Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Sterowania i Elektroniki Przemysłowej

Praca dyplomowa magisterska

na kierunku Informatyka Stosowana

w specjalności Inżynieria oprogramowania

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

Paweł Kowalski
numer albumu 301953

promotor
dr inż. Witold Czajewski

Warszawa 2023

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

Streszczenie

Niniejsza praca magisterska dotyczy zagadnienia interaktywnej symulacji powierzchni wody z wykorzystaniem głębokich sieci neuronowych. Na początku przedstawiono historyczne i współczesne rozwiązania problemu symulacji powierzchni wody, a następnie zaproponowano rozwiązanie oparte na sieciach neuronowych. Do nauki sieci wykorzystano zbiór danych wygenerowany za pomocą narzędzi symulacji cieczy dostępnych w aplikacji SideFX Houdini. W celu przygotowania danych uczących konieczne było stworzenie skryptu w języku Python odpowiedzialnego za nadzór nad procesem generacji danych, wstępną obróbkę rezultatów symulacji oraz zapis danych. Dane opisujące kształt powierzchni wody zapisane są w formie dwuwymiarowych tablic, traktowanych podczas procesu szkolenia sieci jako bitmapy. Do realizacji celu pracy wykorzystano sieć neuronową opartą na architekturze U-Net, stosowaną powszechnie w celu przetwarzania obrazów, a w procesie szkolenia sieci zastosowano koncepcję generatywnych sieci współzawodniczących. W ramach pracy stworzono interaktywną aplikację wykorzystującą silnik gier Unreal Engine 4, pozwalającą na ocenę jakości wyników działania sieci oraz pomiar wydajności zaproponowanego rozwiązania. Praca zawiera jakościową oraz wydajnościową analizę uzyskanych rezultatów, a także porównanie uzyskanych rezultatów z wybranym wiodącym rozwiązaniem. Osiągnięte wyniki wskazują, że zaproponowane rozwiązanie pozwala na uzyskanie wysokiej jakości symulacji powierzchni wody przy zachowaniu odpowiedniej wydajności.

Słowa kluczowe: symulacja powierzchni wody, splotowe sieci neuronowe, generatywne sieci neuronowe, Unreal Engine

Using deep neural networks to simulate the water surface surrounding an obstacle using Unreal Engine 4

Abstract

This master's thesis focuses on the topic of interactive water surface simulation using deep neural networks. Initially, historical and contemporary solutions to the water surface simulation problem are presented, followed by a proposal based on neural networks. A dataset for network training was generated using fluid simulation tools available in the SideFX Houdini application. To prepare the training data, a Python script was created to perform the data generation process, pre-processing of simulation results, and save the data. Data describing the shape of the water surface is stored in the form of two-dimensional arrays, treated as bitmaps during the network training process. To achieve the thesis goal, a neural network based on the U-Net architecture, commonly used for image processing, was employed, and the concept of generative adversarial networks was applied during the network training process. As part of the thesis, an interactive application utilizing the Unreal Engine 4 game engine was developed to assess the quality of network results and measure the performance of the proposed solution. The thesis includes qualitative and performance analysis of the obtained results and compares them with a selected leading solution. The achieved results indicate that the proposed solution allows for high-quality water surface simulation while maintaining adequate performance."

Keywords: water surface simulation, neural networks, generative adversarial networks, Unreal Engine

Spis treści

1 Wstęp **1**
1.1 Wprowadzenie do tematyki pracy . . . 1
1.2 Cel i układ pracy . . . 2

2 Przegląd istniejących rozwiązań **3**
2.1 Wczesne metody symulacji powierzchni wody . . . 3
2.2 Modyfikacja wektorów normalnych . . . 6
2.3 Metody nieinteraktywne . . . 7
2.3.1 Wykorzystanie tekstur . . . 7
2.3.2 Rozwiązania oparte na funkcjach trygonometrycznych . . . 8
2.3.3 Fala trochoidalna . . . 9
2.3.4 Szum Perlina . . . 12
2.3.5 Szybka Transformata Fouriera . . . 12
2.4 Metody interaktywne . . . 15
2.4.1 Rozwiązania oparte na cząsteczkach . . . 15
2.4.2 Równania Naviera-Stokesa . . . 16
2.4.3 Równania falowe . . . 18
2.5 Rozwiązania wykorzystujące Głębokie Sieci Neuronowe . . . 19
2.6 Symulacja wody we współczesnych grach komputerowych . . . 22
2.7 Wtyczka Fluid Flux . . . 24

3 Wprowadzenie do sieci neuronowych **25**
3.1 Budowa sieci neuronowej . . . 25
3.2 Splotowe sieci neuronowe . . . 26
3.3 Generatywne sieci współzawodniczące . . . 27

4 Opis koncepcji rozwiązania problemu **31**
4.1 Opis wykorzystanych narzędzi i bibliotek . . . 32
4.1.1 Unreal Engine . . . 32
4.1.2 SideFX Houdini . . . 32
4.1.3 Jupyter Notebook . . . 33
4.1.4 Shader i Fragment Shader . . . 33

4.1.5 Python . . . 34
4.1.6 Tensorflow, Keras oraz Tensorboard . . . 34
4.1.7 CPPFlow . . . 35
4.2 Konfiguracja sprzętowa komputera . . . 36
4.3 Przygotowanie środowiska . . . 36

5 Przeprowadzone prace **39**
5.1 Prosta symulacja powierzchni wody . . . 39
5.1.1 Przygotowanie symulacji powierzchni wody . . . 39
5.1.2 Generacja zbioru danych . . . 40
5.1.3 Wprowadzone modyfikacje, uzyskanie zadowalających wyników . . . 45
5.2 Złożona symulacja powierzchni wody . . . 48
5.3 Trenowanie sieci neuronowej . . . 53
5.4 Implementacja sieci w silniku Unreal Engine 4 . . . 54
5.5 Przedstawienie oraz analiza wyników . . . 56
5.5.1 Przedstawienie wyników . . . 56
5.5.2 Analiza jakościowa wyników . . . 59
5.5.3 Analiza wydajnościowa wyników . . . 61
5.6 Porównanie wyników z implementacją opartą na narzędziach dostępnych w edytorze Unreal Editor 4 . . . 64
5.6.1 Przygotowanie implementacji . . . 64
5.6.2 Zasada działania implementacji . . . 65
5.6.3 Przedstawienie wyników . . . 66

6 Podsumowanie i wnioski **71**
6.1 Dalsze prace . . . 72

Bibliografia **73**

Wykaz skrótów i symboli **79**

Spis rysunków **81**

Spis tabel **85**

Rozdział 1

Wstęp

1.1 Wprowadzenie do tematyki pracy

Ostatnie lata to czas bardzo dynamicznego rozwoju sieci neuronowych. Kategoryzacja zdjęć przez sieci splotowe jest dziś bardziej precyzyjna, niż wykonywana przez człowieka [68], sieć Alpha Go udowodniła swoją wyższość nad ludzkim przeciwnikiem w niezwykle skomplikowanej grze Go [19], inne liczne inne projekty, dzięki uczeniu przez wzmacnianie, czynią to samo w świecie popularnych gier komputerowych, jak Breakout czy Space Invaders [39]. Firma Google odniosła znaczące sukcesy w dziedzinie generacji i zrozumienia mowy oraz tekstu, co zaprezentowała podczas szeroko opisywanej w mediach prezentacji w czasie Google I/O '18 [39]. Dokonano tego dzięki modelowi językowemu GPT-3, którego możliwości oraz elastyczność zaowocowały powstaniem znaczącej liczby opartych na nim projektów [3]. Sztuczna inteligencja generuje i wspomaga również tworzenie sztuki [58]. Coraz bardziej skomplikowane generatywne sieci współzawodniczące tworzą fotorealistyczne obrazy, których odróżnienie od prawdziwych zdjęć bywa niemożliwe [15]. W istocie oznacza to, że sieci są w stanie odnajdywać leżące w obrazach zależności, których złamanie zauważyłby człowiek. W opinii autora sugeruje to, że sieci takie powinny być zdolne do uczynienia tego samego w dziedzinie symulacji zjawisk fizycznych.

Symulacje komputerowe zjawisk fizycznych są technologią rozwijaną od wielu dekad. Narzędzia takie jak Metoda Elementów Skończonych nieodwracalnie zmieniły pracę inżynierów na całym świecie. Podobnie usprawnienie symulacji cieczy pozwoliło na rozwój aerodynamiki i lepsze zrozumienie obciążeń, na jakie narażone będą wystawiane na działanie wiatrów czy płynów obiekty [46]. Technologie te stosowane są również w przemyśle rozrywkowym oraz w branży gier komputerowych. Wykorzystywane w nich animacje procesów fizycznych, nie muszą co prawda cechować się idealnym odwzorowaniem tych procesów, jak to ma miejsce w przypadku zagadnień związanych z inżynierią, jednak stawiane są przed nimi inne wyzwania, którym również trudno sprostać. Symulacje muszą wyglądać na tyle realistycznie, żeby nie stać się obiektem żartów odbiorców. Jednocześnie muszą spełniać wymogi wizji dyrektora artystycznego odpowiedzialnego za wygląd projektu. Oznacza to, że muszą być wysoce elastyczne. Kierunkiem wyznaczonym przez dyrektora może być na przykład

spowolnienie opadania grzbietu fali wody, zwiększenie lub zmniejszenie ilości piany na wodzie, rozpryskujących się kropel, czy dodanie kuli ognia do symulowanego wybuchu lub skierowanie odłamków w konkretnym kierunku. Symulacje są zazwyczaj zależne od otoczenia, w którym mają miejsce w świecie gry, muszą więc często być generowane w czasie rzeczywistym, w trudnych do przewidzenia i niekontrolowanych warunkach. W takiej sytuacji nie jest wykonalne przygotowanie każdego możliwego ich wariantu przez artystę — przynajmniej część obliczeń wykonywana musi być podczas gry. Skutkuje to kolejnym ważnym ograniczeniem: symulacje muszą być bardzo wydajne, stosowanie metody elementów skończonych lub symulacji cieczy opartej na dużej liczbie cząsteczek jest więc problematyczne. W związku z tym stosowane są liczne uproszczenia geometrii i algorytmów, wykorzystywane są przybliżenia i przygotowane wcześniej animacje.

O ile fizyka zachowania brył sztywnych doczekała się rozwiązania przyjętego jako standard w branży gier, jakim jest Nvidia PhysX [43], nie można tego samego powiedzieć o żadnym z aspektów symulacji cieczy. Złożoność problematyki, omówione ograniczenia i zależne od projektu wymagania względem jakości i wyglądu symulacji skutkują powstaniem niezliczonej liczby rozwiązań. Projektanci zazwyczaj zmuszeni są wymyślać i implementować własne narzędzia, dostosowane do swojej specyficznej sytuacji. Nie istnieje szeroko przyjęte rozwiązanie będące standardem w branży. Ta różnorodność stwarza pewien problem. W jej efekcie nie można więc wyłonić pojedynczej, najważniejszej, która może służyć za rozwiązanie referencyjne dla implementacji opisanej w dalszej części pracy. Można jednak przedstawić wady i zalety tego rozwiązania w porównaniu z różnymi istniejącymi implementacjami.

1.2 Cel i układ pracy

Głównym celem pracy jest stworzenie aplikacji umożliwiającej interaktywną symulację powierzchni wody. Celem badawczym oraz najważniejszym elementem pracy jest zbudowanie i wytrenowanie sieci neuronowej zdolnej do generacji realistycznie wyglądającej symulacji powierzchni wody w okolicy poruszającego się obiektu. Sama interaktywna aplikacja stanowi natomiast cel ukryty, którego realizacja umożliwia ocenę jakości rozwiązania oraz zebranie danych dotyczących jego wydajności. Stworzenie aplikacji pozwala również na porównanie proponowanego rozwiązania z już istniejącymi.

Praca zaczyna się od wprowadzenia do tematyki zawierającego motywację, cel i zakres pracy. W rozdziale 2. dokonano przeglądu istniejących rozwiązań problemu symulacji powierzchni wody, omawiając ich zalety oraz wady. W rozdziale 3. wprowadzono podstawowe pojęcia związane z sieciami neuronowymi, takie jak budowa, rodzaje, algorytmy uczenia i zastosowania. W rozdziale 4. opisano koncepcję rozwiązania problemu symulacji powierzchni wody przy użyciu głębokich sieci neuronowych i silnika Unreal Engine 4, użyte narzędzia i biblioteki oraz konfigurację sprzętową komputera. W rozdziale 5. zaprezentowano przeprowadzone prace nad realizacją rozwiązania problemu symulacji powierzchni wody przy użyciu głębokich sieci neuronowych i silnika Unreal Engine 4, opisując kolejne etapy procesu oraz uzyskane wyniki. W rozdziale 6. podsumowano i oceniono uzyskane rezultaty oraz zaproponowano kierunki dalszych prac nad tematem.

Rozdział 2

Przegląd istniejących rozwiązań

Reprezentacja wody w grach komputerowych od zawsze stanowiła wyzwanie. W ciągu dekad istnienia przemysłu gier powstało wiele podejść do tego aspektu świata przedstawionego w grze. Problem jest więc bardzo złożony. Pewnej jego kategoryzacji dokonać można wyznaczając następujące trzy kategorie:

1. Stworzenie odpowiednio wyglądającej powierzchni cieczy niewymagającej interakcji z otoczeniem, na przykład powierzchnia rzeki widoczna z poziomu mostu.

2. Stworzenie powierzchni cieczy zdolnej do interakcji z obiektami. Jako przykład tej kategorii przedstawić można płynięcie łodzią po oceanie, wbieganie do jeziora lub strzelanie w stronę powierzchni wody.

3. Stworzenie odpowiednio wyglądającej animacji zachowania cieczy w przestrzeni trójwymiarowej. Do tej kategorii zaliczyć można animację przelewania płynu między pojemnikami czy wypełnianie niewielkiego pomieszczenia wodą.

W pracy tej zaproponowane zostaje rozwiązanie problemu należącego do drugiej z wymienionych kategorii. W celu właściwego zrozumienia zagadnienia nakreślona zostanie jednak problematyka wszystkich z nich. Zarysowany zostanie zarazem rozwój wykorzystywanych w nich rozwiązań od najprostszych ich form do najbardziej dziś zaawansowanych ich implementacji.

2.1 Wczesne metody symulacji powierzchni wody

Wczesne gry komputerowe, z uwagi na ograniczenia sprzętowe, cechowały się bardzo uproszczoną grafiką, w związku z tym prosty wzór na niebieskim tle lub tekstura o niskiej rozdzielczości rozłożona na płaskiej powierzchni okazywały się wystarczającą reprezentacją powierzchni wody. Poniżej podano kilka przykładów.

Xevious[22] - wydany w roku 1982, w którym gracz steruje widzianym z góry pojazdem przelatującym nad różnego typu terenami. Woda jest w nim reprezentowana przez niebieski obszar pokryty wzorem złożonym z pojedynczych białych pikseli rozsianych po jego powierzchni (rys. 1).

Rysunek 1. Kadr z gry Xavious(1982 rok), na którym widoczna jest powierzchnia wody przedstawiona przy pomocy niebieskiego obszaru pokrytego białym wzorem. Źródło: [44]

Sonic the Hedgehog[21] - wydany w roku 1991, w którym woda pojawia się jako element tła, a reprezentowana jest przez obszar pokryty animowaną biało-niebieską teksturą (rys. 2)

Rysunek 2. Kadr z gry Sonic the Hedgehog (1991 rok), na którym widoczna jest animowana woda będąca elementem tła. Źródło: [71]

Doom[20] - wydany w roku 1993 tytuł będący jedną z pierwszych „strzelanek pierwszoosobowych" i wykorzystujący rewolucyjny silnik korzystający z grafiki trójwymiarowej, stworzony przez Id Software.

Tytuł, podobnie jak wspominany Sonic, wykorzystywał posiadające kilka klatek animowane tekstury powierzchni płynów, jednak używał ich jako tekstur nałożonych na powierzchnie w przestrzeni trójwymiarowej (rys. 3).

Rysunek 3. Kadr z gry Doom (1993 rok), na którym widoczne są powierzchnie kilku cieczy, zobrazowane za pomocą animowanej tekstury nałożonej na płaszczyznę. Źródło: [2]

Mario 64[42] - wydany roku 1996 tytuł, podobnie jak Doom wykorzystujący grafikę trójwymiarową oraz animowane tekstury, jednak implementujący również pewną formę symulacji powierzchni wody. Sprowadza się ona do wyświetlenia kilku cząsteczek, tak zwanych sprite'ów, naśladujących rozbryzgujące się krople wody, oraz umieszczenie w miejscu gracza i miejscach spotkania kropli z powierzchnią wody decali: specjalnej tekstury nakładanej miejscowo, która powoduje zmianę, koloru wektorów normalnych lub innych parametrów shadera na konkretnym obszarze (rys. 4)

Rysunek 4. Kadry z gry Mario 64 (1996 rok), na którym widoczna jest powierzchnia wody, efekty cząsteczkowe oraz symulacja rozchodzących się fal wody. Źródło: [48]

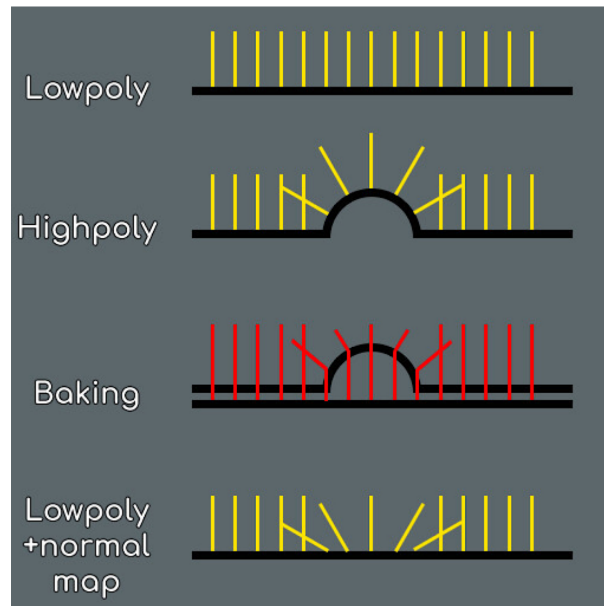
2.2 Modyfikacja wektorów normalnych

Kolejny etap w rozwoju reprezentacji powierzchni wody w grach przedstawia opublikowana w 2001 roku demonstracja silnika CryEngine X-Isle Dinosaurs Island. Była to pod względem użycia zaawansowanych technik graficznych rewolucyjna, jak na owe czasy, produkcja [72]. W celu uzyskania realistycznej powierzchni wody wykorzystano technologię PolyBump [9]. Opiera się na zastosowaniu „mapowania normalnych" [10], czyli technologii zaproponowanej w roku 1996 podczas konferencji SIGGRAPH [6]. Polega ona na zapisaniu wektorów normalnych do powierzchni w formie tekstury. Zazwyczaj poszczególne kanały tej bitmapy: czerwony, niebieski i zielony, przekładają się na wartości współrzędnych x, y oraz ze znormalizowanego wektora normalnego w danym punkcie na powierzchni, przeskalowane do przestrzeni między wartościami 0 a 255, gdzie 128 oznacza wartość zero. Ta informacja wykorzystywana jest przez algorytm cieniujący (shader), który zamiast określać wektor normalny każdego punktu na powierzchni trójkąta interpolując wektory normalne jego wierzchołków, co ma miejsce gdy mapowanie wypukłości (normal mapping) nie jest wykorzystywany, odczytuje informację z mapy i wykorzystuje ją do obliczenia skorygowanego wektora normalnego. Obliczony w ten sposób wektor wykorzystywany jest we fragment shaderze w ten sam sposób, w jaki byłby wykorzystywany interpolowany wektor normalny powierzchni: za jego pomocą oblicza się cieniowanie powierzchni oraz odbicia. W efekcie bryła wydaje się mieć większą liczbę szczegółów, niż pozwalałaby na to jej geometria. Wizualizacja tej technologii została przedstawiona na rys. 5. Żółte linie oznaczają kierunek wektorów normalnych powierzchni. Płaska czarna linia oznacza uproszczoną płaską powierzchnię, linia z wybrzuszeniem oznacza bardziej skomplikowany model. Podczas wypalania wyznaczany jest kierunek wektorów normalnych geometrii skomplikowanej w odpowiadających jej punktach geometrii uproszczonej. Ostatnia część schematu przedstawia uproszczoną powierzchnię oraz wizualizację wypalonych dla niej wektorów normalnych.

W demie technologicznym oraz wydanej kilka lat później, w roku 2004, grze Far Cry, będącej w istocie jego rozwinięciem, woda wciąż jest płaską powierzchnią. Dzięki zastosowaniu technologii map normalnych wydaje się jednak pofalowana (rys. 6). Woda w grze Far Cry jest też interaktywna: reaguje na pociski oraz pływające po niej łodzie. Symulacja ta jednak opiera się wyłącznie na zastosowaniu kalek (częściej określanych przy użyciu anglojęzycznego określenia „decal"), tym razem modyfikujących zarówno kolor, jak i wektory normalne.

Technika polegająca na modyfikacji wektorów normalnych powierzchni wody wykorzystywana jest szeroko w branży gier i dziś możemy ją spotkać w prawie każdej wydawanej produkcji. Ponieważ powstała powierzchnia wciąż jest płaska, co jest łatwe do zauważenia, gdy patrzymy na nią pod ostrym kątem, dziś zazwyczaj stanowi uzupełnienie innych technik tworzących bardziej skomplikowaną geometrię.

Zarówno do reprezentacji powierzchni wody za pomocą samego koloru lub animowanego wzoru, jak i do modyfikacji wektorów normalnych, potrzebne są wcześniej określone wartości, kolory lub kierunki wektorów używane w tym procesie. Do ich uzyskania można zastosować wiele metod.



Rysunek 5. Schematyczne przedstawienie działania mapowania normalnych. Źródło: [35]

2.3 Metody nieinteraktywne

W tej sekcji omówiono metody, których celem jest wygenerowanie w efektywny sposób realistycznie wyglądającej powierzchni wody, jednak bez uwzględnienia żadnej interakcji z nią. Znajomość tych metod jest niezbędna do zrozumienia zagadnienia symulacji powierzchni cieczy oraz oczekiwanych rezultatów tej symulacji. Przedstawienie ich pozwala również na zobrazowanie rozwoju technologii symulacji powierzchni cieczy oraz wyzwań, które z nią się wiążą. W szczególności opisana dalej metoda oparta na szybkiej transformacie Fouriera traktowana być może jako metoda referencyjna, z którą porównywana być może jakość innych rozwiązań. Stanowią one również podstawę niektórych przedstawionych w kolejnym rozdziale rozwiązań interaktywnych.

