthetünk színadatok nélküli pontfelhőket is.



16. ábra Egy tórusz alakzat pontfelhő modellje [16]

3.4. 3D szkennelés

A 3D modelleket kézzel megalkotni időigényes, és általában nem költséghatékony megoldás. A bonyolultabb testek manuális ábrázolása során gyakran ütközhetünk megvalósíthatósági korlátokba. A 3D szkennelés erre kínál alternatívát. A művelet során a szkennerből vagy szkennerekből beolvasott mélységi- és színadatokból generálhatóak lokális pontfelhők, amelyek csúcsai az eredeti objektum felszínének pontjaiból állnak elő. Ezek a lokális felhők matematikai transzformációk és illesztő algoritmusok segítségével egyesíthetőek egy globális koordinátarendszerben ábrázolt felhővé, lehetővé téve az objektumok 360°-os feldolgozását. Mivel az így kapott felhő csak egy térbeli pontthalmaz, nem pedig egy összefüggő felszínnel rendelkező modell, ezért belátható, hogy a szkennelés az objektum felszínének tekintetében információvesztéssel jár. A pontfelhők felhasználási lehetőségei ebből következően erősen korlátozottak. Ha szükségünk van a modell eredeti felszínére, rekonstrukciót kell rajta végeznünk. A modelltől különböző rekonstrukciós algoritmusok segítségével állítható elő például egy sokszögháló modell, amely ideális esetben már rendelkezhet összefüggő felszínnel. Ez az ábrázolási mód már felhasználható érdemi számítógépes grafikák

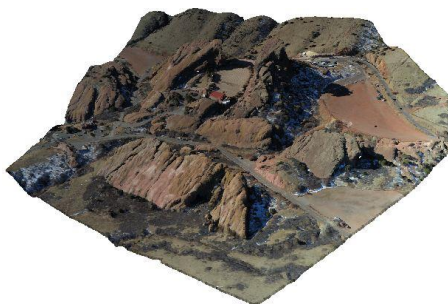
készítéséhez, fizikai számítások alapjául szolgáló modellek tervezéséhez vagy akár 3D nyomtatáshoz is.

3.4.1. A pontfelhők generálása RGBD adatokból

A pontfelhőket alkalmazó módszerek szerint minden képkockához, tartozni fog egy pontfelhő, ami a képkockában található képpontok a kamerához viszonyított helyzetét fogja reprezentálni egy, a képkockához tartozó lokális koordináta-rendszerben. A pontfelhő minden csúcsa egy ilyen rögzített képpont lesz.

Az átalakítás azért lehetséges, mert minden képkockát felfoghatunk két mátrixként is. Az első mátrix elemei a képkocka színadatait tartalmazzák. Ezt a mátrixot akár értelmezhetjük egy 2D RGB színes képként is. A másik mátrix elemei, a szenzor által előállított mélységi adatok, egy relatív távolságot határoznak meg a kamerához képest. Ez a nyers adat kameraként, meghajtó programonként és keretrendszerként változó lehet, de a legtöbb esetben valamilyen egész értéket jelent. Az első lépés ezért mindig a nyers mélységi adatok valamilyen normalizált formájú távolsággá alakítása kell, hogy legyen. Ez a gyakorlatban azt jelenti, hogy a kapott adatokat olyan formátumúra alakítjuk, amivel már tudunk számolni.

Ha előálltak a normalizált mélységi adatok, következhet a képfelhővé alakítás. Ehhez tulajdonképpen nem is feltétlenül szükséges a szín mátrix felhasználása, kialakíthatunk monokróm pontfelhőket is. Amennyiben úgy döntünk, hogy a színeket is tárolni szeretnénk, azokat a színadat mátrix fogja szolgáltatni. Ezek a színes képpontok megfeleltethetők egy 4 bájtól tárolt RGB színintenzitás tömbnek, ahol az adott színhez tartozó intenzitást általában egy-egy bájtól tároljuk (az utolsó bájt nincs használva). A színes rekonstrukciós módszerek alapja az, hogy a két mátrix egyes értékei egyértelműen megfeleltethetők egymásnak, minden 2D képponthoz tartozik egy mélységi adat, és fordítva. Egy adott pont helyzetét a koordináta-rendszerben a 2D kép mátrixként értelmezett képkocka az adott képponthoz tartozó sor és oszlop pozíciója alapján (x és y koordináták), valamint a hozzá tartozó mélységi adatból (z koordináta) lehet kiszámolni. Az objektum pontfelhő modelljének megépítése az első lépés annak érdekében, hogy az objektum eredeti felszínét visszanyerhessük a kiolvasott adatokból.



17. ábra Egy pontfelhőből generált kultéri térkép modell. [16]

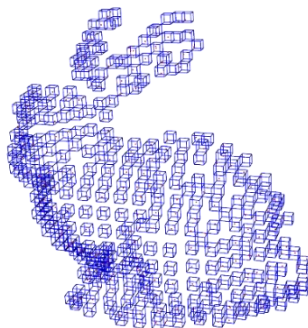
A képkockákból generált felhőket célszerű egyesíteni egy egységes globális pontfelhőbe, ha érdemi, 360°-os látószögű rekonstrukciót szeretnénk végezni. A pontfelhők egyesítéséből fogjuk kinyerni az eredeti jelenet lehető legkevesebb hibával rendelkező modelljét. Ehhez erősen javasolt valamilyen elő feldolgozó, hibajavító algoritmus használata. A pontfelhők

egyesítésével (regisztrálásával) bővebben a 3D modell rekonstrukciója fejezetben foglalkozunk.

3.4.2. A pontfelhő rendezése és tárolása

A legtöbb implementációban a pontfelhőket egy csúcsokból álló, tömörítetlen listaként kezelik, hogy ezzel megkönnyítsék és gyorsítsák a feldolgozást. A tömörítetlen lista alkalmazása azonban megnöveli a felhők tárolásához szükséges tárhely méretét. Ebből kifolyólag a képkockákból generált pontfelhőket külön-külön csak ideiglenesen tároljuk a memóriában, amíg megtörténik a beillesztés a globális felhőbe.

Ha mégis felmerül az igény a lokális pontfelhők tárolására, a modellek kis felbontásnál és mintavételezési sebességnél is jelentős mérete és darabszáma miatt célszerű valamilyen tömörítési eljárást alkalmazni. A legtöbb pontfelhőkkel dolgozó keretrendszer és program szerencsére biztosít erre valamilyen megoldást.



18. ábra A Point Cloud Library `pcl_octree` modulja által októlis fába rendezett pontfelhő. [17]

Ha valós időben végbemenő, folyamatos felszín rekonstrukciót szeretnénk végezni a pontfelhőn, célszerű lehet valahogy tárolni a még fel nem dolgozott változásokat, mert ezáltal egyszerűsíthető és gyorsítható a rekonstrukció folyamata. Bizonyos feldolgozó és osztályozó algoritmusok számára fontos, hogy egy bizonyos szempont szerint rendezett, jól strukturált pontfelhőt adjunk át nekik.

4. Adatkeretek illesztése egységes modellé

Az egymás után következő képkockáinkat szeretnénk egyetlen nagy pontfelhőben eltárolni. Ehhez szükségünk van bizonyos illesztő eljárásokra, amik megkeresik az azonos pontokat a képeken és azok elhelyezkedése alapján össze is tudják illeszteni a két képet. A 3. dimenzió belépésével még több számítási kapacitásra van szükségünk a feldolgozáshoz. Jellemzően az ilyen feladatokat párhuzamosítják, és GPU-ra optimalizálják. A kamerák lokális koordináta rendszereiből mátrix transzformációk által lehet átkonvertálni az adatokat a világ koordináta rendszerbe, hogy el tudjuk tárolni azt a számítógépünkön.

Egy szenzorral dolgozva elég az adatokat átranszformálni a világ koordináta rendszerbe, viszont ha több szenzort használunk, akkor azok egymáshoz viszonyított helyzetét is tudnunk kell. Több Kinect használata esetén figyelni kell az IR adók interferenciájára is, mert ha egy azonos területet próbálnak érzékelni, akkor a „túl sok pont” miatt nem tudja értelmezni a visszavert jelet.

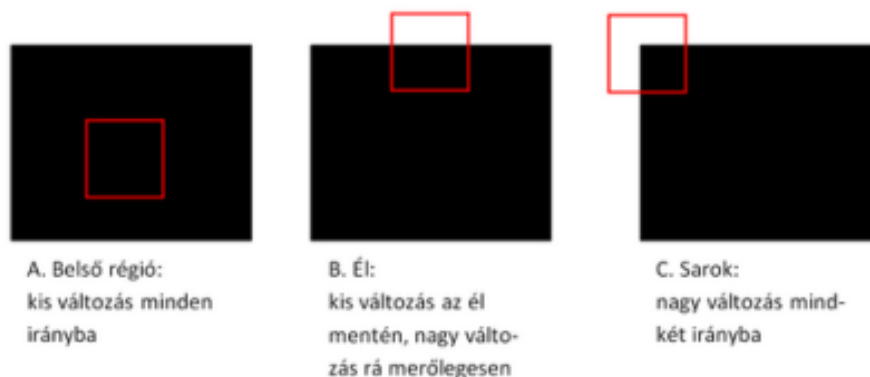
Ha statikus helyzetben van a szenzor és előtte forgatjuk a tárgyat, figyelni kell arra, hogy csak a tárgyon vagy annak kis környezetében keressük a pontokat, különben a modellünk feleslegesen nagyra nő, ezzel megnövelve a hibás működés valószínűségét. Dinamikus szenzor mozgatásnál a környezetet is bele lehet venni az érzékelési tartományba, mert az abban található a pontok az algoritmus robosztusságából kifolyólag nem rontják jelentős mértékben a feldolgozás pontosságát.

A konverziók során végzett kerekítések és simítások elengedhetetlenek a használható eredmény érdekében, de ezzel együtt a pontatlanság is nőhet.

4.1. Jellemző pontokat kereső algoritmusok

A jellemző pontokat kereső algoritmusok meghatározzák a képeken azokat a pontokat, ahol a gradiensek mértéke több irányból is meghalad egy küszöbértéket. Ezek a pontok az illesztő algoritmusok számára nélkülözhetetlenek.

Harris sarokdetektálás



19. ábra Harris detektálás [18]

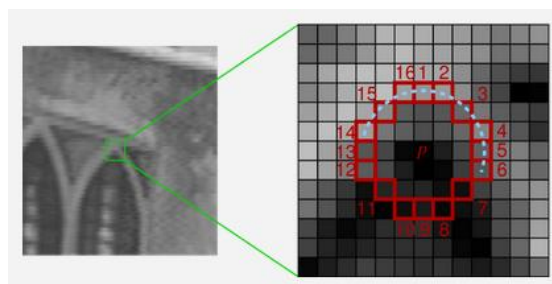
A Harris [19] sarokdetektálás átlagos intenzitást számol egy kis ablakban a kiemelt pont körül. Ha elmozdulunk, akkor az átlagos intenzitás változás értéke:

$$R = \sum (I(x+u, y+v) - I(x,y))^2$$

„A lokálisan eltoló képeket elsőfokú Taylor polinomokkal közelíti, ezzel visszavezeti a különbség négyzetösszegeket a gradiens kép egy függvényére. Egy ún. Harris mátrixot számítja ki, melynek sajátértékei a sajátvektorok irányába történő elmozdulás esetén jellemzik a kép megváltozását. A mátrix sajátvektorai szemléletesen a legnagyobb és legkisebb változás irányába mutatnak, a két változás mértéke pedig jellemezhető a sajátértékekkel. Ebből következik, hogy a Harris detektor rotáció invariáns, hiszen a sajátértékek nem függenek a kép orientációjától, csak a sajátvektorok. Sarokpont van tehát ott, ahol mindkét sajátérték nagy. Az algoritmus a sajátértékeket közelítő módszerrel számolja a sebesség érdekében.

A detektornak két elterjedt módosítása is van. A multi-scale Harris operátor skála térben számol, így különböző nagyságú sarkok megtalálására alkalmas, azaz invariáns a nagyításra.” [18]

FAST (Features from Accelerated Segment Test)



20. ábra Fast működése [20]

A FAST [20] kereső is egy pont körüli kör mentén vizsgál (pl.: 3 sugarú, 16 kerületű), ha ebből valahány (pl.: 9) eltér a pixelnél legalább egy küszöbvel magasabb értékkel, akkor az adott középpont egy jellemző pont. Egyszerre csak egy irányú eltérést érdemes vizsgálni.

4.2. Illesztő algoritmus

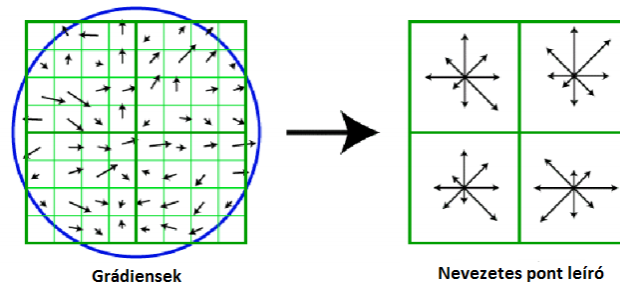
A jellemző pontok alapján az illesztő algoritmusok kiszámítják az egymást követő, azonos jelenetet ábrázoló képeken vagy pontfelhőkön az azokat összekapcsoló transzformációs mátrixokat. Ezen mátrixok segítségével egyesíthetővé válnak.

A kulcspontra illesztésre több megközelítés is létezik:

SIFT (Scale Invariant Feature Transform)

A módszer [21] nagy előnye, hogy gyors, valós időben is lehet használni. Akár 60°-os síkbeli elfordulásig alkalmazható.

Működése:

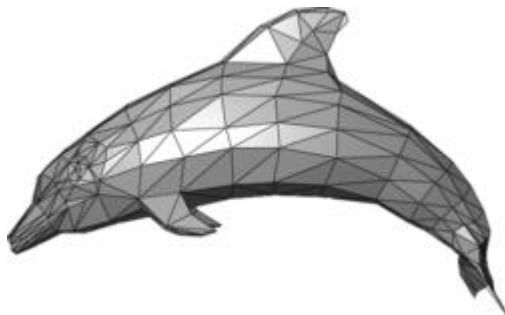


21. ábra David Lowe ábrája a SIFT működéséről [21]

- Vegyünk 16×16 méretű ablakot a detektált jellemző körül
- Határozzuk meg az él orientációt (gradiens szöge - 90°) minden pixelre
- A gyenge éleket hagyjuk el (gradiens nagyságot küszöbölve)
- Készítsünk hisztogramot a megmaradó él orientációkból

5. 3D rekonstrukció

A 3D objektum feldolgozásához és megjelenítéséhez egy összefüggő, folyamatos 3D modellre van szükségünk, ezért külön eljárással kell a jelenetet ábrázoló, egyesített pontfelhőt a megfelelő állapotúvá alakítanunk. Ez a gyakorlatban azt jelenti, hogy valamilyen módszerrel egy felszín rekonstrukciót kell végeznünk a pontfelhőn.



