

Wydział Matematyki i Nauk Informacyjnych
Politechnika Warszawska

Warsztaty badawcze 2

Symulacja Przepływu z Przeszkodami

Autorzy:

Michał Dybowski
Kacper Kurowski
Paweł Lefelbajn

Prowadzący:

mgr Przemysław Kosewski

Warszawa, 2021

Spis treści

1	Wstęp	2
2	Octree	3
2.1	Cel	3
2.2	Opis działania	3
2.3	Kompilacja pakietu <i>Octree</i>	3
2.4	Zawartość pakietu <i>Octree</i>	4
3	Opis algorytmu propagacji cząstek	5
3.1	Opis ogólny	5
3.2	Jedna iteracja	6
3.2.1	Wykrywanie ewntualnego zderzenia	6
3.2.2	Propagowanie cząstek	7
3.2.3	Aktualizowanie prędkości cząstek	8
3.2.4	Wyznaczenie nowych przewidywanych położeń	8
3.3	Wizualizacja działania algorytmu	8
4	Opis aplikacji okienkowej	10
4.1	Sposób użytkowania	10
4.1.1	Przygotowanie środowiska	10
4.1.2	Obsługa aplikacji	11
4.2	Opis modułu	14

Wstęp

Poniższy raport opisuje działanie programu symulującego przepływ cząstek z przeszkodami.

Octree

2.1 Cel

Celem budowy drzewa *octree* jest znaczne skrócenie czasu znajdowania trójkątów znajdujących się dostatecznie blisko poszczególnych części w celu rozważenia ewentualnego odbicia cząstki od powierzchni (trójkąta) obiektu.

2.2 Opis działania

Ogólny przebieg budowy drzewa *octree* oraz następnie przeszukiwanie drzewa cząstkami wygląda następująco:

Na początku interesujący nas obiekt należy obudować jedną kostką (sześciannem), która to będzie stanowiła *świat* dla naszych rozważań i jednocześnie korzeń drzewa *octree*. Każdy wierzchołek drzewa będzie przechowywał trójkąty, które przecinają się z kostką reprezentującą dany wierzchołek, przy czym narzucamy ograniczenie na maksymalną liczbę trójkątów, które może przechowywać jeden wierzchołek.

Mając już pierwszą, globalną kostkę do drzewa dodajemy kolejno trójkąty. Trójkąty dodawane są do korzenia do momentu gdy wierzchołek nie będzie pełen trójkątów - wówczas kostka ta ulega podziałowi na 8 mniejszych kostek równej objętości - kostki te są *dziećmi* dla kostki, którą dzieliliśmy oraz ogólnie rzecz biorąc wierzchołkami. Następnie trójkąty z korzenia przenoszone są na dzieci. Przy czym tak naprawdę bierzemy pod uwagę jedynie te dzieci spośród 8, których kostki przecinają się z co najmniej jednym trójkątem. Taki zabieg służy temu, aby drzewo nie posiadało zbędnych, pustych wierzchołków, które nie odgrywają żadnej roli z punktu widzenia symulacji. Wierzchołki, które nie posiadają dzieci będziemy nazywać *liśćmi*.

Po opróżnieniu korzenia z trójkątów, następuje ponowne dodanie rozważanego trójkąta do drzewa. Wówczas następuje procedura dodania trójkąta do tych dzieci, których kostki przecinają się z tym trójkątem. Dodawanie to wykonywane jest na tej samej zasadzie co dodawanie do korzenia. Procedura ta jest kontynuowana dopóty, dopóki nie dodamy wszystkich trójkątów z siatki.

Uwaga: Na drzewo naniesione jest ograniczenie w postaci maksymalnej wysokości, aby uniknąć "nieskończonego" podziału wierzchołków. W związku z tym liście znajdujące się na najwyższym poziomie drzewa być może będą zawierały więcej trójkątów, niż jest to określone poprzez ograniczenie na maksymalną liczbę trójkątów w wierzchołku.

