FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA XVII.

Kolozsvár, 2012. március 22–23.

MEGMUNKÁLÁS SZIMULÁCIÓ GPU SEGÍTSÉGÉVEL

SIMON Pál, NEHÉZ Károly

Abstract

Modern Graphics Processing Units (GPUs) are multi-core processors that can be used not only in image synthesis, but also for general computing. Taking advantage of the ability to executing tasks massively parallel way, the task execution time can be significantly reduced. Therefore general-purpose computing (GPGPU) on graphics processors are becoming widely used. Thus, they can be used for simulation such as a machining simulation. In this paper, we would like to introduce the potential use of the GPGPU in the milling simulation, where a significant computation time reducing can be achieved. Analytical machining simulation techniques are using CSG operations on implicit surfaces. This simulation method with real-time material removal will be useful for NC programmers, engineers and machining operators for identifying NC programs and checking programming errors visually.

Key words:

GPGPU, OpenCL, milling simulation, implicit surfaces, CSG

Összefoglalás

A modern grafikus kártyák (GPU) már nemcsak képszintézisre használhatók, hanem általános számításokra is. Hatékonyságuk akkor jelentős, ha kifejezetten párhuzamos algoritmusokat futtatunk rajtuk. A megmunkálás szimulációban az anyagleválasztási folyamat számítása jól párhuzamosítható. Jelen cikkben megmutatjuk a CSG módszer megvalósítását grafikus kártyán, OpenCL segítségével. Az anyagleválasztás szimulációban a munkadarab és a szerszám egyaránt implicit formában felírt függvényekkel reprezentált. A példa implementáció letölthető formában is rendelkezésre áll.

Kulcsszavak:

GPGPU, OpenCL, marás szimuláció, implicit felületek, CSG

1. Bevezetés

Napjaink grafikus kártyái már nem csak kizárólag képi megjelenítésre használhatóak. A rajtuk található általános processzor, a GPU képes általános számítások elvégzésére és erősen párhuzamos architektúrájú. A grafikai algoritmusok kifinomultsága miatt, ezeket a hardver eszközöket felhasználók által programozhatóvá tették, ami ahhoz vezetett, hogy a grafikus processzorokat általános számítási műveletek végrehajtására is használni kezdték, mivel a nagyfokú párhuzamossága miatt a számítási teljesítménye jóval meghaladja a CPU számítási teljesítményét. Mára már különféle fejlesztőkörnyezeteket és függvénykönyvtárakat fejlesztettek az általános számítási feladatokhoz. [3]

A GPU-k képesek akár 150-szer gyorsabban is elvégezni egyes feladatokat a hagyományos processzorokhoz képest, a probléma kisebb feladatokra osztásával, majd az eredmények későbbi egyesítésével.

2. Grafikus processzorok és előnyeik

A GPU architektúrájának köszönhetően erősen párhuzamos műveletvégzésre alkalmas, emellett nagy és gyorsan elérhető gyorsító tárat is tartalmaz. Eredetileg ezek a processzorokat arra fejlesztették ki, hogy képernyőképet állítsanak elő 3D modellekből és textúrákból a lehető leggyorsabban. Így a belső felépítésük és utasítás készletük ennek a megvalósítására lett megtervezve. A jelenlegi GPUk több mint 400 lebegőpontos számításra képes processzormagot tartalmaznak, így számítási teljesítményük többszörösen meghaladja a CPUk számítási teljesítményét. A jelenlegi leggyorsabb Tesla GPU dupla pontosságú lebegőpontos számítási sebessége 515 Gflop [2]. Ezt a számítási sebességet a jelenlegi CPU-k meg sem közelítik. A GPUk a belső memóriájukat gyorsabban tudják elérni, mint a CPU a rendszermemóriát. A GPUk belső memóriája 128MB-tól akár 6GB-ig is terjedhet és a sávszélessége akár 144 GB/sec is lehet [1]. Egy számítógépen belül akár több grafikus kártyát is lehet alkalmazni, így a rendszer számítási teljesítménye még inkább megnövelhető. A GPU-k architektúráját folyamatosan fejlesztik és nem csak a számítási teljesítmény növelése céljából, hanem a könnyebb programozhatóság érdekében is. A kezdeti szigorúan feladat-párhuzamos működést támogató architektúrát leváltotta az adat-párhuzamos működési architektúra [3]. Napjainkban a grafikus kártyákat gyártó AMD és NVIDIA egységesített shader architektúrájú GPU-kat fejleszt. Ennek az egységesítésnek köszönhetően a programozhatóságuk egyszerűbb lett. Az alkalmazások ezáltal ugyanolyan architektúrájú, de különböző hardverkörnyezetben futtathatóak.

2.1 Általános célú GPU alkalmazás előnyei és programozási eszközeik

A CPU-t és a GPU-t más céllal fejlesztik, ebből kifolyólag más-más feladatokat képesek hatékonyan végrehajtani. A CPU a számítógép központi egységeként sok folyamat és szál kezelését végzi hatékonyan. A szálak közötti kontextus váltásokra és az egyes szálak szekvenciális végrehajtására tervezték. Ezzel szemben a GPU-t egyetlen folyamat szálainak párhuzamos végrehajtásra tervezték. Emiatt jól lehet alkalmazni olyan esetekben, ahol nagy adathalmazon kell műveleteket végrehajtani és az egyes adatok között minimális a függőség. A GPU-k programozása szinte tetszőleges programozási nyelven történhet, csak a megfelelő függvénykönyvtárakat kell használni. Több ilyen függvénykönyvtár használata is elterjedt. A Compute Unified Device Architecture (CUDA), az nVidia párhuzamos programozási eszköztára, melynek segítségével csak az nVidia által gyártott grafikus kártyák programozhatóak. Ez az eszköztár könnyen használható GPUra optimalizált matematikai függvénykönyvtárakat tartalmaz C és C++ programozási nyelvhez. Az Open Computing Language, egy olyan szabvány, melynek segítségével párhuzamosan programozható heterogén rendszerek. A heterogenitás ebben az esetben azt jelenti, hogy ugyanaz a kód képes módosítás nélkül futni bármelyik gyártó GPU-ján vagy CPU-n. Hasonló használhatóságot biztosít mint a CUDA eszköztár, de annak megkötési nélkül.