22. ábra Egy delfint ábrázoló poligon modell [14]

A pontfelhők legnagyobb hátránya, hogy diszkrét térbeli csúcsokból állnak, amik között üres tér található. A felhők nem tartalmaznak éleket és felszíneket sem, amik lehetővé tennék a textúrázást, vagy az objektum felületével kapcsolatos egyéb számításokat. Vizuális megjelenítés szempontjából ezek képpont csúcsok halmazai a térben. A felületi rekonstrukció lehetőséget ad a pontfelhő pontjai közti lyukak és üres terek kitöltésére, a modell általános finomítására és javítására. Az így kapott felszínekre már rá lehet helyezni 2D képpont adatokból generált textúrákat, megjeleníthetővé és feldolgozhatóvá téve a modellt.

A felszín rekonstrukciójára több megoldás létezik különböző módszerek és algoritmusok formájában. A legfontosabb szempontok a modell felszíneinek helyreállítása során az eredeti felszínek minél pontosabb és gyorsabb visszanyerése a pontfelhőből. A sebességük alapján megkülönböztethetünk valós idejű és nem valós idejű algoritmusokat.

Fontos megjegyezni, hogy bizonyos algoritmusok csak illesztett és rendezett pontfelhőkre működnek, míg mások képesek a felszín rekonstrukciójának a pontfelhő regisztrálása előtt történő elvégzésére is. Ha az utóbbi módszert választjuk, egy adott képkocka feldolgozási ideje jelentősen megnő, mivel a regisztrációs lépés előtt futtatni kell a rekonstrukciós algoritmust a képkockából generált lokális pontfelhőre. Ebben az esetben a modell rekonstrukciójához felhasználhatjuk a pontfelhőben detektált éleket is a nevezetes pontok helyett. A pontfelhők illesztésének lépése így akár el is hagyható, az illesztést valamilyen alternatív algoritmusokkal a rekonstruált modell inkrementesen kell elvégezni.

5.1. Algoritmusok

5.1.1. Powercrust

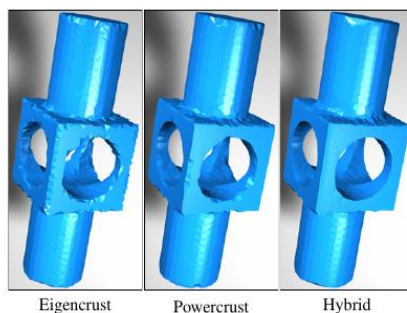
A PowerCrust [22] egy mohó algoritmus, ami egy bemeneti pontthalmaz mintából állít elő egy felszín hálót egy becsült középtengely segítségével. Először az objektum középtengely transzformáltjának (Medial Axis Transform) kiszámítása történik meg. A felszín a MAT-ból áll egy elő egy inverz transzformáció számításával. Zavaros pontfelhők esetén bizonytalan eredményt produkál, viszont ez az egyik leggyorsabb grafikai alapú megoldás.



23. ábra Lézeres mérésből előállított pontfelhő, a belőle generált polygon modell és a hozzá tartozó középtengely transzformált. [22]

5.1.2. Eigencrust

A PowerCrust módszerhez hasonló módon működik [23], felfogható annak továbbfejlesztéseként. Erősen zavaros pontfelhőből is képes teljesen sima felszínt konstruálni. Ezt megnövekedett futásidő árán éri el. Sajátvektorokat használ. A PowerCrusttal ellentétben globális optimumra törekszik, így a kapott eredmény pontosabb lesz.



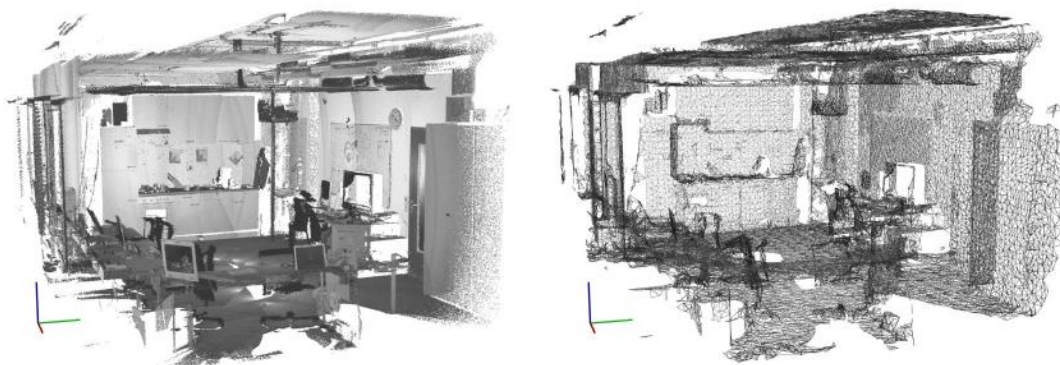
24. ábra A Powercrust algoritmuson alapuló módszerek összehasonlítása [23]



25. ábra Egy az eigencrust algoritmussal rekonstruált objektum. A 2 008 414 bemeneti pontból 12 926 063 tetraéder felhasználásával 3 605 096 kimeneti háromszög állt elő, 437 perc alatt. [23]

5.1.3. On Fast Surface Reconstruction Methods for Large and Noisy Point Clouds

Az eddig említett felszín rekonstrukciós algoritmusokkal szemben [24] zajos, kültéri vagy beltéri adatok alapján előállított pontfelhőkre is hatékonyan alkalmazható módszer. Az alapja egy mohó algoritmus, ami az előállított éleket azonnal kiírja, ezeket később már nem módosítja, ezáltal gyors és memória takarékos algoritmust eredményez. A háromszögelés inkrementálisan történik. A becsült, felszínre merőleges sík mentén történő vetítés az utolsó két lépésben megy végbe. Ez az egyszerűsítés csak akkor működik, ha a felszín valóban simának tekinthető.



26. ábra Egy konyha 15 millió pontból álló szürkeárnyaltos pontfelhő modellje, és a belőle előállított folytonos háló. [24]

5.1.4. Marching cubes algoritmus

Eredetileg CT és MRI felvételek vizualizálására fejlesztették ki [25]. Háromszög alapú modellt készít állandó sűrűségű 3D felszín adatok alapján. A bemeneti adatokon „szeletenként” megy végig. Egyszerre nyolc csúcsot felhasználva virtuális kockát formál, majd meghatározza az ezt metsző megfelelő síkot 256 lehetőségből, hogy ezzel közelítse a felszínt. Több továbbfejlesztése is létezik. A Hoppe és társai féle implementációban [26] többek között előjeles távolság függvények használatával egészül ki az eredeti algoritmus, ami tovább növeli a hatékonyságot.

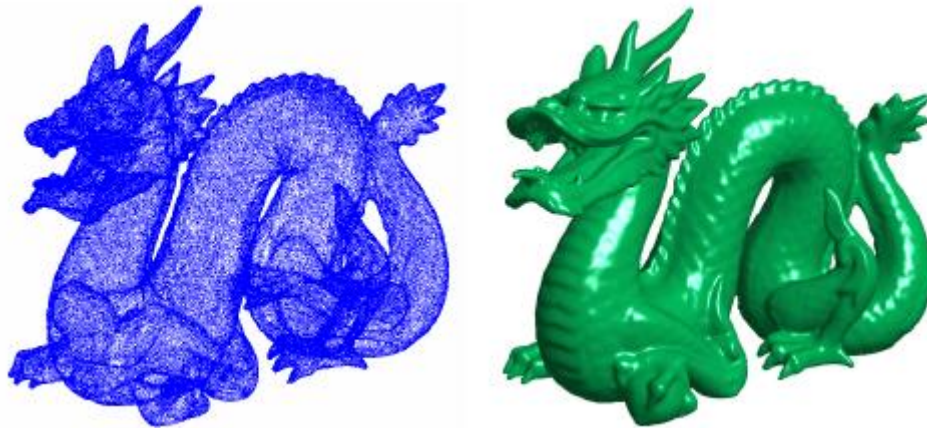
5.1.5. Poisson felszín rekonstrukció

A felszín rekonstrukciójának problémáját visszavezeti egy térbeli Poisson eloszlásra [27]. A függvény egyszerre dolgozza fel az összes pontot, nem végez heurisztikus számításokat, térbeli szétosztást vagy keverést, ezért erősen ellenálló a zajjal szemben. A radiális bázisfüggvényekkel ellentétben ez a megközelítés hierarchikus szervezésű, lokális bázisfüggvényeket használ, így a megoldás egy jól kondicionált lineáris egyenletrendszerre egyszerűsödik. A rekonstrukció sebessége és komplexitása arányos a visszaállított modell méretével.

5.1.6. Radiális bázisfüggvényeken alapuló rekonstrukció

Poliharmonikus radiális bázisfüggvényeket használ [28], hogy sima felületeket állítson vissza pontfelhőkből, vagy félkész hálókából. Az objektum felszínét implicit módon adja meg, ahogy a bázisfüggvényt zérus helyeinél az adott felszín adataira illeszti. A radiál bázisfüggvényeket

felhasználó gyors illesztési módszerek lehetővé teszik egy több millió pontból álló pontfelhő feldolgozását egyetlen függvény segítségével.



27. ábra Egy sárkányt ábrázoló pontfelhő, és a belőle radiális bázisfüggvénnyel előállított rekonstruált modell. [28]

Az illesztés egy mohó algoritmus segítségével történik, ezáltal egyszerűsödik a számítási folyamat, ami jelentős tömörítéshez és további előnyökhöz vezet. A poliharmonikus szplájnok energia minimalizálás karakterizációja sima interpolációt eredményez. Ez a méret független karakterizáció hatékony, nem egységes bemeneti adatok esetén.

Az algoritmus betömi a lyukakat és simára extrapolálja a felszínt. Zajos bemeneti adatok esetén nem interpoláló becslést alkalmaz. A függvény eredménye egy tömör modell, ami azt jelenti, hogy a gradiensek és a felszíni normálisok analitikus módon meghatározhatóak.

5.1.7. A Volumetric Method for Building Complex Models from Range Images

A módszer egy volumetrikus modellt épít egy kumulatív, előjeles, súlyozott távolság függvény alapján [29]. Nagy számban rendelkezésre álló mélységi képek feldolgozására fejlesztették ki, melyeket egy additív séma alapján szekvenciálisan, egyesével integrál a modellbe. Először a pontok felszínétől vett távolságát leíró függvény definiálása történik meg a bemeneti pontfelhő alapján. Ezt követően az új adathalmazt egy nevezetes pont illesztő algoritmus segítségével a modellhez tájoljuk, majd a modellben a felszínt leíró függvény súlyozását újraszámítjuk, ezzel közelítve az objektum eredeti felszínét.

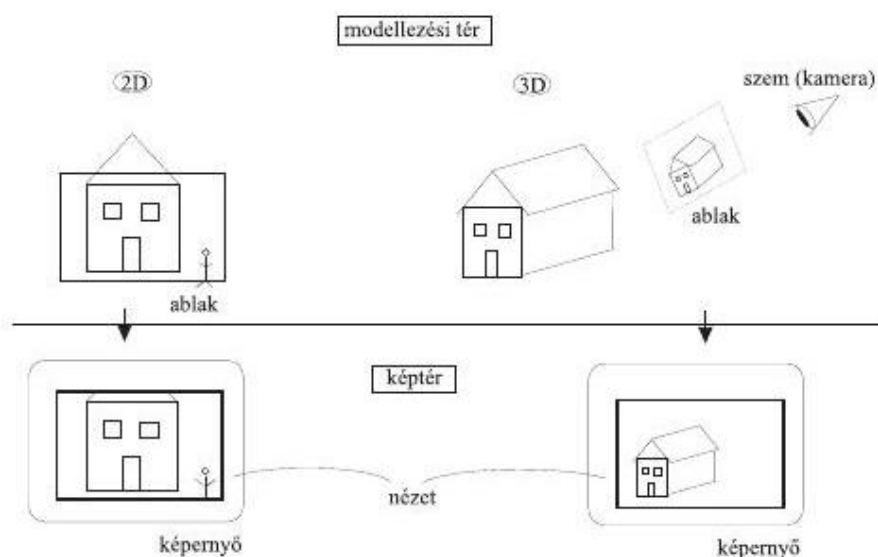
Előnyei:

- Kezeli a pontfelhők használata során fellépő bizonytalanságot
- Az adott szögből történő, ismételt mintavételezés csökkentheti a zajt
- Inkrementális- és feldolgozási sorrendtől független működésű
- Robosztus, a súlyos hibákat kezelő algoritmus
- Képes lyukmentes, watertight felszínt rekonstruálni
- A helyhatékonyság érdekében tömörítést alkalmaz a modellen, ezért nagyszámú mintára is használható

6. 3D modellek 2D leképezése és megjelenítése

6.1. 2D renderelés

A 3D objektumokat úgy tudjuk megjeleníteni a két dimenziós képernyőn, hogy elvégezzük rajtuk egy 2D képszintézis transzformációt, ezáltal egy 2D képet létrehozva. A transzformáció során meghatározunk egy virtuális kamera állást, egy 2D ablakot a 3D térben, amin keresztül a 3D modellt vizsgáljuk. Matematikai értelemben véve a modell 3D lokális koordinátáit először a 3D világ globális koordináta-rendszerébe, majd a virtuális kamera, vagyis a képernyő 2D lokális koordináta-rendszerébe képezzük. A 2D leképezésben a modell azon részei fognak látszódni, amelyek ezen a virtuális ablakon keresztül nézve nincsenek takarásban.



28. ábra 2D-3D leképezés [13]

A transzformáció 4 lépésből áll:

1. Vektorizáció

A 3D modellben található, valamilyen modell ábrázolási módszerrel leírt térbeli formákat poligonok halmazával közelítjük.

2. Transzformáció

Ebben a lépésben valósítjuk meg a modell lokális koordináta-rendszeréből a világkoordináta-rendszerbe, majd a 2D ablak koordináta-rendszerébe történő váltást. Ezeket a transzformációkat név szerint modellezési- és ablak-nézet transzformációknak nevezzük. A két művelet összevonható, a lépés visszavezethető transzformációs mátrixok láncszorzására. Így az egymást követő két transzformációt a továbbiakban egy egyesített mátrixszal történő szorzásként is felfoghatjuk.

3. Vágás

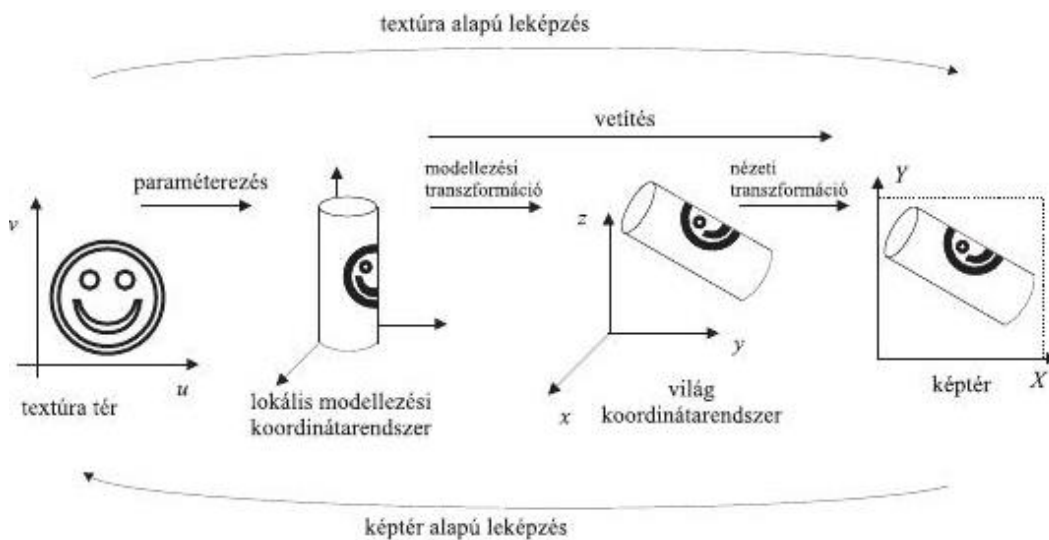
Eltávolítjuk a modell azon részeit, amelyek a definiált ablakon keresztül nézve nem látszódnak. Ezek az ablakon kívül található, vagy takarásban lévő pontok és élek.

