Logo

Description automatically generated

**Damian Andrysiak**

Kierunek: informatyka

Specjalność: informatyka stosowana

Numer albumu: 374341

**Badanie efektywności zwiększania rozdzielczości obrazów za pomocą sieci neuronowych**

Praca magisterska

wykonana pod kierunkiem

dr. hab. prof. nadzw. UŁ.

Pawła Kowalczyka

w Katedrze Fizyki Ciała Stałego

WFiIS UŁ

**Łódź 2023**

# Spis treści

[Spis treści 2](#_Toc134307250)

[1. Wstęp 3](#_Toc134307251)

[2. Zbiór danych 5](#_Toc134307252)

[2.1. Proces generowania danych 5](#_Toc134307253)

[2.2. Przygotowanie silnika do generowania danych 9](#_Toc134307254)

[2.3. Przygotowanie projektu do generowania danych 10](#_Toc134307255)

[2.4. Uruchomienie procesu generowania danych 11](#_Toc134307256)

[2.5. Wygenerowane dane 11](#_Toc134307257)

[3. Siec neuronowa 12](#_Toc134307258)

[3.1. Wczytywanie danych uczących 12](#_Toc134307259)

[3.2. Normalizacja danych uczących 15](#_Toc134307260)

[3.3. Proces normalizacji danych 16](#_Toc134307261)

[4. Podsumowanie i bibliografia 19](#_Toc134307262)

[4.1. Podsumowanie 19](#_Toc134307263)

[4.2. Bibliografia 19](#_Toc134307264)

# Wstęp

Podstawowym celem pracy pt. „Badanie efektywności zwiększania rozdzielczości obrazów z wykorzystaniem sieci neuronowych” będzie zbadanie możliwości powiększania obrazów HDR z rozdzielczości 1080p (tj. 1920x1080) do rozdzielczości 4k (tj. 3840x2160 pikseli) z wykorzystaniem sieci neuronowych.

Na samym wstępie warto wspomnieć o technice, dzięki której można uzyskać zwiększenie rozdzielczości zdjęć, a jest nią SISR (eng. Single Image Super Resolution - Uzyskanie Super Rozdzielczości z Pojedynczego Obrazu). Metoda ta wykorzystuje głębokie sieci konwolucyjne do uzyskania lepszej jakości i rozdzielczości z obrazu o niższej rozdzielczości. W odróżnieniu od metod czysto analitycznych takich jak interpolacja najbliższych sąsiadów (eng. Nearest-neighbors), bilionowa (eng. Bilinear) lub bikubiczna (eng. Bicubic), sieci neuronowe są w stanie nauczyć się bardziej złożonych zależności miedzy pikselami przez co generowane obrazy są wyższej jakości i zawierają więcej szczegółów. Ich popularność rośnie z roku na rok, miedzy innymi dzięki ich możliwościom adaptacji do wielu problemów, dostępności dużej ilości wytrenowanych ówcześnie modeli, które są gotowe do użycia oraz coraz większej ilości API (eng. Application Programming Interface). API umożliwiają tworzenie własnych modeli w oparciu o gotowe dokumentacje, przystępne nawet dla osób nieskorelowanych ze sztuczną inteligencją.

W opisywanej pracy magisterskiej została użyta zmodyfikowana architektura głębokiej sieci neuronowej typu UNET[1]. Składa się ona z trzech głównych bloków, jakimi są kolejno blok zmniejszający rozdzielczość (eng. Downsample Block/Encoder), który jest odpowiedzialny za ekstrakcję cech oraz zmniejszanie rozdzielczości, tudzież kompresję obrazu pozwalająca na wydobywanie szczegółów. Następnie występuje wąskie gardło (eng. Bottleneck), które odpowiada za ostateczną ekstrakcję cech, a ostatnim blokiem jest blok zwiększający rozdzielczość (eng. Upsample Block/Decoder), co najmniej do rozdzielczości wejściowej obrazu. Cześć pracy, związana bezpośrednio z siecią neuronowa została stworzona z użyciem biblioteki PyTorch[2].

Kolejnym ważnym aspektem w pracy jest proces generowania danych. Do tego celu został wykorzystany silnik graficzny o nazwie Unreal Engine[3] w wersji 4.26.2, który został poddany modyfikacjom umożliwiającym zapisywanie na dysk pojedynczych klatek generowanych podczas używania sekwencji w danym projekcie.

Unreal Engine jest to jeden z najbardziej, jak nie najbardziej popularny silnik graficzny służący do generowania wysokiej jakości danych oraz tworzenia gier komputerowych.

