



EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK

Hatékony eljárások mozgó objektumok valósídejű 3D-s rekonstrukciójához GPU segítségével

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Témavezető:

Csetverikov Dmitrij

egyetemi tanár

Algoritmusok és Alkalmazásai

Tanszék

Készítette:

Hapák József

Programtervező informatikus

MSc nappali tagozat, 2. évfolyam

Szoftvertechnológia szakirány

Budapest, 2012

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg (a támogatás száma TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

Tartalomjegyzék

1. Bevezetés	3
2. Dolgozat felépítése	3
3. Szoftver és hardver környezet	4
4. Feladat	6
5. Végrehajtási Futószalag (Pipeline)	7
5.1. Alkalmazott technológiák	8
5.2. Szegmentálás	9
5.3. Vizuális Burok	10
5.4. Simítás	12
5.5. Marching Cubes	12
5.6. Textúrázás	14
5.7. Árnyéktérképek	17
6. Tesztelés	18
7. Eredmények, további tervek	19
8. Köszönetnyilvánítás	21
9. Irodalomjegyzék	23

1. Bevezetés

Napjaink film és játékipara a technológia fejlődés hatására egyre magasabb minőségű számítógépes grafikai megoldásokat alkalmaz. Ennek a magas szintnek az eléréséhez megfelelő minőségű bemenő adatokra van szükség, amik még napjainkban is nagyrészt emberi munka által kerül előállításra. Ennek a módszernek természetes több szempont szerint is korlátai vannak.

Nagy szükség lenne egy teljesen automatikus 4D rekonstrukciós eljárásra, mely a rekonstruálandó színteret magas minőségben tudja digitalizálni. A feladat bonyolultsága miatt speciális stúdiót igényel. Még napjainkban is kevés ilyen stúdió került felépítésre. Még kevesebb készült kifejezetten tudományos kísérletek és a különböző alkalmazások előkészítése céljából.

Hazánkban a Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézetének Geometriai Modellezés és Számítógépes Látás Kutatólaborjában található egy ilyen stúdió. Dolgozatom témája megvizsgálni, hogy lehetséges-e a rögzített időben változó színtér valósi idejű rekonstrukciója. Illetve amennyiben ez kivitelezhetőnek bizonyul, akkor az ezt végrehajtó szoftverkomponenst megvalósítani a stúdió meglévő vezérlőszoftveréhez

2. Dolgozat felépítése

A dolgozat három részre tagolódik. Az első részben összefoglaljuk a stúdió felépítését, majd részletezzük a megoldandó problémát. A második részben megvizsgáljuk a megoldási lehetőségeket és ezek közül egyet bemutatunk. Végül összefoglaljuk az eredményeket és a további fejlesztési lehetőségeket.

3. Szoftver és hardver környezet

A Stúdió egy 5 méter átmérőjű, 3 méter magas fémváz, mely zöld színű vászonnal van takarva. A rekonstruálandó tér rögzítését 13 darab a JAI cég által fejlesztett CB-200GE típusú ipari kamera végzi. Ezek közül 12 egyenletesen elosztva helyezkedik el a terem oldalfalán váltakozó magasságban - hat kamera fentről, hat pedig lentől rögzíti a stúdió belső terét. Az utolsó kamera középen fent került felfogatásra. A kamerák 1624x1236 felbontású képeket küldenek Bayer kódolásban.

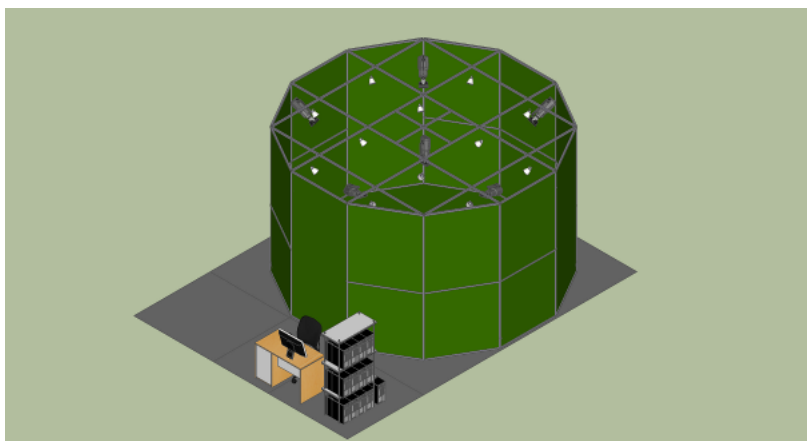


1. ábra. A stúdió

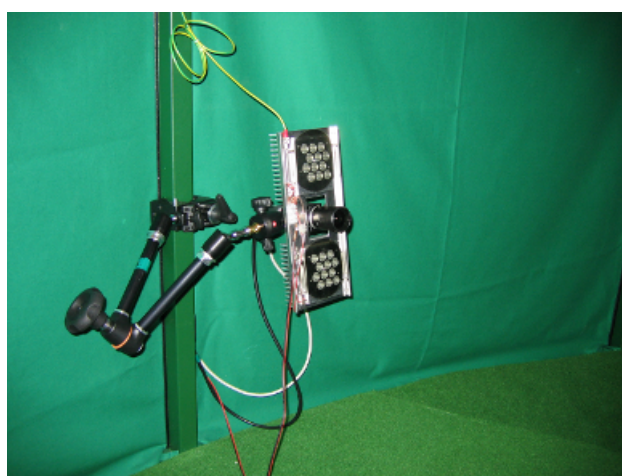
A kamerák által küldött nagy mennyiségű adat mentéséért összesen 7 számítógép felelős. Egy számítógépet kivéve mindegyikre két kamera csatlakozik. A nagy sávszélesség biztosítása érdekében az adatok különböző merevlemezekre kerülnek tárolásra. A számítógépeken Open SUSE operációs rendszer fut. A későbbi feldolgozás érdekében minden számítógép el van látva egy közepes teljesítményű videokártyával. A valós idejű rekonstrukciót végző PC egy nagy teljesítményű Fermi architektúrás GF110 típusú grafikus processzorral szerelt GeForce 580 GTX videokártyával rendelkezik. Az alkalmazott GPU 512 darab feldolgozó processzort tartalmaz, amelyek 32-esével 16 csoportokba szerveződnek. Az elérhető maximális számítási kapacitás 1500 Gflops. A kamerák és a számítógépek Gigabit Ethernet segítségével kerültek összekötésre. Erre a sávszélességre mind a rögzítés, mind a rekonstrukció folyamán szükség van.

A megfelelő minőségű képek előállítása érdekében a stúdió fontos részét képezi a megvilágítás. Túl kevés fény esetén a képek sötétek lesznek. Ha ezt hosszú expozíciós idővel próbáljuk korrigálni, akkor elmosott, részletszegény képeket kaphatunk. Túl erős megvilágítás használata esetén pedig a fényforráshoz közel eső részek telítődhetnek, a távolabbiak pedig sötétek maradnak a fényintenzitás négyzetes csökkenése végett.

A stúdió felülről néhány szórt fényforrással van megvilágítva. Ezen kívül minden kamera mellé fel van szerelve egy-egy LED-eket tartalmazó panel, a kamera által megvilágított tér bevilágítása céljából. Mivel a kamerák körben egyenletesen kerültek elosztásra, ezért adódik, hogy egymás fényforrásait is látják. A szemből világító erős fény elronthatja a felvételek minőségét. A probléma kiküszöbölése érdekében a kamerák két csoportra



2. ábra. A stúdió látványterve



3. ábra. A stúdióban található egy kamera, és a hozzá tartozó fényforrás

vannak osztva. Míg az egyik csoport felvételt készít, addig a másik csoport fényforrásai sötétek maradnak. Ennek a váltakozásnak az irányításáért, egy speciális mikrokontrollert tartalmazó vezérlőegység felelős. Ez a vezérlőegység végzi a kamerák szinkronizálását, és lehetővé teszi az egyes fényforrások külön-külön történő vezérlését. Programja számítógép segítségével könnyen módosítható. A kamerák két csoportra bontása minimális időeltolódást eredményez a felvételek között, de ez érdemben nem befolyásolja a rekonstrukció minőségét.

