Multidiszciplináris tudományok, 4. kötet. (2014) 1. sz. pp. 125-134.

EGYSZERŰSÍTETT VOXEL ALAPÚ VIZUALIZÁCIÓ

Mileff Péter
Adjunktus, 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

Összefoglalás

A voxel alapú technológia már a számítógépes vizualizáció korai éveiben jelen volt, alkalmazása azonban szinte kizárólag az orvosi képszintézis területére korlátozódott. A központi egység és a grafikus hardverek folyamatos fejlődésének köszönhetően szerepe napjainkban újra megnőtt, legfőképp a számítógépes játékok területén került előtérbe jellegzetes tulajdonságai miatt. Jelen publikáció egy olyan egyszerűsített megjelenítő modellt és algoritmust mutat be, amely segítségével kisebb, esetleg animált voxel halmazok megjelenítése bizonyos vizuális kompromisszumok mellett hatékonyan elvégezhető. A megoldás célja nem a fotorealisztikus megjelenítés szintjének megközelítése, hanem egy, a mai számítógépes játékokban és egyéb grafikus alkalmazásokban jól alkalmazható megoldás nyújtása.

Kulcsszavak: *voxel alapú megjelenítés, szoftveres raszterizáció, optimalizálás*

Abstract

The voxel-based technology was already present at the early years of the computer visualization. The technique was used and almost exclusively confined to the field of medical image synthesis. Due to the continuous development of the central processing unit and the graphics hardware, today its role has increased again. It comes to the fore mainly at the field of computer games due to its specific characteristics. This publication shows a simplified visualization model and algorithm, which is able to rasterise smaller, maybe animated voxel sets effectively with some visual compromises. The aim of this approach is not to create photorealistic visual content, but provide a simple and well suited alternative solution for computer games and other graphics oriented applications.

Keywords: *voxel based rasterisation, software rendering, optimization*

1. Bevezetés

A számítógépes grafikai megjelenítést napjainkban a GPU-ra épülő poligon alapú modell uralja. Bár a voxel alapú megközelítés 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.

A hardverek teljesítményével a vizualizációs igény is folyamatosan nőtt, így ma sem mondhatjuk ki nyugodt szívvel, hogy a voxel technológia révbe ért. Egy modern számítógépes játékban több millió poligon van egyszerre a képernyőn. Mindezek voxelizált változatai rengeteg memóriát és CPU/GPU időt emésztenének föl. A mai GPU-k már képesek valós időben különböző effektekkel (pl. fény, árnyék, depth of field, stb.) bemutatni egy kisebb teret, de olyan esetekben, amikor sok modell és elem van a képernyőn a rendelkezésre álló GPU memória már valószínűleg nem elegendő. Hiába a gyors renderelő motor, ha nincs mit kirajzolni. Mindez újabb megoldandó problémát, a valós idejű, háttérben történő stream-elés technikájának kibontakozását eredményezte.

Összességében kijelenthető, hogy a világ a voxelizációs technológia hatékony bevezetése előtt áll. Különböző gyártók, a számítógépes játékipar legnagyobb szereplői folyamatosan keresik a voxel alapú kiegészítő megoldásokat a realisztikusabb megjelenítés érdekében. Ezek a technikák, algoritmusok bonyolultak, különböző területek magas szintű ismeretét igénylik. Jelen cikk éppen ezért a hangsúlyt nem a fotorealisztikus megjelenítésre helyezi, hanem a kisebb voxel halmazok egyszerű megjelenítésére. Hogyan lehetséges ezen voxel halmazokat egy kevesebb matematikai tudást igénylő megközelítéssel megjeleníteni valós időben, elfogadható minőségi kompromisszumok mellett.

