PRACA MAGISTERSKA

Dobór parametrów do modelu fali Gerstnera przy użyciu algorytmu sztucznej inteligencji

Paweł Janusz
283639

Kierunek: informatyka
Specjalność: Uczenie Maszynowe

PROMOTOR
dr hab. inż. Edyta Hetmaniok, prof. PŚ
WYDZIAŁ MATEMATYKI STOSOWANEJ

GLIWICE 2023

Tytuł pracy: Dobór parametrów do modelu fali Gerstnera przy użyciu algorytmu sztucznej inteligencji

Streszczenie:

Tu wpisz streszczenie po polsku

Słowa kluczowe:

Słowa kluczowe

Thesis title:

Tytuł po angielsku

Abstract:

Tu wpisz streszczenie po angielsku

Keywords:

Słowa kluczowe po angielsku

Spis treści

1. Wstęp

1.1. Wprowadzenie do problematyki modelowania fal w grafice komputerowej

W grafice komputerowej wykorzystuje się fale Gerstnera do tzw. renderowania, czyli generowania realistycznie wyglądających fal na podstawie kodów komputerowych. Jest to jednak rozwiązanie bardzo kosztowne w obliczeniach. Wynika to z tego, że dokładna symulacja płynu wymaga obliczania osobno każdej z cząstek tego płynu. Dlatego szuka się jak najlepszych metod na zasymulowanie cieczy, bez potrzeby obliczania wszystkiego. Tutaj z pomocą przychodzą różne metody w tym metoda oparta na fali Gerstnera.[**?**]

1.2. Omówienie istotności efektywnej optymalizacji w grafice komputerowej

Tu będzie treść rozdziału

1.3. Analiza problemu w oparciu o istniejące prace

Tu będzie treść rozdziału

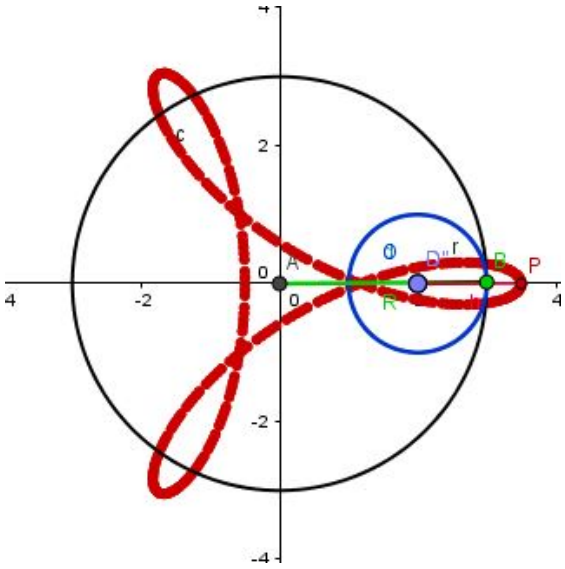
2. Czym jest fala Gerstnera i gdzie znajduje zastosowanie?

2.1. Definicja i charakterystyka fali Gerstnera

Fala Gerstnera to termin używany w dziedzinie dynamiki płynów. Inną jej nazwą jest fala trochoidalna. Opisuje ona ruch cząsteczek cieczy na powierzchni. Fala trochoidalna została odkryta przez Gerstnera w 1802 roku i ponownie odkryta niezależnie przez Rankine'a w 1863 roku[?]

2.1.1. Trochoida

Krzywa jest to krzywa zakreślona przez dowolnie obrany punkt P stale związany z kołem O toczącym się wzdłuż wewnętrznej lub zewnętrznej strony stałego (nie poruszającego się) okręgu bez poślizgu.[?]