2.3 Kompilacja pakietu *Octree*

Procedura opisana wyżej obudowana jest pakietem *Octree*. Aby skompilować pakiet na swoim urządzeniu należy wykorzystać plik *setup.py* wpisując w wiersz poleceń/terminal następującą komendę:

```
python setup.py build_ext -inplace
```

Zbudowany zostanie wówczas pakiet *Octree* na podstawie pliku *Octree.pyx*.

2.4 Zawartość pakietu *Octree*

Pakiet składa się

Opis algorytmu propagacji cząstek

3.1 Opis ogólny

Celem algorytmu jest znalezienie wszystkich położenia cząstek poprzez uprzednio zadaną liczbę iteracji.

Dane wejściowe:

- Związane ze światem:
 - Wektor przyspieszenia grawitacyjnego g ,
 - Wartość współczynnika tarcia powierza C_D ,
 - Prędkość wiatru w ,
 - Gęstość powietrza ρ
- Związane z *octree*, wymienione w poprzedniej części raportu
- Związane z cząstkami:
 - Położenia początkowe x_0^p ,
 - Prędkości początkowe v_0^p ,
 - Masy cząstek m^p ,
 - Pola przekrojów poprzecznych cząstek S^p ,
- Związane z czasem:
 - Liczba kroków czasowych n_t ,
 - Długość kroku czasowego Δt .

Dane wyjściowe:

- Trójwymiarowa tablica położenia cząstek x_t^p ,
- Plik typu *.pkl* przechowujący rzeczoną tablicę.

Z danych początkowych wyznaczamy przewidywane położenia cząstek $\hat{x}_{\Delta t}^p$ w chwili Δt przy założeniu, że w przedziale czasowym $[0, \Delta t]$ będą się one poruszały ruchem prostoliniowym jednostajnym z prędkością v_0^p . Mamy zatem $\hat{x}_{\Delta t}^p := x_0^p + \Delta t v_0^p$. Algorytm iteruje opisaną w sekcji 3.2 procedurę n_t razy.

3.2 Jedna iteracja

Jedna iteracja algorytmu składa się z czterech głównych części:

1. Wykrywanie ewentualnego zderzenia,
2. Propagowanie cząstek (z ewentualnymi zderzeniami) przez dany krok czasowy Δt ,
3. Aktualizowanie prędkości cząstek,
4. Wyznaczenie nowych przewidywanych położań.

Poniżej opisujemy każdą z nich.

3.2.1 Wykrywanie ewntualnego zderzenia

Pierwszy etap algorytmu działa przy następujących założeniach:

- W danym odcinku czasowym $[t, t + \Delta t]$ ruch cząstki jest ruchem prostym, o ile nie dojdzie do zderzenia,
- Krok czasowy Δt jest wystarczająco mały, by poprzednie założenie nie powodowało problemów,
- Zderzenia są idealnie sprężyste,
- Dla każdej cząstki p przechowujemy informację o jej pozycji początkowej x_t^p w chwili t , prędkości chwilowej v^p oraz o jej przewidywanej pozycji $\hat{x}_{t+\Delta t}^p$ w chwili $t + \Delta t$ przy założeniu, że cząstka w czasie $[t, t + \Delta t]$ porusza się ruchem jednostajnym, prostoliniowym, z prędkością v^p . Innymi słowy, $\hat{x}_1^p = x_0^p + \Delta t v^p$.

Niech I_t^p będzie zbiorem indeksów trójkątów, które znajdują się w tym samym fragmencie świata w chwili t , co cząstka p . Podobnie, niech $I_{t+\Delta t}^p$ będzie zbiorem indeksów trójkątów, które znajdują się w tym samym fragmencie świata w chwili $t + \Delta t$, co cząstka p , przy założeniu, że poruszała się ruchem jednostajnym prostoliniowym. Oznaczmy $I_{t,\text{tot}}^p := I_t^p \cup I_{t+\Delta t}^p$.