2.3.1 Wykorzystanie tekstur

Najprostszą metodą jest zastosowanie przygotowanych uprzednio tekstur. Wymaga to jednak wykorzystania dodatkowej pamięci i zwiększa rozmiar samej gry, ponieważ wymagane są dodatkowe pliki, które muszą być razem z grą dystrybuowane. Dodatkowo może to powodować łatwe do zauważenia, powtarzalne wzory. Efekt ten można zniwelować, stosując wiele wzorów o różnym rozmiarze przenikających się w odpowiedni sposób lub poprzez zastosowanie bardzo dużych rozmiarów tekstur. Oba te rozwiązania wiążą się jednak ze zwiększonym zapotrzebowaniem na pamięć. W przypadku tekstur animowanych ilość niezbędnej pamięci powiększa się również w związku z liczbą zapisanych klatek. Oba problemy, powtarzalny wzór powierzchni oraz wymuszona ograniczoną pamięcią niska liczba klatek, są doskonale widoczne we wspomnianej grze Doom.

Rysunek 6. Kadr z demonstracji technologicznej X-Isle Dinosaurs Island (2004 rok). Źródło: [4]

Innym rozwiązaniem problemu jest generowanie wzorów podczas działania samej gry. Zmniejsza to zapotrzebowanie na pamięć oraz wielkość plików gry. Funkcja wykorzystana w tym celu musi cechować się pewnymi własnościami:

- kształt generowanych fal musi przypominać kształt prawdziwych fal na wodzie,
- sposób przemieszczania się fal musi przypominać sposób przemieszczania się fal na wodzie,
- musi istnieć możliwość modyfikacji parametrów fali, w celu symulacji wpływu, jaki ma na nią siła wiatru czy odległość do dna.

Na przestrzeni lat pojawiło się wiele metod generowania takich wzorów powierzchni.

2.3.2 Rozwiązania oparte na funkcjach trygonometrycznych

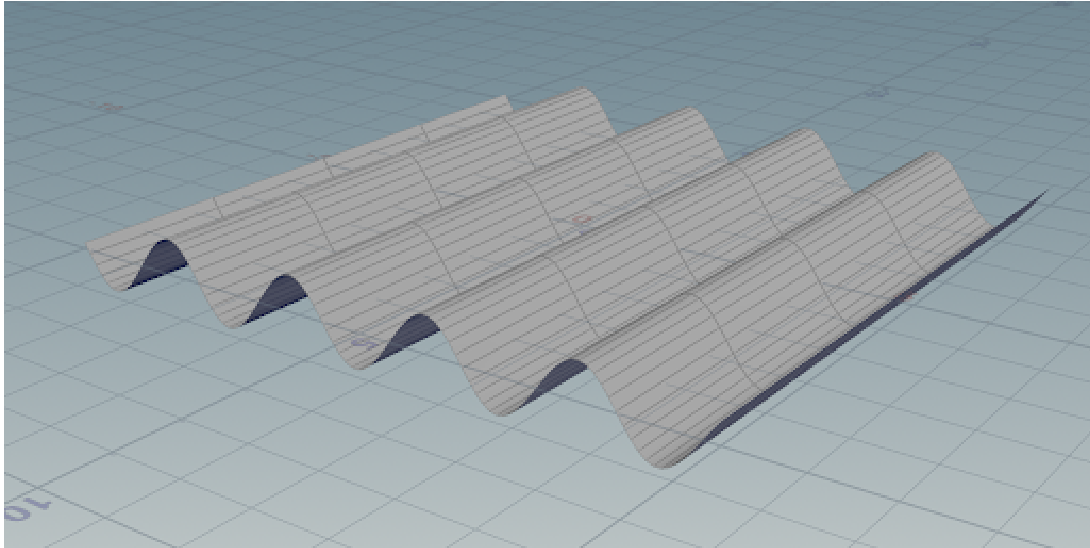
Prostym przykładem metod spełniających uprzednio przedstawione wymagania są metody oparte na wykorzystujące funkcje trygonometryczne. Równanie 1 prezentuje przykład wzoru, który można zastosować w celu generowania powierzchni wody w oparciu o funkcję cosinus:

$$\mathrm{f(x,t) = Acos(\frac{2\pi(x - Ct)}{L})} \tag{1}$$

gdzie:

- x - odległość od źródła fali,
- t - czas,
- C - prędkość propagacji fali,

- L - długość fali.



Rysunek 7. Wizualizacja fali określonej równaniem. Źródło: 1

Przykład fali wygenerowanej przy użyciu powyższego wzoru zobaczyć można na rys. 7. W roku 1981 zaproponowano [41] modyfikację tej funkcji poprzez wykorzystanie serii nakładających się na siebie fal sinusoidalnych:

$$f(x,t) = -y_0 \sum_{i=1}^{N_w} A_i \cos(k_{i_x} + k_{i_z} - w_i t) \tag{2}$$

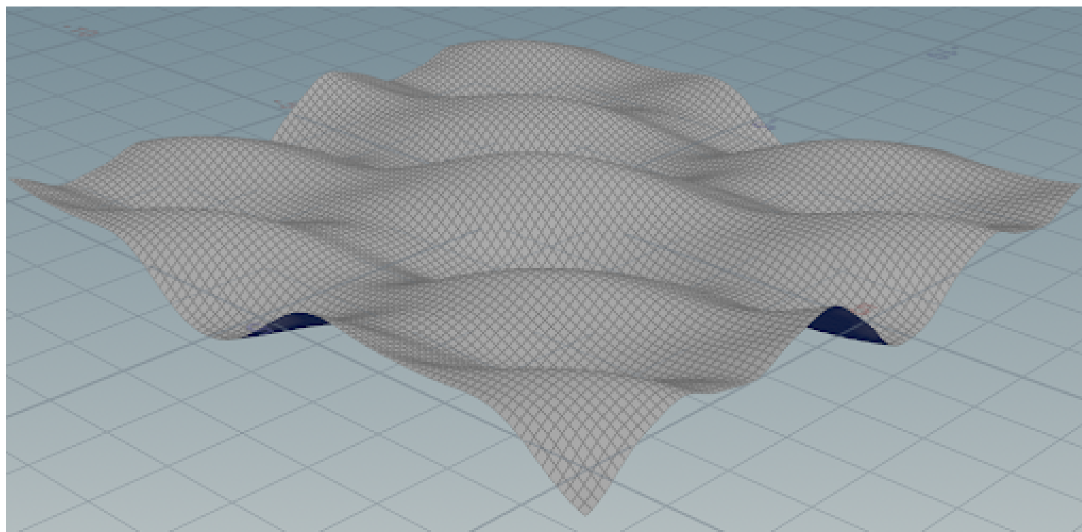
gdzie:

- x - odległość od źródła fali w osi x,
- z - odległość od źródła fali w osi z,
- t - czas,
- N_w- liczba nakładanych fal,
- A_i - amplituda fali o indeksie i,
- k_i - wektor wyznaczający przesunięcie początkowe fali o indeksie x,
- w_i - prędkość propagacji fali o indeksie i,
- y_0- wysokość lustra wody.

Efekt zastosowania tej modyfikacji zobaczyć można na rys. 8. Powierzchnia wygenerowana przy pomocy funkcji trygonometrycznych cechuje się gładkim, zaokrąglonym kształtem. Najbardziej odpowiednia jest dla generowania powierzchni wody w zbiorniku o spokojnej tafli.

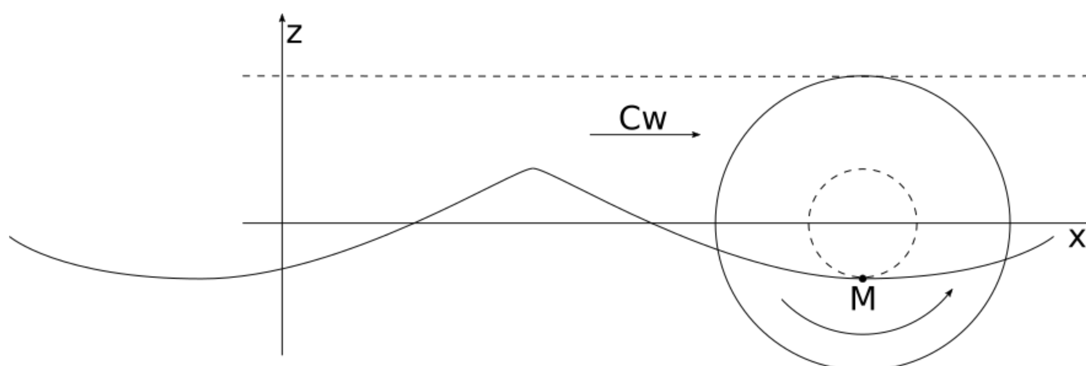
2.3.3 Fala trochoidalna

Kolejną funkcją używaną w celu generacji powierzchni wody, dziś spotykaną zdecydowanie częściej niż metody oparte na funkcjach trygonometrycznych, jest funkcja fali trochoidalnej zwana falą Gerstnera.



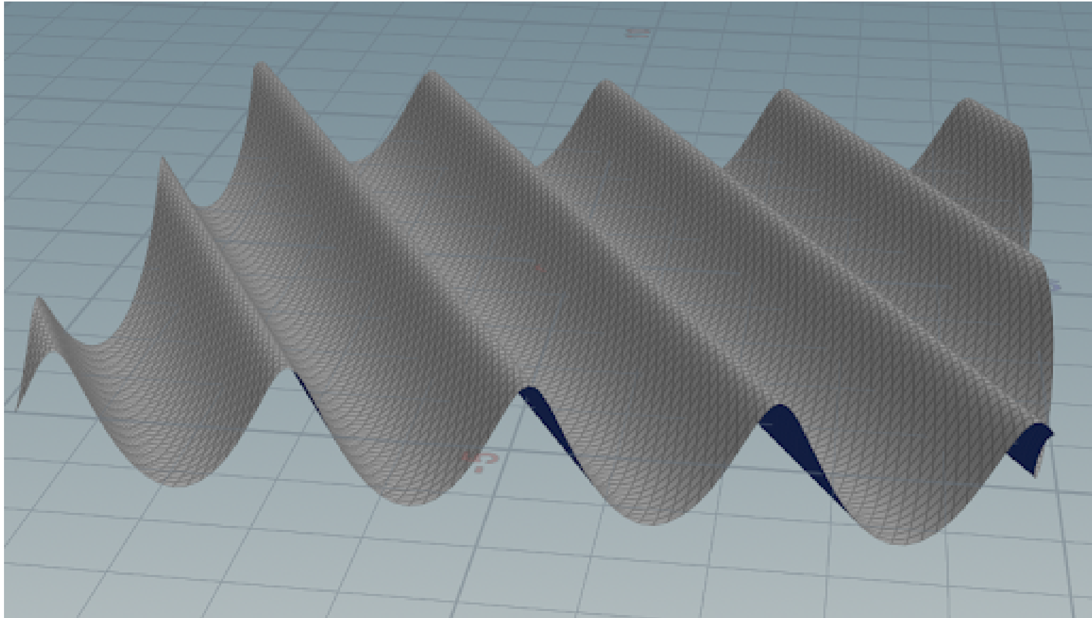
Rysunek 8. Wizualizacja fali określonej równaniem. Źródło: 2

Jej wykorzystanie zaproponowano w roku 1986 [13]. Fala tego typu charakteryzuje się ostrymi grzbietami i szerokimi dolinami między nimi które bardziej przypominają powierzchnię burzliwego oceanu. Przykład powierzchni wygenerowanej przy użyciu tej metody prezentuje rys. 10 oraz rys. 11 Funkcja, na której bazuje ta metoda generacji przedstawiona została w roku 1802 przez Františka Josefa Gerstnera. Wykazał on, że profil fali o skończonej amplitudzie charakteryzuje się opisanymi wyżej cechami. Matematyk uznał, że profil ten zbliżony jest do trochoidy, czyli skróconej cykloidy. Oznacza to krzywą, jaką zakreśla się punkt znajdujący się wewnątrz okręgu toczącego się bez poślizgu po prostej, co pokazuje rys. 9. Funkcję określają wzory:

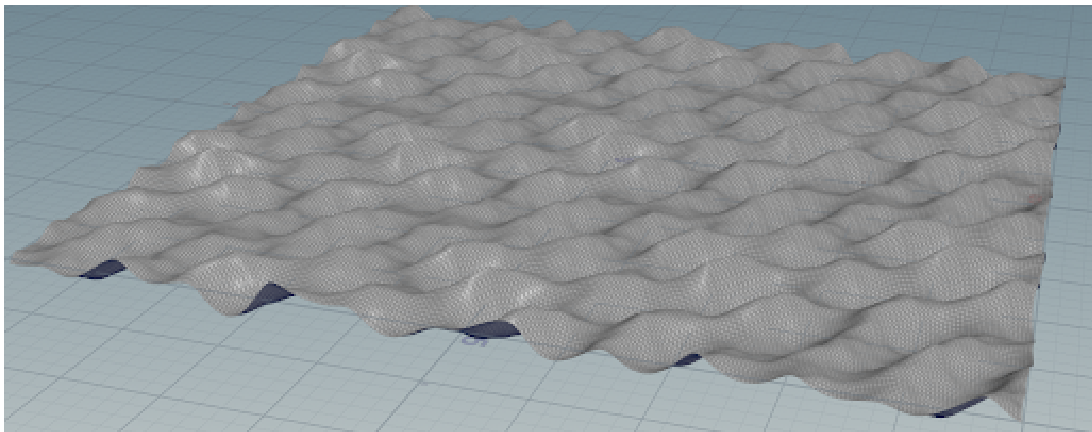


Rysunek 9. Proces rysowania trochoidy za pomoc toczącego się okręgu. M — kierunek obrotu okręgu, Cw — Kierunek rysowania krzywej. Źródło: [35]

$$x = x_0 - \sum_{i=0}^{N_w} \frac{\vec{k_i}}{k_i} A_i \sin(\vec{k_i} \cdot x_0 - w_{it} + \theta_i) \tag{3}$$



Rysunek 10. Powierzchnia powstała przez użycie pojedynczej fali Gerstnera jako wartości przemieszczenia. Źródło: opracowanie własne.



Rysunek 11. Powierzchnia powstała przez użycie superpozycji wielu fal Gerstnera jako wartości przemieszczenia. Źródło: opracowanie własne.

$$y = y_0 - \sum_{i=1}^{N_w} A_i \sin(\vec{k_i} \cdot x_0 - w_i t + \theta_i) \tag{4}$$

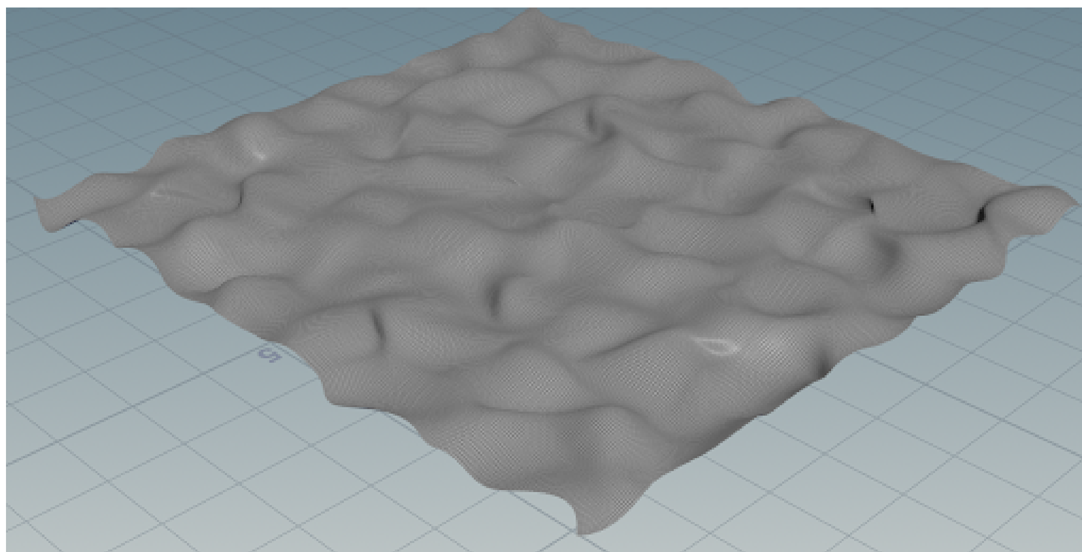
gdzie:

- x_0 - początkowe przesunięcie w osi x,
- y_0 - wysokość lustra wody,
- k_i - kątowa liczba falowa fali o indeksie i,
- i - przesunięcie fazowe fali o indeksie i,
- A_i - amplituda fali o indeksie i,

- $\mathrm{N_w}$ - liczba nakładanych fal,,
- $\mathrm{w_i}$ - prędkość propagacji fali o indeksie i,
- x - odległość od źródła fali w osi x,
- z - odległość od źródła fali w osi z,
- t - czas,
- θ_i - przesunięcie fazowe fali o indeksie i.

2.3.4 Szum Perlina

Spośród funkcji wykorzystywanych do generacji fal warto wyróżnić również szum Perlina. Jest to opatentowany algorytm generowania szumu gradientowego opisany w 1985 roku, który powstał podczas prac nad filmem Tron. Jego autor, Ken Perin, w roku 1997 otrzymał nagrodę Oskara za osiągnięcia techniczne, natomiast sam algorytm doczekał się szerokiej gamy zastosowań w zakresie generacji proceduralnej. Przykład powierzchni powstałej przy użyciu szumu Perlina prezentuje rys.12. Oprócz generowania powierzchni wody stosowany jest między innymi przy tworzeniu szczytów gór, terenu, jaskini czy powierzchni materiałów organicznych, jak skóra gadów. Ze względu na elastyczność szczególną popularnością cieszy się w świecie tak zwanej demosceny, czyli wśród osób zajmujących się tworzeniem programów generujących skomplikowane animacje przy minimalnym rozmiarze pliku wykonywalnego, ponieważ za pomocą jednego algorytmu możliwe jest osiągnięcie wielu różnych efektów wizualnych.



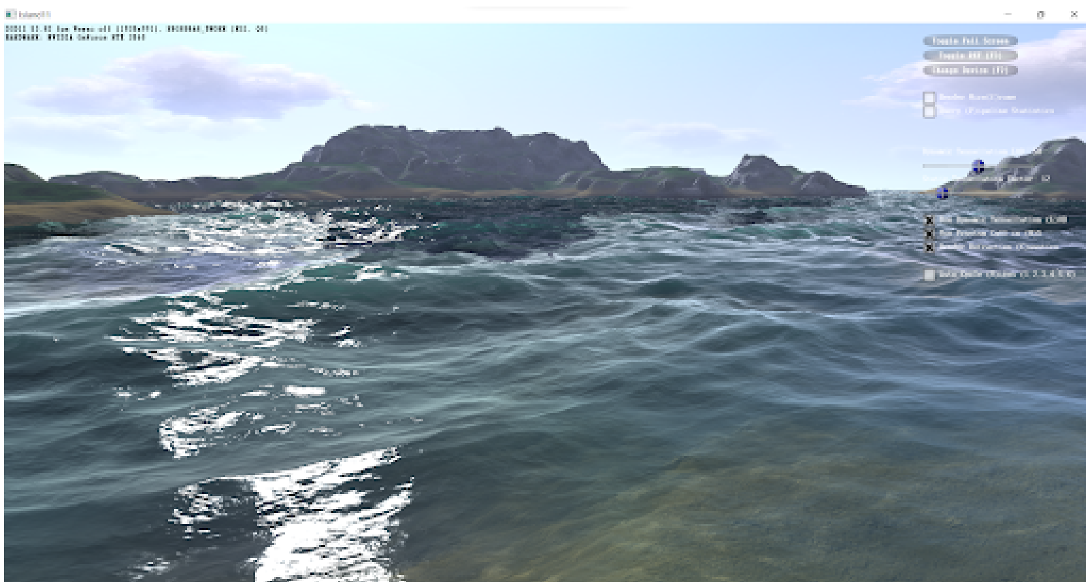
Rysunek 12. Powierzchnia powstała przez użycie szumu Perlina jako wartości przemieszczenia. Wykorzystano implementację algorytmu szumu Perlina wbudowaną w aplikację SideFX Houdini. Źródło: opracowanie własne.

2.3.5 Szybka Transformata Fouriera

Kolejnym rozwiązaniem, znacząco przybliżającym wygląd powierzchni generowanej wody do fotorealizmu, było wykorzystanie szybkiej transformaty Fouriera (Fast Fourier Transform, FFT)

zaproponowanej w roku 1987 [37] i spopularyzowanej dzięki artykułowi napisanemu w roku 2001 [64]. Rozwiązania oparte na niej cechują się wysoką jakością, w związku z czym wykorzystywane są w przemyśle filmowym. Zastosowano je w filmach Titanic oraz Waterworld [64]. Przykładem użycia tej technologii w silnikach czasu rzeczywistego może być imponująca demonstracja o nazwie Island Direct3D 11 Tech Demo wykonana przez firmę NVidia lub gry Assassin's Creed 3 oraz Assassin's Creed Black Flag.

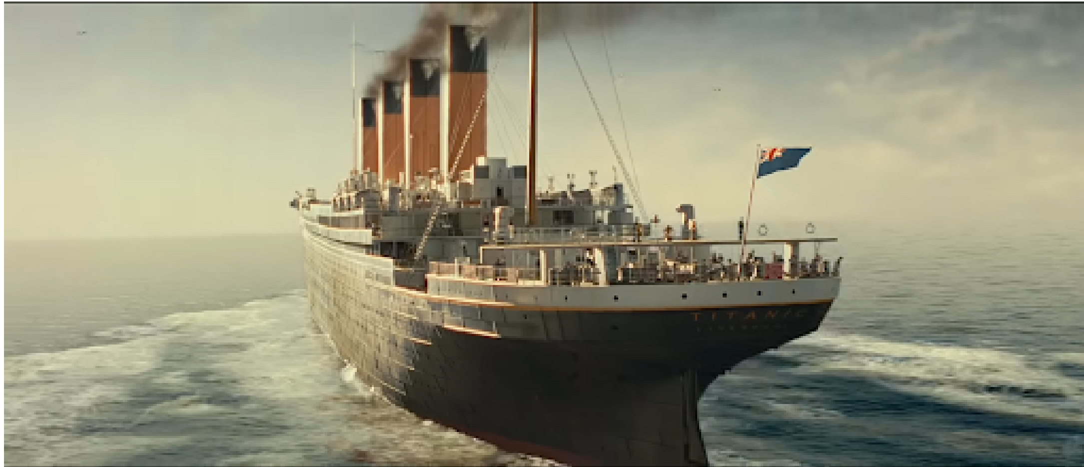
W przypadku tego rozwiązania fala reprezentowana jest za pomocą sumy sinusoid, podobnie jak w pierwszym z opisanych w rozdziale rozwiązań. Dane wykorzystywane w tym wypadku, takie jak długość fali, prędkość i amplituda pochodzą jednak z modeli statystycznych, natomiast liczba rozpatrywanych sinusoid jest znacząco większa. Technika wykorzystywana jest podczas symulacji powierzchni wody oceanu lub dużych zbiorników wodnych, gdy nie zachodzi interakcja między płynem a brzegiem lub obiektami w niej zanurzonymi, ponieważ nie pozwala ona na symulowanie wpływu takich przeszkód. Wadą rozwiązania, w porównaniu do wymienionych wcześniej, jest znacząco wyższa złożoność obliczeniowa.



