Wykorzystanie głębokich sieci neuronowych do symulacji powierzchni wody w otoczeniu przeszkody przy użyciu silnika Unreal Engine 4

Paweł Kowalski

10.11.2023

Rozdział 1

Opis koncepcji rozwiązania problemu

Problemem postawionym w ramach pracy jest stworzenie interaktywnej symulacji powierzchni wody. Istotne jest przy tym spełnienie specyficznych dla zagadnienia symulacji na potrzeby gier wymagań:

- wysoka wydajność, pozwalająca na osiągnięcie zadowalającej liczbie klatek na sekundę. Za minimalną wartość, którą uznać można za zadowalającą, przyjąć można 30 klatek na sekundę. Oznacza to, że czas obliczania symulacji, transferu danych oraz wszelkich innych operacji niezbędnych przed wygenerowaniem każdej klatki nie może przekroczyć 33,(3) ms,
- możliwość osiągnięcia zadanej stylistyki powierzchni (dyrekcji artystycznej), kosztem fizycznej poprawności symulacji,
- możliwość interakcji z powierzchnią wody.

Założeniem projektu jest wykorzystanie sieci neuronowych. W trakcie prac przeanalizowano kilka ich typów. Zdecydowano, że sieć generować będzie dane w postaci dwuwymiarowej macierzy reprezentującej wysokość lustra wody. Danymi wejściowymi dla sieci mają być informacje o interakcji z powierzchnią płynu oraz wysokości lustra wody w momencie następującym przed symulowanym. Jeśli generowana jest klatka inna niż pierwsza, wysokość wody używana jako dane wejściowe wygenerowana została wcześniej przez tę samą sieć dla klatki poprzedniej.

W celu wyszkolenia sieci zdecydowano się na przygotowanie symulacji płynu z wykorzystaniem narzędzi dostępnych w aplikacji SideFX Houdini. Aplikację wybrano z uwagi na łatwość implementacji procesu generowania danych oraz dostępny w niej silnik fizyki cieczy FLIP. Zdecydowano, że symulacja polegać będzie na interakcji poruszającego się obiektu z powierzchnią wody, w której obiekt został częściowo zanurzony. Przyjęto próbkowanie symulacji równe trzydziestu próbkom na sekundę. Odpowiada to przyjętym w założeniach trzydziestu klatkom symulacji na sekundę.

Wygenerowane przykłady zawierają informacje analogiczne do tych, które dostarcza sieci silnik gry podczas działania interaktywnej aplikacji. Są to:

- informacja o kształcie przeszkody zapisana w formie Signed Distance Field (SDF),
- informacja o kierunku poruszania się obiektu,
- informacja o prędkości poruszania się obiektu,
- aktualna wysokość lustra wody,

- aktualne rozłożenie piany na powierzchni wody.

Wszystkie informacje przetwarzane są przed podaniem ich sieci w taki sposób, by miały formę dwuwymiarowych macierzy. Pozwala to na wykorzystanie sieci splotowych. Sieć oparto na modyfikacji architektury U-Net [5]. Jako część funkcji strat wykorzystano sieć pełniącą rolę dyskryminatora w sposób podobny do tego, jaki ma miejsce w przypadku sieci GAN.

Przygotowano interaktywną prezentację zawierającą symulację. Symulacja odpowiada tej, która wykonana została w celu generacji próbek w programie SideFX Houdini, jednak wykorzystującą wytrenowaną sieć jako źródło informacji o wysokości lustra.

Rozdział 2

Przeprowadzone prace

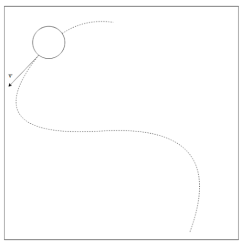
W rozdziale tym opisano prace mające na celu stworzenie interaktywnej symulacji wody przy wykorzystaniu sieci neuronowych.

Prace podzielono na dwa etapy. Zdecydowano, że w pierwszej kolejności rozważona zostanie bardzo prosta forma symulacji. Za pomocą tej uproszczonej symulacji wygenerowano zbiór danych, którego następnie użyto w celu wytrenowania prostej sieci neuronowej. Pozwoliło to na podjęcie decyzji dotyczących szczegółów symulacji, rozpoznanie potencjalnych problemów i ograniczeń, potwierdzenie wykonywalności projektu oraz przygotowanie i przetestowanie niezbędnych narzędzi. Kolejnym etapem było rozbudowanie symulacji.

2.1 Prosta symulacja powierzchni wody

2.1.1 Przygotowanie symulacji powierzchni wody

Pierwszy badany scenariusz polegał na symulacji powierzchni wody na obszarze o nieruchomych granicach. W ramach tego obszaru poruszała się cylindryczna przeszkoda. Zdecydowano się na pominięcie odbić fali od granic obszaru. Schematyczne przedstawienie symulacji znajduje się na rys.1.



Rysunek 1. Schematyczne przedstawienie prostej symulacji. Źródło: opracowanie własne.

Symulację wykonano za pomocą aplikacji SideFX Houdini. Wykorzystano w tym celu narzędzie Flat Tank [6] dostępne w aplikacji. Narzędzie wykorzystuje algorytm FLIP (Fluid-Implicit-Particle, oparty na metodzie Lagrange'a) i pozwala na symulację cieczy na przestrzeni ograniczonej

przez podane wartości. Narzędzie służy do symulacji ograniczonego obszaru wody będącego częścią większego zbiornika. Oznacza to, że nie wymaga definiowania ścian bocznych ani dna, automatycznie utrzymywany jest stały poziom cieczy, natomiast podczas przesuwania symulowanego obszaru cząsteczki reprezentujące ciecz znikają lub pojawiają się automatycznie na jego granicach. Alternatywnym rozwiązaniem mogłoby być zdefiniowanie zbiornika wodnego i symulacja płynu w całej jego objętości, jednak wiązałoby się to ze znacznym zwiększeniem złożoności obliczeniowej symulacji oraz wprowadziło zjawisko odbicia fal od ścian pojemnika, co w rozważanym przykładzie jest niepożądanym zjawiskiem.

Animację przemieszczenia przeszkody wykonano wykorzystując skrypt napisany w języku Python generujący krzywą, oraz dostępne w aplikacji narzędzie przesuwające obiekt wzdłuż ścieżki w miarę postępu animacji. Skrypt opierał się na wykorzystaniu generatora liczb losowych, co pozwalało na uzyskanie różnych przebiegów krzywej, w efekcie różnych animacji ruchu przeszkody. Za pomocą wielokrotnego wykonywania symulacji oraz empirycznej oceny jej wyników wyznaczono odpowiednie parametry skryptu, aby prędkość poruszania się obiektu zamykała się w zakresie generującym pożądany rodzaj fal.

W celu przyspieszenia procesu obliczania symulacji zdecydowano się na wykorzystanie dostępnej w aplikacji możliwości przeniesienia części obliczeń na GPU przy użyciu OpenCL.

2.1.2 Generacja zbioru danych

Zbiór danych postanowiono reprezentować za pomocą kilku tablic elementów. W każdej z tych tablic zapisane były odpowiednie wartości z odczytanej symulacji dla postępujących po sobie klatek. Zbierano następujące dane:

- dwuwymiarowa macierz reprezentująca wysokość tafli wody w danym punkcie wyznaczanym przez współrzędne X oraz Y (rys. 2) reprezentująca efekt końcowy rozpatrywanej klatki,
- trzy wartości typu float reprezentujące położenie przeszkody w klatce o indeksie N.

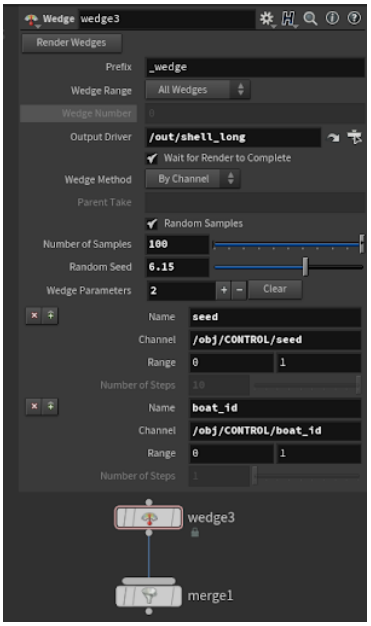
Rysunek 2. Wizualizacja macierzy reprezentującej wysokość tafli wody. Źródło: opracowanie własne.

W celu uzyskania stanu symulacji w klatce o indeksie N należało wybrać element o indeksie N spośród każdej z wygenerowanych tablic. Do określenia pozycji przeszkody wystarczające były

dwie wartości, przekazywanie położenia przeszkody w osi prostopadłej do powierzchni płaskiej wody nie było konieczne, jego wartości zawsze wynosiły zero. Planowana w związku z tym była optymalizacja polegająca na zmianie reprezentacji położenia na tablicę dwuwymiarową. Zdecydowano się na wykonanie symulacji dla wielu wariantów ścieżki oraz ustalonej długości 1000 klatek każdej z nich.

Symulacja płynu jest złożonym obliczeniowo zadaniem, jednak przy wybranym scenariuszu nie wykorzystywała pełni mocy komputera. Na dodatek proces symulacji składa się z wielu etapów, podczas których występuje różne zapotrzebowanie na pamięć RAM, czy moc obliczeniową CPU. W związku z tym planowano równoległe wykonywanie więcej niż jednej symulacji, w celu pełnego wykorzystania mocy obliczeniowej komputera. Można to osiągnąć, uruchamiając więcej niż jedną instancję aplikacji, co autor z powodzeniem stosował w przeszłości w innych projektach, zdecydowano się jednak na wykorzystanie dostępnego w aplikacji SideFX Houdini narzędzia Procedural Dependency Graph (PDG). Narzędzie to służy automatyzacji aplikacji Houdini, w szczególności automatyzacji wykonywania zadań przy możliwości wykonywania ich równolegle. Pozwala na podział pracy na wykonywane asynchronicznie etapy, podzielone dalej na pojedyncze zadania, przy zachowaniu wymaganych zależności między nimi. Oznacza to, że wiele różnych zadań, jak na przykład symulacja wody przy dwóch różnych przebiegach ścieżki, może odbywać się równolegle, jednak jeśli zadania są od siebie zależne, jak symulacja w klatce N+1 animacji zależna od symulacji obliczonej w klatce N, zadania te wykonywane będą w odpowiedniej kolejności. Kolejną zaletą narzędzia PDG jest możliwość wykonania zadań dla różnych danych wejściowych, w tym generowanych za pomocą generatora liczb losowych, co planowano wykorzystać w celu generowania ścieżek. Przeprowadzone eksperymenty pokazały jednak, że nie jest możliwe przeprowadzanie więcej niż jednej symulacji płynu w tym samym czasie.

W związku z brakiem możliwości wykorzystania narzędzia PDG zdecydowano się na uruchomienie pojedynczej instancji aplikacji oraz wykorzystanie narzędzia wedge, uruchamiającego wybrane zadanie dla różnych wartości zadanego parametru w ramach jednego wątku. Symulację przeprowadzano więc dla różnych wartości zmiennej wykorzystywanej następnie przez skrypt generujący ścieżkę, po której porusza się przeszkoda, jako zarodek generatora zmiennych liczb losowych. Przykład grafu wykorzystującego narzędzie wedge przedstawia rys. 3. Rezultat symulacji przetworzono oraz zapisano przy pomocy grafu dostępnego w repozytorium projektu.



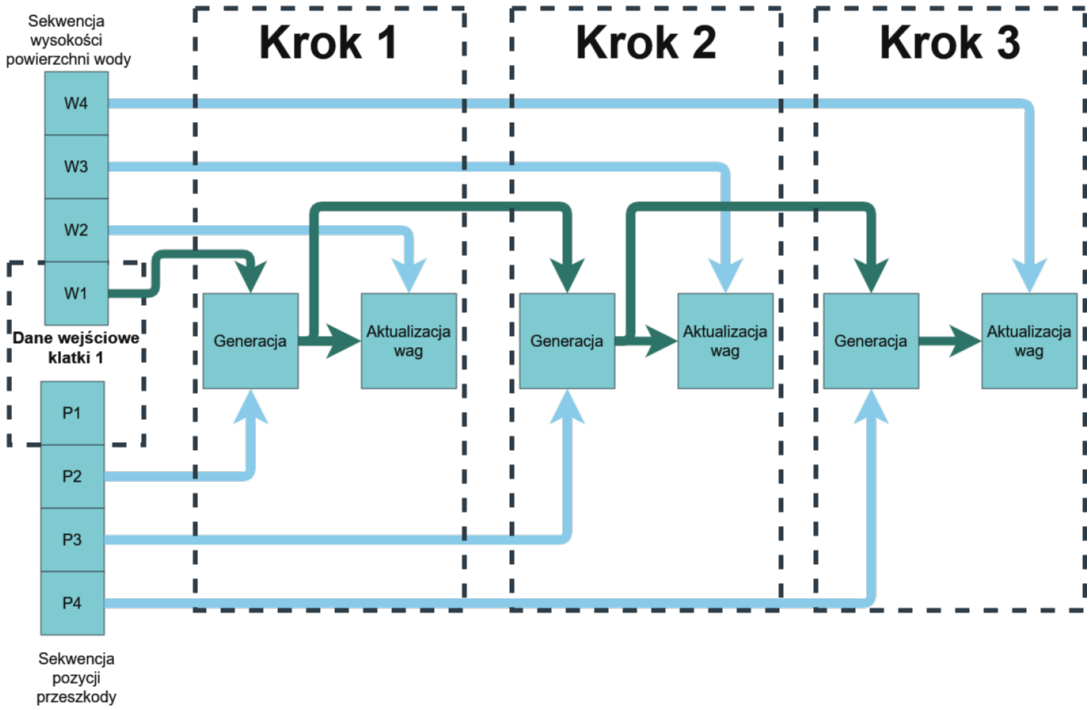
Rysunek 3. Przykład grafu wykorzystującego narzędzie wedge. Źródło: opracowanie własne.

Eksperymenty wykazały, że ma miejsce zbyt wielkie zużycie pamięci przez tablice zawierające dane wejściowe oraz rezultaty symulacji. W efekcie po przekroczeniu liczby kilkuset klatek symulacji czas zapisywania danych dla kolejnej klatki stawał się znaczącym obciążeniem. Zdecydowano się więc na czyszczenie tych tablic i zapisywanie nowych plików zawierających je co 100 klatek symulacji.

Wygenerowane dane należało przekształcić w formę, w której możliwe było ich wykorzystanie przez sieć neuronową. Należało przygotować kilka rodzajów reprezentacji i zbadać, która z nich rokuje najlepiej. W celu wykonania tego zadania zdecydowano się na użycie Jupyter Notebook. Dzięki temu możliwa była wizualizacja i sprawdzenie poszczególnych etapów ich przygotowania, co okazało się istotne, ponieważ proces ten okazał się być bardzo złożonym. W celu przygotowania potoków wejściowych dla sieci neuronowej wykorzystano Tensorflow Dataset.

Zdecydowano się zbadać wiele wariantów sieci neuronowej przy różnych reprezentacjach danych wejściowych w celu odnalezienia kombinacji dających najlepsze rezultaty.W tym celu przygotowano funkcje tworzące różne modele sieci, następnie uruchomiono ich trenowanie na 15 epok, po czym porównano wartość błędu średniokwadratowego oraz, empirycznie, generowane przez nie rezultaty.

Zdecydowano, że dane te wykorzystywane będą w formie sekwencji o zadanej długości danych wejściowych i wyjściowych następujących po sobie klatek, w sposób podobny do tego, który zaleca dokumentacja biblioteki Tensorflow [7]. Przygotowano klasę CustomModel opartą na klasie keras.Model. Dokonano w niej nadpisania metody train_step w taki sposób, aby odpowiadała założonemu mechanizmowi uczenia sieci. natomiast schematyczne przedstawienie zasady trenowania sieci zawiera rys. 4.



Rysunek 4. Graf wykorzystany w celu przetwarzania i zapisania danych opisujących powierzchnię symulowanej wody. Źródło: opracowanie własne.

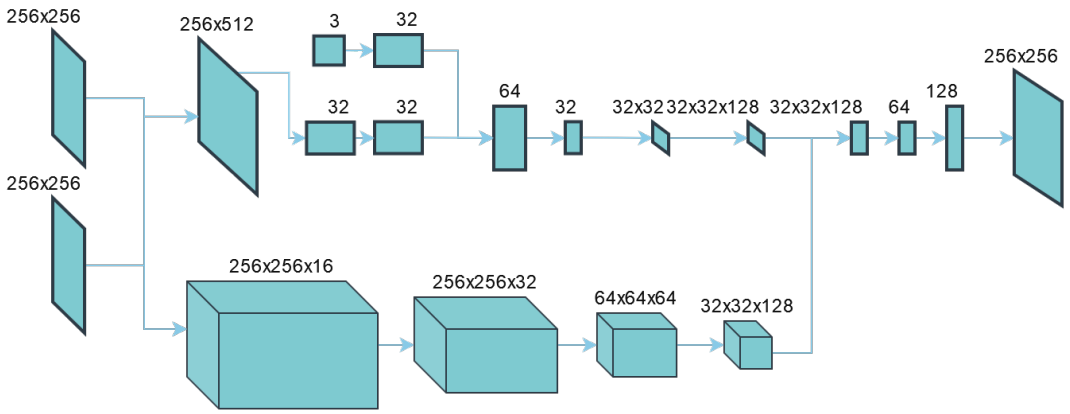
W projekcie wykorzystano osobne potoki Tensorflow Dataset dla obu rodzajów danych wejściowych. Potoki te następnie połączono przy użyciu funkcji Dataset.zip, po czym wykorzystano operację shuffle w celu randomizacji danych przy każdej epoce. Dane wygenerowane w procesie animacji przechowywane były w plikach .np zawierających tablice Numpy o długości 100 elementów, z których każdy reprezentował odpowiednią wartość odczytaną w klatce odpowiadającej jej indeksowi.

Pierwszym krokiem w części kodu odpowiedzialnej za przetwarzanie danych było odnalezienie wszystkich plików w zadanej lokacji na dysku zawierających wygenerowane dane. Dokonano tego poprzez parsowanie ich nazw. Przefiltrowana lista nazw plików zamieniana była w potok danych. Na powstałym potoku uruchamiana była funkcja load_file za pomocą metody map. Metoda map wykonuje akcję splotu: uruchamia zadaną funkcję dla każdego elementu potoku, tworząc nowy element potoku będący tablicą rezultatów działania wspomnianej funkcji. Zadana funkcja otwierała plik określony przez będącą elementem potoku nazwę, następnie zwracała go, wskutek czego elementami potoku stały się stu elementowe tablice wartości wygenerowanych podczas symulacji. Następnie wykorzystano metodę window potoku, w celu utworzenia krótszych sekwencji. Za pomocą metody flat_map usunięto jeden z wymiarów potoku, zamieniając potok zawierający w każdym elemencie tabelę takich sekwencji w potok zawierający sekwencję.

Przekazanie wysokości tafli wody w klatce poprzedzającej generowaną jako macierzy dwuwymiarowej było oczywistym wyborem, jednak format, w jakim reprezentowane być powinny dane dotyczące położenia przeszkody oraz jej przesunięcia między klatkami nie była oczywista. Zdecydowano się na sprawdzenie kilku form reprezentacji oraz kilku architektur sieci.

Początkowo zdecydowano się na przekazanie jedynie informacji o pozycji przeszkody w rozważanej klatce o indeksie N oraz wysokości powierzchni wody w klatce ją poprzedzającej o indeksie N - 1. W przypadku sieci rekurencyjnej przekazano również wysokość powierzchni wody w klatce N-2. Choć informacja o zmianie pozycji lub informacja o pozycji w klatce N-1 może wydawać się niezbędną, zdecydowano, że podczas tych pierwszych eksperymentów nie będzie ona przekazana. Spekulowano, że być może sieć jest w stanie uzyskać tę informację na podstawie porównania położenia czoła fali, będącego w istocie w określonej odległości równej promieniowi cylindra od środka przeszkody, do uzyskanych danych o jej położeniu w rozważanej klatce. Zarazem woda znajdująca się bliżej środka przeszkody niż jej promień musi wpłynąć bezpośrednio na podniesienie się czoła fali. Kolejnym argumentem było to, że zadowalającym rezultatem działania sieci na tym etapie będzie uzyskanie przez nią zdolności do przedłużenia ruchu już istniejących fali. Zdecydowano więc, że uwzględnienie dodatkowych informacji na wejściu do sieci będzie miało miejsce w kolejnych etapach pracy, jeśli wyniki okażą się obiecujące.

Przygotowano wiele wariantów splotowych sieci neuronowych, które na wejściu przyjmowały informację o położeniu przeszkody w klatce o indeksie N oraz wysokości powierzchni wody w klatce N-1. Zbadanych zostało również kilka wersji rekurencyjnej sieci neuronowej oparta na architekturze GRU, które na wejściu przyjmowały informację o położeniu przeszkody w klatce o indeksie N oraz wysokości powierzchni wody w klatkach N-1 oraz N-2. Przetestowano również wariant sieci, w którym połączono splotowe sieci neuronowe z rekurencyjnymi elementami. Schematyczne przedstawienie tego wariantu widoczne jest na rys. 5



Rysunek 5. Schematyczne przedstawienie rekurencyjnego wariantu sieci. Źródło: opracowanie własne.

W roli funkcji strat wykorzystano błąd średniokwadratowy. Za pomocą obserwacji zmiany wartości błędu średniokwadratowego oraz oceny empirycznej dokonywano określenia jakości generowanych wartości. Niezależnie od testowanej sieci wartość błędu kwadratowego stabilizowała się po kilku

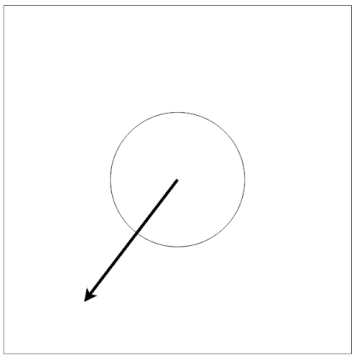
epokach w okolicach liczby 0,0013. Empiryczne badania wyników generacji sieci pozwoliły na stwierdzenie, że sieci nie generują fal, stopniowo zbliżają natomiast wartości do zera. Uznano, że sieć nie będzie w stanie wykonać symulacji zadanego scenariusza i postanowiono go zmodyfikować.

2.1.3 Wprowadzone modyfikacje, uzyskanie zadowalających wyników

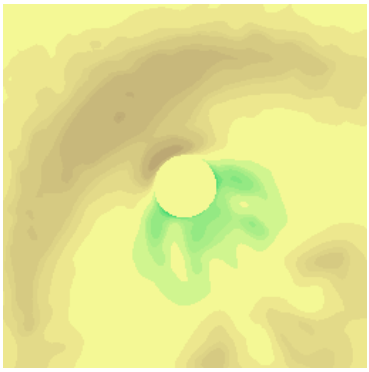
W celu określenia przyczyn porażki przyjrzano się istniejącym rozwiązaniom. Zaobserwowano, że wiele z nich [3][1] opiera się na istnieniu nieruchomej przeszkody. Można przypuszczać, że pozwala to sieci na zrozumienie struktury rozważanej przestrzeni. Można spekulować, że sieć miała utrudnione zadanie zrozumienia zależności między położeniem punktu w przestrzeni a zachowaniem płynu w tym punkcie. Nieruchoma przeszkoda może pozwolić sieci na łatwiejsze rozpoznanie zjawisk zachodzących w konkretnych miejscach. Kolejnym rozpoznanym problemem była reprezentacja danych dotyczących położenia przeszkody w przestrzeni. Trójelementowy wektor jest strukturą, którą trudno wykorzystać podczas pracy ze splotowymi sieciami neuronowymi. Zdecydowano się więc na bardziej skomplikowane przetwarzanie danych pozyskanych z symulacji. W celu określenia pozycji każdego punktu w przestrzeni względem przeszkody oraz przekazania informacji dotyczących kierunku, oraz prędkości poruszania się przeszkody wykorzystano następujące reprezentacje wartości:

- macierz wartości określającą odległość każdego punktu w przestrzeni od środka przeszkody (wartości mniejsze niż promień przeszkody zamieniano w wartość 0),
- wartości funkcji trygonometrycznej określające kąt, między wektorem, którego początkiem był środek przeszkody a końcem rozważany punkt,
- gradient określający kierunek oraz prędkość poruszania się przeszkody (kierunek spadku gradientu wyznacza kierunek poruszania się tejże, natomiast prędkość zmiany jego wartości reprezentuje jej prędkość).

Zdecydowano, że uproszczona symulacja sprawdzać się będzie w symulacji niewielkiego obszaru wody wokół poruszającego się walca. Zdecydowano się również na ograniczenie obszaru symulacji. Pozwoliło to na uzyskanie bardziej szczegółowej symulacji. Schemat symulacji przedstawia rys. 6, natomiast przykład uzyskanych danych widać na rys. 7.



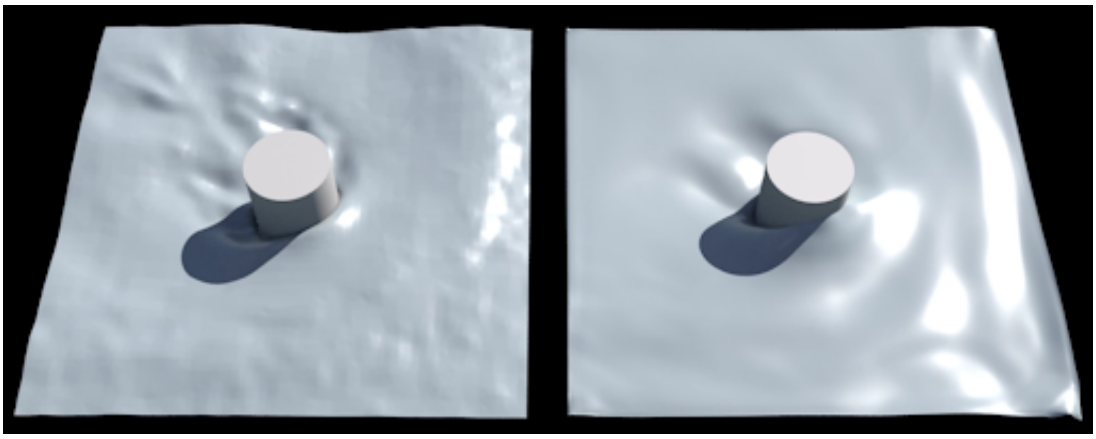
Rysunek 6. Schematyczne przedstawienie zmodyfikowanej prostej symulacji. Źródło: opracowanie własne.



Rysunek 7. Wizualizacja macierzy reprezentującej wysokość tafli wody. Źródło: opracowanie własne.

Zdecydowano się na zbadanie wielu architektur sieci i wybór najbardziej obiecującej z nich. Wykorzystano bibliotekę Pyradox [4] dzięki której stworzono sieć bazującą na architekturze U-Net [2]. Implementacja zawarta w module Pyradox zawierała błąd, który zdecydowano się naprawić udostępniając przy tym tę zmianę twórcy modułu. Zaproponowane zmiany zostały zaakceptowane i dołączone do kodu źródłowego. Eksperymenty wykazały, że sieć ta generowała najlepsze rezultaty spośród rozpatrywanych.

Sieć wykazywała tendencję polegającą na zmierzaniu w stronę płaskiej powierzchni, co prezentuje rys. 8. Zdecydowano się na przygotowanie złożonej funkcji strat nakładającej kary za takie zachowanie.



Rysunek 8. Przykład rezultatu działania sieci, przy której uczeniu jako funkcję strat wykorzystano jedynie błąd średniokwadratowy. Po lewej stronie widoczne są dane referencyjne, prawa strona przedstawia rezultat działania sieci. Źródło: opracowanie własne.

Pierwsza z kar obliczana była poprzez obliczenie odchylenia oczekiwanej wartości od wartości odpowiadającej płaskiej, niewzburzonej, powierzchni wody, następnie przemnożenie kwadratu tej wartości przez kwadrat różnicy między wygenerowaną wartością a wartością spodziewaną. Wskutek takiej operacji w miejscach, w których powinny znajdować się fale, lecz sieć ich nie wygenerowała, wartość błędu była znacząco powiększana. Kod odpowiedzialny za obliczanie tego

błędu przedstawiono poniżej:

```
loss += tf.reduce_mean((1 - tf.square(2 * y_true - 1,0)) * (tf.square(y_true - y_pred)))
```

Listing 2.1. Kod odpowiedzialny za obliczanie kary związanej z tworzeniem zbyt płaskiej powierzchni wody (uproszczony w celu zwiększenia czytelności).

Kolejna kara polegała na porównaniu największej oraz najmniejszej wartości wygenerowanej macierzy z największą, oraz najmniejszą wartością spośród spodziewanych. Kara ta wymuszała na sieci generowanie zakresu wartości bliższego spodziewanym, była szczególnie wysoka w wypadku, w którym sieć wygenerowała całkowicie płaską powierzchnię, mimo że powinna ona być pofalowana.

Wprowadzenie opisanej funkcji strat znacząco poprawiło rezultat działania sieci. Sieć była mniej ostrożna, a wygenerowane dane nie były płaskie. Kolejnym napotkanym problemem była bardzo rozmyta postać wygenerowanej powierzchni. Cechowały się one brakiem ostrych krawędzi. Jest to zachowanie typowe dla sieci wykorzystujących błąd średniokwadratowy. Próbowano przeciwdziałać temu efektowi poprzez wykorzystanie, oprócz błędu średniokwadratowego, błędu SSIM. Oceniany empirycznie rezultat nie był jednak znacząco lepszy.

Opisane czynności zwiększyły wartość błędu średniokwadratowego, jednak poprawiły odbiór efektu działania sieci przez człowieka. W przypadku badanych zagadnień jakość działania sieci nie jest łatwa do oceny. W pracy wykorzystywano w tym celu głównie wartość błędu średniokwadratowego, jednak nie jest to wystarczający sposób na ocenę rezultatu jej działania. Ostateczna ocena jest subiektywna, ponieważ w rozpatrywanym przykładzie istotniejszą niż dokładność odwzorowania symulacji rolę odgrywa uznanie powierzchni za poprawnie wyglądającą przez ludzkiego odbiorcę. Nie musi to koniecznie oznaczać jej fizycznej poprawności. Istotność tego rozróżnienia zauważyć można analizując wymienione w początkowej części tej pracy istniejące rozwiązania symulacji powierzchni wody. Wiele z nich opiera się na mechanizmach niemających nic wspólnego z poprawną fizycznie symulacją. Przykładem takiego, fizycznie niepoprawnego, jednak powszechnie uznawanego za zadowalający na potrzeby gier, rozwiązania mogą być opisane we wcześniejszym rozdziale pracy paczki fal. Jednym z powodów, dla których odbiorcy wydają się one generować poprawny wynik, są drobne detale obecne w generowanych przez to rozwiązanie powierzchniach. Sieć szkolona za pomocą błędu średniokwadratowego i opierająca się wyłącznie na zbliżaniu wyniku swojego działania do wygenerowanych wcześniej wartości ma tendencję do ignorowania ich, tworzy rozmyty rezultat. Dzieje się tak, ponieważ niewielkie detale nie mają wielkiego wpływu na błąd średniokwadratowy. Są jednak bardzo istotne dla ludzkiego odbiorcy, ponieważ zwraca on na nie uwagę. Rezultat bliski oczekiwanemu, jednak pozbawiony małego detalu odbiera jako zbyt gładki lub rozmyty.

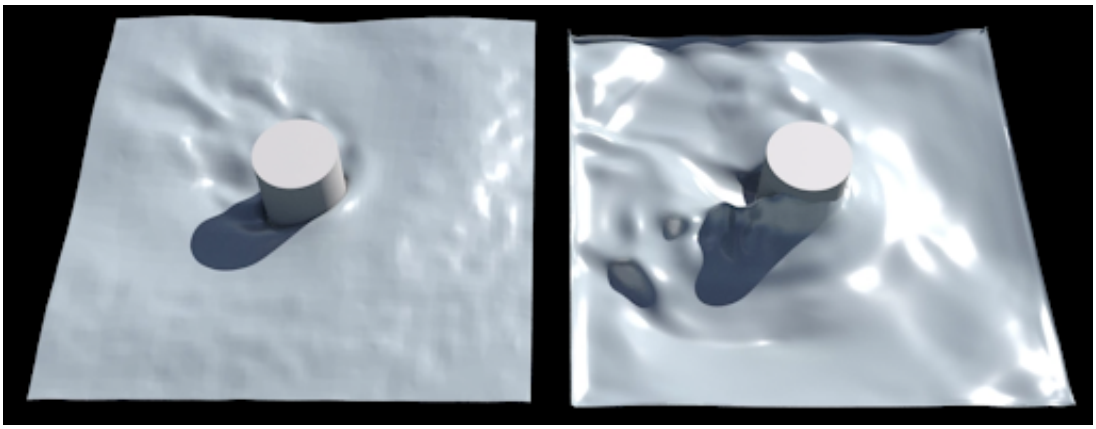
Znaczącą poprawę spowodowało zastosowanie bardziej złożonej architektury, wykorzystującej mechanizmy typowe dla sieci GAN, wykorzystano sieć pełniącą rolę dyskryminatora. Przeprowadzone eksperymenty pozwoliły ustalić, że jej wykorzystanie zmusza sieć do generowania wspomnianych szczegółów, nawet jeśli wpływają negatywnie na błąd średniokwadratowy, jeśli dyskryminator uzna je

za bardziej odpowiadające danym referencyjnym. Zdaniem autora pracy ma to pozytywny wpływ na subiektywnie rozumianą jakość efektu działania sieci.

W celu implementacji sieci używającej dyskryminatora porzucono opisaną wcześniej klasę CustomModel. W miejsce nadpisania metody klasy Model konieczne było przygotowanie kilku złożonych funkcji, odpowiedzialnej za trenowanie modelu dyskryminatora. Stworzono również nowy callback, uruchamiany na początku każdej epoki, którego zadaniem było uruchamianie tej funkcji. Mechanizm jej działania sprowadzał się do pobrania danych wejściowych dla generatora, wykorzystania go, do wygenerowania danych na ich podstawie, następnie przekazaniu tych danych do dyskryminatora, którego zadaniem było rozpoznanie danych wygenerowanych od danych referencyjnych. Tak wytrenowany dyskryminator wykorzystywany był przy obliczaniu wartości funkcji strat podczas szkolenia generatora. Przygotowano funkcję strat generatora, której celem było zwiększenie błędu dyskryminatora.

Rezultat działania tak wyszkolonej sieci prezentuje rys. 9. Można na nim zauważyć ostrzejsze krawędzie oraz większą wysokość fali, niż w przykładach poprzednich. Warto jednak zwrócić uwagę na fakt, że fale te są też znacząco wyższe niż te, znajdujące się w danych referencyjnych. Jest to błąd działania sieci, który można skorygować, dostrajając wagi używane w funkcjach strat.

Uzyskane rezultaty uznano za zadowalające, ponieważ udowadniały one wykonalność projektu. Zdecydowano się wiec na rozbudowanie symulacji.

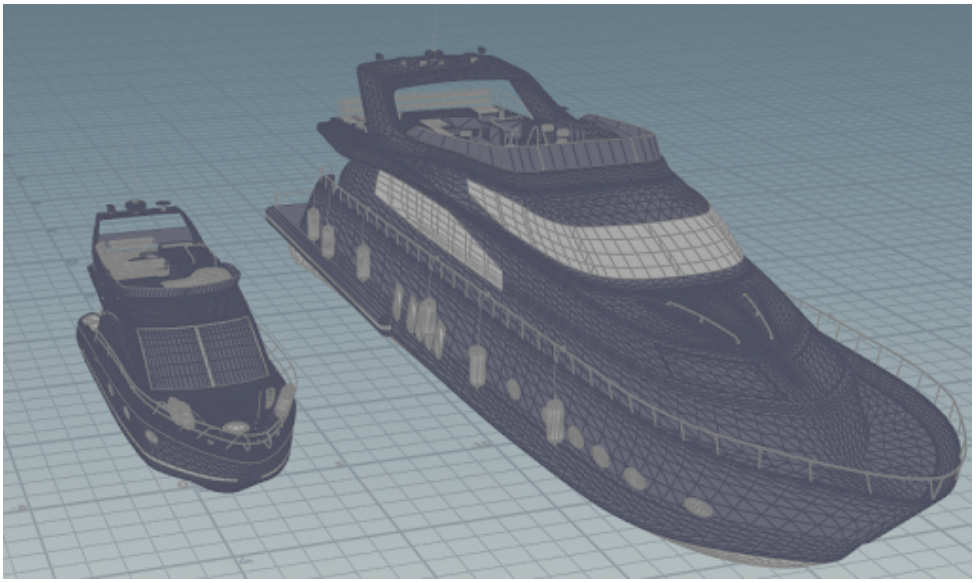


Rysunek 9. Przykład rezultatu działania sieci, przy której uczeniu wykorzystano architekturę wykorzystującą sieć dyskryminatora oraz złożoną funkcję strat. Po lewej stronie widoczne są dane referencyjne, prawa strona przedstawia rezultat działania sieci. Źródło: opracowanie własne.

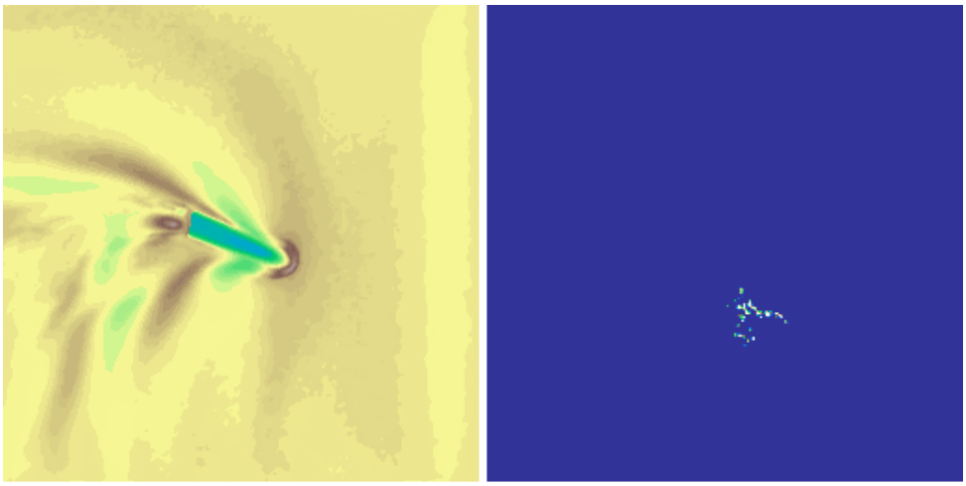
2.2 Złożona symulacja powierzchni wody

Zgodnie z początkowymi założeniami, po odnalezieniu odpowiedniej architektury oraz uzyskaniu wyników potwierdzających wykonalność projektu, przystąpiono do przygotowywania bardziej złożonej wersji symulacji. Postanowiono zamienić kształt przeszkody na geometrię przypominającą łódź. Nie jest to kształt symetryczny, konieczne więc było uwzględnienie kierunku, w jakim zwrócony jest dziób łodzi. Dodatkowym utrudnieniem było to, że kierunek, w który zwrócony jest kadłub łodzi

nie musi być tym samym, w jakim łódź się porusza, zmiana położenia kadłuba powinna wyprzedzać zmianę kierunku. Złożony kształt łodzi wpływa na kształt tworzących się fali w sposób zależny od relacji położenia kadłuba do wektora prędkości. Zdecydowano się na wykonanie symulacji dla dwóch modeli łodzi, różniących się znacząco rozmiarem, co przedstawia rys. 10. Postanowiono również powiększyć symulowany obszar. Ważną zmianą było też rozbudowanie symulacji o symulację piany wodnej, ponieważ uznano, że efekt ten wpływa bardzo pozytywnie na odbiór jakości symulacji przez człowieka. Przykłady wygenerowanych danych widoczne są na rys. 11.



Rysunek 10. Przykład danych będących efektem symulacji. Obraz z lewej strony przedstawia wysokość powierzchni wody, po prawej stronie znajduje się reprezentacja gęstości piany. Źródło: opracowanie własne.

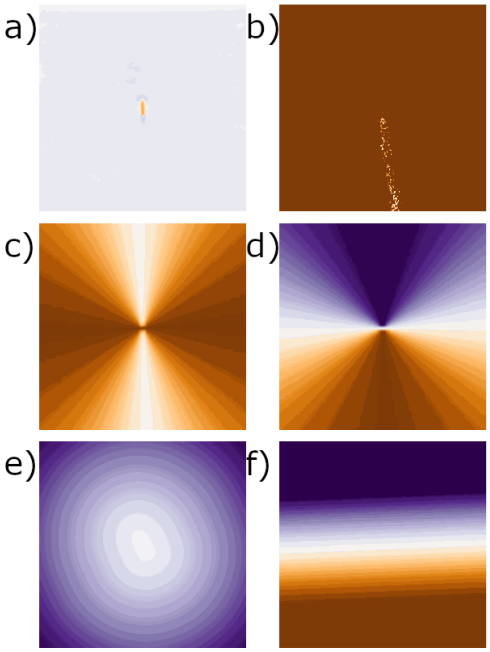


Rysunek 11. Modele łodzi wykorzystane w symulacji.

Wielokrotnie zmieniano architekturę sieci oraz reprezentację danych dotyczącą przemieszczenia oraz pozycji łodzi. Ponownie najlepsze rezultaty osiągnięto wykorzystując architekturę sieci opartą na sieci U-Net. Przetestowano również wiele reprezentacji danych, ostatecznie najlepsze wyniki uzyskując dzięki przedstawieniu ich w następującej formie, której wizualizacja przedstawiona jest na rys. 12 Źródło: opracowanie własne.:

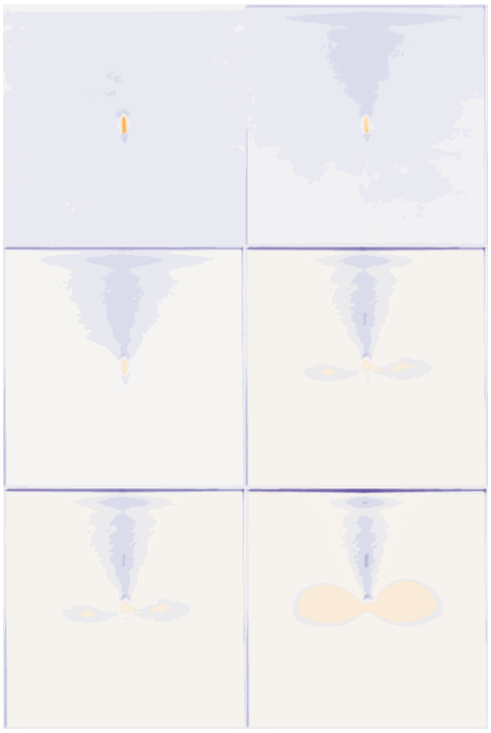
- wysokość powierzchni wody w klatce poprzedzającej rozpatrywaną (a),
- wartość natężenia piany w klatce poprzedzającej rozpatrywaną (b),
- macierz zawierającą wartości odległości od powierzchni dna łodzi (c),
- trójwymiarową macierz o rozmiarach 64x64x3 której wartości oznaczały:
  - wartości funkcji sinus kąta zawartego między wektorem, którego początkiem był środek przeszkody a końcem rozważany punkt oraz wektorem określającym zwrot dziobu łodzi (d),
  - wartości funkcji cosinus dwukrotności kąta zawartego między wektorem, którego początkiem był środek przeszkody a końcem rozważany punkt oraz wektorem określającym zwrot dziobu łodzi (e),
  - gradient określający kierunek oraz prędkość poruszania się przeszkody. Kierunek spadku gradientu wyznacza kierunek poruszania się tejże, natomiast prędkość zmiany jego wartości reprezentuje jej prędkość (f).

Podobnie jak w przypadku uproszczonej animacji wykorzystano sieć dyskryminatora. Rozbudowano sposób, w jaki budowana jest sieć oraz w jaki obliczane są funkcje straty. Stworzono trzy sieci o nazwach: generator, dyskryminator oraz gan. Wyjściami sieci generator były wysokość powierzchni wody oraz gęstość piany. Na podstawie tych danych obliczano wartość błędu za pomocą opisanej wcześniej funkcji strat. Wyjściem sieci dyskryminator było prawdopodobieństwo tego, że podane na jej wejściach dane zostały wygenerowane przez sieć generator. Użyto funkcji cross entropy w celu obliczenia strat tego wyjścia. Sieć gan była kombinacją dwóch wymienionych i miała wszystkie wymienione wyjścia. Na początku każdej partii danych odbywało się uczenie sieci dyskryminator rozpoznawania wartości generowanych przez sieć generator od wartości pochodzących z bazy danych. Podczas procesu uczenia modyfikowano zarówno wagi funkcji strat poszczególnych wyjść sieci, jak i wagi zawarte w opisanej wcześniej funkcji strat.



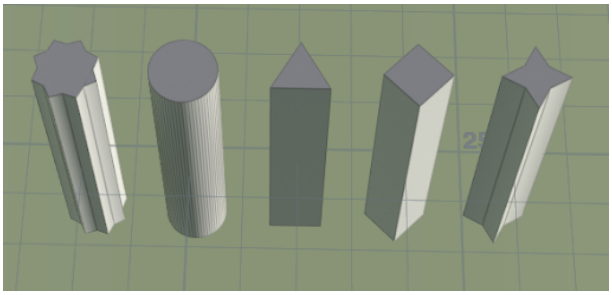
Rysunek 12. Wizualizacja danych przekazywanych sieci. Źródło: opracowanie własne.

Każda zmiana wymagała wielogodzinnych lub wielodniowych eksperymentów polegających na trenowaniu sieci i obserwacji wyników. Ponownie użyto opisanej wcześniej rozbudowanej funkcji strat oraz próbowano doprowadzić do uzyskania zadowalającego efektu poprzez manipulację jej wagami. Nie udało się jednak osiągnąć zadowalającego efektu, a uzyskane rezultaty prezentuje rys. 13. Wyraźnie wskazują one na niezdolność sieci do odtworzenia zachowania powierzchni wody. W przypadku generacji powierzchni wody zauważalna jest tendencja do zmniejszania różnic wysokości. Warto zauważyć, że sieć zdołała nauczyć się, że przed dziobem łodzi, w kierunku jej poruszania się, następuje spiętrzenie wody. W przypadku gęstości piany następuje stopniowe zbliżanie wartości do wartości podawanej na wejściu sieci odległości punktów od powierzchni dna łodzi. Nie zdołano wytłumaczyć tego zjawiska.



Rysunek 13. Wybrane klatki sekwencji wartości wysokości powierzchni wody wygenerowanej przez sieć neuronową. Źródło: opracowanie własne.

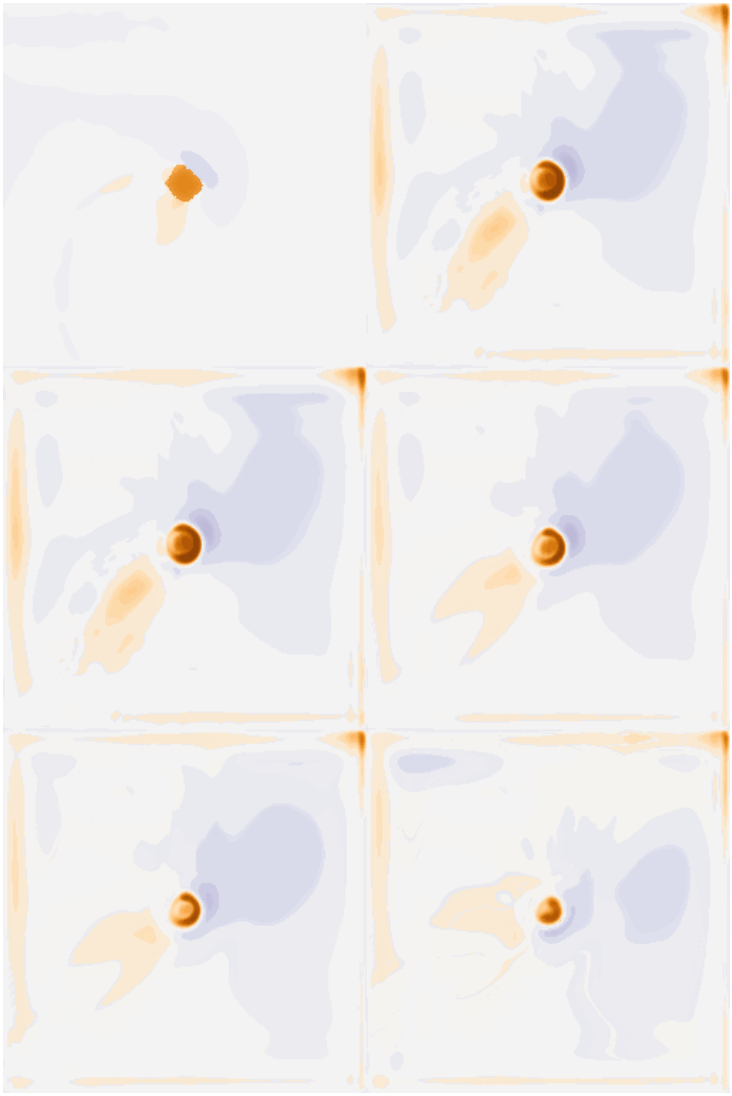
Zdecydowano się na znaczne uproszczenie animacji. Zamiast skomplikowanego kształtu kadłuba wykorzystano kilka rodzajów przeszkód o prostych kształtach przekrojów. Geometrie przeszkód zestawiono na rys. 14. Przeszkody nie obracały się, ich orientacja w przestrzeni pozostawała stała. Przypuszczano, że problemy z nauką wynikały ze złożonej relacji między kierunkiem, w którym zwrócony jest kadłub łodzi, kierunkiem, w którym łódź się porusza oraz generowanymi falami.



Rysunek 14. Geometria przeszkód użytych w symulacji.

Kolejną wprowadzoną modyfikacją była zmiana procesu uczenia sieci. Postanowiono, że przez początkowe epoki sieć nie będzie wykorzystywała funkcji strat opartej na sieci dyskryminatora. Przypuszczano, że może to zmusić sieć do nauczenia się podstawowych mechanizmów mających miejsce w symulacji w bardziej stabilny sposób. W późniejszych epokach wykorzystywano wartość funkcji strat

obliczaną przy użyciu sieci dyskryminatora, zwiększając stopniowo jej wagę. Eksperymenty wykazały, że założenia te były słuszne. osiągnięte rezultaty prezentuje rys. 15.



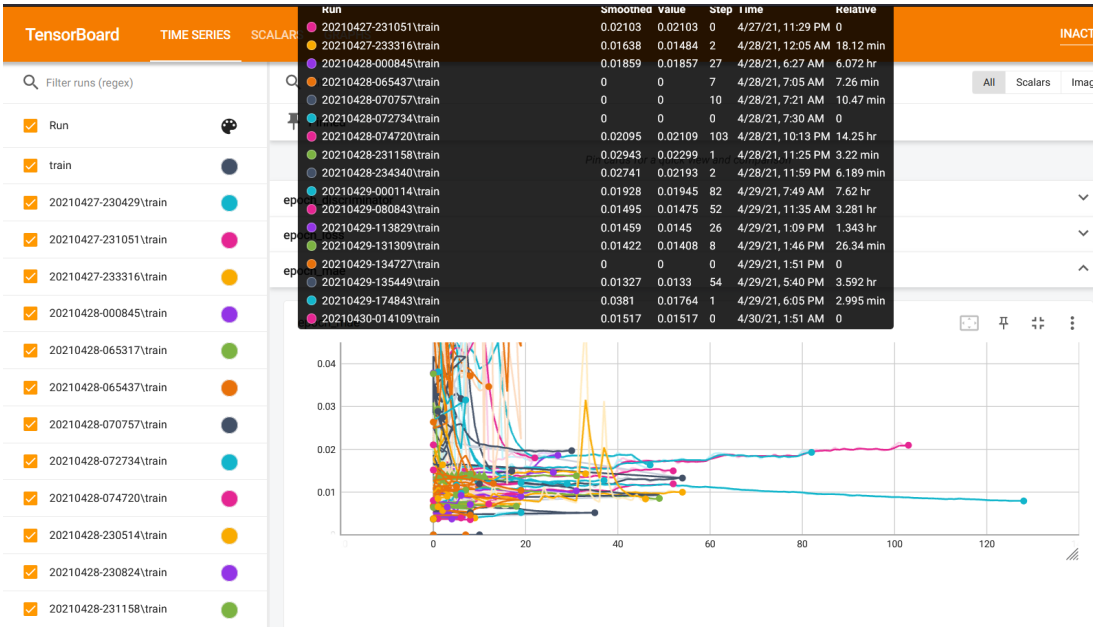
Rysunek 15. Wybrane klatki sekwencji wartości wysokości powierzchni wody wygenerowanej przez sieć neuronową. Źródło: opracowanie własne.

Osiągnięte wyniki uznano za wystarczająco dobre, by przejść do kolejnego etapu prac.

2.3 Trenowanie sieci neuronowej

Trenowanie sieci przeprowadzono przy użyciu narzędzia Jupyter Notebook. W środowisku tym przygotowywano i uruchamiano skrypty napisane w języku Python. W ramach przeprowadzanych badań przetestowano ponad dwadzieścia wersji skryptu trenującego sieć. Wersje różniły się między sobą procesem przygotowania danych, rodzajem danych przekazywanych do sieci oraz szczegółami

jej architektury. Na ostatecznie wykorzystaną wersję skryptu składało się ponad 700 linii kodu odpowiedzialnego za przygotowanie danych, wraz z wizualizacjami pozwalającymi na weryfikację ich poprawności, stworzenie sieci, pętlę trenującą sieć, zapis przebiegu eksperymentu oraz wykonywanie kopii zapasowych i prezentacji wyników na różnych etapach procesu. W celu śledzenia przebiegu procesu wykorzystano aplikację Tensorboard. Na Rys. 16 przedstawiono zrzut ekranu aplikacji, prezentujący część z zapisanych przebiegów procesu trenowania. Długość przebiegu jest różna, ponieważ proces treningu przerywany był w momencie, w którym wyniki nie wykazywały poprawy. Następowała wtedy ich analiza, zmiana skryptu lub parametrów treningu i ponowienie procesu. Warto w tym momencie zaznaczyć, że błąd średniokwadratowy nie oddaje poprawnie jakości wyników generowanych przez sieć, ponieważ w rozważanym projekcie kluczowe znaczenie ma odbiór wyniku przez człowieka i najważniejsze były w nim subiektywne odczucia wywoływane przez wygenerowane wyniki.



Rysunek 16. Zrzut ekranu z aplikacji Tensorboard przedstawiający część zapisów procesu trenowania różnych wariantów sieci. Źródło: opracowanie własne.

2.4 Implementacja sieci w silniku Unreal Engine 4

W celu wykonania interaktywnej demonstracji działania sieci wykorzystano:

- silnik Unreal Engine 4.27.2,
- bibliotekę CPPFlow 2,
- biblioteki Tensorflow dla języka C w wersji 2.7.0.

Za pomocą edytora silnika Unreal Engine przygotowano scenę, składającą się z reprezentacji graficznej przeszkody oraz interfejsu graficznego, za pomocą którego użytkownik może zmieniać parametry symulacji. Kod odpowiedzialny za obsługę sieci neuronowej zawarty został w stworzonej

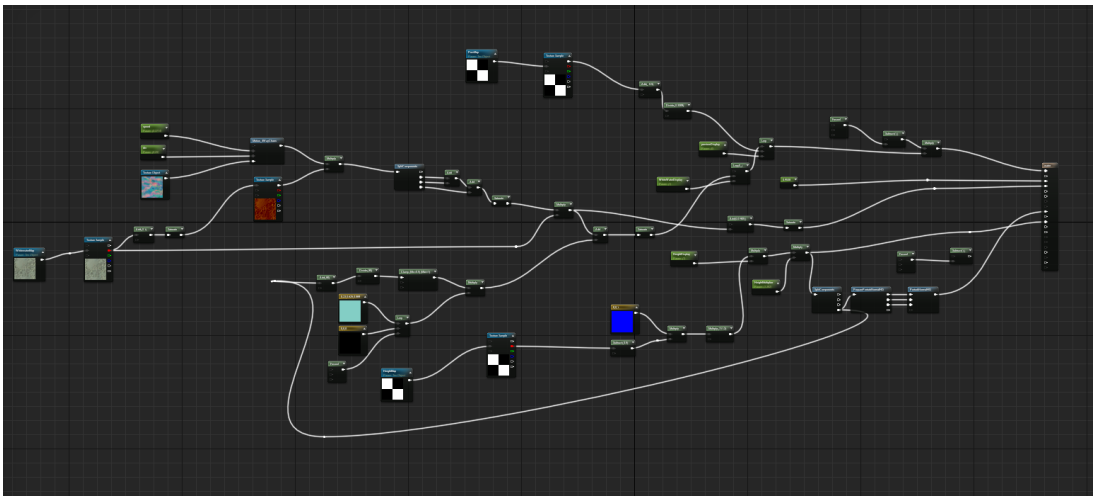
w tym celu klasie ATensorFlowNetwork. Obiekt reprezentujący powierzchnię wody tworzony jest podczas wykonywania konstruktora tej klasy.

Demonstracja pozwala na wizualizację efektów działania obu opisanych wcześniej wersji sieci: prostej, generującej jedynie mapę wysokości powierzchni wody, oraz skomplikowanej, generującej zarówno mapę wysokości powierzchni wody, jak i mapę gęstości piany. Metody klasy ATensorFlowNetwork dostępne są z poziomu dostępnego w silniku Unreal Engine wygodnego systemu wizualnego programowania opartego na tak zwanych „blueprintach" - skryptach wizualizowanych za pomocą grafów skierowanych. Za ich pomocą przygotowano sekwencję inicjalizacji sieci neuronowej, powiązano zmiany wartości interfejsu ze zmianą wartości pól klasy oraz ustanowiono uruchamianie funkcji generującej nowy zestaw danych za pomocą sieci trzydzieści razy na sekundę, co odpowiada przyjętym założeniom. Rezultaty działania sieci zapisywane są w formie tekstur, które wykorzystywane są przez Shader geometrii powierzchni wody. Modyfikacja pozycji wierzchołków geometrii odbywa się podczas działania Vertex Shadera. Graf reprezentujący materiał przedstawia rys. 17.