Opisywana praca magisterska została podzielona na trzy podstawowe pod problemy: zbiór danych wraz z silnikiem graficznym umożliwiający ich generowanie, sieć neuronowa wraz z całokształtem procesu uczenia i wyniki badań. Wszystkie procesy zostały opisane ze szczegółami w następnych rozdziałach.

# Zbiór danych

## Proces generowania danych

Proces generowania danych jest ścisłe związany z wspomnianym w rozdziale 1 silnikiem o nazwie Unreal Engine w wersji 4.26.2. Silnik ten posiada bardzo rozbudowany proces generowania obrazów, który z wersji na wersje zaczyna być coraz trudniej rozróżnialny, kiedy to porównamy obrazy wygenerowane przez silnik, a rzeczywiste zdjęcia. Na przykładzie [Rysunek 2.1] możemy wyróżnić sześć podstawowych potoków graficznych (eng. Graphic pipelines/passes), które maja miejsce podczas generowania pojedynczej klatki (wykluczając dwa ostatnie potoki tj. eng. User Interface oraz eng. Backbuffer Resolution):

1. **Geometria widoku (eng. View Geometry)** jest to proces w potoku graficznych, który służy do przygotowania macierzy projekcji, translacji itp. Dla aktualnej ramki/klatki oraz:
   1. **Tworzenia geometrii/bufora głębokości (eng. Depth Buffer)** odpowiedzialnego za informacje o odległości położenia obiektu od kamery w wartości składowych X,Y oraz Z sceny trójwymiarowej.
   2. **Tworzenia geometrii/bufora prędkości/ruchu (eng. Velocity Buffer)** odpowiedzialnego za informacje o poruszaniu się obiektu w wartościach X,Y oraz Z sceny trójwymiarowej.
2. **Głębokość pola widoku (eng. Depth of Field)** jest odpowiedzialny za efekt rozmycia obiektów na scenie bazując na wartościach bufora głębokości (eng. Depth Buffer). Obiekty, na których obecnie gracz nie skupia swojej uwagi, staja się rozmyte, jak pokazano na [Rysunek 2.2].
3. **Metoda anty-aliasingowa (eng. Anti-Aliasing Method)** jest odpowiedzialna za wygładzanie krawędzi, jak pokazano na [Rysunek 2.3] przed wygładzeniem krawędzi oraz po ich wygładzeniu [Rysunek 2.4]. Najbardziej popularne metody Anty-Aliasingowe to FXAA (eng. Fast Aproximate Anti-Aliasing)[4], MSAA (eng. Multi-Sampling Anti-Aliasing), TAA[5] (eng. Temporal Anti-Aliasing) oraz SSAA (Super-Sampling Anti-Aliasing).
4. **Metoda rozmywania ruchu kamery (eng. Motion Blur)** polega na dodaniu do sceny efektu rozmycia podczas ruchu kamery [Rysunek 2.5].
5. **Efekt rozmycia światła (eng. Bloom)** jest stosowany do uzyskania efektu rozmycia/poświaty ze źródła światła [Rysunek 2.6].
6. **Mapowanie tonów (eng. Tonemapping/Tonemapper)** służy do mapowania wysokiego zakresu kolorow (eng. HDR – High Dynamic Range) do niskiego zakresu kolorów (eng. LDR – Low Dynamic Range).

Table, timeline

Description automatically generatedRysunek . Przykładowy wygląd procesów (potoków) graficznych potrzebnych do wygenerowania pojedynczej klatki w silniku **Unreal Engine**.

A picture containing indoor, engine

Description automatically generatedRysunek . Przykładowy wygląd sceny po aplikacji procesu głębokości pola widoku/sceny (eng. **Depth of Field**), aktualny obiekt (A), na którym skupia się gracz pozostaje w wysokiej ostrości, reszta obiektów (np. B), staje się rozmyta.

Chart, line chart

Description automatically generated

Rysunek . Wygląd krawędzi przed ich wygładzeniem metodą **Anti-Aliasingowa**.

Chart, line chart

Description automatically generated

Rysunek . Wygląd krawędzi po ich wygładzeniu metodą **Anti-Aliasingowa**.

A picture containing indoor, music

Description automatically generated

Rysunek . Ramka po aplikacji efektu rozmycia zależnego od ruchu kamery.





Rysunek . **Obraz A** przedstawia scenę przed efektem rozmycia światła, a **Obraz B** tuż po jego zastosowaniu.

