Kurs: Obliczenia równoległe na kartach graficznych CUDA

Plan projektu

Michał Maras

Cel projektu

Celem projektu jest zrobienie symulatora fizyki brył sztywnych wraz z wizualizacją – zaprogramowanie rozdziału 29. z książki Nvidii GPU GEMS 3.
Symulacja bazuje na reprezentacji bryły sztywnej jako obiektu składającego się z cząsteczek (małych kul), na podstawie których wyliczane są kolizje.

Projekt korzysta z biblioteki CUDA, Thurst (dostępne razem z biblioteką CUDA) oraz OpenGL do wizualizacji.

Instalacja i uruchomienie

Do projektu dołączony jest plik Makefile, który kompiluje program i generuje plik wykonywalny *particle_simulation*. W folderze *src* znajduję się kod źródłowy programu. W folderze *resources* znajdują się pliki konfiguracyjne, .obj z modelami obiektów oraz shadery.

Program uruchamiany jest w następujący sposób:
`./particle_simulation <sciezka_do_pliku_z_modelem:string> <sciezka_do_pliku_konfiguracyjnego:string>`

Dodatkowo, można użyć opcji `--obj_viewer`, aby uruchomić obiekt w przeglądarce obiektów. Za pomocą przycisku `TAB` można zmienić sposób wyświetlania modelu na jego reprezentację w postaci cząsteczek.

Opcja `--particles_only` prezentuje demonstracyjną symulację pojedynczych cząsteczek.

Do projektu dołączone są trzy przykładowe konfiguracje obiektów w folderze *resources/start_configs*:

- `single_object.cfg` – pojedynczy obiekt
- `three_obj.cfg` – trzy spadające na siebie obiekty
- `multiple.cfg` – 100 losowo rozmieszczonych obiektów

Zaawansowana konfiguracja

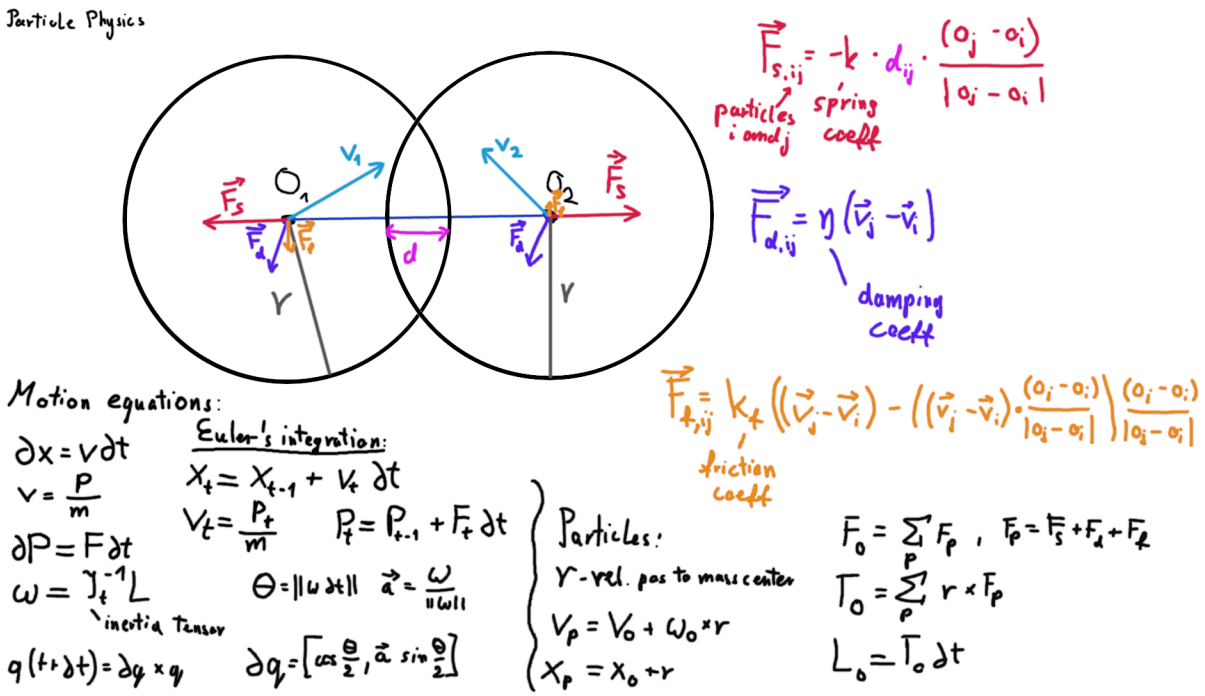
Początkowe konfiguracje są bardzo proste do modyfikacji. Wystarczy stworzyć nowy plik, w którego pierwszej linii znajdzie się liczba obiektów, a w następnych liniach będą po trzy liczby rzeczywiste reprezentujące położenie środka ciężkości kolejnych obiektów.

