Życiorys

Urodziłem się 2 lutego 1988 r. w Rawie Mazowieckiej. W roku 2004 ukończyłem gimnazjum im. Haliny Konopackiej w Rawie Mazowieckiej i rozpocząłem naukę w Liceum Ogólnokształcącym im. Marii Skłodowskiej Curie w klasie o profilu matematyczno-informatycznym. Po zdaniu matury w 2007 roku zostałem studentem Politechniki Warszawskiej na wydziale Elektrycznym na kierunku Informatyka. Aktualnie kształcę się w specjalizacji "Inżynieria Oprogramowania".

W roku 2010 zdobyłem I miejsce na Ógólnopolskiej Konferencji Inżynierii Gier Komputerowych (IGK)."W roku 2009 na tej samej konferencji udało mi się wywalczyć II miejsce na podium.

Moje zainteresowania to: gry komputerowe, nowe technologie, film oraz motocykle.

Streszczenie

Tytuł: „NVIDIA CUDA - Technologia przyśpieszająca metodę śledzenia promieni”

W niniejszej pracy inżynierskiej przedstawiony został problem i przeprowadzone zostały badania nad metodą śledzenia promieni przy wykorzystaniu technologii NVIDIA CUDA. Do pracy zaprojektowana i napisana została aplikacja testowa, badająca wsteczną metodę śledzenia promieni na procesorach CPU oraz kartach graficznych GPU. Głównymi celami niniejszej pracy inżynierskiej są:

- Przyśpieszenie wstecznej metody śledzenia promieni przy użyciu technologii NVIDIA CUDA.
- Projekt i implementacja uniwersalnej aplikacji testującej śledzenie promieni na różnych konfiguracjach sprzętowych.

W pracy tej staram się poznać tajniki technologii przetwarzania strumieniowego CUDA oraz jak najlepiej użyć jej do rozwiązania postawionego problemu śledzenia promieni. Zrealizowana praca badawcza opiera się na przeprowadzonych testach wydajności śledzenia promieni na różnych konfiguracjach sprzętowych. W pracy został opisany projekt, implementacja oraz przeprowadzone testy.

Abstract

Title: „NVIDIA CUDA - Technology for speeding up raytracing”

In this engineering thesis, problem and research about raytracing using NVIDIA CUDA technology is presented. For work was designed and written test application which studies backward raytracing based on computer processors CPU and also on GPU graphics cards. The main goals of this engineering thesis are:

- Speeding up backward raytracing using NVIDIA CUDA technology.
- Project and implementation of universal application for testing raytracing on different hardware configurations.

In this work, I try to learn the technology of processing streaming and how to use it to solve the problem of tracing rays in the best way. Realized research work is based on raytracing performance tests on different hardware configurations. Finally design, implementation and conducted tests were described.

Spis treści

Podziękowania

Chciałbym bardzo podziękować mojemu promotorowi za opiekę naukową oraz cenne rady podczas pisania tej pracy.

Dodatkowo bardzo dziękuję moim rodzicom oraz mojej dziewczynie. Bez ich wsparcia i bodźców motywacyjnych powstanie tej pracy nie było by możliwe.

Na końcu lecz nie mniej ważne podziekowania dla twórców mojego sprzętu, który dużo wycierpiał podczas powstawania niniejszej pracy. Karta graficzna naprawiana była trzy razy...

Rozdział 1

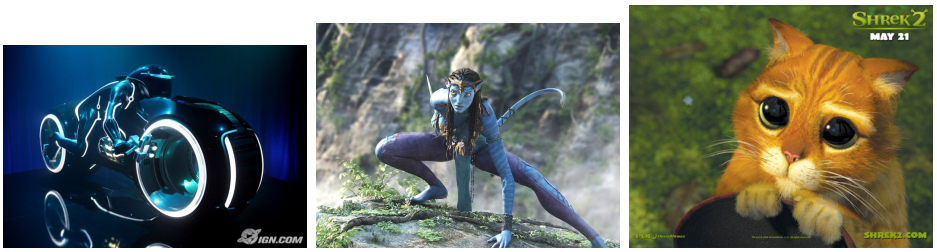
Wstęp

1.1 Wprowadzenie

Raytracing jest techniką służącą do generowania fotorealistycznych obrazów scen 3D. Na przestrzeni lat technika ta ciągle się rozwijała. Doczekała się wielu modyfikacji, które usprawniają proces generowania realistycznej grafiki. Takimi technikami mogą być między innymi PathTracing, PhotonMapping i wiele innych. Z dnia na dzień wykorzystywanie raytracingu ciągle rośnie. W dzisiejszych czasach w grafice komputerowej oraz w kinematografii do uzyskania realistycznych efektów metoda śledzenia promieni używana jest bardzo często. Dzięki takim zabiegom jesteśmy w stanie dosłownie zasymulować sceny oraz zjawiska, które nie muszą istnieć w rzeczywistym świecie. Czas generowania pojedynczej klatki/ujęcia takiej sceny niekiedy potrafi być liczony nawet w godzinach. Dlatego technika ta nie doczekała się jeszcze swojej wielkiej chwili w przemyśle rozrywkowym jakim są np. gry komputerowe oraz inne aplikacje generujące grafikę 3D w czasie rzeczywistym.

1.2 Motywacja

Głównym bodźcem motywacyjnym do napisania tej pracy była chęć poszerzenia dotychczasowej wiedzy na temat przyśpieszania obliczeń przy pomocy nowej technologii NVIDIA CUDA. Dodatkowymi czynnikami motywacyjnymi było zamiłowanie do grafiki komputerowej oraz do tworzenia aplikacji obliczeń czasu rzeczywistego. Na co dzień zajmuję się programowaniem gier/aplikacji na komputery klasy PC oraz urządzenia mobilne. Uważam, że w przyszłości przedstawiona przeze mnie w tej pracy metoda



a) Tron Legacy b) Avatar c) Shrek

Rysunek 1.1: Ujęcia filmowe stworzone za pomocą technik komputerowych a) "http://movies.ign.com/", b) "http://hdtvmania.pl/", c) "http://www.myfreewallpapers.net/"

śledzenia promieni będzie miała zastosowanie w grach oraz aplikacjach wykorzystujących grafikę czasu rzeczywistego.

1.3 Dostępne technologie, pozwalające zrównoleglić obliczenia na kartach graficznych

Spośród wielu metod przyśpieszających śledzenie promieni warto wyrónić trzy technologie wspomagające równoległe obliczenia na kartach graficznych. Niemniej jednak badania przeprowadzone i opisane w dalszej części pracy będą skupiały się na wykorzystaniu jednej z tych metod, a mianowicie technologii NVIDIA CUDA.

1.3.1 Open Computing Language (OpenCL)

Technologia ta zainicjowana została przez firmę Apple. Do inicjatywy i rozwijania tej technologii włączyły się w późniejszym czasie inne firmy takie jak: AMD, IBM, Intel, NVIDIA. W roku 2008 sformowana została grupa Khronos skupiająca powyższe firmy oraz wiele innych należących do branży IT. Grupa ta czuwa nad rozwojem technologii OpenCL. Technologia ta pozwala na pisanie kodu który jest przenośny między wieloma platformami: komputery, urządzenia przenośne, klastry obliczeniowe. OpenCL pozwala rozpraszać obliczenia na jednostki procesorowe CPU oraz na architektury graficzne GPU. Bardzo ważną zaletą OpenCL jest to, że pisanie z użyciem tej technologii nie jest zależne od sprzętu na jakim będzie ona uruchamiana. [11]

1.3.2 ATI Stream Computing

Technologia ta została stworzona przez firmę AMD. Za pomocą tej platformy jesteśmy w stanie przeprowadzać złożone obliczenia na sprzęcie produkowanym przez AMD. W skład całego pakietu ATI Stream Computing wchodzi autorski język ATI Brook+ i kompilator tegoż języka. Dodatkowo ATI wspiera developerów własną biblioteką matematyczną (AMD Core Math Library) oraz narzędziami do profilowania wydajności kodu (Strem Kernel Analyzer). Technologia ATI konkuruje od dawna z technologią NVIDIA CUDA. [8]

1.3.3 NVIDIA CUDA

CUDA (Compute Unified Device Architecture) jest technologią opracowaną przez firmę NVIDIA. Swoje początki CUDA miała w 2007 roku i do dziś jest wiodącą technologią strumieniowego przetwarzania danych z wykorzystaniem układów graficznych GPU. Dalszemu opisu niniejszej technologii poświęcony zostanie osobny rozdział. [9]

a) Logo NVIDIA CUDA

b) Logo ATI Stream Computing

Rysunek 1.2: Loga dwóch konkurencyjnych ze sobą technologii przetwarzania równoległego na kartach graficznych

1.4 Rekonesans

W sieci istnieje wiele rozwiązań metody śledzenia promieni. Istnieje także wiele modyfikacji standardowej metody wstecznego raytracingu takich jak „Path Tracing” oraz „Photon Mapping”. Rozwiązanie firmy NVIDIA przynosi wiele korzyści, wykorzystanie CUDA stało się ostanio wręcz modne. Udało mi się znaleźć jedno rozwiązanie podobne do mojego, które przedsprzedstawia jednoznacznie porównanie wydajności raytracingu

między procesorami CPU oraz kartami GPU firmy NVIDIA, korzystajacymi z technologii CUDA ("http://home.mindspring.com/~eric˙rollins/ray/cuda.html"). Wydaje mi się, że możliwości tej technologii nie zostały w pełni przeanalizowane. Niniejsza praca jest próbą kontynucji badań w tym zakresie. Ciekawym również zestawieniem wydajności raytracingu na wielu konfiguracjach procesorowych jest zestawienie sporządzone przez Johna Tsiombikas'a na stronie "http://www.futuretech.blinkenlights.nl/c-ray.html".

Rozdział 2

Cele pracy

2.1 Opracowanie techniki zrównoleglenia i przyspieszenia metody śledzenia promieni przy użyciu NVIDIA CUDA

Celem niniejszej pracy jest przeniesienie a zarazem zrównoleglenie algorytmu śledzenia promieni na procesory graficzne (GPU) firmy NVIDIA. Celem także jest przyśpieszenie obliczeń standardowego wstecznego Raytracingu w celu jak najszybszego generowania obrazów scen 3D.

2.2 Projekt uniwersalnej aplikacji - benchmark

W ramach projektu napisany został uniwersalny system Raytracingu działający na wielordzeniowych procesorach komputerowych (CPU), a także na kartach graficznych (GPU) firmy NVIDIA które obsługują technologie NVIDIA CUDA. Aplikacja testowa jest benchmarkiem, który jest w stanie przetestować zadane sceny 3D na wielu różnych konfiguracjach sprzętowych. Aplikacja ma za zadanie po uruchomieniu na komputerze użytkownika, testować wszelkie sceny z odpowiedniego katalogu. Dodatkowo zbierać potrzebne informacje o sprzęcie użytkownika oraz czasy generowania obrazów z każdej ze scen. Po przeprowadzeniu wszelkich testów aplikacja jest w stanie wysłać na adres e-mail developera (w tym przypadku autora pracy) wszelkie zgromadzone dane.

Rozdział 3

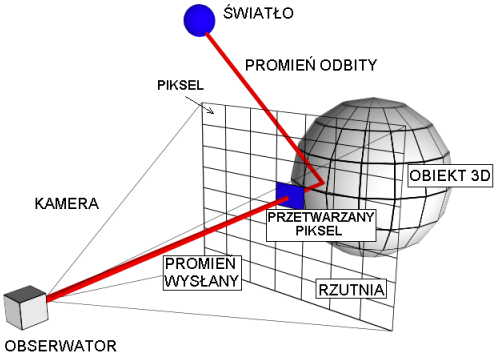
Wprowadzenie do Raytracingu

3.1 Wstępny opis

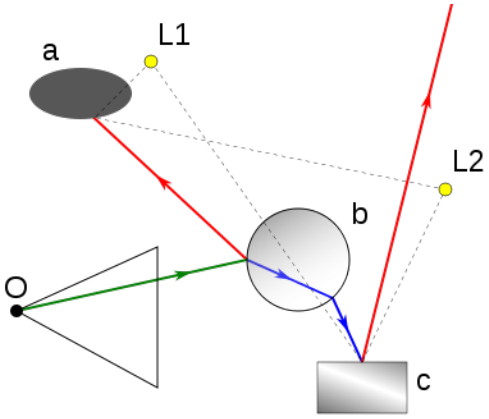
W rzeczywistym świecie promienie świetlne rozchodzą się od źródła światła do obiektów znajdujących się w świecie. Każde źródło światła wysyła nieskończoną liczbę swoich promieni świetlnych. Następnie te promienie odbijając się od obiektów i trafiają do oczu obserwatora powodując, że widzi on określony kolor danego obiektu. Gdyby zaadaptować tą metodę do generowania realistycznej grafiki komputerowej, otrzymalibyśmy dokładny i realistyczny obraz. Jednak z racji tego, że sprzęt komputerowy ma ograniczone możliwości, a metoda ta jest bardzo nieefektywną metodą pod względem obliczeniowym. Najszerzej stosowaną metodą śledzenia promieni jest wsteczne śledzenie promieni (backward raytracing). W odróżnieniu od postępowego algorytmu śledzenia promieni (forward raytrcing), które opiera się na generowaniu jak największej liczby promieni dla każdego źródła światła. Algorytm wstecznego śledzenia promieni zakłada, że promienie śledzone są od obserwatora, poprzez scenę do obiektów z którymi kolidują. Na rysunku 3.1 przedstawiony jest poglądowy schemat śledzenia pojedynczego promienia od obserwatora poprzez określony piksel na ekranie

3.2 Rekursywna metoda śledzenia promieni

Przy omawianiu wstecznej metody śledzenia promieni warto wspomnieć o raytracingu rekursywnym. W zagadnieniu tym bada się rekurencyjnie promienie odbite zwierciadlane oraz załamane, które powstały z kolizji promieni pierwotnych z obiektami na scenie. Tak więc żywotność promienia pierwotnego wcale nie kończy się w momencie kolizji z obiektem sceny. To czy z danego promienia pierwotnego wygenerowane zostaną kolejne promie-



Rysunek 3.1: Sposób określania barwy piksela w raytracigu
[10] "http://pl.wikipedia.org/wiki/Ray˙tracing"



Rysunek 3.2: Zasada działania rekursywnego algorytmu raytracingu
[10] "http://pl.wikipedia.org/wiki/Ray˙tracing"

nie w bardzo dużej mierze zależy od materiału jakim pokryty jest dany obiekt sceny. Z pomocą tej rekursywnej metody śledzenia promieni jesteśmy w stanie zasymulować obiekty lustrzane oraz obiekty półprzezroczyste. Rekurencja w tej metodzie trwa do osiągnięcia maksymalnego stopnia zagłębienia. Kolor wynikowy danego pojedynczego piksela powstaje z sumy kolorów, obiektu w jaki trafił promień pierwotny oraz kolorów obiektów w jakie trafiły promienie wtórne. Na rysunku 3.2 przedstawiony jest poglądowy schemat zasady działania rekursywnej metody śledzenia promieni.