2. Irodalmi áttekintés

A voxel alapú alternatív megjelenítési megoldás már a számítógépes vizualizáció korai éveiben megjelent. Mivel a korai hardverek nem tették lehetővé a professzionális, magas képminőséget lehetővé tevő megjelenítési modellek alkalmazását, így a voxel alapú technikák kevésbé váltak ismertté. Fő alkalmazási területei az orvosi diagnosztikai képalkotás és a számítógépes játékok területe volt főként sugár alapú megközelítést alkalmazva. Bár a központi egységek még nem voltak kellően gyorsak, de az úgynevezett Wave Surfing algoritmusnak köszönhetően alacsony teljesítményű gépeken valós időben vált lehetővé egyfajta kvázi háromdimenziós tér megjelenítése, melyet főként a domborzat megjelenítésére alkalmaztak (pl. Comanche - 1992, Delta Force – 1998, Armored Fist – 1994, stb.). A korai konzolokon (Amiga, Nintendo, Gameboy) szintén hasonló technológiát használtak szoftveres raszterizáció segítségével. A GPU renderelés folyamatos térnyerése után a szoftveres megjelenítés egyre inkább háttérbe szorult. Bár mindig voltak olyan szoftverek (Hexplore – 1998, Outcast – 1999, Motocross Stunt Racer - 2002, stb.), amelyek alkalmazták a voxel megközelítést, a poligon alapú technológiák kerültek előtérbe.

Az utóbbi években újra frekventált területté vált a voxel technológia. A GPU-k fejlettsége végett főként a sugár alapú megközelítések (raytracing, cone tracing, stb.) mutatnak fejlődést. Népszerűek a fa struktúrákat alkalmazó megoldások. A [3] publikációban a grafikus vezérlők egyik vezető képviselője, az NVidia vizsgálja a voxel alapú megközelítések hatékony alkalmazási lehetőségeit oktálfa alapú megközelítéssel. Az [5] publikációban egy speciális eljárást mutatnak be, amely segítségével nagyméretű voxel halmazok hatékonyan vizualizálhatók.

Napjaink trendjeit a valós idejű globális illumináció foglalkoztatja, amelyben a voxel reprezentáció egyre fontosabb szerepet kap. A vezető grafikus technológiákkal foglalkozó Unreal Technologies cég voxel alapú globális illuminációt megvalósító, deferred renderelést használó technológiát mutat be [7], mely fontossága abban rejlik, hogy a technológia kész a modern számítógépes játékok számára. A [8] és [9] cikk szintén voxel alapokon közelíti meg a modern fények kérdését.

A voxel alapú képszintézis tehát folyamatosan fejlődik, néhány éven belül már az átlag felhasználó képernyőjén is megtalálható lesz.

3. Voxel alapú megjelenítés

A voxel alapú megjelenítés nem új keletű a számítógépes vizualizációban. Az elnevezés és eljárás lényege, hogy a napjainkban elterjedt poligon alapú megjelenítéssel ellentétben úgynevezett voxelekből 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. Ezek mellett a különböző technológiákat alkalmazó megjelenítők tárolhatnak egyéb információt (pl. normálvektor) is, melyeket a realisztikusabb megjelenítéshez használnak fel (pl. Ambient Occlusion).

A voxel alapú reprezentáció számos előnyös tulajdonsággal bír a poligon alapú tárolási modellel szemben. Mivel a voxel halmaz tartalmazza a modell minden a megjelenítéshez szükséges adatát, így nincs szükség textúrára és azok mipmapping-olására. A voxelek által meghatározott szín egyértelműen megadja a modell „kinézetét". A reprezentáció további előnye, hogy egy modell felépítése nagyon részletes, atomi egységekből felépülő tud lenni. Ha megnézzük a mai tendenciákat a számítógépes játékiparban, miszerint a realisztikus megjelenítés végett hosszú távon egyre kisebb háromszögeket fognak alkalmazni, úgy mondhatjuk, hogy egyre inkább ebbe az atomi irányba konvergál a folyamat. Természetesen napjainkban még a textúra alapú megoldásokkal (pl. Normal mapping, Parallax mapping, Ambient occlusion, stb.) tolják ki a részletesebb poligonhálósítás folyamatát, mert a textúra műveleteket a GPU gyorsabban tudja elvégezni, mint újabb poligonokkal dolgozni (pl. hardveres tesszelláció).

