Lattice Boltzmann Method
fluid flow

Bartłomiej Szwarga

Styczeń 2025

1 Zastosowane narzędzia

Do realizacji projektu symulacji przepływu płynów metodą Lattice Boltzmann (LBM) zastosowano język programowania C++, który dzięki swojej wydajności stanowi doskonały wybór do implementacji algorytmów wymagających dużej mocy obliczeniowej.

W projekcie LBM fluid flow, podobnie jak w poprzedniej wersji LBM diffusion, do wizualizacji wyników symulacji użyto biblioteki SFML (Simple and Fast Multimedia Library). Dzięki tej bibliotece możliwe było stworzenie efektywnego interfejsu graficznego, obsługi zdarzeń oraz zarządzania multimediami. Ponadto, do renderowania grafiki, zastosowano OpenGL, który umożliwia wykorzystywanie zaawansowanych technik graficznych i zapewnia wysoką wydajność przy renderowaniu dużych ilości danych w czasie rzeczywistym.

Połączenie C++, SFML i OpenGL okazało się kluczowe w realizacji projektu LBM fluid flow, umożliwiając płynne przeniesienie doświadczeń z poprzedniej implementacji LBM diffusion. C++ zapewnił solidną platformę do przeprowadzania obliczeń numerycznych, natomiast SFML i OpenGL umożliwiły stworzenie wizualizacji, pozwalających na analizę dynamiki przepływu płynów w czasie rzeczywistym.

2 Opis i realizacja modelu

W projekcie zaimplementowano model przepływu płynów oparty na metodzie Lattice Boltzmann (LBM), który jest rozwinięciem klasycznego modelu Lattice Gas Automaton (LGA). Model ten wykorzystuje dyskretne stany komórek oraz dwuwymiarową macierz *Matrix*, której elementami są obiekty klasy *Cell*. Klasa *Simulation* jest odpowiedzialna za obliczenia związane z aktualizacją stanów komórek oraz synchronizowanie operacji w każdej iteracji symulacji przepływu płynów.

2.1 Dane o środowisku modelowym

Środowisko modelowe zostało stworzone w oparciu o dwuwymiarową siatkę komórek, której rozmiar został określony przez użytkownika przy inicjalizacji symulacji. Każda komórka w siatce reprezentuje jednostkowy obszar w przestrzeni, który może przechowywać informacje o funkcjach rozkładu cząsteczek, gęstości i prędkości. Symulacja wykorzystuje metodę Lattice Boltzmann, która umożliwia modelowanie przepływów płynów na poziomie mikroskalowym.

Dla celów tego projektu, środowisko symulacyjne zostało zaprojektowane z następującymi założeniami:

- **Rozmiar siatki**: Model symulacji wykorzystuje siatkę o wymiarach $rows \times columns$, gdzie liczba wierszy i kolumn jest definiowana przy tworzeniu obiektu klasy *Simulation*.
- **Rodzaj komórek**: Każda komórka jest inicjalizowana w funkcji *prepare_envir* zgodnie z założeniami: gęstość c = 1.0 w jednej części i c = 0.95-0.99 w innej, prędkość w całym obszarze u = 0.0.
- **Warunki brzegowe**: Symulacja uwzględnia ściany (**WALL**), które ograniczają przepływ płynu w przestrzeni. Komórki brzegowe są inicjalizowane tak, aby oddziaływały z przepływem w odpowiedni sposób.
- **Typ przepływu**: W projekcie zastosowano przepływ płynów w dwóch wymiarach, z symulacją interakcji cząsteczek, ich transportem i kolizjami.

2.2 Dane wejściowe

Dane wejściowe w projekcie Lattice Boltzmann Method (LBM) są zdefiniowane przez szereg parametrów, które określają fizyczne i przestrzenne właściwości modelowanego przepływu. Kluczowe dane wejściowe to m.in. ustawienia związane z funkcjami rozkładu, wektorami prędkości, współczynnikami wag oraz innymi stałymi używanymi w algorytmie. W tej sekcji opisane zostaną szczegóły danych wejściowych wykorzystywanych w projekcie.

