Symulacja ruchu N-ciał

Szymon Bugaj, Lei Peng

29 stycznia 2016

Streszczenie

Sprawozdanie z projektu z przedmiotu RIM. Tematem projektu jest symulacja ruchu N-ciał (N punktów materialnych pod działaniem siły grawitacji).

Spis treści

1 Wstęp teoretyczny

Projekt stanowi symulacja i wizualizacja praw fizyki klasycznej newtonowskiej. Szczególnie 3 praw dynamiki Newtona oraz siły grawitacji pomiędzy zbiorem N punktów materialnych.

Newtonowska siła grawitacji pomiędzy dwoma punktami materialnymi:

$$\vec{F_G} = G\frac{m_1 m_2}{\|r\|^2}\frac{\vec{r}}{\|r\|}$$

W fizyce klasycznej zależnością łączącą masę przyśpieszenie i siłę działającą na ciało jest:

$$\vec{F} = \vec{a}m$$

Związek między położeniem, prędkością a przyśpieszeniem jest następujący:

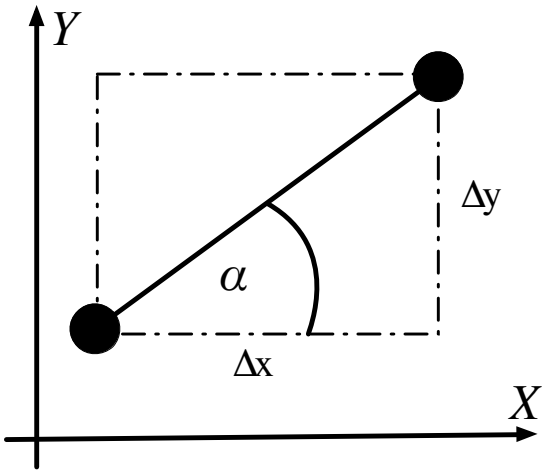
$$v(t) = \frac{dp(t)}{dt}$$
$$a(t) = \frac{dv(t)}{dt}$$

Mając dane położenia wszystkich N cząstek oraz ich prędkości dla danego momentu, korzystając z powyższych równań możemy obliczyć położenie dla dowolnego punktu w dowolnym innym momencie.

2 Rozkładanie siły grawitacji na składowe względem osi współrzędnych

W przypadku 2D należy rozpatrzyć prostokąt o bokach równoległych do osi X i Y, gdzie w przeciwległych wierzchołkach leżą dwa rozpatrywane ciała.

Rozkład wektora siły na składowe



$$a_x(t) = sin\alpha * a$$
$$a_y(t) = cos\alpha * a$$

Dla przypadku 3D należy rozpatrzyć w analogiczny sposób prostopadłościan (ciała powinny znajdować się w wierzchołkach po przeciwnych ścianach, po przeciwnych rogach).

3 Model numeryczny

Interesują nas położenia ciał w kolejnych dyskretnych punktach czasu. Przyjmujemy, że w przeciągu jednostki czasu wielkości prękość, przyśpieszenie są stałe. Aktualizujemy interesujące nas wielkości w kolejności:

- 1) aktualizacja prędkości
- 2) aktualizacja położenia
- 3) aktualizacja przyśpieszenia

Co należy podkreślić, wynika z tego, iż do aktualizacji prędkości brana jest poprzednia wartość przyśpieszenia.

Poniżej główna funkcja step, aktualizująca położenie, prędkość i przyśpieszenie wszystkich cząstek (NBodiesSystem.h/cpp).

```
1  void NBodiesSystem::step( time_type delta_t ) {
2      p_prev = p_curr;
3      v_prev = v_curr;
4
5      /*
6          UPDATE v SPEED and p POSITION
7      */
8
9      for (int d = 0; d < D; ++d)
10         for (int i = 0; i < N; ++i) {
11             v_curr.setVal(d, i,
12                 v_prev.getVal(d, i) +
13                 a.getVal(d, i) * delta_t);
14             p_curr.setVal(d, i,
15                 p_prev.getVal(d, i) +
16                     (v_prev.getVal(d, i) +
17                     v_curr.getVal(d, i))
18                         * 0.5 * delta_t);
19         }
20
21     /*
22         UPDATE a ACCELERATION
23         For each two bodies i,j where i != j;
24     */
25
26     for (int d = 0; d < D; ++d)
27         for (int i = 0; i < N; ++i)
28             a.setVal(d, i, 0.0f);
29
30     for (int i = 0; i < N; ++i) {
```

```
31          for (int j = 0; j < N; ++j) {
32              if ( i == j ) continue;
33
34              /*
35                  delta X, delta Y, delta Z
36              */
37              position_type* r_axis = new position_type[D];
38
39              position_type r_squared = 0;
40              for (int d = 0; d < D; ++d) {
41                  r_axis[d] = (p_curr.getVal(d, i) -
42                      p_curr.getVal(d, j));
43                  r_squared += r_axis[d] * r_axis[d];
44              }
45
46              position_type a_scalar =
47                  G * m.getVal(0, j) /
48                  pow(r_squared + efactor, 1.5);
49
50              for (int d = 0; d < D; ++d) {
51                  /*
52                      If both objects positions are
53                      the same there is division
54                      by zero; what to do then?
55                      I just set acceleration to 0;
56                  */
57                  if (r_axis[d]) {
58                      a.setVal(d, i,
59                          a.getVal(d,i) -
60                              a_scalar *
61                              (r_axis[d]/sqrt(r_squared)));
62                  }
63              }
64
65              delete [] r_axis;
66          }
67      }
68  }
```

4 Implementacja GPU

By uniknąć kopiowania danych pamięć na GPU jest współdzielona przez OpenGL i CUDA. Funkcje zadeklarowane w cuda_gl_interop.h pozwalają na to: cudaGraphicsMapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsUnmapResources (NBodiesSystemCUDA.cu).

Implementacja na GPU jest najprostszym przeniesiem kodu działającego na CPU.