4. Feladat

A munka célja megvizsgálni, hogy megfelelően megválasztott hardver és szoftver technológiák alkalmazásával lehetséges-e a stúdióban rögzített alakzatok valósidejű rekonstrukciója és megjelenítése. A munka kiinduló pontja a már elkészített utófeldolgozáson alapuló offline működés volt. Egy tíz másodperces rögzített adatfolyam feldolgozása 30 percet vesz igénybe. Ennek az időnek a nagy részét a lemezműveletek teszik ki. Az valósidejű implementációban az offline működéssel ellentétben nincs szükség a felvételek rögzítésére, a rendelkezésre álló feldolgozási időt a lehető legjobb minőség elérése kell fordítani

A probléma megoldása jelentős számítási kapacitást igényel, az ezt biztosító grafikus processzorok napjainkra érték el a szükséges teljesítményt. Ezen oknál fogva a problémakör kis irodalommal rendelkezik és az alkalmazott hardver és szoftvermegoldások is különbözőek. Az alkalmazott stúdióban szokványos kamerák állnak rendelkezésre, más hardverkörnyezet, például mélységkamerák esetén más szoftveres megoldások is számításba kerülhetnek.

A megoldandó probléma lényegesen különbözik a film és játékiparban már rég alkalmazott Motion Capture¹ technológiáknál. Hiszen míg ez utóbbiak az alakzatokon elhelyezett markerek segítségével csak a mozgás követésére, úgynevezett ritka rekonstrukcióra alkalmasak, addig esetünkben komplett, sűrű geometria létrehozása a cél. A művelet elvégzéséhez nincs szükség markerek elhelyezésére, így lehetőség nyílik állatok, vagy akár füst rekonstrukciójára is.

A munka eredményét egy a stúdiót működtető már elkészült vezérlőprogramhoz integrálható kiegészítésként kellett megvalósítani. A vezérlőprogram már biztosította számunkra a képek rögzítését, és a kamerák kalibrációs adatait, ezekkel a fejlesztés során már nem kellett foglalkozni. További igény volt az eredmények egyszerű megjeleníthetősége a Visualisation Toolkit² segítségével.

¹http://en.wikipedia.org/wiki/Motion_capture

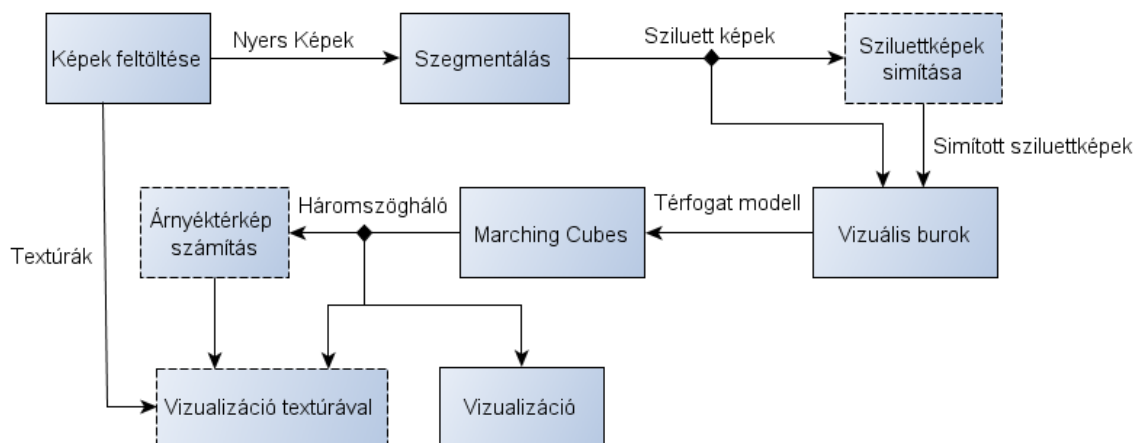
²<http://www.vtk.org/>

5. Végrehajtási Futószalag (Pipeline)

Mivel napkainkban a probléma megoldása által igényelt számítási teljesítményt csak a korszerű grafikus processzorok képesek biztosítani, ezért a megvalósítás során olyan algoritmusokat kellett választanunk, amelyek hatékonyan implementálhatóak sok magos környezetben. Ezeket az algoritmusokat egymás után fűzve kialakításra került egy végrehajtási futószalag, mely lehetővé teszi a valós idejű rekonstrukciót.

Az eljárás első lépése, amint a vezérlőszoftver megkapja minden kamerától az aktuális képkockát a képek feltöltése a videokártya memóriájába a további feldolgozás végett. Ezeket a képeket már a kamerák felvételeit fogadó gépek előfeldolgozzák: a Bayer kódolásból RGB szintérbe alakítják, és elvégzik a megfelelő átméretezést is. Ezután a képeket szegmentáljuk elválasztva a stúdió háttérét az előtértől. A szebb eredmény elérése érdekében igény esetén egy simítást is elvégezhetünk. Az eredményül kapott sziluett képekből a Vizuális Burok algoritmussal térfogat modellt állítunk elő. Ezután a Marching Cubes algoritmus alkalmazásával háromszöghálót generálunk, amelyet már a videokártya hatékonyan meg tud megjeleníteni. A pipeline utolsó opcionális lépése a kapott háromszögháló textúrával történő renderelése. Ez a lépés felbomlik úgynevezett shadowmapok (árnyék-térképek) előállítására és magára a textúrázásra.

A futószalag modulárisan került implementálásra. Annak tetszőleges része külön lefuttatható, a részeredmények kinyerhetők. Ezáltal lehetőség nyílik az egyes fázisok más módszerrel való megoldására, a megoldások teljesítményének, hatékonyságának összevetésére. Egyes fázisok opcionálissá tehetők, mint például a textúrázás vagy a sziluettképek simítása. Ezt az igényt szem előtt tartva az egyes fázisokat osztályokba szerveztük. Az egyes implementációknak meg kell valósítani az adott fázis interfészét.



4. ábra. A GPU munkamenet (a szaggatott vonallal jelölt fázisok opcionálisak)

Mivel az egyes fázisok nagy mennyiségű adatot dolgoznak fel, ezért a felesleges adatmozgatásokat minimalizálni kell. Ezért az egyes fázisok közvetlen használják egymás memóriaterületeit a végrehajtás során.

5.1. Alkalmazott technológiák

Mind grafikus, mind általános számítások futása során a központi egység (CPU) és a grafikus processzor (GPU) egyfajta szerver kliens szerepkörben állnak egymással. A jóval alacsonyabb számítási kapacitással rendelkező CPU-n futó úgynevezett gazdaprogram, egyszerű, főleg adatpárhuzamosságra alapozó feladatokkal bízta meg a nagy teljesítményű GPU-t. A gazdaprogram szokványos programozási nyelvek segítségével készíthető el. Esetünkben ez C++ volt mivel a komponenst beágyazó keretprogram is ezen a nyelven készült. A videokártyák programozása azonban speciális nyelvek, technológiák segítségével valósítható meg, melyek új nyelvi elemeket vezetnek be a hardver hatékony kihasználása érdekében. Ilyen nyelvi elem például a grafikában használatos mátrix, és vektorműveletek, vagy az adatpárhuzamosságot támogató szinkronizációs műveletek. Az általunk a valósidejű rekonstrukcióra használt videokártya programozására a Cuda, DirectCompute, GLSL, HLSL és az OpenCL technológiák segítségével lehetséges.