A voxel technológia legfőbb negatív oldala a nagy adathalmazban rejtőzik. Egy még kevésbé részletes modell felépítése is viszonylag nagy voxel halmazt eredményez. A halmaz nagy mérete nagy központi, illetve GPU memóriát igényel, amely a GPU-k esetében eléggé korlátozva van. Különböző úgynevezett stream-elő technológiát kell kidolgozni a látható részek memóriában való tartására. További problémája a voxel halmazoknak, hogy mivel nagyszámú voxelt képviselnek, a különböző transzformációk során nagy adathalmazokat kell mozgatni, kezelni, amely jelentős hatással van a teljesítményre [1].

További hátránya a voxel alapú technológiáknak, 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). Továbbá nem áll rendelkezésre DirectX vagy OpenGL API rész a támogatásra. Ma már az NVidia vállalat biztosít egy Optix nevű sugárkövető motort, amely GPU alapokon képes elvégezni a sugárkövetést, de mivel nem általánosan támogatott a grafikus API-k által, így az alkalmazási lehetőségei korlátozottak. A számítógépes játékipar addig nem használ ilyen technológiát, amíg a grafikus API-k részévé nem válik.

Egy voxel halmaz reprezentációja független a megjelenítési eljárástól. A gyakorlatban számos megközelítés kialakult, melyekből jelen cikkben a legfontosabbak bemutatásra kerülnek. Továbbá egy egyszerű voxel halmazok megjelenítésére szolgáló egyszerűsített megközelítés is ismertetésre kerül.

3.1. Kocka alapú megjelenítés

A voxel halmazok legegyszerűbb, mondhatnánk naiv megjelenítési megközelítése, amikor az adathalmaz minden elemének a képernyőn egy háromdimenziós kocka felel meg. A voxelek által definiált kocka mérete előre meghatározott. Napjaink számítógépes játékaiban (pl. Minecraft, FEZ, Stonehearth, Voxatron, stb.) népszerű ez a megközelítés, amely során a nagy kocka méretekkel szándékosan szögletes megjelenítést, egyfajta retró látványvilágot terveznek a képernyőre.

A megjelenítés bár egyszerűnek tűnik, hiszen lényegében minden kocka egy adott szín által meghatározott, a gyakorlatban nagyobb megjelenített modellek/világ esetében a sok poligon szám miatt komoly problémát (több millió kocka renderelése) okoz a GPU-nak. Példaként említhetjük az árnyéktérképpel megvalósított árnyékok számítását, amikor a modelleket többször is renderelni kell. Árnyéktest megközelítés esetén pedig egy összefüggő vertex hálót kell kialakítani a fényforrásból vetített látható vertexek halmazából. Ahhoz, hogy elfogadható képernyőfrissítést lehessen elérni, számos kiegészítő optimalizációs eljárást (pl. térfelosztás, Occlusion culling, objektumok Z irányú rendezése, stb.) kell bevezetni.

1. ábra. *Példa kocka alapú voxel megjelenítésre (Voxatron)*

3.2. Sugár alapú megoldások

A voxel alapú raszterizáció további népszerű formája a sugár alapú megközelítések. A sugárvetés, mint egyszerűbb formája már a korai számítógépes játékokban (Comanche, Wolfeinstein, Outcast, Delta Force, stb.) és orvosi diagnosztikai eljárásokban megjelent. A sugár alapú megközelítések alapgondolata az, hogy a raszterizációs és takarási feladatokat a képernyő pixeleire egymástól függetlenül oldja meg. Az algoritmus sugarakat lő ki a képernyőpontokon át a térbe, majd rekurzívan vizsgálja azok terjedését, ütközési pontjait és jellemzőit.

Az eljárás nagy előnye egyszerűségében rejlik. Számos olyan vizualizációs problémát képes megoldani önmagában, amelyet a mai „forward" illetve „deferred rendering" csak különféle kiegészítő, mély technológiai és matematikai ismereteket igénylő technikák segítségével (pl. Árnyéktérképek, „Ambient Occlusion") képes elvégezni. Az eljárás a mai globális illuminációs megjelenítési megoldások elsődleges kiinduló pontját képezi. Napjainkban számos olyan törekvés bontakozik ki, ahol voxel alapokon vagy azok segítségével valós időben kísérleteznek sugárkövető megoldások alkalmazásával (pl. Epic Games - Sparse Voxel Octree Global Illumination, ID Software – Sparse Voxel Octree, NVidia – Efficient Sparse Voxel Octrees [3]).

