HATÉKONY TECHNIKÁK ÉS MEGOLDÁSOK A SZOFTVERES RASZTERIZÁCIÓBAN

EFFICIENT TECHNIQUES AND METHODS FOR SOFTWARE RENDERING

Mileff Péter, Dudra Judit***

ABSTRACT

Computer visualization, under continuous changes has arrived again to a major milestone. Necessary modifications are required at the currently applied physical devices and technologies. This publication presents an overview about new performance improvement methods, what today CPUs can provide utilizing their modern instruction sets and how these methods can be applied in the specific field of the computer graphics, called software rendering. Furthermore the paper focuses on the GPGPU based parallel computing as a new technology for computer graphics where a TBR based software rasterization method is investigated in terms of performance and efficiency.

1. BEVEZETÉS

A számítógépes grafika területe sok éves fejlődésen ment keresztül az utóbbi néhány évtizedben. Legfontosabb állomása a grafikus célprocesszorok megjelenése volt, amely számtalan új lehetőséget nyitott meg a fejlesztők előtt.

Bár kezdetben a kártyák grafikus csővezetéke és programozhatósága nagyon egyszerű modellt nyújtott, az addig érvényben lévő szoftveres raszterizáció gyorsan elvesztette jelentősségét, mert az akkori CPU-k nem voltak képesek versenyezni a célhardverek teljesítményével. A gyártók és az ipar szemszögéből elsősorban a gyorsaság került előtérbe a robosztusság és programozás rugalmassággal szemben.

Az utóbbi években a videókártyák technológia fejlődése ezért főként a csővezeték programozhatóságának növelésére irányult. Ma a fejlődés pedig már egészen kiforrot irányba halad tovább, bevezetve a grafikus processzorok egy teljesen új generációját, az általános célú grafikus processzorokat, melyek már nem csupán a megjelenítés gyorsítására alkalmasak, hanem a CPU-hoz hasonlóan általános számítások végzésére is.

Mindezek ellenére ki kell emelni a GPU alapú számítógépes raszterizáció problémáit. A programozási logikája és modellje lassan eléri határait, a fejlődés intenzitása láthatóan csökken. Hiába áll rendelkezésre gyors hardveres támogatású csővezeték, az évek során a hardver korlátaihoz igazított csővezeték nem ad kellő rugalmasságot a programozónak. A GPU egység általános célú programozása pedig a memória modell és a párhuzamos szálak futtatását végző fix funkciós blokkok miatt limitált [1]. Jól mutatja ez a mai számítógépes játékok homogén és kiszámítható „kinézete" és viselkedése. A mai GPU architektúra tehát megkérdőjelezhető és átalakításra szorul, melyet a vezető ipari felhasználók is erősen szorgalmaznak [2,15].

Mindezek mellett az elmúlt években a számítógépek központi egysége hatalmas fejlődésen ment keresztül. A processzor gyártók kibővített utasításkészlettel reagáltak a piaci igényekre, lehetővé téve a CPU-kon is a gyorsabb és főként vektorizált (SIMD) feldolgozást.

Az új technológiáknak köszönhetően az utasításkészletet megfelelően kihasználó szoftver akár 2-10x sebességnövekedést is elérhet. Mindehhez társítva a GPGPU technológiát felmerül a kérdés, hogy miért ne lehetne készíteni egy teljesen szoftveres megvalósítású grafikus csővezetéket. Jelen cikk központi témája tehát annak a vizsgálata, hogy miként lehetséges egy a mai központi egységekben rejlő technológiai előnyöket kihasználó, előre mutató szoftveres megjelenítőt készíteni.

2. IRODALMI ÁTTEKINTÉS

A szoftveres képszintézis az első számítógépek óta jelen van domináns szereppel bírva egészen 2003-ig, a GPU megjelenéséig. A korai évek során született raszterizálók közül legkimelkedőbb eredmény az ID software által készített Quake I, II szoftveres renderer (1996), amely az első MMX utasításkészletre optimalizált valós háromdimenziós motor volt. A később született eredmények közül főleg az Unreal motort (1998) lehet kiemelni, amely szintén nagyon gazdag funkcionalitással rendelkezett.