3.3 Przedstawienie algorytmu śledzenia promieni

Śledzenie promieni przez scenę rozpoczyna się od obserwatora określanego często jako kamery występującej na scenie. Przez każdy piksel ekranu

śledzone są promienie które poruszają się po scenie. Gdy któryś ze śledzonych promieni napotka obiekt i zacznie z nim kolidować, wtedy z takiego promienia pierwotnego generowane są promienie wtórne odbite i załamane, oczywiście w zależności od materiału jakim pokryty jest obiekt.

Poniżej przedstawiony jest schematyczny przebieg algorytmu wstecznego śledzenia promieni:

```
1
2  Śledź promienie pierwotne

4  Sprawdź kolizje ze wszystkimi obiektami

6  Kolor piksela = kolor otoczenia

8  LOOP( Dla każdego zródła światła ) {
9    Śledź promień cienia
10   Kolor piksela = współczynnik cienia * kolor obiektu w który
11                               trafił promień
12 }

14 IF( Obiekt ma właściwości odbijające ) {
15   kolor piksela += współczynnik odbicia * śledź promień
16                                     odbity
17 }
18 IF( Obiekt ma właściwości załamujące ) {
19   kolor piksela += współczynnik załamania * śledź promień
20                                     załamany
21 }
```

3.4 Sposób zrównoleglenia algorytmu śledzenia promieni

Z racji tego, że w standardowym wstecznym algorytmie śledzenia promieni, promienie przechodzące przez poszczególne piksele ekranu nie są od siebie zależne, jesteśmy w stanie dokonywać na nich równoległych obliczeń. W nieniejszej pracy uwaga skupiona została na technologię NVIDIA CUDA i to właśnie za pomocą niej jesteśmy w stanie dokonać takich równoległych obliczeń. Potrzebne do tego jest przeniesienie algorytmu śledzenia promieni z wersji CPU, gdzie zazwyczaj odbywa się ona w sposób iteracyjny, poprzez przechodzenie w pętli kolejnych pikseli ekranu. Tak przeniesiony algorytm bedzie wykonywany na każdym z wątków udostępnionych przez CUDA dla danej karty graficznej. Wszelkie obliczenia będą wykonywały się równolegle.

Rozdział 4

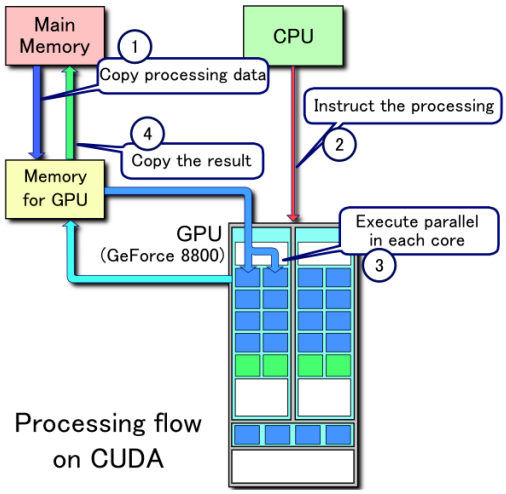
NVIDIA CUDA jako znakomita platforma do zrównoleglenia obliczeń

4.1 Wstępny opis

CUDA(Compute Unified Device Architecture) jest dość nową technologią wprowadzoną na rynek przez firmę NVIDIA. Technologia ta swój początek miała w 2007 roku. Od samego początku stała się ona wiodącą technologią przetwarzania strumieniowego z wykorzystaniem GPU. CUDA jako, że jest technologią stworzoną przez firmę NVIDIA, wspierana jest przez układy graficzne właśnie tej firmy. Wsparcie dla tej technologii rozpoczęło się od układów graficznych serii GeForce 8, Quadro oraz Tesla. Seria układów graficzny Quadro oraz Tesla są wyspecjalizowanymi układami obliczeniowymi do zastosowań naukowych. Natomiast serie GeForce można spotkać na co dzień w komputerach stacjonarnych oraz laptopach. Z pomocą technologii CUDA jesteśmy w stanie uzyskać wielokrotne przyśpieszenie w obliczeniach w stosunku do obliczeń na zwykłym procesorze CPU. na ryskunku 4.1 przedstawiony został przykładowy schemat przepływu obliczeń w CUDA.

4.2 Wspierane karty oraz zdolność obliczeniowa

We wstępnym opisie powiedziane było, że technologia CUDA zapoczątkowana była w układach graficznych serii GeForce, Tesla oraz Quadro. W tabeli 4.1 przedstawione zostało oficjalne wsparcie określonej wersji CUDA w poszczególnych układach graficznych. [9]



Rysunek 4.1: Przykład przepływu przetwarzania w technologii CUDA. "http://en.wikipedia.org/wiki/CUDA"

Kolejną ważną rzeczą wyróżniająca karty graficzne jest ich zdolność obliczeniowa (ang. compute capability). Identyfikuje ona możliwości obliczeniowe danej karty graficznej w odniesieniu do technologii NVIDIA CUDA. W tabeli 4.2 przedstawione zostały możliwości kart graficznych w zależności od profilu CUDA.[9]

4.3 Architektura

Karty graficzne GPU znacznie różnią się architekturą oraz wydajnością od zwykłych procesorów CPU. Różnica w wydajności wynika głównie z faktu, iż procesory graficzne specjalizują się w równoległych, wysoce intensywnych obliczeniach. Karty graficzne składają się z większej liczby tranzystorów które są odpowiedzialne za obliczenia na danych. Nie posiadają natomiast takiej kontroli przepływu instrukcji oraz jednostek odpowiedzialnych za buforowanie danych jak procesory komputerowe CPU. Układy graficzne wspierające technologię CUDA zbudowane są z multiprocesorów strumieniowych (ang. stream multiprocessor). Różne modele kart graficznych firmy NVIDIA posiadają różną liczbę multiprocesorów, co przekłada się także na wydajność i zdolność obliczeniową danej architektury. Na rysunku 4.2 przedstawiona jest przykładowa budowa takiego właśnie multiprocesora.

Każdy z multiprocesorów składa się z: [4]

- I-Cache - bufor instrukcji.

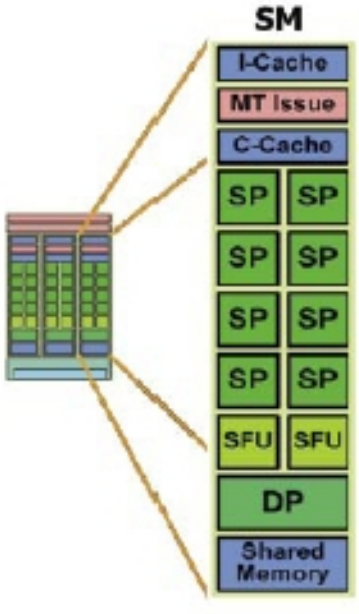
Zdolność obliczeniowa (wersja)	GPUs	Cards
1.0	G80	GeForce 8800GTX/Ultra/GTS, Tesla C/D/S870, FX4/5600, 360M
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT, 9600GT/GSO, 9800GT/GTX/GX2, GTS 250, GT 120/30, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50
1.2	GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M
1.3	GT200, GT200b	GTX 260/75/80/85, 295, Tesla C/M1060, S1070, CX, FX 3/4/5800
2.0	GF100, GF110	GTX 465, 470/80, Tesla C2050/70, S/M2050/70, Quadro 600,4/5/6000, Plex7000, 500M, GTX570, GTX580
2.1	GF108, GF106, GF104	GT 420/30/40, GTS 450, GTX 460

Tablica 4.1: Zestawienie kart graficznych oficjalnie wspierających technologię CUDA. [9]

- MT Issue - jednostka która rozdziela zadania dla SP i SFU.
- C-Cache - bufor stałych (ang. constant memory) o wielkości 8KB, który przyspiesza odczyt z obszaru pamięci stałej.
- 8 x SP - 8 jednostek obliczeniowych tzw stream processors, które wykonują większość obliczeń pojedynczej precyzji (każdy zawiera własne 32-bitowe rejestry).
- 2 x SFU - jednostki specjalne (ang. special function units). Zadaniem ich jest obliczanie funkcji przestępnych, np. trygonometrycznych, wykładniczych i logarytmicznych, czy interpolacja parametrów.

Zdolność obliczeniowa	1.0	1.1	1.2	1.3
Funkcje atomowe w pamięci globalnej	-	✓	✓	✓
Funkcje atomowe w pamięci współdzielonej	-	-	✓	✓
Ilość rejestrów na multiprocesor	8192	8192	16384	16384
Maksymalna liczba warpów na multiprocesor	24	24	32	32
Maksymalna liczba aktywnych wątków na multiprocesor	768	768	1024	1024
Podwójna precyzja	-	-	-	✓

Tablica 4.2: Porównanie zdolności obliczeniowych kart graficznych wspierających NVIDIA CUDA.[9]



Rysunek 4.2: Przykładowy schemat multiprocesora strumieniowego.
[4] "http://software.com.pl/czyn-cuda-czesc-1-architektura/"

- DP - procesor, który wykonuje obliczenia podwójnej precyzji.
- SM - pamięć współdzielona (ang. shared memory) o wielkości 16KB.

4.4 Rodzaje pamięci w architekturze CUDA

- Pamięć globalna (ang. global memory) - Ta pamięć jest dostępna dla wszystkich wątków. Nie jest pamięcią buforowaną. Dostęp do niej trwa od około 400 do 600 cykli. Pamięć ta służy przede wszystkim do zapisuj wyników działań programu obliczeniowego.[4]

- Pamięć lokalna (ang. local memory) - Ma taki sam czas dostępu jak pamięć globalna (400-600 cykli). Nie jest także pamięcią buforowaną. Jest ona zdefiniowana dla danego wątku. Każdy wątek CUDA posiada własną pamięć lokalną. Zajmuje się ona przechowywaniem bardzo dużych struktur danych. Pamięć ta jest najczęściej używana gdy obliczenia danego wątku nie mogą być w całości wykonane na dostępnych rejestrach procesora graficznego.[4]

- Pamięć współdzielona (ang. shared memory) - Jest to bardzo szybki rodzaj pamięci, dorównujący szybkości rejestrom procesora graficznego. Przy pomocy tej pamięci, wątki przydzielone do jednego bloku są wstanie się ze sobą komunikować. Należy jednak obchodzić się ostrożnie z tym rodzajem pamięci, gdyż mogą powstać momenty w których wątki w jednym bloku będą chciały jednocześnie zapisywać i odczytywać z tej pamięci. Występowanie takich konfliktów w odczycie i zapisie powoduje duże opóźnienia.[4]

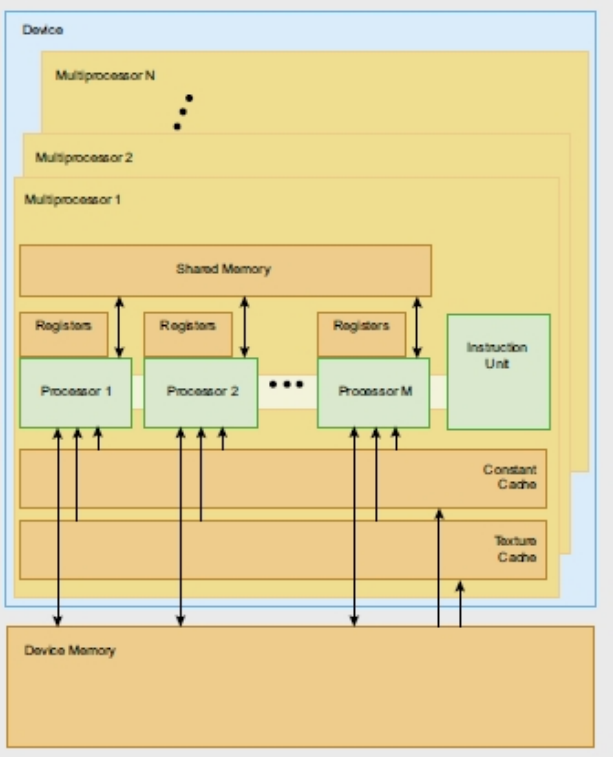
- Pamięć stała (ang. const memory) - Ta pamięć w odróżnieniu do powyższych rodzajów pamięci, jest buforowaną pamięcią tylko do odczytu. Gdy potrzebne dane znajdują się aktualnie w buforze dostęp do nich jest bardzo szybki. Czas dostępu rośnie gdy danych nie ma w buforze i muszą być doczytane z pamięci karty.[4]

- Pamięć Tekstur (ang. texture memory) - Jest pamięcią podobną do pamięci stałej gdyż udostępnia tylko odczyt danych. Jest także pamięcią buforowaną. W pamięci tej bufor danych został zoptymalizowany pod kątem odczytu danych z bliskich sobie adresów. Najkorzystniejszą sytuacją jest gdy wątki dla danego warpa (grupa 32 wątków zarządzanych przez pojedynczy multiprocesor) odczytują adresy, które znajdują się blisko siebie. CUDA w swojej implementacji udostępnia możliwość posługiwania się teksturami 1D,2D,3D.[4]

- Rejestry - Jest to najszybszy rodzaj pamięci. Dostęp do niego nie powoduje żadnych dodatkowych opóźnień, chyba że próbujemy odczytać z rejestru do którego dopiero co zostało coś zapisane. Każdy multiprocesor w urządzeniu CUDA posiada 8192 lub 16384 rejestrów

32-bitowych. Zależy to od wersji(zdolności obliczeniowej) danego urządzenia. W celu uniknięcia powyższych konfliktów ilość wątków na pojedynczy multiprocesor ustawia się jako wielokrotność liczby 64. [4]

Na rysunku 4.3 poniżej przedstawiony został poglądowy schemat pamięci w architekturze CUDA.



