Multidiszciplináris tudományok, 9. kötet. (2019) 4 sz. pp. 444-454 **https://doi.org/10.35925/j.multi.2019.4.46**

A SZÁMÍTÓGÉPES VIZUALIZÁCIÓ MÚLTJA ÉS JÖVŐJE

Mileff Péter
egyetemi docens, Miskolci Egyetem, Informatikai Intézet, Általános Informatikai Tanszék
3515 Miskolc, Miskolc-Egyetemváros, e-mail: mileff@iit.uni-miskolc.hu

Dudra Judit
tudományos munkatárs, Bay Zoltán Alkalmazott Kutatási Közhasznú Nonprofit Kft. Szerkezetintegritás Osztály,
3519 Miskolc, Iglói út 2., e-mail: judit.dudra@bayzoltan.hu

Absztrakt

A számítógépes vizualizáció hosszú, sok éves múltra tekint vissza. A tématerület folyamatosan követte a rendelkezésre álló hardverek fejlődését. Ahogy újabb és nagyobb számítási kapacitású számítógépek váltak elérhetővé, úgy bontakozott ki a számítógépes vizualizáció is. Újabb algoritmusok, technológiák jelentek meg. Ma már nincs olyan terület a mindennapi életünkben, ahol ne jelenne meg a megjelenítés igénye valamilyen formában legyen az egy autó, egy számítógépes játék, vagy akár egy IOT eszköz. Jelen cikkben megvizsgáljuk és röviden áttekintjük azt a meghatározó irányvonalat, amit a modern valós idejű vizualizáció jelenleg követ. Kitérünk azokra a fontosabb technológiákra, amelyek szerves részét képezik a mai megjelenítésnek, és azokra a lehetőségekre, amelyek meghatározók lehetnek a jövőbeli valós idejű megjelenítésben. A megjelenítés jövőjét, az elfogadott technológiát nehéz megjósolni, hiszen erősen függ attól, hogy milyen hardvergyártó által támogatott technológia válik elfogadottá és dominánssá.

Kulcsszavak: *számítógépes vizualizáció, voxel, szoftveres megjelenítés, sugárkövetés*

Abstract

Computer visualization has a long history with many great strides. However, the 21th century has seen the greatest progress in the development. The area continuously followed the evolution of the available hardwares. As more and more computing capacity based computers became available, computer visualization had been unfolded. New algorithms and technologies have appeared. Nowadays, there is no area in our daily lives where the need for visualization, whether it be a car, a computer game, or even an IoT device, does not appear. In this article, we examine and briefly review the dominant trend that modern real-time visualization currently follows. We cover the key technologies that are an integral part of today's rendering and the opportunities that could be decisive for future real-time rendering. It is difficult to predict the future of visualization, the technology adopted, as it depends heavily on what hardware vendor-supported technology becomes accepted and dominant.

Keywords: computer visualization, *voxel based rasterisation, software rendering, ray-tracing*

1. Bevezetés

A számítógépes vizualizáció kezdete az első számítógépek megjelenésének idejére tehető. A hardverek rendelkezésre állása természetes igényként vonta maga után a terület fejlődését. A

vizualizáció mára már életünk szerves részévé vált. Szinte nincs olyan terület, ahol ne jelent volna meg valamilyen formában. Bármely olyan alkalmazás, amely adatokat gyűjt, ott szükség lehet vizualizációra. Gondolhatunk itt akár a modern autók műszerfali illetve a kiegészítő kijelzőire. Vannak olyan alkalmazási területek, ahol a fejlesztés kizárólag vizuális úton történhet. Ilyen például számítógépes játék, egy animációs film, vagy akár egy alkatrész CAD modellje. Nincs olyan hírdetés vagy TV reklám, amiben ne szerepelne valamilyen vizualizáció. Az ember vizuális típus. Bármit is kelljen demonstrálni, megérteni vagy megértetni, előszeretettel alkalmazunk ábrákat, animációkat és egyéb grafikai azért, hogy növeljük az érthetőséget.

A számítógépes vizualizáció legfontosabb célja a valóság minél pontosabb modellezése a rendelkezésre álló hardverek mellett, amely egy nagyon komplex, sokténye zős feladat. Ezek alapján a megjelenítésnek két fontos típusát lehet megkülönböztetni: a *valós idejű renderelés*t, illetve az úgynevezett *offline renderelés*t. A két csoport már a kezdetektől fogva szétválik. Mivel igényeik eltérőek, így fejlődésük is különböző formában haladt előre.