W celu wykorzystania do obliczeń karty graficznej konieczna była instalacja bibliotek CUDNN w wersji 8.1 oraz CUDA w wersji 11.2, ponieważ wykorzystane biblioteki Tensorflow oparte są na tych ich wersjach. Zabieg ten pozwolił na znaczne przyspieszenie obliczeń, wynoszące nawet 40 razy. Wykonywanie obliczeń za pomocą wyłącznie CPU nie pozwalało na osiągnięcie zakładanej liczby odświeżeń geometrii na sekundę, wykorzystanie GPU zmniejszyło jednak ten czas do poziomu, w którym możliwe jest osiągnięcie znacznie częstszego odświeżania.

Wykorzystanie biblioteki CPPFlow 2 wymusiło zmianę standardu języka na C++ 17. Zostało to osiągnięte przez modyfikację skryptu konfiguracyjnego projektu.

Ostatecznie przygotowanie wtyczki wymagało napisania ponad 370 linii kodu c++, w całości dostępnego w repozytorium udostępnionym na platformie GitHub pod adresem: https://github.com/p4vv37/ueflow



Rysunek 17. Graf reprezentujący materiał powierzchni wody. Źródło: opracowanie własne.

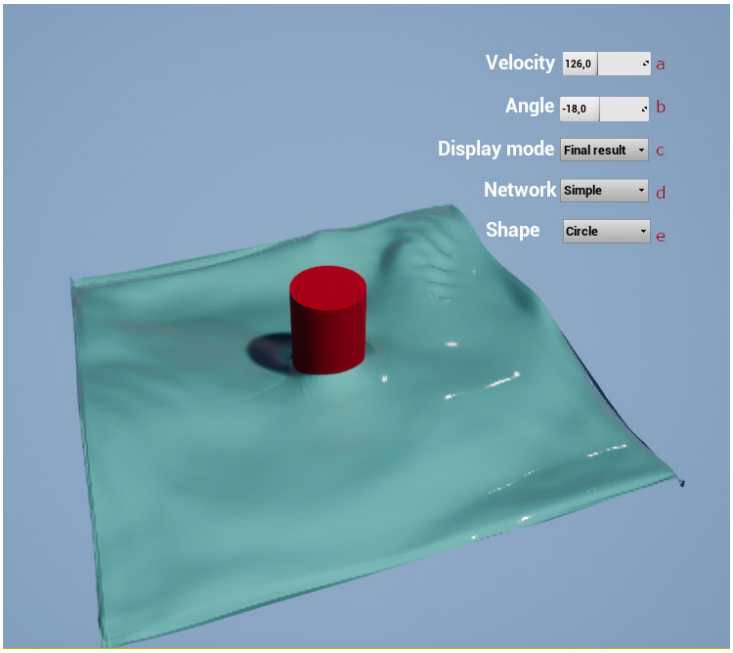
2.5 Przedstawienie oraz analiza wyników

2.5.1 Przedstawienie wyników

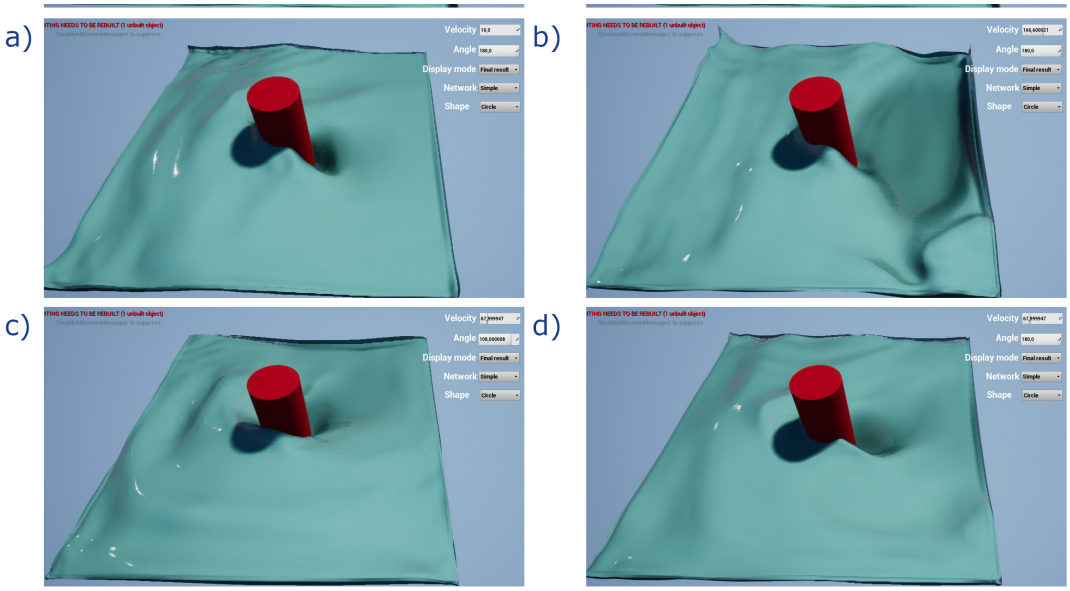
Interaktywna demonstracja działania sieci neuronowej jest w stanie wygenerować powierzchnię wody reagującą na zmianę wartości takich parametrów jak prędkość czy kierunek poruszania się przeszkody. Najważniejsze elementy interfejsu demonstracji przedstawiono na rys. 18. Widoczne są na nim elementy służące do kontroli parametrów symulacji:

- prędkość poruszania się przeszkody (a),
- kierunek poruszania się przeszkody reprezentowany za pomocą kąta odchylenia od osi X (b),
- wybór trybu wyświetlania (c). Zaimplementowano 5 trybów wyświetlania przedstawionych na rys. 20:
  - ostateczny rezultat — widok powierzchni wody uwzględniający wszystkie jej parametry oraz cechującej się złożonym materiałem,
  - wysokość wody — Przedstawienie wysokości wody w postaci koloru nałożonego na płaską powierzchnię,
  - piana — Przedstawienie gęstości piany w postaci koloru nałożonego na płaską powierzchnię,
  - wartości funkcji cosinus dwukrotności kąta zawartego między wektorem, którego początkiem był środek przeszkody, a końcem rozważany punkt oraz wektorem, określającym zwrot dziobu łodzi (e),
  - wartości funkcji cosinus dwukrotności kąta zawartego między wektorem, którego początkiem był środek przeszkody a końcem rozważany punkt oraz wektorem określającym zwrot dziobu łodzi (e),
- wybór między złożonym oraz prostym wariantem sieci (d),
- kształt przeszkody (e).

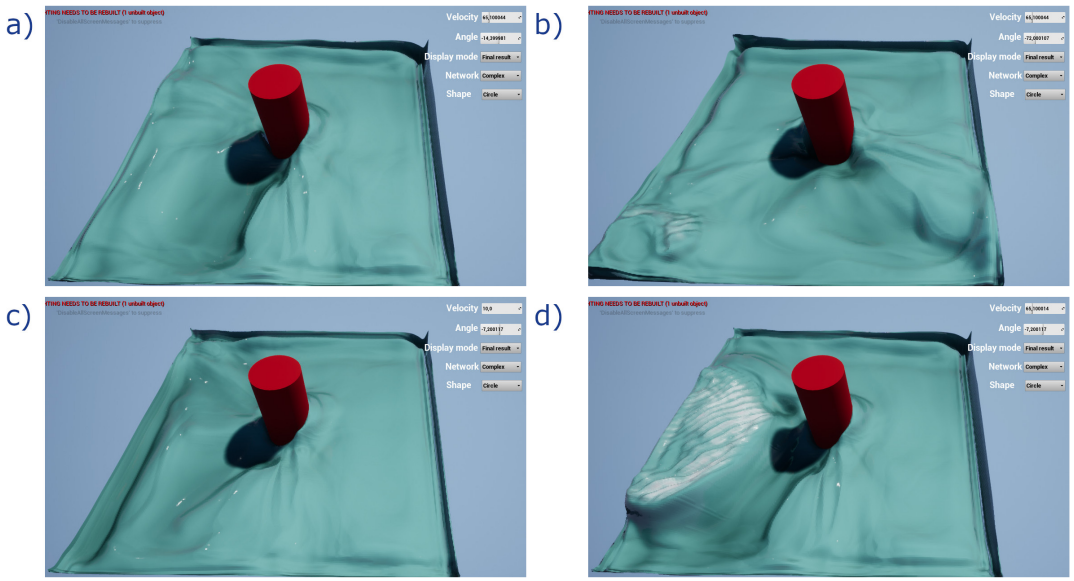
Na rys. 19 oraz rys. 20 przedstawiono przykłady wygenerowanej powierzchni wody, oraz wpływ parametrów symulacji na jej kształt przy wykorzystaniu odpowiednio prostego, oraz złożonego wariantu symulacji.



Rysunek 18. Ekran interaktywnej demonstracji. Źródło: opracowanie własne.

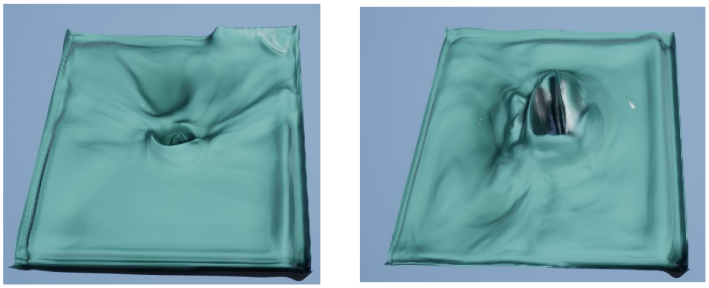


Rysunek 19. Prezentacja wpływu różnych wartości parametrów symulacji na wygląd powierzchni wody generowanej za pomocą prostej sieci neuronowej: obrazy a oraz b prezentują wpływ zmiany prędkości poruszania się przeszkody przy zachowaniu kierunku, obrazy c oraz d prezentują wpływ zmiany kierunku poruszania się przeszkody przy zachowaniu prędkości jej przemieszczania się. Źródło: opracowanie własne.



Rysunek 20. Prezentacja wpływu różnych wartości parametrów symulacji na wygląd powierzchni wody generowanej za pomocą złożonej sieci neuronowej: obrazy a oraz b prezentują wpływ zmiany kierunku poruszania się przeszkody przy zachowaniu prędkości jej przemieszczania się, obrazy c oraz d prezentują wpływ zmiany prędkości poruszania się przeszkody przy zachowaniu kierunku. Źródło: opracowanie własne.

W ramach badań sprawdzono, czy sieć jest w stanie generalizować wyniki dla innych kształtów przeszkody niż kołowa, na której była szkolona. W tym celu przeprowadzono symulacje dla przeszkody o przekroju kwadratowym i porównano wyniki z tymi uzyskanymi dla przeszkody o przekroju kołowym. Analiza rezultatów doprowadza do wniosku, że sieć nie jest w stanie poprawnie generalizować wyników dla przeszkody o przekroju kwadratowym. Wyniki dla tej przeszkody zawierają więcej błędów i obszarów o kształcie odbiegającym od oczekiwań, niż wyniki dla przeszkody o przekroju kołowym. Mimo to warto nadmienić, że uzyskane wyniki w pewnym stopniu odwzorowują nowy kształt przekroju przeszkody. Porównanie rezultatów przedstawiono na rys. 21

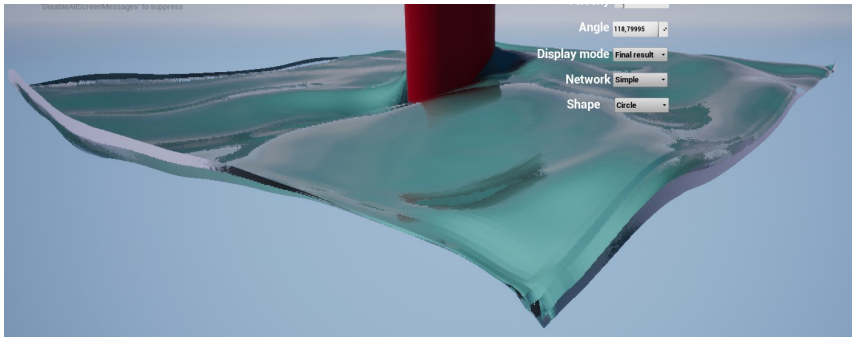


Rysunek 21. Porównanie rezultatów wykonania symulacji dla przeszkody o przekroju w kształcie okręgu (a), więc tym, na którym wykonywane były symulacje używane przez nią podczas nauki, oraz o przekroju kwadratowym(b). Sieć w pewnym stopniu uwzględnia nowy kształt przekroju, jednak zdecydowanie widoczne jest pogorszenie jakości rezultatu. Źródło: opracowanie własne.