3. Prototípus marás szimulátor

A megmunkálás szimuláció problémájának matematikai leírása a következő [5]. Legyen M vonalszakaszok halmaza a térben, amely leírja a szerszámmozgást. Legyen f függvény a szerszám és alakjuk szerint ujjmaró (1) vagy gömbvégű maró (2).

$$f(x, y) := -r, \quad x, y \in D(f) := \{(x, y) \mid x^2 + y^2 \leq r^2\} \quad (1)$$

$$f(x, y) := -\sqrt{r^2 - x^2 - y^2}, \quad x, y \in D(f) := \{(x, y) \mid x^2 + y^2 \leq r^2\} \quad (2)$$

Mindkét esetben az origót tekinthetjük a programozott pontnak. Az x és y értékei leírják a szerszámmozgást a nyersdarab vízszintes síkjában, z pedig a magasságot jelenti. Ha a munkadarabot egy téglatestnek tekintjük $xmin, xmax, ymin, ymax, zmin, zmax$ méreteivel megadva, akkor az anyagleválasztás matematikailag a következő implicit formulával írható le (3).

$$F(x, y) := \min\{z_{\max}, \min\{z \mid z := s(x', y') + f(x - x', y - y'), (x - x', y - y') \in D(f), z \geq z_{\min}, (x', y', s(x', y')) \in M\}\} \quad (3)$$

Az $F(x, y)$ függvény kiszámítása időigényes feladat lehet, főleg ha szabadformájú felületek esetén ahol a szerszámhelyzetek száma több tízezer is lehet. Az irodalomban egzakt és analitikus megközelítések léteznek $F(x, y)$ meghatározására. Jelen cikkben a OpenCL technológiát alkalmazunk GPU segítségével.

3.1 A prototípus alkalmazás bemutatása

Az irodalomban implicit függvények megjelenítésére számos módszer ismert: az u.n. masírozó kockák algoritmus, valamint duális kontúrozás. A prototípusban a masírozó kockák algoritmusát alkalmaztuk, amelynek hátrány az élek automatikus letörése. A szimuláció lépései a következők: 1.) szerszám, munkadarab definíció implicit felületekkel; 2.) szerszámhelyzetek előállítás; 2.) a (3) egyenlet kiszámítása az egyes szerszámhelyzetekre OpenCL segítségével, párhuzamosan;

1. ábra. Gömbvégű szerszám pályaszimulációja ~1600 vonalszakasszal 2500ms számítási idővel (azaz: 1.5 ms szerszámhelyzetenként)

A 1. ábrán a marási folyamat vizualizálását látjuk. Összehasonlítva a GPU alapú megoldással körülbelül 20 szoros sebességnövekedést tudunk elérni. Szimuláció környezet mindössze egy laptop AMD Radeon HD 5700-as videokártyával, ami 4 GPU maggal rendelkezik. Az implementáció Aparapi OpenCL technológiát alkalmaz [4]. A forráskód letölthető: <http://users.iit.uni-miskolc.hu/~simon11/conferences/FMTU2012/millingsimulator.zip>

Összefoglalás

Egyszerű 3 tengelyes marás szimulátor prototípus fejlesztettünk ki OpenCL párhuzamos technológia alkalmazásával, amely összehasonlítva a hagyományos CPU alapú prototípus eljárással, körülbelül 20 szoros sebességnövekedést eredményezett olcsó hardveren is. Jelen 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.

Irodalom

- [1] NVIDIA Corporation: *NVIDIA CUDA compute unified device architecture programming guide*, Verzió 4.0, 5/6/2011, <http://developer.nvidia.com/cuda>, [utoljára megtekintve 2012. January 6.]
- [2] <http://www.nvidia.com/object/personal-supercomputing.html>, [utoljára megtekintve 2012. február 2.]
- [3] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J.: *GPU Computing*, Proceedings of the IEEE 96(5), 2008, 879–899 oldal.
- [4] Aparapi, API for data parallel Java. Allows suitable code to be executed on GPU via OpenCL, <http://code.google.com/p/aparapi/>, 2012
- [5] H. Müller, F. Albarsmann, A. Zabel, Efficient raster-based simulation and visualization of 3-axis Milling of Free-form Shapes. Research report No. 667/1998, Fachbereich Informatik Universität Dortmund, Germany, 1998

Simon Pál, doktorandusz

Miskolci Egyetem, Hatvany József Informatikai Tudományok Doktori Iskola

E-mail: simon11@iit.uni-miskolc.hu

Nehéz Károly, egyetemi docens, PhD

Miskolci Egyetem, Alkalmazott Informatika Tanszék

3515, Miskolc Egyetemváros, Hungary

Telefon / Fax: +36-46-565111/1954

E-mail: nehez@ait.iit.uni-miskolc.hu