Rysunek 1: Trochoida (https://home.agh.edu.pl/ zobmat/2017/3_kozlowskijakub/)

2.2. Typowe zastosowania fali Gerstnera

Tu będzie treść rozdziału

3. Problem analizowany w pracy

Tu będzie treść rozdziału

4. Użyte narzędzia i techniki

4.1. Projekt DeepWave i inne narzędzia diagnostyczne

Tu będzie treść rozdziału

4.2. FFT i jego związek z falą Gerstnera

4.2.1. Czy fala Gerstnera to FFT?

Tu będzie treść rozdziału

4.2.2. Czy da się siecią neuronową optymalizować FFT

Tu będzie treść rozdziału

4.3. Unity

Unity jest jednym z najbardziej popularnych, wieloplatformowych silników do tworzenia gier i symulacji. Charakteryzuje się on edytorem opartym na komponentach, który umożliwia programistom i projektantom gier wizualne konstruowanie scen, postaci oraz mechanik bez konieczności pisania obszernego kodu z poziomu zero. Językiem skryptowym używanym w Unity jest C#, który oferuje zarówno elastyczność, jak i moc obliczeniową potrzebną do tworzenia złożonych systemów.

Unity oferuje zaawansowane funkcje graficzne, takie jak wsparcie dla najnowszych technik renderowania, shaderów oraz systemów cząsteczkowych, co pozwala tworzyć wizualnie imponujące efekty w czasie rzeczywistym. Dodatkowo, silnik posiada wbudowany system fizyki, który pozwala na tworzenie realistycznych interakcji i dynamiki obiektów w środowisku wirtualnym.

Aspektem, który wyróżnia Unity, jest jego uniwersalność - wspiera on szeroką gamę platform, od komputerów PC i konsol, poprzez urządzenia mobilne, aż po platformy VR i AR. Pozwala to deweloperom na łatwe portowanie i dostosowywanie gier między różnymi systemami. Unity jest również integracyjne z wieloma narzędziami zewnętrznymi i usługami, takimi jak systemy kontroli wersji, narzędzia do tworzenia assetów oraz usługi backendowe, co czyni go niezwykle elastycznym w procesie produkcji gier i aplikacji interaktywnych.

4.4. ML-Agents (Framework)

Unity ML-Agents Toolkit jest zaawansowanym frameworkiem stworzonym przez Unity Technologies, który umożliwia tworzenie i trenowanie inteligentnych agentów z wykorzystaniem technik uczenia maszynowego w złożonych środowiskach trójwymiarowych. Jest to narzędzie szczególnie cenione w dziedzinie rozwoju gier i symulacji, które pozwala na zastosowanie algorytmów uczenia głębokiego, uczenia ze wzmocnieniem oraz innych metod sztucznej inteligencji.

5. Wyniki i ich analiza

5.1. Analiza problemu przy użyciu sieci neuronowej

Tu będzie treść rozdziału

6. Finalny projekt i zastosowanie AI

W projekcie, będącym przedmiotem niniejszej pracy dyplomowje, zostanie wykorzystana sztuczna inteligencja i jej zdolność do zrozumienia niuansów działania fali Gerstnera, aby utrzymać optymalną jakość działania aplikacji. Słowo optymalna oznacza dążenie do wybranego najkorzystniejszego celu, zamierzam przetestować kilka scenariuszy działania, takich jak utrzymywanie stałej liczby klatek na sekundę, odpowiednie zużycie zasobów komputera, utrzymanie się w podanych limitach, aby osiągnąć ten cel. Przydatność takiego rozwiązania można by rozwinąć i zastosować we współczesnych grach, gdzie sztuczna inteligencja podejmowałaby decyzje o redukcji pewnych zasobożernych obliczeń, by utrzymać płynny rezultat.

6.1. Konfiguracja Parametrów Treningu

Konfiguracja treningu agentów w ML-Agents jest określana przez plik konfiguracyjny w formacie YAML, który pozwala na precyzyjne ustawienie parametrów mających wpływ na proces uczenia. Tymi parametrami są:

- **batch_size**: Określa ilość doświadczeń wykorzystywanych w pojedynczym kroku aktualizacji, wpływając na stabilność treningu.
- **buffer_size**: Definiuje wielkość bufora do przechowywania doświadczeń, co jest istotne dla adaptacji agenta do zmian w środowisku.
- **learning_rate**: Szybkość uczenia, która ma kluczowe znaczenie dla tempa adaptacji sieci neuronowej.
- **beta**: Waga entropii w funkcji strat, równoważąca eksplorację i eksploatację zdobytej wiedzy przez agenta.
- **epsilon**: Parametr kluczowy dla ograniczenia aktualizacji polityki, zapobiegający zbyt gwałtownym zmianom.
- **lambd**: Faktor wygładzania w algorytmie PPO, wpływający na jakość estymacji przewagi.
- **num_epoch**: Liczba epok na przetworzenie zestawu danych, co ma wpływ na dopasowanie modelu do doświadczeń.

