**TémaLabor**

**GPGPU- általános számítások a GPUn, folyadékszimuláció**

Horváth Ákos – DKILK6

Gurubi Barnabás - DXEXVR

Konzulens: **Tóth Balázs**

Mit is jelent az a GPGPU?

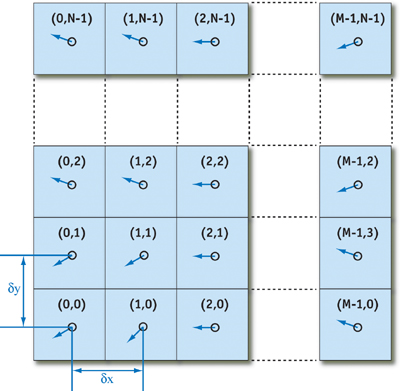
Alapvetően, ha programozásról beszélünk mindenkinek elsőre az egyszálú (esetleg pár szálú) CPU-n futó programok jutnak az eszébe. Ez miért is jelenthet problémát? Vannak olyan feladatok, amelyek nagymértékben párhuzamosíthatóak, és ha ezt észre vesszük és ki is használjuk sokkal gyorsabb programokat érhetünk el. Mint tudjuk a CPU csak korlátozottan képes a többszálúságra így az olyan problémák esetén, ahol egyszerű számításokat tudunk nagy számban párhuzamosan végezni érdemes a GPU-hoz fordulni. Ilyenkor beszélünk GPGPU alkalmazásról. (GPGPU = General-purpose computing on graphics processing units, azaz általános célú számítások a grafikus processzoron)

Az egyik ilyen GPGPU alkalmazási terület a fizikai szimulációk. Aki ismeretes az áramlástanban hallhatott már a Navier-Stokes egyenletekről. Mi a témalaborunk keretében ezek megértésére és alkalmazására fókuszáltunk.

Ahhoz persze hogy a GPU-n ilyen dolgokat végezhessünk, természetesen szükség van a megfelelő eszközökre, gondolok itt mind a megfelelő videokártyákra, ill. a megfelelő driverekre, fejlesztő környezetekre. Alapvetően két különböző környezetet használhatunk: OpenCL, CUDA. Az OpenCL nevének megfelelően is elméletileg mindenféle gyártó kártyáján működik, míg ezzel ellentétben a CUDA csak Nvidia és azon belül csak megfelelő Nvidia kártyákkal működik. Mivel mi rendelkeztünk Nvidia kártyával így az utóbbi környezetet választottuk a fejlesztésre. (Munkák során kétféle kártyát használtunk: 1 egyszerűbb mobil GPU-t: Nvidia GeForce GT 635m és egy jóval erősebb Nvidia GeForce GTX 1070 kártyát)

Fizikai alapok, avagy a Navier-Stokes egyenletek:

A folyadék állapotának leírására jó módszer, ha a folyadékot egyfajta „négyzet-rácsban” képzeljük el, amelyben az adott pozícióban fellépő sebességet tároljuk (sebesség-vektormező)

**x** = (x, y) pozíció

**u** = (u,v) sebesség

T idő

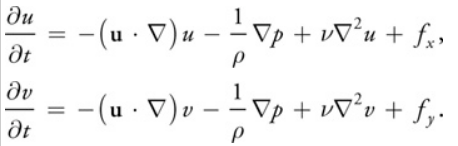
**u**(x, t) = (u(x, t), v(x, t))

A Navier-Stokes egyenleteket Claude Navier és George Gabriel Stokes állították fel 1822-ben a folyékony anyagok mozgásának, áramlásának leírására. Alapvető elképzelésük az volt, hogy az anyagban fellépő feszültségnek két összetevője van: a folyékony anyag sebességgradiensével arányos diffúziós (vagyis egy a viszkozitást jellemző) kifejezés összetevőből és egy nyomás összetevőből áll. Az egyenletek mind elméleti mind gyakorlati haszonnal bírnak hiszen segítségükkel leírható pl. az időjárás, óceánokban áramlatok, repülőgépek szárnyai körül észlelt áramlás, vagy például csillagok galaxisokon belül leírt mozgása is. A repülőgépek és gépjárművek tervezése mellett használhatóak atmoszferikus szennyezés felmérésére, sőt akár véráram szimulálására is.