Az offline renderelés célja a magas képi minőség elérése. Tipikus alkalmazási területei az (animációs) filmek, háromdimenziós modellek (pl autó, épület, stb) egy adott nézetből vett képének renderelése. Mivel a legmagasabb magas képi minőség a legfőbb cél, a valóság minél jobb közelítése, így a képpontok számítása nem valós időben történik. A képkockát szigorú időkorlát nélküli kiszámítása lehetőséget ad a valósághű ábrázolást meghatározó termérdek tényező pontos, fizikailag helyes figyelembevételére. Tipikusan ilyen tényező a fény, a megvilágítási modell, a tükröződés, felületi anyagtulajdonságok és az árnyékok hatékony kezelése. Ezeknél a megjelenítési formáknál az igazán jó eredményt valójában az úgynevezett globális illumináció alkalmazása adja, amely során az indirekt fények is modellezésre kerülnek. Az offline megjelenítés területe sokat fejlődött az évek során, de nem annyit, mint a valós idejű megjelenítés. A valóságot helyesen közelíteni próbáló algoritmusok nagy része már a nagyon korai időkben is rendelkezésre állt, határt minden esetben a rendelkezésre álló számítási kapacitást szabott. Ennél a megjelenítési formánál már a kezdetekben felismerték, hogy a képkockák számítását nem lehet csak egyetlen számítógépre bízni. Az idők során megjelentek a lokális, ma pedig már a globális úgynevezett render farmok. A tervező szoftverekben van lehetőség a számítát különböző kijelölt gépekre elosztani. Ezt nevezhetjük lokális render farmnak. Valamint a szélessávú internet terjedésével és a webes technológiák, felhők rohamos fejlődésével ma már bérelhető számítási kapacitások állnak rendelkezésre egy-egy képkocka vagy akár egy teljes film képi elemeinek kiszámítására. Fontos kiemelni, hogy a területen elég sokáig kizárólag CPU-n alapú számítási lehetőségek álltak rendelkezésre, ma viszont már elérhetők GPU alapú farmok is. A viszonylagos késői GPU megoldások elterjedésének oka abban keresendő, hogy a magas képminőséghez sokszor ragaszkodnak a 64 bites számításokhoz. A GPU-k viszont csak az utóbbi években nyújtanak lehetőséget erre. A másik ok pedig, aminek a késői támogatást köszönhetjük, hogy egy-egy képkocka számítását is akár több gépre feloszthatjuk. GPU-k esetén azonban ez kezdetben szintén nehézkes volt, a driver és API támogatásnak meg kellett érnie ehhez.

A cikk a továbbiakban a valós idejű megjelenítésre fókuszál, amely minden kétséget kizáróan nagy utat tudhat maga mögött, rendkívüli fejlődésen ment keresztül az utóbbi években. A valós idejű renderelés szigorú feltétele (általában) az, hogy legalább annyi képkocka kerüljön kiszámításra másodpercenként, amennyi az emberi szem számára már folyamatosnak látszik, valamint amely szükséges az adott vizualizációhoz, esetleg interakcióhoz. Számítógépes játékok esetében ez tipikusan 50-60 képet jelent másodpercenként. Ez alatt bizonyos típusú játékok esetében - ahol gyorsabban változnak az események - a játékélmény nem kielégítő. Itt lehetne talán kiegészíteni a számítógépes vizualizáció fent említett célját, miszerint a valóság minél pontosabb modellezése a legfontosabb

irányelv. Valójában mindenzt a valós idejű megjelenítésben kelle elérni, odáig fejlődni, hogy a most offline úton előállított képek valós időben álljanak elő. A hardverek fejlesztése és minden törekvés legfőképpen ebbe az irányba mutat ezért is mondhatjuk a legtöbbet fejlődő területnek.

A továbbiakban a vizualizáció legfontosabb lépcsői és technikái kerülnek áttekintésre.

2. Hardverek fejlődése

Nyugodtan kijelenthető, hogy a számítógépes vizualizáció legfőbb hajtómotorja a játékipar. Gyakorlatilag az első számítógépek megjelenése óta egyre növekvő igény van arra, hogy az ember valamilyen formában játszhasson a géppel. És természetesen mivel a megjelenítés kulcsszereplő a végeredményben, így egy nagyon erős fejlődésen ment keresztül. Kezdetben csak játéktermi automatákon volt lehetőség minőségi játékokkal játszani. Az igazi robbanást az otthoni számítógépek megfelelő áron való megjelenése jelentette, amikor már szinte bárki számára elérhetővé vált az, hogy legalább egy asztali számítógépet vásárolhasson. Fontos szereplői voltak ennek a kornak a 386, 486, 586, 686 valamint a Pentium I. típusú gépek. Ezek a hardverek nem voltak erősen a maiakhoz viszonyítva. Nem túl erős CPU, kevés és lassú memória, korlátozott BUS sebesség (ISA, PCI) és természetesen a hardveresen gyorsított megjelenítés teljes hiánya. Összehasonlításképpen egy akkori Intel 486DX-33 (1990) 0.03 GFLOPS teljesítménnyel bírt, míg egy Intel Core i7-3770 (Ivy Bridge) (April 2012) teljesítménye 108.8 GFLOPS volt. Ennek (és az internet hiányának) ellenére a játékipar virágozni kezdett, ezt az időszakot DOS korszaknak nevezzük. Az alacsony hardver teljesítmény ellenére nagyszerű grafikai alkalmazások/játékok születtek. A szoftverek készítésének tipikus nyelve a C, PASCAL volt, valamint a teljesítmény kritikus részek ekkor assembly blokként beágyazva készültek el. A programoknak (főként játékok esetében) jól optimalizáltnak kellett lennie, hiszen se dedikált GPU alapú grafikus kártya, de komoly teljesítmény nem állt rendelkezésre, mindent a CPU számított. Ezt nehezítette továbbá, hogy az akkori rendelkezésre álló programozási eszköztárak, környezetek nem voltak olyan rugalmasak és kényelmesek, mint a maiak.