6.2. Ustawienia Sieci

Parametrami ustawień sieci są:

- **normalize**: Normalizacja wejść, która może zwiększyć stabilność i generalizację sieci neuronowej.
- **hidden_units**: Ilość jednostek w warstwach ukrytych, decydująca o pojemności modelu do nauki złożonych wzorców.
- **num_layers**: Liczba warstw ukrytych w sieci, która powinna być dostosowana do złożoności zadania.
- **vis_encode_type**: Typ kodowania danych wizualnych, który musi odpowiadać złożoności danych wejściowych.

6.3. Sygnały Nagrody

Nagrody regulują następujące parametry:

- **gamma**: Wartość dyskontowa, określająca znaczenie przyszłych nagród w procesie uczenia.
- **strength**: Siła sygnału nagrody, istotna dla oceny wpływu nagród zewnętrznych.

Dalsze parametry, takie jak **time_horizon**, **summary_freq** i **threaded**,oznaczające czas uczenia, co jaki czas zapisywać statystyki treningu oraz czy używać wielu wątków, są również istotne i mogą być dostosowane do potrzeb treningu wpływając na sposób, w jaki agent ocenia stan i jak efektywnie trening jest przeprowadzany.

6.4. Realizacja V1

Pierwsze podejście do zrealizowania projektu polegało na stworzeniu prostego środowiska, w którym uda się skonfigurować sieć neuronową umożliwiającą dopasowanie liczby do jej wyznaczonego celu. Akcje, które może podejmować sieć neuronowa, to dodawanie lub odejmowanie 1. Po kilku próbach udało się wypracować dobry system przyznawania nagrody, który szybko przynosił efekty. Wcześniejsze próby z przyznawaniem kary sprawiały, że model szybko przestawał się uczyć. *Potencjał na rozbudowę, opisać wcześniejsze próby, czemu się nie uczyło, brak normalizacji.*

Dzięki temu wypracowany został model, który na podstawie aktualnej wartości liczby, potrafił ją odpowiednio zmniejszać bądź zwiększać, trzymając się celu. Na ten moment nie było tutaj większych utrudnień.

6.4.1. Konfiguracja treningu

```yaml
behaviors:
    Wave:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.001
      beta: 0.001
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: constant
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    checkpoint_interval: 500000
    max_steps: 5000000
    time_horizon: 64
    summary_freq: 10000
    threaded: true
```

6.5. Realizacja V2

W tym podejściu algorytm operował na prawdziwych parametrach i kontrolował jakość fali Gerstnera, przez zmniejszanie lub zwiększanie jej jakości. Jakość shadera najpierw zdefiniowana została odgórnie, przez kilka różnych poziomów szczegółowości. Algorytm musiał dobrać odpowiednią falę do odpowiedniego scenariusza. Agent działa w dynamicznym środowisku, gdzie statystyki GPU mogą się wahać, co dodaje złożoności procesowi uczenia. Początkowo agent niezależnie kontrolował każdą falę (99 łącznie), prowadząc do dużej dyskretnej przestrzeni akcji (99 dyskretnych akcji, każda z 2 stanami). Ta złożoność stwarzała wyzwania w efektywności i jasności podejmowania decyzji. Projekt był bardzo niewydajny, wolno się uczył. Aby uprościć proces, przejście na parametryzowane podejście z użyciem ciągłych akcji (continuous action space). To znacznie uprościło przestrzeń akcji. Agent obserwował statystyki GPU w czasie rzeczywistym: temperaturę, wykorzystanie i dostępną pamięć. Te obserwacje były kluczowe, aby agent mógł zrozumieć obecny stan GPU i podejmować świadome decyzje. Aby zapewnić kontekst czasowy, dodano historyczne dane GPU, umożliwiając agentowi dostrzeżenie trendów i wpływu jego działań na przestrzeni czasu. Funkcja nagrody przeszła kilka iteracji, aby dostosować cele uczenia agenta do celów projektu. Wstępne wersje karały za wysokie wykorzystanie GPU i nagradzały za liczbę włączonych fal, dążąc do równowagi między wydajnością a złożonością wizualną. Jednakże to prowadziło do unikania przez agenta wyłączania fal, wpływając na wydajność. Udoskonalona strategia skupia się na nagradzaniu za poprawę wydajności GPU, zachęcając agenta do optymalizacji wykorzystania GPU, jednocześnie utrzymując akceptowalny poziom aktywności fal. Nagradza za niższe średnie wykorzystanie GPU przez określoną liczbę kroków, motywując do długoterminowej optymalizacji, a nie krótkoterminowych korzyści. Kolejnym wyzwaniem okazało się losowe wahania w wykorzystaniu GPU, które były przez agenta błędnie interpretowane jako konsekwencje jego działań. Rozwiązaniem była implementacja wygładzania danych i rozszerzenie kontekstu historycznego w obserwacjach, pozwalając agentowi lepiej rozumieć trendy i zmniejszać wpływ losowych fluktuacji. Podejście to jednak nie dawało owocnych rozwiązań, stworzenie środowiska gdzie agent może odczytywać delikatne różnice w wykorzystaniu karty graficznej, było bardzo wymagającym zadaniem. Zająłem się poszukiwaniem innego rozwiązania.

