Projekty zaliczeniowe

Michał Marciniak pokój. 4.30
mmarciniak26@uw.edu.pl

June 16, 2024

1 Problem N ciał

Napisz program symulujący oddziaływanie grawitacyjne N ciał (dla przykładu $N = 500$) o różnych masach i rozmiarach (zakładamy, że wszystkie obiekty są sferyczne). Wylosuj warunki początkowe (użyj funkcji z nagłówka <random>, za korzystanie z rand() będą ucinane punkty). Można użyć jednostek $G = 1$.

1. Rozwiąż równania ruchu. Dla dużych N metoda Runge-Kutty może okazać się bardzo wolna dlatego sugeruję użyć metody 'leapfrog':

$$x(t + dt) = x(t) + dt * v(t + dt/2) \tag{1}$$
$$v(t + dt/2) = v(t - dt/2) + dt * \ddot{x}(t) \tag{2}$$

   Aby zacząć obliczenia należy wyznaczyć $v(dt/2) = v(0) + \frac{dt}{2} * \ddot{x}(0)$

2. Uwzględnij całkowicie nieelastyczne zderzenia cząstek podczas których większe ciało "wchłania" mniejsze w taki sposób, że zwiększa swój rozmiar i masę o rozmiar i masę mniejszego ciała (zachowując sferyczny kształt). Załóż, że masa jest punktowa, a do zderzenia dochodzi, gdy odległość między środkami obiektów jest mniejsza niż suma promieni. Aby otrzymać nowy promień obiektu dodaj do siebie objętości i zakładając sferyczny kształt, oblicz jaki musi być nowy promień.

3. Przechowuj trajektorie pięciu dowolnych obiektów

4. Dla osób korzystających z ImPlota: ImPlot rysuje rozmiary na podstawie rozmiaru w pixel'ach, a nie rozmiarów rzeczywistych - zobacz projekt o gazie doskonałym Tomasza Fąsa. Projekt z gazem ma tą zaletę w rysowaniu, że cząstki są ograniczone do zadanego pojemnika. W przypadku oddziaływań grawitacyjnych nie mamy takich ograniczeń - układ jest otwarty. Żeby dobrze skalować rozmiar obiektów przy zoomowaniu można dzielić promień przez długość wyświetlanych osi. W tym celu można napisać funkcję GetAxesRange() (i dodać nagłówek!):

```
#include "implot_internal.h"

    float GetAxesRange(ImAxis ax){
        ImPlotContext& gp = *GImPlot;
        ImPlotPlot& plot = *gp.CurrentPlot;
        ImPlotAxis& axis = plot.Axes[ax];
        return fabs(axis.ScaleMin - axis.ScaleMax);
    }
```

   Żeby dostać rozpiętość osi x należy wywołać GetAxesRange(ImAxis_X1), dla rozpiętości osi y: GetAxesRange(ImAxis_Y1).

   Teraz aby narysować koło o promieniu r, ze środkiem w punkcie p_0 (typ ImVec2) wystarczy wykorzystać następujące funkcje (między ImPlot::Begin i ImPlot::End):

```
ImPlot::PushPlotClipRect(); //funkcja ta zapewnia poprawne renderowanie;
//bez niej wyświetlane poniżej obiekty będą renderowały się na przodzie i będą zakrywać osie

ImPlot::GetPlotDrawList()->AddCircleFilled(p_0, r/scale, IM_COL32(255, 0, 0, 255), number_of_segments);

//tutaj dodaj całą resztę obiektów do wyświetlenia

ImPlot::PopPlotClipRect(); //był push to musiy być pop
```

, gdzie scale jest dane przez GetAxesRange(ImAxis_X1), IM_COL32(255, 0, 0, 255) odpowiada za kolor wypełnienia, a number_of_segments określa z ilu krawędzi będzie składało się koło. Ostatnim problemem jest znalezienie p_0 - zmienna typu ImVec2 wymagana przez metodę AddCircleFilled jest pozycją w pixelach. Najprostszym rozwiązaniem jest wykorzystanie funkcji PlotToPixels:

```
ImVec2 p0 = ImPlot::PlotToPixels( punkt );
```

gdzie punkt jest zmienną typu ImPlotPoint przechowującą rzeczywistą pozycję obiektu. W ogólności polecam zastosowanie zmiennych typu ImPlotPoint do przechowywania pozycji i prędkości obiektów. Dostęp do składowych:

```
ImPlotPoint pt (1, 2);
cout<< pt.x <<" " << pt.y<<endll;
```

5. Rysuj aktualny stan układu, w tym trajektorie oraz wypisuj zmienną, która wyświetla aktualną liczbę obiektów.

1.1 Proponowane ocenianie

1. Poprawne napisanie klasy reprezentującej pojedyncze ciało niebieskie oraz klasy reprezentującej cały układ (15pkt)
2. Poprawne napisanie metody generującej układ (10pkt)
3. Poprawne napisanie metody rozwiązującej równania ruchu (10pkt)
4. Poprawne napisanie metody odpowiadającej zderzeniom całkowicie nieelastycznym (10pkt)
5. Wyrysowanie aktualnego stan układu (+trajektorie) (10 pkt)
6. Ogólna poprawność kodu (5pkt)