Az egyenletek:



Ahol *ρ* a folyadék sűrűsége, ν a viszkozitás, és **F** = (fx , fy ) minden egyéb külső erőt ír le. Emellett vegyük észre, hogy az első egyenlet lényegében két egyenletet takar, hiszen az u az egy vektor mennyiség, amit külön két részre lehet bontani:



Az egyenlet tagjai:

**Advekció**



Gondoljunk bele, hogy a folyadék hogyan szállíthat minden féle objektumot, sűrűséget a folyammal együtt. Például, ha belecseppentünk a folyadékba egy kis tintát akkor az áramlatok ezt elszállítják. Ezt „előrehaladó”, „szállító” mozgást írja le az advekciós rész.

**Nyomás**



Mivel a folyadék molekulái tudnak áramolni mindenfele, egymással ütköznek is. Ha erőt fejtünk ki a folyadékban egy pontra akkor ez az erő nem fog rögtön végig terjedni a folyadékban. Az erőhöz közel lévő molekulák meglökik a távolabb lévőket és így egy nyomás alakul ki. Mivel a nyomás az egységnyi területre eső erő nagyságát írja le így könnyedén észre vehetjük hogy ez gyorsuláshoz vezet. (Newton első törvénye F = m\*a). Ez az egyenleti rész ezt írja le.

**Diffuzió**



Tudjuk tapasztalatból, hogy a vannak sűrűbb és folyékonyabb anyagok. Azt mondjuk, hogy a sűrű anyagoknak nagy a viszkozitásuk. A viszkozitás egy mérőszám, hogy a folyadék mennyire áll ellen az áramlásoknak. Ez az ellenállás sebességdiffúziót okoz.

**Külső erők**



Mindenféle egyéb külső erőt ír le, amelyek lehetnek lokálisak vagy globálisak (pl. gravitáció)

Látható, hogy a U2207.GIF szimbólum három különböző alkalmazását is használtuk. Ez az ún. nabla operátor. A három különböző dolog, amire használtuk: a gradiens, a divergencia, és a Laplace-operátor.

|  |  |  |
| --- | --- | --- |
|  | Definíció | Véges differencia alak |
| Gradiens: |  |  |
| Divergencia: |  |  |
| Laplace: |  |  |

**Gradiens**: parciális deriváltakból álló vektor

**Divergencia**: a vektormező forrásának (vagy elnyelésének) nagysága egy adott pontban. azaz a sebesség változása a folyadék egy adott pontját határoló felületen.

A második egyenlet (a folytonosság egyenlete: ) biztosítja, hogy a folyadék összenyomhatatlan

Laplace: A divergencia alkalmazása egy vektormező gradiensére. Diffúziós egyenleteknél gyakran megtalálható. U2207.GIF · U2207.GIF = U2207.GIF2. És ezzel meg is kaptuk a harmadik egyenletet.

Az egyenletek megoldása analitikus módon csak ritkán egyszerű esetekben történik. Általában és a mi esetünkben is egy numerikus, inkrementális megoldást alkalmazunk. (főleg hogy mi szeretnék RealTime megjeleníteni is, aminek eredményeképp kevés idő van a számításokra) Az egyenletek megoldása nem volt feladatunk a jelenlegi témalabor keretein belül így megkaptunk egy OpenCL környezetben megírt implementációját a problémának.

Alap implementáció OpenCL ben

A feladat megoldásának kezdetekor rendelkeztünk egy folyadékszimulációs program OpenCL implementációjával. Ez a program a fentebb ismertetett fizikai alapra épül, az ott leírt részleteket implementálja C++ nyelven. Megjelenítés az OpenGL programkönyvtár segítségével történik. Az első lényeges feladatunk tehát - természetesen a fizikai alap megértése után – a megadott implementáció feldolgozása, megértése majd átírása CUDA-ba.

Konvertálás OpenCL-ből Cuda-ba.

A két említett GPU programozási környezet (programkönyvtár) hasonló alapon nyugszik, hiszen a grafikus kártyák alap architektúrája gyártók között is meglehetősen hasonló, így nyilván az alapelvek megegyeznek a két esetben. Továbbá a célfeladat mellett (nagyfokú párhuzamosíthatóság), mind a CUDA, mind az OpenCL C nyelven van implementálva.

Az előzőekből kifolyólag, ha az alap elveket, lépéseket megértjük (OpenCL esetén), akkor ezek nagymértékben alkalmazhatóak a továbbiakban is (CUDA). Ezek után igazából a feladat csak a konkrét „nyelvi” különbségek átírása, vagyis megfeleltetni egymásnak a változótípusokat, függvényeket, lépéseket, stb.

A program alap felépítése

A program két jól elkülöníthető részből áll, ezek a következőek:

* Kernel kód
* CPU (host) kód

Az előbbi felelős a szimuláció leírásáért és annak GPU-n való futtatásáért, míg az utóbbi tekinthető a program hagyományos értelemben vett részének.