- **Funkcje rozkładu**: W projekcie zdefiniowano dwie stałe tablice, które reprezentują różne stany funkcji rozkładu cząsteczek:
  - **EMPTY**: Tablica o wartościach 0,0 we wszystkich 9 kierunkach. Reprezentuje komórki, które są puste, tzn. nie zawierają cząsteczek.
  - **WALL**: Tablica o wartościach -1.0 we wszystkich 9 kierunkach. Reprezentuje komórki, które pełnią rolę ścian, gdzie przepływ cząsteczek jest niemożliwy, a cząsteczki odbijają się od ścian.
- **Współczynniki wag**: Wartości funkcji wag są przechowywane w tablicy **WEIGHT_FACTOR**, która zawiera współczynniki odpowiadające różnym kierunkom cząsteczek w przestrzeni. Służą one do obliczania wpływu każdej z funkcji rozkładu na wyniki symulacji. Wartości te różnią

się w zależności od kierunku, a dla centralnego kierunku (kierunek 0) mają większą wagę (4/9), podczas gdy pozostałe kierunki mają wagę 1/9 lub 1/36, zależnie od swojego kąta względem osi układu współrzędnych.

- **Wektory prędkości**: Tablica **VELOCITY_FACTOR** zawiera wektory prędkości, które wskazują, jak cząsteczki przemieszczają się w przestrzeni w różnych kierunkach. Jest to tablica 9-elementowa, w której każdy element jest parą liczb reprezentujących przesunięcie w kierunku osi x i y. Na przykład, dla kierunku 1, wektor prędkości to {1, 0}, co oznacza, że cząsteczka porusza się o 1 jednostkę wzdłuż osi x.

- **Przeciwne kierunki**: **OPPOSITE_DIRECTION** to tablica zawierająca numery przeciwstawnych kierunków dla każdego z kierunków ruchu cząsteczek. Jest to istotne, gdy cząsteczki odbijają się od ścian lub oddziałują ze sobą. Przykład: dla kierunku 1 (prawo), przeciwnym kierunkiem jest kierunek 2 (lewo).

- **Stała czas relaksacji**: Wartość **RELAXATION_TIME** jest stałą wykorzystywaną w obliczeniach związanych z kolizjami cząsteczek. Określa czas, w którym cząsteczki relaksują się do funkcji równowagi. Wartość ta jest ustawiona na 1.0 i wpływa na stabilność i dokładność symulacji.

Wszystkie te dane wejściowe są używane do konfiguracji przestrzeni symulacyjnej i są przetwarzane podczas każdej iteracji symulacji, zapewniając realistyczne odwzorowanie przepływu płynów oraz oddziaływań między cząsteczkami w ramach modelu LBM.

2.3 Klasa Cell

Klasa *Cell* reprezentuje podstawową jednostkę przestrzeni symulacji w modelu Lattice Boltzmann. Każda komórka przechowuje kluczowe informacje umożliwiające dokładny opis przepływu płynów.

Dane przechowywane w komórce obejmują trzy tablice funkcji rozkładu:

- **fun-in**: funkcję wejściową, która opisuje transport masy do wewnątrz komórki z dziewięciu kierunków w danym kroku czasowym.

- **fun-ex**: funkcję wyjściową, która opisuje transport masy z komórki na zewnątrz.

- **fun-eq**: funkcję rozkładu w stanie równowagi, która reprezentuje stan równowagi cząstek w komórce w przypadku braku zewnętrznych oddziaływań.

Ponadto, każda komórka przechowuje:

- **density**: lokalną gęstość cząstek, obliczaną jako suma wartości w tablicy **fun-in**.

- **velocity**: wektory prędkości w kierunkach poziomym i pionowym, obliczane na podstawie funkcji rozkładu.

