

UNIwersYTET JAGIELLOŃSKI
WYDZIAŁ MATEMATYKI I INFORMATYKI
ZESPÓŁ KATEDR I ZAKŁADÓW INFORMATYKI MATEMATYCZNEJ

Wielowątkowa symulacja N ciał z implementacją w architekturze CUDA

Autor

Damian Stachura

Opiekun

dr Maciej Ślusarek

August 13, 2018

Contents

1	Przedstawienie problemu symulacji N ciał	3
1.1	Szczególne przypadki	3
1.1.1	Problem dwóch ciał	3
1.1.2	Problem trzech ciał	3
1.2	Zastosowania	3
1.3	Implementacja i wykorzystane technologie	4
1.3.1	Architektura CUDA	4
1.3.2	Thrust	4
1.3.3	OpenGL	4
2	Pierwsze podejście implementacyjne	5
2.1	Sformułowanie problemu	5
2.2	Jednowątkowy naiwny algorytm z pseudokodem	6
2.3	Paralelizacja naiwnego algorytmu	7
2.4	Implementacja	7
3	Drugie podejście	7
3.1	Algorytm Barnes-Huta z pseudokodem	7
3.2	Zrównoleglenie algorytmu Barnes-Huta	7
3.3	Implementacja	8
4	Wizualizacja	8
5	Podsumowanie	9

1 Przedstawienie problemu symulacji N ciał

Symulacja N ciał jest zagadnieniem z mechaniki klasycznej, które polega na wyznaczeniu toru ruchów wszystkich ciał danego układu o danych masach, prędkościach i położeniach początkowych w oparciu o prawa ruchu i założenie, że ciała oddziałują ze sobą zgodnie z prawem grawitacji Newtona.

1.1 Szczególne przypadki

Problem wyznaczenia dokładnego ruchu dowolnej liczby ciał jest trudny, więc można znaleźć wiele prac skupiających się jedynie na ustalonej, małej liczbie ciał.

1.1.1 Problem dwóch ciał

Problem dla dwóch ciał podlegających prawom klasycznej dynamiki Newtona i przyciągających się zgodnie z newtonowskim prawem powszechnego ciążenia został rozstrzygnięty przez J. Bernoulliego przy założeniu, że masa obiektu koncentruje się w jego środku. [Rog71] Obydwa ciała poruszają się po krzywych stożkowych, których rodzaj zależy od całkowitej energii układu. Przykładowo, gdy energia jest mała, to ciała nie mogą się od siebie uwolnić, więc krążą wokół siebie po elipsach.

1.1.2 Problem trzech ciał

Problem trzech ciał wciąż nie został rozwiązany w ogólności. Jednakże istnieją rozwiązania dla szczególnych przypadków, jak na przykład [Ala00; GLo03]. Inną wariacją tego problemu jest system, w którym masa jednego z ciał jest zaniedbywalnie mała, jest to tak zwany ograniczony problem trzech ciał - przedstawiony przez J. L. Lagrange'a w XVIII wieku. Badał on układ Słońce-Ziemia-Księżyc.

1.2 Zastosowania

Symulacje N ciał są szeroko wykorzystywanymi narzędziami w fizyce oraz astronomii. Problemem, w którym symulacje są użyteczne jest na przykład dynamika systemu z kilkoma ciałami jak układ Słońce-Ziemia-Księżyc [JEi], co może pomóc w zrozumieniu działania olbrzymich systemów we wszechświecie. [Heg91] W kosmologii symulacje są wykorzystywane do studiowania procesów tworzenia nieliniowych struktur jak galaktyczne halo z wpływem ciemnej materii [EDO02]. Z kolei, bezpośrednie symulacje N ciał są wykorzystywane na przykład do studiowania dynamicznej ewolucji klastrow gwiazd lub do symulacji dynamiki planetozymali. [DL98]. Symulacje są wykorzystywane, również w innych dziedzinach, chociażby w algorytmach rysowania grafu skierowanego siłą [DEg15]

