Lattice Gas Automata

Bartłomiej Szwarga

Grudzień 2024

1 Zastosowane narzędzia

Do pisania i kompilacji programu wykorzystano język programowania C++, który jest szeroko stosowany w aplikacjach wymagających wysokiej wydajności, takich jak symulacje czy przetwarzanie danych.

W celu wizualizacji wyników symulacji zastosowano SFML (Simple and Fast Multimedia Library), bibliotekę, która pozwala na łatwe tworzenie aplikacji graficznych i obsługę okien, dźwięku oraz wejścia/wyjścia. Do renderowania grafiki użyto OpenGL, popularnej technologii graficznej, która pozwala na bezpośrednią manipulację grafiką 2D i 3D. OpenGL zapewnia dużą elastyczność w tworzeniu wizualizacji i jest idealnym wyborem do realizacji tego typu symulacji w czasie rzeczywistym.

Dzięki tym narzędziom i technologiom, możliwe było stworzenie wydajnej symulacji gazu opartej na automacie komórkowym oraz wizualizacji wyników w czasie rzeczywistym. C++ stanowi solidną bazę dla obliczeń i logiki symulacji, podczas gdy SFML i OpenGL pozwalają na płynne i efektowne wyświetlanie wyników na ekranie.

2 Opis i realizacja modelu

W projekcie zaimplementowano prosty wariant modelu Lattice Gas Automaton (LGA), który jest oparty na dyskretnych stanach komórek. Model realizuje dwie główne operacje: kolizję i streaming. Przestrzeń symulacji została zdyskretyzowana do postaci dwuwymiarowej macierzy *Matrix*, której elementami są obiekty klasy *Cell*. Za implementacje logiki symulacji odpowiada klasa *Simulation*.

2.1 Klasa Cell

Każda komórka przechowuje wektor czterech wartości reprezentujących obecność cząsteczek w czterech podstawowych kierunkach.

Komórki mogą przyjmować jeden z trzech stanów:

- **EMPTY**: brak cząsteczek w komórce, co oznacza, że jej wektor zawiera same zera: $\{0, 0, 0, 0\}$. Komórki w tym stanie są wizualizowane jako białe.
- **WALL**: komórka ściany, od której odbijają się cząsteczki gazu. Jej wektor jest wypełniony wartościami -1: $\{-1, -1, -1, -1\}$. Ściany są wizualizowane w kolorze szarym.
- **ALIVE**: komórka zawierająca co najmniej jedną cząsteczkę gazu. Wektor ma przynajmniej jedną wartość równą 1, co oznacza obecność ruchu w określonym kierunku. Przykład: $\{1, 0, 0, 0\}$. Komórki w tym stanie są wizualizowane jako czarne.

Elementy wektora komórki *ALIVE* oznaczają ruch w czterech kierunkach:

- Pierwszy element: ruch w prawo,
- Drugi element: ruch w dół,
- Trzeci element: ruch w lewo,
- Czwarty element: ruch w górę.

2.2 Klasa Matrix

Przestrzeń symulacji domyślnie wypełniona jest komórkami typu **EMPTY**. Przygotowanie środowiska odbywa się za pomocą metody *prepare-environment*, która modyfikuje wybrane komórki, aby odwzorować rozmieszczenie ścian. W przykładzie na rysunku poniżej przedstawiono przygotowane środowisko symulacji.

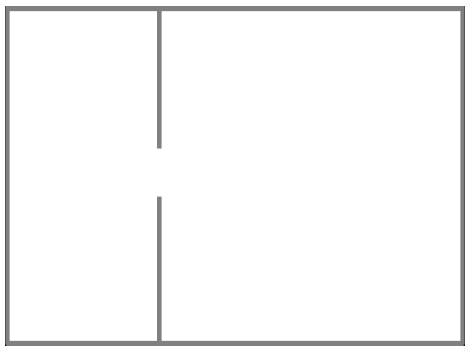

Figure 1: Środowisko symulacji

W symulacji cząsteczki gazu zostały rozmieszczone losowo, maksymalnie jedna cząsteczka gazu w pustej komórce. Ich ilość oraz odpowiednie kierunki ruchu(jeden dla każdej cząstki), zostały wylosowane za pomocą generatora liczb pseudolosowych, zapewniając zróżnicowany stan początkowy.