Więcej opcji symulacji jest dostępne w pliku `src/constants.hpp`, ich zmiana wymaga jednak ponownej kompilacji projektu. Najważniejszymi rzeczami do zmiany są:

- stała `FPS` – limit klatek na sekundę
- ciąg znaków `SHADER_DIR` – ścieżka do folderu z shaderami
- stała `MAX_DEPTH_PEELING` – liczba iteracji algorytmu *depth peeling* do generowania cząsteczek z modelu (domyślnie ustawiona wartość 20 powinna być wystarczająca dla większości obiektów)
- `PARTICLE_RADIUS` – promień cząsteczki, im mniejszy, tym więcej cząsteczek będzie wygenerowanych dla modelu i tym więcej pamięci będzie potrzebne, raczej lepiej nie zmniejszać tej wartości jeszcze bardziej
- `PARTICLE_RADIUS_SIM` – promień cząsteczki używany podczas obliczania fizyki, powinien być odrobinę mniejszy niż `PARTICLE_RADIUS` (np. o 0.0001), zapobiega błędom numerycznym
- `PARTICLE_MASS` – waga cząsteczki, kontrybuuje do całkowitej masy obiektu.
- `TIME_SPEED` – spowolnienie lub przyspieszenie czasu
- `OBJECT_SCALE` – rozmiar obiektu; UWAGA: dla zbyt małych obiektów może generować się zbyt mało cząsteczek!
- `MAX_FORCE` – limit na rozmiar wektora sił, zapobiega błędom numerycznym
- stałe `SPRING_COEFF`, `DAMPING_COEFF` i `FRICTION_COEFF` – kolejno współczynnik sprężystości, współczynnik tłumienia oraz współczynnik tarcia cząsteczki. Ostatni powinien być liczbą mniejszą od 1; należy je dobierać eksperymentalnie, mają znaczny wpływ na symulację; trzeba dostosowywać je przy większości wprowadzanych zmian
- `BORDER_SPRING_COEFF`, `BORDER_DAMPING_COEFF` oraz `BORDER_FRICTION_COEFF` – podobnie co wyżej, ale dla ścian
- `UNIFORM_GRID` lub `BRUTE_FORCE` – wybrać jedno, definiuje sposób obliczania kolizji
- `GRID_SIDE_LEN` – szerokość woksela siatki kolizji, powinna wynosić około średnicy cząsteczki
- `GRID_MAX_X`, `GRID_MAX_Y`, `GRID_MAX_Z` – maksymalny obszar, na którym obliczane są kolizje

Część 1: Symulacja cząsteczek

Symulator fizyki brył sztywnych bazuje na symulacji cząsteczek. Cząsteczka jest małą kulą, na którą oddziałują siły sprężystości oraz grawitacji. Wszystkie obiekty składają się z wielu cząsteczek, na podstawie których obliczane są kolizje.