6.6. Realizacja V3 Nauczanie agentów śledzenia fali sinusoidalnej

W tym podejściu zbudowałem projekt mający na celu nauczenie sztucznych agentów działających w środowisku Unity śledzenia ruchu fali sinusoidalnej.

6.6.1. Napotkane problemy

W trakcie realizacji projektu napotkano szereg wyzwań, w tym:

- Trudności w efektywnym nauczaniu agentów złożonych zachowań, takich jak precyzyjne śledzenie ruchu fali.
- Problem z odpowiednim dostrojeniem funkcji nagród, aby zachęcić agentów do aktywnego i celowego ruchu.
- Wyzywania związane z koordynacją wielu agentów i ich współdziałaniem w celu wspólnego śledzenia fali.

6.6.2. Podejmowane działania

W odpowiedzi na napotkane wyzwania, podjęto szereg działań, w tym:

- Uproszczenie problemu poprzez skoncentrowanie się na pojedynczym agencie, co pozwoliło na dokładniejsze zrozumienie dynamiki uczenia.
- Eksperymentowanie z różnymi strukturami nagród, w tym zastosowanie kar za brak ruchu oraz nagród za ruch w kierunku oczekiwanym.
- Modyfikacja przestrzeni akcji agenta, ograniczając ją do kontroli prędkości, co uprościło problem i pozwoliło na skupienie się na kluczowym aspekcie uczenia.

6.6.3. Wnioski końcowe

Mimo podjętych działań, projekt nie osiągnął pełnego sukcesu w zakresie nauczania agentów precyzyjnego śledzenia fali sinusoidalnej. Ujawniło to ograniczenia zastosowanych metod oraz potrzebę dalszych badań nad optymalizacją algorytmów uczenia w kontekście dynamicznych środowisk. Projekt ten jednak przyczynił się do lepszego zrozumienia problemów związanych z uczeniem maszynowym w środowiskach symulacyjnych i wskazał kierunki dla przyszłych badań.

7. Możliwe rozszerzenia

7.1. Adaptacyjne dostosowywanie szczegółowości

Tu będzie treść rozdziału

8. Nowe podejście

Chciałbym teraz zaprezentować nowe podejście, które jest bardzo obiecujące. (To zostanie prawodpodobnie zamienione w większą część pracy, pozwolę sobie tutaj opisać nad czym ostatnio pracowałem). Inspiracją do tego podejścia są dwa projekty, które łączy jedno nowatorskie podejście do wykorzystywania sztucznej inteligencji. [?] W tych projektach, wykorzystano istniejący w Unity framework o nazwie **Sentis**. Jest to program służący do importowania oraz uruchamiania w czasie rzeczywistym modeli sztucznej inteligencji. Wspiera on takie formaty jak ONNX(Open Neural Network Exchange), co daje mu kompatybilność z każdym projektem opartym o PyTorch albo Tensorflow, ponieważ te platformy oferują eksport do tegoż formatu. Dzięki temu możemy nasz model który rozwijaliśmy w pythonie w notebooku Jupyter, użyć w silniku Unity, który zajmie się optymalizacją i odpowiednim działaniem. Wspomniany nowatorski projekt wykorzystał sieć neuronową służącą do generowania grafiki, aby tworzyła tekstury imitujące swoją szczegółowością, poziom dostępny tylko dla technologii RayTracing. Dzięki temu na urządzeniach o przeciętnej wydajności, takich jak kilkuletnie telefony, udało się uzyskać efekt grafiki jaki jest możliwy na bardzo wydajnych jednostkach. Postanowiłem w tym podejściu oprzeć się o tę metodę.