Rysunek 4.3: Schemat pamięci.
[4] "http://software.com.pl/czyn-cuda-czesc-1-architektura/"

4.5 Przykładowy program pod architekturę CUDA

Poniżej przedstawiony został przykład programu napisanego w języku C dla architektury CUDA. Program ten uruchamiany jest na wielu wątkach karty graficznej, każdy z tych wątków niezależnie wpisuję do tablicy swoje ID. Ważną informacją przy pisaniu kodu dla architektury CUDA jest to, że funkcje uruchamiane przez wątki mają specjalne oznaczenia:

- global - funkcje taką wywołać można tylko z CPU, a wykonuje się ona na GPU

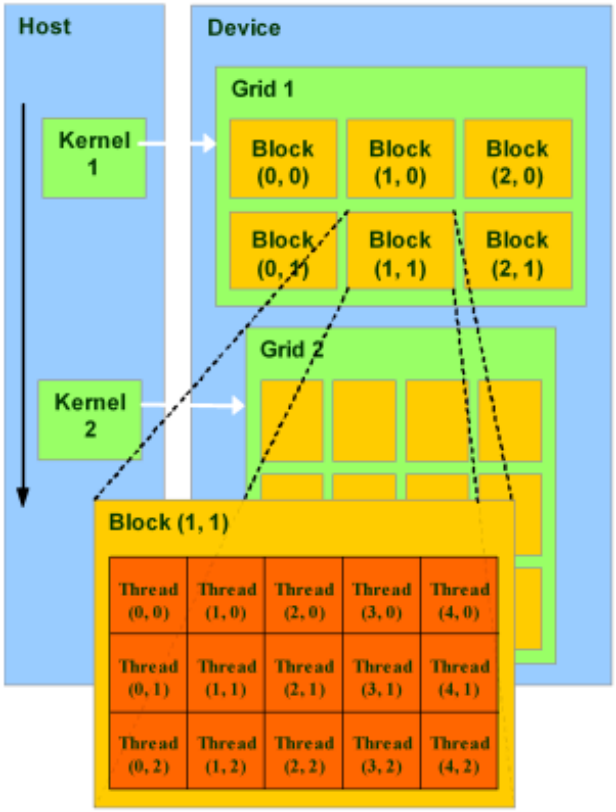
- host - funkcja wykonuje się i może być wywołana tylko z kodu wykonywanego na CPU

- device - funkcja wykonuje się i może być wywołana tylko z kodu wykonywanego na GPU

Należy także pamiętać, że funkcje dla wątków CUDA muszą zawsze zwracac wartość void.

```
1  #include <stdlib.h>
2  #include <cuda_runtime.h>
3  #include <cutil.h>

5  // definicja funkcji która będzie uruchamiana
6  // równolegle na wątkach CUDA
7  __global__ void testFunction(int *data)
8  {
9    // obliczamy index tablicy a zarazem wątku
10   int id = blockIdx.x * blockDim.x + threadIdx.x;
11   // Zaposujemy do tablicy ID wątku
12   data[id] = id;
13 }

15 // W funkcji main wywołujemy powyższą funkcje
16 // dla wątków CUDA
17 int main()
18 {
19   // Na poczatku należey zainicjować urządzenie CUDA
20   cudaSetDevice(0);

22   // alokujemy pamieć na karcie graficznej
23   int *tablica;
24   cudaMalloc((void**)&tablica, sizeof(int) * ARRAY_SIZE);

26   // Ustalamy wielkosc bloku i karty
27   dim3 dimBlock(BLOCK_SIZE, 1);
28   dim3 dimGrid(ARRAY_SIZE / dimBlock.x, 1);

30   // wywołujemy naszą funkcję obliczeniową
31   testFunction<<<dimGrid, dimBlock>>>(tablica);

33   // Tworzymy tablice w pamieci ram i kopujemy
34   // dane z karty graficznej do pamieci ram.
35   int *tablica2 = (int*)malloc(sizeof(int) * ARRAY_SIZE);
36   cudaMemcpy(tablica, tablica2, sizeof(int) * ARRAY_SIZE,
37   cudaMemcpyDeviceToHost);

39   return 0;
40 }
```

Jak widzimy na powyższym listingu kodu gdy wywołujemy funkcję CUDA określamy na ilu wątkach ma się ona uruchomić i w jakie grupy mają być one pogrupowane. Na rysunku 4.4 przedstawiony został schemat pokazujący jak może wyglądać ułożenie używanych wątków w całej kracie, pogrupowanych w odpowiednie bloki. Podczas programowania na karty graficzne CUDA należy pamiętać o różnych dostępnych rodzajach pamięci i wybrać tą najłaściwszą. Jeśli nie przemyślimy dobrze problemu jaki sobie założyliśmy rozwiązać przy pomocy technologii CUDA, może się zdażyć, że nasze rozwiązanie będzie działało gorzej niż na procesorze CPU. Należy także poinformować o tym, że brak jest narzędzi, które wspomagały by śledzenie przepływu wykonywania programu tzw. debugowanie. Z tym problemem borykaja się wszystkie technologie zwiazane z GPGPU(obliczenia przeprowadzane na kartach graficznych).



Rysunek 4.4: Przykładowy schemat pokazujący ułożenie wątków CUDA w blokach oraz w całej kracie [7] "http://home.mindspring.com/˜eric˙rollins/ray/cuda.html"

Rozdział 5

Projekt aplikacji testowej

5.1 Założenia

Na potrzeby niniejszej pracy zostało opracowane autorskie rozwiązanie uniwersalnego wstecznego raytracera działającego zarówno na procesorze CPU jak i również na ko-procesorach graficznych GPU firmy NVIDIA. Aplikacja testowa jest wstanie generować wynikowe obrazy scen 3D składających się z kul, prostopadłościanów oraz płaszczyzn. Na każdy z elementów sceny jest możliwość nałożenia dowolnej tekstury oraz doboru odpowiednich parametrów materiału. Dodatkowo na scenie możliwe jest umieszczanie świateł punktowych. Aplikacja sama w sobie jest benchmarkiem, który potrafi przetestować zadaną liczbę scen 3D na komputerze użytkownika. Zebrane wyniki z obliczeń jest wstanie przesłać na wybrany adres e-mail (w tym przypadku za zgodą użytkownika do developera). Aplikacja przy generowaniu obrazu sceny 3D bierze pod uwagi różne właściwości materiału danego obiektu. Docelowo generowane są takie efekty jak: oświetlenie, odblask, cienie, wielokrotne odbicia i załamania, tekstury. Przy użyciu materiałów o różnych parametrach jesteśmy wstanie uzyskać bardzo ciekawie wyglądające obiekty np: lustro, szkło, metale i wiele innych.

5.2 Implementacja

Aplikacja testowa została napisana w języku C++, wykorzystując biblioteki standardowe pochodzące z języka C. Wersja śledzenia promieni przy użyciu technologii CUDA została napisana w tzw. „C for CUDA". Dodatkowo do wyświetlania wynikowych obrazów użyta została biblioteka Microsoft DirectX 9.0. Wersja śledzenia promieni działająca na CPU jest także zrównoleglona na wszystkie procesory znajdujące się w danym kom-

puterze. Użyta do tego została biblioteka open source „OpenMP". Program przeznaczony jest do uruchamiania na systemach z rodziny Windows. Aplikację testową można nazwać swoistym benchmarkiem. Działanie jej składa się z 5 ważnych punktów:

- wczytywanie scen do testow
- testowanie zadanych scen na procesorze CPU.
- testowanie zadanych scen na karcie graficznej GPU.
- zapisywanie wynikowych obrazów scen na dysk użytkownika
- zebranie informacji o testowanych scenach i wysłanie ich na mail developera.

Przebieg działania:
Aplikacja na samym początku wczytuje plik benchamarku z rozszerzeniem *.rtb. Plik ten zawiera w sobie spis scen (pliki *.rtm) które mają być przetestowane przez raytracer. Następnie rozpoczyna się testowanie zadanych scen na procesorze CPU. Gdy wszystkie sceny zostaną przetestowane na procesorze, rozpoczyna się raytracing na karcie graficznej z użyciem CUDA. Na koniec gdy już wszystkie sceny zostały wygenerowane przy użyciu CPU oraz GPU, wynikowe obrazy generowane przez raytracer zapisywane są na dysku użytkownika. Zebrane informacje z profilowania każdej ze scen zostają zapisane do pliku oraz wysłane na adres e-mail developera.

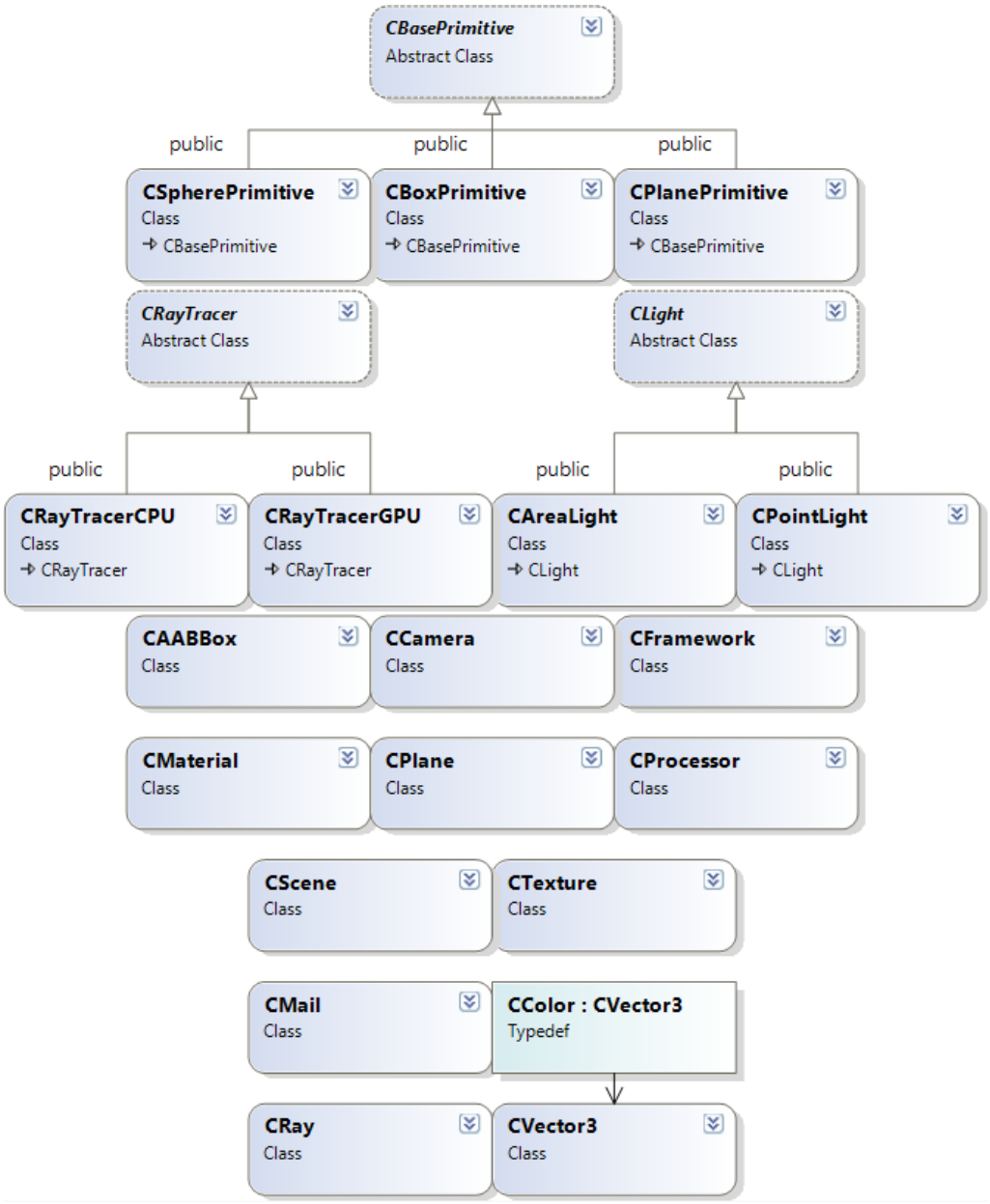
Statystyki związane z kodem aplikacji testowej:

- 61 plików kodu
- 9386 linii kodu
- 272697 bajtów kodu

Na rysunku 4.1 zaprezentowany został diagram najważniejszych klas dla aplikacji testowej raytracera

5.3 Zestaw testów

By wykazać przyśpieszenie pomiędzy śledzeniem promieni na procesorze CPU a kartą graficzną GPU przygotowany został zestaw 9 scen testowych. Testowana jest wydajność generowania scen o różnej budowie i występujących na niej prymitywach. W scenach tych testowanych jest wiele parametrów takich jak:



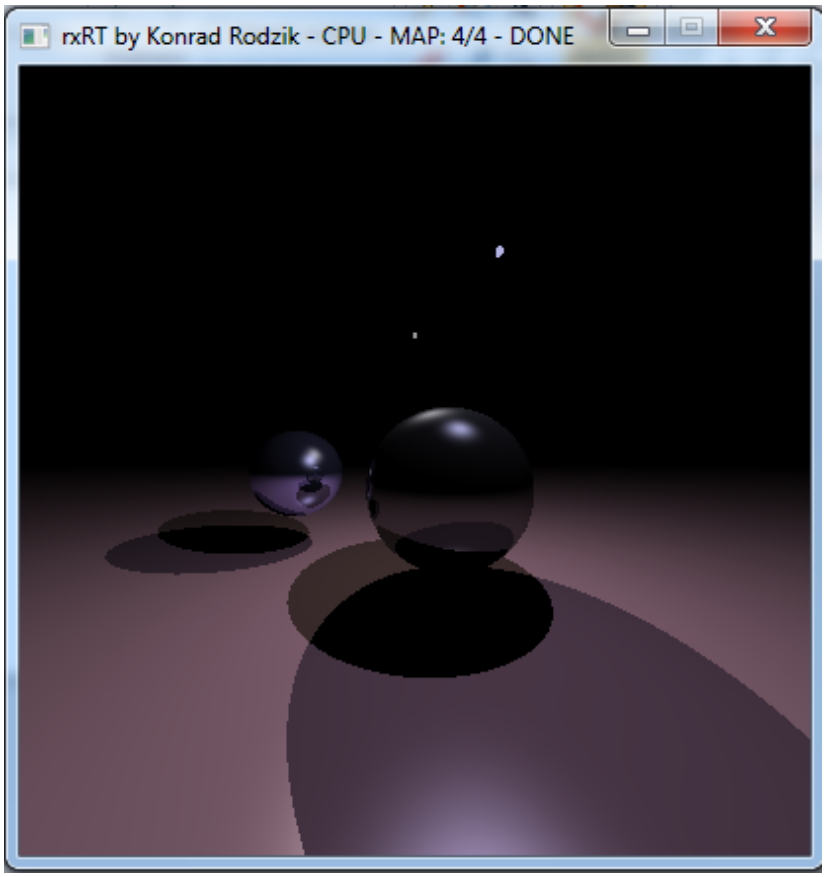
Rysunek 5.1: Diagram klas aplikacji testowej. Obraz wygenerowany własnoręcznie przy użyciu Visual Studio 2008.