## Przygotowanie silnika do generowania danych

Pierwszym krokiem w procesie przygotowania (modyfikacji) silnika Unreal Engine był wybór właściwego miejsca do pobierania wygenerowanych klatek. W celu uzyskania zbioru danych o jak najwyższej możliwej jakości użytego do trenowania sieci neuronowej w opisywanej pracy, dane zostały pobrane z wejścia oraz wyjścia Temporalnego Anty-Aliasingu (eng. Temporal Anti-Aliasing)[Rysunek 2.7].

Następnym krokiem było dopisanie instrukcji w języku C++, które umożliwiły pobieranie danych z określonego miejsca i zapisywanie ich na dysk podczas pracy silnika.

Diagram

Description automatically generated

Rysunek . **Scene Color TAA Input** są to dane wejściowe do sieci neuronowej, a **Scene Color TAA Output** jest celem/referencja w procesie uczenia sieci.

## Przygotowanie projektu do generowania danych

W celu poprawnego pobierania danych z silnika, ważne jest przygotowanie projektu z którego będą pobierane dane (klatki). Odbywa się to za pomocą pobrania projektu ze sklepu Epic (firma, która stworzyła Unreal Engine) oraz dodaniu do Blueprint projektu (język wysoko poziomy stosowany w Unreal Editor) kolejno kilku elementów:

1. **Event BeginPlay** służy do uruchomienia następujących po nim elementów zaraz po rozpoczęciu sceny.
2. **Execute Console Command** służy do wykonania danej komendy. Komenda “**r.BufferVisualizationDumpFramesAsHDR 1”** określa, że scena będzie generowana w szerokim zakresie jasności tj. HDR (eng. High Dynamic Range)**.**
3. **Delay** jest odpowiedzialny za pauzę sekwencji w celu wyeliminowania np. niepoprawnie załadowanych tekstur.
4. **Play** uruchamia sekwencję (ówcześnie nagrany film, animacje itp.).
5. **Execute Console Command** posiadający komendę **“r.DumpPass.Allow 1”**, która rozpoczyna proces pobierania danych z silnika i zapisywania ich na dysk.

A screenshot of a video game

Description automatically generated

Rysunek . Przykładowy wygląd poprawnej sekwencji elementów w Blueprints.

## Uruchomienie procesu generowania danych

Proces generowania danych rozpoczyna się w momencie wywołania pliku wsadowego w formacie .bat oraz ścieżki do projektu z którego chcemy pobierać dane.

**> plik\_wsadowy.bat NazwaProjektu1/NazwaProjektu2.uproject**

Przykładowe wywołanie procesu generacji danych zostało przedstawione na [Rysunek 2.9].



Rysunek . Przykładowe wywołanie procesu generowania danych w wierszu poleceń.

## Wygenerowane dane

Wygenerowane dane są zapisane na dysku w formacie trzech składowych koloru, którymi są kolejno czerwony, zielony oraz niebieski (RGB – Red, Green, Blue) z użyciem biblioteki OpenEXR [6]. Biblioteka OpenEXR obsługuje dane generowane przez silnik Unreal Engine w formacie HDR, zapisując je z kompresja lub bez z użyciem typu danych float (zmiennoprzecinkowe, 32 bit), bądź half (zmiennoprzecinkowe, 16 bit) na każdą składowa (kanał) koloru. Dane użyte w opisywanej pracy magisterskiej zostały zapisane na dysk z użyciem kompresji (ZIP Compression) oraz typu danych half (16 bit). Dane wejściowe do sieci są w rozmiarze 1920x1080 pikseli (eng. Low Resolution Color, LR), a dane referencyjne są w rozmiarze 3840x2160 pikseli (eng. High Resolution Color, HR).

# Siec neuronowa

## Wczytywanie danych uczących

Wczytywanie danych uczących, które będą wykorzystywane przez sieci neuronowa odbywa się w dwóch etapach:

1. **Klasa Dataset** jest odpowiedzialna za:
   1. **Wczytanie danych uczących** do programu z użyciem biblioteki OpenEXR[6] (wspominanej w rozdziale 2.5). Po wczytaniu, dane są zamieniane z typu danych half na typ danych float i zapisywane w tensorach biblioteki PyTorch[2] w celu zwiększenia precyzji późniejszych obliczeń.
   2. **Wycięcie odpowiedniego regionu ze zdjęcia (eng. Cropping).** W opisywanej pracy magisterskiej zostały użyte wycinki (eng. Crops) ze środka zdjęć pełnowymiarowych o rozmiarze kolejno 128x128 pikseli dla wycinka wejścia do sieci (eng. Low Resolution Color, LR) oraz 256x256 pikseli dla wycinka referencyjnego (High Resolution Color, HR). Motywacją użycia wycinków (eng. Crops) jest zmniejszenie użycia pamięci karty graficznej oraz umiejętność głębokich sieci neuronowych (eng. Deep Neural Networks, Deep Convolutions) do uczenia się wzorców (eng. Patterns) na mniejszych obszarze i uogólnienie ich na wzorce na większym obszarze. Głębokie sieci neuronowe również są niezależne od rozmiaru danych wejściowych z wyjątkiem liczby kanałów (eng. Channels), zostało to opisane szczegółowo w części pracy poświęconej architekturze sieci neuronowej.
   3. **Wczytywanie wycinków (eng. Cached Crops Dataset)** danych bezpośrednio do pamięci RAM (eng. Random Access Memory) komputera. Umożliwia to przyspieszenie treningu o 800%, ponieważ podczas treningu, dane są ładowane bezpośrednio z pamięci RAM, a nie z dysku twardego (eng. HDD, Hard Disk Drive). Skutkiem tego jest zmniejszenie do minimum prawdopodobieństwa wystąpienia stanu bezczynności karty graficznej w oczekiwaniu na dane (eng. Idle state). Proces ten został zwizualizowany na [Rysunek 3.1].
2. **Klasa Dataloader** jest odpowiedzialna za:
   1. **Zbior danych (eng. Dataset)**, z którego będą pobierane dane podczas treningu (został on opisany w poprzednim punkcie).
   2. **Rozmiar wsadu danych (eng. Batch Size)** mówi nam o ilości próbek danych, które będą przetwarzane jednocześnie przez siec neuronowa, tudzież ilość próbek w jednej iteracji procesu uczenia.
   3. **Tasowanie danych (eng. Suffle)** jest parametrem, który odpowiada za zmianę kolejności próbek w zbiorze danych, co przekłada się na lepsze wyniki trenowanej sieci.
   4. **Zignoruj ostatni wsad danych (eng. Drop Last)** służy doodrzucenia ostatniego wsadu danych, jeżeli ilość próbek w zbiorze danych jest niepodzielna przez wcześniej określony rozmiar wsadu danych (eng. N\_Samples % Batch\_Size != 0), ponieważ powoduje to niekompletność ostatniego wsadu danych w porównaniu do jego poprzedników.
   5. **Przypięta pamięć (eng. Pin Memory)** jest odpowiedzialna za kopiowanie danych, bezpośrednio do pamięci zarezerwowanej dla komunikacji systemu z karta graficzną [Rysunek 3.1].

A diagram of data transfer and pinned data

Description automatically generated with medium confidence

Rysunek . Rysunek A przedstawia domyślny proces ładowania danych podczas treningu sieci neuronowej, a rysunek B przedstawia ten sam proces, lecz po zastosowaniu ładowania danych do pamięci RAM przed rozpoczęciem treningu. Można zauważyć, ze powoduje to wyłączenie z procesu dysku twardego, którego prędkość przesyłu danych jest bardzo mała. W połączeniu z parametrem **Pin Memory** uzyskujemy znacznie lepsza prędkość treningu sieci neuronowej.

## Normalizacja danych uczących

Normalizacja danych jest jednym z najbardziej kluczowych etapów w procesie uczenia sieci neuronowych. Ilość kroków potrzebnych do osiągniecia lokalnego, czy tez globalnego minimum jest znacznie większa, kiedy to nasze dane pozbawimy normalizacji przed rozpoczęciem treningu [Rysunek 3.2]. Dzieję się tak za sprawa gradientów, które zapewniają dostrajanie (uczenie) parametrów sieci (wag). Dane uczące użyte w opisywanej pracy magisterskiej są przechowywane w szerokim zakresie jasności (HDR), co może powodować bardzo duże rozbieżności i szum w procesie uczenia, kiedy to siec neuronowa otrzyma obraz o znacznym naświetleniu, a wiec obraz, który posiada wartości dążące w kierunku nieskończoności.

A picture containing text, screenshot, diagram, circle

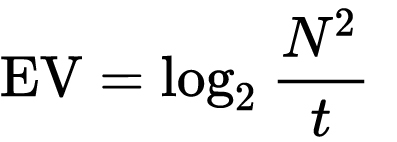
Description automatically generated

Rysunek . **Obraz A** Przedstawia ilość kroków potrzebnych do osiagnięcia lokalnego minimum przed normalizacja, a **Obrak B** po normalizacji danych.