Talán egy külön korszakként említhető a Pentium I., II, III és IV család. Nagyon fontos állomás volt az, hogy az első generációs Pentium-ok központi egységében már rendelkezésre állt az úgynevezett MMX bővített utasításkészlet, a későbbiekben pedig ennek fejlettebb változatai az úgynevezett SSE család. Hardveresen gyorsított GPU ekkor még mindig nem készült, a grafikát továbbra is a CPU számította. Az MMX és SSE egy SIMD (single instruction, multiple data) utasításkészlet. Segítségükkel ugyanazt az aritmetikai műveletet lehet elvégezni egy időben nagy mennyiségű adaton. A SIMD nagy előrelépést jelentett a számítások gyorsításában, segítségével nagy vektorok vagy mátrixok manipulációja rövidebb idő alatt volt elvégezhető. A SIMD utasítások a hang-, kép- és videofeldolgozásban alkalmazott algoritmusok könnyű párhuzamosítását teszik lehetővé. A gyakorlatban, a számítógépes megjelenítésben ez azt jelentette, hogy a processzor egyszerre több pixelen volt képes valamilyen transzformációt végezni. A megjelenést követően a szoftverek hamar támogatni kezdték az új utasítás családot, annak ellenére, hogy egy szoftver sebességkritikus részeinek elkészítése MMX/SSE alapokon nem volt triviális feladat. Az MMX vonal megszületése után kevéssel az AMD cég is előállt egy bővített utasításkészlettel, a 3DNOW!-al, amely egy hasonló, főleg a háromdimenziós megjelenítést segítő kiterjesztés volt.

A hardvergyártók hamar felismerték, hogy a megjelenítés folyamata jól szeparálható egy szoftver többi részétől valamint jól párhuzamosítható. Erre az igényre reagálva sok fejlesztés után születtek meg az első igazi dedikált GPU-k (3DFX Voodoo I (1996), Voodoo II (1998), GeForce 256 (1999)), amelyek nagy előnye az volt, hogy már az első kártyák jelentős javulást eredményeztek a

képminőségben és sebességben. Bár jelen cikk főként a PC vonalra fókuszál, ahhoz hogy ezek a GPU-k elkészülhessenek sok technológiai lépcsőfokot kellett bejárni. Számos kisebb- nagyobb sikeres chip, kártya született. Ilyen volt például az Amiga 4000 számítógépekbe megjelent AGA (Advanced Graphics Architecture - 1992) kiegészítő hardverelem is.

A videokártyák robbanásszerűen terjedni kezdtek, bármely boltban elérhető volt viszonylag kedvező áron. A szoftverek pedig rögtön támogatni kezdték az ekkor már megszületett OpenGL és DirectX API-k révén. A GPU-k piaca ettől a ponttól fogva folyamatosan nőtt annak ellenére, hogy nagy technológiai innováció azóta sem történt. Kezdetben a videokártyák fix funkciós csővezetékkel rendelkeztek, a programozó nem, vagy csak nagyon korlátozott módon szólhatott bele a renderelés folyamatába. Fontos állomás volt a fejlődésben a programozható csővezeték, az árnyalók megjelenése. A fejlesztés hosszú utat járt be, közben megjelentek a hordozható eszközökbe szánt GPU-k is. Nyugodtan kijelenthető, hogy a mai GPU-k már nagyon nagy teljesítménnyel rendelkeznek. Megjelenítésen felül már általános számítási feladatokra is alkalmazhatók. Ha megvizsgáljuk a TOP500 által nyilvántartott leggyorsabb számítógépek listáját, az élmezőnyben számos GPU alapú számítógép szerelepel.

Nem szabad elfelejteni, hogy a GPU nem egy csodaeszköz. Bár gyorsak, de áruk többszörösére emelkedett sz éek során, emellett pedig áramfelhasználásuk is jelentősen megnőtt. Megfigyelve a piacot, egy nagyon erős marketing/média támogatás érezhető. Ma már szinte minden eszköz kap egy GPU-t, magától érthetődik így az, hogy a vizualizációra azt használjuk. Ez nem feltétlenül helyes minden esetben, hiszen a használt hardver általában megköti az alkalmazható renderelési technikát. Mindamellett pedig az ipar elfelejtette, hogy van már egy jó hardver a számítógépben, amit megjelenítésre vagy annak kiegészítésére lehet használni: a CPU.