A DirectCompute és a HLSL csak Windows operációs rendszeren elérhető technológiák, így esetünkben nem voltak alkalmazhatóak.

A GLSL az OpenGL grafikus könyvtár saját C szerű programozási nyelve, mellyel a korszerű renderelési futószalagok egyes fázisait programozhatjuk. Ezt a technológiát a ma kapható hardverek széles köre magas szinten támogatja. Régebben általános számítási feladatok elvégzésére is alkalmazták, de az OpenGL-lel való szoros kapcsolata miatt ezt nehézkesen, a renderelési futószalaggal való trükközéssel, „hekkelésszerűen” lehetett megoldani.

Az OpenCL az utóbbi években jelentősen feltörekvő nyílt technológia, mely lehetővé teszi a sok és többmagos processzorok - legyen az központi feldolgozó egység (CPU), vagy grafikus processzor (GPU) - egy egységes felületen való hatékony kihasználását. Széleskörű támogatottságát a felhasználói programok írásakor kamatoztathatjuk, hiszen az otthoni multimédiás eszközök jelentős hányada támogatja. Felépítése hasonlít az OpenGL grafikus könyvtáréra, amivel magas fokú együttműködésre képes. A hatékony működés érdekében lehetőség van erőforrások megosztására a két technológia között. A CUDA-val ellentétben viszont - a technológia frissességéből adódóan - számottevően kisebb szakirodalom bázissal, a fejlesztést megkönnyítő könyvtárral és eszközzel rendelkezik jelenleg.

A CUDA az nVidia zárt platformja, mely lehetővé teszi a vállalat által tervezett grafikus processzorok általános számításokhoz való hatékony kihasználását. A cég a tudományos, és ipari felhasználáshoz külön termékvonalat tart fenn Tesla néven, melyek kifejezetten általános számítások elvégzésére készültek. A piacon való elsősége révén nagyobb felhasználói bázissal, kiforrottabb eszközkészlettel rendelkezik. Az OpenCL-hez hasonlóan képes együttműködni az OpenGL grafikus környezettel.

Választásunk végül a CUDA platformra esett annak magasabb szintű támogatottsága miatt, A futószalag minden fázisa, a textúrázás kivételével ezen a nyelven került

implementálásra. A megvalósításban kihasználjuk a OpenGL-lel való erőforrás megosztás lehetőségét, és igény esetén a rekonstrukció eredményét, egy már az OpenGL segítségével hatékonyan megjeleníthető Vertex Buffer Objectben adjuk át. A textúrázást a többi fázissal ellentétben GLSL-ben valósítottuk meg.

Gazdaprogram oldalán a matematikai számításokat az OpenCV könyvtár segítségével végeztük el.

A továbbiakban a futószalag egyes fázisait mutatjuk be részletesebben.

5.2. Szegmentálás

A szegmentálás feladata a beérkező képek szétválasztása háttérre és előtérre. Az eljárás végeredménye egy az eredetivel megegyező méretű bináris kép, melyben a fekete pixelek háttérhez, a fehér pixelek pedig az előtérhez tartoznak. Több különböző elvű algoritmus létezik a probléma megoldására, ezek közül az egyik leghatékonyabb, legjobban párhuzamosítható a háttér alapú szegmentálás.

Első lépésként a kalibrált kamerák segítségével rögzítjük az üres stúdiót - ezek az úgynevezett háttérképek. A feldolgozás során meghatározzuk az aktuális képkocka pixelei, és a háttérképek közötti abszolút különbséget az RGB szintérben. Ha a különbség átlépi a paraméterként megadott küszöbértéket, akkor a pixel az előtérhez tartozik, egyébként pedig a háttérhez. Jelöljük i -vel az előtér, bg -vel a háttér, s -sel pedig a sziluett kép aktuálisan feldolgozás alatt álló pixelét. Ekkor a szegmentálás a következő formula alapján hajtódik végre.

$$d = |i_r - bg_r| + |i_g - bg_g| + |i_b - bg_b|$$

$$s = \begin{cases} 255 & \text{ha } d > threshold \\ 0 & \text{egyébként} \end{cases}$$

Látható, hogy az algoritmus nagyon jól párhuzamosítható sok feldolgozó egységgel rendelkező architektúrákon, mint például a grafikus processzorok, hiszen az egyes pixelek feldolgozása egymástól függetlenül zajlik. Az implementációban a bináris képeket bájtok segítségével ábrázoljuk, 0 értékkel jelöljük a háttér 255-el az előtér színeit.

A feldolgozandó képeket egy három dimenziós tömbben tároljuk, az egyes kamerák képeit egymás mögött. A szegmentálás első lépéseként a központi memóriában lévő tömböt átmásoljuk a videó memóriába. Ez sajnos egy meglehetősen időigényes feladat a PCI-Express busz alacsony áteresztőképessége miatt. Egy-egy pixel szegmentálását a grafikus processzor egy-egy feldolgozó processzora végzi el. A feldolgozás gyorsítása érdekében az adatokat textúraként, csak olvasható módon érjük el. Ebben az esetben a futtató környezetnek lehetősége van a grafikus processzor textúra mintavételezőire támaszkodni,



5. ábra. A feldolgozásra beérkező képek

melyek nagy sebességgel képesek egyszerre több adatot beolvasni és gyorsítótárazni.

Mivel az algoritmus magja minden pixelre lefut, ezért műveletigénye a képpontok számában lineáris.

5.3. Vizuális Burok

A szegmentált képekből a Vizuális Burok (Visual Hull) nevű eljárással állítunk elő térfogat modellt. A térfogat modell egy 3 dimenziós adatstruktúra, lényegében a 2 dimenziós képek kiterjesztése. Pixelek (picture elements) helyett úgynevezett voxeleket (volumetric pixel) tartalmaznak, mely a reprezentált térfogat egy-egy pontjáról tárolnak információkat. Például orvosi felhasználás terén az emberi szövetek sűrűségértékeit. Esetünkben egy valószínűségi érték kerül tárolásra, mely jellemzi, hogy a rekonstruálandó objektum, milyen valószínűséggel található az adott voxelben.

Az eljárás bemenő adatként kalibrált kamerákból származó sziluett képeket vár, tehát ismernünk kell az egyes kamerák egymáshoz képesti orientációját és pozícióját. Az ismert transzformációk alapján az egyes sziluettképeket visszavetíthetjük a rekonstruálandó térbe. A visszavetítés a térből, egy „sziluett alapú gúlát” metsz ki. Minden kamerára elvégezve a műveletet, megkapjuk a rekonstruálandó tér durva modelljét. Az eljárást működési módszeréből adódóan térfaragásnak (space carving) is nevezik, hiszen iteratívan a sziluettképek mentén „kifaragja” a rekonstruálandó alakzatot a térből.

Az eljárás korlátai közé tartozik a szintér azon konkáv mélyedései, amelyek nem jelennek meg egyetlen kamera felvételen sem. Ezeket az algoritmus nem képes helyesen



6. ábra. A szegmentálás eredménye

rekonstruálni.