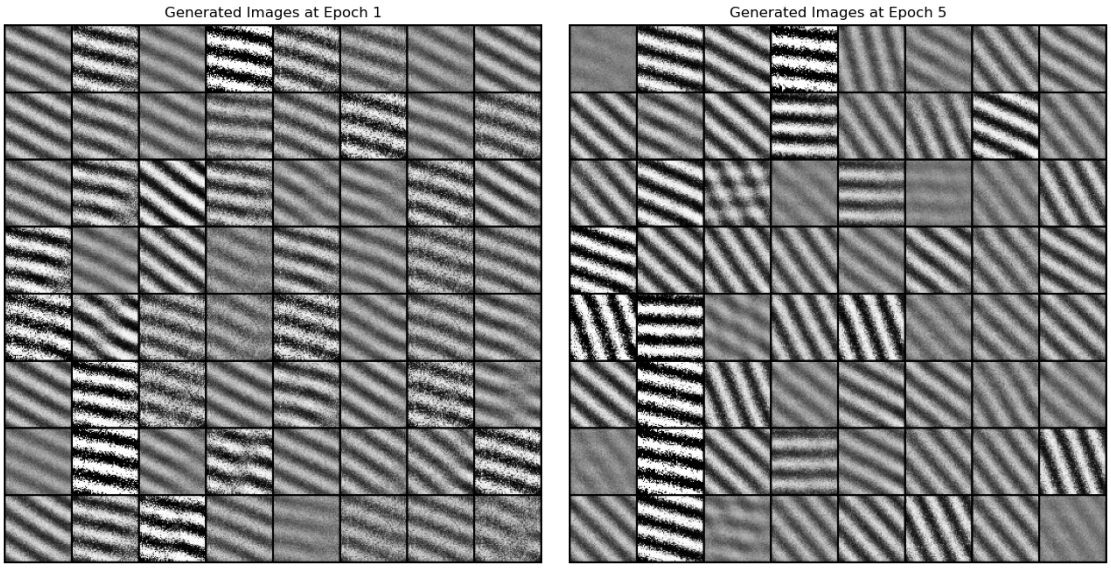
8.1. Przygotowanie

Aby wykorzystać potencjał drzemiący w sieciach zajmujących się generowaniem grafiki, należy przygotować sporą bazę różnorodnych przykładów na których sieć może się uczyć. Wykonałem do tego w Unity prosty skrypt który oblicza fale gerstnera, następnie generuje z niej teksturę widoczną z góry. Dla jednej fali, tekstura nie wygląda zbyt spektakularnie. Jest to jednak wystarczające aby przetestować możliwości sieci. Wygenerowanej tekstury z góry przygotowałem około 50 tys, generując ją pod różnymi kierunkami.

8.2. Sieć cGAN

Typ sieci wykorzystywany do generowania grafiki, wykorzystany w tym przykładzie to sieć typu GAN (Generative Adversarial Network), co można przetłumaczyć

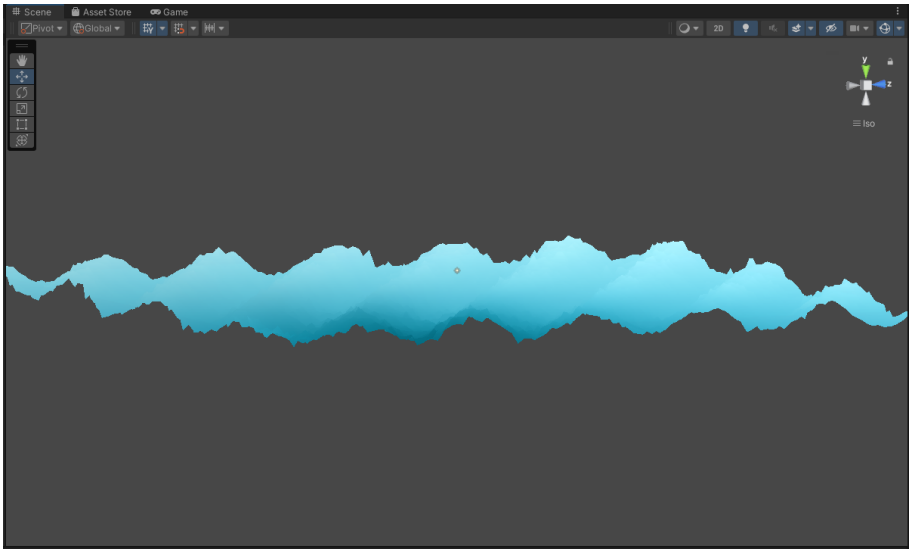
jako Generatywna Sieć Przeciwna. Wykorzystuje ona dwa rozdaje sieci, Generatora oraz Dyskryminatora, Generator na podstawie szumu, tworzy obraz, którym próbuje oszukać sieć Dyskryminującą ten obraz. Generator staje się coraz lepszy w oszukiwaniu Dyskryminatora, co przekłada się na lepszy wynik końcowy. Sieci które do tego pozwalają dodać zmienną modyfikującą ten obraz, nazywamy cGAN (Conditional GAN), dzięki temu możemy uzyskać obraz o odpowiedniej specyfice. W tym przypadku, obrazki opatruję danymi na temat ich kierunku i jest to jedyna zmienna, która wpływa na ten model.



Epoka 1 oraz 5 w pierwszym wstępnym generowaniu pojedynczej fali Gerstnera

8.3. Importowanie modelu

Zaimportowany model został podpięty i sprawdzony w działaniu. Podpięcie polegało na odpowiednim przemapowaniu wynikowego tensora na teksturę(Pomysł na rozbudowę, opisać jak bardzo było to trudne, bo było), następnie tekstura ta służyła jako tzw. Height-Map, czyli mapa wysokości, służąca do tego by za pomocą danych o kolorze w danym miejscu, odkształcić płaską siatkę. Dało to oczekiwany efekt w postaci fali.



Fala powstała z modelu sieci cGan

Jak widać fala jest bardzo zaburzona i nie jest perfekcyjna, jednak już w tej wersji reaguje na zmieniające się parametry i jest bardzo dobrym wstępem do dalszej eksploracji tego projektu. Na tym etapie na pewno nie jest to rozwiązanie bardziej wydajne niż jego tradycyjna wersja, jednak to dopiero wersja wstępna.

9. Podsumowanie

9.1. Refleksja nad osiągnięciami projektu

Tu będzie treść rozdziału

9.2. Potencjalne kierunki dalszych badań

Tu będzie treść rozdziału

10. Bibliografia

https://home.agh.edu.pl/ zobmat/2017/3$_k ozlowskijakub/MLAgents\sum_e dek$

Notatki

https://www.youtube.com/watch?v=2Bw5f4vYL98 https://raphaelstaebler.medium.com/how-i-trained-an-ai-to-play-my-mobile-game-a82bc37f7f7a

Analiza problemu w oparciu o istniejące prace

- Projekt **DeepWave**
- Nvidia Packet Wave Theory
- Nvidia NSight
- Nvdia ocean
- Analiza najprosteszego shadera wody wykorzystującego fale Gerstnera Czym jest fala gerstnera
- Czy jest to FFT?
- Czy da się siecią neuronową optymalizować FFT
- Czemu się nie da
- Czemu się da Typowe zastosowania fali gestnera
- Przykłady z gier, filmów, symulacji

Podjęte kroki i decyzje: Analiza problemu przy użyciu sieci neuronowej

- Python, neural network, nie powiodło się, prawdopodobnie zbyt skomplikowana sieć.

Finalny projekt: Analiza problemu w oparciu o wykorzystanie go w użytecznym celu Projekt który w oparciu o aktualne dane ze środowiska:

- Liczba klatek na sekundę
- Obciążenie CPU
- Obciążenie GPU
- Zużycie pamięci
- Limity poszczególnych wartości