A voxel halmazon végzett sugárkövetés önmagában a nagy adathalmaz miatt különösen lassú folyamat, így elengedhetetlen valamilyen gyorsító struktúra, általában valamilyen térfelosztó fára épülő leképzés (pl. KD-fa, BSP fa, stb.) alkalmazása. Az eljárás lényege, hogy a sugarakat ilyenkor a fa által definiált szintekkel ütköztetik megkeresve azt a voxelt, amely majd a képpont színét adja. Legfőbb hátránya tehát a magas számításigényben rejlik valamint abban, hogy a grafikus gyorsítók közvetlenül nem támogatják az eljárást. A grafikus csővezeték bár árnyalók segítségével programozható, de nem a sugár alapú algoritmusok támogatására lett tervezve. Napjaink gyors GPU-i már képesek a valósidejű megjelenítésre, azonban magas szintű grafikus demókon kívül jelenleg még kommerciális projektekben (pl. játékok) nem alkalmazzák.

4. Egyszerűsített voxel alapú megjelenítés

Az eddig röviden bemutatott technikák nem képesek minden igényt kiszolgálni. Vannak olyan esetek azonban, amikor kisebb voxel halmazokat szeretnénk megjeleníteni, elfogadható teljesítménnyel, bizonyos vizuális kompromisszumok (redukált árnyalás/árnyékolás, stb.) mellett. Példaként említhetjük a mai kétdimenziós számítógépes játékokat, ahol bár a megjelenítés kétdimenziós, bizonyos egyszerűbb felvehető elemek, modellek háromdimenziósak, forognak, vagy egyszerű animációt végeznek.

Mivel a megjelenítés nem akar retró jelleget kölcsönözni a képernyőre, a nagy kockák alkalmazása nem lehet megoldás. A megjelenített voxel halmaznak a pixel szintű kétdimenziós jelleget kell tükröznie a háttérben háromdimenziós modellként reprezentálva. A következő kép (2. ábra) bemutatja a megközelítés alkalmazásának jellegét.

A fenti megoldások egyike sem alkalmas teljes mértékben az ilyen jellegű feladatok elvégzésére. A kocka alapú megközelítés esetében nagyon kisméretű kockákat (vagy esetleg gömböket) lehetne alkalmazni a minőségi megjelenítés érdekében. Azonban olyan mértékű felesleges vertex halmaz alakulna ki, amely komoly terhelést róna a GPU-ra.

2. ábra. *Kétdimenziós játék 3D voxel egységekkel (Red Alert játék)*

Felmerülhet a kérdés ilyenkor, hogy miért van szükség egyáltalán voxel halmazra, miért nem inkább poligon alapokon valósítják meg a megjelenítést. Ezekben az esetekben maga a poligonhalmaz is sűrű lenne, de sokszor a voxelek azon jellemző tulajdonságát szeretnék kihasználni, hogy az objektum atomi részekből épül fel, bontható, rombolható jellegű.

A továbbiakban egy olyan egyszerűsített voxel megjelenítő megoldást mutatunk be, amely képes a fent definiált feladat hatékony elvégezésére.

4.1. Négyzet alapú megközelítés

Az egyszerűsített voxel halmaz raszterizáló megközelítés ismertetéséhez induljunk ki a megjelenítés logikájából. A voxel háromdimenziós egység, a képernyőn való raszterizációja minden esetben igényel valamilyen kétdimenziós matematikai leképzést, ahol a voxel színének, méretének megfelelően meg kell határozni a hozzá tartozó pixelek színeit. Felmerül azonban a kérdés, hogy ha a voxel úgyis a kétdimenziós térre lesz leképezve, miért foglalkozzunk a háromdimenziós kiterjedésével? Modellezhető-e csupán két dimenzióban a voxel kiterjedése? A problémára a megoldásként a négyzet alapú leképzés nyújt segítséget. Látni fogjuk, hogy bizonyos kompromisszumok mellett a megközelítéssel jól vizualizálhatók a kisméretű voxel halmazok.