## Proces normalizacji danych

W pracy magisterskiej proces normalizacji został rozszerzony. Dane przed przekazaniem ich do sieci neuronowej są odpowiednio preprocesowane (eng. PreProcessing). Polega to na wydobyciu jak największej ilości wartościowych informacji z danych, które pomogą w osiągnieciu lepszych wyników podczas treningu sieci neuronowej. Możemy wyróżnić trzy etapy preprocesowania danych:

1. **Dostrajanie ekspozycji (eng. Exposure Value, EV)** jest to dostosowanie ilości światła na scenie (ramce, zdjęciu) poprzez wymnożenie każdej składowej koloru (RGB) poprzez odwrócona wartość ekspozycji (RGB = RGB \* 1.0/EV). Wzór na obliczenie wartości ekspozycji znajduje się na [Rysunek 3.3], a przykład jak dostrajanie ekspozycji wpływa na jasność sceny (zdjęcia) został przedstawiony na [Rysunek 3.4].
2. **Mapowanie Tonów (eng. Tonemapping)** polega na konwersje szerokiego zakresu jasności (HDR) na wąski zakres jasności (LDR), który w większości przypadków zawiera się w przedziale [0-1] dla wartości zmiennoprzecinkowych (float) oraz [0-255] dla wartości całkowitoliczbowych (int). Motywacją tej operacji jest kompatybilność z zakresem w jakim pracują monitory komputerowe (wykluczając monitory obsługujące HDR). W opisywanej pracy, mapowanie tonów zostało użyte jako normalizacja danych na zakres [0-1] dla typu zmiennoprzecinkowego. Przykładowy wygląd sceny przed oraz po zastosowaniu mapowania tonów został przedstawiony na [Rysunek 3.5].
3. **Odwrotna korekcja Gamma (eng. Inverse Gamma Correction)** służy do zmniejszenia różnicy miedzy niskimi a wysokimi tonami kolorów. Polega to na podniesieniu krzywej luminancji koloru sceny [Rysunek 3.6] poprzez podniesienie każdej składowej koloru (RGB) do potęgi współczynnika gamma (w opisywanej pracy magisterskiej współczynnik gamma wynosi 1.0/2.2 = 0.(45)). Odwrotną Korekcję Gamma stosuje się w celu aplikacji nieliniowości, ponieważ ludzkie oko lepiej rozróżnia kolory nieliniowe. Przykład efektu odwrotnej korekcji gamma przedstawia [Rysunek 3.6].



Rysunek . Wzór na obliczenie Wartości Ekspozycji (Exposure Value, EV)

Gdzie:

N – miara zdolności zbierania światła np. przez obiektyw aparatu, kamery.

T – określa czas naświetlania sceny w sekundach.

A picture containing panorama, night, light, city

Description automatically generated

Rysunek . **Obraz A** przedstawia wartość ekspozycji 1.0, **Obraz B** przedstawia wartość ekspozycji 2.0, a **Obraz C** przedstawia wartość ekspozycji 6.0.

A picture containing screenshot, night, city

Description automatically generated

Rysunek . **Obraz A** przedstawia scenę przed operacja mapowania tonów, a **Obraz B** po zastosowaniu operacji mapowania tonów.



Rysunek . Wykres przedstawiający krzywa luminancji koloru. Wykres górny przedstawia odwrotna korekcje gamma, a wykres dolny korekcje gamma.

A picture containing screenshot, digital compositing, panorama, night

Description automatically generated

Rysunek . **Obraz A** przedstawia scenę przed odwrotna korekcja gamma, a **Obraz B** przestawia scenę po odwrotnej korekcji gamma.

# Podsumowanie i bibliografia

## Podsumowanie

ADDDASD

## Bibliografia

[1] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015. doi: 10.1007/978-3-319-24574-4\_28.

[2] Facebook AI Research, “PyTorch,” *https://pytorch.org/*, 2016. https://pytorch.org/ (accessed May 05, 2023).

[3] Epic Games, “Unreal Engine,” *https://www.unrealengine.com/en-US/*, 1998. https://www.unrealengine.com/en-US/ (accessed May 05, 2023).

[4] Timothy Lotes, “FXAA,” *https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA\_WhitePaper.pdf*, 2009.

[5] L. Yang, S. Liu, and M. Salvi, “A Survey of Temporal Antialiasing Techniques,” *Computer Graphics Forum*, vol. 39, no. 2, 2020, doi: 10.1111/cgf.14018.

[6] Academy Software Foundation, “OpenEXR,” *https://openexr.com/en/latest/*, 2003. https://openexr.com/en/latest/ (accessed May 05, 2023).