Symulacja rozpszestrzeniania się dymu

Kamil Skomro Michał Warzecha

3 czerwca 2019

Spis treści

1 Wstęp

Niniejszy projekt ma na celu odwzorowanie rozpszestrzeniania się dymu. Jest to problem dość złożony, ale ze względu na zapotrzebowanie realistycznych symulacji na przykład w inżynierii lub branży filmowej powstało wiele programów oraz literatury, które pozwalają rozwiązać ten problem. Obliczenia zostały wykonane metodą iteracyjną podobnie jak w tej publikacji[3].

Dział mechaniki płynów wykorzystujący metody numeryczne do rozwiązywania zagadnień przypływu płynów nosi nazwę "Obliczeniowej mechaniki płynów". Do odzwierciedlenia poruszania się dymu wykorzystaliśmy równania Naviera-Stoke'a, które wyglądają następująco:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

gdzie:

- $\vec{u}$ to prędkość płynu, czyli w przestrzeni trójwymiarowej wektor (u,v,w),
- ρ to gęstość płynu,
- p to ciśnienie zdefiniowane wzorem $p = \frac{F}{A}$, gdzie F to siła prostopadła do powieszchni i A pole powierzchni,
- $\vec{g}$ to siła grawitacji,
- ν to lepkość płynu.

2 Równania matematyczne

W naszym rozwiązaniu dla każdej klatki wykonujemy kolejne czynności wg następującego schematu:

1: zainicjalizuj zmienne
2: **for** każdej klatki **do**
3: rozwiąż równania matematyczne dla prędkości;
4: rozwiąż równania matematyczne dla gęstości;
5: wyświetl;
6: **end for**

2.1 Rozwiązywanie równań dla prędkości

Numeryczne rozwiązanie dla prędkości ma miejsce w metodzie velocitySolver() w klasie FluidSolver. Wykorzystuje ona kolejne metody, aby obliczyć prędkości dla poszczególnych części naszego wirtualnego pomieszczenia:

1. addSource() - aby dodawać do poszczególnych wektorów drugi wektor * dt, gdzie dt to przyjęta długość symolowanego czasu od ostatniej pętli
2. vorticityConfiment() - która oblicza wirowość
3. buoyancy() - do obliczenia siły wyporowej
4. diffuse() - do obliczenia rozproszenia
5. advect() - do oblicznenia adwekcji
6. project() - do obliczenia projekcji

2.2 Rozwiązywanie równań dla gęstości

Numeryczne rozwiązanie dla gęstości ma miejsce w metodzie densitySolver() w klasie FluidSolver. Jest struktóra jest następująca:

1. addSource() - j.w.
2. diffuse() - j.w.
3. advect() - j.w.

2.3 Siła wyporu termicznego

W fizyce siła wyporowa jest siłą, która wypycha do góry i jest spowodowana ciśnieniem płynu. Wartość siły wyporu jest równa ciężarowi płynu wypartego przez to ciało.

Aby obliczyć tą siłę, potrzebujemy dwóch głównych zmiennych: T - dla temperatury i d - dla gęstości. Jeśli gorący gaz jest otoczony przez zimniejsze składniki powietrza to będzie się unosił.

Do obliczenia siły wyporowej używamy następującego wzoru:

$$F_{wyp} = (\alpha d - \beta(T - T_{śr}))\vec{g}$$

gdzie:

- $\vec{g}$ - to siła grawitacji
- α, β - to dodatnie stałe z odpowiednimi jednostkami
- T - temperatura w danym elemencie siatki
- $T_{śr}$ - to temperatura średnia w naszej siatce
- d - gęstość

W naszej uproszczonej implementacji zakładamy, że temperatura jest jednoznaczna z gęstością (ponieważ dym jest gorący) i nie ma żadnych innych źródeł ciepła niż dym.

2.4 Oblicznie wirowości

Dym porusza się w kierunku sąsiednich regionów o większej prędkości(albo o niższym ciśnieniu) co skutkuje wirowym ruchem, który można obliczyć za pomocą wzoru:

$$\vec{\omega} = \nabla \times \vec{u}$$

co oznacza po prostu jak bardzo pole prędkości obraca się wokół punktu. W trzech wymiarach można to obliczyć za pomocą poniższego wzoru:

$$\vec{\omega} = \nabla \times \vec{u} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} \frac{\partial v}{\partial x} - \frac{\partial w}{\partial y} \right)$$

Jednak przy obliczeniach powyższego równania wirowości, trochę wirowości jest traconej. Aby przywrócić ją z powrotem nowa technika obliczania wirowości została wprowadzona w [2] i jest modyfikacją równań Naviera-Stokes'a, które próbuje zachować wirowość. Wykorzystuje ona poniższy wzór

$$F_{wir} = \vec{N} \times \vec{\omega}$$

gdzie $\vec{\omega}$ to zawijanie się dymu a $\vec{N}$ to wektor skierowany do środka komórki:

$$\vec{N} = \frac{\nabla ||\vec{\omega}||}{||\nabla ||\vec{\omega}||||}$$

2.5 Adwekcja pół-Lagrange'a

Do opisu ruchu powietrza można używać dwóch sposobów. Jeden z nich zwany jest opisem metodą Eulera, drugi zaś metodą Lagrange'a.