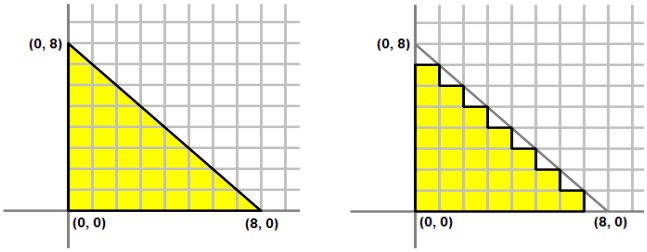
3. Poligon alapú geometria

Az alakzatok memóriában való reprezentációjára számos megközelítés alakult ki, de napjainkban leginkább elterjedt és domináns objektum reprezentáció a poligon alapú megközelítés. Az objektumot, modellt ilyenkor általában a legegyszerűbb konvex poligonra, háromszögekre bontjuk fel a modellezés során, a kirajzolás folyamatában pedig ezek az elemek kerülnek raszterizálásra valamilyen algoritmus alapján. A megjelenítés sebessége és jellege mindig nagyban függ a raszterizáló algoritmustól. Bár számos különböző megoldás kialakult az évek során (pl. raytracing, volume rendering, stb), jelenleg uralkodó eljárásként a háromszög kifestés alapú megközelítést alkalmaznak a GPU gyártók a valós idejű megjelenítésben. A választás oka a teljesítményben keresendő, mert a megvalósítás lényegesen gyorsabb vizualizációt tesz lehetővé, mint például a sugár alapú algoritmusok. A háromszög, tehát mint önálló egység, különös fontossággal bír, gyakorlatilag a valós idejű megjelenítés atomi egységének tekinthető.

3.1 Kifestő algoritmusok

A háromszög kifestés alapú megközelítés már a korai számítógépes vizualizáció alapját képviselte és mai is a domináns megjelenítő megoldás. Nagyon leegyszerűsítve a folyamat lényege a következő.
A képernyő tartományába leképzett háromszögek, mint minden önmagát nem metsző síkbeli poligon a síkot két részre (two regions) bontja: a belső terület, amely véges, és a külső, ami nem. A kettő határát a háromszög határvonalai, azaz az élei alkotják. Ahhoz hogy raszterizáljuk a háromszöget, lényegében a pontok egy olyan halmazát kell bejárni valamilyen algoritmussal, amely valamilyen értelemben része a háromszög leképzésnek, vagy valamilyen mintát követnek, és kiszámolni, hogy mely pontok

vannak a háromszögön belül. A belső pontok színe ezután határozható meg. Az eljárás lényegében egy diszkretizálási folyamat:



1. ábra. *Pixelleképzés mint diszkretizálási folymat*

Klasszikus értelemben a kifestés folyamata pixelenként hajtódik végre így a belső iteráció és a különböző számítások meglehetősen sokszor kell végrehajtódjanak. A folyamat bár egyszerűnek tűnik, a kifestés sebessége, így pedig az alkalmazás sebessége nagyban függ az implementált algoritmustól, annak optimalizálási szintjétől továbbá attól, hogy a valóságot milyen szinten modellezzük (fények, tükröződések, árnyák, stb). Napjainkban két olyan kifestési megközelítés ismert, amely széles körben elterjedt: a scanline [1] és a féltér alapú algoritmusok [1]. Az időben korábban kialakult scanline megközelítés alapgondolata az, hogy a raszterizáció során a háromszögeken fentről lefelé haladunk végig soronként (scanline). Minden sor egy olyan vonalat jelent, amely kezdete és vége a háromszög oldalai és a scanline metszéspontja az x tengely mentén. A végpontok az élek slope értékének kiszámításával inkrementálisan meghatározhatók. A kifestés folyamata lényegében ezen sor pixelszínértékek meghatározása. A féltér alapú megközelítés a poligonok konvexitásából indul ki: egy n élből álló konvex poligon belső területe minden esetben leírható az n darab féltér metszésével. A háromszögek esetében tehát a három féltér egyértelműen definiálja a befestendő tartományt. A háromszög befoglaló dobozát kiszámítva, bejárva és a három sík egyenletét felhasználva a háromszög kifesthető.