4. Raszterizáció

Beállítjuk a transzformált, látható geometriai elemeket reprezentáló pixeleket a 2D képen.

6.2. Tértfogati sugárkövetés

A térfogatmodellek vizualizálására kifejlesztett módszer. A modellt az abban található csúcsok egymáshoz viszonyított távolsága alapján rétegekre bontjuk, majd ezekre merőleges „fényugarakat” indítunk a térfogatba. A különböző térfogati rétegekben található képpontokat a vetített sugárral megegyező vagy azzal ellentétes irányban bejárva meghatározzuk a látható külső pixeleket. Ezek a diszkrét pontok fogják adni az objektum felszínét. Végül egy interpolációval egybekötött háromszögelést végzünk az így kapott felszíni pontokon, aminek eredményeképpen egy megjeleníthető mesht kapunk. Ennek a módszernek az egyik megvalósítása a Marching cubes algoritmus is.



29. ábra Leképezés [13]

6.3. Textúra leképezés

A textúrák segítségével a modellek felszínén található változó szürkeárnyalatokat vagy színintenzitásokat tudjuk megjeleníteni. Ezeket az intenzitások általában a modell geometriai részétől elkülönítve kerülnek tárolásra, megadhatjuk őket például egy függvényvel, vagy egy tömbben tárolt adathalmazzal. Az értékalmazt, amiben a textúrákat értelmezzük, textúra-térnek nevezzük.

7. Az irodalomkutatás összegzése

Az irodalomkutatás alapján úgy döntöttünk, hogy a Kinect Fusion és a Point Cloud Library keretrendszereket szeretnénk felhasználni az alkalmazásunk fejlesztéséhez, mivel ezek a keretrendszerek megfelelő osztályokat és algoritmusokat biztosítanak a pontfelhők hatékony tárolásához és feldolgozásához. A használatuk elterjedt, és széles körben dokumentált.

A Kinect Fusion a fentebb említett „A Volumetric Method for Building Complex Models from Range Images” forrásban ismertetett algoritmus alapján implementált megoldást használja a 3D modell tárolására, valamint a felszíni rekonstrukció elvégzésére. Az új mélységi adatokat az „ICP” nevezetes pont kereső algoritmus segítségével illeszti be a modellbe. A Fusion testreszabhatósága a zárt forráskódja miatt erősen korlátozott.

A Point Cloud Library nyílt forráskódú és számtalan különböző, de funkcionalításban hasonló algoritmust kínál. A felszíni rekonstrukcióhoz többek között a Marching Cubes és a Poisson rekonstrukciós algoritmust, valamint a nevezetes pontok keresésére a Harris sarokdetektáló- és a SIFT algoritmus különböző implementációit is tartalmazza. A nyílt forráskód miatt szükség esetén van lehetőség saját rekonstrukciós és nevezetes pont illesztő függvények írására is.

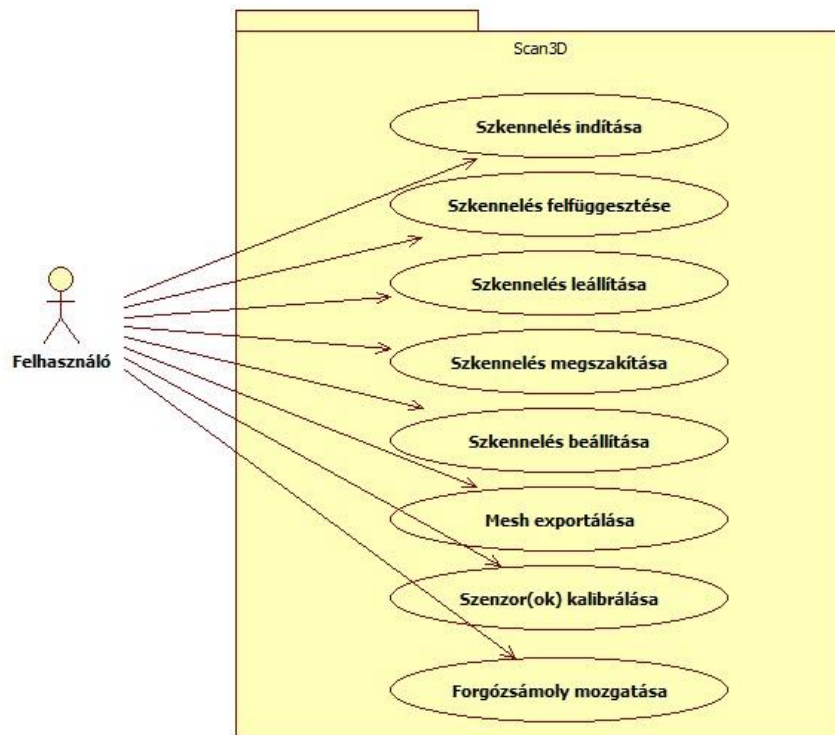
A két függvénykönyvtárat egyidejűleg, de nem feltétlenül kombinálható módon szeretnénk támogatni az alkalmazásunk végleges verziójában. Így lehetőség nyílna rá, hogy össze lehessen hasonlítani a hatékonyságukat a gyakorlatban. A felhasználó maga választhatná ki a használni kívánt alternatívát, amennyiben nem elégedett az általunk előre beállított rendszerrel és annak alapértelmezett algoritmusaival.

8. Rendszerterv

A projektünk fejlesztéséhez a vízesés modellt választottuk, mivel már a projekt kezdetén ismert volt minden a fejlesztés szempontjából fontos követelmény. Első lépésként definiáltuk ezeket, majd irodalomkutatást és analízist végeztünk. Ezek alapján pontosítottunk a kezdeti célkitűzéseken (lásd 8.1. fejezet), majd számba vettük a használati eseteket (lásd 30. ábra). A tervezés fázis során modulárisan felbontottuk a rendszert (lásd 8.2. fejezet), tanulmányoztuk és definiáltuk annak kapcsolatát a használt keretrendszerekkel és eszközökkel. Az alkalmazást Visual Studio 2012 fejlesztői környezetben, C++/CLI nyelven, a .NET keretrendszert felhasználva, Winforms alkalmazás sablon alapján kezdtük el fejleszteni. Mivel a Point Cloud Library 1.7.0 használata során a Visual Studio 2012-vel kompatibilitási problémákba ütköztünk, ezért végül a kód tényleges fordításához a fejlesztői környezet többi részét megtartva a Visual Studio 2010 fordítóját (MSVC 10.0) használtuk.

8.1. *A rendszerrel szemben támasztott követelmények*

- Megbízható működés, a szkennelés folyamata hibás adatok esetén sem szakadhat meg magától, de a felhasználó megszakíthatja azt.
- A szkennelés határozatlan vagy határozott ideig működik. A felhasználó kézzel is leállíthatja a folyamatot.
- Adjon folyamatos visszajelzést az aktuális folyamat állapotáról.
- A program használatához ne legyen szükség mérnöki ismeretekre, de a részletes beállítások szükség esetén legyenek testre szabhatóak.
- A rendszer képes legyen vezérelni a Kinect szenzor beépített motorját, valamint fogadni az RGBD adatfolyamot
- A szoftver képes legyen vezérelni a forgató mechanikát, ami a 360°-os látószöget biztosítja egy darab statikus szenzor használata esetén.
- A szoftver 40cm és 300cm közötti távolság intervallumban tudjon szkennelni.
- Egy szenzor használata esetén lehessen dinamikus és statikus kamerával is rögzíteni a jelenetet. Több szenzor használata esetén a statikus konfiguráció legyen elérhető.
- Átlátható, könnyen kezelhető, felhasználóbarát grafikus felület.
- A szkennelt modelleket ply, obj és stl formátumban el lehessen menteni
- A pontfelhő építése és felszín rekonstrukciója lehetőség szerint valós idejű legyen (Ez erősen hardver és beállítás függő).
- Az alkalmazás legyen moduláris felépítésű a későbbi bővíthetőség érdekében.



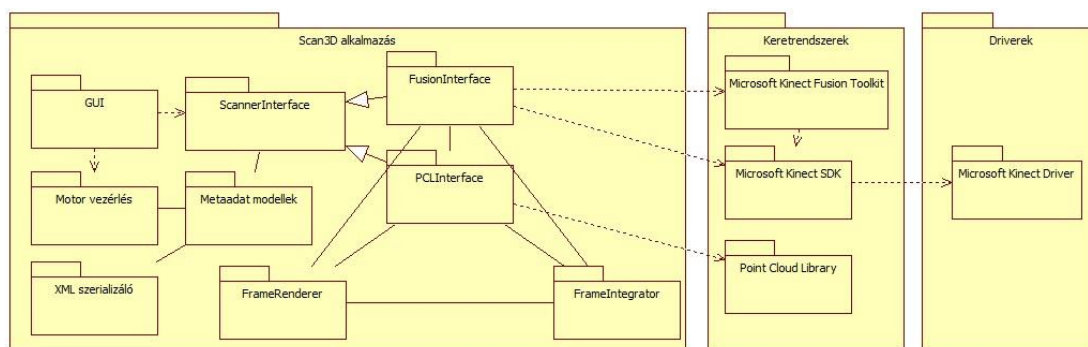
30. ábra Felhasználói esetek

8.2. A rendszer moduljai

Az alkalmazásunk moduláris felépítésű, négy nagy szerkezeti szintből áll. Ezek felülről lefelé a következők:

1. Winforms alapú grafikus felhasználói felület szintje
2. A saját fejlesztésű interfész könyvtár (üzleti logikát megvalósító) szint
3. A keretrendszerek szintje
4. A szkennert meghajtó driverek szintje

Az interfész könyvtár szinten találhatóak azok a saját fejlesztésű osztályok, amelyekkel megvalósítjuk a különböző keretrendszerek közötti konverziós és mérési algoritmusokat. A moduláris felépítés miatt minden keretrendszernek és függőségnek saját interfészt definiáltunk (lásd 8.3. fejezet). A rekonstrukciót végző objektumok is ezeken keresztül képesek kommunikálni a használt keretrendszerrel. A szinten találhatóak még a forgózsámolyt vezérlő metódusok is.

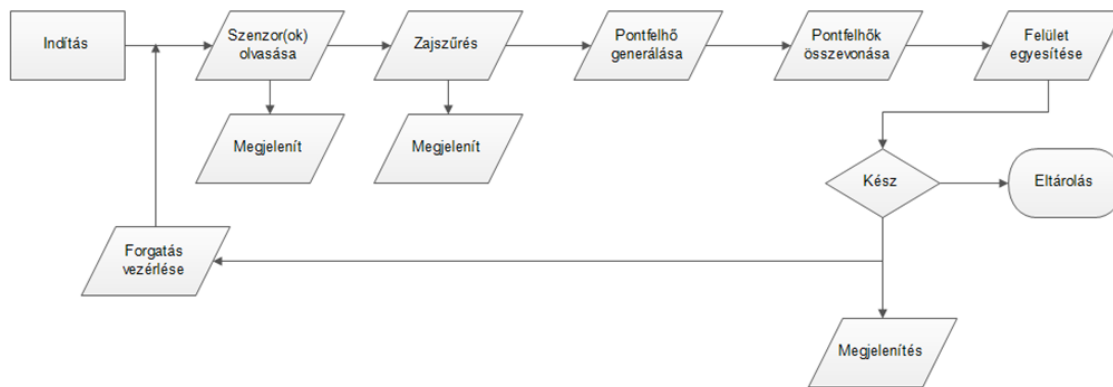


31. ábra Csomag diagram

A csomag diagramon (lásd 31. ábra) láthatóak a különböző keretrendszerek és driver függőségek. A Point Cloud Library az OpenNI rendszerre épül, azon keresztül fér hozzá a szenzorok adatfolyamaihoz. Ehhez alapesetben a Primesense által biztosított drivereket használja. A csomagban található ugyan meghajtó program a Kinect Xbox360-as verziójához, de ez a technikai különbségek miatt a Kinect for Windows-zal nem használható, továbbá a Primesense Kinect driver nem telepíthető egyidejűleg a Microsoft Kinect Driverrel (továbbiakban MKD), mert a két program a használat során összeütközik. Ennek a problémának az áthidalását a későbbiekben fejtjük ki (lásd 8.5.2. fejezet).

8.3. *Interfész modulok*

A modularitás elvét tükrözik az egymástól elkülönülő, különböző keretrendszereket csatoló interfészek is. A Microsoft Kinect Fusion (FusionInterface osztály) és a Point Cloud Library (PCLInterface osztály) rendszerek egyaránt használhatóak a modell felépítésére, és a felszín rekonstrukció elvégzésére. A szenzorból származó adatokat jelenleg mindenképpen a Kinect SDK segítségével nyeri ki az alkalmazás, mivel a zárt forráskódú Kinect SDK Driver és a különböző nyílt forráskódú driver implementációk inkompatibilisek egymással (lásd 8.5.2. fejezet). A problémát kezdetben egy nyílt forráskódú program, a kinect-mssdk-openni-pcl-bridge [30] segítségével próbáltuk megoldani, ami egy áthidalást hoz létre a Kinect SDK Driver és a PCL rendszerek között. Ezt úgy valósítja meg, hogy az SDK Driver köré írt wrapperként beépülő modul formájában települ az OpenNI-be, olyan módon, hogy az OpenNI-t alkalmazó PCL-lel is használható legyen. Ez az eleinte ígéretesnek tűnő módszer a gyakorlati alkalmazása során zsákcához vezetett, mert a kinect-mssdk-openni-pcl-bridge használata során kompatibilitásbeli problémákba ütköztünk a Kinect SDK 1.8-as verziójával. A Kinect SDK 1.8-nál korábbi keretrendszer verzió használata nem volt járható út, mert így elvesztettük volna a szakdolgozat írásának idején kizárólag az 1.8-as verzió által biztosított színes rekonstrukció képességét. A problémára alternatív megoldást kellett keresnünk, amit a két keretrendszer ötvöztetésének képében találtunk meg. A PCLInterface osztály használata esetén az adatok kinyerése ezért a Kinect SDK-n keresztül történik, ezeket az adatokat konvertáljuk át egy a PCL által támogatott natív formátumba. Ezáltal használhatóvá válik a PCL a Kinect SDK Driverrel különböző, a drivereket áthidaló megoldások nélkül.