- Odbicia promieni od obiektów na scenie
- Załamania promieni w obiektach na scenie
- Teksturowanie obiektów sceny
- Różna liczba świateł punktowych na scenie

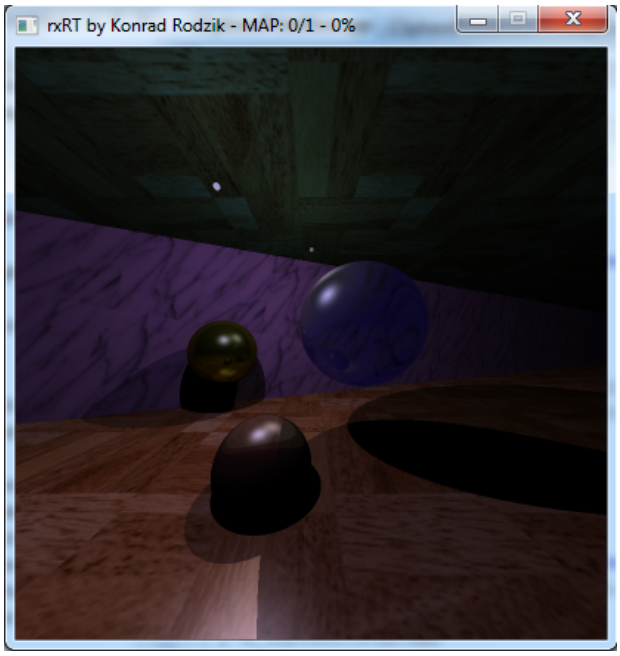
- Rodzaj oraz liczba prymitywów wyświetlanych na scenie
- Jakość generowanego obraz (super sampling)
- Rozdzielczość generowanego obrazu

5.4 Przykłady wygenerowanych obrazów

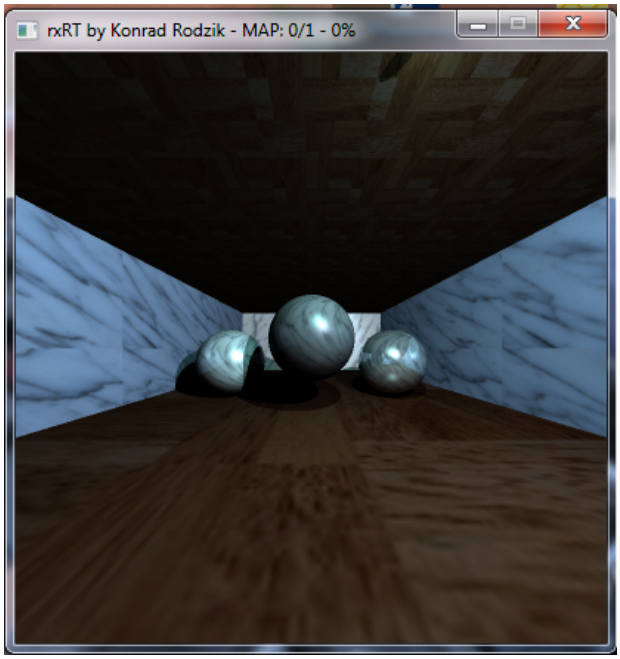
W rozdziale tym przedstawione zostały wyniki generowania scen przez aplikacje testową. Każda z tych scen była generowana na procesorze CPU oraz na karcie graficznej GPU.



Rysunek 5.2: Przykładowa wygenerowana scena 1. Przedstawione wielokrotne odbicia kul na scenie z dwoma źródłami światła punktowego przy włączonych cieniach rzucanych przez obiekty.



Rysunek 5.3: Przykładowa wygenerowana scena 2. Wielokrotne załamania i odbicia promieni świetlnych. Dodatkowo włączone teksturowanie oraz cienie.



Rysunek 5.4: Przykładowa wygenerowana scena 3. Teksturowanie wszelkich możliwych prymitywów sceny (kulke, pudełka, płaszczyzny). Dodatkowo włączone cieniowanie oraz odblask (ang. specular)

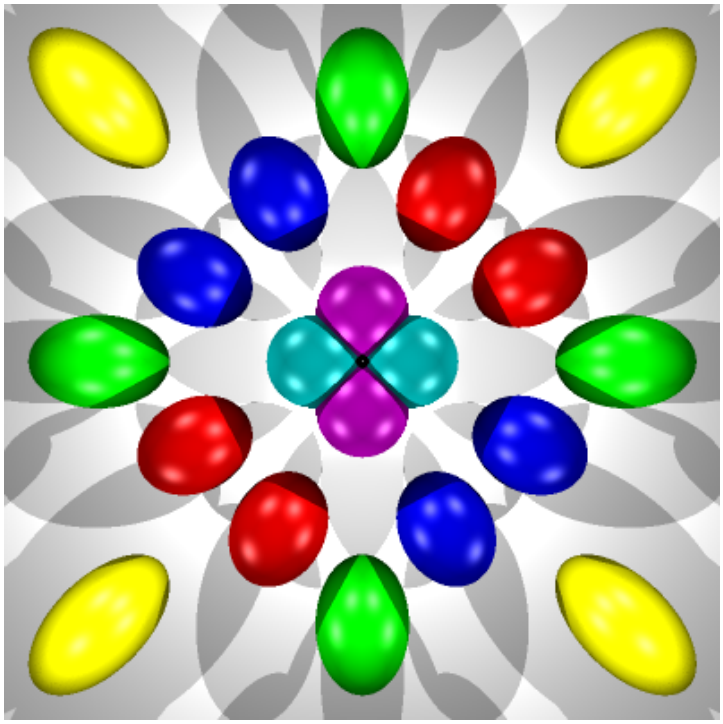
Rozdział 6

Porównanie wydajności Raytracera działającego na CPU oraz na GPU

W rozdziale tym przeprowadzone zostały testy wydajnościowe wstecznej metody śledzenia promieni, działającej na CPU oraz GPU. Przeprowadzonych zostało dziewięć testów, każdy z nich opisany jest w osobnym rozdziale. Do każdego z testów został załączony wygenerowany obraz z danej sceny oraz wykresy wydajności raytracera na różnych konfiguracjach sprzętowych. Wykresy przedstawiają czasy generowania każdej ze scen w milisekundach, a tym samym pokazują przyspieszenie jakie udało się uzyskać pomiędzy procesorami CPU a kartami graficznymi GPU.

6.1 Test 1 - Scena złożona z samych sfer

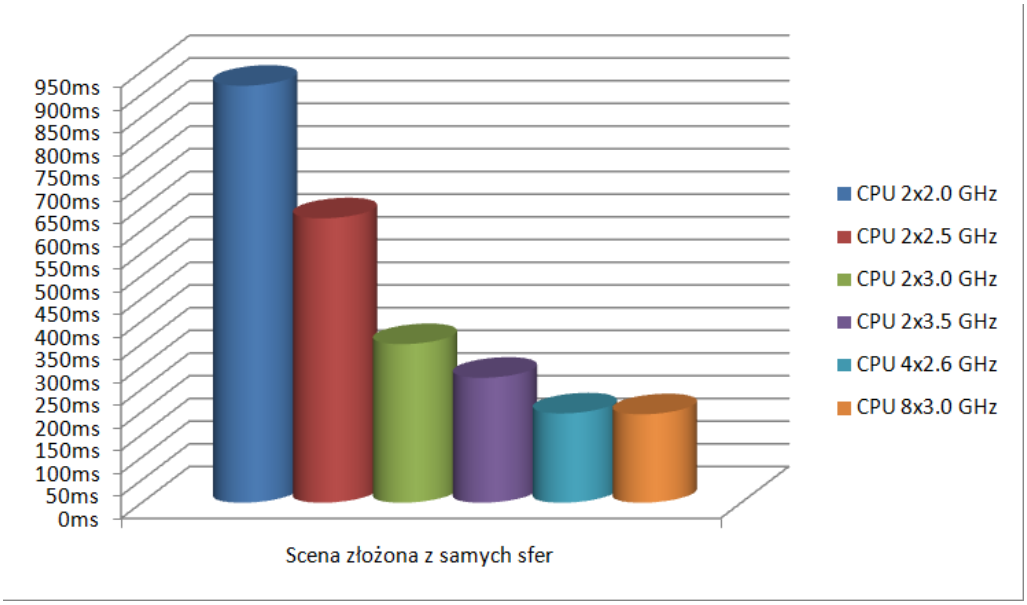
Testowi poddana została scena zawierająca jako prymitywy same sfery. Na scenie jest dokładnie 20 sfer, które mają nałożone różne materiały. Na scenie tej znajdują się także 4 światła punktowe. Na rysunku 6.1 widoczna jest scena, która została poddana testowi śledzenia promieni na różnych konfiguracjach procesorów CPU oraz różnych konfiguracjach kart graficznych.



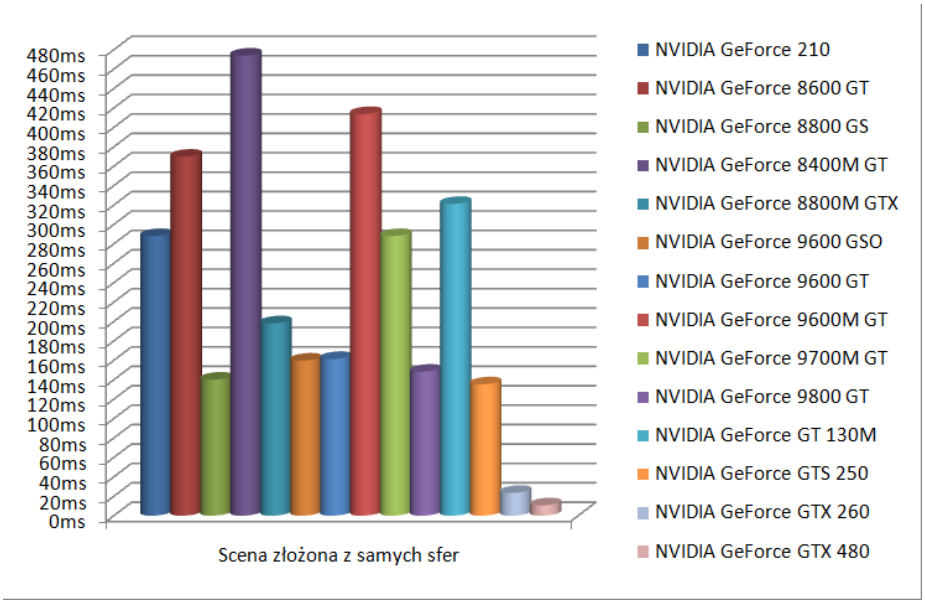
Rysunek 6.1: Scena złożona ze sfer

Na poniższych wykresach zauważyć możemy że już przy prostych scenach, złożonych z samych sfer, widoczne jest przyspieszenie obliczeń uzyskane na kartach graficznych NVIDIA. Najlepszy z rodziny procesorów (Intel Core I7)brany pod uwagę w testach osiągnął wynik w granicach 150ms. Natomiast najlepsza dostępna na rynku karta użytkowa GeForce GTX 480 zmiejszyła czas generowania sceny poniżej 10ms.