Dodatkowo, każda komórka zawiera informacje o stanie komórki, takie jak *WALL* (ściana) czy *EMPTY* (pusta przestrzeń), które są wykorzystywane do określania warunków brzegowych w symulacji. Klasa ta jest również odpowiedzialna za obliczanie gęstości, prędkości, funkcji równowagi oraz funkcji wyjściowej, co stanowi kluczowy element w iteracjach symulacji.

2.4 Klasa Matrix

Przestrzeń symulacji jest reprezentowana w postaci dwuwymiarowej macierzy obiektów klasy *Cell*, tworzącej układ komórek w obrębie całego obszaru symulacji.

Za przygotowanie środowiska symulacji odpowiada funkcja *prepare_ environment*, która iteruje po wszystkich komórkach macierzy i dokonuje odpowiednich modyfikacji, zależnie od ich położenia i roli w symulacji. W szczególności, funkcja ta ustawia odpowiednie warunki brzegowe dla komórek znajdujących się na krawędziach przestrzeni symulacji, jak również inicjalizuje gęstość i funkcje rozkładu dla innych komórek.

Komórki na brzegach macierzy, określone przez warunki: pierwszy i ostatni wiersz oraz pierwsza i ostatnia kolumna (czyli ściany), otrzymują stan **WALL**, co oznacza, że nie przepływa przez nie masa. Dodatkowo, w określonym fragmencie przestrzeni, na przykład w środkowej części lewej strony, również ustawiane są komórki jako **WALL**. Pozostałe komórki, które nie są brzegami, są traktowane jako przestrzeń, przez którą może przepływać masa. Dla tych komórek funkcja *prepare_ environment* ustawia lokalną gęstość cząsteczek, która wynosi 1.0 dla komórek znajdujących się w określonych obszarach i 0.96 dla pozostałych.

Po przypisaniu gęstości, dla każdej komórki obliczane są funkcje rozkładu w stanie równowagi (**fun_eq**) za pomocą metody *calculate_fun_ eq()*. Funkcja rozkładu w stanie równowagi jest ustawiana jako funkcja wejściowa (**fun_in**) dla każdej komórki, co zapewnia, że każda komórka początkowo znajduje się w stanie równowagi.

2.5 Klasa Simulation

Klasa *Simulation* odpowiada za przeprowadzanie symulacji przepływu przy użyciu metody Lattice Boltzmann (LBM). Symulacja składa się z dwóch głównych etapów: streamingu oraz kolizji. W obu tych etapach wykorzystuje się dwuwymiarową macierz obiektów klasy *Cell*, a operacje są przeprowadzane na kopii oryginalnej macierzy, co zapewnia spójność wyników w obrębie każdej iteracji. Poniżej przedstawiono szczegóły implementacji tych operacji.

2.5.1 Obsługa streamingu

Etap streamingu polega na przesuwaniu cząsteczek pomiędzy komórkami zgodnie z kierunkami ich bieżącego ruchu. W tej operacji każda komórka przekazuje

swoje funkcje wyjściowe do sąsiednich komórek, a cząsteczki przemieszczają się w kierunku wskazanym przez te funkcje, pod warunkiem że nie napotkają komórki typu **WALL**. W przypadku napotkania ściany, cząsteczka jest odbijana, a jej kierunek ruchu zmienia się na przeciwny. Dzięki temu procesowi odzwierciedlane są realistyczne zjawiska przepływu w ograniczonej przestrzeni.