Oznaczmy:

$$T[I_{t,\text{tot}}^p] := \left\{ \Delta_i \in T : i \in I_{t,\text{tot}}^p \right\}$$

Zakładamy zatem, że krok czasowy Δt jest wystarczająco mały, by cząstka p nie mogła przejść przez więcej niż jeden kawałek świata w czasie $[t, t + \Delta t]$.

Dla każdego trójkąta $\Delta_i \in T[I_{t,\text{tot}}^p]$ wyznaczamy teoretyczny czas zderzenia τ_i^p cząstki p z jego powierzchnią w następujący sposób.

1. Wyznaczamy początkową wysokość względną h_t^p cząstki p nad Δ_i przy pomocy wzoru

$$h_t^p := (x_t^p - v_{i,1}) \cdot n_i,$$

gdzie $v_{i,1}$ jest położeniem pierwszego wierzchołka Δ_i , zaś n_i unormowanym wektorem normalnym do powierzchni trójkąta Δ_i .

2. Wyznaczamy końcową wysokość względną $\hat{h}_{t+\Delta t}^p$ cząstki p nad Δ_i przy pomocy wzoru

$$\hat{h}_{t+\Delta t}^p := (\hat{x}_{t+\Delta t}^p - v_{i,1}) \cdot n_i.$$

3. Przy założeniu, że cząstka p porusza się ruchem prostoliniowym jednostajnym z prędkością v_t^p , zderzenie z płaszczyzną indukowaną przez trójkąt Δ_i zajdzie po czasie

$$\tau_i^p := -\frac{h_t^p}{\hat{h}_{t+\Delta t}^p - h_t^p} \Delta t$$

w punkcie:

$$\tilde{x}_{\tau_i^p}^p := x_t^p + \tau_i^p v_t^p.$$

4. Wyrażamy punkt (hipotetycznego) zderzenia \tilde{x}_τ^p we współrzędnych barycentrycznych względem trójkąta Δ_i . Innymi słowy, szukamy \tilde{y}_τ^p , które rozwiązuje poniższe równanie:

$$[v_{i,1}, v_{i,2} - v_{i,1}, v_{i,3} - v_{i,1}] \tilde{y}_{\tau_i}^p = \tilde{x}_{\tau_i}^p,$$

gdzie $v_{i,2}$ i $v_{i,3}$ są współrzędnymi, kolejno, drugiego i trzeciego wierzchołka trójkąta Δ_i .

Aby nie rozwiązywać w każdej iteracji powyższego równania liniowego, dla każdego trójkąta Δ_i przechowujemy informację o macierzy $B_i := [v_{i,1}, v_{i,2} - v_{i,1}, v_{i,3} - v_{i,1}]^{-1}$, dzięki czemu znalezienie $\tilde{y}_{\tau_i}^p$ odbywa się poprzez proste przy pomocy wzoru:

$$\tilde{y}_{\tau_i}^p = B_i \tilde{x}_{\tau_i}^p,$$

5. Zderzenie z trójkątem Δ_i może zajść (od strony zewnętrznej do powierzchni Δ_i (, wyznaczonej na podstawie n_i), przy następujących założeniach:

- $h_t^p > 0$, czyli cząstka p w chwili t była nad trójkątem Δ_i ,
- $\hat{h}_{t+\Delta t}^p < 0$, czyli przewidujemy, że cząstka po czasie Δt znajdzie się pod trójkątem Δ_i ,
- $\tilde{y}_{\tau_i}^p[2] \geq 0$, $\tilde{y}_{\tau_i}^p[3] \geq 0$ oraz $\tilde{y}_{\tau_i}^p[2] + \tilde{y}_{\tau_i}^p[3] \leq 1$, czyli zderzenie z płaszczyzną indukowaną przez Δ_i zachodzi w istocie wewnątrz trójkąta Δ_i , gdzie $\tilde{y}_{\tau_i}^p[2]$ i $\tilde{y}_{\tau_i}^p[3]$ są, kolejno, drugą i trzecią współrzędną miejsca zderzenia z płaszczyzną indukowaną przez Δ_i wyrażonymi we współrzędnych barycentrycznych.