Figure 2: Stan początkowy

2.3 Klasa Simulation

Klasa *Simulation* odpowiada za sterowanie przebiegiem symulacji i realizację dwóch głównych operacji modelu LGA: **kolizji** oraz **streamingu**. W każdej iteracji algorytm przetwarza macierz symulacji w dwóch etapach:

1. **Kolizja** – przekształcenie wektorów ruchu w każdej komórce w oparciu o zasady zderzeń cząsteczek.
2. **Streaming** – przesunięcie cząsteczek między komórkami zgodnie z ich kierunkami ruchu.

Oba etapy są wykonywane na nowej macierzy (kopia oryginalnej macierzy), co zapewnia spójność wyników w obrębie każdej iteracji. W poniższych podsekcjach przedstawiono szczegóły implementacji obu operacji.

2.3.1 Obsługa kolizji

Kolizja polega na modyfikacji kierunków ruchu cząsteczek w komórkach typu **ALIVE**, gdy dochodzi do ich interakcji. W implementacji założono, że kolizja prowadzi do równomiernego rozproszenia cząsteczek w kierunkach przeciwnych. Na przykład, jeśli w danej komórce występuje wektor ruchu w kierunku poziomym (w prawo i w lewo), to po kolizji wektor zostaje zmieniony na odpowiadający ruchowi w kierunku pionowym (w górę i w dół). Takie podejście zapewnia zachowanie zasad zachowania pędu i energii w symulacji.

```
void Simulation::collision() {
    Matrix next_matrix = s;
    int rows = s.get_rows_num();
    int columns = s.get_columns_num();
    vector<int> left_right{ 1, 0, 1, 0 };
    vector<int> up_down{ 0, 1, 0, 1 };
    for (size_t i = 1; i < rows-1; i++) {
        for (size_t j = 1; j < columns-1; j++) {
            const vector<int>& info = s.get_element(i, j).get_info();
            Cell& future_cell = next_matrix.get_element(i, j);
            if (info == left_right) future_cell.set_info(up_down);
            if (info == up_down) future_cell.set_info(left_right);
        }
    }
    s = next_matrix;
}
```

2.3.2 Obsługa streamingu

Streaming odpowiada za przesunięcie cząsteczek między komórkami zgodnie z ich bieżącymi kierunkami ruchu. Każda cząsteczka przemieszcza się do sąsiedniej komórki w wyznaczonym kierunku, pod warunkiem, że nie jest to komórka typu **WALL**. Jeśli cząsteczka napotka ścianę, zostaje odbita, a jej kierunek ruchu ulega zmianie na przeciwny. Dzięki temu operacja streamingu odzwierciedla realistyczne zjawiska przemieszczania się cząsteczek w ograniczonej przestrzeni.

```
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_info() != WALL) {
                for (int d = 0; d < 4; d++) {
                    next_matrix.get_element(i, j).set_direct_info(d, 0);
                }
            }
        }
    }
```

```
    for (size_t i = 1; i < rows - 1; i++) {
        for (size_t j = 1; j < columns - 1; j++) {
            Cell& current_cell = s.get_element(i, j);
            vector<int> info = current_cell.get_info();
            if (info[0] == 1) {
                Cell& right_cell = s.get_element(i, j + 1);
                if (right_cell.get_info() == WALL) {
                    next_matrix.get_element(i, j).set_direct_info(2, 1);
                }
                else {
                    next_matrix.get_element(i, j + 1).set_direct_info(0, 1);
                }
            }
            if (info[1] == 1) {
                Cell& down_cell = s.get_element(i + 1, j);
                if (down_cell.get_info() == WALL) {
                    next_matrix.get_element(i, j).set_direct_info(3, 1);
                }
                else {
                    next_matrix.get_element(i + 1, j).set_direct_info(1, 1);
                }
            }
            if (info[2] == 1) {
                Cell& left_cell = s.get_element(i, j - 1);
                if (left_cell.get_info() == WALL) {
                    next_matrix.get_element(i, j).set_direct_info(0, 1);
                }
                else {
                    next_matrix.get_element(i, j - 1).set_direct_info(2, 1);
                }
            }
            if (info[3] == 1) {
                Cell& up_cell = s.get_element(i - 1, j);
                if (up_cell.get_info() == WALL) {
                    next_matrix.get_element(i, j).set_direct_info(1, 1);
                }
                else {
                    next_matrix.get_element(i - 1, j).set_direct_info(3, 1);
                }
            }
        }
    }
    s = next_matrix;
}
```

Te dwie operacje, wykonywane na przemian w każdej iteracji symulacji, pozwalają na odwzorowanie dynamicznego zachowania gazu w układzie, w tym rozprzestrzeniania się, odbijania od ścian i dążenia do równowagi.

3 Wyniki symulacji

Wyniki symulacji modelu Lattice Gas Automaton (LGA) przedstawiono na serii rysunków, które ukazują kolejne etapy ewolucji układu gazu w przestrzeni symulacyjnej. Na rysunku przedstawiającym stan początkowy widzimy losowe rozmieszczenie cząsteczek gazu w przestrzeni symulacyjnej. W każdej komórce znajduje się maksymalnie jedna cząsteczka, której kierunek ruchu został przypisany losowo, co zapewnia zróżnicowany i realistyczny stan początkowy układu.

W kolejnych etapach symulacji obserwujemy, jak cząsteczki zaczynają się przemieszczać zgodnie z przypisanymi kierunkami. Proces ten jest realizowany przez operację streamingu, która przesuwa cząsteczki do sąsiednich komórek. Z czasem niektóre z nich natrafiają na komórki ściany (**WALL**), które odbijają cząsteczki, zmieniając ich kierunki ruchu. Efekt ten widoczny jest na rysunkach przedstawiających pierwsze i drugie odbicie cząsteczek od ścian, gdzie można zaobserwować tworzenie się charakterystycznych wzorców wynikających z interakcji gazu z otoczeniem.

Na ostatnim rysunku ukazany jest stan końcowy symulacji, w którym cząsteczki rozproszyły się po całej przestrzeni symulacyjnej. Widoczny jest równomierny rozkład gazu, przypominający normalny rozkład cząsteczek w układzie w stanie równowagi. Wynik ten wskazuje na zgodność modelu z oczekiwanym zachowaniem gazu w zamkniętej przestrzeni.

Figure 3: Stan początkowy

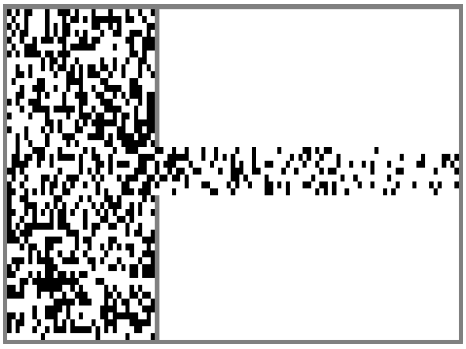

Figure 4: Rozprzestrzenianie się komórek

Figure 5: Kolizja ze ścianą

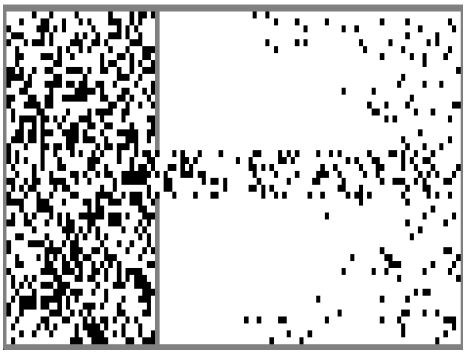

Figure 6: Kolizja z drugą ścianą

Figure 7: Stan końcowy