WYKRESY:



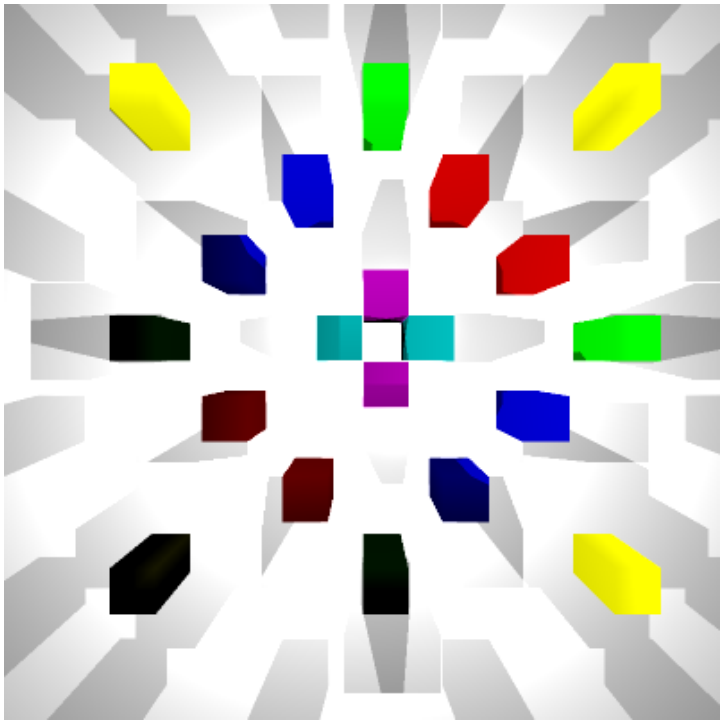
Rysunek 6.2: Wykresy CPU dla sceny złożonej z samych sfer



Rysunek 6.3: Wykresy GPU dla sceny złożonej z samych sfer

6.2 Test 2 - Scena złożona z samych pudełek

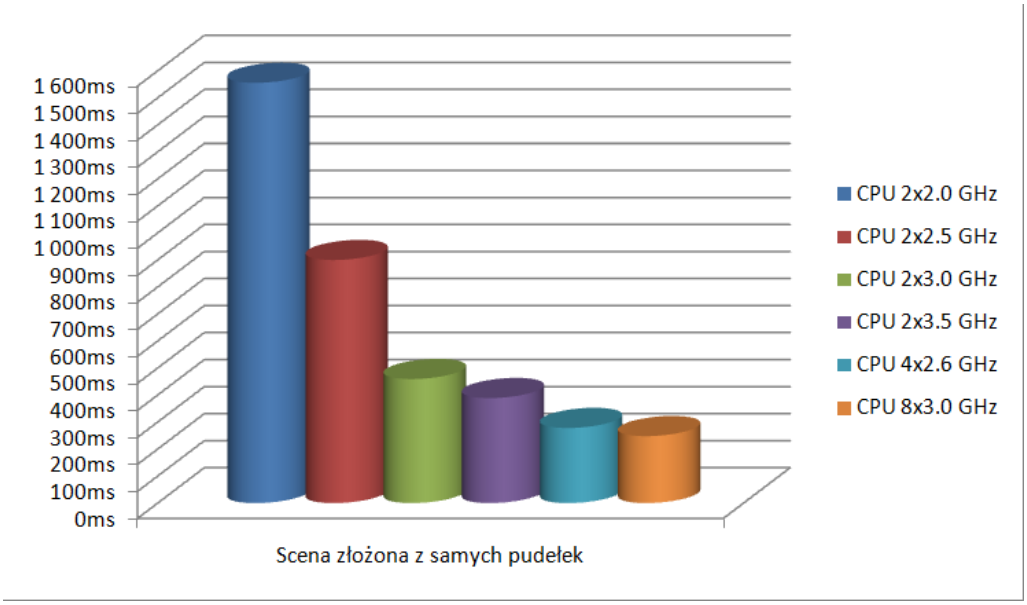
W tym teście na scenie zostało umieszczonych 20 pudełek (boxów). Każde z tych pudełek posiada swój własny materiał. Scena również oświetlana jest przez 4 źródła światła punktowego. Na rysunku 6.4 widoczna jest scena samych pudełek, która została poddana testowi śledzenia promieni na różnych konfiguracjach procesorów CPU oraz różnych konfiguracjach kart graficznych.



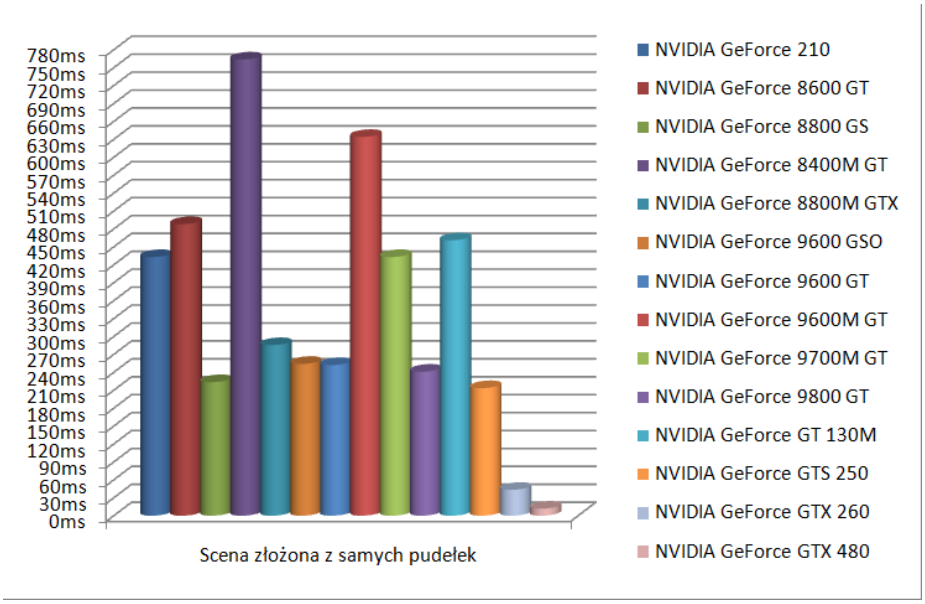
Rysunek 6.4: Scena złożona ze sfer

Na poniższych wykresach zauważyć możemy, że scena samych pudełek potrzebuje więcej czasu na wygenerowanie się niż scena samych sfer. Wynika to z faktu iż potrzeba więcej obliczeń przy testowaniu kolizji promienia z pudełkiem, niż ze sfera. Przy pudełku musimy sprawdzić wszystkie sześć scian danego prymitywu. Niemniej jednak jak i przy scenie sfer tak i przy scenie pudełek widoczne jest przyspieszenie uzysnake na kartach grficznych NVIDIA. Najnowszej generacji procesor uzyskał czasy rzędu 200ms. Karta graficzna GTX480 zmiejszyła czas generowania do niecałych 13ms.

WYKRESY:



Rysunek 6.5: Wykresy CPU dla sceny złożonej z samych pudełek



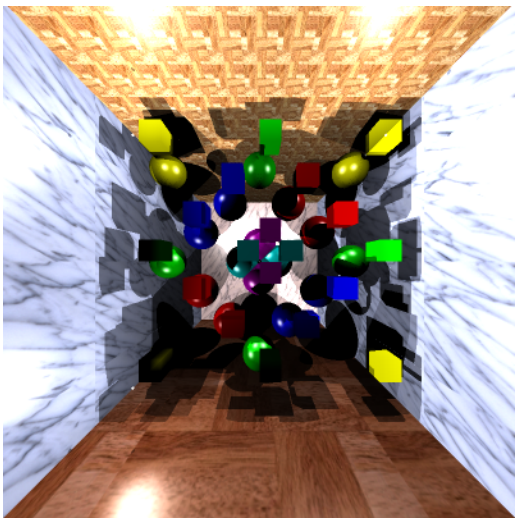
Rysunek 6.6: Wykresy GPU dla sceny złożonej z samych pudełek

6.3 Test 3 - Scena z różnymi prymitywami

Testowi poddana została scena z różnego rodzaju prymitywami. Każdy z prymitywów posiada swój materiał, niektóre także teksture. Scena składa się dokładnie z:

- dwudziestu sfer
- dwudziestu pudełek
- sześciu płaszczyzn
- czterech źródeł światła punktowego.

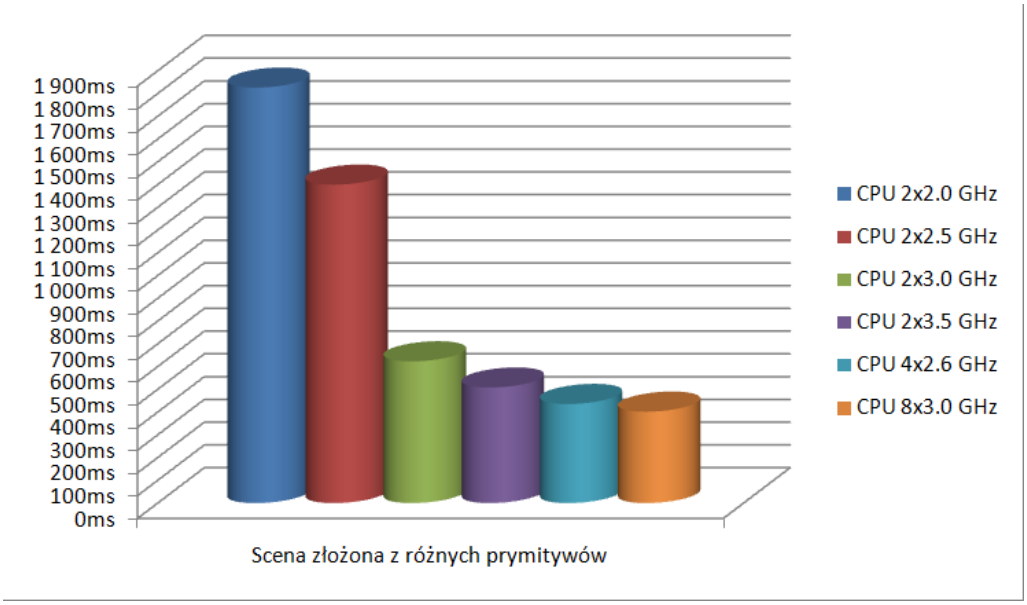
Na rysunku 6.7 widoczny jest obraz z wygenerowanej sceny składającej się z róznego rodzaju prymitywów. Jak wcześniejsze sceny, także i ta została poddana testowi na różnych konfiguracjach procesorów CPU oraz kart graficznych GPU.



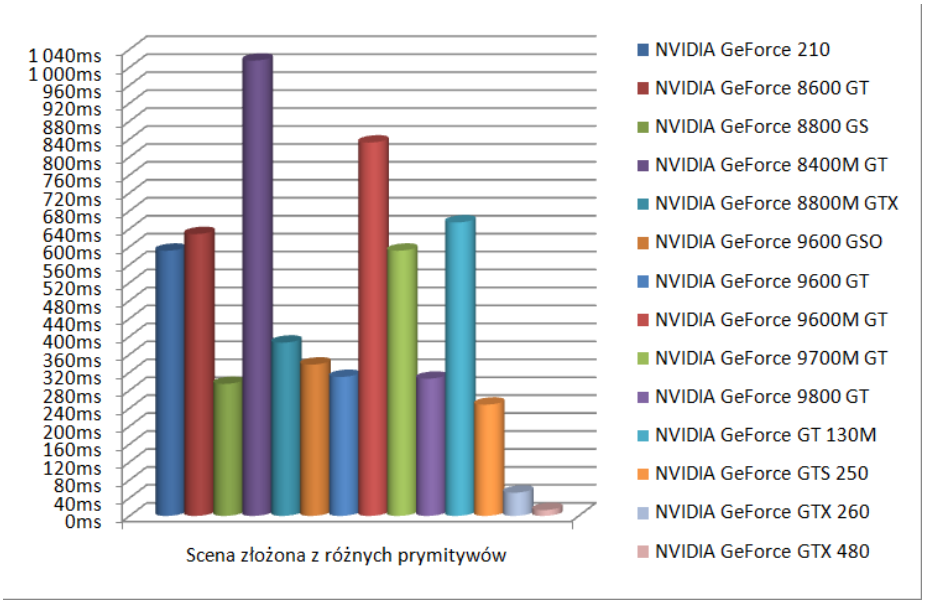
Rysunek 6.7: Scena złożona z różnego typu prymitywów

Z wykresów tej sceny widać iż ogólny czas generowania zwiększył się dość znacznie. Spowodowane jest to tym iż na scenie znajduje się około 50 prymitywów. Dodatkowo widzimy, że na niektórych z nich zastosowane jest teksturowanie. Karty graficzne NVIDIA nawet przy takich scenach radzą sobie dość dobrze, i wyprzedzają procesory CPU. Czas generowania procesora Intel Core I7 wynosi ponad 300ms. Natomiast karta graficzna GTX480 nadal generuje sceny poniżej 15ms.

WYKRESY:



Rysunek 6.8: Wykresy CPU dla sceny złożonej z różnych prymitywów



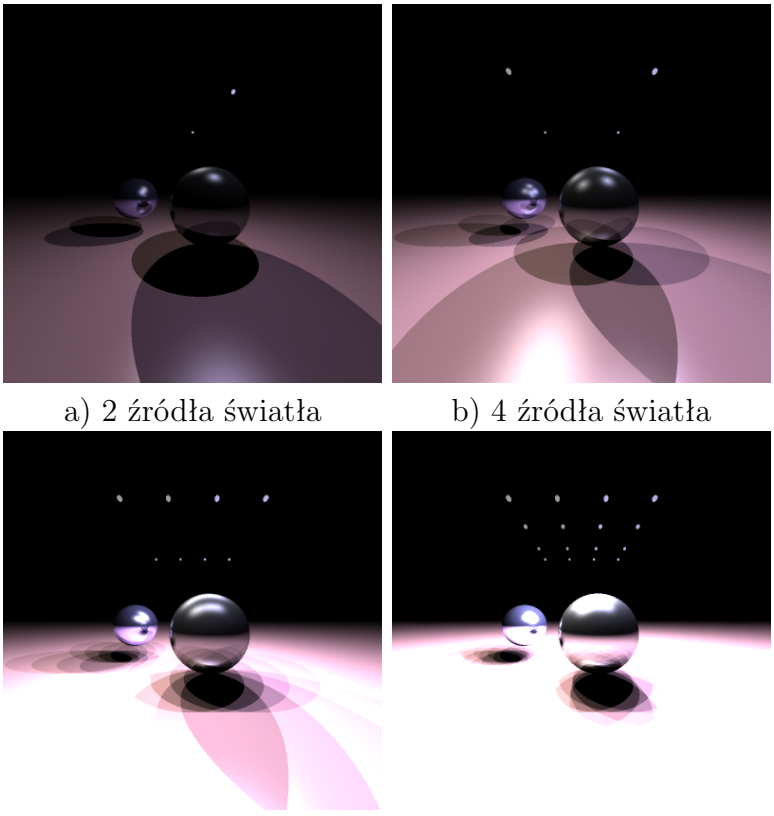
Rysunek 6.9: Wykresy GPU dla sceny złożonej z różnych prymitywów

6.4 Test 4 - Różna liczba świateł punktowych na scenie

W teście sprawdzone zostało jaki wpływ ma liczba świateł punktowych na szybkość generowanej sceny. Testy zostały wykonane dla czterech różnych wariantów:

- dwa źródła światła punktowego.
- cztery źródła światła punktowego.
- osiem źródeł światła punktowego.
- szesnaście źródeł światła punktowego.