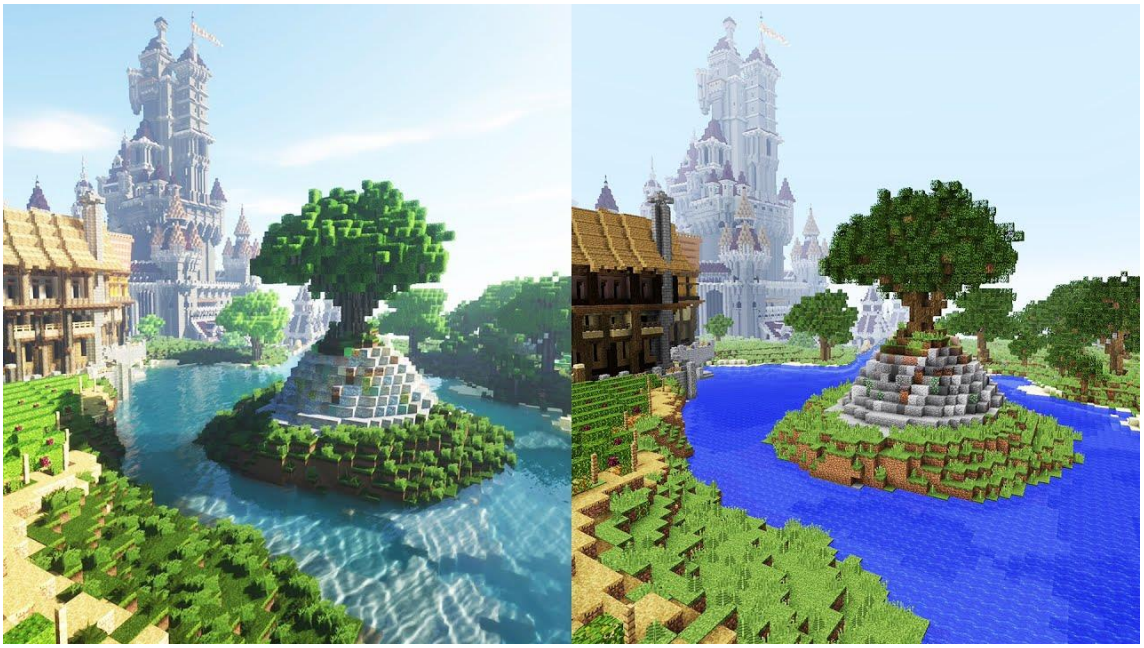
Mindkét algoritmusnak az irodalomban megtalálhatók különböző módosított változatai [10]. A GPU-k a második megközelítést alkalmazzák a könnyebb párhuzamosítás [10] miatt.

3.2 Sugár alapú megoldások

Egy további domináns irányvonal a raszterizációban a sugár alapú megoldások. Néhány fontosabbat említve: Ray-tracing, Ray-marching, Cone tracing, Beam tracing, stb. Ezen megoldások alapjaiban eltérnek a kifestésen alapuló algoritmusoktól. Ezen megközelítések a virtuális világban elhelyezett fényforrásokból érkező sugarakkal dolgoznak. A valóságot próbálják meg modellezni azzal, hogy a térbeli objektumokat a fényforrásokból származó fotonok/fénysugarak világítják meg úgy, hogy közben a sugarak pattognak objektumokról objektumra. Ezzel a megoldással globális illumináció állítható elő amennyiben kellő számú sugár mozgását követjük le. A gyakorlatban egy több millió sugarat jelent. Éppen ezért ezek a módszerek nagy számításigénnyel rendelkeznek, mindezidáig főként offline renderelésnél alkalmazták őket filmek és egyéb magas szintű grafikai modellezésre [4].

A mai modern hardverek azonban már erősek, egyre több törekvés születik arra, hogy valamilyen formában, akár korlátozott minőségben is, de használhatók legyenek ezek az algoritmusok valós

időben is. Az animációs filmek készítésekor a leggyakoribb probléma az, hogy a tervezők szeretnék az adott jelenetet több szemszögből, több fényforrással, azok paraméterinek módosításával megnézni. Mindezt pedig úgy, hogy nem kelljen egy ilyen jelenet renderelésére perceket várni. A törekvések jól láthatók a videokártyagyártók szemszögéből is [9]. A sugárkövetés sosem volt GPU támogatott, csupán az utóbbi években kezdtek megjelenni bizonyos eszközök (pl. NVIDIA OptiX™ API, NVidia RTX alapú kártyák). 2019-ben pedig megjelent egy Crytek által készített valós idejű sugárkövetésen alapuló, az első megfelelő minőséget mutató valódi futtatható demó alkalmazás.



2. ábra. *A híres Minecraft készülőben lévő, sugárkövetést használó változata. Jól látszik a minőségbeli küldönbség.*

A sugár alapú megközelítés egy jó jövőbeli irány. Mint a képen (2.ábra) példaként látható, egyre több izgalmas sugáralapú átirat van készülőben. A jelenlegi hardverekkel már-már a valós idejű futtatás határán vagyunk.

3.3 Poligon alapú megjelenítés jövője

A sugárkövetés területén megjelenő újabb és újabb hírek alátámasztják azt a tényt, hogy a megjelenítés (közeli) jövője valamilyen sugár alapú globális illuminációs technikán fog alapulni. Feltehető a kérdés, hogy mi a probléma a kifestéssel, ha olyan sok évig ezt a megközelítést támogatta minden. Ha megnézzük a mai AAA számítógépes játékokat, belátható, hogy pazar látványvilággal rendelkeznek. Azonban ennek a minősége nem fejleszthető a végletekig. Ahhoz, hogy egy ilyen minőségű eredményt lássunk a képernyőn rengeteg kis trükköt, algoritmust és egyéb kiegészítő a valóságot hamisító megoldást kell alkalmazni. Egy ilyen játékmotor nagyon komplex. Már egy egyszerű(nek tűnő) pontszerű fényforrás esetében négy árnyéktérképet (shadowmap) kell létrehozni és azok minőségéről még nem is beszéltünk. A tükröződések, a csillanások, a felületi érdességek mind-

mind egy komplex gépezet részei, aminek megvannak a határai. A globális illumináció csupán a kifestésen alapuló megoldásokkal nem érhető el. A hardverek teljesítményei elértek egy olyan pontot, amely nagy valószínűséggel a közeljövőben technológiai váltást fog eredményezni a piacon. A sugár alapú megoldásokkal az említett problémák nagy része azonnal megszűnik, hiszen ott az eltérő képszintézis már alapjában véve reális képet nyújt, nincs szükség egyéb kiegészítő „trükkökre".