Zgodnie z opisem Lagrange'a układ skupia się na śledzeniu pojedyńczych "paczek" powietrza wzdłuż ich trajektorii w przeciwieństwie do opisu Eulera, który rozważa ocenia zmiane zmiennych układu w umiejscowionych punktach w przestrzeni. Schemat pół-Lagrange'a używa szkieletu Eulera, ale dyskretyzacja równań jest zgodna z podejściem Lagrange'a.

Adwekcja (zamodelowana jako $\vec{u}\cdot\nabla\vec{u}$ w równaniu 1) mówi nam jak ciepło albo materia przemieszcza się z przepływem płynu. Aby znaleźć nowe położenie P dla cząstki z początkowym położeniem G metodą Eulera posłużylibyśmy się wzorem:

$$\vec{u_P} = \vec{u_G} - \Delta t\vec{u}(\vec{x})$$

Jednak jako, że my posługujemy się metodą pół-Lagrange'a nie obliczamy położenia dla cząsteczek, ale obliczamy prędkości dla pojedyńczych komórek naszej siatki. W jednym wymiarze wyglądało by to następująco.

Dla komórki x_i, która leży na przedziale $[x_j, x_{j+1}]$, nasza wirtualna cząstka jest śledzona wstecz do $x_P = x_i - \Delta tu$ i $\alpha = (x_P - x_i)\Delta x$. α reprezentuje "ułamek przedziału w którym cząstka ląduje". Więc nasz wzór na prędkość w punkcie P wygląda następująco:

$$q_P^n = (1-\alpha)q_j^n + \alpha q_{j+1}^n$$

Czyli w rezultacie pole prędkości jest przbliżone względem sąsiednich punktów.

2.6 Rozproszenie

Rozproszenie jest zmieszaniem się substancji pod wpływem ich ruchu. W równaniach Naviera-Stokes'a jest zamodelowane jako $\nu\nabla\cdot\nabla\vec{u}$, gdzie ν to lepkość, a $\nabla\cdot\nabla$ lub ∇^2 to operator Laplace'a, zatem równanie wygląda następująco:

$$\frac{\partial\vec{u}}{\partial t} = \nu\nabla^2 u = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

Załóżmy, że odległości między poszczególnymi komórkami siatki dla trzech wymiarów są następujące $(\Delta x, \Delta y, z)$, po dyskretyzacji i wydzieleniu związków z t na lewo i związków z $t+\Delta t$ na prawo nasze równanie wygląda następująco:

$$d^t_{x,y,z} = (1+6\beta)d^{t+\Delta t}_{x,y,z} - \beta(d^{t+\Delta t}_{x-1,y,z} + d^{t+\Delta t}_{x+1,y,z}d^{t+\Delta t}_{x,y-1,z} + d^{t+\Delta t}_{x,y+1,z}d^{t+\Delta t}_{x,y,z-1} + d^{t+\Delta t}_{x,y,z+1})$$

gdzie

$$\beta = \frac{\nu\Delta t}{\Delta x^2}, \Delta x = \Delta y = \Delta z$$

Aby rozwiązać powyższe równanie musimy przekształcić to w układ równań liniowych:

$$A\cdot d^{t+\Delta t} = d^t$$

Powyższy układ równań rozwiązujemy metodą Gaussa-Seidla, który jest iteracyjną metodą numerycznego rozwiązywania układów równań liniowych.

2.7 Projekcja

"Projekcja jest specjalnym operatorem liniowym, którego zaaplikowanie 2 razy daje taki sam efekt jak zaaplikowanie go raz" [1]. Na przykład macierz D jest projekcją jeśli $D^2 = D$. Fizycznie nasza transformacja z $u(x,t)$ na $u(x,t+\Delta t)$ jest również liniową projekcją.

Celem użycia projekcji jest uzyskanie z prędkości zachowującego masę, nieściśliwego pola. Jest to osiągnięte przy pomocy dekompozycji Helmholtz-Hodge'a. Najpierw obliczamy dywergencje pola wektorowego naszej prędkości przy pomocy ilorazu różnicowego i wtedy stosujemy iteracyjną metodę

Gaussa-Siedla, aby rozwiązać układ równań. Na końcu odejmujemy gradient wyniku, aby otrzymać pole prędkości zachowujące mase.

