Programowanie II R

Zadania – seria 9.

Algorytmy STL

1 Cel ćwiczenia

Praktyczne zapoznanie się z biblioteką STL oraz wyrażeniami lambda w języku C++. W ramach zadania zaimplementowana zostanie symulacja gazu doskonałego (helu) w zamkniętym pojemniku. Wykorzystując narzędzia STL, wyznaczymy parametry makroskopowe układu.

2 Wymagania techniczne

Zadanie wymaga standardu **C++20**. Do kompilacji należy użyć flagi `--std=c++20`. Program powinien wykorzystywać biblioteki: `<iostream>`, `<vector>`, `<random>`, `<algorithm>`, `<numeric>` oraz `<cmath>`. W zadaniu nie wolno używać pętli.

3 Wstęp Fizyczny

W gazie doskonałym, każda ze składowych prędkości cząsteczek ma rozkład Gaussa o zerowej średniej. Z temperatury gazu T wynika odchylenie standardowe tego rozkładu:

$$\sigma = \sqrt{\frac{k_B T}{m}}$$

W programie należy przyjąć następujące stałe fizyczne:

- Stała Boltzmanna: $k_B = 1.38 \times 10^{-23}$ J/K
- Masa atomu helu: $m = 6.65 \times 10^{-27}$ kg
- Początkowa temperatura: $T = 300.0$ K

Moduł wektora prędkości opisuje rozkład Maxwella-Boltzmanna.

4 Zadania do wykonania

Etap 1: Inicjalizacja i generowanie danych (`std::generate`)

1. Zdefiniuj strukturę (`struct` - klasa o domyślnie publicznych polach) `Czasteczka` zawierającą jedynie składowe prędkości `vx, vy, vz`.
2. Utwórz `std::vector<Czasteczka>` o rozmiarze 100 000 elementów.
3. Skonfiguruj generator `std::mt19937` oraz rozkład normalny `std::normal_distribution` z parametrem σ obliczonym we wstępie.

4. Użyj algorytmu `std::generate` do wygenerowania prędkości cząsteczek. Jako trzeci argument funkcji `std::generate` podaj wyrażenie lambda. Przykład:

```
std::generate(czasteczki.begin(), czasteczki.end(), [&](double x) { return ... });
```

Etap 2: Transformacja do szybkości skalarnych (`std::transform`)

1. Utwórz wektor `std::vector<double> szybkosci`, w którym przechowywane będą moduły prędkości (szybkości) cząsteczek.
2. Wykorzystaj algorytm `std::transform` oraz `std::back_inserter`, aby obliczyć szybkość każdej cząsteczki i wstawić je do wektora `szybkosci`. Przykład:

```
      std::transform(czasteczki.begin(), czasteczki.end(), back_inserter(szybkosci),
    LAMBDA)

```

Etap 3: Weryfikacja termodynamiki (`std::accumulate`)

1. Oblicz całkowitą energię kinetyczną układu przy użyciu `std::accumulate`.
2. Wyznacz temperaturę gazu ze wzoru: $T = \frac{2}{3}\frac{\langle E_k \rangle}{k_B}$, gdzie $\langle E_k \rangle$ to średnia energia kinetyczna.
3. Sprawdź, czy uzyskany wynik jest bliski założonym 300 K.

Etap 4: Poszukiwanie ekstremów (`std::min_element, std::max_element, std::minmax_element`)

Znajdź obiekty o najmniejszej i największej szybkości. Wypisz te szybkości.

Etap 5: Poszukiwanie cząsteczek mogących reagować chemicznie (`std::count_if`)

Załóżmy, że cząsteczki mogą wziąć udział w reakcji chemicznej, jeśli ich szybkość przekracza 2500 m/s. Policz, ile jest takich cząsteczek.

Etap 6: Chłodzenie (`std::copy_if, std::erase_if`)

Załóżmy, że gaz jest w pojemniku z zaworem, który przepuszcza cząsteczki o szybkości większej od 2500 m/s/.

1. Skopiuj do nowego wektora `uciekinierzy` cząsteczki, których prędkość przekracza 2500 m/s (`std::copy_if`).
2. Usuń te cząsteczki z oryginalnego pojemnika (`std::erase_if`).
3. Ponownie oblicz temperaturę gazu w pojemniku i zaobserwuj jej spadek.

Etap 6: Analiza uciekinierów (`std::sort`)

1. Posortuj wektor uciekinierów malejąco według ich prędkości.
2. Wypisz prędkości trzech najszybszych cząsteczek, które opuściły układ. Do wypisywania ich wartości użyj `ostream_iterator`.

Opracowanie: Piotr Dziekan