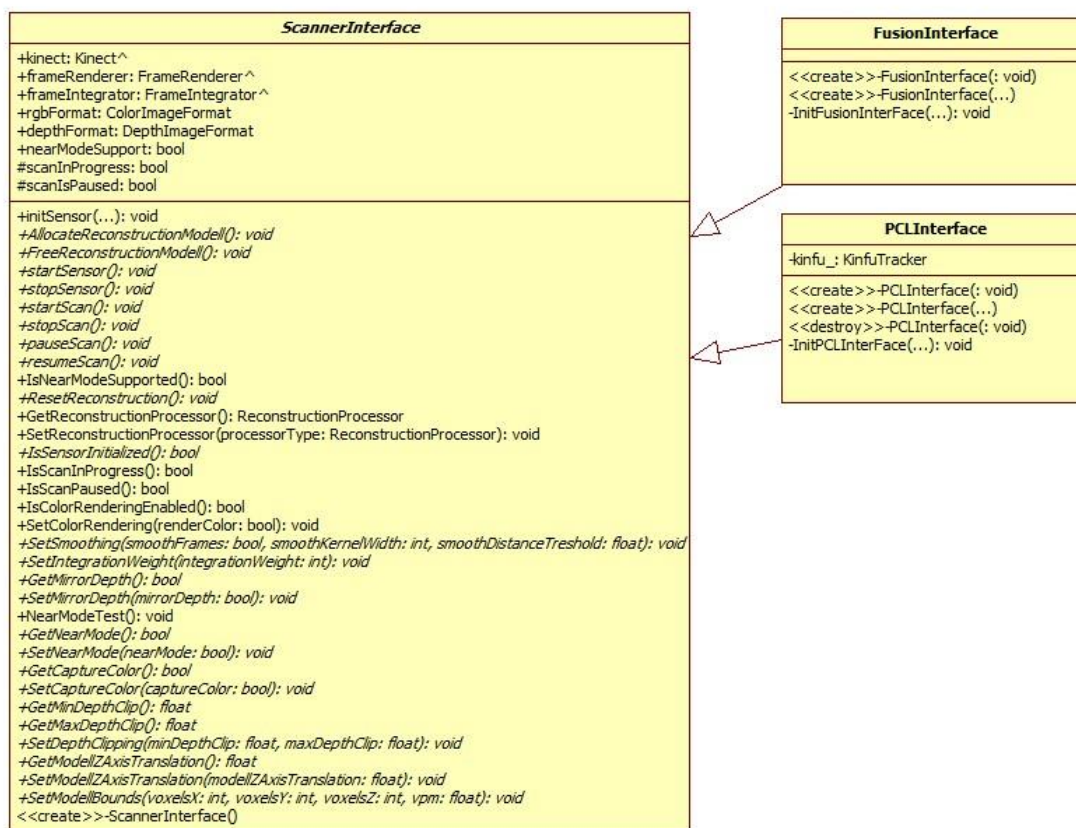


32. ábra Általános működés folyamat diagramja

8.4. A rendszer osztályai és azok kapcsolatai

8.4.1. Keretrendszer interfészek

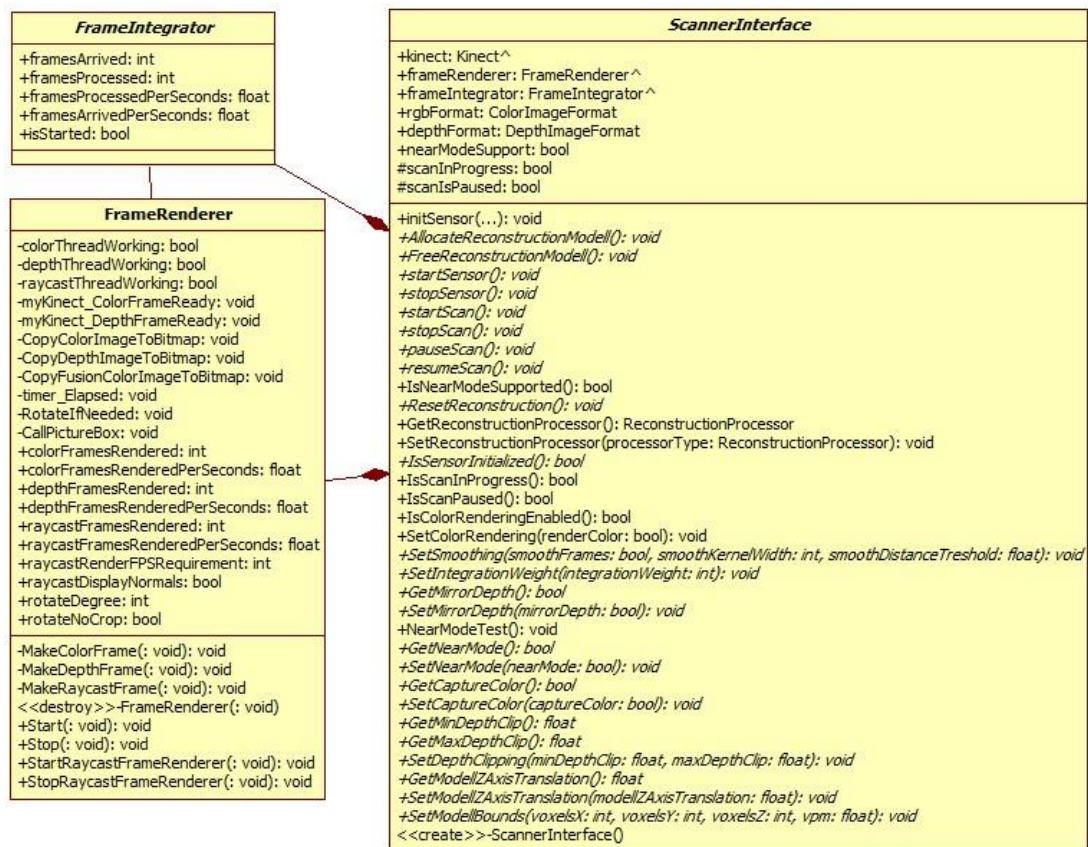
A Kinect Fusion és a PCL keretrendszerekkel egy-egy adott interfészen keresztül kommunikál a rendszerünk (lásd 33. ábra). A két háttérrendszer eltéréseit a felsőbb szinteken egy ScannerInterface típusú absztrakt osztállyal rejtjük el. Ennek leszármazottai a FusionInterface és a PCLInterface. A kiválasztott keretrendszertől függően mindig a megfelelő interfész töltődik be. A GUI elemei a ScannerInterface-en keresztül kommunikálnak a Scan3D könyvtárral.



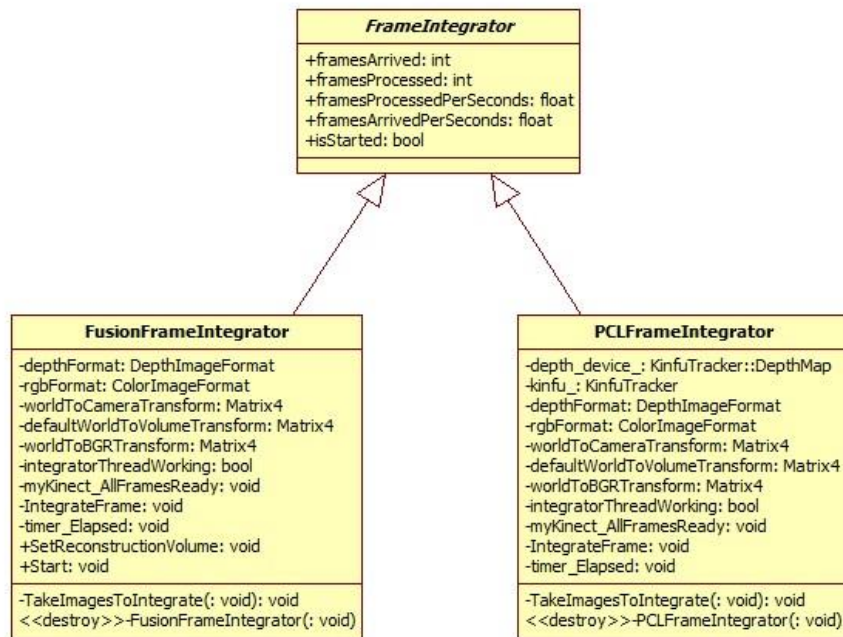
33. ábra A keretrendszerekkel kapcsolatot teremtő interfész osztályok főbb függvényei és adatai

8.4.2. Adatkeretek kezelése és integrációja

Az adatkeretek kezelését a FrameIntegrator és FrameRenderer osztályok segítségével valósítjuk meg (lásd 34. ábra). A FrameIntegrator absztrakt osztály és leszármazottai (lásd 35. ábra) felelősek a használt keretrendszerből újonnan érkező adatcsomagok megfelelő formátumba konvertálásáért, a kamerapóz követéséért és a képkockák beillesztéséért. Minden keretrendszer interfészhez pontosan egy FrameIntegrator és egy FrameRenderer tartozik. Ezek az osztályok a gyorsabb és hatékonyabb többszálúsítás miatt közvetlenül egymással is kommunikálnak. A különböző nagy erőforrásigényű és hosszú ideig tartó műveletek külön szálhoz rendelése is itt történik.



34. ábra Az adatkereteket kezelő osztályok és kapcsolataik



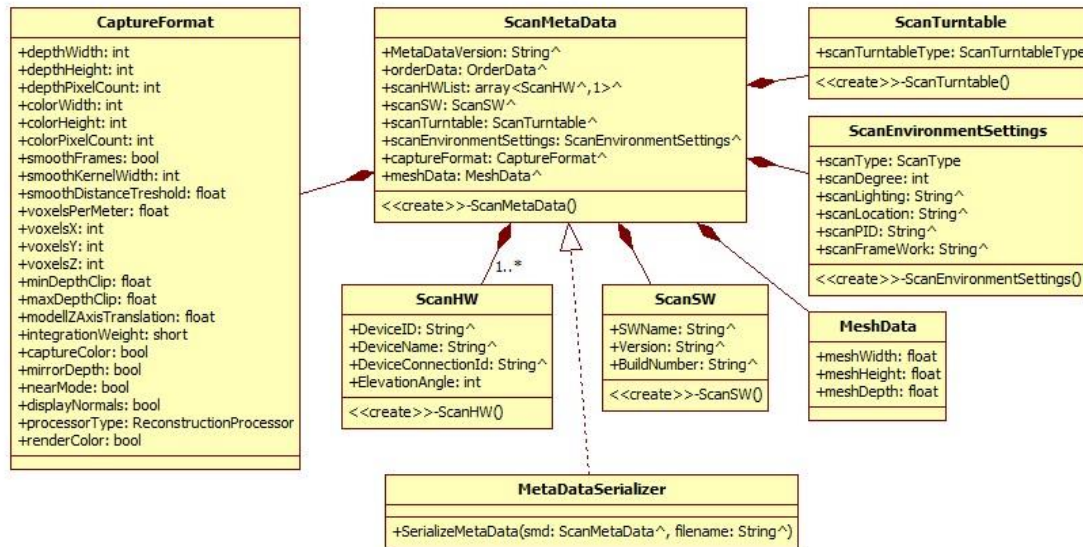
35. ábra A modellbe integrálást végző osztályok és kapcsolataik

8.4.3. Metaadat-osztályok

Az exportált modellek kategorizálása és tárolása során szükség lehet a szkennelés során használt beállítások megőrzésére, és a modell tulajdonságainak szöveges formában történő mentésére. Ezeket a metaadatokat a ScanMetaData osztályban és annak alosztályaiban tároljuk (lásd 36. ábra).

A metaadat alosztályok és feladataik:

- **CaptureFormat:** Ez az osztály felelős a szkennelési és modell- és kamera beállítások tárolásáért.
- **ScanTurntable:** Ehhez az osztályhoz köthető minden, a forgató zsámolyhoz kapcsolódó érték és beállítás.
- **ScanEnvironmentSettings:** A külső környezeti értékeket rögzítő osztály.
- **MeshData:** A generált háromszög háló adatait tárolja.
- **ScanHW** és **ScanSW:** A használt hardver és szoftver beállításokat menti el.
- **MetaDataSerializer:** A .NET beépített XML szerializálóján keresztül XML formátumban menti a ScanMetaData osztály tagváltozóit.



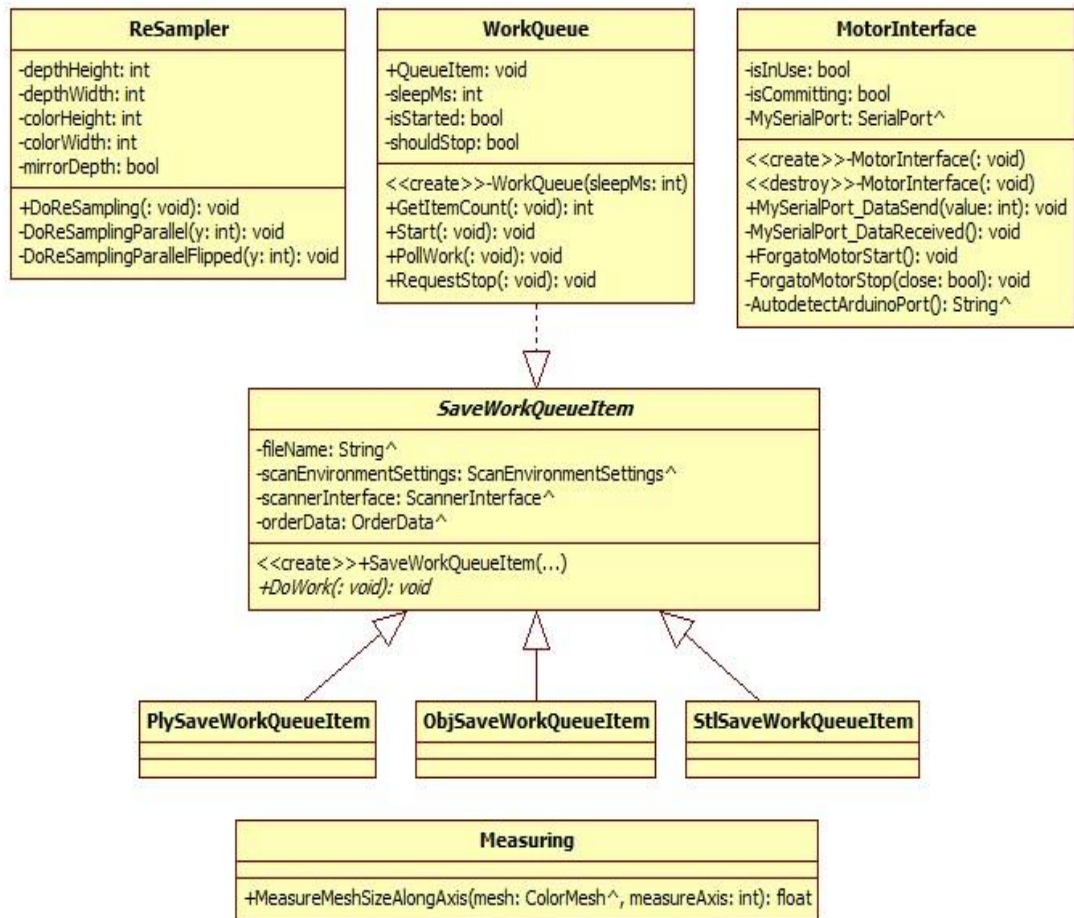
36. ábra Metaadat-osztályok és kapcsolataik

8.4.4. Segítőosztályok

A feldolgozás során olyan saját osztályokat is készítettünk, amelyeket az előző kategóriákba csak nehezen lehetne besorolni. Ilyenek például a különböző konverziós műveleteket, méréseket megvalósító, vagy a modell exportálása során a többszázalást végző egységek (lásd 37. ábra).