Rysunek 13. Kadr z dema technicznego Island Direct3D 11 Tech Demo. Źródło: [69]

Transformata Fouriera jest stworzonym przez Josepha Fouriera narzędziem matematycznym będącym podstawowym instrumentem analizy harmonicznej oraz teorii analizy i przetwarzania sygnału. Pozwala ono na zmianę dziedziny funkcji ciągłej z czasu na funkcję ciągłą dziedziny częstotliwości w bezstratny sposób. Uzyskana w ten sposób funkcja nazywana jest transformatą Fouriera. Dzięki odwrotnej transformacie Fouriera możliwe jest odtworzenie oryginalnej funkcji z jej transformaty. Transformację Fouriera określa wzór:

$$\vec{\mathrm{f}}(\xi) = \int_{-\infty}^{\infty} \mathrm{f}(\mathrm{t})\mathrm{e}^{-2\pi \mathrm{i} \mathrm{t} \xi} \mathrm{dt} \tag{5}$$



Rysunek 14. Kadr z filmu Titanic. Źródło: [40]

gdzie:

- $f(t)$ - funkcja w dziedzinie czasu, której transformata jest obliczana,
- t - czas,
- i - jednostka urojona ($i^2 = -1$),
- ξ - częstotliwość.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} k = 0, ..., N-1 \tag{6}$$

gdzie:

- $f(t)$ - funkcja w dziedzinie czasu, której transformata jest obliczana,
- N - liczba wartości funkcji $f(t)$,
- i - jednostka urojona ($i^2 = -1$),
- x_n - wartość o indeksie n.

Złożoność obliczeniowa powyższego algorytmu jest wysoka i wynosi $O(N^2)$. Szybką transformacją Fouriera nazywamy algorytm wyznaczania dyskretnej transformaty Fouriera oraz transformaty do niej odwrotnej o znacznie niższej złożoności obliczeniowej $O(N\log_2 N)$. Algorytm wykorzystywany jest szeroko w dziedzinie cyfrowego przetwarzania sygnałów, a także w kompresji danych, korzysta z niego kompresja obrazów JPEG, dźwięku MP3 oraz wiele innych.

Wykorzystanie szybkiej transformacji Fouriera do symulacji powierzchni wody polega na zastosowaniu jej w celu określenia wysokości powierzchni wody w oparciu o posiadane dane statystyczne opisujące jej spektrum. Dane mogą pochodzić z pomiarów prawdziwego lustra wody lub wyliczeń teoretycznych. Często wykorzystywane jest tak zwane Spectrum Phillipsa [23] powstałe na podstawie zebranych danych empirycznych. Na podstawie wybranego spektrum tworzona jest funkcja będąca zestawieniem sinusoid, których superpozycja wyznacza wysokość powierzchni wody w danym miejscu na jej płaszczyźnie określonym za pomocą dwóch współrzędnych. Powstała w ten sposób mapa

wysokości, zwana mapą przemieszczenia (displacement map) często konwertowana jest następnie na mapę wektorów normalnych, która poprawia szczegółowość powierzchni.

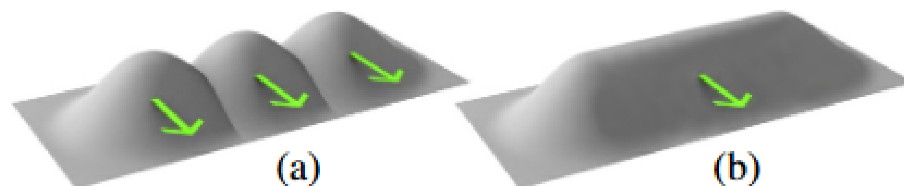
Większa rozdzielczość wygenerowanych map oznacza większą jakość powierzchni, jednak zwiększa ilość potrzebnej pamięci. Generacja powierzchni wody wykorzystanej w filmie Titanic odbywała się przy wykorzystaniu map o rozdzielczości 2048 na 2048 pikseli. Dla porównania pierwsze implementacje tej technologii w grach, jak powstały w roku 2007 Crysis, wykorzystywały rozdzielczość 64 na 64 piksele. Obecnie wydawane tytuły zazwyczaj stosują rozdzielczość 512 na 512 pikseli.

2.4 Metody interaktywne

W tej sekcji omówiono metody, których celem jest wygenerowanie powierzchni wody zdolnej do interakcji. Oznacza to, że powierzchnia generowana przez te metody zdolna jest do reakcji na bodźce zewnętrzne, takie jak pływające po niej obiekty czy akcje wykonywane przez gracza. Efektem końcowym pracy ma być aplikacja zdolna do symulacji takiej właśnie powierzchni.

2.4.1 Rozwiązania oparte na cząsteczkach

W roku 2007 zaproponowano rozwiązanie oparte na cząsteczkach [75]. Polega ono na reprezentowaniu fal przez zdolne do interakcji z otoczeniem, takich jak odbicia od różnych powierzchni, cząsteczki. Następnie, na podstawie położenia cząsteczek, aproksymowany jest kształt fali. Cząsteczki poruszają się jedynie po powierzchni reprezentującej poziom wody, są niewidoczne dla gracza, widzi on jedynie powierzchnię powstałą na ich podstawie. Znajdujące się w niewielkiej odległości cząsteczki łączone są w kształt pojedynczej fali, w sposób przedstawiony na rys. 15. Technika pozwala na uzyskanie wysokiej szczegółowości, podobnie jak omawiane wcześniej FFT, wprowadza jednak dodatkowy element interakcji z otoczeniem. Przykład powierzchni stworzonej za pomocą opisywanej metody przedstawia rys. 16.



Rysunek 15. Przedstawienie sposobu tworzenia czoła fali na podstawie cząsteczek. Źródło: [75]

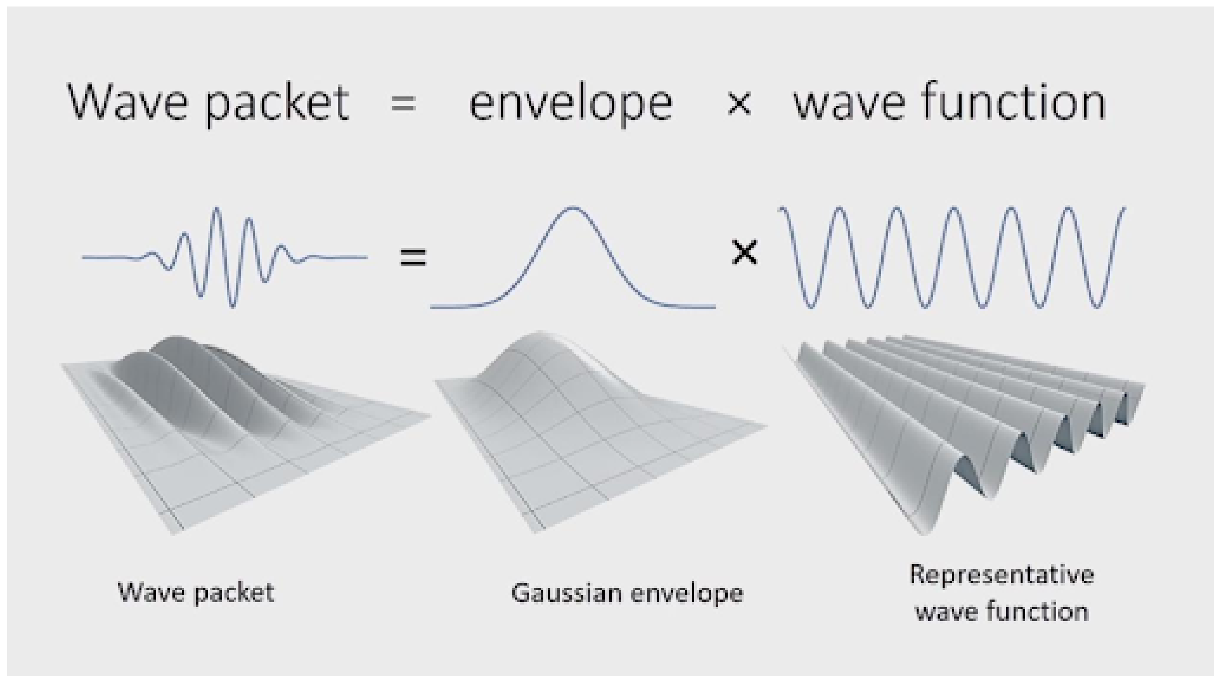
Technikę tę wykorzystywała między innymi seria gier Uncharted. W grze powierzchnię morza symulowaną za pomocą losowo rozsianych cząsteczek poruszających się z losową prędkością, co stymulowało pozorny chaos morskich fal. Następnie generowano na tej podstawie wektorową mapę przemieszczenia.

Rysunek 16. Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75]

Kolejną techniką opartą na cząsteczkach, oraz rozszerzeniem omawianej wcześniej techniki, jest użycie paczek fal wody (Water Wave Packets). Rozwiązanie zostało zaproponowane podczas SIGGRAPH 17 [27], następnie rozszerzone i przemianowane na pakiety falek powierzchni (Water Surface Wavelets) podczas SIGGRAPH 18 [26]. Metoda polega na użyciu cząsteczek, zwanych paczkami, których wpływ na kształt fali można rozumieć jako zmianę jej amplitudy i ustalenie kierunku poruszania się jej, oraz sinusoid o mniejszej długości fali, oraz większej prędkości falowej, których amplituda modyfikowana jest przez wspomnianą paczkę. Można powiedzieć, że sinusoidy o mniejszej długości niż paczka przesuwają się po niej, zmieniając w tym czasie swoją amplitudę. Zasadę działania metody prezentuje rys. 17 natomiast przykład wygenerowanej powierzchni widoczny jest na rys. 18. Metoda ta łączy więc niewielką złożoność obliczeniową i duże możliwości interakcji rozwiązań opartych na cząsteczkach z wysokim poziomem detalu właściwym rozwiązaniom opartym na transformacie Fouriera.

2.4.2 Równania Naviera-Stokesa

Wszystkie wymienione dotąd rozwiązania mają na celu jedynie naśladować wygląd wody lub jej powierzchni. Nie są jednak w żadnym stopniu fizycznie poprawne. Kolejną kategorię metod symulacji wody stanowią metody oparte na równaniach Naviera-Stokesa [55]. Ważną publikacją, na której opiera się wiele z takich rozwiązań jest praca "Real-Time Fluid Dynamics for Games" opublikowana w roku 2003 [61]. Jest to zestaw równań opisujących zasadę zachowania pędu dla poruszających się cieczy. W przyjętym w nich modelu zmiany pędu zależą wyłącznie od zewnętrznego ciśnienia, sił lepkości oraz sił masowych w cieczy. W praktyce rozwiązanie równania możliwe jest jedynie dla prostych przypadków, jak laminarny przepływ nieściśliwego płynu o niskiej lepkości. Nie jest więc



Rysunek 17. Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75]

Rysunek 18. Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75]

możliwe użycie tej techniki przy analizie często występującego turbulentnego przepływu. Mimo to, równania te wykorzystywane są powszechnie w meteorologii czy aerodynamice.

Równanie można przedstawić w następujący sposób:

$$\frac{\delta \mathrm{u}}{\delta} = -(\mathrm{u} \cdot \nabla)\mathrm{u} + \rho\nabla^2\mathrm{u} + \mathrm{f} \tag{7}$$

gdzie:

- u - wektor prędkości cieczy,
- p - ciśnienie cieczy,
- ρ - gęstość cieczy,
- ϵ - lepkość dynamiczna cieczy,
- ∇^2 - wektorowy operator Laplacea,
- f - siły zewnętrzne. Często zastępuje się je przyspieszeniem ziemskim (g) zakładając, że jedyną oddziaływującą na ciecz siłą jest grawitacja.

Stosowane są trzy metody podejścia do rozwiązania przedstawionego zadania:

1. **Metoda Eulera** - polega na ustaleniu wartości w nieruchomych, z góry ustalonych, punktach, tworzących zazwyczaj dwu lub trójwymiarową siatkę. Wartości związane z symulacją przechowywane są jedynie dla nieruchomych punktów.

2. **Metoda Lagrange'a** - polega na obliczaniu wartości dla poruszających się punktów, reprezentujących symbolicznie cząsteczki płynu lub gazu. Oznacza to, że każdy rozpatrywany punkt musi posiadać dodatkowe informacje o swoim stanie, oraz że zmienia swoje położenie. Nie mamy do czynienia z regularną siatką punktów, mogą występować różnice w zagęszczeniu punktów w przestrzeni, przez co metoda ta cechuje się mniejszą stabilnością. Metoda ta stosowana bywa w połączeniu z metodą Smoothed Particle Hydrodynamics(SPH).

3. **Zmodyfikowana metoda Lagrangea (semi-lagranian, SL)** - stara się połączyć zalety obu wymienionych metod. Używa ona siatki nieruchomych punktów, znanej z metody Eulera, jednak wartości obliczone dla niej są wykorzystywane do uzyskania wartości w punktach reprezentujących ciecz lub gaz za pomocą interpolacji.

Przykładem zastosowania opisanego podejścia może być wykorzystywany w wielu tytułach silnik fizyki PhysX produkowany przez firmę NVidia. Posiada on implementację fizyki płynów korzystającą z metody SPH.

2.4.3 Równania falowe

Podejście oparte na równaniach Naviera-Stokesa znajduje zastosowanie przy symulacji całej objętości cieczy. Jeśli wymagane jest jedynie symulowanie jej powierzchni zastosowanie znajdują również metody, oparte na równaniu falowym [60]. Równanie to opisuje rozprzestrzenianie się wielu rodzajów fali, takich jak fale dźwiękowe, światło, czy fale na wodzie, w związku z czym znalazło liczne zastosowania w różnych dziedzinach fizyki, jak fizyka płynów, akustyka czy optyka.

Równanie ma postać:

$$\triangle \mathrm{f}(\mathrm{x},\mathrm{t}) - \frac{1}{\mathrm{c}^2}\frac{\delta^2 \mathrm{f}(\mathrm{x},\mathrm{t})}{\delta \mathrm{t}^2} \tag{8}$$

gdzie:

- t - czas,
- x - pozycja na powierzchni wody,
- c - prędkość falowa,
- ∇^2 - wektorowy operator Laplace'a.

Wykorzystanie równania podczas symulacji powierzchni wody sprowadza się do ustalenia źródła fali, następnie użycia równania w celu przeprowadzenia jej propagacji w przestrzeni.

2.5 Rozwiązania wykorzystujące Głębokie Sieci Neuronowe

Istnieją przykłady zastosowania sieci neuronowych przy symulacji lub wspomaganiu symulacji wody [5]. Są to wciąż niszowe rozwiązania, stanowią jednak dowód możliwości, jakie niesie proponowane rozwiązanie.

Wtyczka Zibra Liquids

Opublikowana w roku 2022 wtyczka dla silnika gier Unity o nazwie Zibra Liquids [76] zasłynęła dzięki przykładom wysokiej jakości symulacji wykonywanych za jej pomocą. Kadr przedstawiający efekt symulacji wykonanej za pomocą tego narzędzia widać na rys. 19 Szczegóły techniczne jej działania nie zostały opublikowane, jednak z danych zawartych w dokumentacji wynika, że wykorzystuje ona sieć neuronową. Sieć nie jest odpowiedzialna za wykonanie całości symulacji, lecz za stworzenie tak zwanego Signed Distance Field (SDF), czyli wokselowej reprezentacji obiektu. Wykorzystywana jest ona dalej przez bardziej tradycyjny algorytm symulacji. Ważną cechą wtyczki jest działająca w obie strony interakcja między płynem a innymi obiektami fizycznymi. Istnieją też przykłady wykorzystania głębokich sieci neuronowych podczas symulacji cieczy. Przykładami mogą być:

- opublikowane w roku 2020 [45] zestawienie implementacji sieci neuronowych zdolnych do aproksymowania wyników różnych symulacji fizycznych,
- sieć, oparta na splotowej głębokiej sieci neuronowej, zdolna do aproksymacji ustalonego przepływu cieczy przy zadanych wartościach brzegowych [24][50].

Eksperymenty DeepMind

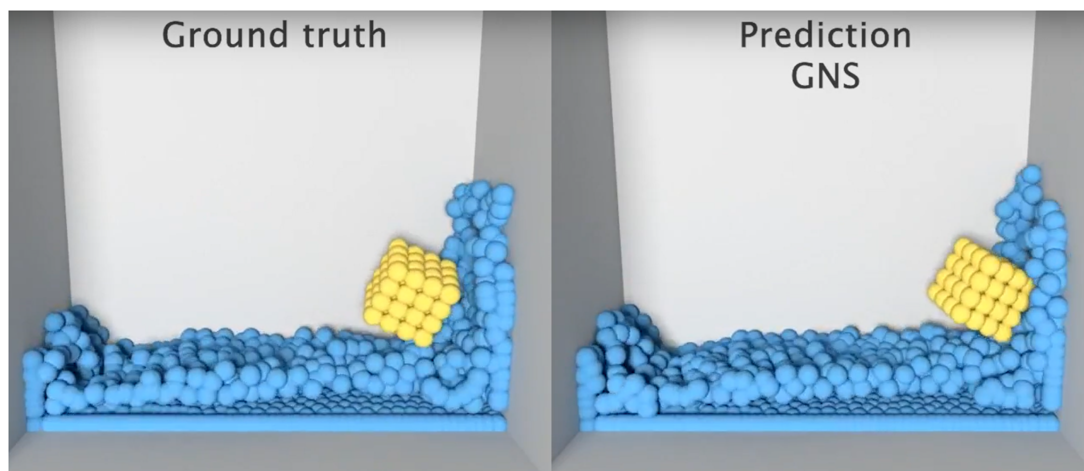
Zespół Deep Mind zaprezentował liczne przykłady wykorzystania grafowych sieci neuronowych w celu odtworzenia rezultatów symulacji przeprowadzonej za pomocą cząsteczek [56].

W celu szkolenia sieci przygotowano bazę danych zawierającą wyniki symulacji przeprowadzonej za pomocą różnych silników symulacji. Zadaniem sieci jest przewidzenie kolejnego stanu cząsteczki na podstawie jej stanu obecnego oraz informacji o jej najbliższych sąsiadach. Grafowa sieć neuronowa

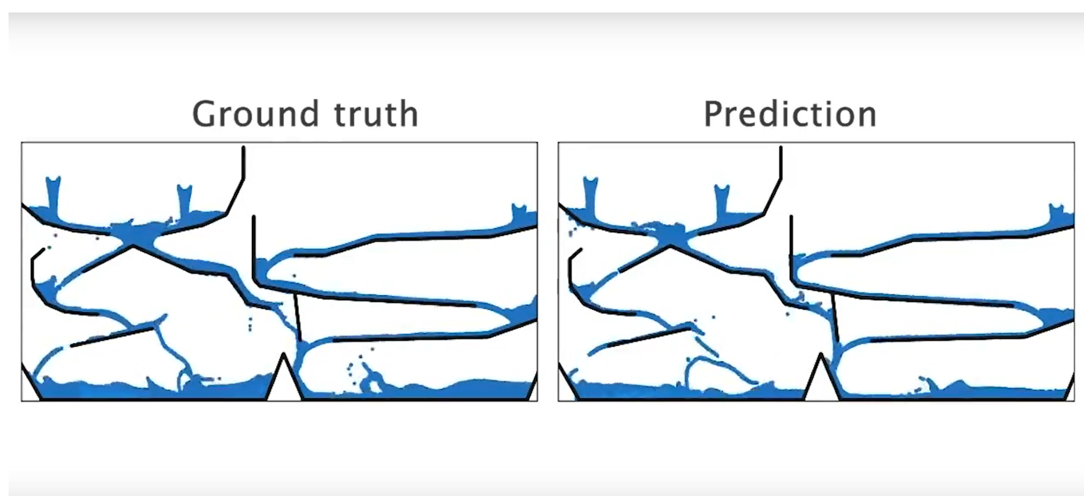
Rysunek 19. Kadr przedstawiający efekt symulacji wykonanej za pomocą narzędzia Zibra Liquids. Źródło: [76]

ewaluowana jest dla każdej z cząsteczek osobno. Przedstawiono wyniki zastosowania tej architektury w celu przewidzenia szerokiej gamy symulacji, zarówno w dwóch21, jak i w trzech wymiarach20. Twórcy podkreślają wysoką skalowalność ich rozwiązania: mechanizmy, których sieć nauczyła się na symulacji wykonanej przy niewielkiej liczbie cząsteczek, generują poprawne rezultaty przy znacznym zwiększeniu ich liczby. Warto podkreślić również stabilność generowanych rezultatów, na którą głównym dowodem jest możliwość generacji stosunkowo długich sekwencji, które zachowują się w spodziewany sposób.

Celem projektu nie jest osiągnięcie wysokiej wydajności, co podkreślają sami twórcy [11]. Wykorzystanie rezultatów w celu interaktywnej symulacji, co jest celem tej pracy, jest dodatkowo utrudnione przez reprezentację danych: w celu ich wykorzystania konieczne jest wygenerowanie siatki płynu w oparciu o pozycje cząsteczek, co dodatkowo zwiększa złożoność obliczeniową. Rozwiązanie nie jest więc przeznaczone do stosowania w celach podobnych tym, które przyświecają niniejszej pracy, jest jednak wspaniałym pokazem możliwości, jakie niesie użycie sieci neuronowych w zakresie odtwarzania wyników symulacji.



Rysunek 20. Kadry przedstawiające porównanie wyników działania grafowej sieci neuronowej stworzonej przez zespół DeepMind z wynikiem symulacji. Źródło: [56]



Rysunek 21. Kadry przedstawiające porównanie wyników działania grafowej sieci neuronowej stworzonej przez zespół DeepMind z wynikiem symulacji. Źródło: [56]

Inne projekty badawcze

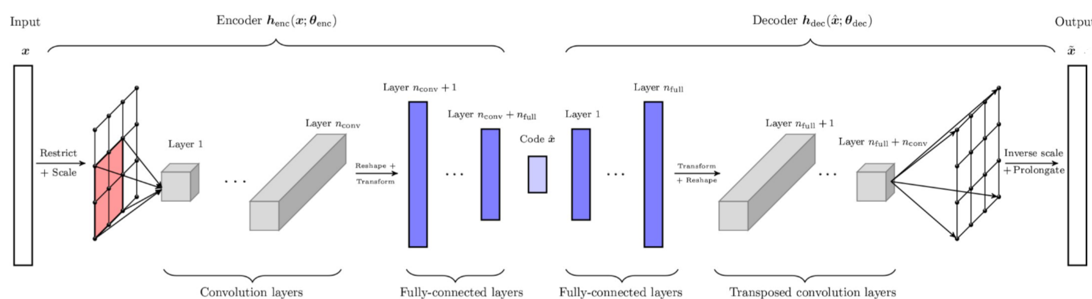
Istnieje wiele projektów badających możliwość wykorzystania sieci neuronowych w celu symulacji zachowania cieczy. Warto wymienić kilka z nich, które przynoszą obiecujące rezultaty w tej dziedzinie.