A következő ábra 4 darab, egymás mellett és mögött elhelyezkedő összetartozó voxel négyzet alapú felnagyított leképzését mutatja be:

3. ábra. *Voxelek reprezentációja két dimenzióban*

A megjelenítés során tehát előre meghatározott méretű kifestett „lapokkal" (négyzet/téglalap) reprezentáljuk a voxeleket. A szín értéke maga a voxel által meghatározott szín. A két voxel sor közötti x és y irányú eltérés a háromdimenziós leképzés természetes velejárója, az első sor a térben előrébb található és a „modell" nem az origóban helyezkedik el.

Egy voxel kétdimenziós leképzését pszeudokóddal a következőképpen írhatjuk fel:

```
float per_z = 1.0f/z;
float voxel_size = 0.8f;

x_screen = +(( y - voxel_size) * per_z) + half_screen_width ;
y_screen = -(( x - voxel_size) * per_z) + half_screen_height;
x_screen2 = +(( y + voxel_size) * per_z) + half_screen_width;
y_screen2 = -(( x + voxel_size) * per_z) + half_screen_height;
```

Az összefüggésekben (*x*,*y*,*z*) jelenti a voxel térbeli koordinátáit, *voxel_size* paraméter pedig a kétdimenziós leképzés mérete, amely tetszőlegesen hangolható. A további összefüggések pedig a leképzés négy koordinátáját határozzák meg.

4.1.1 Voxelek kirajzolása

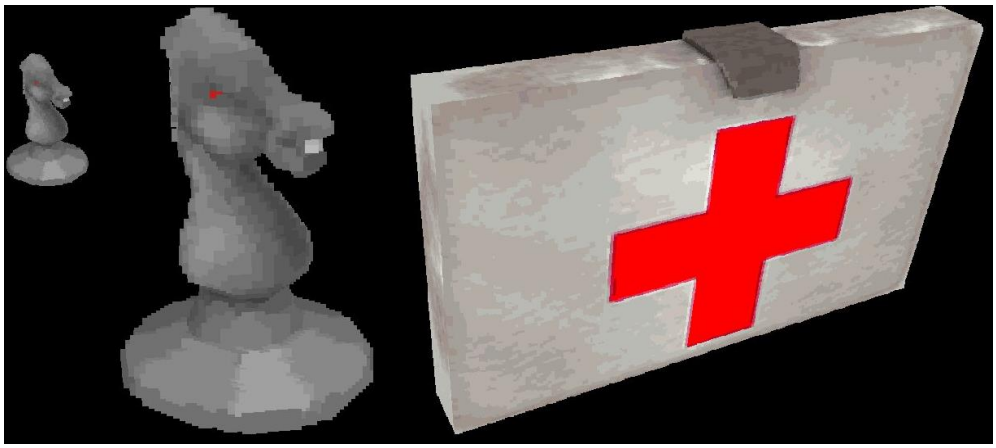
A voxel kirajzolásának legegyszerűbb módja egy szoftveres megjelenítő alkalmazása. Minden voxel kirajzolása a szoftveres framebuffer-be történik, majd a buffert a grafikus megjelenítőnek átadva megjelenítődik a képernyőn. A megoldás nem összeférhetetlen a GPU alapú vizualizációval, a két megjelenítő integrálása ilyenkor a valódi cél. A hardveresen gyorsított modellek kirajzolása közben vagy utána a kirajzolás megfelelő fázisában a szoftveres buffer is kirajzolása kerül.