Gyakorlatban az eljárás végrehajtása során bejárjuk a készítendő modell voxeleit (volumetric pixel). A voxel térbeli pozíciójának ismeretében, lehetőségünk van annak helyét meghatározni az egyes kamerák terében. Legyen K_i R_i és t_i az i -k kamera, forgatási mátrixai és pozíciója megfelelően. T a térfogatmodell indexteréből a rekonstruálandó térbe átvivő transzformáció. Ekkor a képpont koordinátái a szegmentált képen

$$c_i = K_i(R_i T[x, y, z]^T + t_i)$$

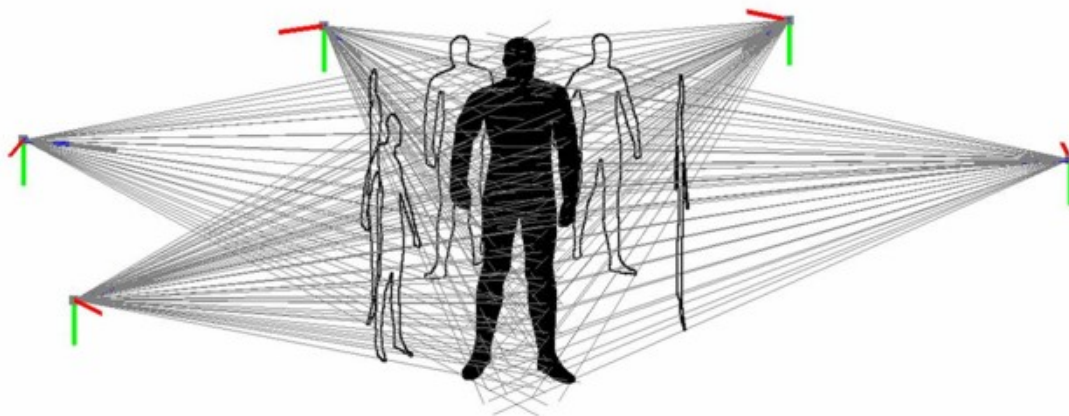
ahol x, y, z az aktuálisan feldolgozás alatt álló voxel indexei.

A kapott eredmény segítségével mintavételezzük a sziluettképet. A voxel végeredményben, a mintavételezett értékek minimumát kapja eredményül.

$$v(x, y, z) = \min_{1..n}(\text{tex2D}(S_i, c_i)),$$

ahol n a kamerák száma

Mivel a sziluettből kapott gúla a kamerától való távolság alapján szélesedik, ezért mintavételezési hibák jelentkezhetnek a szegmentált képek mintavételezése során. Mivel a mipmapping algoritmus alkalmazására teljesítménybeli korlátok miatt nincs lehetőség, ezért kompromisszumokat kell kötni. A szegmentált képekre a Vizuális Burok eljárás előtt opcionálisan alkalmazhatunk egy simítási fázist. Ezzel valamilyen szinten kiküszöbölhetjük a mintavételezési hibákat, a teljesítmény drasztikus csökkenése nélkül.



7. ábra. A Vizuális Burok algoritmus

Az algoritmus futás során a paraméterként megadott méretű térfogatmodellt járja be. Minden voxelben mintavételezés történik az egyes sziluettképekből. Az műveletigény tehát a képek számában, és a térfogatmodell méretével lineáris.

5.4. Simítás

A Vizuális Burok algoritmusban felmerülő mintavételezési problémák részleges kiküszöbölése végett a szegmentált képeken egy simítást hajtunk végre. A valósidejű implementációban ezt egy egyszerű szeparábilis dobozszűrő segítségével valósítjuk meg. Az egyes pixelek feldolgozását ebben az esetben is a grafikus processzor egy-egy végrehajtó processzora végzi. A szűrő mérete futásidőben nem módosítható, csak a teljes program újrafordításával. Ez lehetőséget ad a fordítónak, hogy optimalizációkat hajtson végre a generált kódon (ciklusok kibontása).

A szegmentált képeket csak olvasható módon textúráként érjük el. Ez esetben a grafikus processzornak lehetősége van a textúra mintavételezők alkalmazására, mely egyszerre több adatot képes elérni, mindezt gyorsítótárazva. Ez különösen előnyös a simítás szempontjából, hiszen a szűrés folyamán sokszor van szükségünk ugyanazokra az értékekre.

Műveletigény szempontjából az algoritmus a képpontok számában lineáris.

5.5. Marching Cubes

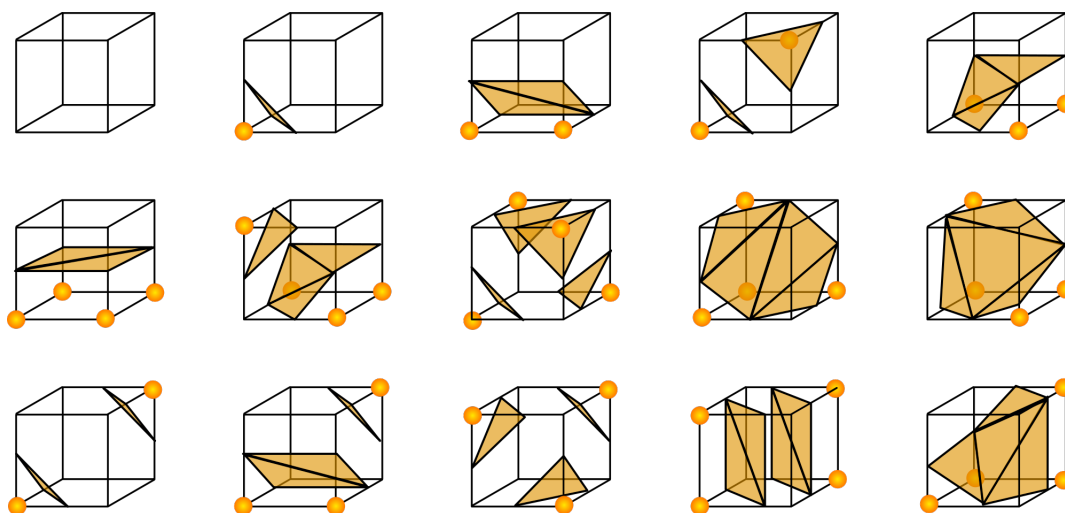
A Vizuális Burok algoritmus által előállított térfogatmodell a Marching Cubes algoritmus segítségével háromszöghálót állítunk elő. Mivel az eljárás meglehetősen népszerű a térfogatmodellek vizualizációja terén, ezért már több hatékony implementációval rendelkezik. Mi az nVidia SDK-ban található megoldást választottuk. Ennek oka, hogy meglehetősen egyszerű, de hatékony implementáció és könnyen testre szabható igényeinknek



8. ábra. Az elmosott sziluettképek

megfelelően.

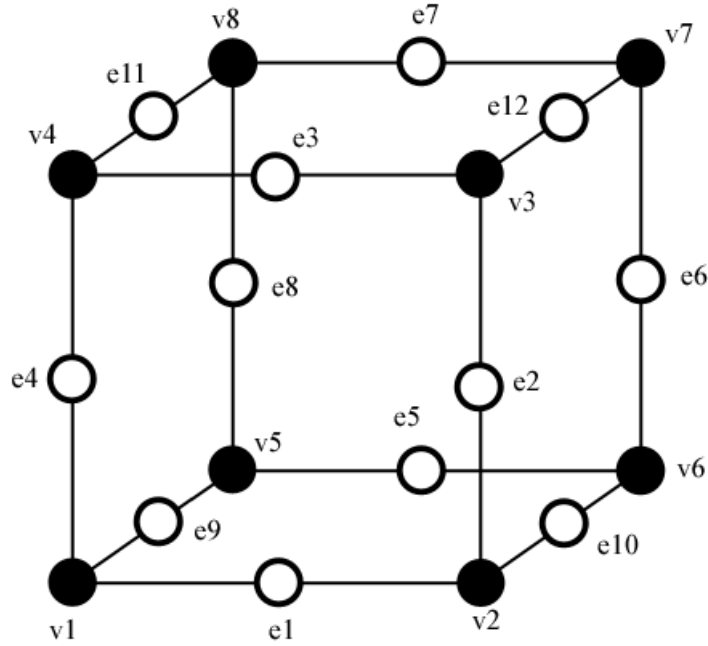
A végrehajtás során az algoritmus bejárja a térfogatmodellt. A voxeleket nyolcasával veti össze a megadott küszöbértékkel. Az összehasonlítás eredménye 8 logikai érték, mely a megfelelő voxeleket minősíti aszerint, hogy az objektumban van-e vagy sem. A 8 logikai érték, mint egy bájt segítségével két úgynevezett lookup táblát címzünk.