Prostym, jednak udanym podejściem jest zastosowanie sieci U-Net w celu replikacji wyników obliczeń opartych na równaniach Naviera-Stokesa [16]. Umożliwia to modelowanie i symulację zachowania cieczy, uwzględniając skomplikowane przepływy i turbulencje. Innym interesującym pomysłem jest wykorzystanie sieci neuronowych do symulowania rozbryzgów cieczy [70]. Próbuje się tutaj modelować dynamikę cząsteczek cieczy w momencie rozpryskiwania, co pozwala na generowanie realistycznych efektów rozbryzgów.

Kolejnym interesującym projektem jest próba zwiększenia rozdzielczości symulacji [30]. Podczas tego procesu uzupełnione muszą zostać efekty w skali zbyt małej, by uwzględnione zostały w pierwotnej symulacji. Powoduje to powiększające się z czasem odstępstwa od poprawnego wyniku. W związku z tym sieć neuronowa musi być w stanie je aproksymować podczas zwiększania rozdzielczości symulacji. Twórcy pracy deklarują przyspieszenie obliczeń rzędu 40 do 80 krotności.

Interesującym podejściem jest wykorzystanie rozkładu według wartości osobliwych wspieranej za pomocą sieci neuronowych realizującej proces redukcji wymiaru problemu obliczeniowego [34]. Architektura ta przedstawiona jest na rys. 22 W tym celu zastosowano głęboki auto enkoder kompresujący dane, które następnie wykorzystano do odnalezienia rozwiązania między innymi za pomocą metody Galerkina.

Obiecujące efekty uzyskują naukowcy starający się przyspieszyć niektóre kroki symulacji cieczy za pomocą sieci neuronowych [73]. Trwają również prace nad wykorzystaniem sieci neuronowych w celu przewidywania przepływów turbulentnych. [36]. Ponadto badacze podejmują również próby wykorzystania sieci neuronowych opartych na innych architekturach, niż grafowe do symulacji cieczy opartych na modelu cząsteczkowym [47]. Takie podejście umożliwia dokładne odwzorowanie zachowania cząsteczek cieczy, uwzględniając siły międzycząsteczkowe i interakcje cząsteczek.



Rysunek 22. Architektura sieci splotowego autoenkodera, który przyjmuje stan układu dynamicznego jako wejście i produkuje przybliżony stan jako wyjście. Stan reprezentowany przez x wykorzystany został w obliczeniach. Źródło: [34]

2.6 Symulacja wody we współczesnych grach komputerowych

Rozwiązania stosowane w celu tworzenia interaktywnej powierzchni wody w największych tytułach zwykle nie są ujawniane publicznie. Okazjonalnie twórcy tytułów decydują się na przedstawienie takich informacji podczas branżowych konferencji. Ma to miejsce przeważnie w sytuacji, gdy zastosowali nietypowe rozwiązanie, które chcą przedstawić szerszej publiczności.

W przypadku symulacji zachowania ciał sztywnych istnieją powszechnie używane narzędzia, które są stosowane zarówno przez mniejsze, jak i większe tytuły o najwyższych budżetach (zwane AAA lub Triple-A). Jednakże, w przypadku symulacji powierzchni wody nie występuje taka standaryzacja. Dzieje się tak, ponieważ potrzeby związane z symulacją powierzchni wody są znacznie bardziej zróżnicowane. Większość tytułów musi wykorzystywać pewną formę symulacji

zachowania ciał sztywnych, która stanowi istotny element rozgrywki. Dlatego ta symulacja musi zawsze charakteryzować się wysoką jakością, a związane z nią błędy często są przedmiotem żartów w skompilowanych nagraniach z błędami.

Sytuacja wygląda inaczej w przypadku symulacji powierzchni wody: wiele tytułów w ogóle nie wymaga takiej symulacji, a dla znacznej części pozostałych wystarczające są podstawowe efekty specjalne, takie jak reakcja na pociski. Równocześnie dla niektórych tytułów, takich jak Noita czy Hydrophobia: Prophecy, symulacja powierzchni wody stanowi kluczowy element mechaniki gry. Pełna i dokładna symulacja powierzchni wody może być bardzo kosztowna, więc jeśli nie jest ona niezbędna, jest upraszczana, lub z integracji rezygnuje się całkowicie. Zaoszczędzoną w ten sposób moc obliczeniową wykorzystać można w innych celach. Taką drogę przyjęto podczas tworzenia gry Cyberpunk 2077. Premierowa wersja gry nie posiadała implementacji interaktywnej wody. Pojawiła się ona dopiero w łatce 1,5, opublikowanej ponad rok po premierze samej gry [7]. Istnieje również wiele różnych metod wyświetlania samej powierzchni wody oraz technik renderowania, co dodatkowo utrudnia stworzenie jednego uniwersalnego rozwiązania odpowiedniego dla każdego tytułu. W związku z tą sytuacją większość dużych produkcji opracowuje własne implementacje symulacji powierzchni wody dostosowane do swoich potrzeb.

Jedną z produkcji, której twórcy postanowili opowiedzieć o swojej implementacji symulacji powierzchni wody, jest Atlas [62]. W przypadku tego projektu zdecydowano się na zastosowanie rozwiązania opisanego pierwotnie w pracy eWave: Using an Exponential Solver on the iWave Problem [65].

Często stosowanym rozwiązaniem jest przeniesienie obliczeń związanych z symulacją powierzchni wody do cieniowania [8]. W opisywanym podejściu symulacja korzysta z tekstury, która przechowuje informacje o wysokości fali w bieżącym oraz poprzednim momencie. W każdym kroku symulacji aktualna wartość wysokości wody jest zapisywana w celu wykorzystania jej w kolejnej klatce. Jednocześnie obliczana jest suma wartości sąsiednich pikseli, a od niej odejmowana jest wartość wysokości wody z poprzedniej klatki. Proces ten został przedstawiony w kodzie zaprezentowanym we frag.1. Podobna, lecz bardziej zaawansowana metoda symulacji jest wykorzystywana między innymi w grze Red Dead Redemption 2[28].

```
vec2 simStep(in vec2 fragCoord) {
   return vec2((
      (
       texture(iChannel0, (fragCoord + vec2(-1,0 ))/iResolution.xy).x +
       texture(iChannel0, (fragCoord + vec2( 1,0 ))/iResolution.xy).x +
       texture(iChannel0, (fragCoord + vec2( 0,-1))/iResolution.xy).x +
       texture(iChannel0, (fragCoord + vec2( 0,1 ))/iResolution.xy).x) * 0,5 -
      texture(iChannel0, fragCoord/iResolution.xy).y),
       texture(iChannel0, fragCoord/iResolution.xy).x);
}
```

Listing 1. Kod reprezentujący krok symulacji wykonywnywanej przy użyciu ceiniowania. Źródło: [17]

2.7 Wtyczka Fluid Flux

Rosnącą popularnością cieszy się w ostatnim czasie wtyczka Fluid Flux [31], która zawiera bardzo rozbudowaną implementację interaktywnej wody. Wtyczka ta dostarcza zaawansowane narzędzia i funkcjonalności umożliwiające programistom oraz artystom generowanie realistycznej i dynamicznej powierzchni wody w swoich projektach z wykorzystaniem Unreal Engine. Wtyczka wykorzystuje zaawansowane techniki symulacji wody płytkiej, które umożliwiają modelowanie zachowań powierzchni wody oraz interakcje z otoczeniem. Mechanizmy zaimplementowane we wtyczce sprawdzą się w projektach cechujących się stosunkowo małym światem, z uwagi na ograniczenie dotyczące obszaru symulacji. W związku z tym nie zaleca się jej używania w grach opartych na otwartym świecie. Nie pozwala ona również na pełną integrację z istniejącym w silniku systemem wody. W związku z tymi ograniczeniami nie będzie możliwe jej użycie w każdym projekcie. Stanowi jednak przykład łatwej w implementacji technologii pozwalającej na uzyskanie imponującej, interaktywnej powierzchni wody, dlatego postanowiono o niej wspomnieć w niniejszej pracy.

Rozdział 3

Wprowadzenie do sieci neuronowych

W rozdziale tym zawarte jest wprowadzenie teoretyczne do tematyki głębokich sieci neuronowych niezbędne do poprawnego zrozumienia zaimplementowanego rozwiązania.

Sieciami neuronowymi nazywamy zainspirowane budową mózgu statystyczne modele przeznaczone do przetwarzania danych [57]. Sieci neuronowe, dzięki możliwości uczenia na podstawie przykładów oraz automatycznemu uogólnianiu zdobytej wiedzy, które jest skutkiem procesu nauczania sieci, zdolne są do rozwiązywania skomplikowanych zagadnień bez uprzedniej matematycznej ich formalizacji. Cecha ta sprawia, że znalazły zastosowanie w rozwiązywaniu skomplikowanych zadań, które trudno jest opisać matematycznie.

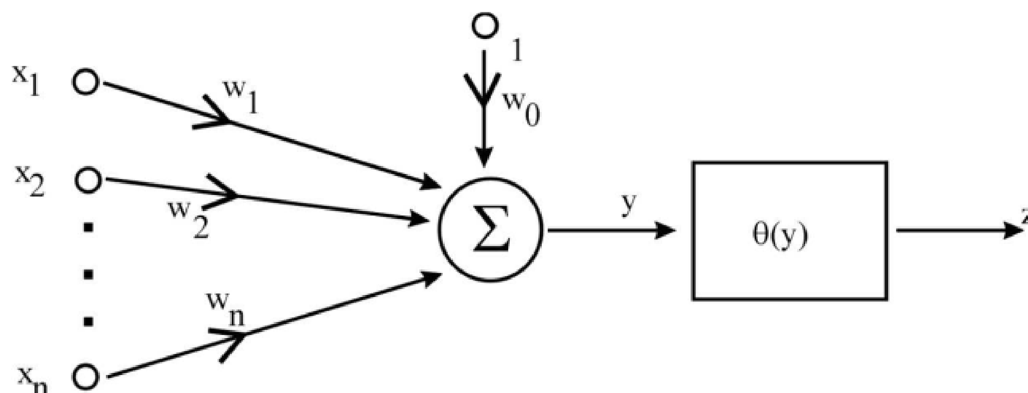
3.1 Budowa sieci neuronowej

Kluczowe dla działania sieci neuronowej jest zaczerpnięte z biologii pojęcie neuronu (rys. 23), za którego przykład posłużyć może Neuron McCullocha-Pittsa złożony z:

- **wejścia** - dane, które będą przetwarzane,
- **wyjścia** - pojedyncza wartość uzyskana jako efekt działania perceptronu,
- **funkcji pobudzenia** - suma wartości uzyskanych po przemnożeniu każdej wartości wejściowej przez odpowiadającą jej wagę (weight) oraz odjęciu odpowiadającego jej obciążenia (bias)
- **funkcji aktywacji** - funkcja, której poddany jest wynik działania funkcji pobudzenia przed zwróceniem go jako wynik działania neuronu.

Pojedyncze neurony reprezentowane są przez odpowiadające im wartości organizowane w macierze, które grupowane są w warstwy. Głęboką siecią neuronową nazywamy taką sieć, która posiada wiele warstw. Powszechnie wykorzystywane są właśnie głębokie sieci neuronowe. W budowie głębokiej sieci neuronowej wyszczególnić należy:

- **warstwę wejściową** - jej wejścia przekazywane są dane wejściowe,
- **warstwy ukryte** - których zazwyczaj jest więcej niż jedna. Ich wejścia i wyjścia obsługiwane są w sposób opisany wcześniej,
- **warstwę wyjściową** - jej wyjście traktuje się jako efekt działania sieci.



Rysunek 23. Schematyczne przedstawienie neuronu: x_n-wejścia neuronu, w_n-wagi neuronu, 1-obciążenie, θ-funckja aktywacji, z-wyjście neuronu. Źródło: [54]

.

Modyfikacja wartości wag neuronów odbywa się podczas procesu uczenia. Jest to operacja, podczas której za pomocą mechanizmu propagacji wstecznej oraz spadku w kierunku gradientu - (Stochastic Gradient Descent, SGD)[57] dzięki funkcji strat obliczającej błąd działania sieci dąży się do uzyskania modelu zdolnego do przybliżania nieliniowych funkcji wielu zmiennych. W uproszczeniu uczenie sieci polega na iteracyjnej zmianie wartości wykorzystywanych przez neurony na podstawie wartości błędu i gradientów. Zasadę działanie SGD prezentuje równanie [38]:

$$w := w - \gamma \frac{\partial}{\partial w} f(w, x_i, y_i) \tag{9}$$

gdzie:

- w - wartość wagi zapisana w neuronie,
- γ - współczynnik prędkości nauki (Learning Rate),
- $f(w, x_i, y_i)$ - funkcja strat.

Efektem działania może być zarówno pojedyncza wartość, jak i macierz wartości o dowolnych wymiarach, mogąca reprezentować na przykład kolorowy obraz.

3.2 Splotowe sieci neuronowe

Splotowe sieci neuronowe to specjalny wariant sieci, stosowany powszechnie w sieciach przeznaczonych do przetwarzania lub generowania obrazów [1]. Wariant ten cechuje się wykorzystaniem operacji splotu. Operacja ta polega na zastosowaniu specjalnej funkcji (tak zwanego filtra), za pomocą której dane wejściowe są przetwarzane częściami, dając w efekcie przetworzoną wersję tych danych. Schematyczne przedstawienie operacji splotu znajduje się na rys. 24. Operację splotu na wartościach dyskretnych opisuje równanie [12]:

$$f|x| * g|x| = \sum_{k=-\infty}^{\infty} f|k| \cdot |g|x - k| \quad (10)$$

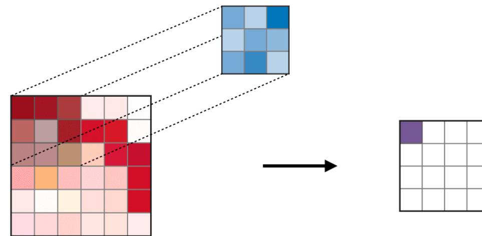
gdzie:

- f - funkcja, poddawana operacji splotu,
- g - funkcja pełniąca rolę filtra.

Z perspektywy sieci neuronowych najważniejszy jest wariant dwuwymiarowy operacji [12]:

$$f|x, y| * g|x, y| = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f|n1, n2| \cdot |g|x - n1, y - n2| \quad (11)$$

Do przykładowych zastosowań splotu zaliczyć można filtr uśredniający oraz filtr wyostrzający, co przedstawiono na rys. 25.

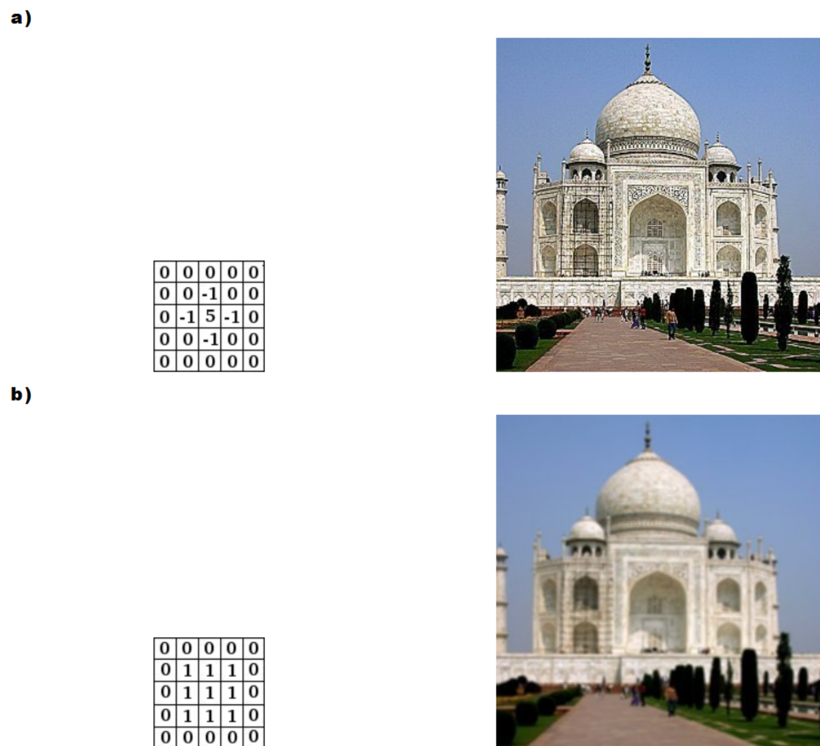


Rysunek 24. Schematyczne przedstawienie operacji splotu. Prostokąt po lewej stronie rysunku reprezentuje macierz poddawaną operacji splotu. Ciemniejszy obszar na tej macierzy reprezentuje dane przekazywane funkcji filtra, który w tym wypadku operuje na macierzy 3x3 i zwracająca pojedynczą wartość. Niebieski prostokąt reprezentuje macierz wykorzystywaną przez filtr. Prostokąt po prawej stronie rysunku reprezentuje macierz będącą wynikiem operacji. Źródło: [12]

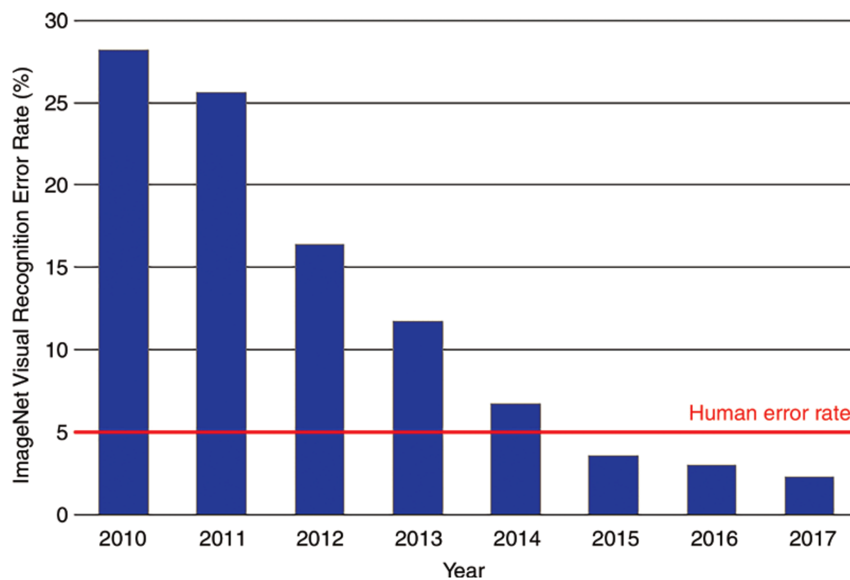
Historia splotowych sieci neuronowych sięga roku 1980 [14] jednak gwałtowną popularność zaczęły zyskiwać po roku 2012, kiedy to sieć AlexNet stworzona przez Alexa Krizhevskiego odniosła zwycięstwo znacząco wyprzedzając konkurencję w zorganizowanym przez Google konkursie ImageNet Large Scale Visual Recognition Challenge [33]. Postęp w tej dziedzinie obrazuje rys. 26. Sieci splotowe są obecnie powszechnie używane do przetwarzania oraz kategoryzacji obrazów.

3.3 Generatywne sieci współzawodniczące

Generatywne sieci współzawodniczące (Generative Adversarial Network, GAN) to sieci neuronowe złożone z dwóch części, będących w istocie niezależnymi sieciami neuronowymi. Zadaniem pierwszej z sieci (Generatora) jest wygenerowanie, zazwyczaj na podstawie dostarczanych do niej zmiennych losowych, danych, które trudno odróżnić od danych referencyjnych. Zadaniem drugiej sieci (Dyskryminatora) jest rozpoznawanie danych wygenerowanych od danych referencyjnych. Pierwsza sieć próbuje nieustannie oszukać drugą, druga natomiast nieustannie poszukuje różnic między wygenerowanymi danymi a danymi "prawdziwymi", które pozwolą jej wykonać poprawnie swoje zadanie. Generatywne



Rysunek 25. Przedstawienie operacji wyostrzenia (a) oraz uśredniania (b) wraz z odpowiadającymi im filtrami. Źródło: [66]



Rysunek 26. Porównanie błędów sieci w konkursie ImageNet Large Scale Visual Recognition Challenge an przestrzeni lat. Czerwona linia wyznacza błąd popełniany przez człowieka. Źródło: [33]

sieci współzawodniczące używane są powszechnie do generowania obrazów. Zarówno Generator jak i Dyskryminator są w takim wypadku najczęściej sieciami splotowymi.

Architektura tego typu została po raz pierwszy zaproponowana w 2014 roku [18]. Ważnym jej rozwinięciem była zaproponowana w 2015 roku [49] sieć Deep Convolutional Generative Adversarial Networks (DGAN). Wiele spośród podanych dalej przykładów opiera swoją architekturę na rozwinięciach tej sieci. Modele tego typu pozwalają na osiągnięcie imponujących rezultatów. Przykładem może być sieć StyleGAN [29] opatrzona logiem firmy NVidia, liczne inne przykłady, między innymi generator obrazów przedstawiających konie, koty czy przekąski, zebrane są na stronie https://thisxdoesnotexist.com/. Przykład wygenerowanej twarzy przedstawia rys. 27

Rysunek 27. Przykładowy obraz twarzy wygenerowanej za pomocą sieci StyleGAN2. Źródło: [67]

Uczenie tego typu sieci potrafi przebiegać w bardzo nieprzewidywalny sposób i cechuje się licznymi trudnościami [52].

Rozdział 4

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 [53]. 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.

4.1 Opis wykorzystanych narzędzi i bibliotek

4.1.1 Unreal Engine

Unreal Engine to jeden z najpopularniejszych silników gry. Silnikiem gry nazywamy rozbudowane narzędzie, będące fundamentem powstających przy jego pomocy produkcji. Silnik zajmuje się interakcją między elementami gry, posiada moduły odpowiedzialne za symulację fizyki, dźwięk, rozbudowane biblioteki obsługujące grafikę czy sztuczną inteligencję. Narzędzie jest bardzo złożone, odpowiada w dużej części za możliwości produkcji, jej wydajność, wygodę pracy oraz liczbę wspieranych przez nią platform. W związku z powyższym tylko większe studia, posiadające duże zespoły programistów, używają własnego silnika, jak CD Projekt Red (Red Engine), Techland (Chrone Engine) czy EA (Frostbite). Zdecydowanie popularniejszym rozwiązaniem jest zakup licencji jednego z uniwersalnych silników dostępnych na rynku np: Unity, Unreal lub Crysis. Istnieją też oparte na otwartym oprogramowaniu alternatywy, jak silnik Godot.

Unreal Engine 4 nie posiada żadnej wbudowanej integracji z sieciami neuronowymi. Silnik został napisany w języku C++, dzięki temu możliwe jest stworzenie integracji z sieciami neuronowymi za pomocą bibliotek Tensorflow dla języka C. W ramach części pracy magisterskiej zdecydowano się na wykorzystanie w tym celu biblioteki CPPFlow, pozwalającej na uruchamianie sieci stworzonych za pomocą biblioteki Tensorflow, napisanej przez Google Brain Team.

Silnik Unreal Engine 4 wybrano ze względu na łatwość jego modyfikacji, wolny dostęp do jego kodu źródłowego, licencję udostępniającą możliwość bezpłatnego wykorzystania w celach niekomercyjnych, dużą popularność i potwierdzone przez znaczną liczbę i różnorodność wykonanych na nim produkcji, możliwości.

4.1.2 SideFX Houdini

Aplikacja Houdini stworzona przez firmę SideFX jest rozbudowanym narzędziem wykorzystywanym głównie w celu tworzenia animacji, efektów specjalnych, symulacji oraz generowaniu zawartości gier. Cechuje go nietypowa reprezentacja danych w postaci grafu skierowanego, duża elastyczność i łatwość rozbudowy za pomocą wtyczek napisanych w języku C++ oraz skryptów wykorzystujących język Python lub język programowania Vex. Vex to język przypominający C, stworzony przez SideFX

na potrzeby aplikacji Houdini. Narzędzie to pozwoliło na szybką generację dużych ilości danych niezbędnych podczas trenowania sieci neuronowej. W pracy wykorzystano wbudowaną w aplikację implementację fizyki płynów FLIP (Fluid-Implicit-Particle).

W aplikacji Houdini wykonano symulację powierzchni wody wokół poruszającego się obiektu. Spośród wielu dostępnych w aplikacji rozwiązań, po wykonaniu licznych eksperymentów, zdecydowano się na wykorzystanie narzędzia Flat Tank. Narzędzie to pozwala na symulację powierzchni płynu w zbiorniku. Podczas symulacji tego rodzaju obliczenia przeprowadzane są wyłącznie dla obszaru płynu znajdującego się na wyznaczonym obszarze oraz nie głębiej, niż określona wartość. Pomijana jest też kwestia dna oraz interakcji płynu z dnem. Oba te zabiegi mają na celu przyspieszenie obliczeń. W tym samym celu wykorzystano również technologię OpenCL, pozwalającą na przeprowadzenie części obliczeń przy użyciu karty graficznej. Zdecydowano się na rozszerzenie symulacji o symulację białej piany wody za pomocą narzędzia Whitewater.

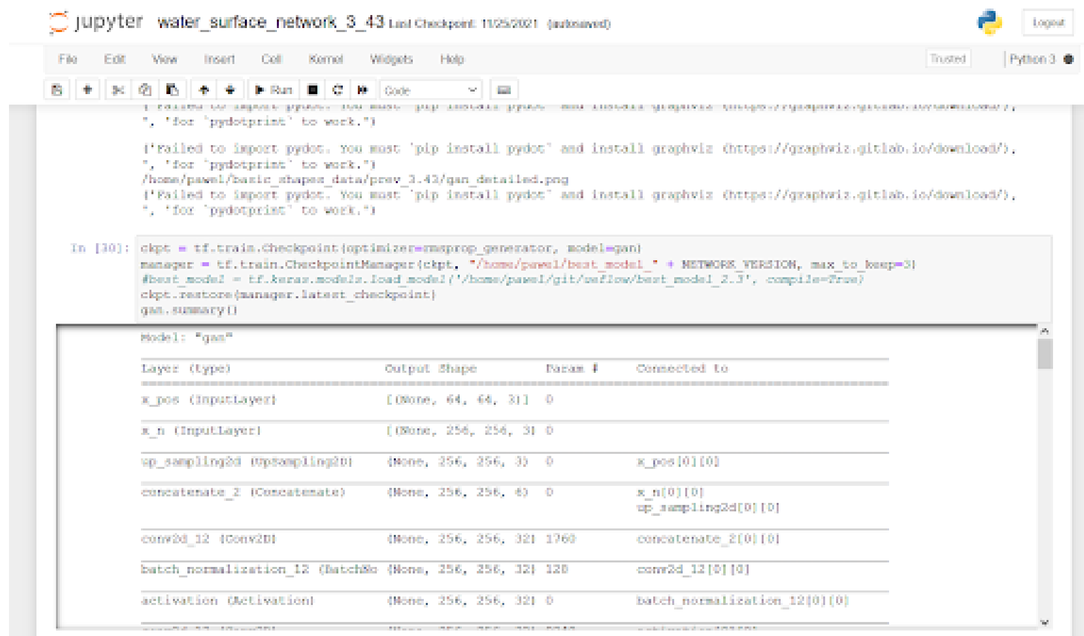
4.1.3 Jupyter Notebook

Jupyter Notebook to narzędzie udostępniane na zasadach Open Source oraz licencji BSD, pozwalające na wykonywanie kodu napisanego w jednym z wielu popularnych języków, jak Python, Julia, czy R, w sposób wygodny dla pracy nad przetwarzaniem danych, ich wizualizacją czy sieciami neuronowymi. Kod zapisywany jest w komórkach, z których każdą wykonywać można w dowolnym momencie i w dowolnej kolejności. Wszystkie z komórek wykonywane są jednak w ramach jednego procesu interpretera co oznacza, że zmienne zdefiniowane lub zmienione w jednej komórce dostępne będą w dowolnej komórce uruchamianej później. Wynik działania komórek może być również wyświetlony bezpośrednio pod nimi, co często jest wygodnym sposobem weryfikacji działania kodu. Jupyter Notebook wspiera też wyświetlanie wykresów, wykonanych w popularnej bibliotece MatPlotLib, wyświetlanie tabeli, w tym bezpośrednie wyświetlanie danych zapisanych w tabelach Pandas, czy obrazów, w tym animowanych. Istnieje również wsparcie rozbudowanych komórek tekstowych, w tym wsparcie dla języka Markup, HTML oraz składni LaTex.

Aplikacja uruchamia serwer, który udostępnia interface dostępny z poziomu przeglądarki. Przykład ekranu aplikacji widoczny jest na rys. 28.

4.1.4 Shader i Fragment Shader

Shader to program wykonywany na karcie graficznej. Shadery są pisane w języku OpenGL ES Shader Language (znanym jako ES SL). ES SL ma własne zmienne, typy danych, kwalifikatory, wbudowane wejścia i wyjścia. Dwa warte wyróżnienia rodzaje shaderów to Vertex Shader oraz Fragment Shader. Vertex Shader to program wykonywany dla każdego wierzchołka przetwarzanej siatki. Można za jego pomocą modyfikować wektory normalne, pozycje, współrzędne w przestrzeni mapowania lub inne parametry wierzchołków. Fragment Shader to kod wykonywany na każdym pikselu fragmentów. Fragmentem natomiast określamy rozpięte między wierzchołkami trójkąty. W programie Fragment Shader mamy dostęp do interpolowanych danych pobliskich wierzchołków i wykonujemy zapytania



Rysunek 28. Przykładowy ekran aplikacji Jupyter Notebook. Źródło: opracowanie własne.

do tekstur. Efektem działania fragment shadera jest ustalenie wartości przetwarzanego pixela, takich jak kolor.

4.1.5 Python

Python jest skryptowym językiem programowania, którego twórcą jest Guido van Rossum. Powstał w roku 1990 i rozwijanym dziś przez fundację Python Software Foundation. Język znalazł szerokie zastosowanie w dziedzinie przetwarzania danych oraz sieci neuronowych z uwagi na swoją prostą składnię, dynamiczne typowanie, szeroką gamę modułów, w tym ogromne zasoby bibliotek ułatwiających przetwarzanie danych oraz związanych z sieciami neuronowymi.

4.1.6 Tensorflow, Keras oraz Tensorboard

Kluczowe dla projektu są biblioteki Tensorflow oraz Keras, wykorzystano również narzędzie Tensorboard w celu ułatwienia oceny przebiegu procesu uczenia się sieci.

Tensorflow jest otwartą biblioteką programistyczną napisaną i rozwijaną przez Google Brain Team, wykorzystywaną w pracy z głębokimi sieciami neuronowymi. Biblioteka wydana została w roku 2015, jej aktualna wersja to 2,7. Biblioteka napisana została w języku C++, potrafi wykorzystywać moc obliczeniową procesorów, kart graficznych czy układów scalonych dedykowanych dla uczenia maszynowego. Posiada API dla języków Python, Javascript, C++ oraz Java.

Keras to kolejna wykorzystana w projekcie biblioteka. Jest to projekt o otwartych źródłach, ułatwiający tworzenie oraz trenowanie głębokich sieci neuronowych. Biblioteka stanowi interface

dla bardziej niskopoziomowego, opisanego wcześniej, Tensorflow. Celem projektu jest stworzenie wygodnego dla programisty, modularnego, łatwo rozszerzalnego narzędzia.

Tensorboard jest narzędziem pozwalającym na wizualizację przebiegu nauczania sieci tworzonych za pomocą biblioteki Tensorflow. Z poziomu przeglądarki możliwe jest dzięki niemu śledzenie wykresów przedstawiających wartość funkcji strat lub dowolnych innych metryk. Posiada narzędzia pozwalające na ocenę wydajności sieci i znalezienie przyczyn ewentualnych problemów z wydajnością. Zaimplementowano w nim również wiele narzędzi wizualizacji wielu rodzajów danych. Przykład ekranu aplikacji widoczny jest na rys. 29

Rysunek 29. Przykładowy ekran aplikacji Tendorboard. Źródło: opracowanie własne.

4.1.7 CPPFlow

CPPFlow to biblioteka umożliwiająca uruchomienie modeli stworzonych przy użyciu biblioteki TensorFlow w programach napisanych w języku C++. Stanowi ona interfejs wykorzystujący pliki binarne biblioteki Tensorflow. Co prawda biblioteka Tensorflow również jest napisana w tym języku, jednak cechuje ją bardzo skomplikowany proces budowania wykorzystujący autorskie narzędzie firmy Google o nazwie Bazel. Proces kompilacji jest w związku z tym złożony i trudny do przeprowadzenia. Dodatkowo wykorzystywany w pracy silnik Unreal Engine również wymaga rozbudowanego proces budowania i kompilacji. W związku z tym zdecydowano się na wykorzystywanie opisywanej biblioteki. Jej ogromną zaletą jest prostota użycia, przykładowy kod uruchamiający sieć rozpoznającą obrazy oraz jej użycie sprowadzić się może do linii opisanych we Frag. 2

```
// Read the graph
cppflow::model model("saved_model_folder");

// Load an image
auto input = cppflow::decode_jpeg(cppflow::read_file(std::string("image.jpg")));

// Cast it to float, normalize to range [0, 1], and add batch_dimension
input = cppflow::cast(input, TF_UINT8, TF_FLOAT);
input = input / 255.f;
input = cppflow::expand_dims(input, 0);

// Run
auto output = model(input);

// Show the predicted class
std::cout << cppflow::arg_max(output, 1) << std::endl;
```

Listing 2. Przykład wykorzystania biblioteki CPPFlow. Źródło: [25]

4.2 Konfiguracja sprzętowa komputera

Wszystkie symulacje oraz naukę sieci przeprowadzono na jednostce o następujących parametrach:

- **Procesor AMD Ryzen 9 3900XT**,
  - 12 rdzeni,
  - 24 wątki logiczne,
  - taktowanie bazowe: 3,8GHz,
  - taktowanie w trybie MaxBoost: 4,7GHz,
  - pamięć Cache:
    - L1: 32kB,
    - L2: 6MB,
    - L3: 64MB,
  - technologia wykonania: TSMC 7nm FinFET,
- **karta graficzna Gigabyte GeForce RTX 3080 GAMING OC 10G**,
  - taktowanie: 1800MHz,
  - pamięć: 10GB GDDR6X ,
- **dysk twardy Samsung 970 Evo**,
  - rodzaj złącza: M.2,
  - pPojemność: 1TB,
- **system operacyjny Linux Ubuntu 20,04,3 LTS**.

4.3 Przygotowanie środowiska

Przygotowanie interpretera języka Python wykorzystywanego przez Jupyter Notebook używany w celu wykonania tej pracy sprowadzało się do zainstalowania odpowiednich bibliotek.

Początkowo wykorzystywano wersję 2,7 interpretera Python wraz z bibliotekami TensorFlow 1, w późniejszym czasie postanowiono jednak wykorzystać nowszą wersję 2,7 biblioteki Tensorflow, co wiązało się ze sporymi zmianami w napisanym kodzie oraz potrzebą reinstalacji serowników wraz z bibliotekami Compute Unified Device Architecture (CUDA) oraz Deep Neural Network library (cuDNN). Wykorzystanie nowej wersji przynosiło znaczne korzyści w związku z większymi możliwościami i udogodnieniami dostępnymi w tej wersji.

Istotne jest dobranie odpowiednich wersji bibliotek oraz dopasowanie zainstalowanych sterowników karty graficznej wraz z bibliotekami CUDA, oraz cuDNN w taki sposób, aby w procesie uczenia wykorzystywana była karta graficzna. Ostateczna konfiguracja wygląda następująco:

- python 3,8,8,
- biblioteka Tenorflow-GPU 2,4,1,
- sterowniki Nvidia w wersji 470,
- biblioteki CUDA 11,0,
- biblioteki cuDNN 8.

Wszelkie zmiany tak przyjętej konfiguracji powodowały zaprzestanie wykorzystywania karty graficznej w procesie uczenia, co spowalniało go do stopnia, w którym był w praktyce niewykonalny.

W celu wykorzystania biblioteki CppFlow w projekcie należało umieścić jego pliki nagłówkowe w odpowiednim miejscu w strukturze projektu. Konieczne było również pobranie bibliotek Ternsorflow dla języka C oraz umieszczenie ich w odpowiednim miejscu w strukturze projektu. Dostępne są dwie ich wersje: wykorzystująca procesor oraz procesor graficzny. Obie z nich z powodzeniem uruchamiano, w ostatecznej wersji rozwiązania wykorzystano biblioteki wykorzystujące procesor graficzny z uwagi na wyższą wydajność tego rozwiązania. Kolejnym istotnym krokiem było zadbanie o to, by kompilacja silnika ignorowała ostrzeżenia generowane przez moduł CppFlow. Unreal Engine skonfigurowany jest w taki sposób, by traktować wszystkie ostrzeżenia jako błędy, ponieważ zazwyczaj takie podejście wymusza wyższą jakość kodu. Konieczne było więc wprowadzenie pewnych zmian w plikach nagłówkowych biblioteki Tensorflow. Zdecydowano się na usunięcie linii 942 do 944 w pliku nagłówkowym `tensorflow/c/c_api.h` oraz dodanie brakującej linii $\#include < iterator >$

Rozdział 5

Przeprowadzone prace

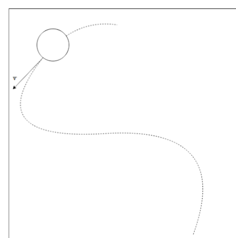
W rozdziale tym opisano prace mające na celu stworzenie interaktywnej symulacji wody przy wykorzystaniu sieci neuronowych.

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.

5.1 Prosta symulacja powierzchni wody

5.1.1 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.30.



Rysunek 30. Schematyczne przedstawienie prostej symulacji. Źródło: opracowanie własne.

Symulację wykonano za pomocą aplikacji SideFX Houdini. Wykorzystano w tym celu narzędzie Flat Tank [59] 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.

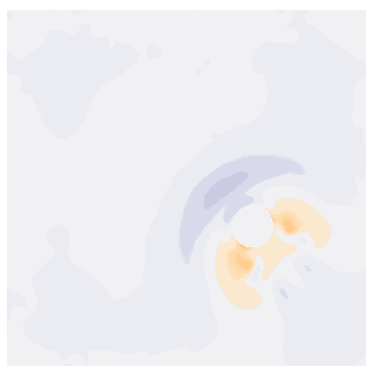
Animację przemieszczenia przeszkody wykonano wykorzystując skrypt napisany w języku Python generujący krzywą, oraz dostępne w aplikacji narzędzie przesuwające obiekt wzdłuż ścieżki w miarę postępu animacji. Skrypt opierał się na wykorzystaniu generatora liczb losowych, co pozwalało na uzyskanie różnych 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.

5.1.2 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. 31) reprezentująca efekt końcowy rozpatrywanej klatki,
- trzy wartości typu float reprezentujące położenie przeszkody w klatce o indeksie N.

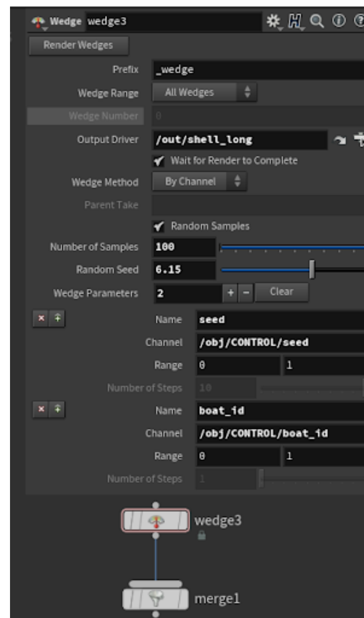


Rysunek 31. Wizualizacja macierzy reprezentującej wysokość tafli wody. Źródło: opracowanie własne.

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. 32. Rezultat symulacji przetworzono oraz zapisano przy pomocy grafu dostępnego w repozytorium projektu.



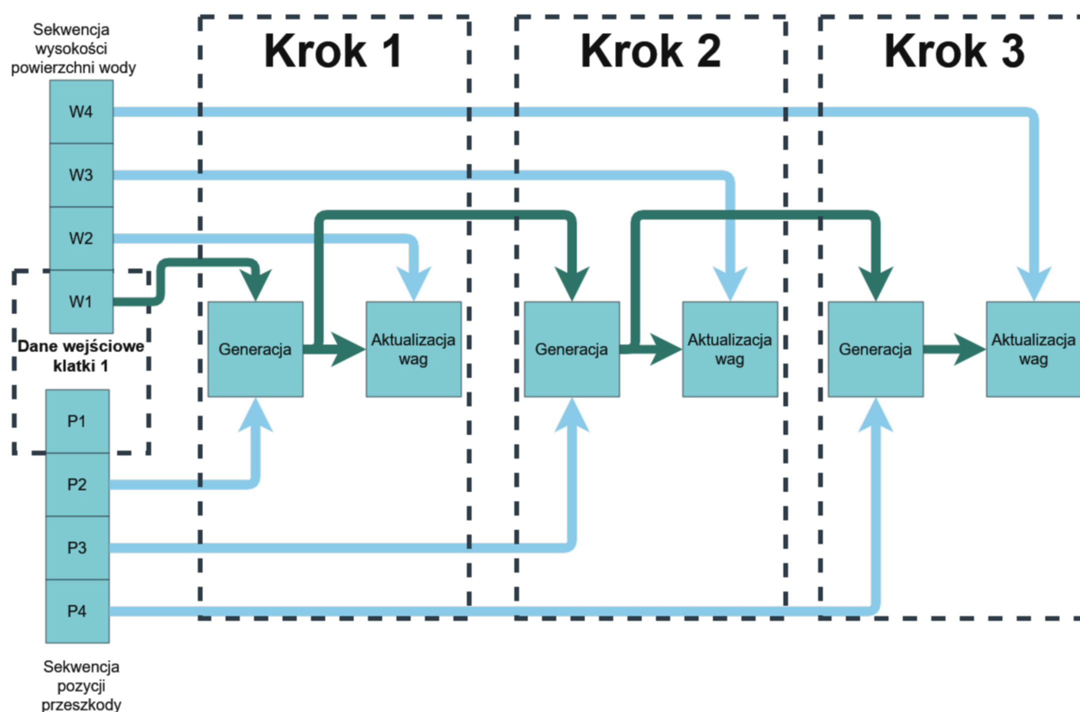
Rysunek 32. Przykład grafu wykorzystującego narzędzie wedge. Źródło: opracowanie własne.

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 [63]. 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. 33.



Rysunek 33. Graf wykorzystany w celu przetwarzania i zapisania danych opisujących powierzchnię symulowanej wody. Źródło: opracowanie własne.

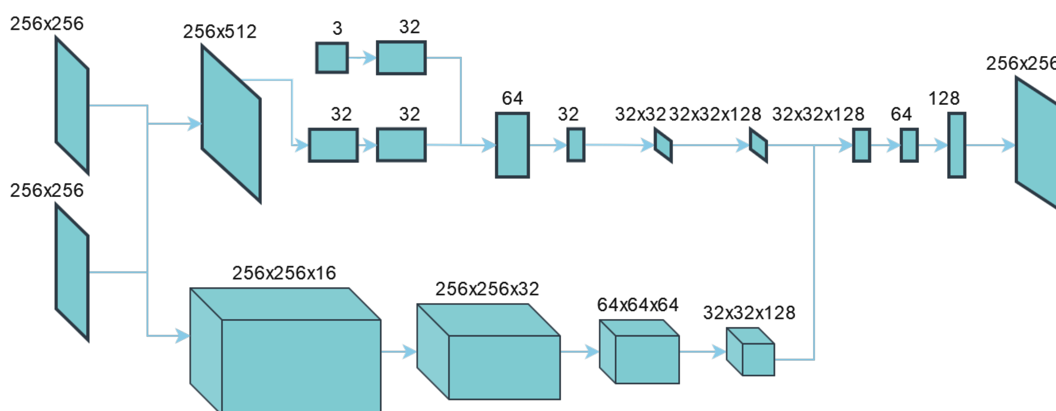
W projekcie wykorzystano osobne potoki Tensorflow Dataset dla obu rodzajów danych wejściowych. Potoki te następnie połączono przy użyciu funkcji Dataset.zip, po czym wykorzystano operację shuffle w celu randomizacji danych przy każdej epoce. Dane wygenerowane w procesie animacji przechowywane były w 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. 34



Rysunek 34. 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

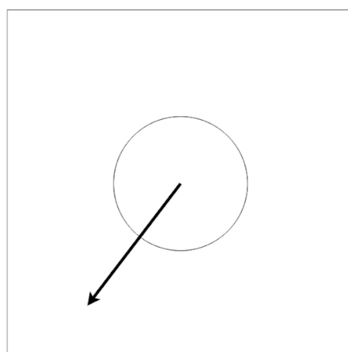
epokach w okolicach liczby 0,0013. Empiryczne badania wyników generacji sieci pozwoliły na stwierdzenie, że sieci nie generują fal, stopniowo zbliżają natomiast wartości do zera. Uznano, że sieć nie będzie w stanie wykonać symulacji zadanego scenariusza i postanowiono go zmodyfikować.