A segítőosztályok és feladataik:

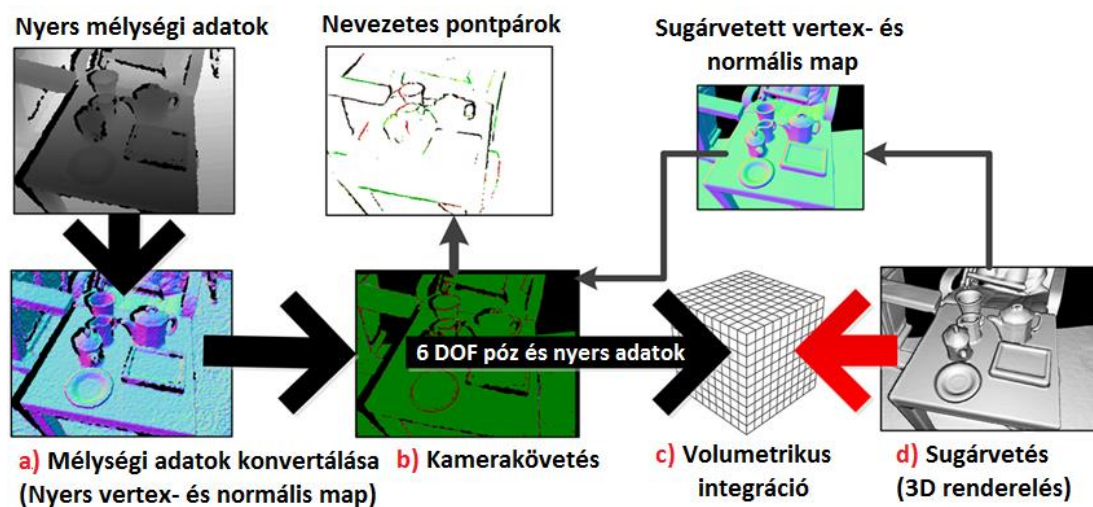
- **WorkQueue:** A mesh különböző formátumokba történő exportálását egy SaveWorkQueueItem osztályokból álló feladatsor létrehozásával kezelő osztály.
- **SaveWorkQueueItem:** Az exportformátumok különbségeit elrejtő absztrakt osztály. DoWork absztrakt metódusa a leszármazott osztályokban felülbírálásra kerül, ezáltal minden formátumnak más mentési algoritmusuk lehet.
- **ReSampler:** A Kinect Fusion SDK használata esetén az RGB- és a mélységi frame-ek közti transzformációt és átskálázást kezelő osztály.
- **Measuring:** A mesh tulajdonságait meghatározó statikus osztály.
- **MotorInterface:** A forgató számmal kapcsolatot teremtő interfész osztály.



37. ábra A Scan3D működését segítő osztályok

8.5. A szkennelés folyamata

8.5.1. A szkennelés lépései a Fusion keretrendszer használata esetén



38. ábra Kinect Fusion Pipeline [31]

1. Kiolvassuk a mélységi- és az RGB streamet a szenzor(ok)ból
2. Átkonvertáljuk és normalizáljuk a mélységi adatokat

3. Pozíciókövetéssel vagy az ICP algoritmussal (lásd 4.2. fejezet) megállapítjuk a kamera pozícióját a modellhez képest, és meghatározzuk a transzformációs mátrixot
4. A transzformációs mátrix segítségével transzformáljuk az új adatokat a modell koordináta rendszerébe.
5. Elvégezzük az új kép- és mélységi pontok beillesztését.
6. Sugárkövetéssel (lásd 6. fejezet) 2D képet készítünk az első- vagy a legutóbb rögzített Kinect póz alapján.
7. Az 1-7. lépést a szkennelés leállításáig iteráljuk
8. Ha a felhasználó el kívánja menteni a modellt háromszög alapú poligon hálót készítünk, majd a kiválasztott formátumban (ply, stl, obj) (lásd 10. fejezet) mentjük azt.

8.5.2. A szkennelés lépései a Point Cloud Library használata esetén

Mivel a Point Cloud Library Kinfu modulja a Kinect Fusion-höz nagyon hasonlóan működik, ezért az elméleti lépései megegyeznek az előző alfejezetben tárgyaltakkal. A szenzor adatokat a már korábban ismertetett driver inkompatibilitási probléma miatt a program jelenleg a Kinect SDK-ból nyeri ki, és alakítja át a PCL által használható formátumúvá. Ezt a PCLFrameIntegrator objektum kezeli, aminek ezáltal a Fusion SDK irányába is van egy függősége.

8.6. Tértfogatmodell létrehozása

A megfelelő interfész kiválasztása és példányosítása után megtörténik a volumetrikus modell létrehozása. Ez egy téglatest alakú háromdimenziós térrészletet reprezentál, aminek beállítható az X, Y, Z irányú voxel felbontása, valamint a sűrűsége. Az első három paraméter adja meg, hogy a modell egy adott tengelye mentén hány különböző voxelt tud megkülönböztetni. Ezek a voxelek a valódi térből beolvasott mélységi adatok alapján definiált térbeli pontokat jelentik a modellben, önmagukban nem utalnak a modell térbeli méretére. Ezt az információt a sűrűséggel képzett hányadosuk adja meg. Például ha adott tengely felbontása 512 voxel, és a sűrűség 256 voxel/méter (vpm), akkor a tengely mentén a modell egy $512 \text{ voxel} / 256 \text{ vpm} = 2 \text{ m}$ -es térrészt reprezentál. Ha minden irányban 512 voxelnek választjuk a felbontást, egy 2 m^3 -es térfogatú modell generálódik. Ebben a modellben két pontot akkor lehet megkülönböztetni, ha azok egymástól mért távolsága legalább $2000 \text{ mm} / 512 \text{ voxel} = 3,9 \text{ mm} / \text{voxel}$.)

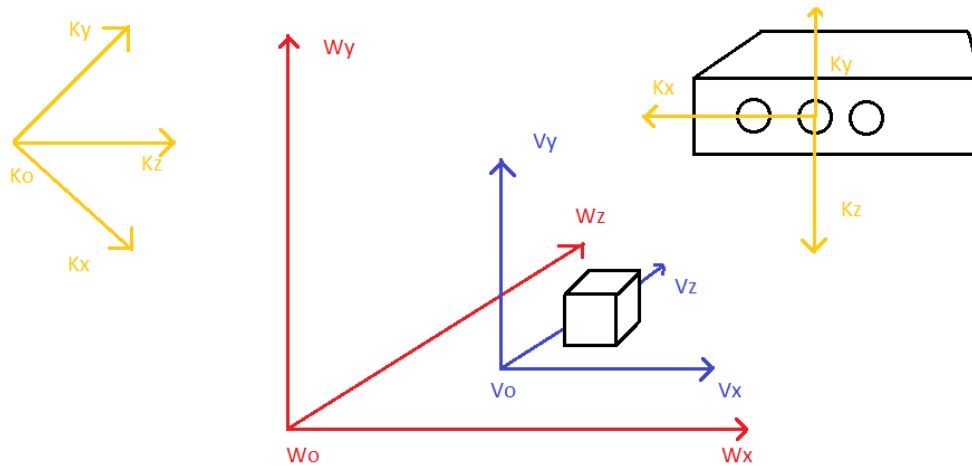
Mivel a Kinect Fusion és a Point Cloud Library támogatják az NVidia CUDA technológiáját, ezért lehetőség van rá, hogy a programunk a beállításoktól függően a modell tárolását és a hozzá kapcsolódó számítások egy részét a GPU-n végezze. Ehhez egy DirectX 11 kompatibilis GPU-ra van szükség. GPU alapú rekonstrukció esetén a modellhez szükséges tárhelyet a VRAM-ban foglalja le, így annak mérete határozza meg a rekonstrukciós modell maximális lehetséges méreteit. Ha a CPU alapú rekonstrukciót választjuk, az allokáció a RAM-ban történik, a műveleteket a CPU végzi. Ebben az esetben a jelenleg elterjedt asztali számítógép rendszerekkel a valós idejű rekonstrukció nem lehetséges.

8.7. Kamerakövetés

A modell felépítése során mélységi- és színadatokat használunk, amelyeket adatkeretek formájában olvasunk be. A színadatok esetében ez 30 FPS-t feltételezve másodpercenként 30

RGB képkockát jelent, a mélységi adatok esetében 30, mélységi adatokat tartalmazó 2D mátrixot. Ezeket a mátrixokat megjelenítés szempontjából természetesen felfoghatjuk szürkeárnyaltos képek sorozataként is. Ahhoz, hogy integrálni tudjuk az adatokat, szükség van transzformációkra, amelyekkel megállapítható az elkapott adatkeretek egymáshoz viszonyított- és a modell által leírt térben elfoglalt helyzete. Ehhez 4×4-es transzformációs mátrixokat alkalmazunk. Három koordináta-rendszert különböztethetünk meg:

- Világ-koordináta-rendszer (W)
- Modell-koordináta-rendszer (M)
- Kamera-koordináta-rendszer (K)



39. ábra A koordináta-rendszerek kapcsolata

A szkennelés kezdetekor meghatározásra kerül egy alapértelmezett világ-modell- és egy kezdeti világ-kamera transzformáció. Az alkalmazásunkban a világ-koordináta-rendszer mindig megegyezik az első mélységi képkockához tartozó kamerapozíció koordináta-rendszerével, tehát $\overline{Kx0} = \overline{Wx}$, $\overline{Ky0} = \overline{Wy}$, $\overline{Kz0} = \overline{Wz}$.

A kezdeti kamera-világ transzformációs mátrix a 4x4-es identitás mátrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A modell koordináta-rendszert a gyakorlatban úgy határozzuk meg, hogy a világ-koordináta-rendszert a Wz tengely mentén egy méterben megadott értékkel (Tz), általában a beállított minimális távolság szűrővel azonos értékkel eltoljuk.

A világ-modell transzformációs mátrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -(Tz * vpm) & 1 \end{bmatrix}$$

A világ-modell relációt leíró mátrix csak a modell újrainicializálásakor vagy kiürítésekor („resetelésekor”) változik, a világ-kamera transzformáció minden sikeres integráció után frissül.

Minden frame-hez párosításra kerül egy kamera-világ és egy világ-kamera transzformáció. Mivel ezek egymás inverz műveletei, ezért bármelyik ismerete esetén a másik is kiszámítható. A kamera-világ és világ-modell transzformációk alapján egyértelműen meghatározható a kamera által mért távolság adatokat a modell koordináta-rendszerébe képező művelet. Ennek legutóbb használt paraméterei minden sikeres integráció után mentésre kerülnek, mert segítségükkel gyorsítható a következő adatkeret beillesztése. Ez a gyorsítás azért lehetséges, mert általánosságban feltételezhetjük, hogy két képkocka között csak kismértékű elmozdulás történt.

A transzformáció meghatározása történhet valamilyen elmozdulás (pl: gyorsulás) érzékelő segítségével, vagy nevezetes pontok alapján. Az utóbbi megoldást alkalmazó módszer az általunk használt ICP algoritmus is.

8.8. ICP (Iterative Closest Point) algoritmus

A Kinect Fusion és a PCL KinFu által is használt algoritmus [32], amit háromdimenziós pontfelhők közel valós idejű összeillesztéséhez optimalizáltak. A paraméterek optimális megválasztása esetén akár valós idejű illesztést is lehetővé tehet.

Működésének főbb lépései:

- Kulcspontokat választ ki a pontfelhőből és a modellből
- Megkeresi a hasonló mintákat, és ez alapján pont párokat hoz létre
- Súlyozza az összeillő párokat
- Eldobja a nem megfelelő párokat
- Mérési hibaarányt rendel a pont párokhoz
- Minimalizálja ezt az arányt

Az algoritmus sebességét jelentősen befolyásolják annak paraméterei, mint a pontfelhők mérete, a két pontfelhő egymáshoz képest értelmezett elmozdulása, az általuk bezárt szög, valamint az iterációk száma. Eredménye egy transzformáció, ami az új pontfelhő elemeit a kezdeti pontfelhő koordináta-rendszerébe viszi át. Bizonyos implementációi nem csak a transzformáció meghatározását, hanem az új pontok súlyozott beillesztését is elvégzik.

8.9. Színes szkennelési módszerek

Az alkalmazásunk két különböző módszert kíván biztosítani a színek rögzítésére. Ezek közül eddig az első került implementálásra, a második algoritmus integrálását a szoftverünkbe a közeljövőben tervezzük.

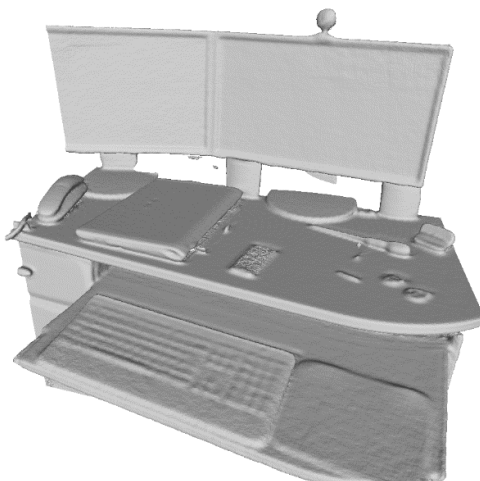
8.9.1. Vertex színek használata

A mélységi adatkeretek mellett a színesek is felhasználásra kerülnek a rekonstrukció során. Ezekből nyerhető ki a modell egy adott csúcsához tartozó RGB színintenzitása. A módszer előnye, hogy egy egyszerű leképezéssel kiszámítható az RGB- és a mélységi kamera koordináta-rendszerei közötti transzformáció. Miután a csúcsoknak tudjuk a színintenzitásait, a

mesh elkészítésekor a felület színét a poligoncsúcsok színintenzitásainak átlaga adja (lásd 40. ábra, valamint 41. ábra).



40. ábra Egy a saját alkalmazásunkkal készített, vertex színeket használó modellből generált nagy részletességű színes mesh.



41. ábra A fenti mesh színek nélkül renderelt változata.

8.9.2. UV leképezés használata

A voxel színes megoldás problémája, hogy a színadatokat nem a mesht alkotó háromszögek síkjához, hanem az azt felépítő csúcsokhoz rendeli hozzá. Ebben az esetben a rekonstrukciós lépés során a háromszög síkjának színét annak három csúcsa közti színátmenetként vagy átlagaként lehet értelmezni. A pontok közti színinformáció így elveszik, helyette egy átlagolt érték kerül a meshbe. Erre a problémára nyújt megoldást az UV leképezés. Ez a módszer nem a csúcsokhoz, hanem a háromszög által kifeszített síkhoz rendeli hozzá egy 2D textúra adott tartományát. A modell színeinek részletességét a textúra felbontása határozza meg. A módszer problémája, hogy a leképezéshez létre kell hozni egy hálót a modell alapján, ami nagyon elbonyolíthatja az algoritmust. A textúrákat nem lehet a térfogatmodellben tárolni, a leképezést a rögzített képkockák és a hozzájuk tartozó kamera pózok (transzformációk) alapján kell a meshre vetíteni. A módszerhez megfogalmaztuk egy algoritmus lépéseit.

A nagy felbontású textúrákat a meshre leképező algoritmus:

1. A mesh alapján meghatározzuk annak hálóját (unwrapping) és elkészítjük az UV mapot, amivel egyértelműen meghatározható a kapott háló minden pixeléhez a hozzá tartozó RGB érték.
2. Végigmegyünk minden pózon (minden pózhoz tartozik egy RGB frame).
3. Végigmegyünk minden face-en (aminek még nincs textúrája más pózból).
4. Ha a face normálisa és a kamera póz vektora egymással megfelelő szöget zárnak be, és a face a kamera vektor irányából nincs takarásban, akkor a face látható az adott pózból.
5. Kivágjuk a megfelelő háromszöget az RGB frame-ből, és átmásoljuk az UV map alapján meghatározott helyre a textúra fájlba.