Detekcja kolizji

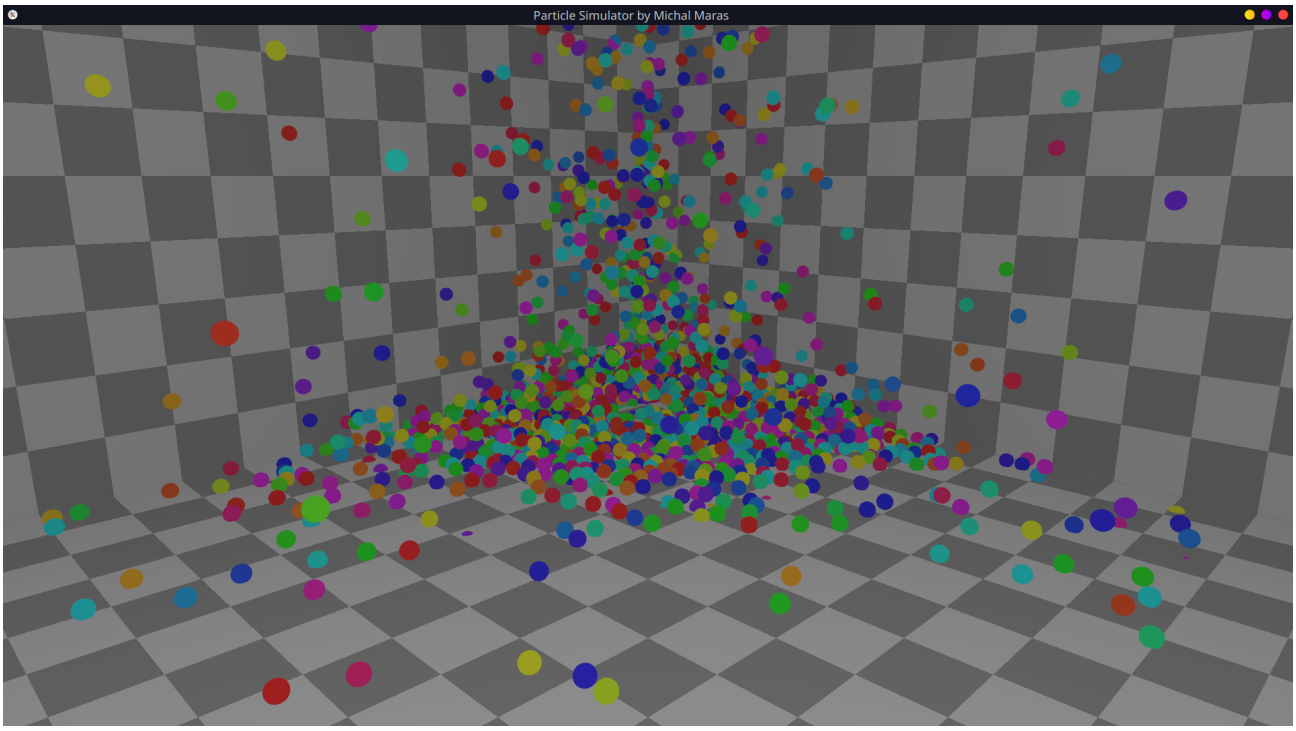
Sprawdzanie kolizji pomiędzy każdą parą cząsteczek jest bardzo nieefektywne (tryb `BRUTE_FORCE`). Zamiast tego użyta jest technika wstępnie opisana w książce Nvidii GPU GEMS 3 oraz trochę dokładniej w poradniku odnoszącym się do symulacji cząsteczek. Metoda polegaja na podziale przestrzeni na sześcienną siatkę. Cząsteczki są przyporządkowywane do pewnego sześcianu tej siatki. Gdy bok sześcianu zostanie odpowiednio ustalony (około średnicy cząsteczki), do sprawdzenia kolizji należy wziąć pod uwagę tylko cząsteczki z sąsiednich sześcianów (dla boku długości średnicy cząsteczki około 4 cząsteczki na sześcian, 27 sześcianów do sprawdzenia, łącznie maksymalnie 108 sprawdzeń).

Algorytm wygląda następująco:

- obliczamy tablicę par {cząsteczka, indeks sześcianu},
- sortujemy wygenerowaną tablicę za pomocą algorytmu radix sort (GPU),
- dla każdego sześcianu w siatce obliczamy cząsteczki, które się w nim znajdują,
- dla każdej cząsteczki sprawdzamy sąsiadujące sześciany i wykrywamy kolizje pomiędzy stykającymi się cząstkami.

Taka implementacja umożliwia złożoność czasową $O(N)$, dla N będącego liczbą cząsteczek, oraz pamięciową $O(N+M)$, gdzie M to liczba sześcianów siatki.

Efekt:



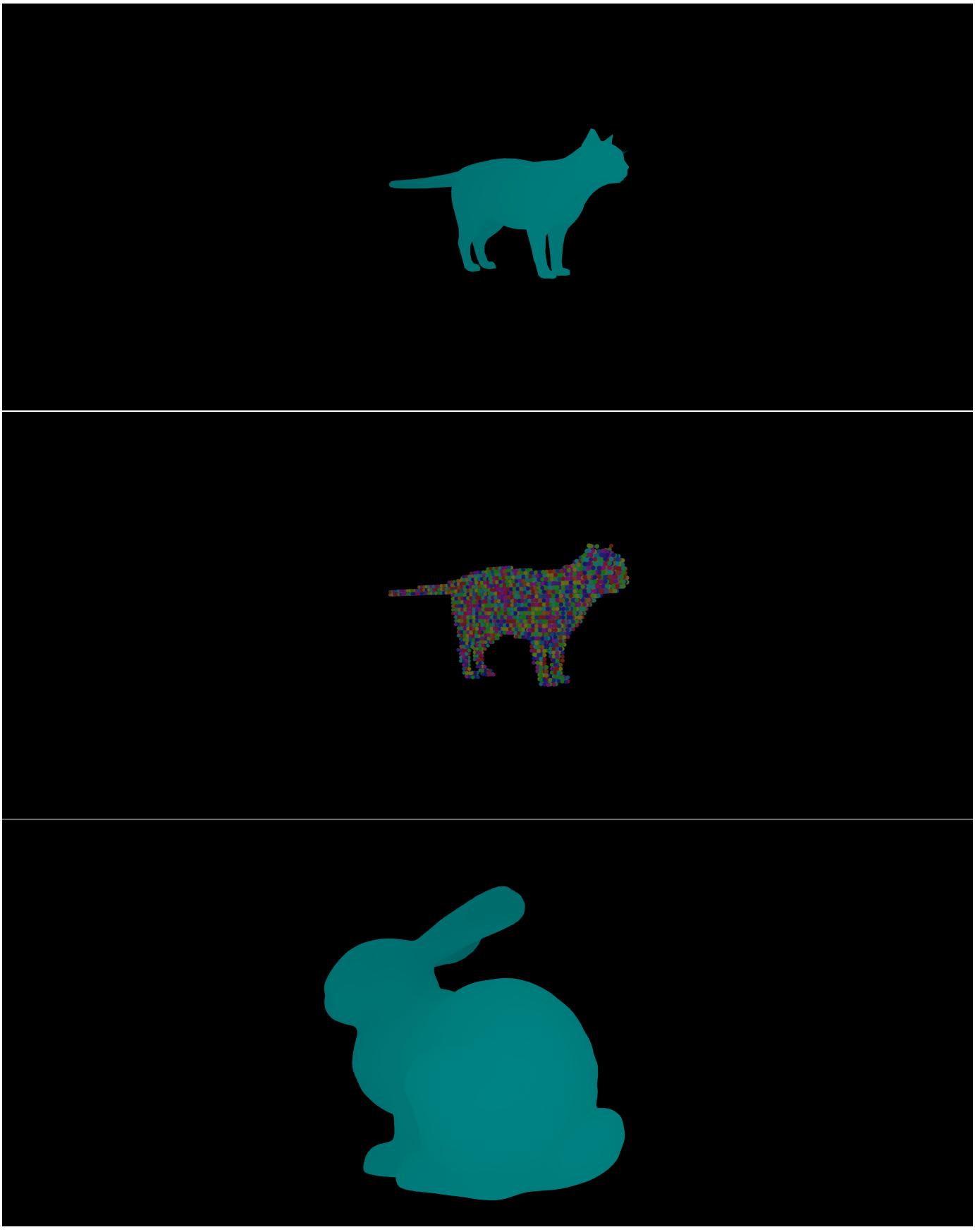
Część 2: Zamiana trójwymiarowego obiektu w cząsteczki