2.5.2 Analiza jakościowa wyników

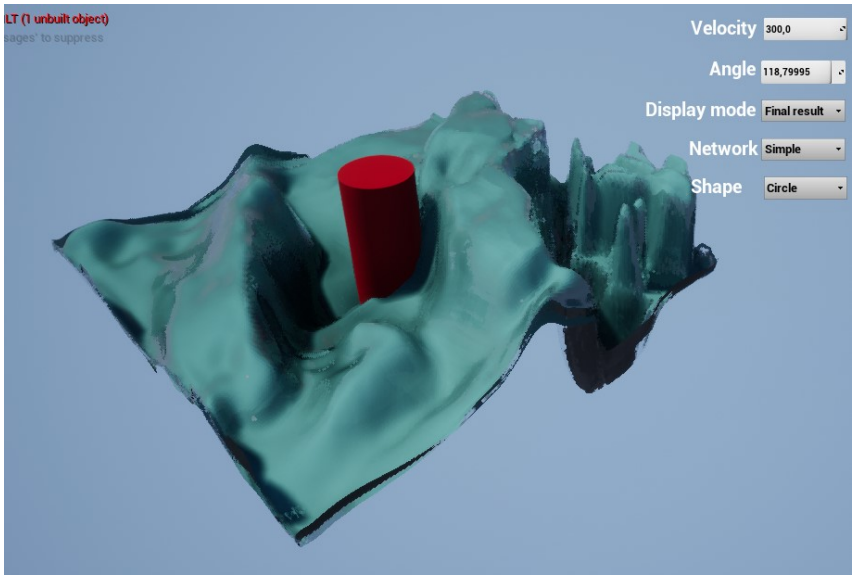
Przedstawione wcześniej przykłady powierzchni wody wygenerowane zostały przy zastosowaniu wartości parametrów symulacji zawartych w zakresie tych, które wykorzystywano podczas generowania zbioru danych użytego w procesie uczenia sieci. Ocena ich jakości nie może jednak polegać wyłącznie na wygenerowaniu symulacji w tych samych warunkach i porównaniu ich z efektami działania sieci. Porównanie takie nie określiłoby poprawnie subiektywnych odczuć człowieka. W związku z tym, że w projekcie najważniejszy jest odbiór wyniku przez człowieka, nie istnieje więc konkretny i oczywisty sposób obiektywnej oceny wyników. Jak wspomniano we wcześniejszym fragmencie pracy wartości błędu reprezentowane przez błąd średniokwadratowy lub algorytmy typu Structural Similarity Index (SSIM) nie odzwierciedlają prawidłowo empirycznej oceny jakości dokonywanej przez ludzkiego odbiorcę. Dla przykładu wyższy wynik uzyskałaby w nim nienaturalnie wyglądająca, gładka powierzchnia, niż złożona i wzburzona, wizualnie bardziej zbliżona do referencji. Kolejnym problemem przy takiej ocenie byłby jej dynamiczny i miejscami turbulentny charakter. W związku z brakiem możliwości podania obiektywnych wartości reprezentujących należycie jakość uzyskanych rezultatów postanowiono pozostawić tą kwestię subiektywnej ocenie odbiorcy. Warto jednak dokładnie opisać i przeanalizować miejsca, w których widoczne są ewidentne błędy jej działania. Zostaną one wymienione w dalszej części tego rozdziału,

Niezależnie od przyjętych parametrów symulacji na krawędziach generowanej mapy wysokości widoczne są artefakty. Efekt ten zaprezentowano na rys. 22 Błąd ten można w łatwy sposób ukryć zmniejszając obszar wyświetlanej powierzchni.



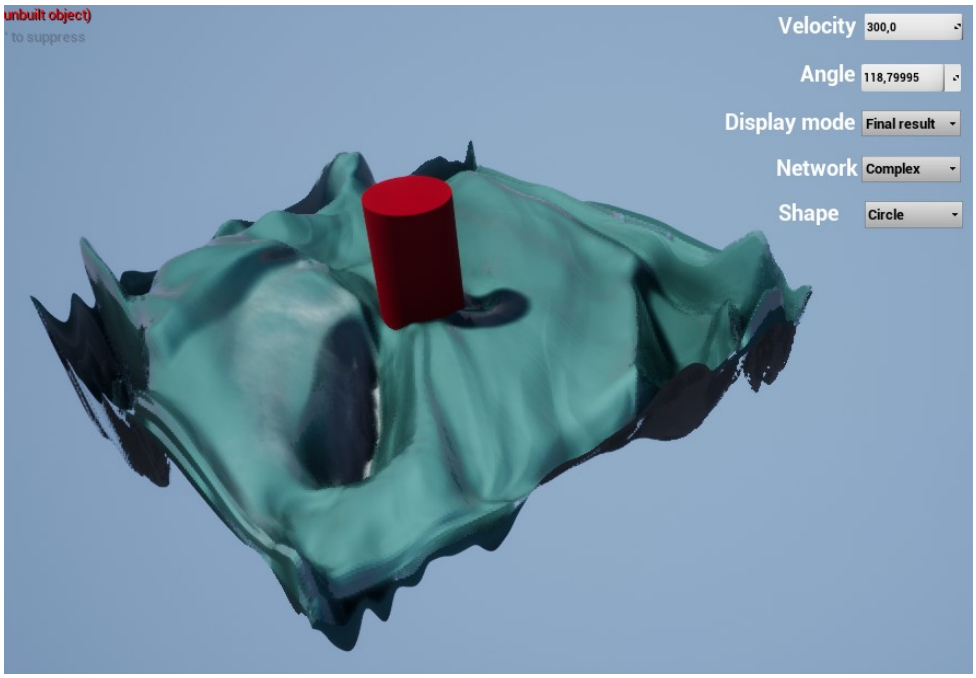
Rysunek 22. Zbliżenie krawędzi symulowanej powierzchni przedstawiające powstałe w tym obszarze artefakty. Źródło: opracowanie własne.

Przekroczenie wartości parametrów wykorzystanych podczas generacji bazy danych użytych do nauki skutkuje bardzo zauważalnymi błędami. Błędy te zauważalne są w szczególności w obszarach znajdujących się przed oraz za przeszkodą w kierunku jej ruchu, podczas, gdy w obszarach znajdujących się równolegle do kierunku ruchu generowana powierzchnia wygląda poprawnie. Przykłady tego efektu odpowiednio dla sieci prostej oraz złożonej widoczne są na rys. 24 oraz rys. 24 Obecność tego efektu wskazuje na niską zdolność sieci do generalizacji.



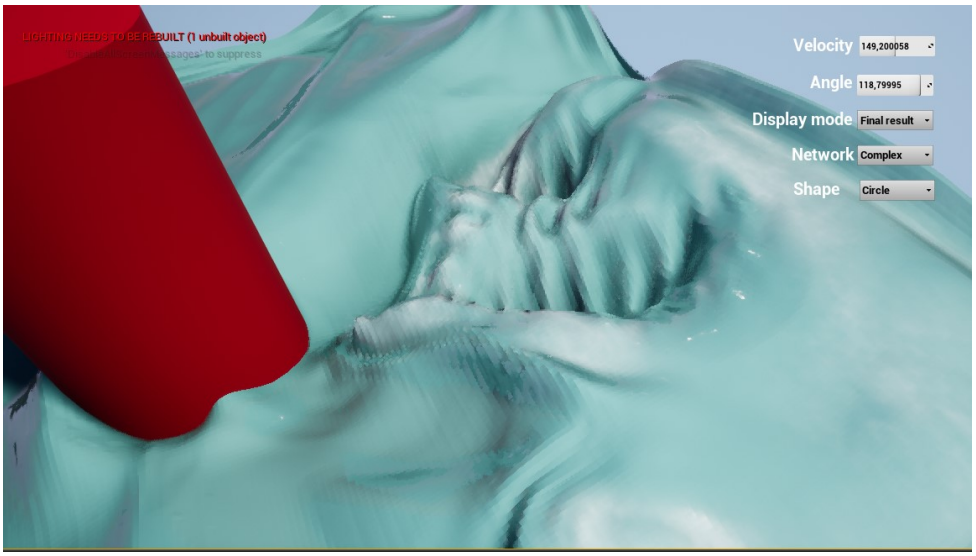
Rysunek 23. Przykład artefaktów generowanych przez prosty wariant sieci po znacznym zwiększeniu wartości prędkości poruszania się przeszkody poza zakres wykorzystany podczas uczenia sieci. Źródło: opracowanie własne.

Symulacja generowana przez skomplikowany wariant sieci bardzo szybko ulega stabilizacji, zamieniając się w statyczną powierzchnię przypominającą pojedynczą klatkę symulacji. Efekt ten nie występuje w przypadku sieci prostej.



Rysunek 24. Przykład artefaktów generowanych przez złożony wariant sieci po znacznym zwiększeniu wartości prędkości poruszania się przeszkody poza zakres wykorzystany podczas uczenia sieci. Źródło: opracowanie własne.

Te obszary sieci, które przykryte są gęstą pianą, często cechują liczne, dynamicznie zmieniające się detale o wysokiej częstotliwości, których istnienia nie odnotowano na danych wykorzystywanych w procesie jej uczenia. Efekt ten prezentuje rys. 25



Rysunek 25. Przykład artefaktów występujących na obszarach powierzchni pokrytych gęstą pianą. Źródło: opracowanie własne.

2.5.3 Analiza wydajnościowa wyników

Za pomocą dostępnych w bibliotece Tensorflow narzędzi określono parametry obu rozpatrywanych sieci. Sieć generująca złożony wariant wykorzystuje 7 859 330 parametrów. Sieć generująca uproszczony wariant symulacji jest bardziej złożona, cechuje się 21 961 986 parametrami. Z liczb tych wynika, że złożoność rozwiązywanego przez sieć problemu nie odpowiada złożoności samej sieci. Jest to efektem nieco odmiennej architektury obu implementacji wynikającej z prób odnalezienia najlepiej spełniających zadanie rozwiązań. Nie była to sytuacja zaplanowana na etapie projektowania sieci.

W celu oceny wydajności wykorzystano narzędzie nvidia-smi oraz dokonano pomiaru czasu, jaki trwało wykonywanie przez sieć predykcji podczas działania interaktywnej demonstracji. Warto zaznaczyć, że wykorzystane metody pomiarowe cechują się niewielką dokładnością, ponieważ na pozyskane wartości wpływ może mieć obciążenie karty graficznej niewynikające z działania samej sieci neuronowej. Pomiarów dokonano przy wykorzystaniu dwóch stacji roboczych. W obu wypadkach sieć neuronowa wykorzystywała akcelerator graficzny. Stacje wyposażone były w karty graficzne o następujących parametrach [8][8]:

- NVIDIA GeForce RTX 3080,
  - architektura Ampere,
  - 10 GB pamięci GDDR6X,
  - szerokość szyny pamięci: 320 bity,
  - przepustowość pamięci: 760,3 GB/s,

  - proces technologiczny 8 nm.,
  - liczba jednostek Cuda Cores: 8704,
  - liczba jednostek Tensor Cores: 272,
  - 28300 milionów tranzystorów,
  - taktowanie wynoszące 1440 MHz,
  - taktowanie w trybie Boost wynoszące 1440 MHz,
  - taktowanie pamięci 1188 MHz,
  - TDP wynoszące 320 W,
  - podana przez producenta maksymalna liczba operacji zmiennoprzecinkowych wykonywanych na sekundę: 29,77,
- NVIDIA GeForce RTX 2060 Mobile,
  - architektura Turing,
  - 6GB pamięci GDDR6,
  - szerokość szyny pamięci: 192 bity,
  - przepustowość pamięci: 336 GB/s,
  - proces technologiczny 12 nm.,
  - 10800 milionów tranzystorów,
  - liczba jednostek Cuda Cores: 1920,
  - liczba jednostek Tensor Cores: 240,
  - taktowanie wynoszące 960 MHz,
  - taktowanie w trybie Boost wynoszące 1200 MHz,
  - taktowanie pamięci 1188 MHz,
  - maksymalny zmierzony pobór mocy podczas działania sieci na poziomie 91 W,
  - podana przez producenta maksymalna liczba operacji zmiennoprzecinkowych wykonywanych na sekundę: 9,216.

Należy zaznaczyć, że określenie różnicy w wydajności kart wyłącznie na postawie porównania ich parametrów nie jest możliwe, ponieważ między architekturami Turing oraz Ampere występują istotne różnice. Jako jedną z ważniejszych przyczyn podać można deklarowaną przez producenta dwukrotnie większą przepustowość elementów Tensor Cores w kartach opartych na nowszej z architektur.

Wyniki pomiarów zawarte zostały w tabeli 1. Osiągnięcie zakładanych trzydziestu odświeżeń w ciągu sekundy wymaga wykonania obliczeń w czasie mniejszym niż 33,30 ms. Najwyższa z zanotowanych wartości wyniosła 29 ms. Oznacza to, że obie badane stacje robocze były w stanie dokonać obliczeń dostatecznie szybko, by możliwe było spełnienie zakładanych trzydziestu odświeżeń w ciągu sekundy.

Wykorzystywane przez nie karty graficzne znacząco różnią się parametrami, w związku z czym zauważalna jest istotna różnica w czasie ewaluacji sieci na badanych stacjach. W obu przypadkach karty dysponowały znaczącym zapasem wolnej pamięci. W przypadku karty 3080 znana jest deklarowana przez producenta wartość TDP. Porównując ją z wartościami zmierzonymi podczas pracy można zauważyć, że została ona przekroczona. Pozwala to przypuszczać, że czynnikiem

ograniczającym wydajność karty jest moc obliczeniowa kart, nie jest nim pamięć ani przepustowość żadnego z elementów. Warto zauważyć również znacznie większe taktowanie kart podczas pracy w trybie Boost niż wartość deklarowana dla producenta.

Prosta symulacja wykonywana jest 3,76 razy szybciej przy użyciu karty graficznej GeForce RTX 3080, jednak w przypadku symulacji skomplikowanej (wykonywanej przy pomocy mniej złożonej sieci) różnica jest mniejsza i wynosi 2,49 raza.

Tabela 1. Zmierzone dane opisujące wydajność działania sieci na kartach GeForce RTX 3080 oraz GeForce RTX 2060 Mobile dla dwóch wariantów symulacji.

Wariant	Karta	Czas obliczeń [ms]	Użycie GPU[%]	Użycie pamięci[%]	Taktowanie GPU[MHz]	Moc[W]
Prosty	3080	7,46	81.64	26.53	1923	334
Prosty	2060 Mobile	28,00	90.30	41.65	1680	89
Złożony	3080	4.85	76.52	19.29	1918	327
Złożony	2060 Mobile	12.07	88.39	24.44	1727,5	87

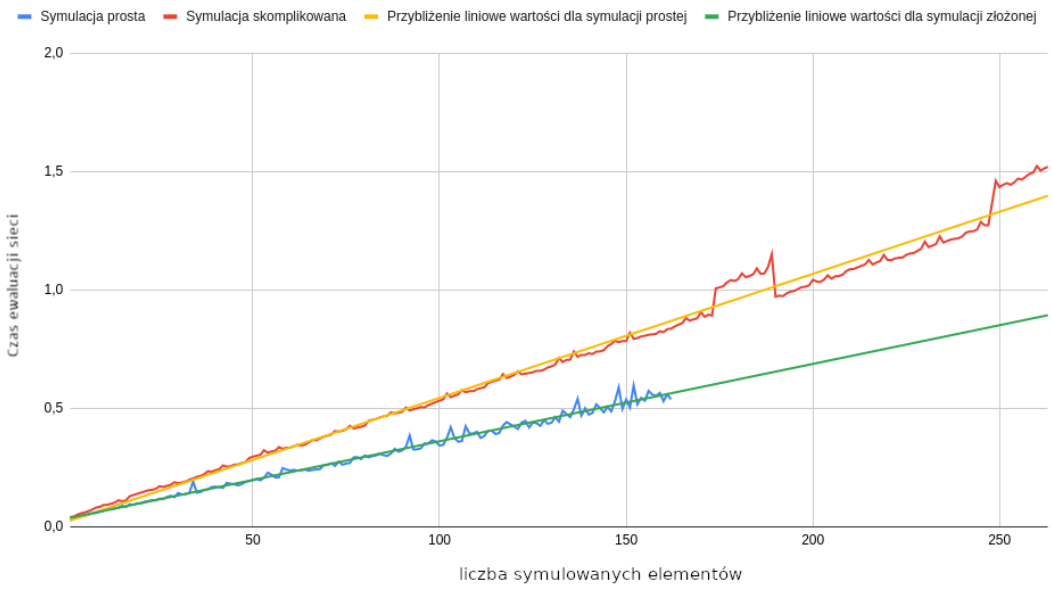
Wykres widoczny na rys. 26 prezentuje wpływ liczby wykonywanych jednocześnie symulacji na czas ewaluacji sieci. Zwiększenia liczby wykonywanych symulacji dokonano poprzez rozbudowę danych wejściowych. Dane zebrano jedynie przy użyciu stacji roboczej wyposażonej w kartę GeForce RTX 3080. Widoczne na wykresie anomalie polegające na czasowym zwiększeniu się czasu ewaluacji sieci wynikają prawdopodobnie z chwilowego zmniejszenia taktowania karty graficznej w związku z nadmiernym jej obciążeniem.