9. ábra. A Marching Cubes algoritmus különböző esetei

Az első tábla segítségével döntjük el, hogy a 8 voxel által alkotott egységkocka élei közül melyik metszi el az alakzat határvonalát, azaz melyiken helyezkednek el az végeredmény alkotó háromszögháló csúcpontjai. (Lásd 9. ábra). A második tábla pedig egy index lista,

mely a kapott vertexekből alkotja meg a poligonokat.



10. ábra. A voxelek, és a csúcspontok indexei

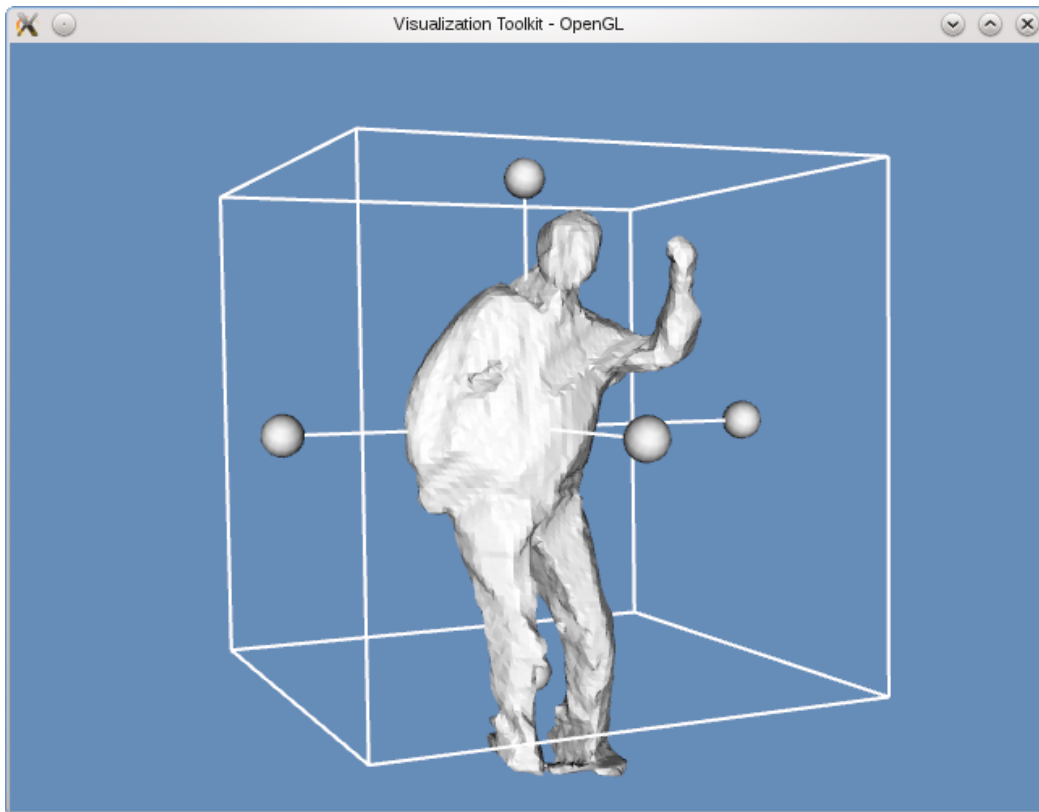
A csúcspontok végső elhelyezését a voxelek pozícióinak lineáris kombinációjával számítjuk ki a küszöbértéknek megfelelően. Legyen v_0 , v_1 a két voxel koordinátái, f_0 , f_1 a voxelekben mért érték, i pedig az aktuális küszöbérték. Ekkor a csúcspont koordinátái:

$$e = \frac{(i - f_0)v_1 + (f_1 - i)v_0}{(f_1 - f_0)}$$

Az algoritmus végeredményképp vertexek listáját generálja, melyek hármasával egy-egy poligont alkotnak. A végeredményt visszaolvashatjuk a videokártya memóriájából egy tömbbe, vagy az OpenGL közreműködést kihasználva egy Vertex Buffer Objectekben is eltárolhatjuk a további feldolgozás céljából.

5.6. Textúrázás

A munka első fázisa a valósídejű geometriai rekonstrukció volt. Mivel ezt sikerült megvalósítani, a generált geometria valósídejű textúrázása következett. Ezzel a problémával kapcsolatban napjainkban még meglehetősen kevés eredmény született. A létező megoldások, is általában speciális feltételek mellett biztosítanak értékelhető eredményt. Ezért a stúdióban tapasztalható körülményekhez saját módszereket dolgoztunk ki. Az offline verzióban alkalmazott textúra atlaszt előállító megoldás nem alkalmazható, annak nagy erőforrás költsége miatt. A valósídejű eredmény elérése érdekében közvetlenül a bejövő



11. ábra. A generált háromszögháló VTK-ban

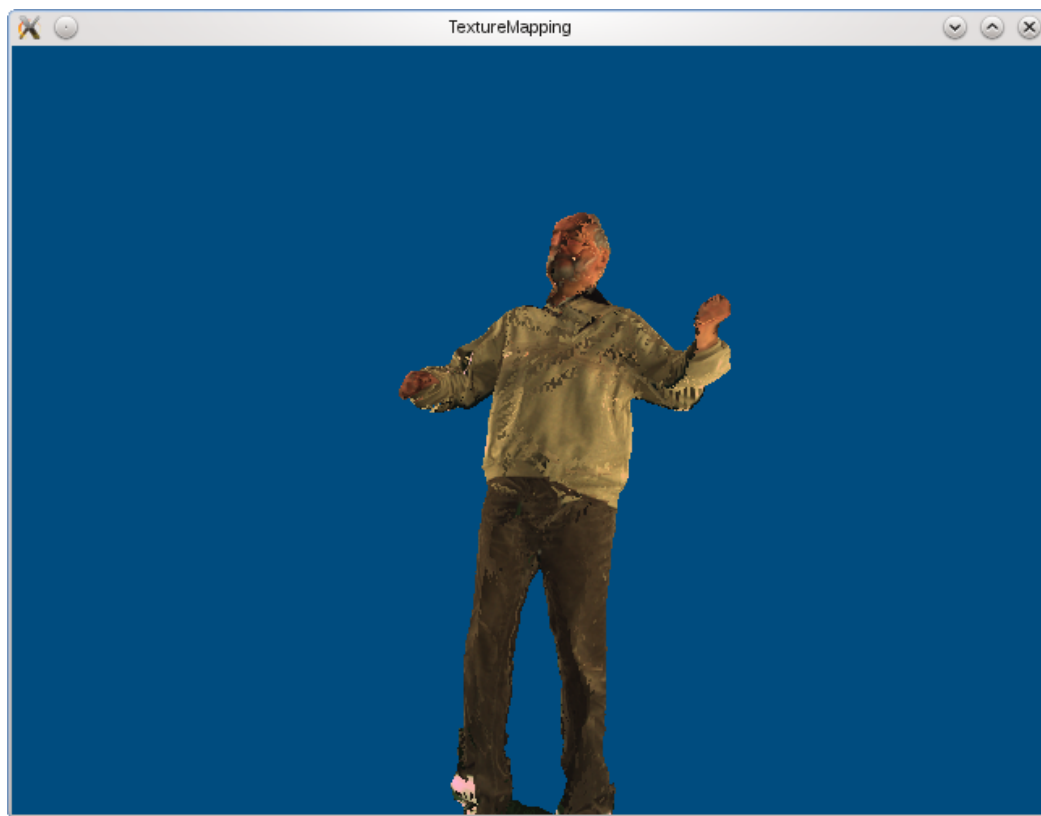
képekből kellett megoldanunk a feladatot. A textúrázást a renderelés alatt, fragment shader segítségével oldottuk meg GLSL nyelv használatával.