Dekompozycja Helmholtz-Hodge'a utrzymuje, że każde pole wektorowe może być unikalnie rozłożone do formy:

$$\vec{u} = \vec{u}^{n+1} + \Delta t \frac{1}{\rho} \nabla p \tag{3}$$

gdzie $\vec{u}^{n+1}$ jest wolne od dywergencji (czyli $\nabla \vec{u}^{n+1} = 0$). Jeśli zastosujemy operator projekcji do obu stron równania (3) otrzymamy:

$$\nabla \vec{u} = \nabla \vec{u}^{n+1} + \Delta t \nabla \cdot \frac{1}{\rho} \nabla p$$

Po odjęciu dywergencji mamy:

$$\nabla \vec{u} = \Delta t \nabla \cdot \frac{1}{\rho} \nabla p$$

co jest równaniem Poissona w postaci $\nabla^2 \varphi = f$

Teraz możemy zdyskretyzować naszą dywergencje na siatkę 3D używając ilorazu różnicowego:

$$(\nabla \cdot \vec{u})_{i,j,k} \approx \frac{u_{i+1,j,k} - u_{i-1,j,k}}{\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{\Delta x} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{\Delta x}$$

oraz

$$\begin{aligned}
\frac{1}{\Delta x} \times \{ &\big(u_{i+1,j,k} - \Delta t \frac{1}{\rho} \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x}\big) - \big(u_{i-1,j,k} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x}\big) \\
&+\big(v_{i,j+1,k} - \Delta t \frac{1}{\rho} \frac{p_{i,j+1,k} - p_{i,j,k}}{\Delta x}\big) - \big(v_{i,j-1,k} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k} - p_{i,j-1,k}}{\Delta x}\big) \\
&+\big(w_{i,j,k+1} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta x}\big) - \big(w_{i,j,k-1} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k} - p_{i,j,k-1}}{\Delta x}\big)\} = 0
\end{aligned}$$

co prowadzi do:

$$\begin{aligned}
\frac{\Delta t}{\rho} \left(\frac{6p_{i,j,k} - p_{i+1,j,k} - p_{i,j+1,k} - p_{i,j,k+1} - p_{i-1,j,k} - p_{i,j-1,k} - p_{i,j,k-1}}{\Delta x^2} \right) = \\
- \left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{\Delta x} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{\Delta x} \right)
\end{aligned}$$

To spory układ równań linowych do którego użyjemy metody Gaussa-Siedla. Naszym celem jest znalezienie $\vec{u}_{n+1}$. Znamy już $\vec{u}$, więc rozwiązujemy układ równań i otrzymujemy p. W ostatnim kroku odejmujemy gradient p i uzyskujemy nasze zachowujące mase pole prędkości:

$$\vec{u}_{n+1} = \vec{u} - \Delta t \frac{1}{\rho} \nabla p$$

2.8 Warunki brzegowe

W naszym projekcie definiujemy warunki brzegowe jako ścianki otaczające dym, w które nie może on się dostać ani wydostać na zewnątrz. Zatem matematycznie wektor normalny pola wektorowego prędkości musi być równy 0:

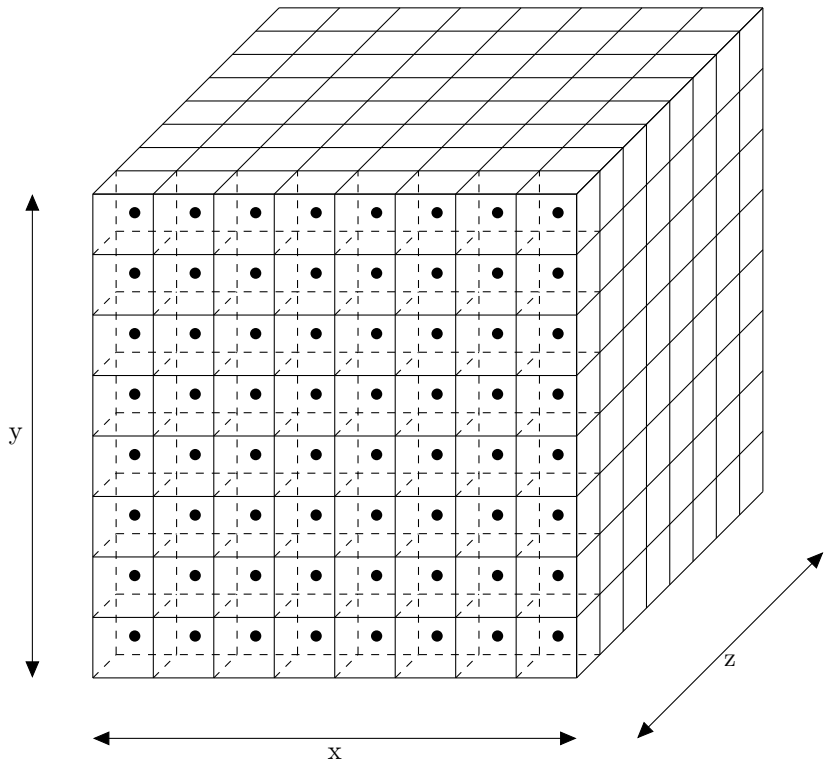
$$\vec{u} \cdot \hat{n} = 0$$

3 Implementacja

Do stworzenia naszej aplikacji do symulacji wykostaliśmy środowisko Unity, ponieważ naszym zdaniem posiada bardzo wygodny interfejs do tworzenia wszelkiego rodzaju animacji i symulacji.