Egy voxel kirajzolását mutatja be a következő pszeudó leírás:

```
for  i=x_scr to x_scr2
   for j=y_scr to y_scr2
         index = i * framebufferWidth + j;
         if (j >= framebufferWidth || i >= framebufferHeight || j <= 1 || i <= 1)
            continue;
         if (voxel.z < zBuffer[index] ) {
                  zBuffer [index] = voxel.z;
                  m_pFrameBuffer[index] =  voxel_color;
         }
```

A megoldáshoz jól láthatóan szükség van egy z-buffer implementációra is. Ennek oka, hogy a voxel halmaz térbeli elhelyezkedése miatt a téglalapok/négyzetek bizonyos részeken fedhetik egymást.

Egy példa modell segítségével a vizualizáció eredményét az alábbi képek foglalják össze:



4. ábra. *Kétdimenziós voxel reprezentáció különböző z értékek esetén*

Az 4. ábra két modellt ábrázol, melyek közül az elsőt különböző z távolságokból. Látható, hogy a ló figurát távolabbról nézve kielégítő eredményt kapunk, közelről azonban már megjelennek a szögletes reprezentáció jellegzetességei. A szögletesség mértéke a voxel sűrűsséggel és a méret paraméter hangolásával megszűntethető, azonban a számítási idő így jelentősen megnőhet. A második modell egy 1.548.288 darab voxelből álló halmaz. Láthatóan a vizuális eredmény lényegesen jobb, azonban a teljesítmény a lónál (49.152 voxel) mért 355 FPS-ről (optimalizáció nélkül) 63-ra esett. Éppen ezért az eljárás valós idejű megjelenítésre szánva főként kisebb modellek nem túl közeli távolságokból való vizualizációjára alkalmas.

4.2. A megjelenítés gyorsítása

Az eljárás – bár nem tartalmaz semmilyen matematikai formulát – a dupla iterációs ciklus miatt meglehetősen számításigényes. Minden voxel esetén be kell festeni a buffer egy területét. Mindezt pedig akár redundánsan is elvégezve, ha a voxelek éppen úgy helyezkednek el, hogy hátulról előrefelé kell kirajzolni őket. Ilyenkor a rajzolási sorrend miatt a z buffer nem képes visszautasítani a nem látható pixeleket, a pixelek folyamatosan felülírásra kerülnek. A megjelenítés tehát mindenféleképpen valamilyen gyorsítási kiegészítést igényel, amelyek során csökkenteni kell a belső iterációk számát.

Az egyik legfontosabb gyorsítási lehetőség a nem látszó voxelek kihagyása. Olyan reprezentációt kell felépíteni a voxel halmaz számára, amely képes meghatározni a külső voxeleket, a raszterizáció során így jelentős terheléstől szabadul meg a megjelenítő.

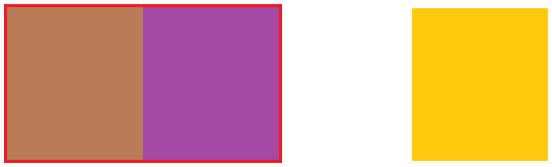
További, megfigyelésen alapuló gyorsítási lehetőség lehet, amennyiben a megjelenítés z távolsága is változik, hogy egy bizonyos távolságon túl nincs értelme négyzeteket rajzolni a framebufferbe. A távolság miatt a voxel határoló pontjai összemosódnak, így elegendő, ha a megadott távolságon túl minden voxelnek egy pixelt feleltetünk meg. Ezzel a kiegészítéssel a sebesség szintén jelentősen javítható.

A korábban említett „szerencsétlen” voxel renderelési sorrend miatti redundáns raszterizáció szintén eliminálható. Erre két megoldás is adódik. Egyrészt vagy rendezzük a

voxeleket z irányban, majd a legközelebbivel kezdjük a rajzolást, vagy rendezés nélkül meghatározzuk azokat az eseteket, amelyek során megadjuk, hogy melyik kamera állásból melyik tér negyed látszik a modellből, így pedig a megfelelő voxelek rajzolása előre hozható. Nevezhetjük egyfajta nézőpont orientált megjelenítésnek is.