A **kernel** **kód** tartalmazza a GPU-n párhuzamosan futtatható, úgynevezett kernel vagy device függvényeket, ezek feladata a szimulációs lépések konkrét elvégzése, hiszen ezek nagyfokúan párhuzamosíthatóak. Nagyvonalakban úgy képzelhető el, hogy minden fent leírt lépést az említett négyzetrács minden elemére párhuzamosan elvégezhetünk, ezáltal minden négyzetrácsnak „saját” kernel függvény példányai lehetnek és ezek futnak egymással egyidőben a videókártyán. A kernel kód másik fele ezeknek a kernel függvények használatának előkészítése: az adatnak hely foglalása a GPU memóriájában, szimulációhoz szükséges változók karbantartása, ha kell GPU-ra (vagy onnan vissza) másolása, valamint természetesen a kernel függvények kezelése, használata.

A **CPU másnéven host kód** feladata a program vezérlése, valamint itt található a megjelenítésért felelős rész is, ami OpenGL alapú. A két rész kommunikációja egyszerű, a kernel kód részben lévő függvények szabadon hívhatóak a cpu kódból, **kivéve** a kernel függvények. Tehát szükség van csomagoló (emelett vezérlő) függvényekre a kernel kódban, irányítják a kernel függvényeket, ezáltal a szimulációt is. Ezek már közvetlenül hívhatók a host kódból.

A program részei: **kernel.cu** (kernel kód), **kernel.h** (kernel kód header-je), **main.cpp** (cpu kód)

A két programkönyvtárnak természetesen megvannak a maga előnyei és hátrányai. Az OpenCL egyértelmű előnye a már említett multi-platform-osság, azonban ebből kifolyólag, hogy a CUDA-nak kevesebb eszközön kell működnie, így a hatékonyság magasabb lehet, valamint a támogatottság is (segéd programok, Nvidia által nyújtott megoldások, leírások). Személyes preferencia, de mi a CUDA-ban írt kódot letisztultabbnak és könnyebben érthetőnek is találtuk.

Optimalizálás/ monitorozás

A CUDA mellett a keretrendszer által nyújtott eszközök segítségével a kódot monitorozni és optimalizálni tudtuk. Ennek a lényege a fentebb leírt „kernel függvény példányok” optimális futtatási egység (csoport) számának és módjának a megtalálása.

Új funkciók

**Testek, határok**

Az alap szimuláció egy üres négyzet alakú 2D tartályban lévő folyadékra irányult melybe festék cseppenthető. Ezen felül mi tetszőleges új objektum tartályba helyezését valósítottuk meg. A belehelyezés egy kép megadásával történik, amin fehérnek kell lennie azon részeknek, ahol a folyadék szabadon áramolhat. A megoldás alapja, hogy az adott képből előállítunk egy „boundaryBuffer-t”, ami a négyzetrács méretének megfelelő tömbben tárolja a határokat 0-1 formában (ahol 1-es található, az a négyzetrács elem falnak minősül). Ezen határok alapján számolja a szimuláció a folyadék áramlását a megfelelő módon.

**Állandóan mozgó közeg előállítása**

A célunk az volt, hogy egy állandóan mozgó közeget hozzunk létre, amivel így jól szimulálható egy folyó áramlása vagy akár egy szélcsatorna is. Ehhez az első próbálkozásként minden négyzetrácson hozzáadtunk a sebességvektorhoz egy apró értéket, hogy ezzel biztosítsuk az közeg állandó mozgását. Azonban látható az ábrán is hogy nem a kívánt eredményt értünk el, hiszen így hogy minden mezőhöz hozzáadunk a falakba lényegében „belenyomjuk” a folyadékot és így alkalmatlan a különböző testek áramvonalasságának elemzésére:



Ezt követően próbáltuk ki azt a megoldást, hogy csak a képernyő szélén lévő pár oszlophoz adunk hozzá erőt és majd ez az egész közeget mozgásba fogja hozni.



Ez a megoldás sokkal jobbnak minősült a valóság szimulálására.

Összefoglalás, jövőbeli tervek

Összegzésben egy olyan GPU alapú programot valósítottunk meg, mely képes folyadékok 2D térben való áramlásának szimulálására, tetszőleges 2D objektumok körül. A program CUDA alapú.

A program többféleképp továbbfejleszthető. Egy kézenfekvő irány, hogy a szimuláció térben és ne síkban működjön. Másik irány lehet, hogy ne csak folyadékokat, hanem gázokat is szimulálni tudjunk, hiszen a fizikai alap hasonló.