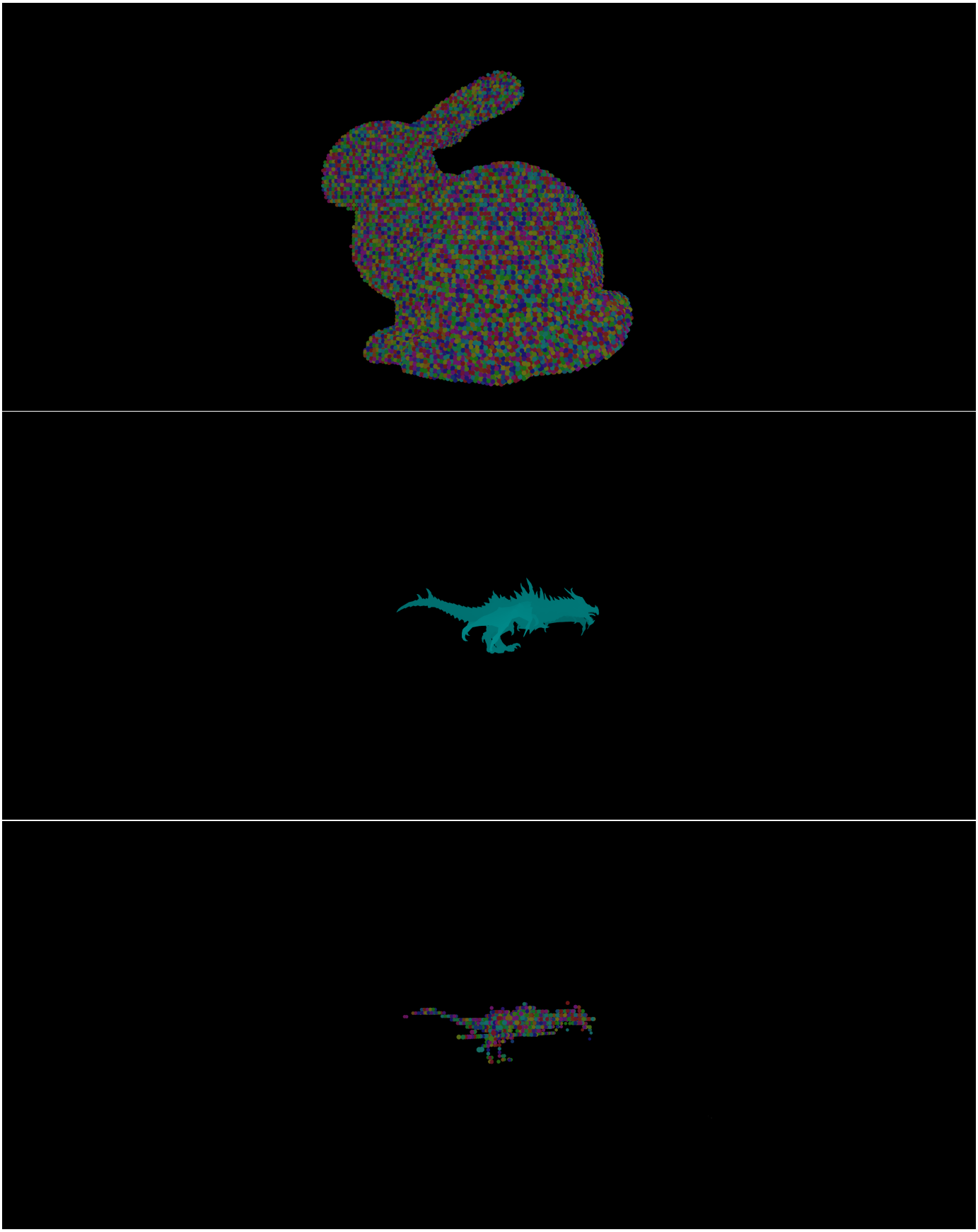
Dla poprawnego działania symulacji konieczne jest wygenerowanie dla wczytanego modelu `.obj` zawartych w nim cząsteczek. Najprostszą metodą jest użycie *ray castingu* do testu, czy dany punkt znajduje się wewnątrz modelu, jednak jest to bardzo nieoptymalne dla dużej liczby testowanych punktów. Można to przyspieszyć za pomocą techniki *depth peeling*, często używanej do renderowania przezroczystych obiektów na scenie.

Algorytm wykorzystuje bufor głębokości otrzymywany podczas renderowania sceny z obiektem:

1. Ustawiamy model w punkcie $(0,0,0)$ i skalujemy go, aby był całkowicie zawarty w sześcianie o szerokości 1. Ustawiamy kamerę z rzutem ortograficznym prostopadle do modelu.
2. Renderujemy obraz, zapisując bufor głębokości do tekstury.
3. Renderujemy obraz ponownie, ale we *fragment shaderze* odrzucamy wszystkie fragmenty, które mają głębokość mniejszą lub równą tej zapisanej na ostatniej teksturze (mniejsza głębokość – bliżej kamery). Również zapisujemy bufor do kolejnej tekstury i powtarzamy, aż wszystkie fragmenty zostaną odrzucone. Dla wygodniejszej implementacji w programie zawsze wykonywane jest `MAX_DEPTH_PEELING` iteracji.
4. Uzyskane tekstury z głębokościami posłużą nam do przyspieszenia *ray castingu*. Aby sprawdzić, czy dany punkt znajduje się w środku modelu, sprawdzamy, na ilu teksturach na pozycji tego punktu zostały wyrenderowane fragmenty o mniejszej głębokości. Jeśli liczba ta jest parzysta – punkt leży poza obiektem, w przeciwnym przypadku znajduje się w środku. Wykonujemy powyższy test dla punktów będących potencjalnymi pozycjami cząsteczek. Jeśli test zakończył się pozytywnie, generujemy w pozycji punktu cząstkę.

Efekt:





Niektóre z powyższych modeli nie są dołączone do projektu ze względu na prawa autorskie.

Część 3: Symulacja brył sztywnych

Łącząc ze sobią Części 1 i 2 otrzymujemy symulację brył sztywnych. Dla każdej z cząsteczek, na które zamienione zostały modele, obliczamy kolizje i równania niezależnie, a na podstawie wyliczonych wartości obliczamy translację i rotację brył. Część z potrzebnych równań znajduje się na pierwszym rysunku z równaniami fizyki cząstek.

Efekt: