Symulacja Rozprzestrzeniania Zanieczyszczeń

Wstęp teoretyczny

Krzysztof Żywiecki Szymon Duda Piotr Krześniak

20 maja 2020

Opis programu

Do symulowania rozprzestrzeniania zanieczyszczeń wykorzystano automat komórkowy. W tym modelu przestrzeń zostaje podzielona na kwadratowe komórki o jednakowym rozmiarze. Każda komórka jako swój stan przechowuje liczbę zmiennoprzecinkową z przedziału $[0, +\infty]$ reprezentującą ilość substancji w danym momencie.

Program przechowuje dwie tablice dwuwymiarowe: stan obecny, oraz stan w poprzedniej iteracji, Na początku działania programu obie tablice są wyzerowane. Dodatkowo program do działania potrzebuje danych o środowisku:

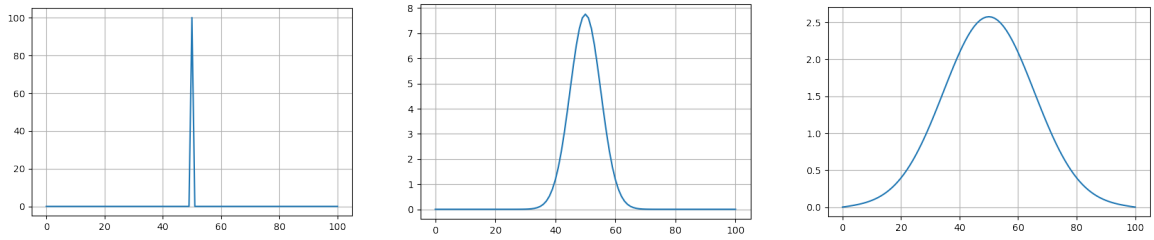
- Tablica w wymiarach mapy zawierająca liczbę zmiennoprzecinkową reprezentującą emisję zanieczyszczenia z każdej komórki
- Tablica o dowolnym wymiarze zawierająca wektory dwuwymiarowe z prędkością wiatru na danym obszarze
- Tablica o dowolnym wymiarze zawierająca ogólną trudność terenu, na przykład jego wypiętrzenie (jeszcze nie zaimplementowano)
- Dodatkowo wysyłane są parametry na temat fizycznej wielkości obszaru oraz przyspieszenia symulacji (jeszcze nie zaimplementowano)
- Czas wykonania poprzedniej iteracji w sekundach dla zapewnienia stałego tempa symulacji

Działanie programu

W obecnej fazie program nie uwzględnia wszystkich przewidzianych czynników, jednak zasada działania pozostanie podobna. Program uwzględnia dyfuzję zanieczyszczeń do sąsiednich komórek, oraz wpływ wiatru.

Dyfuzja jest liczona na podstawie średniej arytmetycznej obszaru wielkości 3 x 3 wokół komórki (z badaną włącznie). Wpływ dyfuzji wyliczany jest za pomocą wzoru

$$diffusion = (1 - damping)(average) + (damping)(current\_state)$$



(a) Dane początkowe (b) Wysoki damping (c) Mały damping

Rysunek 1: Wpływ wartości damping na wynik działania dyfuzji dla stu punktów po dwustu iteracjach

$damping$ jest wartością tłumienia, i w założeniu jest charakterystyczny dla rodzaju zanieczyszczenia.

Do stworzenia zrzutu ekranu aplikacji użyto algorytmu symulującego wpływ wiatru na zanieczyszczenia. Mechanizm działa na zasadzie iloczynu skalarnego między wektorem wiatru a wektorami prowadzącymi do sąsiednich komórek. Okazało się jednak że algorytm jest błędny, i włączenie samego wiatru daje wynik niezgodny z przewidywaniami. Algorytm został przedstawiony poniżej, gdyż jest on elementem aplikacji. W przyszłości zostanie on jednak zmieniony na inny.

```
1  struct cell{
2      vec2 position;
3      float pollution;
4  }
5
6  vec2 wind
7  cell neighbors[8]
8  cell current_cell
9
10 float balance = 0
11 /*------------------------------------------------------------*/
12
13 for each neighbor in neighbors:
14     vec2 rel_position = neighbor.position - current_cell.position
15     float weight = dot(wind, rel_position)
16     if(weight < 0):
17         balance = balance - neighbor.pollution * weight
18     else:
19         balance = balance - current_cell* weight
```

Finalny stan komórki otrzymuje się za pomocą wzoru

$$current\_state \cdot (1 - frame\_time) + (diffusion + balance + inflow) \cdot frame\_time$$

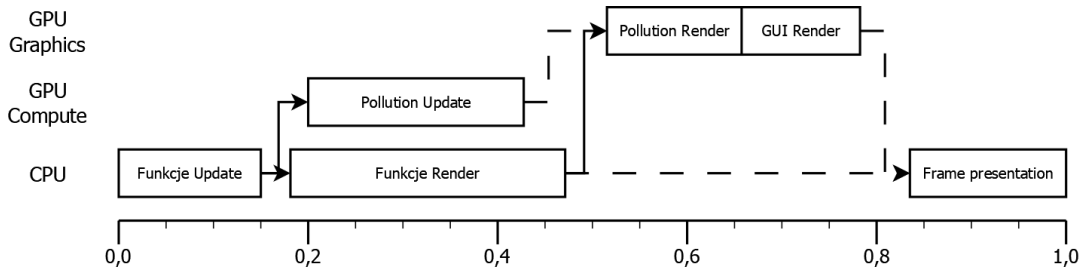
gdzie $frame\_time$ jest równe czasowi wykonania poprzedniej iteracji w sekundach, a $inflow$ to ilość zanieczyszczeń wpływająca co sekundę do komórki. $diffusion$ i $balance$ są wzięte z poprzedniej części.

Rysunek 2: Zrzut ekranu z aplikacji przy dziesięciu źródłach o intensywności 6. Podziałka umieszczana jest co 0.2 jednostki. Dana jest mapa wiatru złożona z czterech wektorów tworzących kwadrat. Tworzące się okręgi wynikają z interpolacji między tymi wartościami. Niebieska ramka wynika z tego że jest to kolor czyszczenia ekranu.

Wydajność

Wydajność programu w obecnym stanie jest znośna. Wszystkie obliczenia niezbędne do prowadzenia symulacji wykonywane są na karcie graficznej, co sprawia że program byłby prawdopodobnie dziesiątki może nawet setki razy szybszy niż odpowiednik wykorzystujący CPU.

Do wykonywania obliczeń wykorzystano standard Vulkan, gdyż umożliwia on zarówno operacje graficzne, jak i obliczeniowe. Standard ten zaprojektowany został z myślą o wydajności i niskim narzucie procesora na kartę graficzną. Problemem jest bardzo słabe obeznanie w tym standardzie, przez co potencjał technologii może nie być wykorzystywany najlepiej.



Rysunek 3: Uproszczony schemat przedstawiający operacje niezbędne do wykonania jednej iteracji symulacji i narysowania klatki. Linie ciągłe oznaczają wywołania, a linie przerywane reprezentują zależności logiczne.

Do wykonania powyższego obrazka posłużyło narzędzie Nvidia Nsight, które pozwala między innymi na takie przedstawienie wydajności. Można zauważyć że karta graficzna nie pracuje przez cały czas, jest przerwa między operacjami obliczeniowymi a rysowaniem kiedy karta nic nie robi. Żeby temu zapobiec, niezbędne jest albo zoptymalizowanie działania aplikacji po stronie procesora, albo zwiększenie nakładu pracy na karcie graficznej.

Test został przeprowadzony na mapie zanieczyszczeń wielkości 256 x 256 i karcie graficznej Nvidia GTX 1050 TI, która jest sprzętem ze średniej półki cenowej. Pozwala ona na uzyskanie między 300 a 700 iteracji na sekundę przy obecnej wersji programu.