Jeżeli spełnione są wszystkie powyższe warunki, to wiemy, że odcinek łączący punkty x_t^p i $\hat{x}_{t+\Delta t}^p$ przecina się z trójkątem Δ_i z odpowiedniej strony, a więc, pod warunkiem, że nie przeciął się wcześniej (tj. bliżej x_t^p) z odpowiedniej strony z innym trójkątem, to wówczas zderzenie nastąpi z trójkątem Δ_i .

Jeżeli którykolwiek z powyższych warunków nie jest spełniony, to wiemy, że zderzenie z trójkątem Δ_i zajść nie powinno, co pozwala nam na ustawienie $\tau_i^p := +\infty$.

Stosując powyżej opisany algorytm otrzymujemy listę czasów τ_i^p do teoretycznego zderzenia cząstki p z trójkątem Δ_i . Wybieramy z tych czasów najmniejszy z nich:

$$\tau^p := \min \left\{ \tau_i^p : i \in I_{t,\text{tot}}^p \right\}.$$

Jeżeli $\tau^p \neq +\infty$, to wiemy, że istnieje trójkąt Δ_i , z którym dojdzie do zderzenia. Zapamiętujemy indeks i_t^p tego trójkąta.

3.2.2 Propagowanie cząstek

Następnym krokiem jest propagowanie cząstek od czasu t do czasu $t + \Delta t$. Proces ten może zachodzić na dwa sposoby, zależnie od tego, czy cząstka p zderzyła się z jakimś trójkątem (, a więc, $\tau^p \neq +\infty$), czy nie.

- W pierwszym przypadku, gdy zaszło zderzenie (, a więc, $\tau^p \neq +\infty$), propagujemy cząstkę przez czas τ^p ruchem prostoliniowym jednostajnym z prędkością v_t^p aż do osiągnięcia zderzenia z trójkątem i_t^p .

Następnie symulujemy zderzenie idealnie sprężyste, otrzymując nową prędkość:

$$\tilde{v}_t^p := v_t^p + 2(v_t^p \cdot n_{i_t^p})n_{i_t^p}.$$

Finalnie, propagujemy cząstkę z nową prędkością ruchem jednostajnym prostoliniowym aż do końca rozważanego czasu.

- W drugim przypadku propagujemy cząstkę przez cały krok czasowy Δt z prędkością v_t^p ruchem jednostajnym prostoliniowym.

3.2.3 Aktualizowanie prędkości cząstek

Ostatnim krokiem każdej iteracji jest aktualizowanie prędkości cząstek. Rozważamy działanie dwóch sił:

- Siły grawitacyjnej $F_g = m^p g$, gdzie wartość wektora g oraz masy m^p cząstki p jest zadana od początku,
- Siły oporu powietrza $F_w = -C_D S^p \rho \frac{\|w^p\|}{2} w^p$, gdzie C_D , S^p , ρ , w^p to kolejno: współczynnik oporu, pole przekroju cząstki p , gęstość powietrza, w^p prędkość względna cząstki p względem wiatru.