Egy pixel színének kiszámítása során első lépésben el kell döntenünk, hogy egy adott kamera egyáltalán látja az általa reprezentált térbeli pontot. Ezt a problémát az árnyéktérképek (shadow maps) segítségével oldja meg a program. A kamerákra – jelen esetben mint fényforrásokra tekintve – előállítjuk a megfelelő árnyéktérképeket minden geometria rekonstrukciós fázis végén. Ez esetben a takarási probléma az árnyékolási problémával analóg módon megoldható. Amely pixel „árnyékban” van azt a kamera nem látja.

A pixel meghatározására két módszer került kidolgozásra. Az egyik módszer robusztusabb, minden kamerák által látható pixelhez képes színinformációt rendelni, de az eredménye meglehetősen zajos. A másik megoldás jóval simább életszerűbb képminőséget biztosít, de működéséhez több kamera alkalmazása szükséges. Alapértelmezésben ez utóbbi módszert alkalmazzuk, de ha a textúrázás nem sikerülne visszaváltunk az elsőre. Mindkét esetben a láthatósági problémát az árnyéktérképek segítségével oldjuk meg.

Az első módszer kiszámítja a textúrázandó felület normálvektora és a kamera irányvektora által bezárt szöget. Egy maximumkereséssel eldöntjük melyik kamera látja legnagyobb szögből az adott pontot, majd a pixel színének mintavételezést az ehhez tartozó bejövő képből végezzük. Az algoritmus előnye a robusztusság, azaz minden esetben működik ha az adott pontot legalább egy kamera látja. Hátránya a meglehetősen zajos kép.

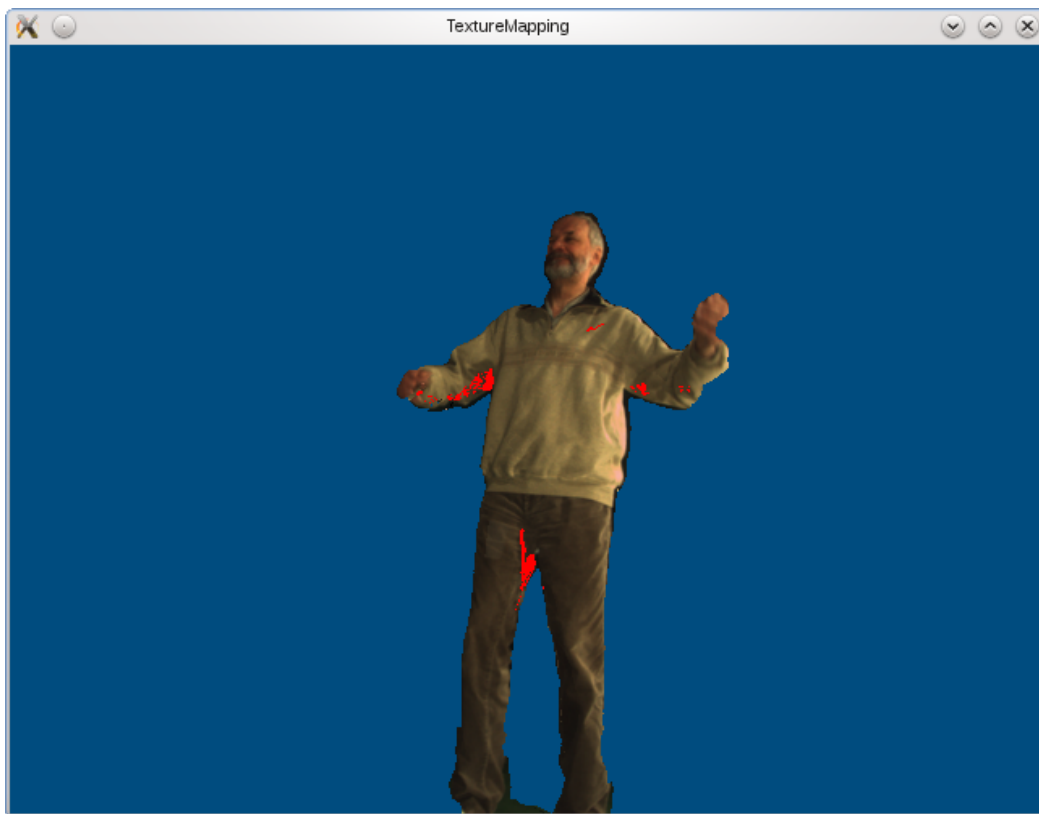
Ezt a kamerák közötti éles váltások okozzák (12. ábra).



12. ábra. Az első textúrázási módszer eredménye

Sokkal egyenletesebb eredményt kapunk, ha a textúrázást annak a kamerának a képe alapján végezzük, amelynek irányvektora aktuális nézőpontunkhoz képest a lehető legkisebb szöget zár be. Ezt használja ki a második módszer. Első lépésként, még a renderelés megkezdése előtt CPU oldalon kiszámoljuk, a saját és a kamerák nézeti irányvektora által bezárt szöget. A takarási probléma miatt, sajnos csak pixel szinten tudjuk meghatározni, hogy mely három kamera a legmegfelelőbb a mintavételezés szempontjából. A művelet gyorsítása érdekében sorba rendezzük a kamerákat a bezárt szög alapján. Ezt a sorrendet küldjük el a fragment shadernek. A shader a takarási információkat is figyelembe véve megpróbálja kiválasztani a három legmegfelelőbb kamerát. Ha sikerül, akkor a kamerák nézeti irányvektorában, mint bázisban felírjuk a mi nézeti irányvektorunkat. Ez a művelet egy mátrix szorzással elvégezhető. A kapott együtthatókat, mint súlyokat használva súlyozzuk az egyes kamerák képéből mintavételezett színinformációkat. Az eredmény lesz a pixel végleges színe (13. ábra. A piros színnel jelöltük az eljárás alkalmazásával nem feltextúrázható pixeleket)

Ha nem sikerülne három megfelelő kamerát találni, akkor visszaállunk az első módszerre. Ez csak nagyon kevés kamera alkalmazása esetén fordul elő, illetve ha a felület eleve nem látható (14. ábra).



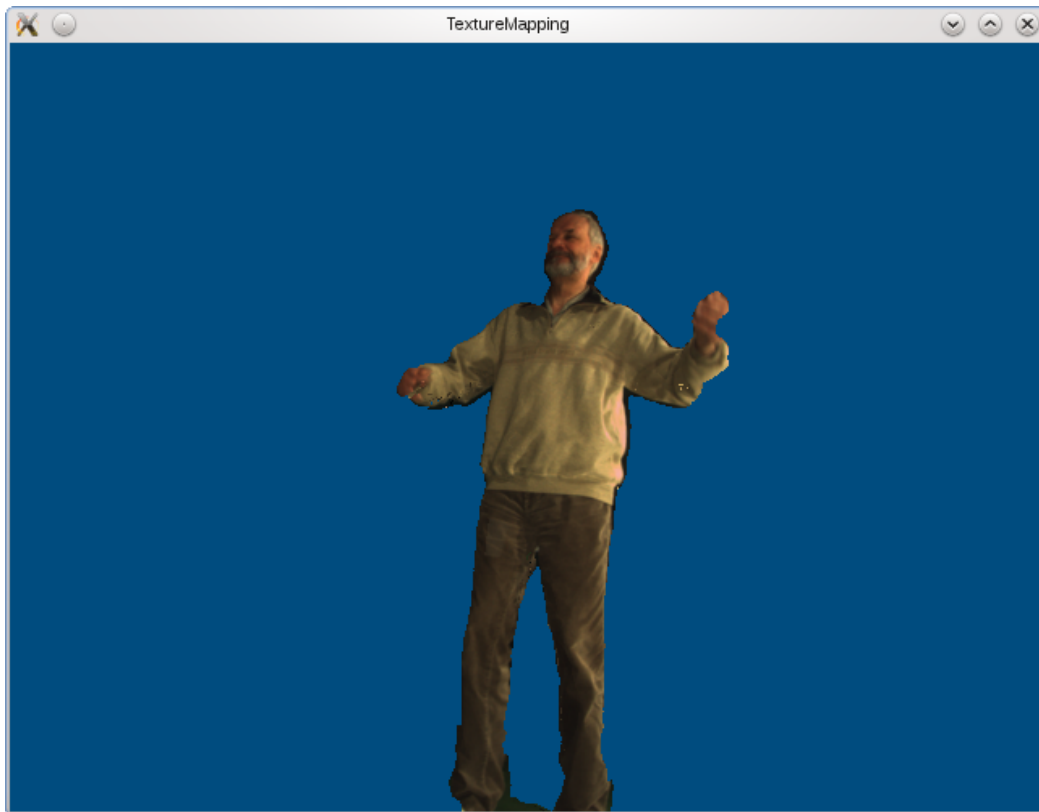
13. ábra. Az második textúrázási módszer eredménye