Funkcja `streaming` w klasie *Simulation* wykonuje tę operację, aktualizując funkcje wejściowe komórek na podstawie funkcji wyjściowych ich sąsiadów. Wartości funkcji wyjściowych są przesuwane do odpowiednich komórek sąsiednich, a funkcje wejściowe komórek są resetowane na wartość **EMPTY**. W przypadku, gdy cząsteczka napotka ścianę, jej funkcja wejściowa w odpowiedniej komórce zostaje zaktualizowana zgodnie z kierunkiem przeciwnym do kierunku przepływu. Poniżej przedstawiono realizację tej operacji:

```
void Simulation::streaming() {
    Matrix next_matrix = s;
    int rows = s.get_rows_num();
    int columns = s.get_columns_num();
    for (size_t i = 0; i < rows; i++) {
        for (size_t j = 0; j < columns; j++) {
            if (s.get_element(i, j).get_fun(FUN_IN) != WALL) {
                Cell& cell = next_matrix.get_element(i, j);
                cell.set_fun(FUN_IN, EMPTY);
                cell.set_fun(FUN_EQ, EMPTY);
                cell.set_fun(FUN_EX, EMPTY);
            }}}
    for (size_t i = 0; i < rows; i++) {
        for (size_t j = 0; j < columns; j++) {
            Cell& current_cell = s.get_element(i, j);
            array<double, 9> fun_ex = current_cell.get_fun(FUN_EX);
            for (int d = 0; d < 9; d++) {
                if (fun_ex[d] != 0) {
                    int ni = i + VELOCITY_FACTOR[d][0];
                    int nj = j + VELOCITY_FACTOR[d][1];
                    int opposite_d = OPPOSITE_DIRECTION[d];
                    if (ni >= 0 && ni < rows && nj >= 0 && nj < columns) {
                        Cell& target_cell = next_matrix.get_element(ni, nj);
                        if (target_cell.get_fun(FUN_IN) != WALL) {
                            target_cell.set_direct_fun(FUN_IN, d, \
                            target_cell.get_fun(FUN_IN)[d] + fun_ex[d]);
                        }
                        else {
                           Cell& same_cell = next_matrix.get_element(i, j);
                           same_cell.set_direct_fun(FUN_IN, opposite_d, \
                           same_cell.get_fun(FUN_IN)[opposite_d] + fun_ex[d]);
                        }}}}}}
    s = next_matrix;
}
```

2.5.2 Obsługa kolizji

Kolizja w metodzie LBM polega na modyfikacji funkcji rozkładu cząsteczek w komórkach w wyniku ich interakcji. W implementacji przyjęto, że kolizja prowadzi do przekształcenia funkcji rozkładu poprzez relaksację do funkcji równowagi. Każda komórka w obrębie symulacji oblicza nową gęstość, a także wartości funkcji rozkładu w stanie równowagi oraz po kolizji.

Proces kolizji realizowany jest w funkcji `collision`, która wykonuje obliczenia w każdej komórce, aktualizując odpowiednie parametry. W pierwszej kolejności dla każdej komórki obliczana jest gęstość cząsteczek, a także prędkość, na podstawie której wyliczane są funkcje równowagi. Następnie obliczane są nowe funkcje wyjściowe, a system przechodzi do kolejnej iteracji. Ponadto, obliczana jest całkowita gęstość systemu, której suma jest wyświetlana po zakończeniu iteracji. Na następnej stronie przedstawiono fragment kodu realizującego operację kolizji:

```
void Simulation::collision() {
    int rows = s.get_rows_num();
    int columns = s.get_columns_num();
    double total_density = 0.0;
    int y = 0;
    for (size_t i = 0; i < rows; i++) {
        for (size_t j = 0; j < columns; j++) {
            Cell& next_cell = s.get_element(i, j);
            if (s.get_element(i, j).get_fun(FUN_IN) != WALL) {
                next_cell.calculate_density();
                next_cell.calculate_velocity();
                next_cell.calculate_fun_eq();
                next_cell.calculate_fun_ex();

               total_density += next_cell.get_density();
            }
        }
    }
    cout << "Całkowita masa w iteracji: " << total_density << endl;
}
```

Dzięki tej metodzie możliwe jest realistyczne odwzorowanie interakcji cząsteczek w płynach oraz ich przepływów, a suma gęstości w systemie pozwala na monitorowanie integralności fizycznej symulacji w każdej iteracji.