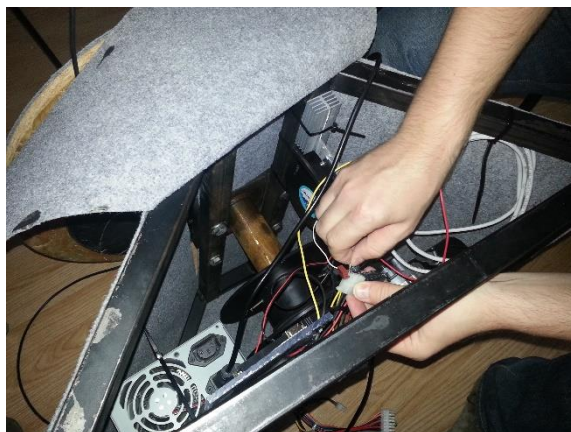
A jövőben egy nagy felbontású HD kamerával szeretnénk kiváltani a szenzor beépített kameráját, amivel ugyanarra a modellre sokkal több képpontot tudnánk leképezni. A részletgazdagabb modell ezáltal sokkal eredményesebb színrekonstrukciót tenne lehetővé.

A feldolgozás során kiemelten foglalkoznánk a HD kamera kalibrációjával és a kamerák szinkronizálásának kérdésével.

9. A felhasznált és támogatott hardverek

9.1. *Forgózsámoly és vezérlés*

Az automatikus feldolgozás érdekében jelenleg egy részben saját kivitelezésű, számítógépről irányítható, elektromos forgatható „zsámoly” áll rendelkezésünkre (lásd 42. ábra, valamint 43. ábra), amit egy Arduino Uno panellel vezérlünk. A hardver szerkezetileg nem, de az Arduino panelen található motor vezérlő firmware saját fejlesztésű.



42. ábra Forgózsámoly 1.0

A zsámoly használatakor a szenzor stabil helyzetben van, és a modell forog a saját tengelye körül.

A szenzor stabilitása több okból is fontos lehet. Az apróbb elmozdulásokat az algoritmusok kiszűrik, de ha túl gyors a mozgás, akkor a kamera képe elmosódhat, ami a feldolgozást megnehezítheti, vagy akár meg is akadályozhatja. Érdekes egyenletes ütemben, egy adott, nem túl nagy forgatási sebességen végezni a szkennelést. A szkennelési térbe „belógó”, a modell részét nem képező fix objektumok szintén problémát jelenthetnek, mivel az algoritmus így nem tudja egyértelműen detektálni a forgatást.



43. ábra Forgózsámoly 2.0

A mobilitást és a megbízhatóságot szem előtt tartva készült egy második forgózsámoly. Az előző verzió hibái kiküszöbölésre kerültek, a zsámoly újabb motorvezérlőt, rendezett kábelezést és fix tápellátást kapott. Az Arduino panelen lecseréltük a firmware-t, egy sokkal

több sebességi fokozatot támogatóra, amivel könnyebbé vált az irányítás. Ezeket a változtatásokat a szoftverünkbe is implementálnunk kellett, amit még egyéb fejlesztésekkel együtt el is végeztünk.

9.2. *Arduino Uno*

Az Arduino [33] egy hobbi elektronikusoknak szánt fejlesztőplatform. Segítségével némi programozói és elektronikai tudással is készíthetünk apróbb projekteket, az egyszerű villogó áramkörtől a robotokig.

Arduino-nak a szoftver és a hardver együttesét hívjuk: a hardver egy Atmel AVR mikrokontroller köré épülő nyákból, a szoftver az Arduino IDE-ből (fejlesztőkörnyezet) áll.



44. ábra Arduino Uno [33]

A boardok áramköri rajzai elérhetőek a gyártó honlapján, így akár magunk is elkészíthetjük, ha értünk hozzá, vagy megrendelhetjük a gyártótól, esetleg vehetünk egy olcsóbb után gyártott változatot is. A fejlesztőkörnyezet ingyenes, nyílt forráskódú és keresztplatformos (Windowson kívül van Mac-es, Linuxos, BSD-s kliens is).

A board USB-n keresztül csatlakoztatható a számítógépre, így tölthetjük fel a kész kódot (firmwaret), valamint áramot is ezen kap (5V), ha egyéb külső áramforráshoz nem csatlakoztatjuk. A rajta levő csatlakozókat pin-eknek nevezzük, ezekhez kábeleket, kiegészítő nyákokat és egyéb áramköri elemeket csatlakoztathatunk.

Az Arduino Uno egy mikrokontrolleren, az ATmega328-on alapul. Ez 14 digitális bemeneti / kimeneti csatlakozással rendelkezik (ebből 6 használható PWM kimenetre, amiből jelenleg mi kettőt alkalmazunk) 6 analóg bemenet, 16 MHz-es kerámia rezonátor, USB-kapcsolat, hálózati csatlakozó, ICSP fejléc, és a reset gomb. Egyszerűen lehet csatlakoztatni a számítógéphez egy USB-kábelrel vagy 5V-ot is lehet neki adni AC-DC adapter vagy egy akkumulátor segítségével.

Az Uno különbözik az összes korábbi board-tól, mert nem használja az FTDI USB-soros meghajtó chipet.

Az erre a célra megírt Arduino programnak köszönhetően, a modellt tudjuk forgatni, változtatni a sebességét, irányát, ahogy a feldolgozás sebessége engedi.

Létezik egy kiegészítő alkalmazás, Arduino Ide for Visual Studio néven, amivel az általunk jól megszokott Microsoft Visual Studio 2012 fejlesztő környezetben lehet hangolni az Arduino kódját. [34]

Vezérlőkód 1.0:

Első megoldásként hexadecimális számokat küldtünk Com porton az Arduino panelnek, ami a rátöltött firmware segítségével a hozzákapcsolt motorvezérlőt irányítva 16 különböző sebességi fokozatot értünk el. Úgy programoztuk fel a panelt, hogy 8-as értékre álljon meg a forgás, illetve 0-7 között balra, 9-15 között jobbra forogjon egy előre beállított értékkel.

Vezérlőkód 2.0:

Az alap elgondolást továbbfejlesztve Com-porton most már a forgás sebességét és irányát is egy valós értékkel adjuk meg, így a firmware leegyszerűsödik és több száz sebességet tudunk használni. Negatív számok a balra, pozitívak pedig jobbra forgatják a zsámoly forgó részét.

Továbbfejlesztési lehetőséget hagyva magunknak, a 2.0-ás forgózsámoly, és 2.0-ás vezérlőkódhoz egy újabb motorvezérlőt kapcsoltunk, ami későbbiek során 2db léptető-motort is képes vezérelni a jelenlegi 1 lineáris motor helyett.

9.1. *Firmware és szoftver implementáció*

A fizikai USB kapcsolatra egy virtuális Com portos kommunikáció társul.

A szkennelő program úgy van megírva, hogy ne egy fix porthoz rendelve, hanem automatikusan megtalálja a panelt és felépítse a kapcsolatot. Ezt 57600as adatátviteli sebességgel, paritásbit használata nélkül, 8 biten küldjük az adatokat és 1 stopbitet használunk. Az adatokat sorban küldjük, megvárjuk a visszajelzést a paneltől és csak utána lehet neki az újabb adatsort küldeni. Minden adatsort végét egy „\n” karakterrel zárjuk le.

Amíg motor feszültsége nem éri el a minimális forgatóerőt addig nem mozdul a forgópad. Ezt a programunkban egy fix eltolással szabályozzuk, hogy a vezérlés közben ne legyen olyan, hogy véletlenül megáll. Körülbelül 350 különböző sebességi fokozatunk van felosztva két irányba. A firmware úgy van megírva, hogy a panel 9és 10-es Pin-jén vezérelje a motorvezérlőt.

A motorvezérlő egy dupla H-hidas DC Léptető motor vezérlő (L298N vezérlőmodullal). Ez 2db léptetőmotor vezérlésére képes. Jelen konfigurációban még csak egy lineáris motort használunk PWM jellel irányítva, de későbbiek során egy automata állvány elkészítése is megvalósítható lesz, ami a szenzor(ok)at mozgatja vertikálisan.

A vezérlőkód maximum 50 karakter hosszú csomagokat képes fogadni.

10. Exportált fájl formátumok

A modellek exportálhatósága fontos a későbbi felhasználás és mobilitás szempontjából. Jelenleg a 3 leggyakrabban használt formátumot implementáltuk. Ezek felépítését és háttérét érdemes jobban is megismerni.

10.1. PLY (*Polygon File Format*)

Két változata van [35], egyik az ASCII a másik a bináris tárolás.

Az ASCII PLY [36] fájlok általános felépítése:

Fejléc, Csúcs lista, Oldal lista

A formátumot elsősorban 3D szkennerek adatainak tárolására találták ki. Színeket, átlátszóságot, a felületek normálisát, textúrák koordinátáit és konfidencia értékeket tárolhatunk benne. A formátum lehetővé teszi, hogy egy poligon két oldalát különböző tulajdonságokkal tudjuk ellátni.

10.2. OBJ (*3D Object Format*)

Az OBJ [37] egy geometriai definíció alapján kifejlesztett fájlformátum. A Wavefront Technologies tervezte az Advanced Visualizer animációs csomagja számára. A formátuma nyílt és elfogadott más 3D-s grafikus alkalmazások számára is. A legtöbb grafikusnak ez az általánosan használt formátuma, amelyet minden fejlett szerkesztőszoftver ismer.

A fájl egyszerű adat-formátum, amely egyedül a 3D geometriát figyeli. Tárolja az egyes vertexek helyzetét, textúráját, normálisát, felületét, oldalát, és egy listát a csúcsairól. A csúcsok az óramutató járásával ellentétes sorrendben szerepelnek. Nincs kötelező séma az OBJ fájlokhoz, ezért bárhova be lehet szűrni megjegyzéseket.

10.3. STL (*STereoLithography*)

Az STL [38] fájlformátum a legrégebbiek között szerepel, ami a sztereo litográfiai CAD szoftverek által lett létrehozva 3D feldolgozásra. Széles körben használják a gyors prototípus- és számítógéppel segített gyártási folyamatokban. 3D nyomtatók vezérlőszoftverei is ezt használják fel, mert ott csak ezekre az adatokra van szükség, és mert ebben a témakörben ez a legelterjedtebb formátum. Az STL [39] fájlok csak a felület geometriája írják le, mindenféle szín, textúra vagy más közös CAD modell tulajdonságok nélkül. Itt is jelen van az ASCII és bináris ábrázolás is. A bináris fájlok sokkal gyakoribbak, mivel sokkal kompaktabbak.

Az STL fájl tartalmazza a nyers, strukturálatlan háromszög felületeket, a normálisokat és csúcsokat (jobb sodrás szerinti szabály alapján) egy háromdimenziós derékszögű koordináta rendszerben. Az STL koordinátáknak pozitív számoknak kell lenniük, de nincsen meghatározott skála, így egy egység bármekkora lehet.

11. Tesztesetek

A tesztek során az alábbi 4 számítógépet használtuk:

	Típus	Processzor	Memória	Videokártya
Laptop_1	Dell Inspiron 5110	Intel i7-2630QM	8 GB	nVidia Geforce GT525M (1 GB)
Laptop_2	Asus N53SV	Intel i5-2430M	4 GB	nVidia Geforce GT540M (1 GB)
Desktop_1	GIGABYTE GA-Z87X-D3H	Intel i7-4770K 4 mag, 8 szál @ 3.9 GHz	8 GB	GIGABYTE GV-N770OC-2GD GTX770 2GB GDDR5 PCIE (2 GB)
Desktop_2	HP Z800	2x Intel Xeon X5650 12 mag, 24 szál @ 2.66 GHz	16 GB	ASUS Geforce GTX 770 DirectCU II OC (2 GB)

4. táblázat Teszt konfigurációk

A rendszerekre 64 bites Windows 8.1 operációs rendszert telepítettünk, valamint az alábbi szoftvereket:

- Kinect For Windows Runtime 1.8 szabadon terjeszthető futtatókörnyezet
- Kinect Toolkit dll-ek:
 - KinectFusion180_32.dll
 - KinectFusion180_64.dll
 - Microsoft.Kinect.Toolkit.dll (Saját fordítású 32 és 64 bites verzió)
 - Microsoft.Kinect.Toolkit.Fusion.dll (Saját fordítású 32 és 64 bites verzió)
- PCL 1.7.0 (Trunk alapján saját fordítású 32 és 64 bites verzió) dll-ek:
 - pcl_gpu_containers_release.dll
 - pcl_gpu_kinfu_release.dll
- Nvidia CUDA SDK 5.5 szabadon terjeszthető dll-ek:
 - cudart32_55.dll
 - cudart64_55.dll

A Kinect SDK részét képező Kinect Studio szoftver segítségével előre definiált tartalmú jeleneteket rögzítettünk, amelyeket felhasználtunk a teszteléshez (lásd 7. táblázat, valamint 8. táblázat). A rekonstruált háromszög alapú poligonhálót minden esetben sikeresen elmentettük színek nélkül - vagy ahol külön jeleztük színekkel - ply formátumban, és megtekintettük a Meshlab nevű programban. A szkennelési távolság 0,4 métertől 2 méterig terjedt, az intervallumon kívül eső képpontokat a modellbe építéskor eldobtuk. A képek első oszlopa az RGB szín-, a második a mélységi adatfolyamot ábrázolja színekkel megjelenítve. Az egyik teszt során készült képen a harmadik oszlopban látható az objektum felszínét ábrázoló, a térfogatmodellből sugárkövetéssel kinyert kép (lásd 45. ábra).

A különböző fényviszonyok hatását is teszteltük a szkennelések során. Azt tapasztaltuk, hogy a közvetlen, erős napfényt leszámítva a mélységi rekonstrukciót a megvilágítás nem befolyásolta. A színes rekonstrukció esetén a gyenge vagy inhomogén megvilágítás színillesztési problémákat okozott egyes modelleknél.

A szkennelt objektumok alakja és fényvisszaverő/átengedő tulajdonságai jelentősen befolyásolták a róluk alkotott modellek minőségét. A matt sötét, csillogó vagy átlátszó tárgyak nem verték vissza megfelelően az infra sugarakat, ezért az ilyen tulajdonságokkal rendelkező objektumok szkennelésére az RGBD technológia a legtöbb esetben nem, vagy csak jelentős mérési pontatlanságok mellett volt képes. Azt tapasztaltuk, hogy a forgató zsámoly használata esetén a kezdőpozíció megválasztása is befolyásolhatja a modell minőségét.

Apróbb tárgyak szkennelése kis felbontásoknál nehézségekbe ütközött, mivel a felszínükről nem érkezett elegendő adat a rekonstrukcióhoz. Ezt a felbontás növelésével vagy a követést segítő tárgyak hozzáadásával küszöböltük ki.