5.7. Árnyéktérképek

A shadowmapping (árnyéktérképek) algoritmus, egy gyakran használt gyors eljárás valószerű árnyékok létrehozására. Az eredeti algoritmus irányított, illetve fényszórószerű fényforrásokkal dolgozik, de többszöri alkalmazással pontszerű fényforrások esetén is alkalmazható. Az eljárás a mélységbuffer (z-buffer) segítségével határozza meg az árnyékban lévő részeket.

Az első fázisban a fényforrás szemszögéből lerendereli a jelenetet. Közben a mélységbuffert egy speciális textúrába, úgynevezett árnyéktérképbe (shadowmap) mentve. A textúra texelei fényforrás és a tér egyes pontjai közötti távolságot tartalmazzák. A második fázisban ezt a textúrát a fényforrásnak megfelelő vetítéssel alkalmazzuk a jelenetre. Minden fragmentre összevetjük a textúrából mintavételezett távolság értéket a vetítés által megadott távolság értékkel. Ha utóbbi érték nagyobb akkor az adott fragment árnyékban van, egyébként pedig nem.

Az implementáció során a teljesítmény növelése érdekében speciális shader programokat alkalmaztunk, amelyek az árnyéktérképek számítása során csak a mélységértékek meghatározásához szükséges műveleteket végzik el.

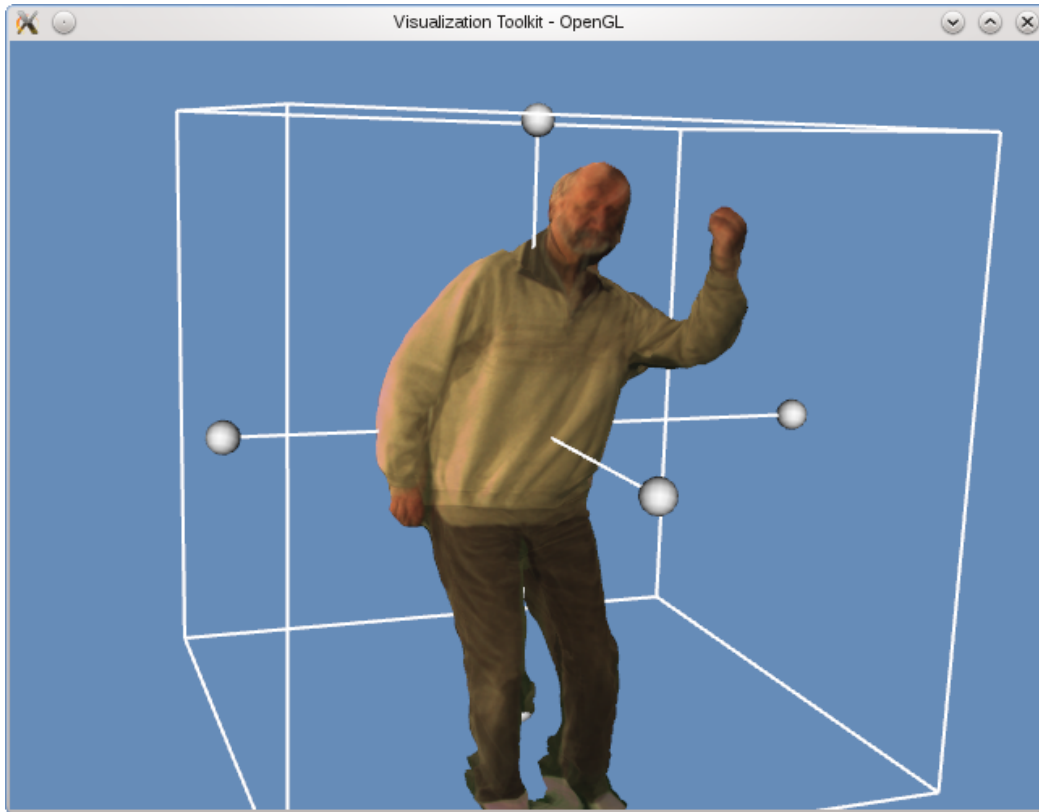


14. ábra. A textúrázási módszerek összevonása

6. Tesztelés

A rekonstrukciós futószalag moduláris felépítéséből köszönhetően, az egyes fázisok önállóan is futtathatóak, tesztelhetőek. Ezt elősegítette az offline verzió által lementett mérési adatok, mint például kameraképek, kalibrációs adatok, sziluett képek és térfogat modellek. Az egyes fázisok teszteléséhez önálló segédprogramok készültek. Mivel a teljes futószalag futtatása jelentős időt és teljesítményt emészt fel, ezért a hibakeresést elősegítendő az egyes fázisok feladatát szimuláló megvalósítások is készültek. Ezek az offline módszer által lementett adatokat töltötték be és szolgáltatták eredményként.

A teljes futószalag tesztelésének biztosítása érdekében egy a vezérlőszoftver által teremtetett feltételeket szimuláló tesztprogram készült. Megadva a háttérképek, felvételképek és a kamera kalibrációs adatok elérési útját az algoritmus betölti azokat a memóriába, majd a vezérlőszoftverhez hasonló módon használja a valósídejű rekonstrukciós komponens szolgáltatásait. Indításkor lehetőség nyílik a használni kívánt kamerák kiválasztására, a szimuláció során pedig küszöbértékek állítására.