3 Wyniki symulacji

W niniejszej sekcji przedstawiono wyniki symulacji, które obrazują ewolucję stanu systemu w kolejnych iteracjach. Szczególną uwagę zwrócono na wczesne etapy symulacji, gdzie zachodzą najbardziej dynamiczne zmiany w strukturze przepływu. Obserwacje te są kluczowe, ponieważ pozwalają na lepsze zrozumienie mechanizmów rządzących dynamiką przepływu oraz jego stabilizację w późniejszych etapach.

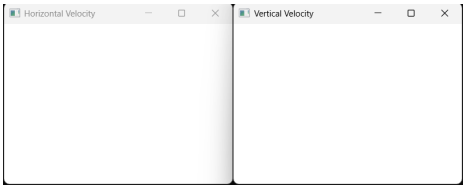

Figure 1: Stan początkowy, iteracja: 0

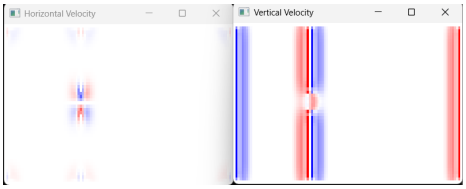

Figure 2: Stan przejściowy, iteracja: 10

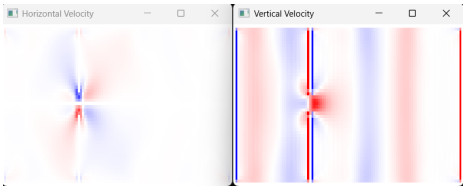

Figure 3: Stan przejściowy, iteracja: 50

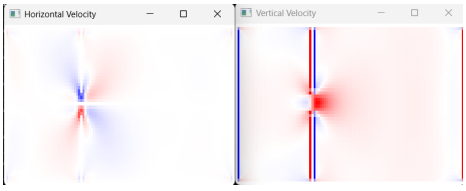

Figure 4: Stan przejściowy, iteracja: 150

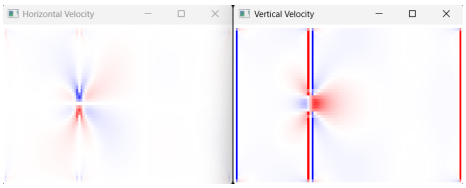

Figure 5: Stan przejściowy, iteracja: 250

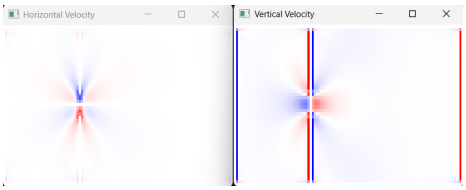

Figure 6: Stan przejściowy, iteracja: 500

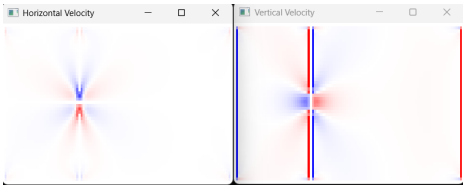

Figure 7: Stan końcowy, iteracja: 900

4 Porównanie symulacji LGA, LBM D2Q4 i LBM D2Q9

W tej sekcji przedstawiono porównanie trzech różnych symulacji: Lattice Gas Automata (LGA), Lattice Boltzmann Method (LBM) z układem D2Q4 oraz LBM dla przepływu płynów z układem D2Q9. Celem tego porównania jest zrozumienie, jak różne modele wpływają na wyniki symulacji oraz jakie mają zalety i ograniczenia w kontekście modelowania procesów fizycznych.

4.1 Lattice Gas Automata (LGA)

Lattice Gas Automata (LGA) to metoda, która jest uważana za prekursora Lattice Boltzmann Method. W tej metodzie cząsteczki płynów poruszają się w siatce, a ich zachowanie jest opisane przez zbiór reguł, które ustalają, jak cząsteczki mogą zmieniać swoje stany w wyniku kolizji i przesunięcia. Model ten jest stosunkowo prosty i pozwala na modelowanie podstawowych zjawisk płynów, takich jak transport masy, pęknięcia oraz podstawowe przepływy. Jednak, LGA jest ograniczona w kontekście bardziej złożonych symulacji, takich jak przepływy turbulentne.