* *adjunktus, Miskolci Egyetem, Ált. Inf. T.*

** *tudományos munkatárs, Bay Zoltán Alkalmazott Kutatási Közhasznú Nonprofit Kft., Miskolc*

A GPU renderelés folyamatos térnyerése után a szoftveres megjelenítés egyre inkább háttérbe szorult. Ennek ellenére született néhány nagyszerű eredmény, mint a Rad Game Tools által fejlesztett Pixomatic 1, 2, 3 [3] és a TrasGaming által készített Swiftshader [2]. Mindkét termék nagyon komplex, magas szinten optimalizált raszterizációt tesz lehetővé.

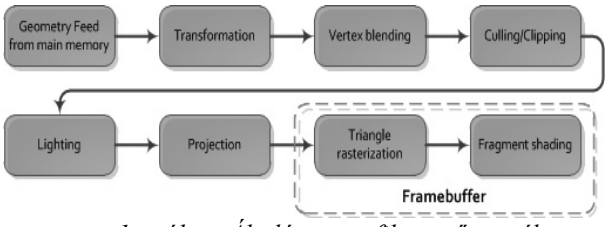
Bár a Microsoft a DirectX fejlesztésével nyújtott alapot a GPU technológiák terjedésének, mindezek mellett kifejlesztette saját szoftveres megjelenítőjét a WARP-ot [1]. A renderer jól skálázható több szálra egyaránt és teljesítményben felveszi a versenyt az alacsony kategóriás integrált grafikus kártyákkal.

A problémákat és az igényeket jól felmérve az Intel 2008-ban egy hibrid, x86 alapú videokártya (Larrabee) fejlesztését tűzte ki célul [9], melynek célja egy teljesen programozható csővezeték kidolgozása volt [4].

Loop and Eisenacher [2009] parametrikus felületek számára készített GPU alapú szoftveres megjelenítőt. Az NVidia pedig saját GPGPU CUDA platformjának segítségével készített tanulmányt [6]. Napjaink vezető grafikai színvonalú számítógépes játéka [5] új technológiájaként egy részben SPU alapú raszterizációt valósított meg a fényforrás hatékonyan kezelésére. Az [1] publikációban a szerző egy több szálat használó modern „Tile” alapú megjelenítő technikát vázol.

3. SZOFTVERES RASZTERIZÁCIÓ

Szoftveres raszterizációnak nevezzük azt a képalkotási folyamatot, amikor a képszintézis teljes folyamatát a számítógép központi egysége (CPU) végzi. Az alakzatokat felépítő geometriai primitívek a központi memóriában helyezkednek el. A képalkotás logikája egyszerű: a központi egység elvégzi a szükséges műveleteket a tárolt adatokon, ennek eredményét eltárolja egy *framebuffer*ben, majd az elkészített képet kiküldi a videó vezérlőnek. A következő ábra ennek általános csővezetékét mutatja be.



1. ábra. Általános grafikus csővezeték

Az 1. ábrán bemutatott csővezetékben a számítások két domináns csoportja alakul ki egy képszintézis során. Az első csoportba főként *vertex* transzformációs műveletek tartoznak, melyek egészen a *Framebuffer* műveletekig tartanak. Ezekben a fázisokban számos mátrix és vektortranszformációval előkészítjük a pixel szintű raszterizációt. A második csoportba olyan pixelenkénti operációk tartoznak, mint a háromszögek diszkretizálása és a pixelek árnyalása. A grafikus motorok számára a megjelenítés szempontjából e két csoport, de főként a második a leginkább számításigényes feladat.

4. ÁLTALÁNOS GYORSÍTÁSI LEHETŐSÉGEK

A mai modern processzor architektúrának köszönhetően számos lehetőség áll rendelkezésre a grafikus csővezeték számításigényes részeinek gyorsítására. Az igazán jó eredmény elérése érdekében természetesen ezek együttes alkalmazása szükséges, azonban jelen cikkben terjedelmi okok miatt csak néhány megközelítést emelünk ki.

