Uniwersytet Mikołaja Kopernika
Wydział Matematyki i Informatyki

Daniel Nadolny
nr albumu: 312887

Praca inżynierska
na kierunku informatyka

Ray Tracing w czasie rzeczywistym

Opiekun pracy dyplomowej
doktor Jakub Narębski
Wydział Matematyki i Informatyki

Toruń 2026

Pracę przyjmuję i akceptuję

. . . . . .

data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowej

. . . . . .

data i podpis pracownika dziekanatu

Spis treści

Wstęp

W 2018 roku NVIDIA przedstawiła światu nową generację kart graficznych nazwanych RTX. Od tego roku każda kolejna seria począwszy od serii 20 do teraz (seria 50) jest wyposażona w tzw. RT Cores. Rdzenie RT są specjalnie stworzone do przyspieszania obliczeń związanych ze śledzeniem promieni (dalej będę używał nazwy ray tracing), w szczególności przy testowaniu przecięcia promienia z trójkątem i przechodzenia przez strukturę danych zwaną BVH (ang. bounding volume hierarchy). Odpowiednikiem RT Cores w kartach graficznych od AMD są ”Ray accelerators”. Ray tracing jest bardzo wymagającym algorytmem pod względem obliczeniowym. Dodanie powyższych rozwiązań sprzętowych do GPU pozwoliły programistom implementowanie ray tracingu w czasie rzeczywistym np. w grach, gdzie obecnie w jednej scenie może pojawić się kilka milionów trójkątów (dotychczas ray tracing wykorzystywany był głownie w filmach).

W ramach pracy inżynierskiej stworzony został silnik graficzny przedstawiający ray tracing w czasie rzeczywistym, napisany jest w języku C++, wykorzystując bibliotekę DirectX 11 i win32. Interfejs użytkownika stworzony został za pomocą biblioteki ImGui.

Pierwszy rozdział tej pracy będzie poświęcony przedstawieniu podstaw ray tracingu, od opisania idei algorytmu do wyprowadzenia dwóch podstawowych procedur badania przecięć promienia z obiektami w scenie (promień-sfera i promień-trójkąt). W rozdziale drugim poruszone będzie zagadnienie optymalizacji silnika używając struktury danych BVH. W następnym rozdziale zostanie opisane zagadnienie materiałów. Materiały są jednym z najważniejszych tematów w ray tracingu, jak i ogólnie w grafice komputerowej, jeśli chodzi o aspekty wizualne. W czwartym rozdziale

opisany będzie stworzony silnik i implementacje przedstawionych wcześniej algorytmów. W końcowej części pracy przedstawione zostaną wyniki testów wydajnościowych programu. Testy były przeprowadzone przed optymalizacją silnika i po optymalizacji, aby wykazać różnice wydajności.

Rozdział 1

Podstawy Ray Tracingu

1.1 Definicje i oznaczenia

W dalszej części będę posługiwać się takimi oznaczeniami:

- a - wartość skalarna.
- $\boldsymbol{C}$ - punkt w przestrzeni trójwymiarowej.
- $\boldsymbol{v}$ - wektor, w większości przypadków $\boldsymbol{v} = (x, y, z)$.
- $\boldsymbol{n}$ - wektor normalny. Wektor prostopadły do danej powierzchni.
- $\boldsymbol{a} \cdot \boldsymbol{b}$ - iloczyn skalarny.
- $\boldsymbol{a} \times \boldsymbol{b}$ - iloczyn wektorowy.
- Promień - promień (ang. ray) jest podstawową strukturą w ray tracingu. Składa się on z punktu początkowego (ang. origin) i kierunku (ang. direction). Oba elementy zdefiniowane są jako wektory trójwymiarowe, z czego kierunek jest wektorem znormalizowanym (długość równa 1).

1.2 Algorytm ray tracingu

Algorytm ray tracingu znany jest już od 1979 roku, kiedy John Turner Whitted opublikował artykuł ”An improved illumination model for shaded display” opisujący rekurencyjny ray tracing [PRZYPIS].

Ideą algorytmu jest naśladowanie światła. W opisie zachowania się światła i jego oddziaływania z materią korzysta się z teorii falowej i optyki geometrycznej. W rzeczywistości światło porusza się po liniach prostych, od źródła np. Słońca. Ray tracing działa odwrotnie, tzn. źródłem promieni jest ”oko” kamery i od niego wychodzi światło w generowaną scenę, ponieważ jak w rzeczywistości mózg człowieka ”renderuje” obraz korzystając tylko z tych promieni, które padają na siatkówkę w oku [PRZYPIS]. Gdy wystrzelony z kamery promień uderza w jakiś obiekt punkt przecięcia staje się punktem początkowym kolejnego promienia (rekurencyjna natura algorytmu). [TUTAJ WSTAWIĆ OBRAZEK PRZEDSTAWIAJĄCY ALGORYTM].