A másik súly, amit a számítógépes grafika magával cipel, a háromszög reprezentáció. Egyszerűbb alkalmazások esetén a poligon reprezentáció megfelelő, azonban egyben nagy megkötés is. A poligon alapú reprezentáció és a GPU annak az oka, hogy a mai játékok „statikusak". A környezet szerkezete nem változtatható meg, nem rombolható le egy fal, stb. Bizonyos AAA játékokban találkozhatunk ehhez hasonló megoldásokkal, de minden esetben egy előre „bedrótozott" logika alapján törik szét az objektum, omlik le a fal. A dinamizmus hiányának oka a háromszög reprezentáció. Egy hosszú fal két háromszögből és némi effektből való felépítése megfelelő raszterizációs sebességet és minőséget jelent, a fal struktúrájának módosításához, kilyukasztásához viszont nincs elég információ. Ahhoz, hogy egy lyuk keletkezzen a falban valós időben kell módosítani a poligonhálót, amin általában több textúrát is alkalmaznak. Tehát a törés helyének és logikájának alapján új vertexeket és háromszögeket kell alkotni, mindezekhez pedig valamilyen textúra koordinátát társítani. Ez már önmagában sem egyszerű feladat, de mivel az érintett geometriai adatok a GPU-ban tárolódnak a gyors megjelenítés miatt, mindezt a CPU oldalra kellene visszahozni, elvégezni a szükséges módosításokat, majd újra feltölteni a GPU memóriába. Belátható, hogy ennek megvalósítása rendkívüli komplexitást igényelne és nem kizárt, hogy az ilyen jellegű feladatokat nem is lehet ezzel a technikával maradéktalanul megoldani.

Az utolsó felmerülő probléma a képi minőség tovább javítása. A videokártyák „nem szeretik" a sok vertex-et, ezért a játékokban úgy alakítják ki világ modelljét, hogy ahol lehet, spórolnak a poligonokkal. Mivel a görbületek magasabb felbontású poligonhálót igényelnek, így ezeken a pontokon fedezhetők fel a hiányosságok leginkább. Bár a GPU-k sokat fejlődtek az évek során, egy grafikai színtér részletgazdaságát továbbra sem főként vertex szám növeléssel, hanem főként felületi "effektekkel" próbálják meg növelni. Ilyen megoldás egy téglafal esetében például a Parallax Mapping, Relief Mapping, stb. E téren a GPU-k sokat fejlődtek, számos textúrát képesek kezelni egyszerre nagy teljesítménnyel. Textúra effektekkel azonban nem lehet hosszú távon a képi minőséget a valósághoz még inkább közelíteni. A minőség javításának egyik kulcstényezője a geometriai háló felbontásának növelése, ha a téglafal egy részletgazdag poligonhálóként épülne fel, sok háromszögből és vertexből. Ideális esetben, tehát ahogy nő a GPU-k teljesítménye, egyre komplexebb világot képesek modellezni, amelyben egyre több és egyre kisebb háromszögekből épülnek fel az objektumok. Gondoljunk csak egy erdőre a temérdek részlettel. Amennyiben a gyártók ezt az irányvonalat folytatják, egyre kisebb háromszögekkel kell a GPU-nak dolgoznia. Az apró részletek megjelenítése nagyon apró poligonokat eredményez, amely jelenleg sem, és a távoli jövőben sem megfelelő alternatíva a GPU (és a CPU) számára. Ennek oka, hogy minden háromszög kifestése egy úgynevezett előkészítési (setup) költséget von maga után [10]. A túl sok apró háromszögből felépülő világ esetében a teljesítményt várhatóan így felemészti a sok setup költség.

A GPU-k az OpenGL 4.0 és DirectX 11-től lehetőséget nyújtanak az úgynevezett hardveresen gyorsított GPU alapú tesszellációra [1]. A gyártók is felismerték azt a természetes igényt, amely akkor jelentkezik, amikor a kamera egy geometriai objektumhoz közelebb kerül. Amíg az objektum távol vagy nincs szükség nagyon részletese modellre, hiszen az apró részletek nem látszanak. Az objektumhoz érve, azonban már minden (akár továbbiakra is) háromszögre szükség van. A korábbi évek bevált megoldása, hogy ugyanazon objektumból több részletezettségi változatot készítettek, majd

a kamera távolság függvényében azt a változatot mutatták, amire szükség volt (LOD - Level of Detail). A tesszelláció képes egy adott felületet további háromszögekre bontani a kamera távolságának függvényében. Ez a technika nagyon hasznos, látszólag orvosolhatná a korábban említett problémákat, azonban itt egy a programozó által befolyásolható, de automatizált, valós időben történő felbontásról van szó. Belátható, hogy a teljes virtuális világ részleteit nem bízhatjuk egy ilyen megoldásra. Bár általában javítja a végeredményt, de sok esetben nem különböztethető meg egy tesszellált objektum egy helyesen alkalmazott Displacement Mapping-tól.