45. ábra Pontossági mérés egy téglatesten

12. Eredmények értékelése

Az általunk meghatározott szkennelési tartományon belül jó minőségben, az elvárásainknak megfelelően sikerült digitalizálni a tárgyakat, embereket. Pontosság tekintetében a szenzor maximális pontosságát kihasználva fél méteres távolságból vizsgálva, a paramétereket jól megválasztva, szinte minden esetben 1cm alatt volt. (lásd 5. táblázat). Ezt a mozgás során az algoritmus a különböző szögekből nyert információkkal pontosította. Az algoritmussal a tesztelés során $512 \times 512 \times 512$ -es voxel számnál, 512 voxel/méter felbontásnál másodpercenként legalább 20-24 képkockát dolgoztunk fel számítógép függően, így közel valós időben tudtuk építeni a modellünket. A csúcsteljesítményű asztali számítógépeken azonos felbontásnál sikerült elérni a teljesen valós idejű, 30 FPS sebességű integrációt. Az exportálás több elterjedt fájlformátumba minden esetben sikeres volt. A beállításokat, meta adatokat egy meghatározott struktúrájú xml-fájlba mentettük a későbbi felhasználás megkönnyítése érdekében. Több 3D modellező szoftverbe (például: Meshlab, Blender) is sikeresen importáltuk az objektumokat. Mind fix kamera és modell forgatás, mind csak a kamera mozgásával és ezek ötvözésével sikeres szkenneléseket hajtottunk végre.

Teszt sorszám	Mesh mérete (m)			Gyakorlati pontosság, eltérés (m)			Eltérés százalékban		
	X	Y	Z	X	Y	Z	X	Y	Z
34	0,231	0,402	0,077	0,003	0	0,012	1,30%	0,00%	15,58%
35	0,231	0,4	0,088	0,003	0,002	0,001	1,30%	0,50%	1,14%
36	0,232	0,406	0,081	0,002	0,004	0,008	0,86%	-0,99%	9,88%
37	0,232	0,401	0,08	0,002	0,001	0,009	0,86%	0,25%	11,25%
38	0,231	0,407	0,084	0,003	0,005	0,005	1,30%	-1,23%	5,95%
39	0,23	0,408	0,085	0,004	0,006	0,004	1,74%	-1,47%	4,71%
40	0,228	0,404	0,084	0,006	0,002	0,005	2,63%	-0,50%	5,95%
41	0,232	0,403	0,075	0,002	0,001	0,014	0,86%	-0,25%	18,67%
42	0,23	0,404	0,081	0,004	0,002	0,008	1,74%	-0,50%	9,88%
43	0,232	0,407	0,084	0,002	0,005	0,005	0,86%	-1,23%	5,95%
44	0,23	0,402	0,085	0,004	0	0,004	1,74%	0,00%	4,71%

5. táblázat Eltérések, pontosság

13. Konklúzió

Az irodalomkutatás elvégzése során megismerkedtünk a különböző 3D ábrázolási és rekonstrukciós és megjelenítő módszerekkel, valamint a célkitűzésünkhöz hasonló funkcionalitást megvalósító rendszerekkel. A saját rendszerünk tervezése során figyelembe vettük ezek előnyeit és hátrányait, amiket felhasználva kialakítottuk a saját megoldásunkat. A vizsgált RGBD kamerák közül a Microsoft Kinect két változatát választottuk azok más rendszerekkel tanúsított magas kompatibilitása miatt. Mivel napjainkra már elterjedt az említett szenzorok használata, ezért az alkalmazásukról sok tapasztalat gyűlt össze. A Microsoft Kinect a legtöbb alternatívához képest olcsó és megbízható, normalizálás után megfelelő minőségű adatokat biztosít a rekonstrukcióhoz.

A megismert elméleti algoritmusok közül az ICP-t választottuk annak sebessége, robosztussága, jól dokumentáltsága és elterjedt használata miatt. Az ICP-t implementálás könyvtárak közül a Kinect Fusion-t és a Point Cloud Library-t használtuk fel, mivel a szakirodalom alapján, ezek bizonyultak a legmegfelelőbbnek a célkitűzéseink megvalósításához.

A programunk jelenlegi verziója az említett keretrendszerek adatszerkezeteit és algoritmus implementációit alkalmazza, de a rekonstrukció teljes folyamata csak a Kinect Fusion használata esetén járható végig. Az alkalmazás a szenzor RGB és mélységi adatfolyamából volumetrikus modellt épít, ebbe integrálja szekvenciálisan az új képkockákat. A modellt sugárvetéssel megjeleníti. Igény szerint a modellen felszín rekonstrukciót végez, a belőle előálló háromszöghálót ply, stl vagy obj formátumú fájlba exportálja. Az egy kamerás megoldások közül a dinamikus és a statikus módszereket egyaránt támogatja, az irodalomkutatásban említett szenzorok közül a Kinect for Windows és Kinect Xbox 360 eszközökkel kompatibilis. Fontos megjegyezni, hogy a Kinect Xbox 360-hoz kiadott változata jogi megkötések miatt végfelhasználói környezetben nem használható.

Kialakítottunk egy Arduino alapú, elektronikusan forgatható zsámolyt is a szkennelés automatizálhatóságának érdekében, amit az alkalmazásunkból virtuális COM porton keresztül tudunk vezérelni. A zsámoly segítségével lehetővé válik a modellek 360°-os beolvasása anélkül, hogy a kamerát a szkennelt tárgy körül körbevinnénk. Ezáltal a szkennelés folyamata kényelmesebbé tehető, csökkenthető annak minimális időigénye, valamint a kézi szkennelés során jelentkező hibák jelentős része kiküszöbölhető.

A rendszerünk több szálal feldolgozást alkalmaz, és képes a voxel alapú színes rekonstrukcióra. A végleges verzióban szeretnénk a Kinect-en kívül más RGBD szenzorokat is támogatni. A későbbiekben többszenzoros módszerekkel is lehetne biztosítani a 360°-os látószöveget, PCL használata esetén akár vegyes típusú szenzor konfigurációkkal is. Ebben az esetben a szenzorok fix helyzetben, statikus módon lennének elhelyezve.

14. Mellékletek

Mikrokontroller	ATmega328
Üzemi feszültség	5V
Bemeneti feszültség (ajánlott)	7-12V
Bemeneti feszültség (limitek)	6-20V
Digitális I / O Pins	14 (ebből 6 nyújtanak PWM kimenet)
Analóg bemenet Pins	6
DC áram I / O Pin	40 mA
Egyenáram 3.3V Pin	50 mA
Flash memória	32 KB (ATmega328), amelyből 0,5 KB által használt bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

6. táblázat Arduino Uno specifikációja

Jelenet sorszám	Távolság	Tárgy	Méret	Szög	Forgatás sebessége	Megjegyzés
1	2 m	Labda	~19-20 cm átmérő téglatest: $4,7 \times 8,65 \times 14,8$ henger: 6,6 cm átmérő $\times$ 20 cm magas	90° jobbra	60 (default)	Nem lehet szkennelni, mert túl kicsi a forgótest
2	0,5 m	Labda + segéd tárgyak(téglatest, henger)	~19-20 cm átmérő	0°	70	
3	0,5 m	Labda + segéd tárgyak(téglatest, henger)	20 cm	0°	60	
4	0,5 m	téglatest	$23,4 \times 8,9 \times 40,2$ cm	0°	60	
5	0,5 m	téglatest	$23,4 \times 8,9 \times 40,2$ cm	0°	120	
6	0,5 m	téglatest	$23,4 \times 8,9 \times 40,2$ cm	0°	15	
7	1,4 m	személy	166 cm magas, 9,7 cm a jobb könyök, 40,4 cm a bal nadrág elől	90° balra	60	
8	1,4 m majd 0,7 m	személy	166 cm magas, 9,7 cm a jobb könyök szélessége, 40,4 cm a bal nadrág alja elől	90° balra	60	

9	1,5 m majd 0,6 m	személy	166 cm az orr aljáig, 181cm hajjal	90° balra	60 majd 0	Kézi
10	0,75 m majd alatta	apróbb tárgyak	kis súly: 12,53 cm nagy súly: 15,8 cm átmérő	90° balra	60 majd 0 és kézi	alsó doboz éleinél, mert hasa van 11,9 × 15 × 37,4 cm felső téglatest 23,4 × 8,9 × 40,2 cm labda ~19-20 cm átmérő
11	0,75 m majd alatta	apróbb tárgyak	kis súly: 12,53 cm nagy súly: 15,8 cm átmérő	90° balra	0 és kézi	alsó doboz éleinél, mert hasa van 11,9 × 15 × 37,4 cm felső téglatest 23,4 × 8,9 × 40,2 cm labda ~19-20 cm átmérő
12	70 - 40 cm	monitor(ok)	42,3 × 36,5 cm	0°	kézi	

7. táblázat Kinect Studioval előre felvett teszt jelenetek

Teszt sorszám	Gép_ID	Jelenet sorszám	Verzió	Min (m)	Max (m)	Eltolás (m)	Szenzor	Szín	Simítás	Integráció súlya
1	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	3x3 0,02	150
2	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	150
3	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	200
4	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	200
5	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	200
6	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	300
7	Desktop_2	4	0.152	0,4	1	0,4	Win	nem	-	500
8	Desktop_2	4	0.152	0,4	1	0,4	Win	igen	-	500
9	Desktop_2	4	0.152	0,4	1	0,4	Win	igen	-	150
10	Desktop_2	4	0.152	0,4	1	0,4	Win	igen	5x5 0,32	150
11	Desktop_2	4	0.152	0,4	1,3	0,4	Win	igen	-	150
12	Desktop_2	10	0.152	0,4	1	0,4	Win	igen	3x3 0,02	150
13	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	150
14	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	200
15	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	200
16	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	200
17	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	300
18	Desktop_2	10	0.152	0,4	1,7	0,4	Win	nem	-	500
19	Desktop_2	10	0.152	0,4	1,7	0,4	Win	igen	-	500
20	Desktop_2	10	0.152	0,4	1,7	0,4	Win	igen	-	150
21	Desktop_2	10	0.152	0,4	1,7	0,4	Win	igen	5x5 0,32	150
22	Desktop_2	10	0.152	0,4	1,7	0,4	Win	igen	-	150
23	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	3x3 0,02	150
24	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	150
25	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	200
26	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	200

27	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	200
28	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	300
29	Desktop_2	8	0.152	0,4	2	1,6	Win	nem	-	500
30	Desktop_2	8	0.152	0,4	2	1,6	Win	igen	-	500
31	Desktop_2	8	0.152	0,4	2	1,6	Win	igen	-	150
32	Desktop_2	8	0.152	0,4	2	1,6	Win	igen	5x5 0,32	150
33	Desktop_2	8	0.152	0,4	2	1,6	Win	igen	-	150
34	Laptop_2	4	0.152	0,4	1	0,6	360	nem	3x3 0,02	150
35	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	150
36	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	200
37	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	200
38	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	200
39	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	300
40	Laptop_2	4	0.152	0,4	1	0,6	360	nem	-	500
41	Laptop_2	4	0.152	0,4	1	0,6	360	igen	-	500
42	Laptop_2	4	0.152	0,4	1	0,6	360	igen	-	150
43	Laptop_2	4	0.152	0,4	1	0,6	360	igen	5x5 0,32	150
44	Laptop_2	4	0.152	0,4	1,3	0,6	360	igen	-	150
45	Desktop_1	8	0.152	0,4	2	0,7	360	nem	3x3 0,02	150
50	Desktop_1	8	0.152	0,4	2	0,7	360	nem	-	300
51	Desktop_1	8	0.152	0,4	2	0,7	360	nem	-	500
52	Desktop_1	8	0.152	0,4	2	0,7	360	igen	-	500
53	Desktop_1	8	0.152	0,4	2	0,7	360	igen	-	150

8. táblázat Mérési paraméterek és tesztesetek

Teszt sorszám	Voxel sűrűség	Motor	Szkenelési tér méretei (voxel)			Szkenelési tér méretei (m)			GPU/CPU
			X	Y	Z	X	Y	Z	
1	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
2	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
3	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
4	640	Fusion	384	384	256	0,6	0,6	0,4	GPU
5	640	Fusion	256	384	256	0,4	0,6	0,4	GPU
6	512	Fusion	384	512	384	0,75	1	0,75	GPU
7	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
8	640	Fusion	384	512	384	0,6	0,8	0,6	GPU
9	512	Fusion	384	512	384	0,75	1	0,75	GPU
10	512	Fusion	384	512	384	0,75	1	0,75	GPU
11	512	Fusion	512	512	512	1	1	1	GPU
12	512	Fusion	512	512	512	1	1	1	GPU
13	512	Fusion	512	384	384	1	0,75	0,75	GPU
14	512	Fusion	512	384	384	1	0,75	0,75	GPU
15	640	Fusion	384	384	256	0,6	0,6	0,4	GPU
16	640	Fusion	256	384	256	0,4	0,6	0,4	GPU

17	512	Fusion	384	512	384	0,75	1	0,75	GPU
18	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
19	640	Fusion	384	512	384	0,6	0,8	0,6	GPU
20	512	Fusion	384	512	384	0,75	1	0,75	GPU
21	512	Fusion	384	512	384	0,75	1	0,75	GPU
22	512	Fusion	512	512	512	1	1	1	GPU
23	256	Fusion	512	384	384	2	1,5	1,5	GPU
24	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
25	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
26	640	Fusion	384	384	256	0,6	0,6	0,4	GPU
27	640	Fusion	256	384	256	0,4	0,6	0,4	GPU
28	512	Fusion	384	512	384	0,75	1	0,75	GPU
29	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
30	640	Fusion	384	512	384	0,6	0,8	0,6	GPU
31	512	Fusion	384	512	384	0,75	1	0,75	GPU
32	512	Fusion	384	512	384	0,75	1	0,75	GPU
33	512	Fusion	512	512	512	1	1	1	GPU
34	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
35	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
36	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
37	640	Fusion	384	384	256	0,6	0,6	0,4	GPU
38	640	Fusion	256	384	256	0,4	0,6	0,4	GPU
39	512	Fusion	384	512	384	0,75	1	0,75	GPU
40	512	Fusion	256	384	256	0,5	0,75	0,5	GPU
41	640	Fusion	384	512	384	0,6	0,8	0,6	GPU
42	512	Fusion	384	512	384	0,75	1	0,75	GPU
43	512	Fusion	384	512	384	0,75	1	0,75	GPU
44	512	Fusion	512	512	512	1	1	1	GPU
45	256	Fusion	512	384	384	2	1,5	1,5	GPU
50	256	Fusion	512	512	512	2	2	2	GPU
51	384	Fusion	768	512	512	2	1,333	1,333	GPU
52	384	Fusion	768	512	512	2	1,333	1,333	GPU
53	384	Fusion	768	640	640	2	1,667	1,667	GPU

9. táblázat Használt beállítások

Teszt sorszám	Meshlab info		
	Pontok száma a modellben	Pontok száma a meshben	Háromszögek száma a meshben
34	732963	732963	244321
35	615330	615330	205110
36	640407	640407	213469
37	864816	864816	288272
38	891162	891162	297054
39	714606	714606	238202

40	604275	604275	201425
41	1059825	1059825	353275
42	694113	694113	231371
43	774825	774825	258275
44	778941	778941	259647
45	1516908	1516908	505636
50	1360584	1360584	453528
51	4030956	4030956	1343652
52	3198879	3198879	1066293
53	4127847	4127847	1375949