5.1.3 Wprowadzone modyfikacje, uzyskanie zadowalających wyników

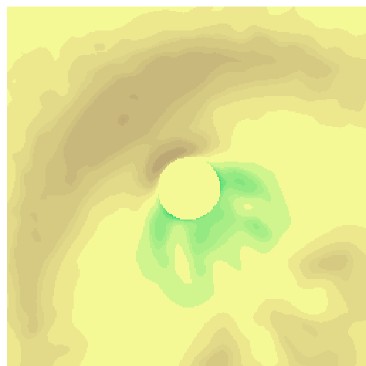
W celu określenia przyczyn porażki przyjrzano się istniejącym rozwiązaniom. Zaobserwowano, że wiele z nich [50][24] opiera się na istnieniu nieruchomej przeszkody. Można przypuszczać, że pozwala to sieci na zrozumienie struktury rozważanej przestrzeni. Można spekulować, że sieć miała utrudnione zadanie 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. 35, natomiast przykład uzyskanych danych widać na rys. 36.



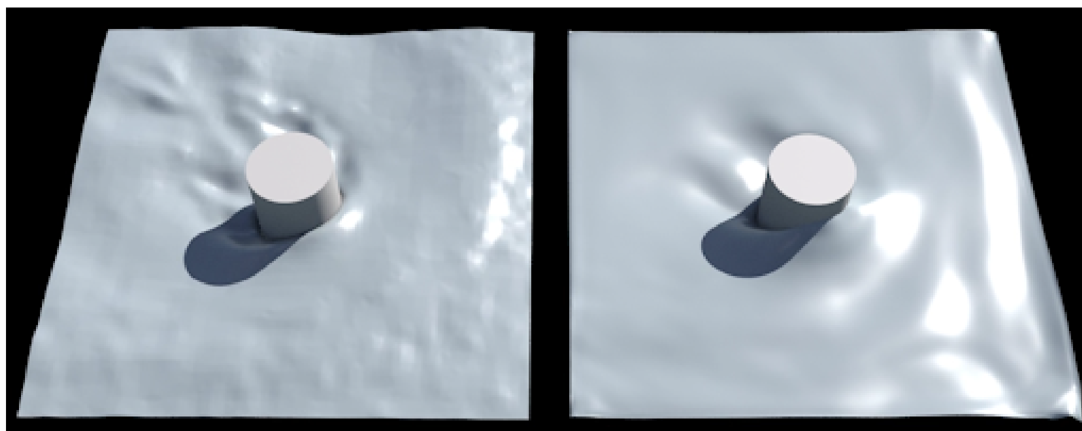
Rysunek 35. Schematyczne przedstawienie zmodyfikowanej prostej symulacji. Źródło: opracowanie własne.



Rysunek 36. 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 [51] dzięki której stworzono sieć bazującą na architekturze U-Net [32]. 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. 37. Zdecydowano się na przygotowanie złożonej funkcji strat nakładającej kary za takie zachowanie.



Rysunek 37. 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 3. 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.

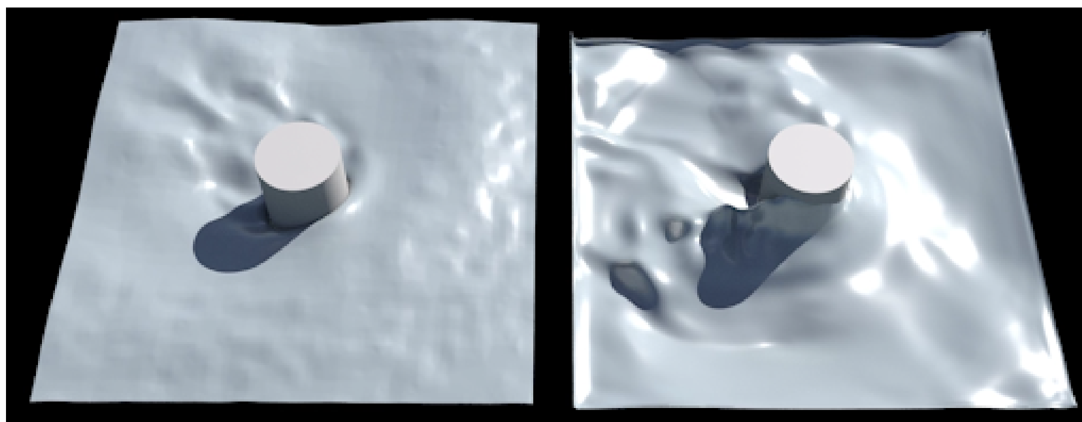
Znaczącą poprawę spowodowało zastosowanie bardziej złożonej architektury, wykorzystującej mechanizmy typowe dla sieci GAN, wykorzystano sieć pełniącą rolę dyskryminatora. Przeprowadzone eksperymenty pozwoliły ustalić, że jej wykorzystanie zmusza sieć do generowania wspomnianych szczegółów, nawet jeśli 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. 38. 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.

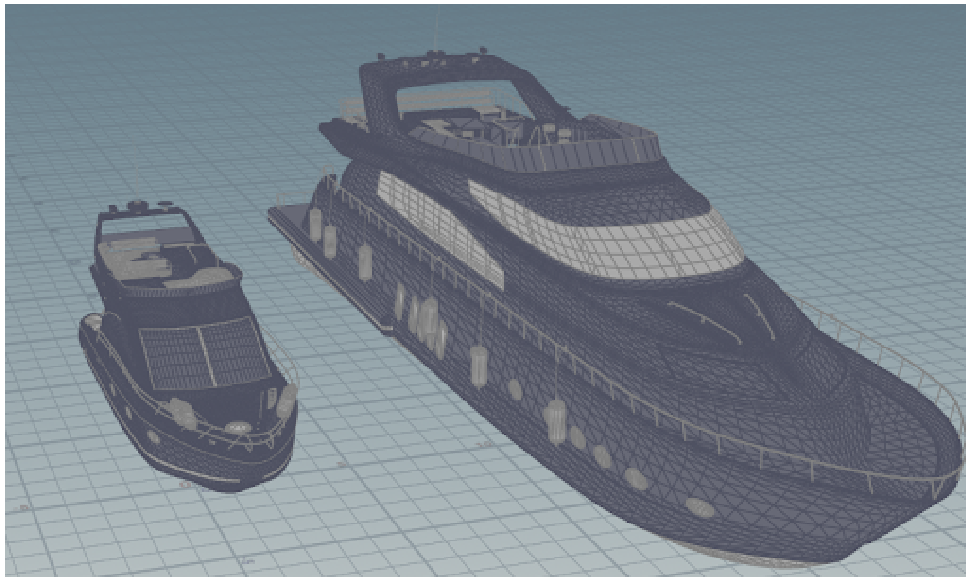


Rysunek 38. Przykład rezultatu działania sieci, przy której uczeniu wykorzystano architekturę wykorzystującą sieć dyskryminatora oraz złożoną funkcję strat. Po lewej stronie widoczne są dane referencyjne, prawa strona przedstawia rezultat działania sieci. Źródło: opracowanie własne.

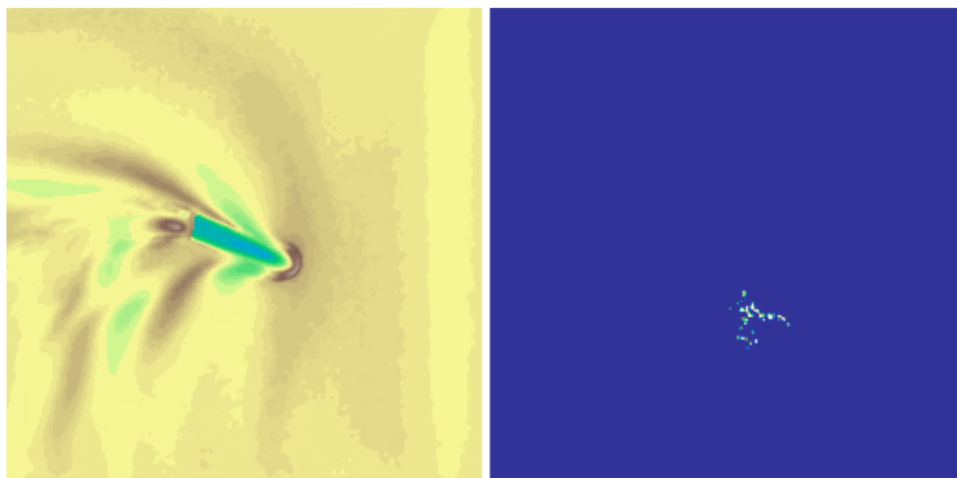
5.2 Złożona symulacja powierzchni wody

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. 39. 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. 40.



Rysunek 39. Przykład danych będących efektem symulacji. Obraz z lewej strony przedstawia wysokość powierzchni wody, po prawej stronie znajduje się reprezentacja gęstości piany. Źródło: opracowanie własne.

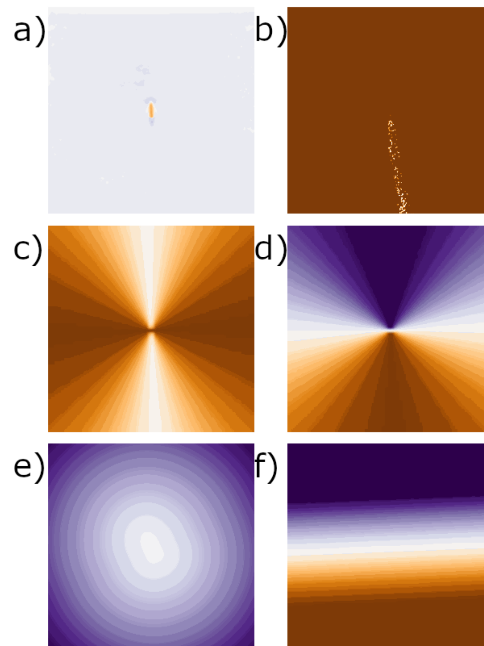


Rysunek 40. Modele łodzi wykorzystane w symulacji.

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. 41 Źródło: opracowanie własne.:

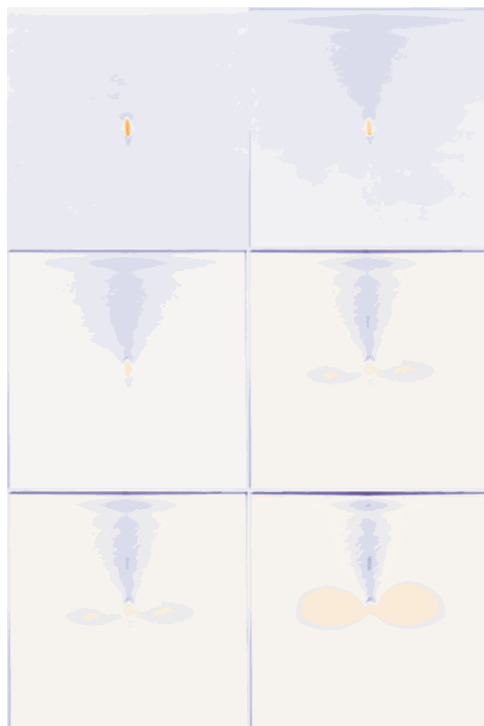
- wysokość powierzchni wody w klatce poprzedzającej rozpatrywaną (a),
- wartość natężenia piany w klatce poprzedzającej rozpatrywaną (b),
- macierz zawierającą wartości odległości od powierzchni dna łodzi (c),
- trójwymiarową macierz o rozmiarach 64x64x3 której wartości oznaczały:
  - wartości funkcji 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.



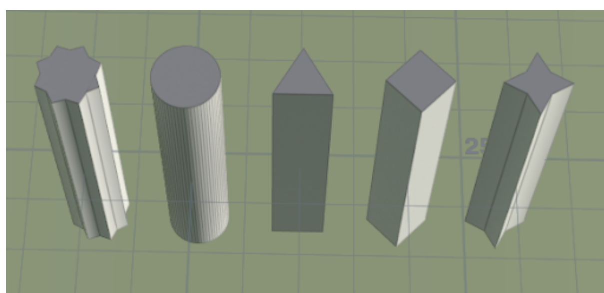
Rysunek 41. Wizualizacja danych przekazywanych sieci. Źródło: opracowanie własne.

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

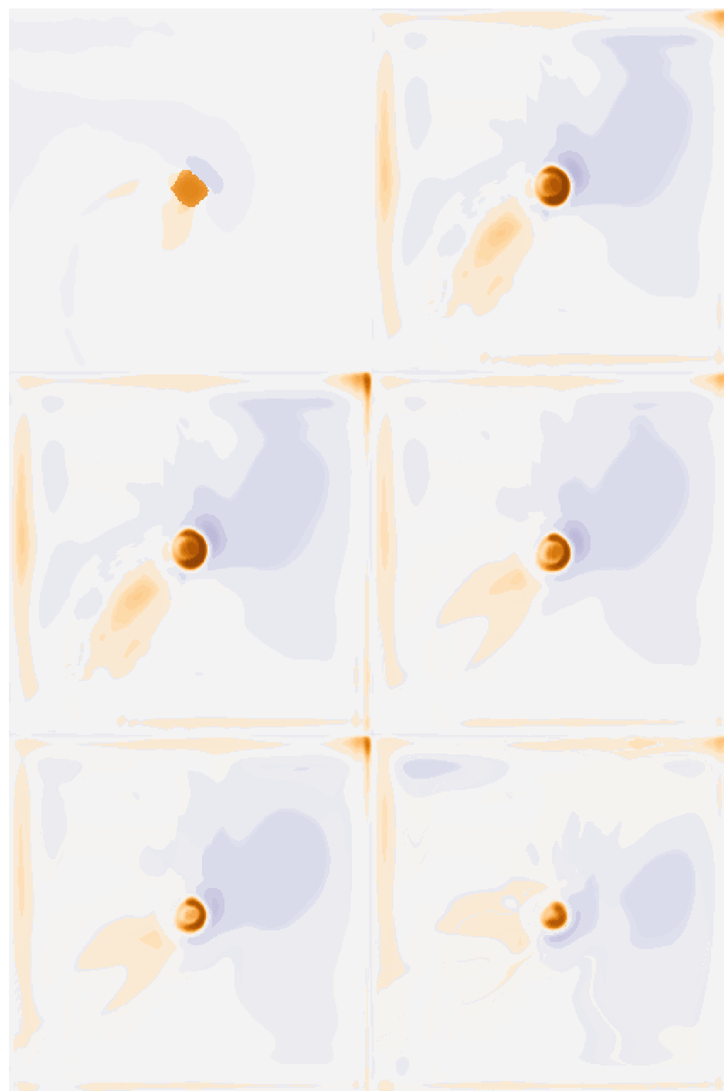
Zdecydowano się na znaczne uproszczenie animacji. Zamiast skomplikowanego kształtu kadłuba wykorzystano kilka rodzajów przeszkód o prostych kształtach przekrojów. Geometrie przeszkód zestawiono na rys. 43. Przeszkody nie obracały się, ich orientacja w przestrzeni pozostawała stała. Przypuszczano, że 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 43. 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. 44.



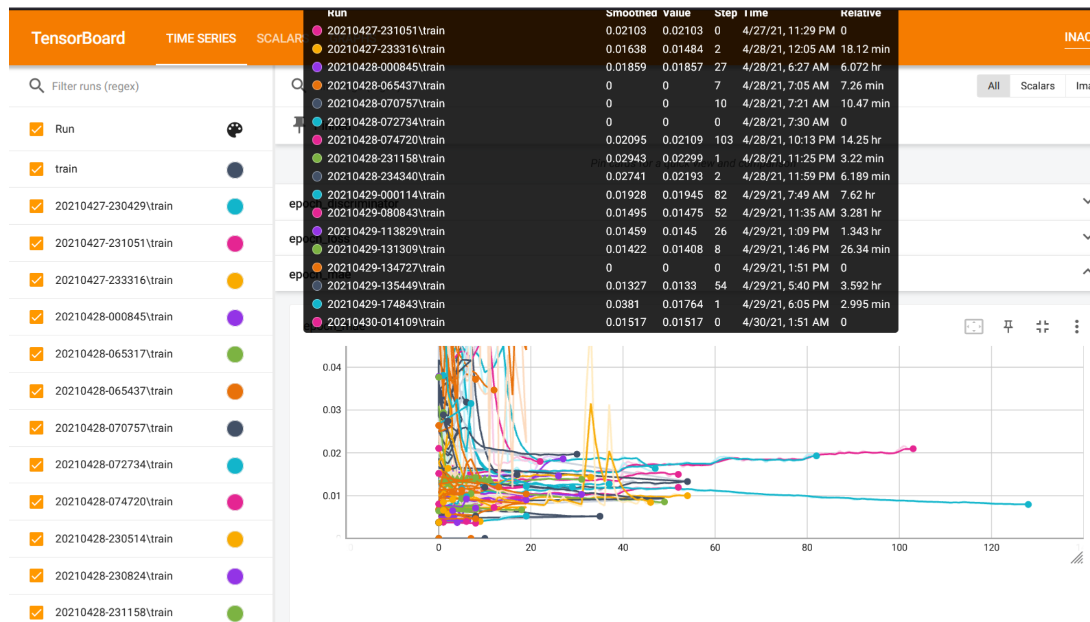
Rysunek 44. Wybrane klatki sekwencji wartości wysokości powierzchni wody wygenerowanej przez sieć neuronową. Źródło: opracowanie własne.

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

5.3 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. 45 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 45. Zrzut ekranu z aplikacji Tensorboard przedstawiający część zapisów procesu trenowania różnych wariantów sieci. Źródło: opracowanie własne.

5.4 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

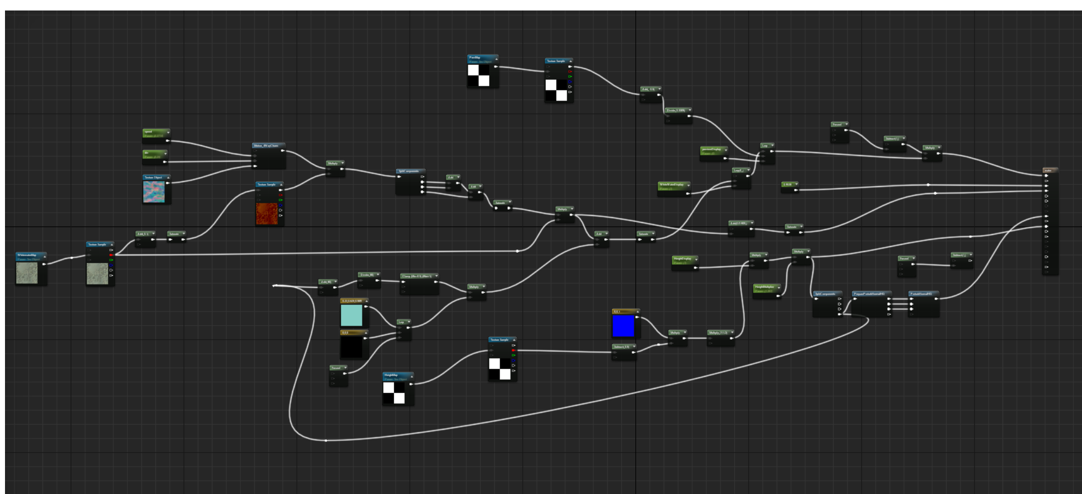
w tym celu klasie ATensorFlowNetwork. Obiekt reprezentujący powierzchnię wody tworzony jest podczas wykonywania konstruktora tej klasy.

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 ATensorFlow-Network 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. 46.

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 46. Graf reprezentujący materiał powierzchni wody. Źródło: opracowanie własne.

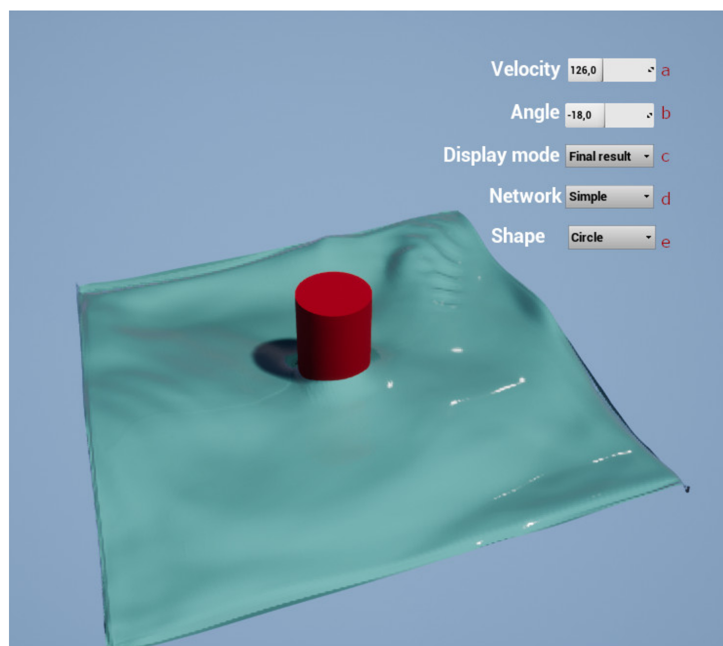
5.5 Przedstawienie oraz analiza wyników

5.5.1 Przedstawienie wyników

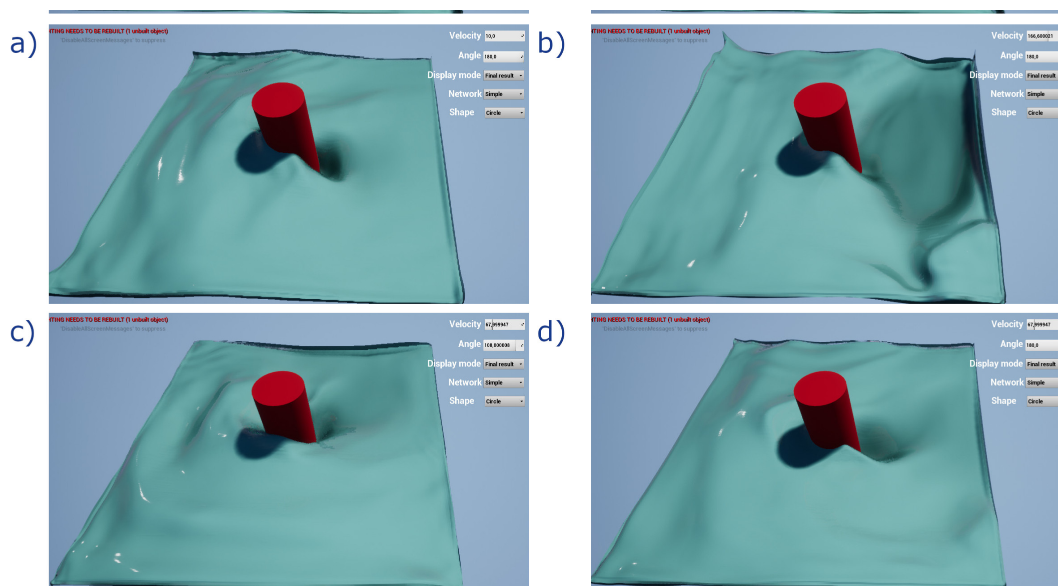
Interaktywna demonstracja działania sieci neuronowej jest w stanie wygenerować powierzchnię wody reagującą na zmianę wartości takich parametrów jak prędkość czy kierunek poruszania się przeszkody. Najważniejsze elementy interfejsu demonstracji przedstawiono na rys. 47. Widoczne są na nim elementy służące 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. 49:
  - 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).