A (távoli) jövőben vélhetően háromszög alapú kifestés és maga a poligon reprezentációnak is eltűnik majd a számítógépes vizualizációból. Amikor már egy objektum minden apró részletére szükség lesz, úgy olyan apró háromszögekre lenne szükség, amellyel már elérnék a voxelek szintjére.

4. Voxel alapú megoldások

A voxel alapú megjelenítés nem új keletű a számítógépes vizualizációban, már a kezdetektől rendelkezésre állt, de a korai lassú hardverek teljesítményben nem álltak még készen az atomi felépítésen alapuló megközelítésre. Memóriában és háttértárban is korlátozott lehetőségeik voltak, így nem csoda, hogy a poligon kifestés alapú képszintézis vált egyeduralkodóvá. Eleinte a raszterizáció folyamatát kizárólag egy központi egység végezte el, csak később jelentek meg a grafikus gyorsító hardverek. Ennek ellenére a technika folyamatosan fejlődött az évek során főként a számítógépes játékoknak köszönhetően, azonban napjainkban az egyik legfontosabb tendenciájának látszik a fotorealisztikus megjelenítés területén [2].

Az elnevezés és eljárás lényege, hogy a napjainkban elterjedt poligon alapú megjelenítéssel ellentétben úgynevezett voxelekből (volumetric pixel) építi fel modellt, amelyet egy voxel halmaznak is nevezhetünk. Az, hogy mit jelent egy voxel, nehéz definiálni, a szakirodalomban sokszor háromdimenziós pixelnek is nevezik, de nevezhetjük akár atomnak is. A tipikus reprezentációja általában a modellbeli pozícióját, kiterjedését és a szín információt tartalmazza [5]. Ezek mellett a különböző technológiákat alkalmazó megjelenítők tárolhatnak egyéb információt (pl. normálvektor) is, melyeket a realisztikusabb megjelenítéshez használnak fel (pl. Ambient Occlusion). Alkalmazzák az orvosi képfeldolgozásban, geológiai adatok megjelenítéséhez, és természetesen számítógépes játékokban is (pl.: Blade Runner, Delta Force, Crysis, Voxatron, Minecraft, Motocross Stunt Racer, Red Alert, stb) a terep és különböző kisebb-nagyobb objektumok modellezésére.

A voxel halmazok magukban rejtik azt az előnyt, hogy igény esetén tetszőlegesen alakíthatóvá válik a modell, rombolható környezetet kapunk. Mindezeknek azonban nagy ára van, a nagy adathalmaz. Amikor a térben változik az objektumok pozíciója, orientációja, a voxeleket egyesével transzformálni kell. Ez önmagában is rengeteg adatot jelent, amelyhez sok memória és CPU/GPU számítási kapacitás szükséges. Bár az irodalomban számos számítógépes játék használt voxel technológiát, olyan, amely teljes mértékben kizárólag voxelekre építette volna a játéktér minden objektumát, csak nagyon kevés volt (pl. Hexplore, Voxelstein, Voxatron, stb). Ezeknél gyakran valamilyen limitációval rendelkeztek (pl. szabadsági fokok száma). Amennyiben voxelekkel szeretnénk dolgozni, több akadályba is ütközünk. Nincs megfelelő eszközrendszer jelenleg, a piacon elérhető modellező szoftverek többsége kizárólag poligonhálót tud kezelni. A statikus objektumok mellett pedig szükség van animációra is. Csontváz alapú voxel animációt megvalósító szoftver jelenleg nem elérhető a piacon. Ennek ellenére a Voxelstein 3D játékban már találkozhattunk ezzel a megoldással.

A voxel alapú technológiák terjedésének egyik gátlója, hogy a mai grafikus hardverek közvetlenül nem támogatják a voxel halmazok megjelenítését. Vannak kialakult irányok, trükkök főként a sugár alapú megjelenítésre alapozva, de nincs olyan megfelelő egységes támogatási irány a GPU gyártók részéről a hatékony megjelenítésre, mint a poligon alapú megoldásoknál. Míg poligon alapokon elég megadnunk a vertexek és textúrák halmazát, a GPU közvetlenül képes a modell megjelenítésére, addig a voxel alapú modellek esetében a programozónak saját árnyalót kell készítenie ennek megvalósítására (sugár alapú megoldások) [3].