Uzyskujemy z nich w chwili t przyspieszenie $a_t^p := \frac{1}{m^p}(F_g + F_w)$ i na jego podstawie obliczamy nowe prędkości:

$$v_{t+\Delta t}^p := v_t^p + \Delta t a_t^p.$$

3.2.4 Wyznaczenie nowych przewidywanych położeń

Finalnym etapem iteracji jest wyznaczenie nowych przewidywanych położeń cząstek $\hat{x}_{t+2\Delta t}^p$ zadanych wzorem:

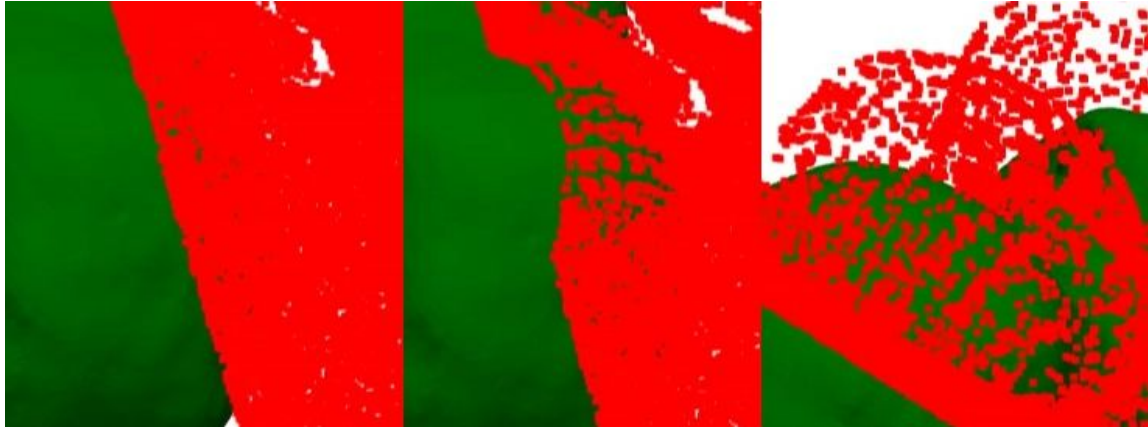
$$\hat{x}_{t+2\Delta t}^p := x_{t+\Delta t}^p + \Delta t v_{t+\Delta t}^p.$$

3.3 Wizualizacja działania algorytmu

Głównymi częściami algorytmu, które możemy zwizualizować są:

- Odbijanie się cząstek od powierzchni,
- Aktualizowanie prędkości cząstek

Poniższe zrzuty ekranu pochodzą z jednej z wizualizacji działania algorytmu w sytuacji, gdy przeszkodą jest królik.



Rysunek 3.1: Proces odbijania się cząstek; lewe zdjęcie: przed odbiciem, środkowe: niedługo po, prawe: na dłuższy czas po

Jak widać, zderzenia rejestrują się i kierunek odbicia jest uzależniony od wektora normalnego do powierzchni trójkątów, którymi przybliżamy powierzchnię królika. Z tego też powodu na moment po odbiciu możemy zauważyć powierzchnie poszczególnych trójkątów, które przyczyniają się do odrzucenia cząstek w sposób nieciągły.

Przejdźmy następnie do wizualizacji aktualizacji prędkości cząstek:



Rysunek 3.2: Proces aktualizacji prędkości cząstek: lewe zdjęcie: zaraz po odbiciu, środkowe: nieco dłużej po, prawe: na dłuższy czas po

Jak widać, cząstki po odbiciu opadają — stwierdza to występowanie siły grawitacyjnej, która sprawia, że cząstki niedługo po odbiciu od powierzchni królika do góry, zaczynają opadać. Siła oporu powietrza jest nieznaczna — jej wpływ jest niewielki, co wynika zarówno z niewielkiej prędkości wiatru, współczynnika tarcia oraz gęstości powietrza jak i tego, iż pola przekroju cząstek zostały przyjęte jako niewielkie.

Opis aplikacji okienkowej

4.1 Sposób użytkowania

4.1.1 Przygotowanie środowiska

Linux

W celu uruchomienia aplikacji należy zainstalować odpowiednie moduły, znajdujące się w pliku *python-requirements.txt*.

W tym celu można wykorzystać skrypt *makeenv.sh*, który utworzy wirtualne środowisko i zainstaluje wszystkie wymagane moduły. Aby uruchomić skrypt należy mieć zainstalowany instalator paczek pip oraz virtualenv. W tym celu można wykorzystać skrypt *install_venv_ubuntu.sh*.