Na rysunku 6.10 przedstawione są cztery warianty ułożenia oraz liczby źródeł światła. Testy przebiegły pomyślnie dla procesorów komputerowych CPU oraz kart graficznych GPU.



a) 2 źródła światła b) 4 źródła światła

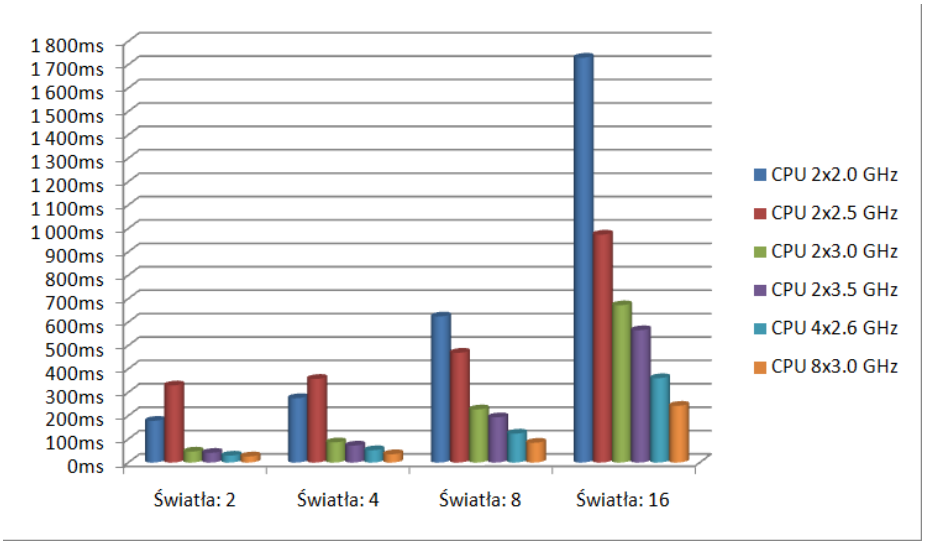
c) 8 źródeł światła d) 16 źródeł światła

Rysunek 6.10: Obrazy wygenerowane w teście o różnej liczbie świateł

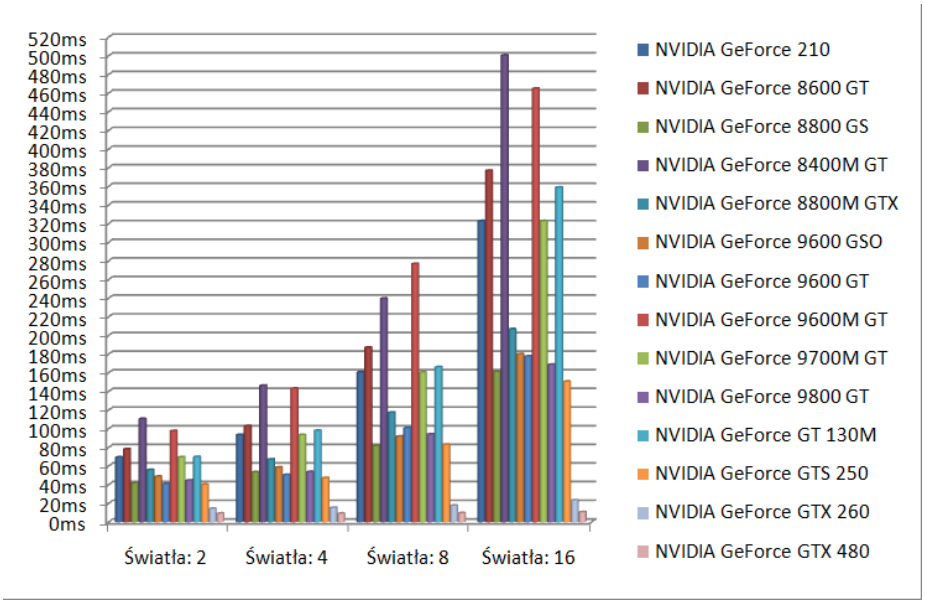
Jak widzimy na poniższych wykresach, czas generowania sceny zwiększa się wraz z ilością świateł punktowych umieszczonych na scenie. Dzieje

się tak dlatego, ponieważ w procesie raytracingu od każdego źródła światła śledzone są promienie cienia do punktów intersekcji z prymitywami. Dla GeForce GTX 480 nawet scena z wieloma źródłami świateł nie stanowi dużego problemu.

WYKRESY:



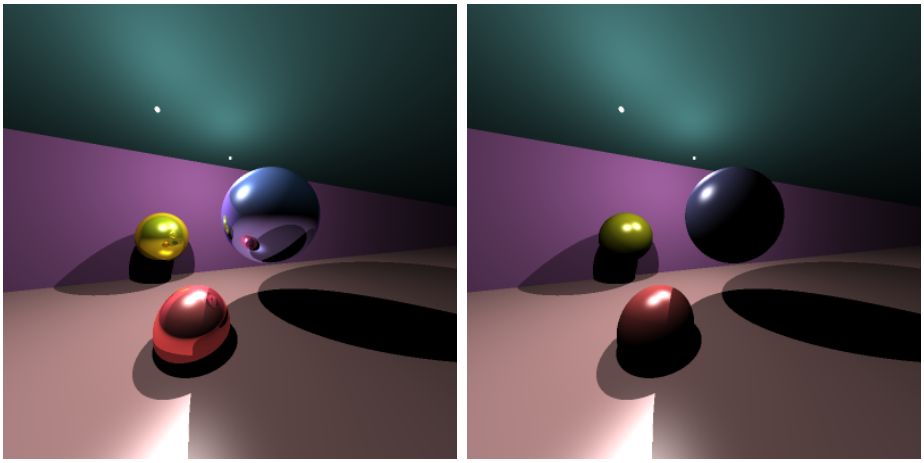
Rysunek 6.11: Wykresy CPU dla sceny złożonej z różnej ilości źródeł światła



Rysunek 6.12: Wykresy GPU dla sceny złożonej z różnej ilości źródeł światła

6.5 Test 5 - Scena tesująca odbicia promieni od obiektów

Na scenie w tym teście znajdują się trzy płaszczyzny oraz trzy sfery. Jednakże podczas testu brane pod uwagę są tylko parametry materiału sfer, które powodują odbicia, lub ich brak. Na rysunku 6.13 przedstawiona została dwa razy ta sama scena wygenerowana z włączonymi odbiciami promieni od obiektów sferycznych oraz z wyłączonymi odbiciami od tych właśnie obiektów.

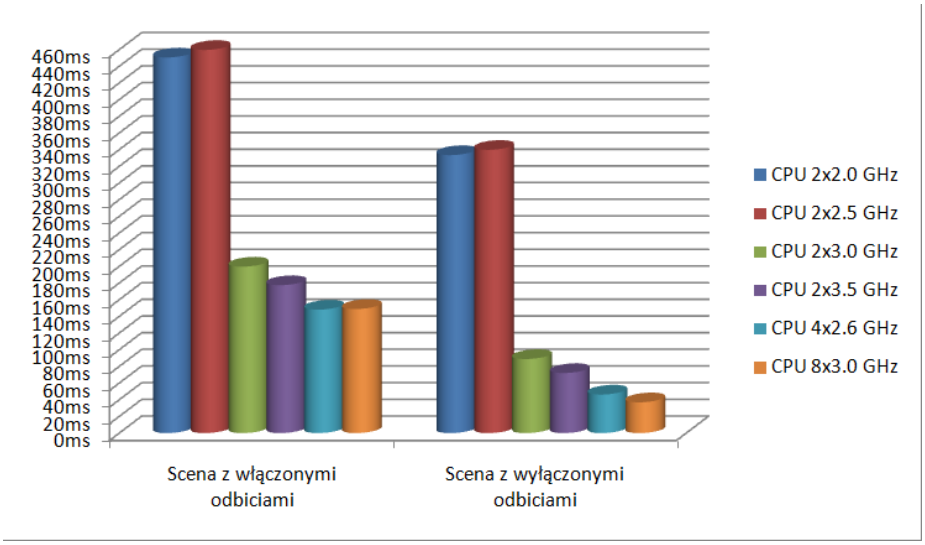


a) włączone odbicia promieni b) wyłączone odbicia promieni

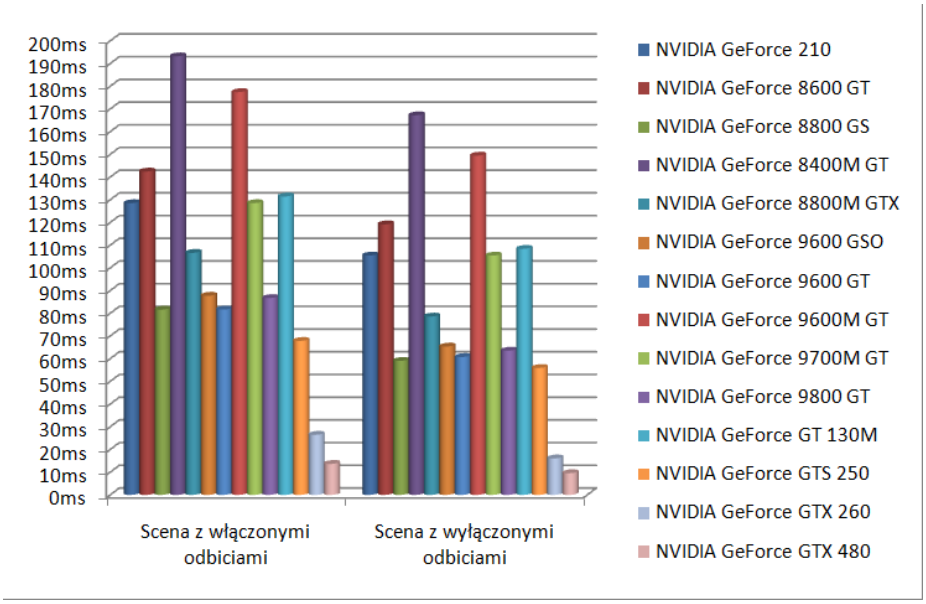
Rysunek 6.13: Obrazy wygenerowane w teście odbić promieni świetlnych

Na obu wykresach(procesorów oraz kart graficznych) widać różnice w czasach generowania scen. Różnice w czasach miedzy włączonymi, a wyłączonymi odbiciami nie są liniowe gdyż prymitywy na których te odbicia były testowane zajmują tylko cześć z wygenerowanego obrazu. Czas generowania procesora Intel Core I7 wynosi około 30ms z wyłączonymi odbiciami oraz około 140ms z włączonymi. Karta graficzna NVIDIA GTX480 radzi sobie znacznie lepiej: 9ms z wyłączonymi odbiciami oraz 13 ms z włączonymi.

WYKRESY:



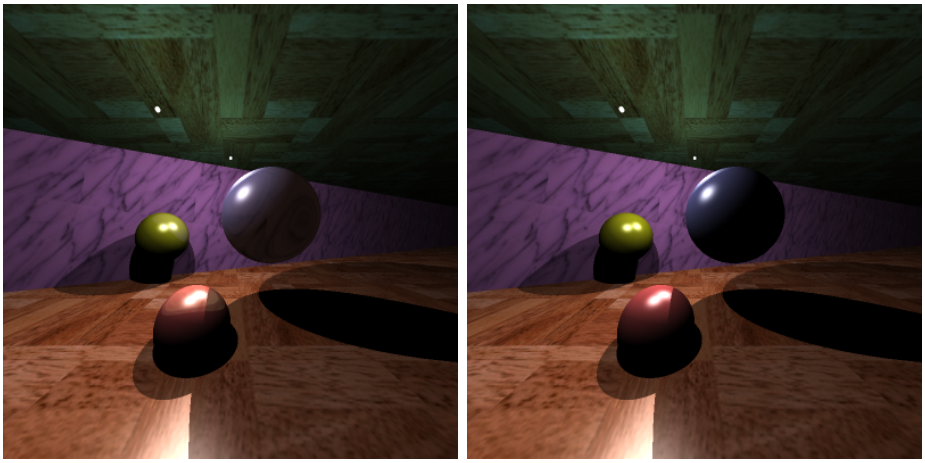
Rysunek 6.14: Wykresy CPU dla sceny z włączonymi/wyłączonymi odbiciami promieni od obiektów



Rysunek 6.15: Wykresy GPU dla sceny z włączonymi/wyłączonymi odbiciami promieni od obiektów

6.6 Test 6 - Scena testująca załamania promieni w obiektach

Test związany z załamaniami promieni w prymitywach wykonywany jest na tej samej scenie co powyższy test odbić promieni. Warunki początkowe są takie same jak i liczba prymitywów. Na rysunku 6.16 przedstawiona została dwa razy ta sama scena wygenerowana z włączonymi załamaniami promieni w obiektach sferycznych oraz z wyłączonymi załamaniami w tych właśnie obiektach.



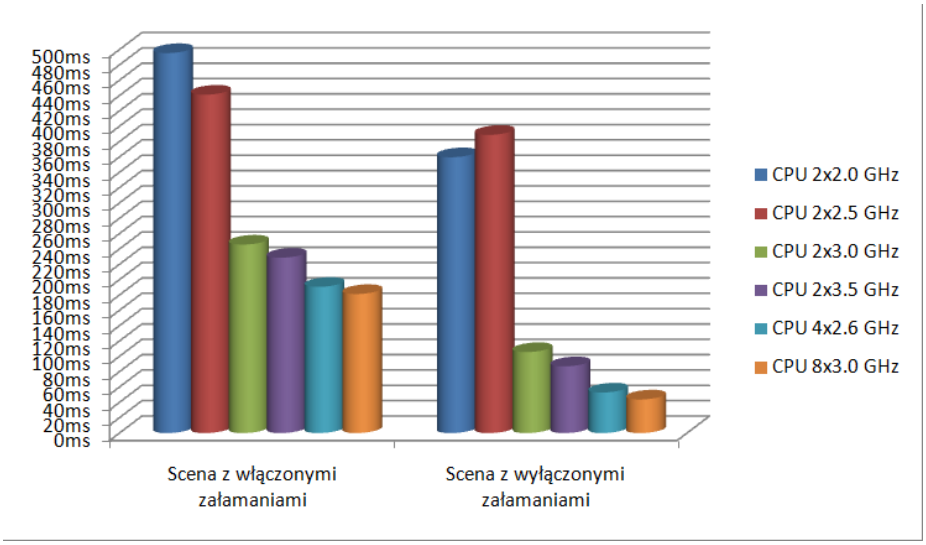
a) włączone załamania promieni

b) wyłączone załamania promieni

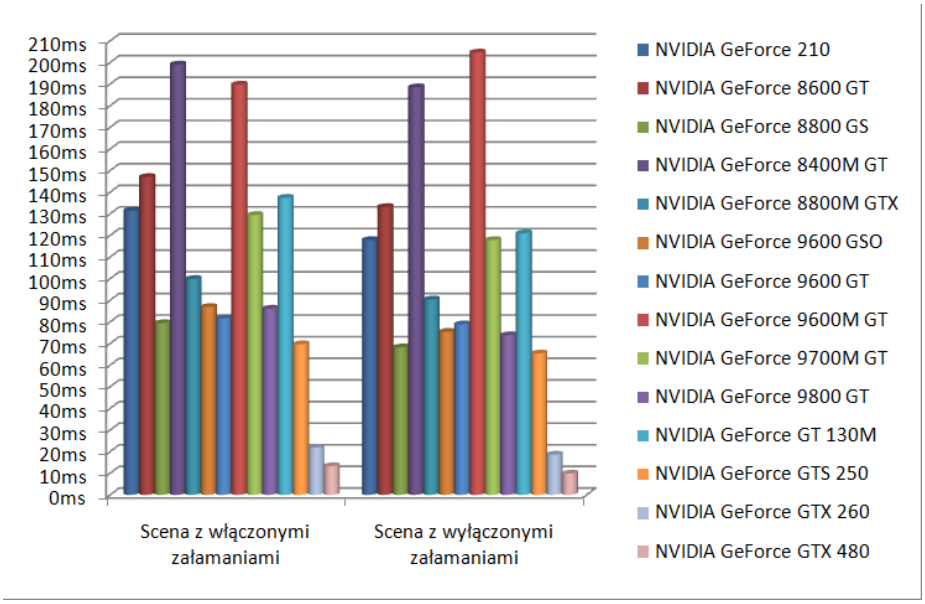
Rysunek 6.16: Obrazy wygenerowane w teście załamań promieni świetlnych

Na wykresach poniżej tak jak w teście związanym z odbiciami tak i teraz widoczne są różnice w czasach generowania scen na kożyść wariantu bez załamań promieni. Czas generowania procesora Intel Core I7 wynosi około 40ms z wyłączonymi załamaniami promieni oraz około 180 ms z włączonymi. Karta graficzna NVIDIA GTX480 radzi sobie znowu lepiej: 10 ms z wyłączonymi odbiciami oraz 14 ms z włączonymi. Porównując testy odbić promieni z testami załamań promieni widzimy, że załamania kosztują i procesor
i kartę graficzną więcej mocy obliczeniowej. Dzieje się tak dlatego iż więcej instrukcji potrzebnych jest do wykonania załamania, gdyż potrzeba dodatkowo obliczyć współczynnik absorbcji danego materiału przez który załamuje się promień.

WYKRESY:



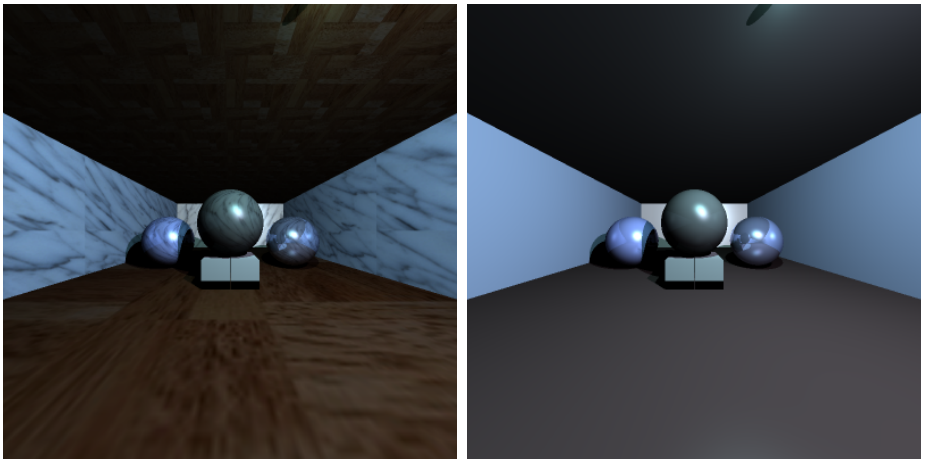
Rysunek 6.17: Wykresy CPU dla sceny z włączonymi/wyłączonymi załamaniami promieni w obiektach



Rysunek 6.18: Wykresy GPU dla sceny z włączonymi/wyłączonymi załamaniami promieni w obiektach

6.7 Test 7 - Scena tesująca teksturowanie obiektów

Scena testowa składa się z pięciu płaszczyzn, jednego pudełka(box) oraz 3 sfer. Testowane są dwa warianty. Pierwszy to scena w której każdy prymityw pokryty jest dodatkowo własną teksturą. Drugi wariant to ta sama scena ale już bez tekstur na prymitywach. Na rysunku 6.19 przedstawiona została dwa razy ta sama scena wygenerowana z włączonym teksturowaniem prymitywów oraz bez teksturowania.



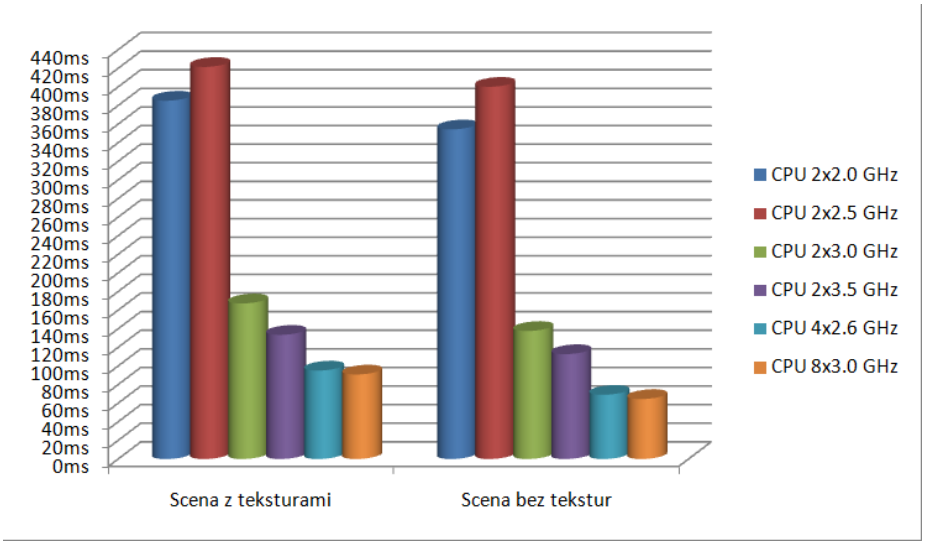
a) włączone teksturowanie obiektów

b) wyłączone teksturowanie obiektów

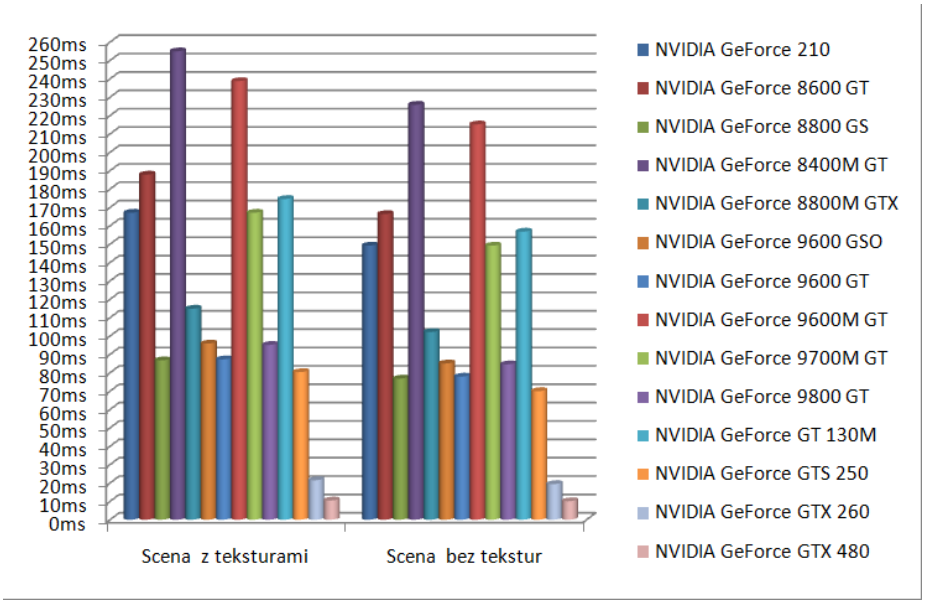
Rysunek 6.19: Obrazy wygenerowane w teście teksturowania obiektów

Po wykresach widać, że koszt teksturowania prymitywów sceny nie jest aż tak duży jak w przypadku odbić i załamań promieni. Czas generowania sceny na procesorze Intel Core I7 wynosi około 60ms z wyłączonymi teksturami oraz około 80ms z włączonymi. Na karcie graficznej NVIDIA GTX480 różnica w teksturowaniu jest prawie nie zauważalna: 8ms z wyłączonymi teksturami oraz 9 ms z włączonymi.

WYKRESY:



Rysunek 6.20: Wykresy CPU dla sceny z teksturami na obiektach



Rysunek 6.21: Wykresy GPU dla sceny bez tekstur na obiektach

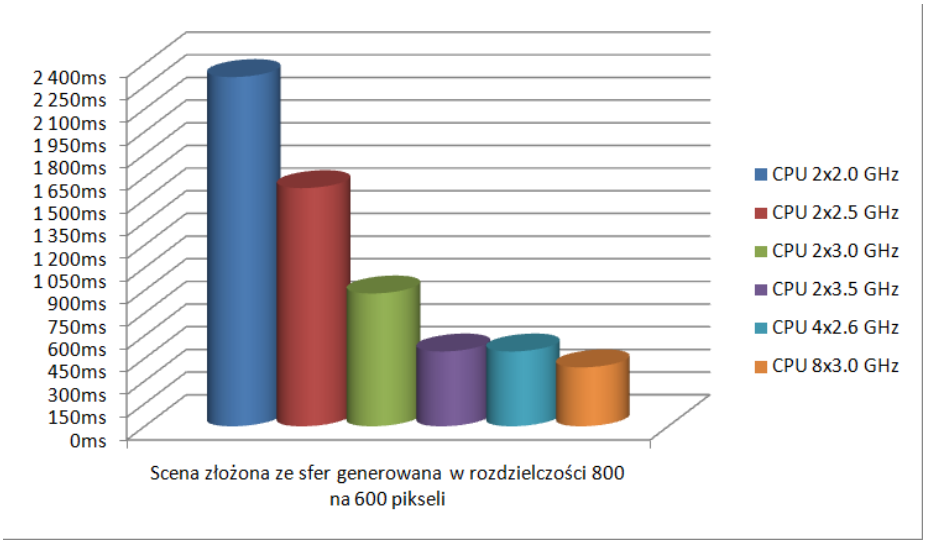
6.8 Test 8 - Różna rozdzielczość generowanych scen

W teście zbadana zostanie zależność szybkości generowania sceny w zależności od rozdzielczości wynikowego obrazu. Testowany jest tutaj spadek wydajności od liczby generowanych promieni. Przetestowane zostały dwa warianty scen:

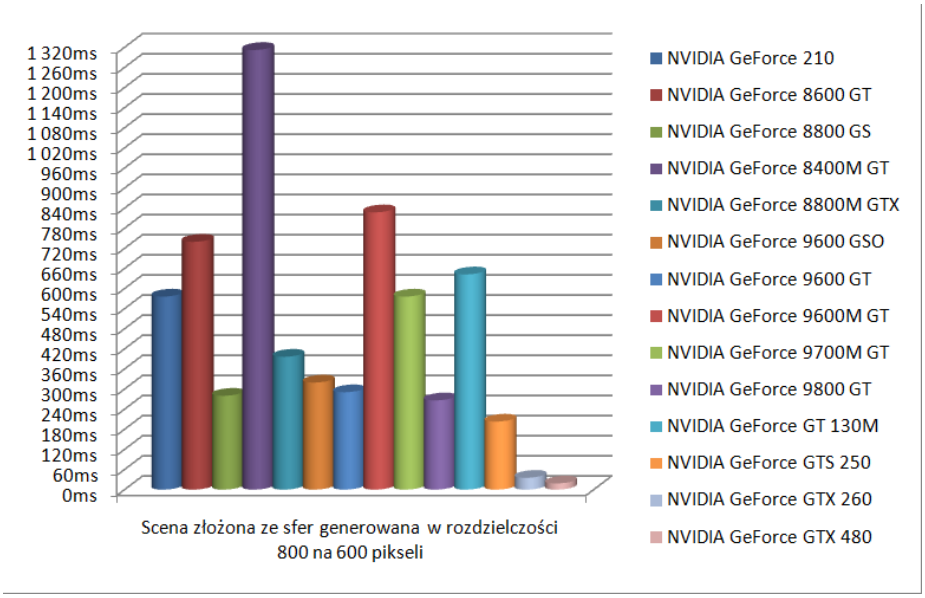
- Rozdzielczość obrazu 400 na 400 pikseli.
- Rozdzielczość obrazu 800 na 600 pikseli.

Generowaną sceną w tym teście była kolejny raz scena z testu pierwszego, złożona z samych sfer. Wygenerowany obraz przedstawiony jest na rysunku 6.1. Pierwszy wariant testu czyli generowanie w rozdzielczości 400 na 400 pikseli został przeprowadzony w teście pierwszym, wiec nie ma potrzeby generowania kolejnego wykresu dla tego wariantu. Poniżej przedstawione zostaną wykresy wydajności dla tej sceny wygenerowanej w rozdzielczości 800 na 600 pikseli.