4.1 SSE alapú csővezeték optimalizálás

Az utóbbi évek során született legfontosabb újítás a bevezetett SIMD processzor utasításkészletek köré csoportosul (Intel – SSE család, AMD - 3DNow, Apple - AltiVec, ARM - NEON). Segítségükkel lehetővé válik a számítások vektorizált gyorsítása, melyekkel akár többszörös sebességnövekedés is elérhető. Fontos szempont azonban a programozhatóság és a hordozhatóság kérdése is. Az SSE család mára már széles körben elfogadott, az utasításkészlet alapú programozást pedig a mai fordítók (GCC, Intel) már jól támogatják. A modern fordítók több lehetőséget is biztosítanak erre. Egyrészt Assembly kódként implementálni, amely mély programozói ismereteket igényel és a programkód nem lesz mindig hordozható, vagy használni a fordítók magasabb szintű *Intrinsics* függvénykönyvtárát. A második lehetőség (pl. *GCC: __m128 z = _mm_setzero_ps();)* már elég magas szintű, kényelmes programozást tesz lehetővé és nem korlátozza a hordozhatóságot sem.

4.1.1 SSE alapú vektor optimalizálás

Az SSE (Streaming SIMD Extensions) az Intel által kidolgozott SIMD instrukciókészlet család (jelenleg SSE4.2) az x86 architektúrák számára. Legfőbb újítása a 8 darab, külön-külön 128 bit hosszúságú SSE vektor (xmm0 ~ xmm7). A kibővített utasításkészlet pedig lehetőséget nyújt arra, hogy két vektoron elvégzett műveletet a processzor a vektor adatain párhuzamosan hajtsa végre.

A grafikus csővezeték első műveleti csoportja jellemzően valamilyen nagyobb adathalmazon elvégzendő vektortranszformáció. Háromdimenziós megjelenítés esetén tehát az SSE instrukciókészlet alkalmazása lehetővé teszi, hogy a 128 bites SSE vektorban 4 darab különböző 32 bites floating point számot tároljunk, mégpedig x,y,z, és w-t. Ez azt jelenti,

hogy ezen a 4 számon egyszerre tudunk számításokat végezni, amely jelentős sebességnövekedést eredményez a csővezeték transzformációs műveleteinek végrehajtásában.

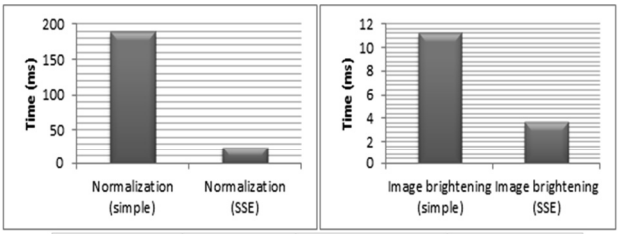
4.1.2 SSE alapú képfeldolgozás

Az SSE család szintén sikeresen alkalmazható a csővezeték raszterizációs fázisában, mert a pixeleken végzett műveletek nagy része független egymástól, így párhuzamosan egyszerre végrehajtható. A mai grafikus motorok tipikusan 32 bites (RGBA) színkomponensű megjelenítést használnak, ahol minden komponens 4 byte hosszúságú. Ez a fajta leképzés jól illeszkedik az SSE alapú megközelítéshez, mert egy 128 bites regiszterben 4 darab különböző pixel színét tudjuk tárolni és műveletet végezni rajta párhuzamosan.

4.1.3 SSE teszteredmények

A továbbiakban két tesztfeladaton keresztül bemutatjuk az SSE megoldás hatékonyságát. Az első feladatban egy minden grafikai motorban megtalálható számítást, a vektor normalizációt vizsgáltuk. A formula számításigényes, mert gyökvonást és osztást is tartalmaz. A feladatban 50.000 vektor normalizációját ismételjük 200.000-szer.

A második teszt számára egy mindennapi pixelszintű képfeldolgozási feladatot választottunk. Példánkban egy 32 bites 1024x768 felbontású képen egy világosítási transzformációt végzünk el 1000-szer. A programok elkészítéséhez az SSE2 utasításkészletet, a C++ nyelvet és a GCC 4.4.1 fordítót használtuk, a méréseket pedig egy Core i7 870-es 2.93GHz CPU-val végeztük. Az alábbi táblázat tartalmazza mindkét feladat eredményeit:



Normalization (simple)	Normalization (SSE)	Image brightening (simple)	Image brightening (SSE)
188,7903 ms	24,1256 ms	11,2148 ms	3,6719 ms

2. ábra. Számítási eredmények összehasonlítása

Az eredmények kiértékelése jól mutatja az SSE alapú programozás erősségét. Míg az első esetben ez 7.8 szoros, addig a második teszt 3.05 szörös sebességnövekedést jelent a hagyományos kódhoz képest.