Aby wykonać wyżej opisane czynności należy otworzyć terminal w katalogu projektu i wpisać

```
sudo bash install_venv_ubuntu.sh
```

```
sudo bash makeenv
```

Aby aktywować wirtualne środowisko należy wpisać w terminalu

```
source /venv/bin/activate
```

Teraz możemy uruchomić aplikację wpisując w terminal

```
python3 run.py
```

Aby dezaktywować wirtualne środowisko należy wpisać

```
deactivate
```

Windows

W przypadku windowsa również potrzeba zainstalować odpowiednie pakiety. W tym celu również można wykorzystać wirtualne środowiska. Żeby móc z nich skorzystać można zainstalować je np za pomocą skryptu

```
install_venv_windows.bat.
```

Następnie aby utworzyć virtualenva, należy otworzyć terminal i będąc w katalogu projektu wpisać

```
python3 -m venv venv
```

```
venv/scripts/activate.bat
```

```
python3 -m pip install -r python-requirements.txt
```

Teraz możemy uruchomić aplikację wpisując w terminal

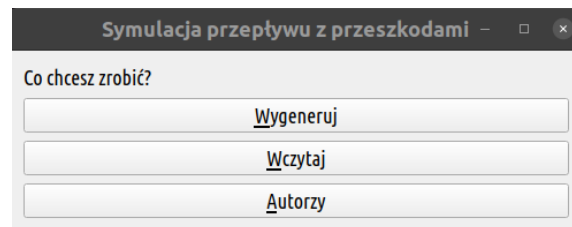
```
python3 run.py
```

Aby dezaktywować wirtualne środowisko należy wpisać

```
deactivate
```

4.1.2 Obsługa aplikacji

Po uruchomieniu aplikacji pojawi się następujące okno



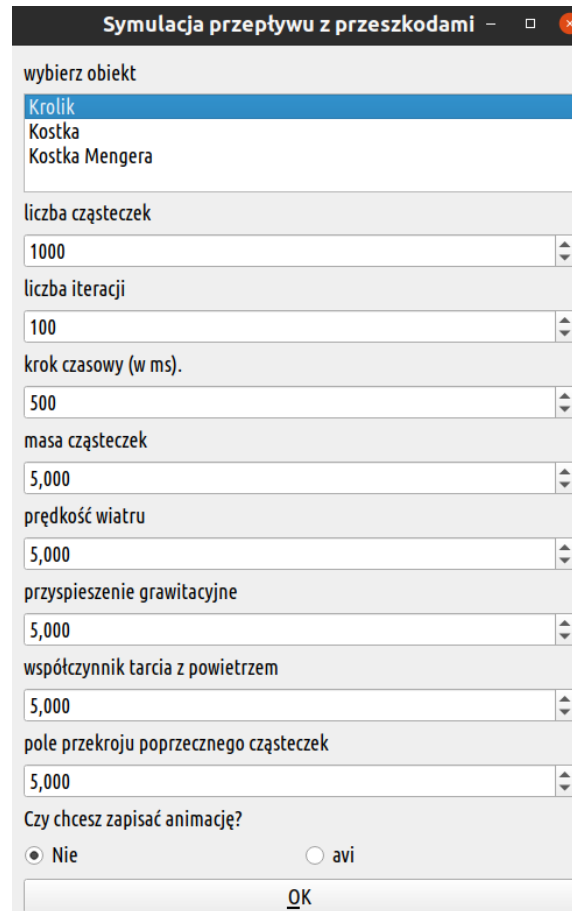
Rysunek 4.1: Okno startowe aplikacji

Mamy następujące możliwości

- **Wygeneruj** - możliwość wygenerowania nowej symulacji
- **Wczytaj** - możliwość wczytania pliku w rozszerzeniu .pkl z już obliczonymi trajektoriami cząsteczek
- **Autorzy** - wyświetlenie listy autorów projektu