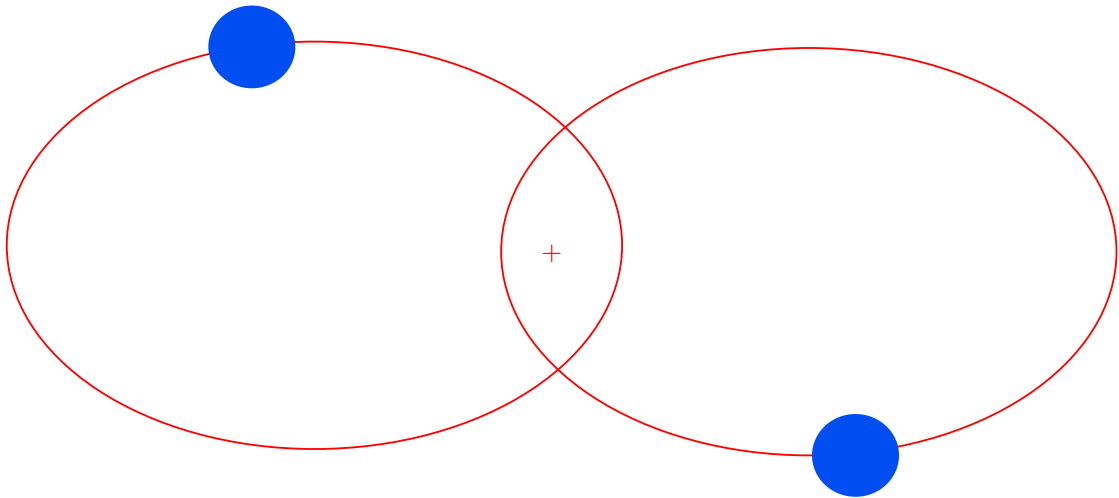


Figure 1: Symulacja dwóch ciał poruszających się po elipsach

1.3 Implementacja i wykorzystane technologie

W pierwszej części mojej pracy przedstawię równoległą implementację naiwnego algorytmu symulacji N ciał. W każdym kroku algorytm bezpośrednio wyznacza siły oddziałujące wzajemnie pomiędzy każdymi dwoma ciałami w systemie, czyli wyznacza siłę jaką pozostałe obiekty działają na wybrany. W drugiej części zaimplementuję wielowątkowo algorytm Barnes Hut'a, który korzysta z drzew ósemkowych.

Repozytorium jest dostępne pod tym [linkiem](#). Całość została zaimplementowana w C++. Poniżej wymienione są pozostałe technologie wykorzystane w projekcie.

Pełna instalacja niezbędnego oprogramowania do uruchomienia symulacji jest zawarta w repozytorium (dla linuxa Ubuntu).

1.3.1 Architektura CUDA

CUDA to uniwersalna architektura procesorów wielordzeniowych (głównie kart graficznych) umożliwiająca zaimplementowanie ich mocy obliczeniowej w wielu problemach, które mogą się wykonywać zarówno sekwencyjnie i wielowątkowo. Wykorzystałem CUDE w wersji v9.1.85 do paralelizacji dwóch wspomnianych powyżej algorytmów.

1.3.2 Thrust

Thrust jest szablonową biblioteką dla CUDA bazująca na bibliotece STL z C++. Thrust umożliwia implementację aplikacji wielowątkowych za pośrednictwem interfejsu wysokiego poziomu, który jest w pełni zgodny z CUDA C. Korzystałem z wersji v9.2.88.

1.3.3 OpenGL

OpenGL jest API do tworzenia grafiki. Skorzystałem z OpenGL3, w celu zwizualizowania symulacji w 3D.

2 Pierwsze podejście implementacyjne

2.1 Sformułowanie problemu

W celu przedstawienia ogólnego sformułowania problemu potrzebujemy przytoczyć trzy prawa dynamiki sformułowane przez Isaaca Newtona [Rog71]

Prawo 1. *Każde ciało pozostaje w stanie spoczynku lub ruchu jednostajnego w linii prostej, chyba że jest zmuszone zmienić ten stan przez zewnętrzne oddziaływanie z innymi ciałami, czyli każde ciało jest w układzie inercyjnym.*

Prawo 2. *Szybkość zmiany pędu jest proporcjonalna do siły wywieranej i znajduje się w tym samym kierunku co siła.*

Co oznacza, że w inercyjnym układzie odniesienia zachodzi równość $F = ma$, gdzie F jest wektorem sum sił działających na obiekt, m to masa obiektu, a to jego przyspieszenie.

Prawo 3. *Każdej akcji towarzyszy reakcja równa co do wartości i kierunku, lecz przeciwnie zwrotna.*

Co oznacza, że jeśli ciało A działa na ciało B siłą F (akcja), to ciało B działa na ciało A siłą (reakcja) o takiej samej wartości i kierunku, lecz o przeciwnym zwrocie.

Niezbędne jest również przytoczenie prawa powszechnego ciążenia Newtona [Rog71]

Prawo 4. *Każdy obiekt przyciąga każdy inny obiekt z siłą, która jest wprost proporcjonalna do iloczynu ich mas i odwrotnie proporcjonalna do kwadratu odległości między ich środkami.*

Czyli między dowolną parą ciał posiadających masy pojawia się siła przyciągająca, która działa na linii łączącej ich środki, a jej wartość rośnie z iloczynem ich mas i maleje z kwadratem odległości.