10. táblázat Meshlabbal mért eredmények

Teszt sorszám	Elméleti maximális pontosság (mm)			Fizikai méret (m)		
	X	Y	Z	X	Y	Z
34	1,953125	1,953125	1,953125	0,234	0,402	0,089
35	1,953125	1,953125	1,953125	0,234	0,402	0,089
36	1,953125	1,953125	1,953125	0,234	0,402	0,089
37	1,5625	1,5625	1,5625	0,234	0,402	0,089
38	1,5625	1,5625	1,5625	0,234	0,402	0,089
39	1,953125	1,953125	1,953125	0,234	0,402	0,089
40	1,953125	1,953125	1,953125	0,234	0,402	0,089
41	1,5625	1,5625	1,5625	0,234	0,402	0,089
42	1,953125	1,953125	1,953125	0,234	0,402	0,089
43	1,953125	1,953125	1,953125	0,234	0,402	0,089
44	1,953125	1,953125	1,953125	0,234	0,402	0,089
45	3,90625	3,90625	3,90625	1,66	N/A	N/A
50	3,90625	3,90625	3,90625	1,66	N/A	N/A
51	2,604167	2,604167	2,604167	1,66	N/A	N/A
52	2,604167	2,604167	2,604167	1,66	N/A	N/A
53	2,604167	2,604167	2,604167	1,66	N/A	N/A

11. táblázat Mérés eredmények

15. Irodalomjegyzék

- [1] P. N. Interaction, „PrimeSense_3Dsensors,” [Online]. Available: http://www.primesense.com/wp-content/uploads/2012/12/PrimeSenses_3DsensorsWeb.pdf. [Hozzáférés dátuma: 05. 11. 2013.].
- [2] P. LTD, „Technology,” PrimeSense LTD, [Online]. Available: <http://www.primesense.com/solutions/technology/>. [Hozzáférés dátuma: 05. 11. 2013.].
- [3] Asus, „Asus Xtion PRO - Overview,” [Online]. Available: http://www.asus.com/Multimedia/Xtion_PRO/#overview. [Hozzáférés dátuma: 05. 11. 2013.].
- [4] Asus, „Asus Xtion PRO Live - Overview,” [Online]. Available: http://www.asus.com/Multimedia/Xtion_PRO_LIVE#overview. [Hozzáférés dátuma: 05. 11. 2013.].
- [5] Microsoft, „MSDN - Kinect for Windows Sensor,” Microsoft, [Online]. Available: <http://msdn.microsoft.com/en-us/library/hh855355.aspx>. [Hozzáférés dátuma: 07. 11. 2013.].
- [6] M. Viager, Analysis of Kinect for mobile robots, Denmark: Technical University of Denmark, 2011.
- [7] „Depth Sensors Comparison,” [Online]. Available: http://wiki.ipisoft.com/Depth_Sensors_Comparison. [Hozzáférés dátuma: 05. 11. 2013.].
- [8] A. Nüchter, 3D Robotic Mapping, Springer, 2009.
- [9] 3. T. laboratórium, „3D Tech laboratórium,” [Online]. Available: <http://www.3d-tech.hu/>. [Hozzáférés dátuma: 07. 11. 2013.].
- [10] N. University, „Applied Geometric Representation and Computation,” Northeastern University, [Online]. Available: <http://www.ccs.neu.edu/course/cs5350sp12/>. [Hozzáférés dátuma: 05. 11. 2013.].
- [11] M. Rozovits, „ASHAQ - Automatic Shape Aquisition,” [Online]. Available: http://iar.nik.uni-obuda.hu/2010_2011/ashaq/index.php?page=info. [Hozzáférés dátuma: 05. 11. 2013.].
- [12] H. Wang és J. Hu, „Active stereo method for three-dimensional shape measurement,” *Optical Engineering*, %1. kötet51, %1. szám6, 2012.
- [13] D. L. Szirmay-Kalos, Számítógépes grafika, Budapest: ComputerBooks, 1999.
- [14] „Wikipedia - Polygon mesh,” [Online]. Available:

- http://en.wikipedia.org/wiki/Polygon_mesh. [Hozzáférés dátuma: 07. 11. 2013.].
- [15] „Wikipedia - Non-uniform rational B-spline,” [Online]. Available: http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline. [Hozzáférés dátuma: 07. 11. 2013.].
- [16] „Wikipedia - Point cloud,” [Online]. Available: http://en.wikipedia.org/wiki/Point_cloud. [Hozzáférés dátuma: 07. 11. 2013.].
- [17] „Point Cloud Library (PCL): Module octree,” [Online]. Available: http://docs.pointclouds.org/trunk/group__octree.html. [Hozzáférés dátuma: 07. 11. 2013.].
- [18] G. Orbán, „Sarokpont detektálás. Mesterséges Intelligencia Almanach,” BME, [Online]. Available: http://mialmanach.mit.bme.hu/erdekessegek/sarokpont_detektalas.. [Hozzáférés dátuma: 01 05 2013].
- [19] C. Harris és M. Stephens, „A combined corner and edge detector,” in *Alvey Vision Conference*, Plessey Research Roke Manor, United Kingdom, 1988.
- [20] E. Rosten és T. Drummond, „Fusing Points and Lines for High Performance Tracking,” in *ICCV*, Beijing , 2005.
- [21] D. G. Lowe, „Object Recognition from Local Scale-Invariant Features,” Computer Science Department, University of British Columbia, Vancouver, B.C., V6T 1Z4, Canada, 1999.
- [22] N. Amenta, S. Choi és R. K. Kolluri, „The Power Crust,” in *SMA '01*, ACM New York, NY, USA, 2001.
- [23] R. Kolluri, J. R. Shewchuk és J. F. O’Brien, „Spectral Surface Reconstruction from Noisy Point Clouds,” *Eurographics Symposium on Geometry Processing*, 2004.
- [24] M. Zoltán-Csaba, R. B. Rusu és M. Bee, „On Fast Surface Reconstruction Methods for Large and Noisy Point Clouds,” Kobe, Japan, 2009.
- [25] W. E. Lorensen és H. E. Cline, „Marching Cubes: A high resolution 3D surface construction algorithm,” *Computer Graphics*, %1. kötet21., 1987.
- [26] H. Hoppe, T. DeRose és T. Duchampy, „Surface Reconstruction from Unorganized Points,” *SIGGRAPH Computer Graphics*, %1. kötet26., %1. szám2., 1992.
- [27] M. Kazhdan, M. Bolitho és H. Hoppe, „Poisson Surface Reconstruction,” *Eurographics Symposium on Geometry Processing*, 2006.
- [28] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright és B. C. McCallum, „Reconstruction and Representation of 3D Objects with Radial Basis,” in *ACM*, New York, NY, USA, 2001.
- [29] B. Curless és M. Levoy, „A Volumetric Method for Building Complex Models from

- Range Images,” in *ACM*, New York, NY, USA, 1996.
- [30] M. Dingerkus, „kinect-mssdk-openni-pcl-bridge,” [Online]. Available: <https://github.com/mdkus/kinect-mssdk-openni-pcl-bridge>. [Hozzáférés dátuma: 15 05 2014].
 - [31] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux és R. Newcombe, „KinectFusion: Real-time 3D Reconstruction and Interaction,” in *UIST '11*, Santa Barbara, CA, USA, 2011.
 - [32] S. Rusinkiewicz és M. Levoy, „Efficient Variants of the ICP Algorithm,” Stanford University, Stanford, 2001.
 - [33] Arduino, „Arduino Uno,” Arduino, [Online]. Available: http://store.arduino.cc/eu/index.php?main_page=product_info&cPath=11_12&products_id=195. [Hozzáférés dátuma: 06. 11. 2013.].
 - [34] „Arduino Ide for Visual Studio,” [Online]. Available: <http://www.visualmicro.com/>. [Hozzáférés dátuma: 05. 11. 2013.].
 - [35] M. Domo, „The Stanford 3D Scanning Repository,” Stanford, [Online]. Available: <http://graphics.stanford.edu/data/3Dscanrep/>. [Hozzáférés dátuma: 05 11 2013].
 - [36] P. Bourke, „PLY - Polygon File Format,” University of Western Australia, [Online]. Available: <http://paulbourke.net/dataformats/ply/>. [Hozzáférés dátuma: 05. 11. 2013.].
 - [37] M. Reddy, „OBJ files,” [Online]. Available: <http://www.martinreddy.net/gfx/3d/OBJ.spec>. [Hozzáférés dátuma: 05. 11. 2013.].
 - [38] fabbers.com, „The StL Format: Standard Data Format for Fabbers,” [Online]. Available: <http://www.ennex.com/~fabbers/StL.asp>. [Hozzáférés dátuma: 07. 11. 2013.].
 - [39] C. K. Chua, K. F. Leong és C. S. Lim, *Rapid Prototyping: Principles and Applications* (2nd ed.), World Scientific Publishing Co, 2003, p. 237.
 - [40] „ReconstructMe,” PROFACTOR, [Online]. Available: <http://reconstructme.net/>. [Hozzáférés dátuma: 15 05 2014].
 - [41] N. Tisserand és N. Burrus, „Skanect,” Occipital, [Online]. Available: <http://skanect.occipital.com/>. [Hozzáférés dátuma: 15 05 2014].
 - [42] Microsoft, „Using Kinect for Windows with a Virtual Machine,” [Online]. Available: <http://msdn.microsoft.com/en-us/library/jj663795.aspx>. [Hozzáférés dátuma: 15 05 2014].

16. Ábrajegyzék

1. ÁBRA EGY MODELL, AMIT A SAJÁT SZOFTVERÜNKKEL KÉSZÍTETTÜNK EGY MANKÓS EMBERRŐL.	5
2. ÁBRA EGY EMBER SZKENNELÉSE	5
3. ÁBRA PRIMESENSE SZENZOR [1].....	6
4. ÁBRA PRIMESENSE SOC KAPCSOLATAI [2]	6
5. ÁBRA ASUS XTION PRO [3].....	7
6. ÁBRA ASUS XTION PRO LIVE [4].....	8
7. ÁBRA KINECT FOR WINDOWS [5].....	8
8. ÁBRA KINECT SZERKEZETI FELÉPÍTÉSE [5]	8
9. ÁBRA KINECT ÉRZÉKELÉSI TARTOMÁNYOK [5].....	9
10. ÁBRA KINECT KOORDINÁTA RENDSZERE [5].....	9
11. ÁBRA LÉZER SZKENNER MŰKÖDÉSE [11].....	13
12. ÁBRA STRUKTURÁLT FÉNNYEL TÖRTÉNŐ SZTEREO FELDOLGOZÁS [12].....	13
13. ÁBRA A SZÁMÍTÓGÉPES GRAFIKA ADATFOLYAM-MODELLJE [13].....	14
14. ÁBRA A HÁROMSZÖG HÁLÓK FELÉPÍTÉSÉT BEMUTATÓ ÁBRA. A CSÚCSOK ÉS ÉLEK OLDALAKAT ALKOTNAK, AZ OLDALAK POLIGONOKAT, A POLIGONOK FELSZÍNEKET. EZEKBŐL ÁLL ÖSSZE A 3D OBJEKTUM. [14].....	15
15. ÁBRA NURBS MODELL SZERKEZETI ÁBRA [15].....	16
16. ÁBRA EGY TÓRUSZ ALAKZAT PONTFELHŐ MODELLJE [16].....	16
17. ÁBRA EGY PONTFELHŐBŐL GENERÁLT KÜLTÉRI TÉRKÉP MODELL. [16]	17
18. ÁBRA A POINT CLOUD LIBRARY PCL_OCTREE MODULJA ÁLTAL OKTÁLIS FÁBA RENDEZETT PONTFELHŐ. [17].....	18
19. ÁBRA HARRIS DETEKTÁLÁS [18].....	19
20. ÁBRA FAST MŰKÖDÉSE [20]	20
21. ÁBRA DAVID LOWE ÁBRÁJA A SIFT MŰKÖDÉSÉRŐL [21]	21
22. ÁBRA EGY DELFINT ÁBRÁZOLÓ POLIGON MODELL [14].....	22
23. ÁBRA LÉZERES MÉRÉSBŐL ELŐÁLLÍTOTT PONTFELHŐ, A BELŐLE GENERÁLT POLIGON MODELL ÉS A HOZZÁ TARTOZÓ KÖZÉPTENGELY TRANSZFORMÁLT. [22].....	23
24. ÁBRA A POWERCRUST ALGORITMUSON ALAPULÓ MÓDSZEREK ÖSSZEHASONLÍTÁSA [23]23	23
25. ÁBRA EGY AZ EIGENCRUST ALGORITMUSSEL REKONSTRUÁLT OBJEKTUM. A 2 008 414 BEMENETI PONTBÓL 12 926 063 TETRAÉDER FELHASZNÁLÁSÁVAL 3 605 096 KIMENETI HÁROMSZÖG ÁLLT ELŐ, 437 PERC ALATT. [23]	23
26. ÁBRA EGY KONYHA 15 MILLIÓ PONTBÓL ÁLLÓ SZÜRKEÁRNYALATOS PONTFELHŐ MODELLJE, ÉS A BELŐLE ELŐÁLLÍTOTT FOLYTONOS HÁLÓ. [24].....	24
27. ÁBRA EGY SÁRKÁNYT ÁBRÁZOLÓ PONTFELHŐ, ÉS A BELŐLE RADIÁLIS BÁZISFÜGGVÉNNYEL ELŐÁLLÍTOTT REKONSTRUÁLT MODELL. [28]	25
28. ÁBRA 2D-3D LEKÉPEZÉS [13]	26
29. ÁBRA LEKÉPEZÉS [13].....	27
30. ÁBRA FELHASZNÁLÓI ESETEK.....	30
31. ÁBRA CSOMAG DIAGRAM.....	31
32. ÁBRA ÁLTALÁNOS MŰKÖDÉS FOLYAMAT DIAGRAMJA.....	32
33. ÁBRA A KERETRENDSZEREKKEL KAPCSOLATOT TEREMTŐ INTERFÉSZ OSZTÁLYOK FŐBB FÜGGVÉNYEI ÉS ADATTAGJAI.....	32
34. ÁBRA AZ ADATKERETEKET KEZELŐ OSZTÁLYOK ÉS KAPCSOLATAIK	33
35. ÁBRA A MODELLBE INTEGRÁLÁST VÉGZŐ OSZTÁLYOK ÉS KAPCSOLATAIK	34
36. ÁBRA METAADAT-OSZTÁLYOK ÉS KAPCSOLATAIK	35
37. ÁBRA A SCAN3D MŰKÖDÉSÉT SEGÍTŐ OSZTÁLYOK.....	36
38. ÁBRA KINECT FUSION PIPELINE [31].....	36

39. ÁBRA A KOORDINÁTA-RENDSZEREK KAPCSOLATA.....	38
40. ÁBRA EGY A SAJÁT ALKALMAZÁSUNKKAL KÉSZÍTETT, VERTEX SZÍNEKET HASZNÁLÓ MODELLBŐL GENERÁLT NAGY RÉSZLETESÉGŰ SZÍNES MESH.....	40
41. ÁBRA A FENTI MESH SZÍNEK NÉLKÜL RENDERELT VÁLTOZATA.	40
42. ÁBRA FORGÓZSÁMOLY 1.0.....	42
43. ÁBRA FORGÓZSÁMOLY 2.0.....	42
44. ÁBRA ARDUINO UNO [33].....	43
45. ÁBRA PONTOSSÁGI MÉRÉS EGY TÉGLATESTEN.....	47