Wygenerowanie nowej symulacji

Po wybraniu przycisku Wygeneruj pojawi nam się następujące okno



The dialog box titled "Symulacja przepływu z przeszkodami" contains the following elements:

- wyberz obiekt**: A list box with three items: "Krolik" (selected), "Kostka", and "Kostka Mengera".
- liczba cząsteczek**: A spin box with the value "1000".
- liczba iteracji**: A spin box with the value "100".
- krok czasowy (w ms).**: A spin box with the value "500".
- masa cząsteczek**: A spin box with the value "5,000".
- prędkość wiatru**: A spin box with the value "5,000".
- przyspieszenie grawitacyjne**: A spin box with the value "5,000".
- współczynnik tarcia z powietrzem**: A spin box with the value "5,000".
- pole przekroju poprzecznego cząsteczek**: A spin box with the value "5,000".
- Czy chcesz zapisać animację?**: Two radio buttons, "Nie" (selected) and "avi".
- OK**: A button at the bottom.

Rysunek 4.2: Okno do wygenerowania nowej symulacji

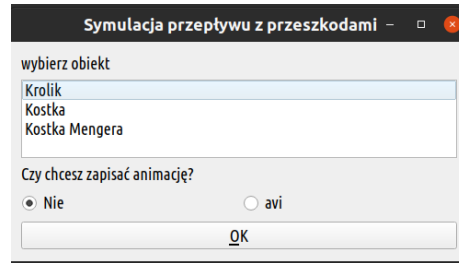
Możemy tutaj wykonać następujące czynności:

- wybrać obiekt spośród dostępnych
- podać liczbę cząsteczek od 1 do 1000000
- podać liczbę kroków czasowych, które chcemy wykonać (od 1 do 2000)
- podać długość kroku czasowego (w milisekundach)
- podać masę cząsteczek w (?)
- podać prędkość wiatru w (?)
- podać przyspieszenie grawitacyjne
- podać współczynnik tarcia cząsteczek z powietrzem
- podać pole przekroju poprzecznego cząsteczek
- wskazać czy chcemy zapisać symulację jako film w formacie .avi.

Po wybraniu odpowiednich opcji wybieramy przycisk OK

Wczytanie trajektorii cząsteczek z pliku

Po wybraniu przycisku Wczytaj pojawi nam się następujące okno



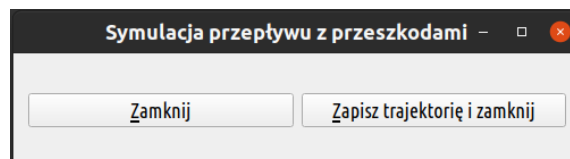
Musimy wybrać obiekt, dla którego zostały wygenerowane trajektorie cząsteczek, które chcemy wczytać. Możemy również wybrać opcję zapisu symulacji w postaci filmu w formacie .avi. Po wybraniu przycisku OK zostaniemy poproszeni o wybranie pliku w rozszerzeniu .pkl.

Animacja

Wówczas pojawi nam się okno programu PyVista z wybranym przez nas obiektem.



Za pomocą myszy możemy ustawić obiekt w taki sposób jak chcemy. Następnie wciskamy klawisz 'q' w celu rozpoczęcia animacji. Jeśli wybraliśmy opcję zapisania animacji, pojawi nam się okno w którym możemy wybrać lokalizację oraz nazwę zapisywanego pliku. Domyślna lokalizacja do zapisywania to katalog MovieSample) Po zakończeniu animacji pojawi się okno, w którym będzie można wybrać dwie opcje



- Zamknij - zakończ działanie aplikacji
- Zapisz trajektorie i zamknij - zapisz trajektorie cząsteczek do pliku w formacie .pkl. Domyślna ścieżka do zapisu to katalog PickleSample)

4.2 Opis modułu

Do wykonania graficznego GUI zostały wykorzystane następujące biblioteki:

- **PyQt5**
- **pyvista**

Kod interfejsu graficznego znajduje się w katalogu *Interfejs* w skrypcie *aplikacja_okienkowa.py*

Każde okienko aplikacji jest osobną klasą. W skrypcie występują następujące klasy:

1. **BaseWindow** - podstawowa klasa, z której dziedziczą pozostałe.