4.2.1 Összefüggő voxel részhalmazok

Amennyiben globálisan nézzük a voxel kirajzoló eljárást, jelentős redundancia figyelhető meg a számításokban. Az eljárás folyamatosan halad végig a voxel „sorokon", kiszámítja vetített reprezentációjuk pontjait, majd kifesti a buffer megfelelő részét. Az egymás mellett elhelyezkedő voxelek esetében azonban nincs értelme újra végighaladni a számítások egy részén (pl. vetítés), hiszen mivel egy síkban helyezkednek el, a pozíciójuk vetítés nélkül iteratívan kiszámítható. A raszterizáló ciklus így jelentősen felgyorsítható. Azon voxeleket tekintjük egyazon csoport részének, amelyek y koordinátája azonos, x koordináta alapján pedig egymás mellett helyezkednek el lényegében egy láncot alkotva egészen a legszélső vagy egy üres voxelig. A következő egyszerű ábra piros körvonallal határolja körbe azon voxeleket, amik egy csoportba tartoznak.



5. ábra. *Voxel részhalmazok értelmezése*

A megoldás implementálása jelentős változtatást igényel az alap struktúrához képest, amikor a voxelek tulajdonságát külön-külön tároljuk. Szükség van egy befoglaló adatstruktúrára, amely egyértelműen azonosítja az egymás melletti voxeleket. A struktúra előnye, hogy a transzformációk során a műveletet ilyenkor nem a voxeleken külön-külön kell elvégezni, hanem elég az összefüggő részhalmazon. A raszterizáció során pedig ezeket a kiszámolt csoportos paramétereket használjuk fel a voxelek kirajzolásához.

5. Összefoglalás

A számítógépes vizualizáció területét ma a poligon alapú modell reprezentáció uralja, a grafikus hardvergyártók erre a megközelítésre alapozták a raszterizálást gyorsító hardvereiket. A voxel alapú technológia már a kezdetektől fogva jelen van, azonban csak az utóbbi években kezdett kibontakozni a számítógépek megfelelő teljesítménye miatt. A cikkben bemutatott eljárás egy olyan űrt próbál meg betölteni a népszerű voxel megjelenítési algoritmusok területén, amely bizonyos vizuális kompromisszumok mellett lehetővé teszi a kisebb voxel halmazok gyors megjelenítését. A megoldás teljesítményben nem veszi fel a versenyt a poligon alapú vizualizációval, azonban a voxelizáció előnyös tulajdonságai miatt bizonyos területeken jól alkalmazható. Jó példa erre, ha a grafikus

motorban ötvözzük a voxel alapú megoldásokat a poligon alapú vizualizációval, lehetőséget biztosítva ezzel a fejlesztőnek a kevert megjelenítési technológia használatára.

6. Köszönetnyilvánítás

A kutató munka a TÁMOP-4.2.2.B-10/1-2010-0008 jelű projekt részeként – az Új Magyarország Fejlesztési Terv keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

7. Felhasznált irodalom

[1] Mileff P., Dudra J.: *Efficient 2D Software Rendering*, Production Systems and Information Engineering, Volume 6, 2012. pp. 99-110.

[2] Akenine-möller, T., haines, E.: *Real-Time Rendering*, A. K. Peters. 3nd Edition, 2008.

[3] Agner, F.: *Optimizing software in C++ An optimization guide for Windows, Linux and Mac platforms*. Study at Copenhagen University College of Engineering, 2011.06.08.

[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.

[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.

[7] Martin Mittring: *The Technology Behind the "Unreal Engine 4 Elemental Demo*, The 39th International Conference and Exhibition on Computer Graphics and Interactive Techniques, 2012.

[8] 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.

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

[10] Vega-Higuera, F., Hastreiter, P., Fahlbusch, R., Greiner, G.: *High performance volume splatting for visualization of neurovascular data*, Visualization, 2005. VIS 05. IEEE, pp 271 - 278, 2005.

[11] Sakamoto, N.; Nonaka, J.; Koyamada, K.; Tanaka, S.: *Particle-based volume rendering*, Visualization, 2007, APVIS '07. 6th International Asia-Pacific Symposium, pp 129 - 132, 2007.