A jövő megjelenítő motorjai vélhetően voxel technológián fognak alapulni, amikor rendelkezésre fog állni a szükséges számítási kapacitás. Napjainkban már megfigyelhetők kiegészítő technikaként például a globális illuminációs fények számításának támogatására [7[8]]. De vannak törekvések a teljes virtuális világ valós idejű modellezésére is [6] (pl. Atomontage Engine [11], Unlimited Details [12]). Itt azonban már jelentősen érezhető a GPU-k limitációi. Bár gyorsak, de memóriájuk limitált. Bár a gyártók folyamatosan bővítik, ma már akár 6 GB-os kártyák is elérhetők, nem tudunk akármekkora világot feltölteni a kártyára. Ráadásul, ha módosítani kell a világ részleteit, akkor szintén a CPU oldalra kellene mozgatni a szükséges adatokat. Ennek hiányában a játékok továbbra is statikusak maradnak.

3. ábra. *Romos voxel épület, bontható falakkal. Részlet az Atomontage grafikus motor bemutatójából*

Programozói szempontból a távoli jövőben a CPU lenne a megfelelő eszköz a voxel halmazok megjelenítésére. Ennek oka, hogy a grafika programozási modellje és nyelve nem térne el a megszokott környezettől, nagyobb memória állna rendelkezésre, az adatok cserélése könnyebben, rugalmasabban megoldható lenne. Ezt a törekvést próbálta meg megvalósítani az Intel Larrabee néven

futó kísérleti videokártyája [13]. Az architektúra egy hibrid megközelítés volt a GPU és CPU között, ahol a programozhatóság megtartotta az X86 alapokat.

5. Összefoglalás

A számítógépes vizualizáció területét ma a poligon alapú modell reprezentáció uralja, a grafikus hardver gyártók erre a megközelítésre alapozták a raszterizálást gyorsító hardvereiket. Bár a hardverek folyamatosan fejlődtek az évek során, eddig nem volt meg a lehetőség a klasszikus háromszög kifestésen alapuló raszterizációs modell lecserélésére. Napjainkban egy erős törekvés látszik a sugárkövetés valós időben való alkalmazására a GPU gyártók részéről is. Ez jelentősen előre mozdítaná, akár más alapokra helyezné a vizualizációt még akkor is, ha képi minőségben egy lépést vissza kellene lépni a mostani, erősen effekt alapú megoldásokhoz képest. Bár a marketing gépezetből folyamatosan ömlik a GPU „nagysága”, a fejlesztők újra fel fogják ismerni a CPU-ban rejlő lehetségeket, hiszen már 12 magos asztali számítógépek is elérhetők. A CPU alapú vizualizációk vélhetően rugalmasabb környezetet biztosítana akár részletgazdag voxel halmazokat kell megjeleníteni, akár sugárkövetéssel raszterizált poligon világot.

6. Köszönetnyilvánítás

A cikkünkben ismertetett kutatómunka az Európai Unió és a magyar állam támogatásával, az Európai Regionális Fejlesztési Alap társfinanszírozásával, a GINOP-2.3.4-15-2016-00004 projekt keretében valósult meg, a felsőoktatás és az ipar együttműködésének elősegítése céljából.

Irodalom

[1] Akenine-möller, T., haines, E.: Real-Time Rendering, A. K. Peters. 3nd Edition, 2008. **https://doi.org/10.1201/b10644**

[2] Mileff P., Dudra J.: Egyszerűsített voxel alapú vizualizáció, Multidiszciplináris tudományok, 4. kötet. (2014) 1. sz. pp. 125-134.

[3] Samuli L., Tero K: Efficient Sparse Voxel Octrees - Analysis, Extensions, and Implementation, NVIDIA Technical Report NVR-2010-001, February 2010.

[4] Daniel P.: Tracing Rays Through the Cloud, Intel Developer Zone, 2012.

[5] Szymon Rusinkiewicz, Marc Levoy: QSplat: A Multiresolution Point Rendering System for Large Meshes, SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000. **https://doi.org/10.1145/344779.344940**

[6] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, Elmar Eisemann: GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering, I3D '09 Proceedings of the 2009 symposium on Interactive 3D graphics and games, pp 15-22, 2009. **https://doi.org/10.1145/1507149.1507152**

[7] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, Elmar Eisemann: Interactive Indirect Illumination Using Voxel Cone Tracing, Computer Graphics Forum, Volume 30, Issue 7, pages 1921-1930, September 2011. https://doi.org/10.1111/j.1467-8659.2011.02063.x

[8] Sinje Thiedemann, Thorsten Grosch, Stefan Müller: Voxel-based global illumination, I3D '11 Symposium on Interactive 3D Graphics and Games, pp 103-110, 2011. **https://doi.org/10.1145/1944745.1944763**

[9] Akenine-möller, T., Haines, E.: Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs, Apress, 2019.
[10] Péter Mileff, Károly Nehéz, Judit Dudra (2015), Accelerated Half-Space Triangle Rasterization, Acta Polytechnica Hungarica, Volume 12, Issue Number 7, 2015, pp. 217-236. **https://doi.org/10.12700/APH.12.7.2015.7.13**
[11] Voxel-based Atomontage Engine, https://www.atomontage.com/, 2020.
[12] Euclideon Unimited 3D, https://www.euclideon.com/, 2020.
[13] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P.: Larrabee: a Many-Core x86 Architecture for Visual Computing. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008 Volume 27 Issue 3, August 2008 **https://doi.org/10.1145/1360612.1360617**