Aplikując to prawo do symulacji N ciał, uzyskujemy że na każde i -te ciało działa siła F_i zdefiniowana następująco:

$$F_i = -G \cdot m_i \sum_{j=1, j \neq i}^N \frac{m_j(r_i - r_j)}{|r_i - r_j|^3},$$

gdzie G to stała grawitacji, m_i masa ciała na które oddziałują inne ciała, m_j masa ciała oddziałującego na i -te ciało, $r_i - r_j$ to różnica wektorów pozycji dwóch ciał, $|r_i - r_j|$ to dystans między ciałami.

Z wykorzystaniem powyższych praw możemy podać następującą definicję

Symulacja N ciał — Dla N ciał mających ustalone masy oraz początkowe położenie i prędkość, ruch każdego obiektu jest symulowany z wykorzystaniem prawa powszechnego ciążenia oraz poprzez wyznaczenie przyspieszenia obiektu korzystając z drugiego prawa dynamiki Newtona.

2.2 Jednowątkowy naiwny algorytm z pseudokodem

Najprostszy algorytm dla problemu N ciał zadany pseudokodem może wyglądać tak:

Listing 1: pseudokod

```
1 ustaw mase oraz początkową pozycję i prędkość dla każdego ciała
2 while (true)
3     for i in {1...N}:
4         uaktualnijPozycje()
5         narysujNowePozycje()
```

Każde ciało na początku symulacji ma losowaną pozycję, prędkość oraz wagę. Jednostki przyjęte w symulacji są następujące :

- jednostką wagi jest masa słońca. Masa słońca jest definiowana następująco

$$M_{\odot} = 1.9884 \cdot 10^{30}$$

Symulowane gwiazdy mają wagi z zakresu $[0.5, 10] M_{\odot}$.

- jednostką odległości jest parsek, czyli odległość, dla której paralaksa roczna położenia Ziemi widzianej prostopadle do płaszczyzny orbity wynosi 1 sekundę łuku. W przeliczeniu na metry i po zaokrągleniu jest to

$$1 \text{ pc} \approx 3,2616 \text{ roku świetlnego} \approx 3,086 \cdot 10^{16} \text{ m}$$

Najbardziej złożoną operacją w algorytmie jest uaktualnienie pozycji w każdym obiegu pętli nieskończonej dla każdego ciała w symulacji. W tym celu wykorzystane zostało, wcześniej wprowadzone, Prawo 4. W ten sposób może zostać policzona siła, którą na dane ciało działają wszystkie pozostałe obiekty w symulacji.

Przypominając, wzór na siłę działającą na ciało wygląda następująco :

$$F_i = -G \cdot m_i \sum_{j=1, j \neq i}^N \frac{m_j (r_i - r_j)}{|r_i - r_j|^3},$$

gdzie stała grawitacji wynosi

$$G = 6,67408(31) \cdot 10^{-11} \frac{m^3}{kg s^2},$$

W astronomii stała grawitacji jest wyrażana jako

$$G = 4,3 \cdot 10^{-3} \frac{pc}{M_{\odot}} \frac{km^2}{s^2}$$

Złożoność obliczeniowa naiwnej operacji uaktualnienia pozycji wszystkich obiektów, przedstawia poniższy fakt.

Prawo 5. *Uaktualnienie pozycji wszystkich ciał ma złożoność obliczeniową $\mathcal{O}(N^2)$.*

Dowód: Dla każdego ciała najpierw musimy wyznaczyć siłę działającą na nie poprzez interakcję z innymi ciałami, czyli dla każdego z N ciał musimy policzyć siłę oddziałującą nań z każdym innym obiektem, więc musimy policzyć wartość wzoru wynikającego z prawa 4 $N \cdot N - 1$ razy, czyli złożoność tej podoperacji $\mathcal{O}(N^2)$. Następnie dla każdego obiektu musimy wyznaczyć jego przyspieszenie oraz nową pozycję i prędkość, co jesteśmy w stanie zrobić w czasie $\mathcal{O}(N)$. Poprzez zsumowanie złożoności obu podoperacji, widzimy że złożoność obliczeniowa jednego kroku symulacji naiwnego algorytmu wynosi $\mathcal{O}(N^2)$. \square