Zauważalna jest wyraźnie liniowa zależność między tymi wartościami. Przybliżenie liniowe wartości uzyskanych dla wariantu sieci odpowiedzialnego za złożoną symulację określa wzór 1 natomiast przybliżenie liniowe wartości uzyskanych dla wariantu sieci odpowiedzialnego za prostą symulację określa wzór 2.

$$\mathrm{y} = 3,66 * 10^{-2}\mathrm{x} + 3,26 * 10^{-3} \tag{1}$$

$$\mathrm{y} = 5,23 * 10^{-3}\mathrm{x} + 2,26 * 10^{-2} \tag{2}$$

Oznacza to, że zwiększenie liczby wykonywanych równocześnie symulacji z jednej do dwóch zwiększa czas ewaluacji sieci o 19% lub 8% odpowiednio dla symulacji prostej oraz złożonej. Przekroczenie dwukrotnego zwiększenia czasu ewaluacji sieci ma miejsce przy wykonywaniu siedmiu lub czternastu symulacji równocześnie odpowiednio dla symulacji prostej oraz złożonej. Zjawisko to można uznać za zaletę przedstawianego rozwiązania, ponieważ wskazuje ono na możliwość wykonywania wielu symulacji w czasie zbliżonym do czasu, który zajmuje wykonywanie pojedynczej symulacji.



Rysunek 26. Przedstawienie zależności pomiędzy czasem ewaluacji sieci oraz liczba wykonywanych symulacji. Źródło: opracowanie własne.

2.6 Porównanie wyników z implementacją opartą na narzędziach dostępnych w edytorze Unreal Editor 4

2.6.1 Przygotowanie implementacji

W tej części pracy przedstawiono proces tworzenia symulacji opartej na mechanizmach zaimplementowanych przez producenta silnika Unreal Engine. Opisano też zasadę działania tej implementacji.

Wraz z silnikiem Unreal Engine 4 dystrybuowana jest wtyczka „Water", tworzona przez producenta oprogramowania, firmę Epic Games. Wtyczka oznaczona jest jako eksperymentalna, co oznacza, że obecnie nie nadaje się do użycia w produkcji, powinna być wykorzystywana jedynie w celach badawczych oraz w celu udzielenia twórcom informacji zwrotnej na temat zawartych w niej mechanizmów. Wtyczka zawiera złożoną implementację powierzchni wody pozwalającą na szybkie umieszczenie jezior, rzek lub innych zbiorników wodnych. Kształt powierzchni wody wyznaczany jest za pomocą parametrów jej materiału. Wśród plików dostarczanych razem ze wtyczką znajdują się przykłady przedstawiające sposób, w jaki możliwe jest stworzenie interaktywnej symulacji w oparciu o ten system. W dalszej części pracy postanowiono porównać rezultaty działania proponowanego rozwiązania z rezultatami uzyskanymi poprzez rozbudowanie jednego z tych przykładów. Wykonanie implementacji rozpoczęto od stworzenia obiektu reprezentującego powierzchnię wody, w tym celu wykorzystano element o klasie WaterBodyOcean zaimplementowany przez omawianą wtyczkę. Utworzono również obiekt reprezentujący przeszkodę klasy „character" o odpowiedniej geometrii.

Zaimplementowano mechanizmy pozwalające na kontrolę poruszania się aktora w sposób podobny do tego, który wykorzystywany jest w demonstracji implementacji opartej o działanie sieci neuronowej. Materiał powierzchni wody zmodyfikowano tak, by wyeliminować z niego fale powstające na skutek działania wiatru, ponieważ rozpatrywana jest jedynie symulacja jej interakcji w kontakcie z przeszkodą, w związku z czym obecność innych deformacji powierzchni wody utrudniłaby porównanie.

Zawarty wśród plików wtyczki przykład implementacji interaktywnej powierzchni wody w wersji silnika używanej podczas pisania pracy do poprawnego działania wymaga wprowadzenia pewnych zmian w plikach znajdujących się wewnątrz katalogu WaterContent:

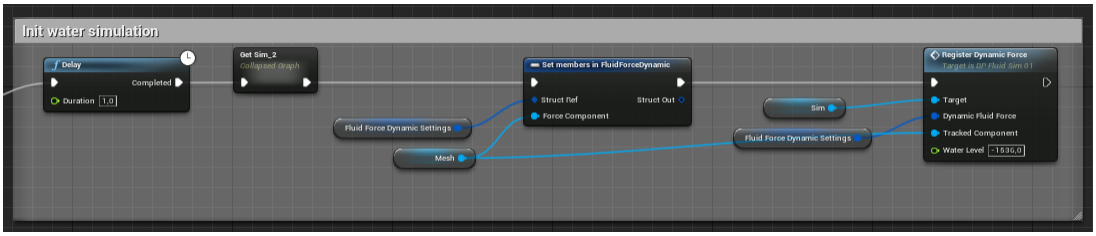
1. W pliku `FluidSimulation/Materials/Simulation/M_Fluid_Sim_01` należy zmienić parametr FoamForceRT w taki sposób, by wskazywał na obiekt :

   `/Water/FluidSimulation/WaveFoam/FoamRT`.

2. W grafie reprezentującym obiekt przeszkody należy stworzyć funkcję inicjalizującą symulację powierzchni wody. W tym celu przekopiować można sekwencję zaimplementowaną w przykładzie zawartym w pliku

   `FluidSimulation/Blueprints/Examples/BP_Dynamic_Force_Skeletal_Mesh`.

Należy również zapewnić istnienie wymaganych przez tę sekwencję komponentów w obiekcie przeszkody. Przebieg tej sekwencji wraz z wymaganymi komponentami przedstawiono na rys. 27. Należy przy tym pamiętać, by zmienna używana jako parametr „Force Component" wskazywała na odpowiedni element obiektu reprezentującego przeszkodę, czyli jego reprezentację graficzną. Największa część procesu inicjalizacji zawarta jest w elemencie grafu o nazwie „Register Dynamic Force", który odpowiedzialny jest za zapisanie informacji o tym, że dany obiekt powinien mieć wpływ na symulację.



Rysunek 27. Wspomniana w tekście sekwencja elementów grafu przeszkody. Źródło: opracowanie własne.

2.6.2 Zasada działania implementacji

W tej sekcji pracy przedstawiono uproszczoną zasadę działania omawianej implementacji symulacji powierzchni wody. Proces symulacji odbywa się z użyciem danych zapisanych podczas inicjalizacji w grafie o nazwie „Event Tick" blueprintu `,,BP_FluidSim_01''`. Symulacja wykonywana jest przy użyciu obiektów typu „Render Target", będących wirtualnymi i łatwymi w modyfikacji

z poziomu blueprintów teksturami, oraz specjalnego materiałów zawierających kod odpowiedzialny za przeprowadzenie symulacji. Za pomocą blueprintów ustawiane są parametry poszczególnych materiałów odpowiedzialnych za przeprowadzanie poszczególnych etapów symulacji, takich jak propagacja wysokości powierzchni wody. Rezultaty działania materiałów są następnie zapisywane we wspomnianych render targetach.

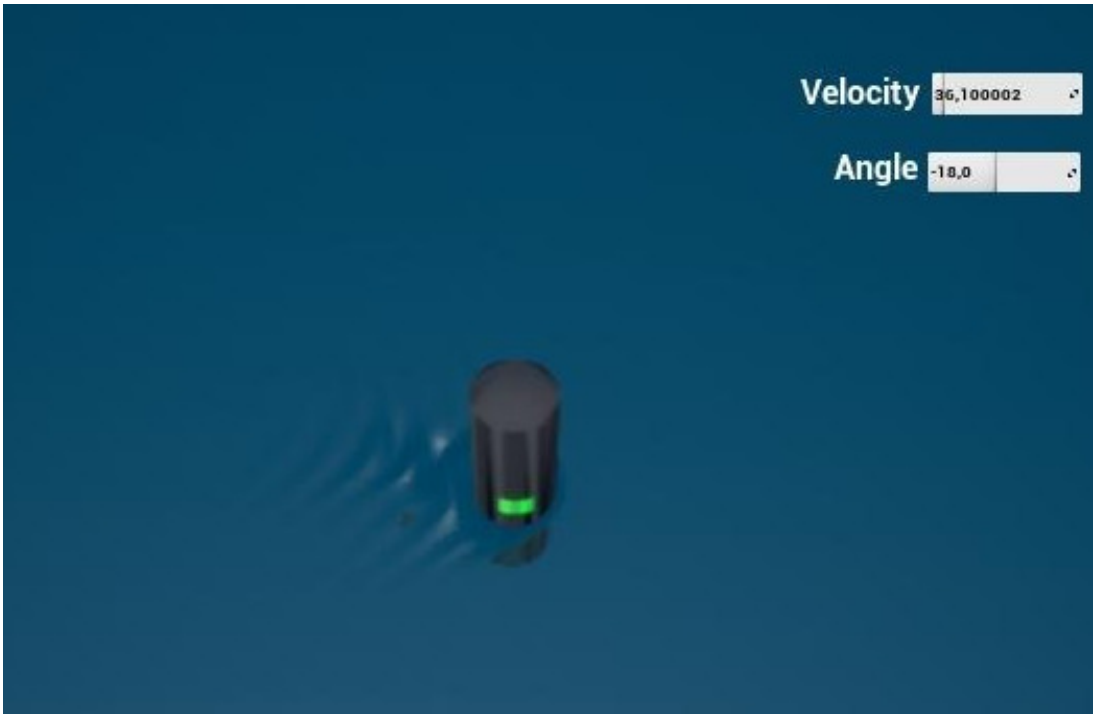
Symulacja opiera się na prostej propagacji fali w celu symulacji śladu torowego. Nie uwzględnia kształtu obiektu, jedynie jego położenie oraz przesunięcie w czasie. Za pomocą tych dwóch wartości ustalany jest wpływ, jaki obiekt powinien mieć na powierzchnię wody oraz miejsce, w którym wpływ ten powinien wywrzeć. Wpływ ten przemnażany jest przez wartość funkcji sinus czasu rzeczywistego. Wartość ta wykorzystywana jest w celu zmiany wysokości powierzchni wody w punkcie, w którym oddziaływać powinna siła. W każdej klatce następuje propagacja tej wartości na kolejne piksele zgodna z przyjętą prędkością poruszania się fali.

2.6.3 Przedstawienie wyników

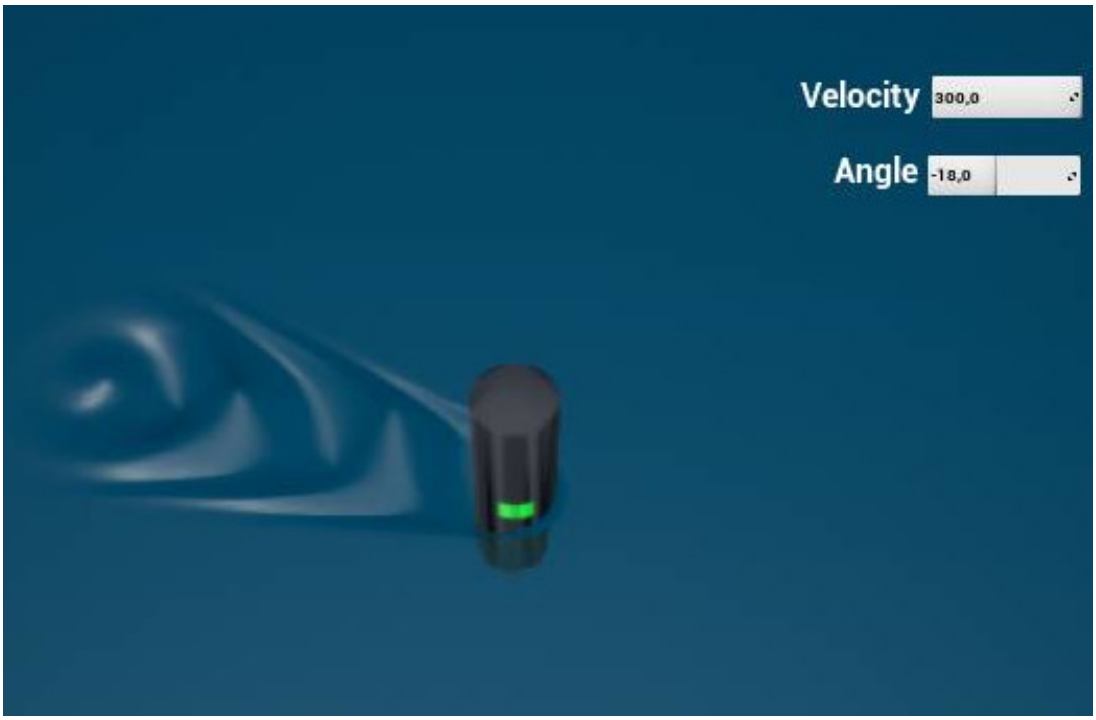
W tej sekcji pracy przedstawiono porównania rezultatów działania implementacji symulacji powierzchni wody opartej na pluginie „Water" podczas symulacji podobnych warunków. Zdecydowano się na taką formę porównania ich. ponieważ obiektywna ocena jakości działania implementacji nie jest możliwa. Możliwe jest jednak wskazanie kluczowych różnic oraz mocnych stron poszczególnych rozwiązań.

Ruch w linii prostej z niewielką prędkością

Rezultaty działania obu sieci podczas ruchu w linii prostej z niewielką prędkością przedstawia rys. 28 oraz rys. 29. Przy niewielkiej prędkości obie implementacje generują niewielkie fale. W rezultatach działania implementacji opartej na wtyczce „Water" widoczny jednak zaczyna być powtarzalny wzór przypominający widoczny za statkami ślad torowy.