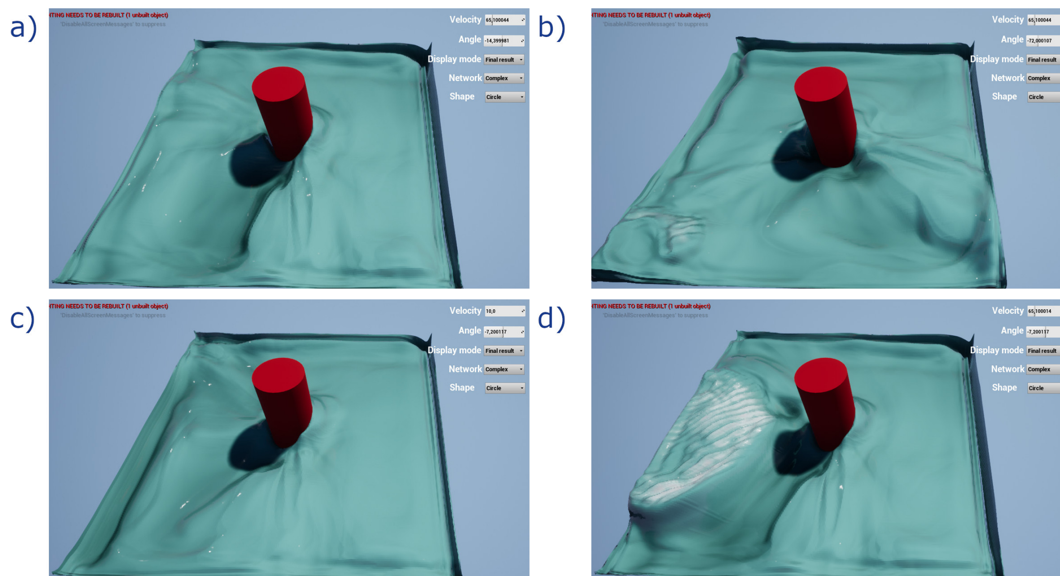
Na rys. 48 oraz rys. 49 przedstawiono przykłady wygenerowanej powierzchni wody, oraz wpływ parametrów symulacji na jej kształt przy wykorzystaniu odpowiednio prostego, oraz złożonego wariantu symulacji.



Rysunek 47. Ekran interaktywnej demonstracji. Źródło: opracowanie własne.

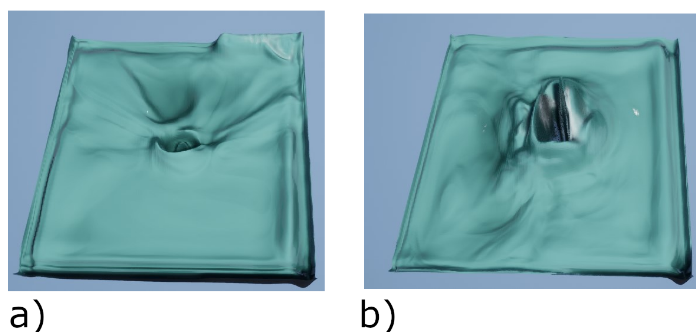


Rysunek 48. Prezentacja wpływu różnych wartości parametrów symulacji na wygląd powierzchni wody generowanej za pomocą prostej sieci neuronowej: obrazy a oraz b prezentują wpływ zmiany prędkości poruszania się przeszkody przy zachowaniu kierunku, obrazy c oraz d prezentują wpływ zmiany kierunku poruszania się przeszkody przy zachowaniu prędkości jej przemieszczania się. Źródło: opracowanie własne.



Rysunek 49. 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. 50

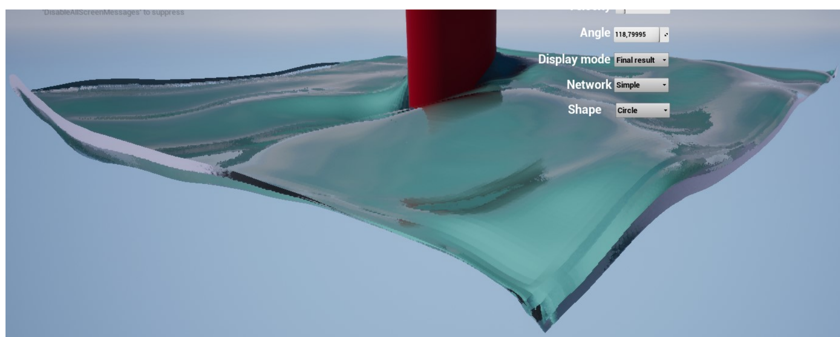


Rysunek 50. Porównanie rezultatów wykonania symulacji dla przeszkody o przekroju w kształcie okręgu (a), więc tym, na którym wykonywane były symulacje używane przez nią podczas nauki, oraz o przekroju kwadratowym(b). Sieć w pewnym stopniu uwzględnia nowy kształt przekroju, jednak zdecydowanie widoczne jest pogorszenie jakości rezultatu. Źródło: opracowanie własne.

5.5.2 Analiza jakościowa wyników

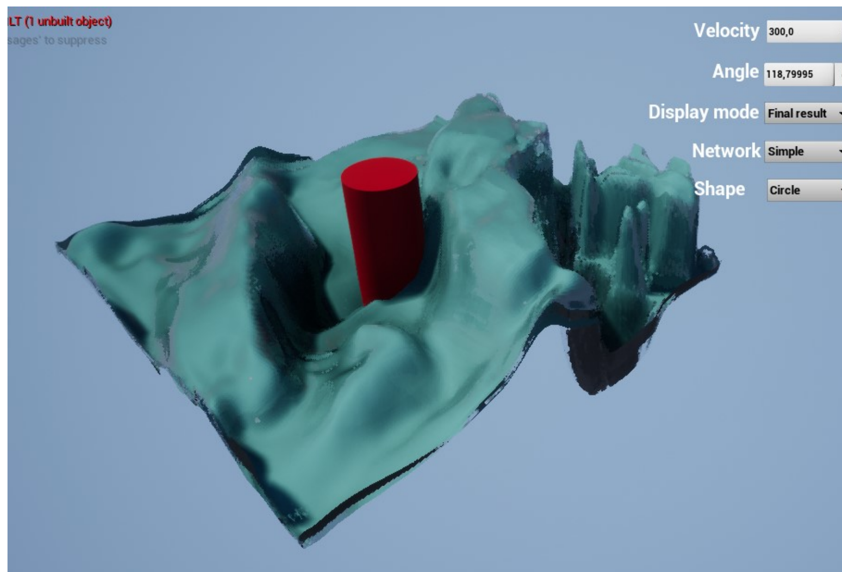
Przedstawione wcześniej przykłady powierzchni wody wygenerowane zostały przy zastosowaniu wartości parametrów symulacji zawartych w zakresie tych, które wykorzystywano podczas generowania zbioru danych użytego w procesie uczenia sieci. Ocena ich jakości nie może jednak polegać wyłącznie na wygenerowaniu 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. 51 Błąd ten można w łatwy sposób ukryć zmniejszając obszar wyświetlanej powierzchni.



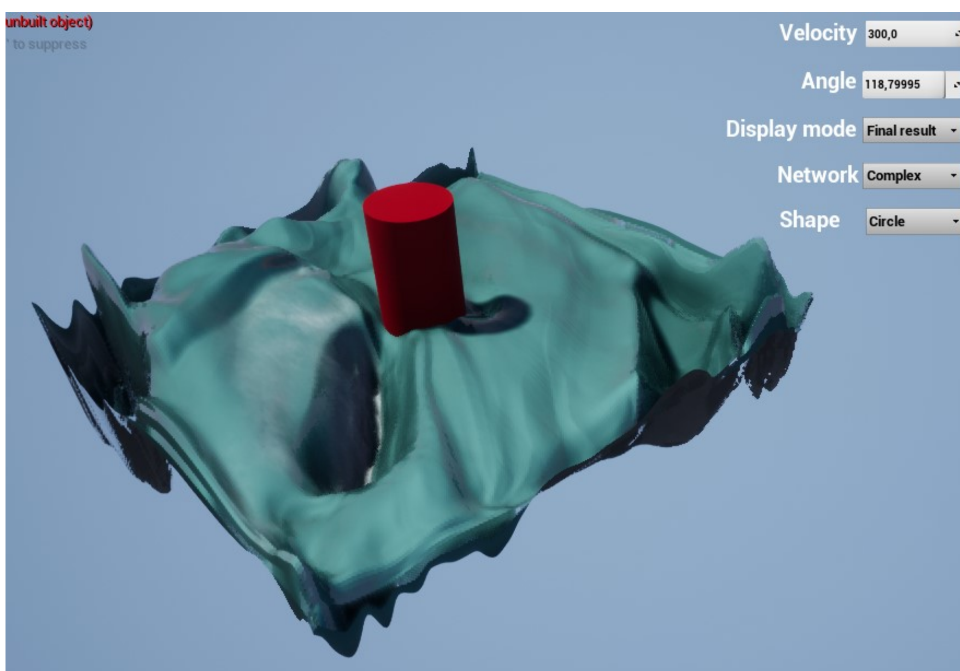
Rysunek 51. Zbliżenie krawędzi symulowanej powierzchni przedstawiające powstałe w tym obszarze artefakty. Źródło: opracowanie własne.

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. 53 oraz rys. 53 Obecność tego efektu wskazuje na niską zdolność sieci do generalizacji.



Rysunek 52. 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 53. 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. 54

Rysunek 54. Przykład artefaktów występujących na obszarach powierzchni pokrytych gęstą pianą. Źródło: opracowanie własne.

5.5.3 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 [74][74]:

- 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

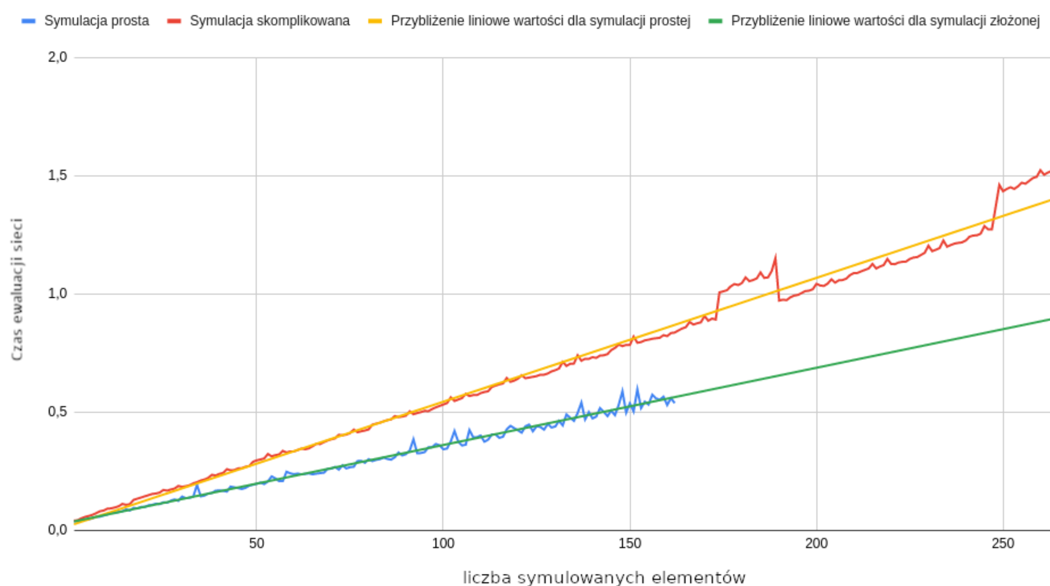
Wykres widoczny na rys. 55 prezentuje wpływ liczby wykonywanych jednocześnie symulacji na czas ewaluacji sieci. Zwiększenia liczby wykonywanych symulacji dokonano poprzez rozbudowę danych wejściowych. Dane zebrano jedynie przy użyciu stacji roboczej wyposażonej w kartę GeForce RTX 3080. Widoczne na wykresie 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 12 natomiast przybliżenie liniowe wartości uzyskanych dla wariantu sieci odpowiedzialnego za prostą symulację określa wzór 13.

$$\mathrm{y} = 3,66 * 10^{-2}\mathrm{x} + 3,26 * 10^{-3} \tag{12}$$

$$\mathrm{y} = 5,23 * 10^{-3}\mathrm{x} + 2,26 * 10^{-2} \tag{13}$$

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

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

5.6.1 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:

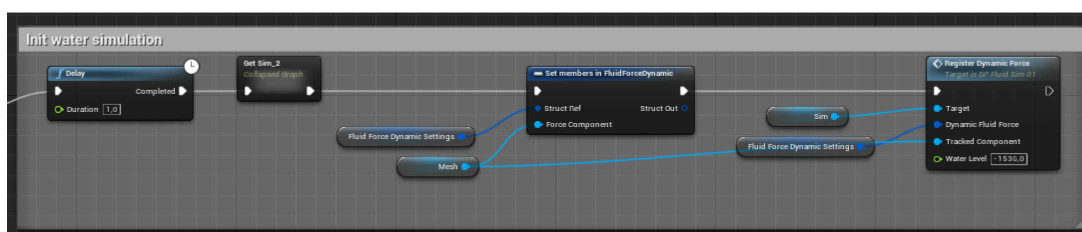
1. W pliku `FluidSimulation/Materials/Simulation/M_Fluid_Sim_01` należy zmienić parametr FoamForceRT w taki sposób, by wskazywał na obiekt :

   `/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. 56. 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 56. Wspomniana w tekście sekwencja elementów grafu przeszkody. Źródło: opracowanie własne.

5.6.2 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.

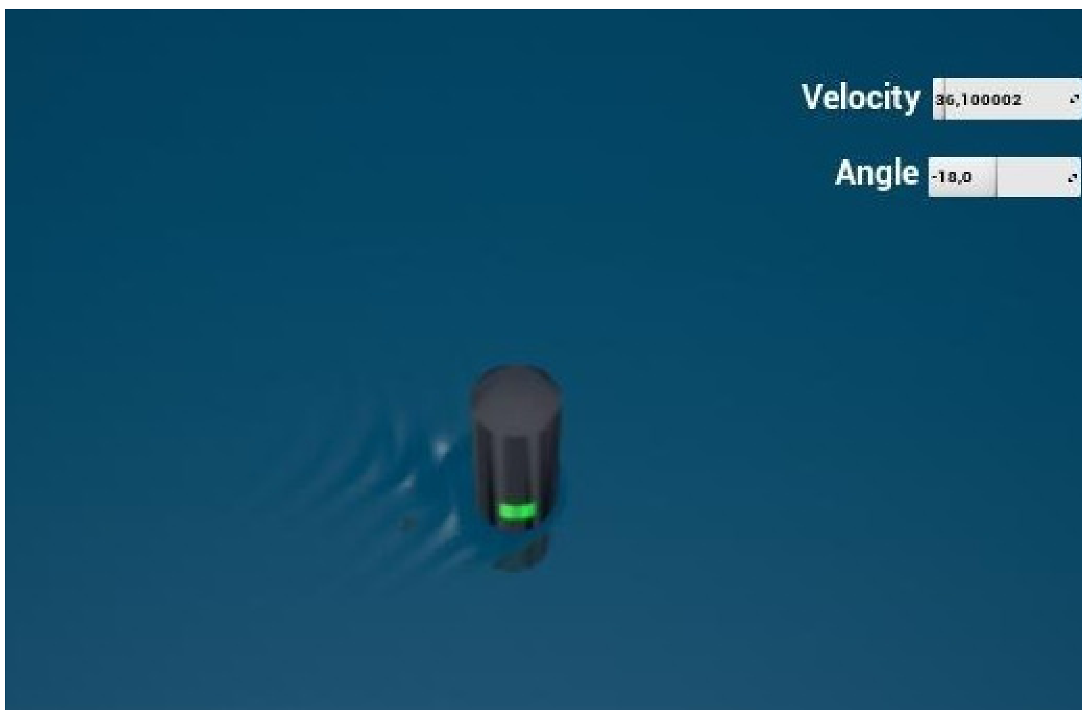
Symulacja opiera się na prostej propagacji fali w celu symulacji śladu torowego. Nie uwzględnia kształtu obiektu, jedynie jego położenie oraz przesunięcie w czasie. Za pomocą tych dwóch wartości ustalany jest wpływ, jaki obiekt powinien mieć na powierzchnię wody oraz miejsce, w którym wpływ ten powinien 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.

5.6.3 Przedstawienie wyników

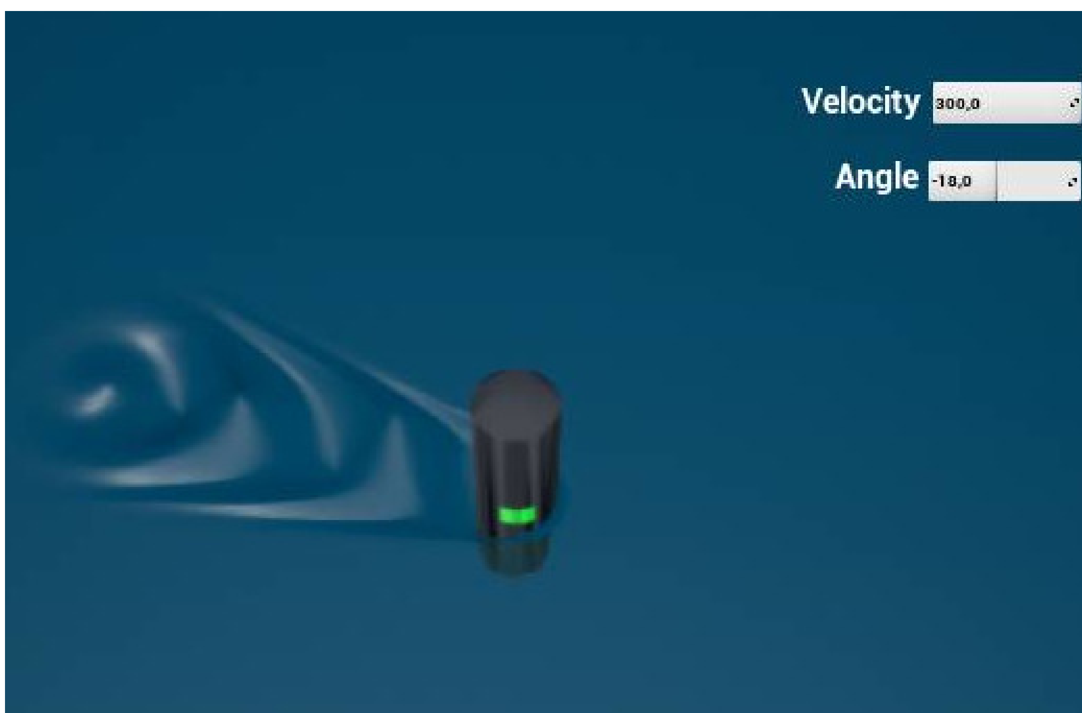
W tej sekcji pracy przedstawiono porównania rezultatów działania implementacji symulacji powierzchni wody opartej na pluginie „Water" podczas symulacji podobnych warunków. Zdecydowano się na taką formę porównania ich. ponieważ obiektywna ocena jakości działania implementacji nie jest możliwa. Możliwe 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. 57 oraz rys. 58. 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.



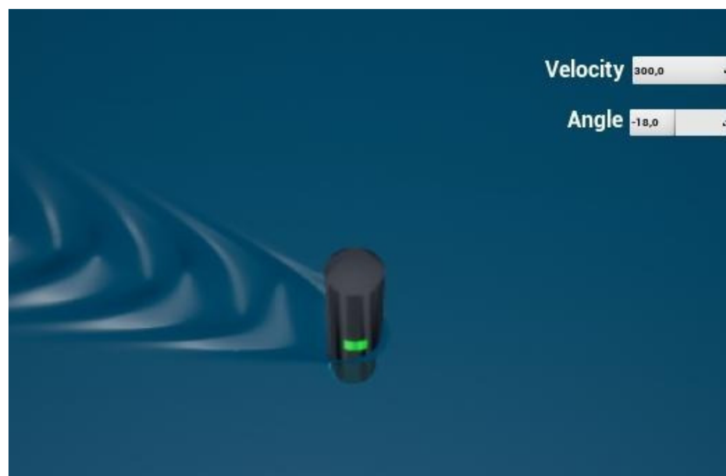
Rysunek 57. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne.



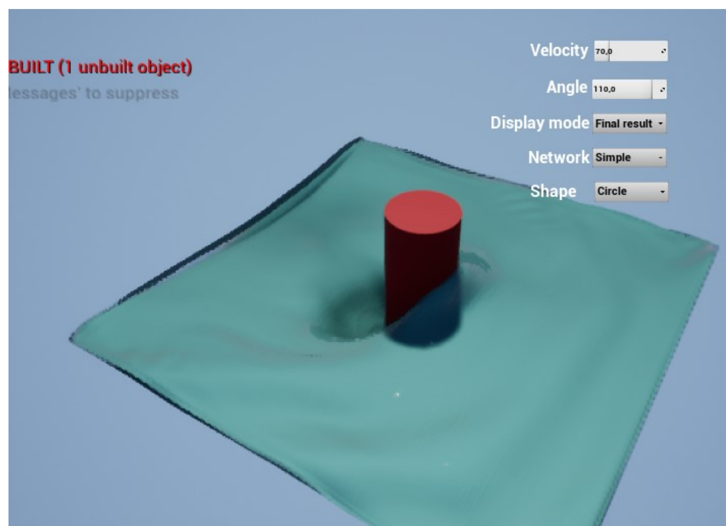
Rysunek 58. Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne.

Ruch linii prostej z większą prędkością

Rezultaty działania obu sieci podczas ruchu w linii prostej ze znaczną prędkością przedstawia rys. 59 oraz rys. 60. 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.