Listing 2: uaktualnienie pozycji ciał

```

1 typedef thrust::host_vector<float> tf3;
2 void StepNaive::compute(tf3& positions, float dt) {
3     std::fill(forces.begin(), forces.end(), 0);
4     for(unsigned i=0; i<N; i++) {
5         for(unsigned j=0; j<N; j++) {
6             float distX = positions[j*3] - positions[i*3];
7             float distY = positions[j*3+1] - positions[i*3+1];
8             if(i!=j && fabs(distX) > 1e-10 && fabs(distY) > 1e-10) {
9                 float F = G*(weights[i]*weights[j]);
10                forces[i*3] += F*distX/(distX*distX+distY*distY);
11                forces[i*3+1] += F*distY/(distX*distX+distY*distY);
12            }
13        }
14    }
15    for(unsigned i=0; i<N; i++) {
16        for(int j=0; j<2; j++) { // x, y
17            float acceleration = forces[i*3+j]/weights[i];
18            positions[i*3+j] += velocities[i*3+j]*dt + acceleration*dt*dt/2;
19            velocities[i*3+j] += acceleration*dt;
20        }
21    }
22 }
```

2.3 Paralelizacja naiwnego algorytmu

2.4 Implementacja

3 Drugie podejście

3.1 Algorytm Barnes-Huta z pseudokodem

3.2 Zrównoleglenie algorytmu Barnes-Huta

sudo apt-get install texlive-full

3.3 Implementacja

http : //www.deltami.edu.pl/temat/fizyka/mechanika/2015/11/26/Problem_dwochcial/ apt-

get install texlive-lang-polish

Random citation embeddeed in text.s

sudo apt-get install texlive-bibtex-extra

sudo apt-get install texlive-bibtex-extra biber

biber Praca

https://www.sharelatex.com/learn/Bibliography_management_with_biblatex

inkscape -D -z -file=drawing.svg -export-pdf=draw.pdf -export-latex

4 Wizualizacja

5 Podsumowanie

References

- [Aar03] Sverre J. Aarseth. *Gravitational N-Body Simulations. Tools and Algorithms 1 edition*. Cambridge University Press, 2003.
- [Ala00] Richard Montgomery Alain Chenciner. “A remarkable periodic solution of the three-body problem in the case of equal masses”. In: *Annals of Mathematics* 152 (2000), pp. 881–901.
- [Cora] NVIDIA Corporation. *CUDA C Programming Guide*. v9.1.85, 2018. URL: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html> (visited on 03/05/2018).
- [Corb] NVIDIA Corporation. *NVIDIA CUDA Runtime API*. v9.1.85, 2018. URL: <http://docs.nvidia.com/cuda/cuda-runtime-api/index.html> (visited on 03/05/2018).
- [DEg15] A.Bezgodov D.Egorov. *Improved Force-Directed Method of Graph Layout Generation with Adaptive Step Length*. 2015.
- [DL98] J. Stadel D. Richardson T. Quinn and G. Lake. *Direct Large-Scale N-Body Simulations of Planetesimal Dynamics*. 1998.
- [EDO02] G.Chincarini E.D’Onghia C.Firmani. *The Halo Density Profiles with Non-Standard N-body Simulations*. 2002.
- [GLo03] M.Kramer G.Lodge J. A. Walsh. “A Trilinear Three-Body Problem”. In: *International Journal of Bifurcation and Chaos* 13 (2003), pp. 2141–2155.
- [GOO] GOOGLE? *THRUST*. v9.2.88, 2018. URL: <https://docs.nvidia.com/cuda/thrust/index.html> (visited on 05/15/2018).
- [Gro17] Khronos Group. *OpenGL API, OpenGL Shading Language and GLX Specifications*. OpenGL 4.6. 2017. URL: https://www.khronos.org/registry/OpenGL/index_gl.php (visited on 07/30/2017).
- [Heg91] Douglas C. Heggie. *CHAOS IN THE N-BODY PROBLEM OF STELLAR DYNAMICS*. 1991.
- [JEi] J.Eiland. *N-Body Simulation of the Formation of the Earth-Moon System from a Single Giant Impact*.
- [Lar07] Jan Prins Lars Nyland Mark Harris. “GPU Gems 3”. In: 2007. Chap. Fast N-Body Simulation with CUDA. Chapter 31, pp. 677–694.
- [Lin99] Tancred Lindholm. “Seminar presentation. N-body algorithms”. In: *University of Helsinki* (1999).
- [Mar11] Keshav Pingali Martin Burtscher. “GPU Computing Gems Emerald Edition”. In: NVIDIA Corporation, Wen-mei W. Hwu, 2011. Chap. An Efficient CUDA Implementation of the Tree-Based Barnes HUT N-Body Algorithm. Chapter 6, pp. 75–92.
- [Rog71] Jerry E. White Roger R. Bate Donald D. Mueller. “Fundamentals of astrodynamics”. In: DOVER PUBLICATIONS, 1971. Chap. 1 TWO-BODY ORBITAL MECHANICS, pp. 1–49.