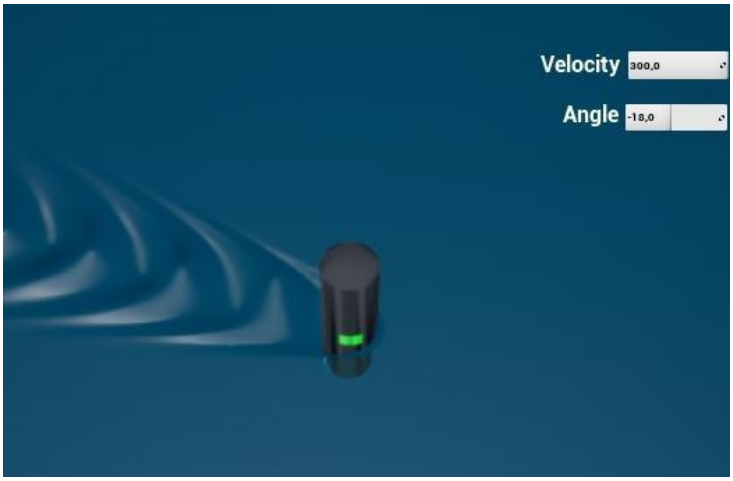
Rysunek 28. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne.



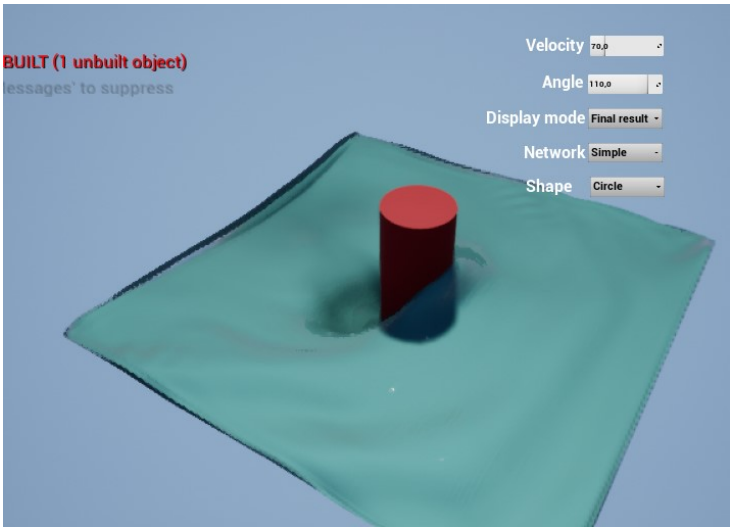
Rysunek 29. Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne.

Ruch linii prostej z większą prędkością

Rezultaty działania obu sieci podczas ruchu w linii prostej ze znaczną prędkością przedstawia rys. 30 oraz rys. 31. Widoczne są znaczne różnice w wygenerowanych powierzchniach. Implementacja oparta na sieciach neuronowych cechuje się dużym spiętrzeniem wody przed przeszkodą oraz sporym spadkiem jej wysokości za nią. Powierzchnia wody jest znacznie bardziej skomplikowana. W przypadku implementacji opartej na wtyczce „Water" widoczny jest dłuższy ślad torowy, niż w przypadku niższej prędkości, jednak symulacja jest bardzo prosta i podobna do tej zaprezentowanej wcześniej. Zauważalną różnicą jest powierzchnią, którą obejmuje symulacja w obu wypadkach: powierzchnia symulowana za pomocą sieci neuronowej jest znacząco mniejsza.



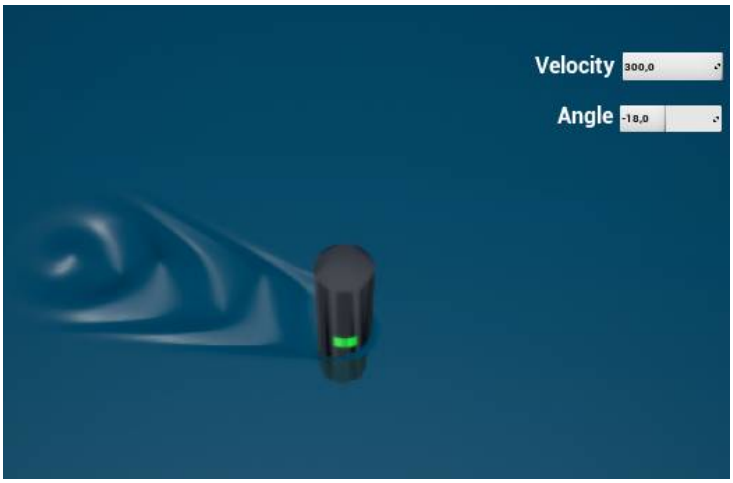
Rysunek 30. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne.



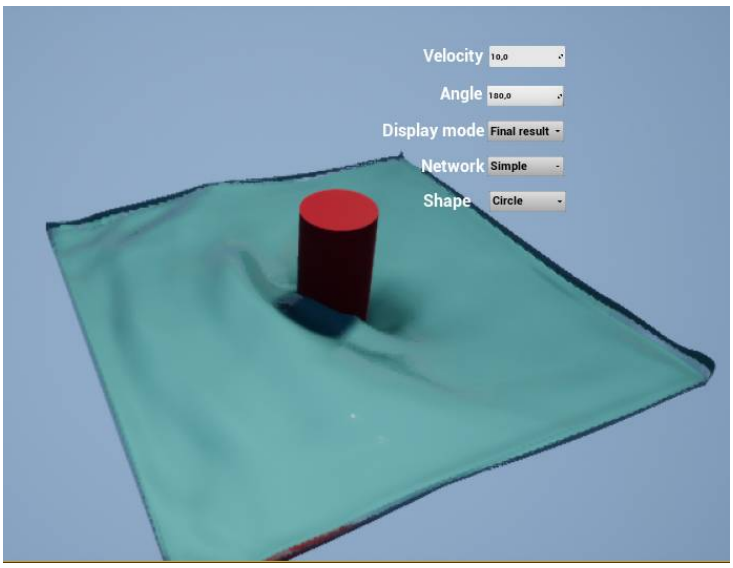
Rysunek 31. Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne.

Szybkie zmniejszenie prędkości

Rezultaty działania obu sieci podczas ruchu w linii prostej przy znacznym zmniejszeniu prędkości przedstawia rys. 32 oraz rys. 33. Rezultat generacji powierzchni wody przez sieć neuronową cechuje się szybkim spadkiem różnic wysokości wody przed i za przeszkodą oraz falą znajdującą się przed przeszkodą, która po jej zatrzymaniu przesuwa się w przód. W przypadku wyników wygenerowanych za pomocą wtyczki „Water" widoczne jest powolne zmniejszanie się długości wzoru śladu torowego. Niewidoczne jest spiętrzenie wody przed przeszkodą ani wspomniana fala ją wyprzedzająca.



Rysunek 32. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu przy znacznym zmniejszeniu prędkości. Źródło: opracowanie własne.



Rysunek 33. Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu przy znacznym zmniejszeniu prędkości. Źródło: opracowanie własne.

Wydajność implementacji

Zaletą implementacji opartej na wtyczce „Water" jest jej duża wydajność. Pomiary czasu wykonywania funkcji „Event Tick" aktora `,,BP_Fluid_Sim_01''` odpowiedzialnej za wykonanie wszystkich kroków symulacji pozwalają stwierdzić, że mediana czasu jej wykonywania wynosi 0,627 ms. Jest to czas wystarczający do osiągnięcia wysokiej liczby klatek na sekundę. Dla porównania, zmierzony czas ewaluacji sieci wynosi od 4,85 ms do 7,45 ms.

Podsumowanie porównania rezultatów działania implementacji

Widoczne są znaczące różnice między wygenerowanymi rezultatami. Generacja oparta na wtyczce „Water" cechuje się dużą prostotą: generowany jest jedynie prosty ślad torowy za przeszkodą. Nie jest zauważalne spiętrzenie wody przed nią. Gwałtowne zwiększenie lub zmniejszenie prędkości nie powoduje znaczących zmian na powierzchni wody, skutkuje jedynie powolną zmianą długości generowanego wzoru.

Rezultat działania rozwiązania opartego na sieci neuronowej cechuje się bardzo dynamiczną, bogatą w szczegóły, powierzchnią. Szybkie zmiany prędkości poruszania się przeszkody powodują zauważane i gwałtowne zmiany na powierzchni wody. Rezultat jest jednak znacznie mniej stabilny. Niewielka powierzchnia generowania symulacji wymusza brak widocznego śladu torowego.

Rozdział 3

Podsumowanie i wnioski

W niniejszej pracy magisterskiej zostało zaproponowane rozwiązanie pozwalające na symulację powierzchni wody w otoczeniu przeszkody z wykorzystaniem głębokich sieci neuronowych i silnika Unreal Engine 4. Podczas prac osiągnięto wszystkie zakładane cele: cel główny, którym było stworzenie aplikacji zawierającej interaktywną symulację powierzchni wody, oraz cel badawczy, który polegał na wykorzystaniu sieci neuronowej w celu wykonania symulacji. Sama aplikacja została stworzona jako cel ukryty pracy, pozwalając na porównanie proponowanego rozwiązania z już istniejącymi. Pracę rozpoczęto od przedstawienia wiedzy wstępnej dotyczącej symulacji powierzchni wody. Przedstawiono rys historyczny zagadnienia oraz popularne obecnie rozwiązania tego problemu. Opisano również zasady działania sieci neuronowych, przedstawiono zakres, w jakim obecnie wykorzystywane są one w ramach zagadnienia symulacji cieczy oraz w ramach innych zagadnień. Zostały również omówione silniki gier, w tym Unreal Engine 4, oraz sposoby ich integracji z sieciami neuronowymi.

Następnie przedstawiono konkretne kroki realizacji celu pracy, w tym przygotowanie symulacji referencyjnej, wykorzystywanej w celu szkolenia sieci, szczegółowy opis integracji biblioteki Tensorflow z silnikiem Unreal Engine 4 za pomocą biblioteki cppflow oraz tworzenie i trenowanie samej sieci neuronowej. Pokazano również szczegóły dotyczące interaktywnej aplikacji stworzonej jako cel ukryty pracy. Kod źródłowy aplikacji udostępniono pod aresem: https://github.com/p4vv37/ueflow.

W pracy zawarto również wyniki i analizę osiągniętych rezultatów. Przedstawiono wyniki symulacji powierzchni wody z wykorzystaniem sieci neuronowej oraz porównano je z wynikami uzyskanymi dla innych metod symulacji. Na podstawie uzyskanych danych można stwierdzić, że implementacja oparta na sieciach neuronowych uruchomiona przy użyciu karty graficznej jest w stanie zapewnić wystarczającą wydajność dla zastosowań praktycznych. Czas ewaluacji sieci wynoszący w omawianym przypadku od 4,85 ms do 7,45 ms jest znacząco niższy niż czas ewaluacji pojedynczej ramki przy prędkości odświeżania 60 klatek na sekundę wynoszący 16,(6) ms. Jest to wartość umożliwiająca znacznie wyższą częstotliwość odświeżania niż zakładana początkowo wartość 30 klatek na sekundę.

Na podstawie osiągniętych rezultatów można stwierdzić, że zaproponowane rozwiązanie pozwala na uzyskanie wysokiej jakości symulacji powierzchni wody przy zachowaniu odpowiedniej wydajności. Aplikacja stworzona jako cel ukryty pracy umożliwia interaktywne korzystanie z symulacji oraz

porównanie wyników działania zaproponowanej sieci neuronowej z innymi metodami symulacji powierzchni wody.

3.1 Dalsze prace

Osiągnięte w ramach pracy magisterskiej rezultaty pozwalają na wskazanie kierunków dalszych prac, które mogą prowadzić do poprawy jakości symulacji powierzchni wody. Poniżej przedstawiono kilka propozycji takich działań:

1. Oparcie implementacji na zasadzie modeli dyfuzyjnych. Gwałtowny rozwój modeli dyfuzyjnych takich jak Stable Diffusion, udowodnił ich zdolność do generacji wysokiej jakości bitmap. Rozwiązania oparte na takiej architekturze mogą dodatkowo pozwolić na ułatwienie zmiany jakości rozwiązania kosztem wydajności poprzez kontrolę liczby kroków tworzenia bitmapy, podobnie jak ma to miejsce w przypadku sieci Stable Diffusion.

2. Zwiększenie powierzchni symulacji wody wokół przeszkody może pozwolić na zauważenie na niej ciekawych zjawisk oraz zwiększenie użyteczności rozwiązania.

3. Zmiana siatki z regularnej na nieregularną, o zwiększonej gęstości w pobliżu przeszkody i zmniejszonej w większej odległości od niej może pozwolić na zwiększenie symulowanego obszaru bez zwiększania złożoności obliczeniowej sieci.

4. Wykorzystanie biblioteki TFLite. Taki krok może pozwolić na wykorzystanie rozwiązania na urządzeniach mobilnych.

5. Uwzględnienie większej liczby parametrów. W chwili obecnej symulacja uwzględnia jedynie prędkość poruszania się wody. Możliwe jest jednak rozszerzenie symulacji o dodatkowe parametry, takie jak odległość od dna, pozwalająca na symulację wody przy linii brzegowej lub w otoczeniu innych przeszkód.

Bibliografia

[1] Hennigh, O., *Computational-Fluid-Dynamics-Machine-Learning-Examples*, https://github.com/loliverhennigh/Computational-Fluid-Dynamics-Machine-Learning-Examples data dostępu: 14.12.2021.

[2] Kowalski, P., *Zmiany dokonane w module Pyradox na potrzeby wykonania pracy.* https://github.com/Ritvik19/pyradox/commit/88eafb5dcd44c332f92c4f6d6da756016a2d6518 data dostępu: 13.09.2023.

[3] Rao, C., *PINN-laminar-flow*, https://github.com/Raocp/PINN-laminar-flow data dostępu: 14.12.2021.

[4] Rastogi, R., *Repozytorium modułu Pyradox zawierającego implementacje architektur sieci neuronowych.* https://github.com/Ritvik19/pyradox data dostępu: 13.09.2023.

[5] Ronneberger, O., Fischer, P. i Brox, T., „U-Net: Convolutional Networks for Biomedical Image Segmentation", *arXiv:1505.04597*, 2015.

[6] SideFX, *Opis narzędzia Flat Tank*, https://www.sidefx.com/docs/houdini/shelf/flattank.html data dostępu: 13.09.2023.

[7] TensorFlow, *Dokumentacja TenssorFlow: Prognozowanie szeregów czasowych*, https://www.tensorflow.org/tutorials/structured_data/time_series data dostępu: 13.09.2023.

[8] www.techpowerup.com, *NVIDIA GeForce RTX 3080.* https://www.techpowerup.com/gpu-specs/geforce-rtx-3080.c3621 data dostępu: 13.09.2023.