Metody:

- **__init__** - konstruktor
- **center** - metoda pozwalająca wyświetlać okno aplikacji w centrum aktywnie używanego ekranu

2. **StartWindow** - klasa odpowiadająca za okno startowe.

Metody:

- **__init__** - konstruktor
- **interfejs(self)** - metoda tworząca interfejs okna
- **switch_vizualize_window** - funkcja przekierowująca do okna z wyborem danych do wygenerowania symulacji
- **switch_ChooseLoadObject_window** - funkcja przekierowująca do okna z wyborem gotowego pliku z wyliczonymi trajektoriami do wczytania
- **switch_authors_window** - funkcja przekierowująca do okna z autorami

3. **AuthorsWindow** - klasa odpowiadająca za okno z autorami projektu.

Metody:

- **__init__** - konstruktor
- **interfejs** - metoda tworząca interfejs okna
- **switch_start_window** - funkcja przekierowująca z powrotem do okna startowego.

4. **ChooseLoadObjectWindow** - klasa odpowiadająca za okno do ładowania trajektorii z pliku.

Metody:

- **__init__** - konstruktor
- **interfejs** - metoda tworząca interfejs okna
- **Load** - metoda ładująca plik w formacie .pkl z trajektoriami cząsteczek do *numpy.ndarray*
- **choose_object** - metoda odpowiedzialna za wybór obiektu, na którym wskazana symulacja była przeprowadzana
- **btnstate** - funkcja pomocnicza do **choose_if_save**
- **choose_if_save** - metoda odpowiedzialna za umożliwienie wyboru czy chcemy zapisać daną symulację w formacie .avi
- **switch_result_window** - funkcja przekierowująca do okna z symulacją

5. **VizualizeWindow** - klasa odpowiadająca za okno do wpisania danych do wygenerowania symulacji.

Metody:

- **__init__** - konstruktor
- **interfejs** - metoda tworząca interfejs okna
- **choose_object** - metoda odpowiedzialna za wybór obiektu, na którym wskazana symulacja była przeprowadzana
- **btnstate** - funkcja pomocnicza do **choose_if_save**
- **choose_if_save** - metoda odpowiedzialna za umożliwienie wyboru czy chcemy zapisać daną symulację w formacie .avi
- **set_QSpinBox** - metoda pomocnicza umożliwiająca wpisywanie przez użytkownika danych całkowitoliczbowych
- **set_QDoubleSpinBox** - metoda pomocnicza umożliwiająca wpisywanie przez użytkownika danych zmiennoprzecinkowych
- **Calculations** - metoda wywołująca algorytm tworzący trajektorię na podstawie danych wprowadzonych przez użytkownika.
- **switch_result_window** - funkcja przekierowująca do okna z symulacją. Podczas przekierowania uruchamiana jest funkcja **Calculations**.

6. **ResultWindow** - klasa odpowiadająca za okno z symulacją.

Metody:

- **__init__** - konstruktor
- **interfejs** - metoda tworząca interfejs okna
- **single_animation** - metoda odpowiedzialna za wykonanie animacji w jednym kroku czasowym
- **animate** - metoda odpowiadająca za całkowitą animację. Wywoływana jest w niej metoda **single_animation**
- **save** - metoda pozwalająca zapisać wygenerowaną trajektorię do pliku w formacie .pkl
- **close_window** - funkcja zamykająca aplikację
- **save_and_close_window** - funkcja wywołująca metodę **save** a następnie zamykająca aplikację