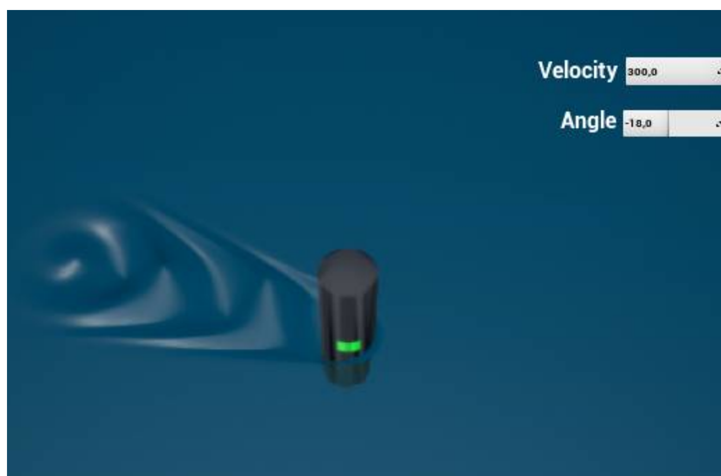
Rysunek 59. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne.



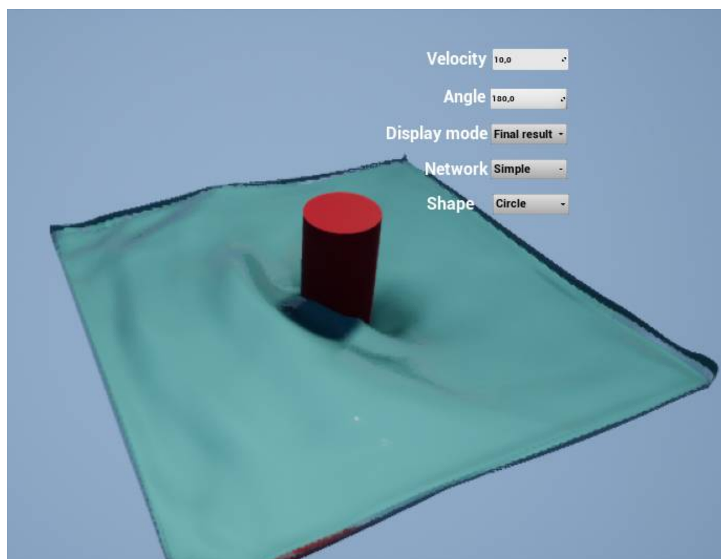
Rysunek 60. Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne.

Szybkie zmniejszenie prędkości

Rezultaty działania obu sieci podczas ruchu w linii prostej przy znacznym zmniejszeniu prędkości przedstawia rys. 61 oraz rys. 62. 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 61. Rezultaty działania implementacji opartej na pluginie Water podczas ruchu przy znacznym zmniejszeniu prędkości. Źródło: opracowanie własne.



Rysunek 62. 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 6

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.

6.1 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] Afshine, A. i Shervine, A., *Convolutional Neural Networks cheatsheet, Stanford CS 230 - Deep Learning*, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks data dostępu: 12.12.2021.

[2] *Animated flat, www.doom.fandom.com*, https://doom.fandom.com/wiki/Animated_flat data dostępu: 13.12.2021.

[3] Asghar, S., *GPT3 Website*, https://gpt3.website/ data dostępu: 12.12.2021.

[4] Bernie 2018, *X-Isle Dinosaurs Island Demo Crytek 2001*, https://www.youtube.com/watch?v=zAi2Nigk9gs data dostępu: 13.12.2021.

[5] Brunton, S. L., Noack, B. R. i Koumoutsakos, P., *Annual Review of Fluid Mechanics Vol. 52:477-508*. 2020, s. 477–508.

[6] By Wikipedians, *3D Rendering*. PediaPress, s. 105.

[7] CD Projekt Red, *Patch 1.5 & Next-Generation Update — list of changes*, https://www.cyberpunk.net/en/news/41435/patch-1-5-next-generation-update-list-of-changes data dostępu: 14.12.2021.

[8] Crigz Vs Game Dev, *WATER SIMULATION in Godot 4*, https://www.youtube.com/watch?v=VSwVwIYEypY data dostępu: 03.02.2023.

[9] CRYENGINE Team, *TBT – Check out the original Far Cry launch trailer in glorious HD*, https://www.cryengine.com/news/view/-tbt-check-out-the-original-far-cry-launch-trailer-in-glorious-hd data dostępu: 13.12.2021.

[10] Davidka, P., *PolyBump w dokumentacji silnika Crytec*, https://docs.cryengine.com/display/SDKDOC2/Polybump data dostępu: 13.12.2021.

[11] DeepMind, *Tobias Pfaff (DeepMind): Learning to Simulate Complex Physics with Graph Networks*, https://www.youtube.com/watch?v=8v27_jzNynM data dostępu: 03.02.2023.

[12] Dektar, Katie and Levoy, Marc and Adams, Andrew, *Spatial convolution*, http://scroll.stanford.edu/courses/cs178-14/applets/convolution.html data dostępu: 12.12.2021.

[13] Fournierand, A. i Reeves, W. T., *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, s. 75–84.

[14] Fukushima, K., „Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”, *Biol. Cybernetics36, 193 202*, 1980.

[15] Generated Media, Inc., *Strona Generated Photos*, https://generated.photos/faces data dostępu: 12.12.2021.

[16] Gerasev, A., *Repozytorium FlowNN*. https://github.com/agerasev/flownn data dostępu: 13.09.2023.

[17] Golinad, *Julia Islands*, https://www.shadertoy.com/view/Nl2Gzw data dostępu: 14.12.2021.

[18] Goodfellow, I. J. i in., „Generative Adversarial Networks”, *arXiv:1406.2661 [stat.ML]*, 2014.

[19] Google Inc., *Case study: AlphaGo*, https://deepmind.com/research/case-studies/alphago-the-story-so-far data dostępu: 12.12.2021.

[20] *Gra Doom w serwisie doom.fandom.org*, https://doom.fandom.com/wiki/Animated_flat data dostępu: 13.12.2021.

[21] *Gra Sonic the Hedgehog w serwisie wikipedia.org*, https://en.wikipedia.org/wiki/Sonic_the_Hedgehog data dostępu: 13.12.2021.

[22] *Gra Xevious w serwisie strategywiki.org*, https://strategywiki.org/wiki/Xevious data dostępu: 13.12.2021.

[23] Hao, X. i Shen, L., *Journal of Fluid Mechanics, vol. 4*. 1958, s. 426–434, 10.1017/S0022112058000550.

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

[25] Izquierdo, S., *Projekt CPPFlow*, https://github.com/serizba/cppflow data dostępu: 14.12.2021.

[26] Jeschke, S., Skřivan, T., Müller-Fischer, M., Chentanez, N., Macklin, M. i Wojtan, C., *ACM Transactions on GraphicsVolume 37, Issue 4, August 2018 Article No.: 94*. 2018, s. 1–13.

[27] Jeschke, S. i Wojtan, C., *ACM Transactions on GraphicsVolume 36, Issue 4, Article No.: 103*. 2017, s. 1–12.

[28] Karl, K., *The Water in Red Dead Redemption 2*, https://www.youtube.com/watch?v=50MoJBi-ywM data dostępu: 03.02.2023.

[29] Karras, T., Lain, S. i Ail, T., „A style-based generator architecture for generative adversarial network”, *CVPR 2018*, 2018.

[30] Kochkov, D. i Smith, J. A., *Machine learning-accelerated computational fluid dynamics*, 2021.

[31] Komisarek, K., *Wtyczka Fluid Flux*, https://unrealengine.com/marketplace/en-US/product/fluid-flux data dostępu: 13.12.2021.

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

[33] Langlotz, C. P., Allen, B., Erickson, B. i Kalpathy-Cramer, J., „A Roadmap for Foundational Research on Artificial Intelligence in Medical Imaging: From the 2018 NIH/RSNA/ACR/The Academy Workshop", *Radiology 291(3):190613*, 2019.

[34] Lee, K. i Carlberg, K., *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*, 2018. arXiv: 1812.08373 [cs.CV].

[35] Lemos, C., *Tutorial: How Normal Maps Work & Baking Process*, https://docs.cryengine.com/display/SDKDOC2/Polybump data dostępu: 13.12.2021.

[36] Ling, J., Kurzawski, A. i Templeton, J., *Reynolds averaged turbulence modelling using deep neural networks with embedded invariance*, 2016.

[37] Mastin G. A. , Watterberg P. A., Mareda J. F., *IEEE Computer Graphics and Applications, vol. 7, no. 3*. 1987, s. 16–23, 10.1109/MCG.1987.276961.

[38] Mielniczuk, J., *WYKŁAD: Estymacja funkcji regresji. Sieci neuronowe*, https://home.ipipan.waw.pl/j.mielniczuk/AML_Regression_II_final.pdf data dostępu: 12.12.2021.

[39] Mnih, V. i Kavukcuoglu, K., *Human-level control through deep reinforcement learning*. Nature Research, 2015, t. 518, s. 529–533.

[40] Movieclips Trailers, *Titanic 3D Re-Release Official Trailer #1*, https://www.youtube.com/watch?v=kVrqfYjkTdQ&t=57s data dostępu: 14.12.2021.

[41] Nelson, L. M., *ACM SIGGRAPH Computer GraphicsVolume 15, Issue 3*. 1981, s. 317–324.

[42] Nintendo, *Mario through the years*, https://mario.nintendo.com/history/ data dostępu: 13.12.2021.

[43] NVidia, *NVIDIA PhysX website*, https://developer.nvidia.com/physx-sdk data dostępu: 12.12.2021.

[44] Old Classic Retro Gaming, *Arcade Game: Xevious (1982 Namco)*, https://www.youtube.com/watch?v=VpyFHvPizYM&t=176s data dostępu: 13.12.2021.

[45] Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. i Battaglia, P. W., „Learning Mesh-Based Simulation with Graph Networks.", *arXiv:2010.03409 [cs.LG]*, 2020.

[46] Plutecki, Z., Sattler, P. i Ryszczyk, K., „160 INNOWACYJNY PROCES PROJEKTOWANIA I OPTYMALIZACJI PRODUKTÓW Z WYKORZYSTANIEM SOLVERA ADJOINT", *Konferencja Przemysł 4,0 a Zarządzanie i Inżynieria Produkcji*, 2015, http://www.ptzp.org.pl/files/konferencje/kzz/artyk_pdf_2015/T1/t1_0160.pdf data dostępu: 12.12.2021.

[47] Prantl, L., Ummenhofer, B., Koltun, V. i Thuerey, N., *Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics*, https://arxiv.org/abs/2210.06036 data dostępu: 12.12.2021.

[48] ProsafiaGaming, *Super Mario 64 HD - Full Game Walkthrough*, https://www.youtube.com/watch?v=Yahw-4bFYjY&t=2896s data dostępu: 13.12.2021.

[49] Radford, A., Metz, L. i Chintala, S., „Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", *arXiv:1511.06434 [cs.LG]*, 2015.

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

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

[52] *Real World GANs: Common Problems*, https://developers.google.com/machine-learning/gan/problems data dostępu: 13.12.2021.

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

[54] Rosenblatt, F., „The Perceptron, a perciving and recognizing automation", *Cornel Aeronautical Labolatory, Inc Report No. 85-460-1*, 1957.

[55] *Równania Naviera-Stokesa*, https://pl.wikipedia.org/wiki/Równania_Naviera-Stokesa data dostępu: 28.05.2023.

[56] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J. i Battaglia, P. W., *Advances in Human Action Recognition: A Survey*, 2020. arXiv: 2002.09405 [cs.CV].

[57] Sato, K., *Understanding neural networks with TensorFlow Playground*, https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground data dostępu: 12.12.2021.

[58] Shahriar, S., „GAN Computers Generate Arts? A Survey on Visual Arts, Music, and Literary Text Generation using Generative Adversarial Network", *arXiv:2108.03857v2 [cs.AI]*, 2021.

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

[60] Staadt, O., „Real-Time Open Water Environments with Interacting Objects.", 2009.

[61] Stam, J., „Real-Time Fluid Dynamics for Games", 2003, Corpus ID: 9353969.

[62] Studio WildCard's Mark Mihelich & NVIDIA Tim Tcheblokov, *Wakes, Explosions and Lighting: Interactive Water Simulation in Atlas*, https://www.youtube.com/watch?v=Dqld965-Vv0&t=2518s data dostępu: 03.02.2023.

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

[64] Tessendorf, J., „Simulating Ocean Water", *SIG-GRAPH'99 Course Note*, 2001.

[65] Tessendorf, J., *eWave: Using an Exponential Solver on the iWave Problem*, https://people.computing.clemson.edu/~jtessen/reports/papers_files/ewavealgorithm.pdf data dostępu: 03.02.2023.

[66] The GIMP Documentation Team, *Convolution Matrix, Gimp*, https://docs.gimp.org/2.6/en/plug-in-convmatrix.html data dostępu: 13.12.2021.

[67] *thispersondoesnotexist*, https://thispersondoesnotexist.com/ data dostępu: 13.12.2021.

[68] Thomsen, M., *Microsoft's Deep Learning Project Outperforms Humans In Image Recognition*, https://www.forbes.com/sites/michaelthomsen/2015/02/19/microsofts-deep-learning-project-outperforms-humans-in-image-recognition/?sh=60db8a91740b data dostępu: 12.12.2021.

[69] Thundereus, *Nvidia island Water Techdemo DirectX11 / ATI Readon 5870M Full HD*, https://www.youtube.com/watch?v=cCN6BIYEkr8 data dostępu: 14.12.2021.

[70] Um, K., Hu, X. i Thuerey, N., *Liquid Splash Modeling with Neural Networks*, https://onlinelibrary.wiley.com/doi/10.1111/cgf.13522 data dostępu: 12.12.2021.

[71] Underground Gaming Entertainment, *Sonic the Hedgehog (1991)*, https://www.youtube.com/watch?v=JqQYDLcvkBc&t=419s data dostępu: 13.12.2021.

[72] Vorobiev, A. i Barkovoi, A., *Brief account of FarCry v.1.2 tests and of the first Shader 3.0 implementation*, http://ixbtlabs.com/articles2/gffx/fc12.html data dostępu: 13.09.2023, 2004.

[73] Wang, R. i Yu, R., *Physics-Guided Deep Learning for Dynamical Systems: A Survey*, 2021. arXiv: 2107.01272 [cs.CV].

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

[75] Yuksel, C., House, D. H. i Keyser, J., *Wave Particles*, http://www.cemyuksel.com/research/waveparticles/ data dostępu: 14.12.2021.

[76] *Zibra Liquids*, https://zibra.ai/ data dostępu: 14.12.2021.

Wykaz skrótów i symboli

ANN	sztuczna sieć nauronowa (ang. Artificial Neural Network)
CNN	splotowa sieć neuronowa (ang. Convolutional Neural Network)
FLOPS	liczba operacji zmiennoprzecinkowych na sekundę
GPU	procesor graficzny (ang. Graphical Processing Unit)

Spis rysunków

1 Kadr z gry Xavious(1982 rok), na którym widoczna jest powierzchnia wody przedstawiona przy pomocy niebieskiego obszaru pokrytego białym wzorem. Źródło: [44] . . 4
2 Kadr z gry Sonic the Hedgehog (1991 rok), na którym widoczna jest animowana woda będąca elementem tła. Źródło: [71] 4
3 Kadr z gry Doom (1993 rok), na którym widoczne są powierzchnie kilku cieczy, zobrazowane za pomocą animowanej tekstury nałożonej na płaszczyznę. Źródło: [2] . 5
4 Kadry z gry Mario 64 (1996 rok), na którym widoczna jest powierzchnia wody, efekty cząsteczkowe oraz symulacja rozchodzących się fal wody. Źródło: [48] 5
5 Schematyczne przedstawienie działania mapowania normalnych. Źródło: [35] 7
6 Kadr z demonstracji technologicznej X-Isle Dinosaurs Island (2004 rok). Źródło: [4] . 8
7 Wizualizacja fali określonej równaniem. Źródło: 1 9
8 Wizualizacja fali określonej równaniem. Źródło: 2 10
9 Proces rysowania trochoidy za pomoc toczącego się okręgu. M — kierunek obrotu okręgu, Cw — Kierunek rysowania krzywej. Źródło: [35] 10
10 Powierzchnia powstała przez użycie pojedynczej fali Gerstnera jako wartości przemieszczenia. Źródło: opracowanie własne. 11
11 Powierzchnia powstała przez użycie superpozycji wielu fal Gerstnera jako wartości przemieszczenia. Źródło: opracowanie własne. 11
12 Powierzchnia powstała przez użycie szumu Perlina jako wartości przemieszczenia. Wykorzystano implementację algorytmu szumu Perlina wbudowaną w aplikację SideFX Houdini. Źródło: opracowanie własne. 12
13 Kadr z dema technicznego Island Direct3D 11 Tech Demo. Źródło: [69] 13
14 Kadr z filmu Titanic. Źródło: [40] 14
15 Przedstawienie sposobu tworzenia czoła fali na podstawie cząsteczek. Źródło: [75] . . 15
16 Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75] 16
17 Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75] 17
18 Przykład powierzchni wody wygenerowany opisywaną metodą. Źródło: [75] 17
19 Kadr przedstawiający efekt symulacji wykonanej za pomocą narzędzia Zibra Liquids. Źródło: [76] 20
20 Kadry przedstawiające porównanie wyników działania grafowej sieci neuronowej stworzonej przez zespół DeepMind z wynikiem symulacji. Źródło: [56] 21

21 Kadry przedstawiające porównanie wyników działania grafowej sieci neuronowej stworzonej przez zespół DeepMind z wynikiem symulacji. Źródło: [56] 21
22 Architektura sieci splotowego autoenkodera, który przyjmuje stan układu dynamicznego jako wejście i produkuje przybliżony stan jako wyjście. Stan reprezentowany przez x wykorzystany został w obliczeniach. Źródło: [34] 22
23 Schematyczne przedstawienie neuronu: $\mathrm{x_n}$-wejścia neuronu, $\mathrm{w_n}$-wagi neuronu, l-obciążenie, θ-funckja aktywacji, z-wyjście neuronu. Źródło: [54] 26
24 Schematyczne przedstawienie operacji splotu. Prostokąt po lewej stronie rysunku reprezentuje macierz poddawaną operacji splotu. Ciemniejszy obszar na tej macierzy reprezentuje dane przekazywane funkcji filtra, który w tym wypadku operuje na macierzy 3x3 i zwracająca pojedynczą wartość. Niebieski prostokąt reprezentuje macierz wykorzystywaną przez filtr. Prostokąt po prawej stronie rysunku reprezentuje macierz będącą wynikiem operacji. Źródło: [12] 27
25 Przedstawienie operacji wyostrzenia (a) oraz uśredniania (b) wraz z odpowiadającymi im filtrami. Źródło: [66] 28
26 Porównanie błędów sieci w konkursie ImageNet Large Scale Visual Recognition Challenge an przestrzeni lat. Czerwona linia wyznacza błąd popełniany przez człowieka. Źródło: [33] 28
27 Przykładowy obraz twarzy wygenerowanej za pomocą sieci StyleGAN2. Źródło: [67] . 29
28 Przykładowy ekran aplikacji Jupyter Notebook. Źródło: opracowanie własne. 34
29 Przykładowy ekran aplikacji Tendorboard. Źródło: opracowanie własne. 35
30 Schematyczne przedstawienie prostej symulacji. Źródło: opracowanie własne. 39
31 Wizualizacja macierzy reprezentującej wysokość tafli wody. Źródło: opracowanie własne. 40
32 Przykład grafu wykorzystującego narzędzie wedge. Źródło: opracowanie własne. . . . 42
33 Graf wykorzystany w celu przetwarzania i zapisania danych opisujących powierzchnię symulowanej wody. Źródło: opracowanie własne. 43
34 Schematyczne przedstawienie rekurencyjnego wariantu sieci. Źródło: opracowanie własne. 44
35 Schematyczne przedstawienie zmodyfikowanej prostej symulacji. Źródło: opracowanie własne. 45
36 Wizualizacja macierzy reprezentującej wysokość tafli wody. Źródło: opracowanie własne. 46
37 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. 46
38 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. 48

39 Przykład danych będących efektem symulacji. Obraz z lewej strony przedstawia wysokość powierzchni wody, po prawej stronie znajduje się reprezentacja gęstości piany. Źródło: opracowanie własne. . . . 49
40 Modele łodzi wykorzystane w symulacji. . . . 49
41 Wizualizacja danych przekazywanych sieci. Źródło: opracowanie własne. . . . 51
42 Wybrane klatki sekwencji wartości wysokości powierzchni wody wygenerowanej przez sieć neuronową. Źródło: opracowanie własne. . . . 52
43 Geometria przeszkód użytych w symulacji. . . . 52
44 Wybrane klatki sekwencji wartości wysokości powierzchni wody wygenerowanej przez sieć neuronową. Źródło: opracowanie własne. . . . 53
45 Zrzut ekranu z aplikacji Tensorboard przedstawiający część zapisów procesu trenowania różnych wariantów sieci. Źródło: opracowanie własne. . . . 54
46 Graf reprezentujący materiał powierzchni wody. Źródło: opracowanie własne. . . . 55
47 Ekran interaktywnej demonstracji. Źródło: opracowanie własne. . . . 57
48 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. . . . 57
49 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. . . . 58
50 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. . . . 58
51 Zbliżenie krawędzi symulowanej powierzchni przedstawiające powstałe w tym obszarze artefakty. Źródło: opracowanie własne. . . . 59
52 Przykład artefaktów generowanych przez prosty wariant sieci po znacznym zwiększeniu wartości prędkości poruszania się przeszkody poza zakres wykorzystany podczas uczenia sieci. Źródło: opracowanie własne. . . . 60
53 Przykład artefaktów generowanych przez złożony wariant sieci po znacznym zwiększeniu wartości prędkości poruszania się przeszkody poza zakres wykorzystany podczas uczenia sieci. Źródło: opracowanie własne. . . . 60
54 Przykład artefaktów występujących na obszarach powierzchni pokrytych gęstą pianą. Źródło: opracowanie własne. . . . 61
55 Przedstawienie zależności pomiędzy czasem ewaluacji sieci oraz liczba wykonywanych symulacji. Źródło: opracowanie własne. . . . 64

56 Wspomniana w tekście sekwencja elementów grafu przeszkody. Źródło: opracowanie własne. 65
57 Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne. 67
58 Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej z niewielką prędkością. Źródło: opracowanie własne. 67
59 Rezultaty działania implementacji opartej na pluginie Water podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne. 68
60 Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu w linii prostej ze znaczną prędkością. Źródło: opracowanie własne. 68
61 Rezultaty działania implementacji opartej na pluginie Water podczas ruchu przy znacznym zmniejszeniu prędkości. Źródło: opracowanie własne. 69
62 Rezultaty działania implementacji opartej na sieci neuronowej podczas ruchu przy znacznym zmniejszeniu prędkości. Źródło: opracowanie własne. 69

Spis tabel

1 Zmierzone dane opisujące wydajność działania sieci na kartach GeForce RTX 3080 oraz GeForce RTX 2060 Mobile dla dwóch wariantów symulacji. 63