4.2 Adatok igazítása

A grafikus csővezeték, így a raszterizáció sebessége tovább gyorsítható, ha az adatokat a memóriában megfelelően igazítva tároljuk. Az igazítás azt jelenti, hogy az adat címe milyen értékkel (1,2,4,8) osztható, reprezentálva ezzel az igazítás byte hossz értékét. A CPU nem byte-onként olvassa vagy írja a memóriát, hanem 2, 4, 8, 16, vagy 32 byte-os darabokban. Ezért ha a grafikus csővezeték összefüggő adathalmazait nem igazítjuk megfelelően 4, 8, vagy 16 byte-os határra, akkor ez súlyos sebességcsökkentést idézhet elő, mert a CPU-nak extra műveletekre van szüksége az adatok betöltésekor [6].

Az igazítási probléma a csővezeték struktúrái esetén viszonylag egyszerűen orvosolható az alacsonyabb szintű nyelvekben (pl. C, C++, D). Az igazítást mindig a leginkább korlátozó tagra kell alapozni, amely általában a legnagyobb valós típus. Ezért egy struktúra adattagjait tárolási méretük alapján csökkenő sorrendbe kell rendezni, hogy igazított memóriaterületű struktúrához jussunk. Nagyobb struktúratömbök esetén így nemcsak memóriát takarítunk meg, hanem jelentősen növelhetjük a raszterizáció hatékonyságát.

5. GPGPU ALAPÚ SZOFTVERES CSŐVEZETÉK

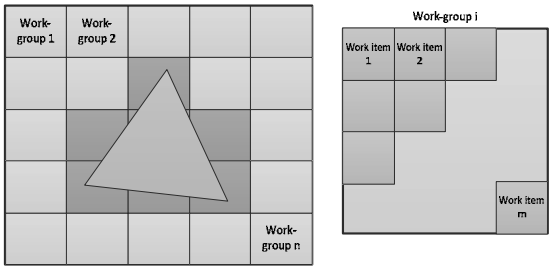
A GPU alapú megjelenítési technológia napjaikban az általános célú feldolgozás irányába halad tovább. A legújabb GPGPU kártyák hatalmas számítási kapacitást rejtenek (~1 Tflops/s) magukban köszönhetően a bennük rejlő egyre növekvő processzorok számának (pl. NVidia GTX 480, 480 mag), a gyors elérésű memóriának (GDDR5), és a fejlett technológiának.

Logikailag a GPU általános célú lehetőségei a grafikus csővezeték bármely szakaszaiban használhatók bizonyos korlátozásokkal. Mégis mivel a raszterizáció lényegesen számításigényesebb feladat, így célszerű, hogy a projekciós fázistól kezdve a GPU végezze el a feladatokat kihasználva a GPU párhuzamosítási erősségeit.

5.1 GPGPU alapú képszintézis

A raszterizáció fázisának feladata a megjelenítendő alakzatok háromszög halmazainak pixelekre való leképzése. A folyamat GPGPU-val való ideális megoldáshoz figyelembe kell venni az eszköz jellegzetességeit és programozás nyelvi (OpenCL, CUDA) lehetőségeit. A programozási modell legkisebb egysége a végrehajtandó kódot (kernel) futtató work-item, melyek work-group-okba csoportosulnak. Minden work-group külön processzoron fut szeparáltan, az egy csoportba tartozó work-item-ek pedig azonos számítási egységen. Ezen jellegzetességekhez logikailag legközelebbi a „Tile” alapú megjelenítés áll

legközelebb. A következő ábra a TBR renderelési megoldást szemlélteti a GPGPU szemszögéből.