4.2 Lattice Boltzmann Method (LBM) D2Q4

Lattice Boltzmann Method (LBM) z układem D2Q4 jest jedną z wersji LBM, która używa czterech kierunków (D2Q4) do modelowania przepływów dwuwymiarowych. W tym układzie cząsteczki poruszają się w czterech kierunkach na siatce, co pozwala na modelowanie podstawowych procesów dyfuzji w dwóch wymiarach. Jest to stosunkowo prosty model, ale oferuje więcej możliwości niż LGA, pozwalając na lepsze odwzorowanie zjawisk fizycznych, takich jak dyfuzja i transport. Choć jest mniej złożony od układu D2Q9, jego wydajność jest wyższa przy mniejszym obciążeniu obliczeniowym, co może być przydatne w przypadku prostszych symulacji dyfuzji.

4.3 Lattice Boltzmann Method (LBM) D2Q9 - Przepływ Płynów

Układ D2Q9 w LBM jest bardziej złożoną metodą, która pozwala na modelowanie przepływów płynów w dwóch wymiarach, przy użyciu dziewięciu kierunków. Dzięki tej większej liczbie kierunków, LBM D2Q9 może odwzorować bardziej skomplikowane zjawiska, takie jak turbulentne przepływy czy inne nieliniowe efekty fizyczne, których LGA i D2Q4 nie są w stanie w pełni odwzorować. Model ten jest bardziej skomplikowany, ale zapewnia lepszą dokładność w symulacjach fluidów, a jego zastosowanie znajduje się w różnych dziedzinach, takich jak aerodynamika, meteorologia oraz inżynieria płynów.

4.4 Porównanie wyników symulacji

Figure 8: Wyniki symulacji LGA: dyfuzja i transport w układzie 2D.

Figure 9: Wyniki symulacji LBM D2Q4: dyfuzja w układzie 2D przy użyciu czterech kierunków.

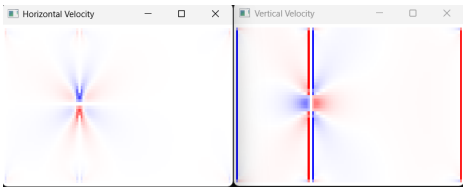

Figure 10: Wyniki symulacji LBM D2Q9: przepływ płynów w układzie 2D przy użyciu dziewięciu kierunków.

4.5 Wnioski

Porównanie trzech metod - LGA, LBM D2Q4 i LBM D2Q9 - pozwala na zauważenie kilku istotnych różnic:

- **LGA** jest najprostszą metodą, ale jej dokładność jest ograniczona. Nadaje się głównie do prostych symulacji przepływów i dyfuzji.
- **LBM D2Q4** oferuje większą dokładność w symulacjach dyfuzji, choć jest mniej rozbudowany niż D2Q9. Model ten jest stosunkowo wydajny, co sprawia, że jest dobrym wyborem do prostszych symulacji, które wymagają niewielkich zasobów obliczeniowych.
- **LBM D2Q9** zapewnia najdokładniejsze wyniki, zwłaszcza w przypadku przepływów płynów, w tym bardziej złożonych i turbulentnych. Jest bardziej wymagający pod względem zasobów obliczeniowych, ale daje najlepsze wyniki w bardziej zaawansowanych symulacjach.

Podsumowując, wybór odpowiedniej metody zależy od specyficznych potrzeb symulacji. Jeśli celem jest szybkie uzyskanie wyników z mniejszym obciążeniem obliczeniowym, LGA lub LBM D2Q4 będą dobrym wyborem. Natomiast, jeśli celem jest dokładna analiza bardziej skomplikowanych przepływów płynów, LBM D2Q9 będzie najodpowiedniejszą metodą.