15. ábra. A textúrázott eredmény VTK-ban

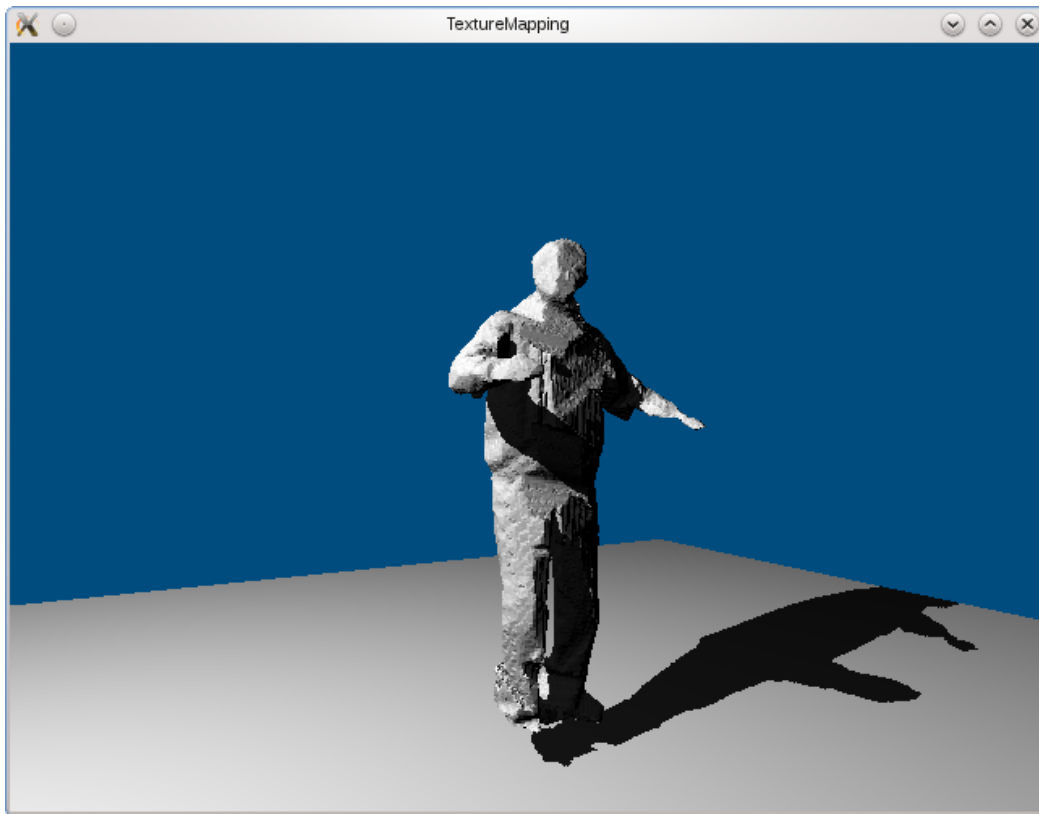
1. táblázat. Futási idők fázisonként

Felbontás	Seg.	Blur.	VH	MC	SM	Tex.	Összesen
320×244	5.1	5.7	1.5	1.6	1.5	1.5	16.9
480×364	11.9	12.8	1.6	1.5	1.5	1.5	30.7
640×484	20.5	22.5	1.6	1.5	1.5	1.5	49.0
1624×1236	129.1	147.8	1.6	1.5	1.5	1.5	282.9

7. Eredmények, további tervek

A munka eredményeképp sikerült egy a rekonstrukciót elvégző munkafolyamatot grafikus processzoron megvalósítani, ezzel lehetővé téve annak valósidejű végrehajtását. A pusztán geometria rekonstrukción túl, opcionális lehetőség annak eredményének textúrákkal való ellátása. Az offline működéshez képest a textúrázáshoz teljesen eltérő módszerek kerültek kidolgozásra a valósidejű végrehajtás által támasztott szűk időhatárok betartása végett.

A 1. táblázatban néhány mérési eredmény található a rekonstrukciós eljárás sebességéről (A táblázatban használt rövidítések: Seg. - szegmentálás, Blur. - simítás, VH - Vizuális Burok, MC - Marching Cubes, SM - árnyéktérképek számítása, Tex. - textúrázás). A mérések különböző felbontású bemeneti képekkel készültek, a rekonstruált térfogatmodell minden esetben $128 \times 128 \times 128$ -as volt. Jól látható, hogy a munkamenetben a legtöbb időt



16. ábra. Árnyékszámítás a generált modellből

a szegmentálás és a simítás igényli. Ezek a fázisok kevés számítási műveletet végeznek, de intenzíven használják a memóriát. Megjegyezzük, hogy a szegmentálás tartalmazza a képek videokártyába való feltöltését is. A feldolgozás szűk keresztmetszete tehát a nagy mennyiségű adat mozgatása a PCI-Express buszon keresztül és a memóriaműveletek.

A stúdióban alkalmazott kamerák másodpercenként 25 képkockát küldenek, tehát 40ms-enként egyet. A rekonstrukció idejét is ez alá az érték alá kellett szorítani a valós idejű megjelenítés érdekében. Ezt a célt a két kisebb felbontás alkalmazása esetén sikerült is elérni. Magasabb felbontású bemenő adatok alkalmazása esetén a minőség már számottevően nem javul, illetve a nagy mennyiségű adat hálózaton való átküldése is hardveres korlátokba ütközik.

Terveink szerint a közeljövőben a stúdió összeköttetésre kerül a VirCA ³ rendszerrel, hogy a valós idejű rekonstrukció eredményét, egy bejárható 3D-s környezetben megjeleníthetővé tegyünk. További lehetőségként fennáll a végrehajtási pipeline rugalmasságából adódóan az egyes feladatok más módszerekkel való megoldása a teljesítmény, vagy a minőség további javítása céljából. Nem került még kihasználásra a stúdió fejlett, programozható világítása sem. Megfelelő alkalmazásával növelhetjük a rekonstrukció minőségét, esetleg finomabb részletek előállítását is lehetségessé válhat.

³<http://www.virca.hu/>

8. Köszönetnyilvánítás

Először is szeretnék köszönetet mondani témavezetőmnek Csetverikov Dmitrij tanár úrnak, aki lehetőséget adott a projektben való részvételre, és munkám során felmerülő problémák, kérdések megoldásában mindig azonnal segített.

Továbbá köszönettel tartozom az MTA SZTAKI Geometriai Modellezés és Számítógépes Látás Kutatólabor tagjainak, akik szakmai tapasztalatukkal, tanácsaikkal hozzájárultak a dolgozat elkészültéhez.

Ábrák jegyzéke

1.	A stúdió	4
2.	A stúdió látványterve	5
3.	A stúdióban található egy kamera, és a hozzá tartozó fényforrás	5
4.	A GPU munkamenet (a szaggatott vonallal jelölt fázisok opcionálisak) . .	7
5.	A feldolgozásra beérkező képek	10
6.	A szegmentálás eredménye	11
7.	A Vizuális Burok algoritmus	12
8.	Az elmosott sziluettképek	13
9.	A Marching Cubes algoritmus különböző esetei	13
10.	A voxelek, és a csúcspontok indexei	14
11.	A generált háromszögháló VTK-ban	15
12.	Az első textúrázási módszer eredménye	16
13.	Az második textúrázási módszer eredménye	17
14.	A textúrázási módszerek összevonása	18
15.	A textúrázott eredmény VTK-ban	19
16.	Árnyékszámítás a generált modellből	20

9. Irodalomjegyzék

- [1] Bryce E. Bayer. Color imaging array. US Patent, 1976-07-20. Patent Number: US3971065.
- [2] FORTH ICS, 2010. From multiple views to textured 3D meshes: a GPU-powered approach.
www.ics.forth.gr/~argyros/research/gpu3Drec.htm.
- [3] JANKO, Z., AND PONS, J.-P. 2009. Spatio-temporal image-based texture atlases for dynamic 3-D models. In Proc. ICCV Workshop 3DIM'09, 1646-1653.
- [4] KUTULAKOS, K., AND SEITZ, S. 1999. A theory of shape by space carving. In Proc. International Conference on Computer Vision, vol. 1, 307-314.
- [5] LAURENTINI, A. 1994. The Visual Hull concept for silhouettebased image understanding. IEEE Trans. Pattern Analysis and Machine Intelligence 16, 150-162.
- [6] LORENSEN, W., AND CLINE, H. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In Proc. ACM SIGGRAPH, vol. 21, 163-169.
- [7] SZTAKI GMCV, 2011. The 4D Reconstruction Studio.
<http://vision.sztaki.hu/4Dstudio/index.php>.
- [8] LANCE WILLIAMS Casting curved shadows on curved surfaces Computer Graphics Lab New York Institute of Technology Old Westbury, New York 11568
- [9] nVidia CUDA SDK Marching Cubes Isosurfaces
<http://developer.nvidia.com/cuda-cc-sdk-code-samples#marchingCubes>