WYKRESY:



Rysunek 6.22: Wykresy CPU dla sceny ze sferami w rozdzielczości 800 na 600 pikseli



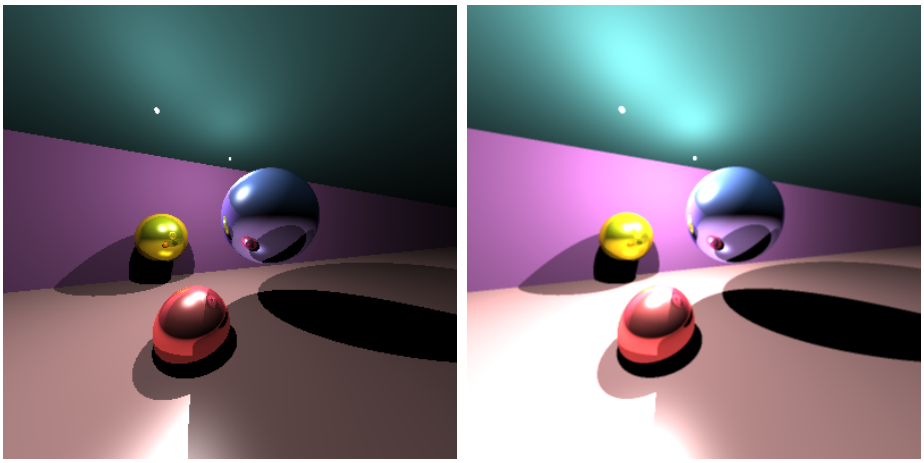
Rysunek 6.23: Wykresy GPU dla sceny ze sferami w rozdzielczości 800 na 600 pikseli

Jak można było się domyśleć, czasy generowanych scen wzrosły diametralnie. Spowodowane jest to większą liczbą promieni pierwotnych $800 * 600 = 480000$, które raytracer musi śledzić. Spoglądając na wykresy 6.22 oraz 6.23 można zobaczyć, że śledzenie promieni przeprowadzone na dość dobrych kartach CUDA, daje sobie radę z wiekszymi rozdzielczościami generowanego obrazu o wiele lepiej niż najlepsze procesory CPU.

6.9 Test 9 - Rożny stopień dokładności generowanych scen

W tym teście zbadana została szybkość wygenerowanego obrazu od jakości tego obrazu. Stosowany jest tu tak zwany super-sampling. Jest to wzmożone próbkowanie promieni na pojedyńczy piksel ekranu. Przetestowane zostały dwa warianty scen:

- Scena ze śledzeniem pojedyńczego promienia na piksel.
- Scena ze śledzeniem czterech promieni na pojedyńczy piksel.

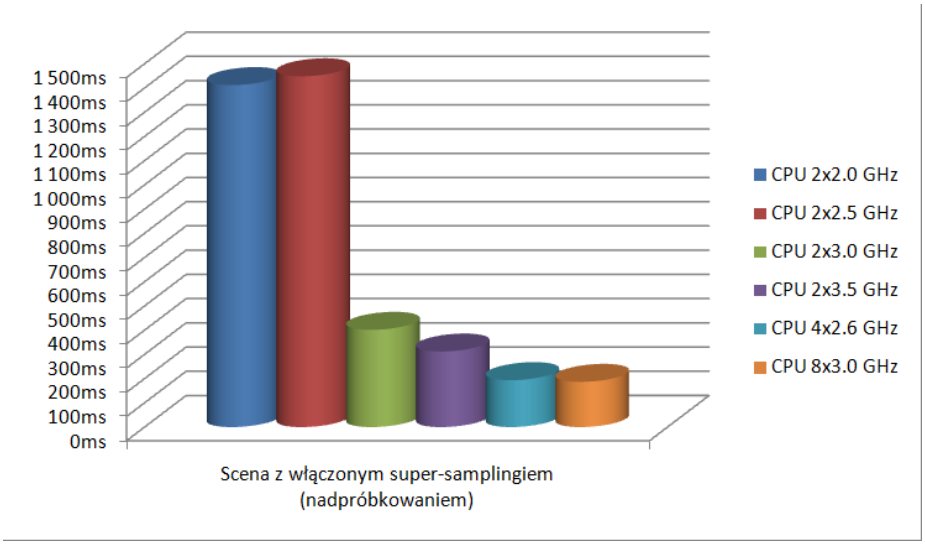


a) wyłączony super-sampling b) włączony super-sampling

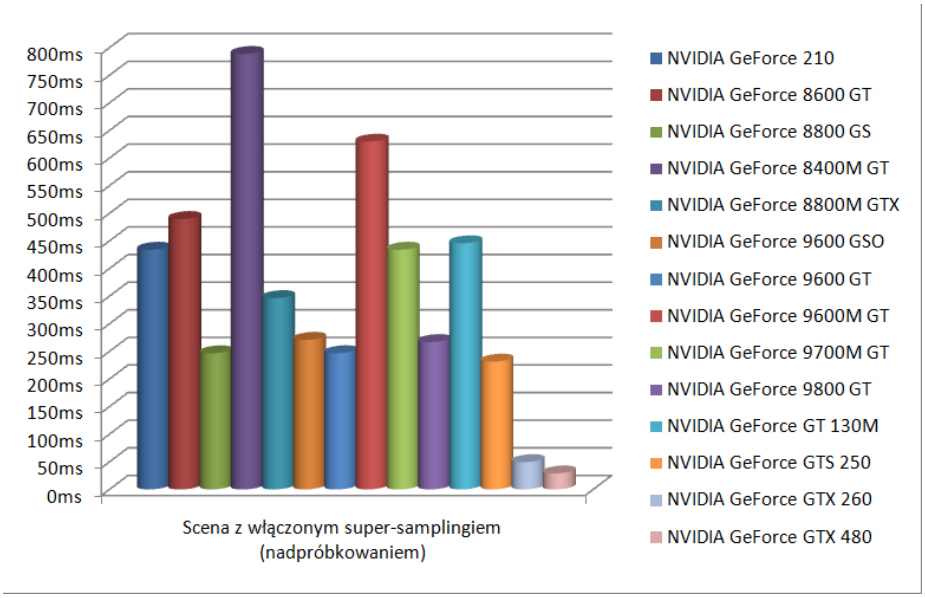
Rysunek 6.24: Obrazy wygenerowane w teście z włączonym/wyłączonym nadpróbkowaniem pikseli

Czasy generowanych scen wzrosły około 4 krotnie jak i na procesorze CPU tak i na kartach graficznych NVIDIA. Spowodowane jest to wzmożonym próbkowaniem promieni na pojedyńczy piksel. W tym teście zastosowany był czterokrotny super-sampling, dlatego czasy generowania sceny zwiększyły się około czterokrotnie. Wnioskujemy z tego, że im mniejszy współczynnik super-samplingu tym szybciej generowana jest dana scena. Tracimy jednak wtedy na jakości generowanego obrazu. Pojawia się tu zjawisko tzw. aliasingu, któremu przeciwdziała właśnie super-sampling.

WYKRESY:



Rysunek 6.25: Wykresy CPU dla sceny z włączoną opcją nadpróbkowania



Rysunek 6.26: Wykresy GPU dla sceny z włączoną opcją nadpróbkowania

6.10 Podsumowanie testów

W powyższych podrozdziałach przeprowadzonych zostało wiele testów badających poszczególne części metody śledzenia promieni. Z testów od razu widoczne jest, że udało uzyskać się znaczne przyśpieszenie w generowaniu na kartach graficznych NVIDIA w stosunku to zwykłych procesorów komputerowych CPU. Wiemy także iż dość kosztownymi operacjami w raytracingu są odbicia i załamania promieni świetlnych, co potwierdzają czasy przedstawione na wykresach. Dosyć szybkie jest natomiast teksturowanie poszczególnych prymitywów. Należy także pamiętać, że czas generowania sceny rośnie bardzo szybko gdy chcemy uzyskać dość dobrej jakości oraz rozdzielczości obrazy.

Podsumowując badania można wyznaczyć liczbowy wzrost przyśpieszenia uzyskanego przez technologie CUDA w stosunku do czasu generowania przez procesor CPU. Porównując najlepszą kartę graficzną na jakiej były przeprowadzone testy czyli GeForce GTX 480 do dwurdzeniowego procesora CPU o taktowaniu 2.0GHz, uzyskano przyśpieszenie rzędu około 125 razy.

Rozdział 7

Podsumowanie i wnioski

W pracy udało osiągnąć się zamierzone na początku cele. Została stworzona uniwersalna aplikacja do testowania wydajności śledzenia promieni na procesorach komputerowych CPU oraz na kartach graficzych NVIDIA obsługujących technologię CUDA. Uzyskano też znaczne przyspieszenie podczas generowania scen na kartach graficznych. Zasługą tej wydajności jest zrównoleglenie obliczeń w postaci wątków obliczeniowych realizowanych przez procesor graficzny. Jasno można tutaj powiedzieć, że w kartach graficznych tkwi wielka potęga i obliczenia na nich przeprowadzane przewyższają wydajnością zwykłe wielordzeniowe procesory komputerowe. Aktualnie, stale utrzymuje się rosnący trend na kolejne lepsze układy graficzne. Wydaje mi się, że już niedługo będziemy mogli we własnych domowych warunkach uruchamiać złożone, działające w czasie rzeczywistym aplikacje multimedialne, oparte o grafikę generowaną metodą raytracingu.

7.1 Napotkane problemy

Pracując nad niniejszą pracą, oraz mając do czynienia z nową technologią NVIDIA CUDA napotkałem kilka problemów. Warto zwrócić uwagę na trzy najważniejsze. Dwa natury softwarowej oraz jeden natury hardwarowej.

- Technologia NVIDIA CUDA nie wspiera rekurencji. W tym wypadku implementacja odbić i załamań światła nie była możliwa w standardowy sposób. Poradziłem sobie, tworząc sztuczną rekurencję w pętli, używając własnoręcznie stworzonego stosu.
- Testowanie(debugowanie) algorytmów pod technologię NVIDIA CUDA jest możliwe tylko i wyłącznie jeśli posiada się sprzęt z dwoma kartami graficznymi wspierającymi właśnie tą technologię. Nie posiadałem takiego sprzętu więc moje sprawdzanie poprawniści algorytmów

odbywało się metodą prób i błędów oraz porównywaniem wynikowych obrazów z wersją raytracera działającą na procesorze CPU.

- Ostanim ważnym problemem, któremu musiałem stawić czoło był problem natury sprzętowej. Sprzęt jaki miałem do dyspozycji posiadał fabrycznie wadliwy układ graficzny (GeForce 8400M). Podczas implementacji raytracingu układ ten przepalił się około 4 razy i tyle samo razy był wymieniany w serwisie na nowy.

7.2 Perspektywy kontynuacji

Przeprowadzone testy pokazały, że CUDA jest bardzo dobrym rozwiązaniem technologicznym dla realizacji metody śledzenia promieni. Biorąc pod uwagę przeprowadzone analizy oraz zdobyte doświadczenie, warto byłoby kontynuować prace związane z wykorzystaniem technologii CUDA do przyśpieszania śledzenia promieni w postaci następujących zadań:

- Ulepszenie metody wstecznego raytracingu adaptując algorytmy photon-mappingu oraz path-tracingu
- Próba opracowania własnego hybrydowego algorytmu śledzenia promieni w celu kolejnych przyśpieszeń generowania scen.
- Przeniesienie raytracingu z CUDA na inne platformy obliczeń strumieniowych: OpenCL, ATI Stream Computing.
- Modyfikacja aplikacji (benchmarku) by było możliwe uruchomienie jej na innych systemach operacyjnych niż rodzina Windows.

Bibliografia

[1] Gabriele Jost Barbara Chapman and Ruud van der Pas. Using OpenMP. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11387f.

[2] Jacco Bikker. Raytracing: Theory and Implementation. http://www.devmaster.net/articles/raytracing˙series/part1.php.

[3] Andrew D. Britton. Full CUDA Implementation Of GPGPU Recursive Ray-Tracing. http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1024&context=techmasters.

[4] Software Developer's Journal. Czyń CUDA (część 1) – architektura. http://software.com.pl/czyn-cuda-czesc-1-architektura/.

[5] NVIDIA. NVIDIA CUDA home page. http://www.nvidia.com/object/cuda˙home˙new.html.

[6] NVIDIA. NVIDIA Optimizing CUDA. http://www.speedup.ch/workshops/w38˙2009/pdf/speedup˙cuda˙optimizations.pdf.

[7] Eric Rollins. Real-Time Ray Tracing with GPGPU and Multi-Core. http://home.mindspring.com/˜eric˙rollins/ray/cuda.html.

[8] Wikipedia. ATI Technologies. http://en.wikipedia.org/wiki/ATI˙Technologies.

[9] Wikipedia. CUDA. http://en.wikipedia.org/wiki/CUDA.

[10] Wikipedia. Śledzenie promieni. http://pl.wikipedia.org/wiki/Ray˙tracing.

[11] Wikipedia. OpenCL. http://en.wikipedia.org/wiki/OpenCL.

Dodatek A

Terminologia stosowana w pracy

- CUDA - Technologia stworzona przez firmę NVIDIA w 2007 roku. Umożliwia równoległe obliczenia na mikroprocesorach karty graficznej.
- Benchmark - Aplikacja testowa, która profiluje wydajność i zbiera informacje.
- Warp - blok wątków przydzielony na multiprocesor.
- DirectX - technologia graficzna firmy Microsoft. Umożliwia wyświetlanie wysokiej jakości grafiki 2D/3D.
- Aliasing - Zdeformowany, o złej jakości obraz który powstaje podczas rastaryzacji, powodowany przez zbyt mała częstotliwość próbkowania na pojedyńczy piksel obrazu. Przeciwdziała się temu efektowi poprzez antyaliasing oraz w raytracingu poprzez super-sampling.
- Super-sampling - sposób na zwiększenie jakości generowanych scen. Polega na śledzeniu wielu promieni świetlnych na pojedyńczy piksel generowanego obrazu.

Spis rysunków

Spis tablic