3.1 Siatka

W naszej implementacji do symulacji przestrzeni wykorzystaliśmy trójwymiarową siatkę. Przyjęliśmy, że dla każdego elementu siatki jego parametry znajdują się w jego środku(rysunek poniżej).



3.2 Funkcje

Główne skrypty rozwiązujące symulacje od strony matematyczne to *FluidSolver.cs* oraz *SmokeManager.cs* znajdujące się w katalogu *Assets/MyProject/Scripts* naszego projektu. W pliku *SmokeManager.cs* znajduje się kod odpowiedzialny za pętle główną programu. Do rozwiązywania równań używamy klasę FluidSolver znajdującą się w pliku *FluidSolver.cs*. Poniższej przedstawiamy sygnatury metod i opis parametrów.

Funkcja do obliczania siły wyporowej:

$$public\ void\ buoyancy(float[]\ Fbuoy)$$

gdzie $Fbuoy$ to tablica w której przechowywana będzie obliczona siła wyporowa dla każdego elementu siatki.

Funkcja do obliczania skrętu dla pojedyńczego elementu siatki:

$$public\ float\ Curl(int\ x,\ int\ y,\ int\ z)$$

gdzie x, y, z to współrzędne elementu.

Funkcja do obliczania wirowości:

$$public\ void\ vorticityConfinement(float[]\ Fvc\_x,\ float[]\ Fvc\_y,\ float[]\ Fvc\_z)$$

gdzie Fvc_x, Fvc_y, Fvc_z to tablice, w których będą przechowywane wartości wirowości dla wszystkich elementów siatki dla kierunków x, y, z.

Funkcja do rozwiązywania równań matematycznych dla prędkości:

$$public\ void\ velocitySolver()$$

Funkcja do rozwiązywania równań matematycznych dla gęstości:

$$public\ void\ densitySolver()$$

Funkcja do obliczania adwekcji:

$$public\ void\ advect(int\ b,\ float[]\ d,\ float[]\ d0,\ float[]\ du,\ float[]dv,\ float[]\ dw)$$

gdzie:

- b - flaga, która mówi jak obsłużyć brzegi
- d - tablica do przechowywania obliczonej adwekcji
- $d0$ - tablica dla której obliczamy adwekcje
- du - składowa x prędkości
- dv - składowa y prędkości
- dw - składowa z prędkości

Funkcja do obliczania rozproszenia:

$$public\ void\ diffuse(int\ b,\ float[]\ c,\ float[]\ c0,\ float\ diff)$$

gdzie:

- b - flaga, która mówi jak obsłużyć brzegi
- c - tablica do przechowywania obliczonego rozproszenia
- $c0$ - tablica dla której obliczamy rozproszenie
- $diff$ - współczynnik rozproszenia

Funkcja do obliczania projekcji:

$$public\ void\ project(float[]\ x,\ float[]\ y,\ float[]\ z,\ float[]\ p,\ float[]\ div)$$

gdzie:

- x - tablica, w której będą przechowywane finalne wartości składowej x prędkości
- y - tablica, w której będą przechowywane finalne wartości składowej y prędkości
- z - tablica, w której będą przechowywane finalne wartości składowej z prędkości
- p - tablica pomocniczna do wykonywania obliczeń
- div - tablica w której będziemy przechowywać dywergencję

Funkcja do obliczania równań liniowych metodą Gaussa-Siedla:

$$void\ linearSolver(int\ b,\ float[]\ x,\ float[]\ x0,\ float\ a,\ float\ c)$$

gdzie:

- b - flaga, która mówi jak obsłużyć brzegi
- x - tablica dla której wykonujemy obliczenia
- $x0$ - tablica w której są wartości z poprzedniego kroku
- a, c - współczynniki

Funkcja dla warunków brzegowych:

$$private\ void\ setBoundry(int\ b,\ float[]\ x)$$

Literatura

[1] R. Bridson. *Fluid Simulation for Computer Graphics*. 2008.

[2] John Steinhoff i David Underhill. Modification of the euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 1994.

[3] J. Stam. *Real-time fluid dynamics for games*. 2003.