3. ábra. GPGPU alapú ideális képszintézis

A megoldás ötlete, hogy a *framebuffer*t felosztjuk azonos méretű területekre. A párhuzamos végrehajtás kihasználása végett minden területet egy külön work-group-hoz célszerű rendelni. A raszterizáció során a csővezeték minden háromszögét egy-egy területhez rendeljük az alapján, hogy átfedi-e vagy sem. Minden terület egymástól függetlenül, párhuzamosan kerül feldolgozásra külön processzoron, ahol a pixelenkénti raszterizációt pedig a work-item-ek végzik el. A framebuffer felosztásánál célszerű tipikusan 16x16 vagy 32x32 méretű részeket választani. A felbontás ekkor ideális a mai videokártyák hardveres jellemzőik alapján.

A csoporton belül egy „Tile" képének előállításáért a work-item-ek felelősek. Sorra veszik a csoporthoz tartozó háromszögeket, kiszámítják azok határait, és elvégzik a raszterizációt. Egy csoporton belül a work-item-ek szintén párhuzamosan futnak és azonos lokális memóriaterület használnak. Minden item egy külön háromszöget dolgoz fel. A háromszögek takarási problémáinak megoldására az itemek közötti szinkronizáció nyújt lehetőséget. A teljes kép akkor készül el, amikor mindegyik csoport elvégezte a saját feladatát.

5.2 Megszorítások

A GPU és a központi memória távol van egymástól és így az adatmozgatás a GPU és központi memória között a PCIe bus transzfer sebessége által erősen limitált (~2.4 GB/s). Célszerű tehát a csővezeték háromszöghalmazának minden adatát a kártyamemóriában tárolni és minimalizálni az adatok mozgatását. Rövid tesztként egy központi memóriában tárolt 1024x768 méretű 32 bites üres *framebuffer*t osztottunk meg a GPU számára. Egyéb számítást nem végzünk, csak az adatok megosztását és a *framebuffer* megjelenítését. A tesztben felhasznált hardver: ATI Radeon HD 5670 1 GB RAM. A csővezeték GPU megosztás nélküli változatában az üres *framebuffer* rajzolási sebessége ~1100 FPS, míg a GPU megosztás következményeképpen ez rögtön ~620 FPS-re redukálódott.

További probléma, hogy egy kernel futtatása egy meghatározott előkészítési időt igényel. Második tesztként megvizsgáltuk, hogy hogyan befolyásolja a kernel inicializálása a futási sebességet. A tesztben egy üres kernelt hoztunk létre, amelynek 4 darab float típusú változót osztottunk meg. A sebesség ezúttal ~580 FPS-re redukálódott. Természetesen vannak lehetőségek a kommunikációból fakadó sebességvesztés javítására. Például ha az egész framebuffert a kártya memóriájában tároljuk OpenGL textúraként. Ezzel azonban elveszítjük a csővezeték szoftveres jellegének egy részét.

4. ÖSSZEFOGLALÁS

A cikkben bemutatott technikák rámutatnak arra, hogy egy megfelelően gyors szoftveres raszterizáló készítése sok erőfeszítést igényel. Nélkülözhetetlen a több technológia együttes alkalmazása valamint alacsonyabb szintű nyelvek használata. A központi egység sokat fejlődött az elmúlt években, lehetőségeinek megfelelő kihasználásával a szoftveres csővezeték sebessége sokszorosára gyorsítható. Bemutattuk, hogy miként alkalmazható a GPGPU technológia új megközelítésként a raszterizációs fázisban. A párhuzamosítási lehetőségei bizonyos korlátok mellett jól adaptálhatók a TBR renderelési eljáráshoz.

5. KÖSZÖNETNYILVÁNÍTÁS

A bemutatott kutató munka a TÁMOP-4.2.1.B-10/2/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

6. IRODALOM

[1.] ZACH B.: A Modern Approach to Software Rasterization, University Workshop, Taylor University 2011.

[2.] SWENNY T.: The End of the GPU Roadmap. Proceedings of the Conference on High Performance Graphics, pp. 45-52, 2009.

[3.] RAD GAME TOOLS: Pixomatic Advanced Software rasterizer, 2012.

[4.] TRANSGAMING Inc: Swiftshader Software GPU Toolkit, 2012.

[5.] COFFIN C.: SPU-based Deferred Shading for Battlefield 3 on Playstation 3. Game Developer Conference Presentation, March 8, 2011.

[6.] LAINE S., KARRAS T.: High-Performance Software Rasterization on GPUs. High Performance Graphics, Vancouver, Canada, aug 5. 2011.