Osztott rendszerek specifikációja és implementációja 3. beadandó - Dokumentáció

Név: Soós Bálint
Neptun kód: HDX9MU
Elérhetőség: soosbalint95@gmail.com

Gyakorlatvezető: Horpácsi Dániel
Csoport: 5.
2016. december 2.

Háttértörténet

Egyre közelebb a karácsony, ami a hallgatók számára a vizsgaidőszakot jelenti. A diákok a projektmunkák végére értek, a leadási határidő pedig vészjóslóan közel került hozzájuk. Az ELTEngine nevű játék-motrot használták a fejlesztésekre, ebben valósították meg az év elején kitűzött célokat. A végső tesztek során azonban hibásnak érezték a program működését, a karakteranimációk megvalósításáért felelős modul műveletigénye hatalmas volt, így használhatatlanná vált éles projektben való alkalmazásra. Annak érdekében, hogy helyesen működjön a szoftver, a fejlesztőkhöz fordultak, akik - ismerve a B szakirányos hallgatók példaértékű munkáját - ismét az ELTE-s diákokat kérték meg a probléma elhárítására. A rosszul működő komponens forráskódját végignézve rájöttek, hogy a hiba oka egy figyelmetlen fejlesztő barkácsolásának köszönhető, aki nem ismerte a pipeline fogalmát, így négy, egymásba ágyazott ciklusba szervezve futott a kódrészlet, ami a karaktereket reprezentáló vertexek sorozatán elvégezte a szükséges mátrix-transzformációkat. Szerencsére a B-s fejlesztők szorgalmasan jártak ORSI-előadásra és gyakorlatra, így magától értetődő volt, hogy adatcsatorna tételére visszavezetve máris N+M-es futásidejű kódot kaphatnak, ami már megfelelt a megrendelők igényének. A patch kiadása és az ELTEngine frissítése után a hallgatók megelégedve készülhettek a vizsgaidőszakra, hiszen az év során felmerülő összes akadályt sikerült leküzdeniük a kemény munkával.

A bemeneti fájlok egyike, az `input_matrices.txt` első sorában egy M pozitív egész olvasható, ennyi lineáris transzformációt kell elvégezni az objektumokat reprezentáló vektorokon, míg a következő 4xM sorban egy-egy 4x4es mátrix sorfolytonos reprezentációja található szóközökkel elválasztva. (4 egymás utáni sor jelent egy mátrixot.) A másik fájl, az `input_points.txt` tartalmazza a pontok listáját, amikre alkalmazni kell az előbbi mátrixokkal való szorzást. Ennek első sora egy egész szám, N, ez után található N sor egy-egy 3 dimenziós vektor koordinátáit írja le (szintén szóközzel tagolva).

A számítást a későbbiekben megkönnyíti, ha a bemenet során a 3D-s vektorokat 4 dimenzióban reprezentáljátok, a negyedik koordinátát 1-re inicializálva. A mátrixszal való szorzás így lényegesen könnyebbé válik.

Egy lehetséges `input_matrices.txt` fájl: (Az alábbi mátrix-transzformációk pl. a 2x-es méretezés, 90°-os forgatást az X tengely körül, majd a (3;-2;4) vektorral való eltolást jelentik.)

`input_matrices.txt`

```
1
2 0 0 0
0 2 0 0
0 0 2 0
0 0 0 1
```

`input_points.txt`

```
2
1 1 1
1 1 -1
```

Feladat

A feladatban az adatcsatorna tételére visszavezetve kell megoldani a kitűzött problémát!

A főfolyamat dolga, hogy beolvassa a mátrixokat, majd M threadet létrehozva, s azokat egy-egy transzformációnak megfeleletetve átadja nekik az megfelelő mátrixokat. A pontokat az első mátrixot reprezentáló szálnak kell elküldeni, majd az utolsó leképezést megvalósító gyerektől fogadja a megfelelően transzformált vektorokat. Ezek után írja soronként az output.txt fájlba az így kapott eredményt.

Az indított szálak feladata, hogy fogadják a vektorokat a megelőző gyerektől (az első transzformáció esetén a mastertől), elvégezzék a mátrix-vektor szorzást, majd továbbítsák az így kapott részeredményt a következő folyamatnak. (Az utolsó számítás esetén a masternek.) A teljes memóriahasználat csökkentése érdekében használhattok külön szálat a beolvasás és kiírás elvégzésére.

A dokumentációban mindenképp szerepeljen a visszavezetés mikéntje, azaz a megfeleltetés az absztrakt programhoz! A fejlesztői fejezetben szeretnénk mérésekkel alátámasztva látni, hogyan is skálázódott a program a különböző méretű bemenetek esetén!

Felhasználói dokumentáció

1. ***Környezet***

   A program fordítástól függően `.out` kiterjesztés esetén Linux/OSX oprendszereken, `.exe` kiterjesztés esetén Windows oprendszereken használható. Telepítésre nincs szükség, elegendő a futtatható állományt elhelyezni a számítógépen.

2. ***Használat***

   A program elindításához nincs szükség parancssori paraméterekre, így parancssoron kívül is lehet futtatni. A programmal egy mappaszinten kell elhelyezni a bemeneti fájlokat. A program eredményét ugyanezen a szinten az `output.txt` kimeneti fájlba írja.

   Egy lehetséges bemenetet tartalmaznak a mellékelt fájlok, illetve a feladathoz csatolt `tests` mappában további példák találhatóak. Saját bemeneti fájlok esetén fontos, hogy a feladatban megadott szempontok alapján írjuk az adatokat a fájlba, mivel ezek helyességét a programban nem ellenőrizzük.

Fejlesztői dokumentáció

1. ***Megoldás módja***

   A programot logikailag két részre, főfolyamatra és az adatcsatornákkal összekötött alfolyamatokra bonthatjuk. A főfolyamat végzi a bemeneti adatok beolvasását, amiket egy `vector<array<int, 4>` adatszerkezetben tárol. Létrehozza az adatcsatornákat és az alfolyamatokat, amelyeknek átad egy mátrixot, illetve két csatornát. Az egyiken keresztül kapja transzformálandó vektorokat, a másikon pedig továbbküldi az eredményeket. A végeredményt szintén a főfolyamat írja a kimeneti fájlba.

2. ***Visszavezetés az adatcsatorna tételére***

   TODO

3. ***Implementáció***

   Az adatcsatorna implementációja a `pipeline.hpp`, a főprogramé pedig a `main.cpp` fájlban található.

4. ***Fordítás***

   A program forráskódja a `main.cpp` fájlban található. Fordításához egy C++11 szabványt támogató fordítóprogram szükséges. A fejlesztéshez a `g++` fordítót használtam: `g++ -std=c++11 main.cpp`

5. ***Tesztelés***

   Teszteléshez a feladathoz csatolt `tests` mappában található bemeneti fájlokat használtam, amelyek mindegyikére az elvárt kimenetet állítja elő a program.

6. ***Mérések***