Należy się teraz zastanowić, w jaki sposób obiekty w naturze ”dostają” swój kolor. Innymi słowy - dlaczego pomidor jest czerwony? Obserwując eksperyment przedstawiający rozszczepienie światła pryzmatem, lub tęczę, możemy zauważyć, że białe światło rozszczepia się na kilka barw (w uproszczeniu). Dzieje się tak, ponieważ barwy światła mają różne długości fal. Zakres długości fal dla światła widzialnego przez człowieka wynosi: 380-780 nanometrów. Gdy promienie ze źródła uderzą w jakiś obiekt, (np. w pomidor) to materiał obiektu wchłonie w siebie pewną część światła o danej długości, a odbije resztę. Przykładowy pomidor odbije głównie barwę czerwoną, czyli fale o długości w zakresie 620-780 nm, następnie mózg interpretuje daną długość fali na odpowiedni kolor. W ray tracingu procedura jest podobna, jedynym dodatkiem do światła w scenie jest jego źródło, reszta oświetlenia pochodzi od promieni odbitych od obiektów. [DODAJ PRZYPIS I OBRAZEK ZAKRESU]

Symulowanie światła pozwala programistom na uzyskanie szczegółowych i poprawnych fizycznie efektów takich jak: odbicie, refrakcja itd. Wcześniej, przed erą ray tracingu w czasie rzeczywistym, programiści musieli korzystać z różnych sztuczek aby zaimplementować te efekty, dla

przykładu odbicia tworzone były poprzez zrenderowanie danego obiektu drugi raz ale odwrotnie w np. kałurzy, lustrze. Korzystając z ray tracingu efekt odbicia jest dużo prostszy w implementacji.

Minusem algorytmu jest jego złożoność obliczeniowa. Dla przykładu, bez optymalizacji BVH mając model złożony z 100 000 trójkątów, w rozdzielczości 2560x1440, mając ustawione 2 próbki na piksel i 5 odbiciach, program musiałby dla każdego piksela wykonać (w tej rozdzielczości mamy 3686400 pikseli) 1 000 000 testów.

1.3 Badanie przecięcia promienia ze sferą

Test przecięcia promienia ze sferą jest jednym z najprostszych algorytmów w tej tematyce i idealnie nadaje się do przedstawienie procesu wyprowadzania takich algorytmów.

Należy zacząć od zapisania równania sfery o środku w punkcie $\boldsymbol{C} = (c_x, c_y, c_z)$.

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2 \tag{1.1}$$

Punkt $\boldsymbol{P} = (p_x, p_y, p_z)$ jest punktem leżącym na sferze. Można zapisać wektor o długości r ze środka sfery do tego punktu zapisując:

$$(\boldsymbol{P} - \boldsymbol{C})$$

Teraz równanie sfery można zapisać tak:

$$(\boldsymbol{P} - \boldsymbol{C}) \cdot (\boldsymbol{P} - \boldsymbol{C}) = r^2 \tag{1.2}$$

Każdy punkt $\boldsymbol{P}$ spełniający to równanie leży na sferze.

Szukaną wartością w tym badaniu jest zmienna t, na podstawie której można obliczyć współrzędne punktu P. Równanie opisujące promień o punkcie początkowym O i kierunku D:

$$\boldsymbol{P}(t) = \boldsymbol{O} + t\boldsymbol{D} \tag{1.3}$$

Wstawiając (1.3) do (1.2):

$$(\boldsymbol{O} + t\boldsymbol{D} - \boldsymbol{C}) \cdot (\boldsymbol{O} + t\boldsymbol{D} - \boldsymbol{C}) = r^2 \tag{1.4}$$

Nie ma potrzeby rozpisywania całego iloczynu skalarnego, zamiast tego można zapisać:

$$(\boldsymbol{D} \cdot \boldsymbol{D})t^2 + 2t\boldsymbol{D} \cdot (\boldsymbol{O} - \boldsymbol{C}) + (\boldsymbol{O} - \boldsymbol{C}) \cdot (\boldsymbol{O} - \boldsymbol{C}) - r^2 = 0 \tag{1.5}$$

Jest to równanie kwadratowe o współczynnikach:

$$a = \boldsymbol{D} \cdot \boldsymbol{D}, b = 2\boldsymbol{D} \cdot (\boldsymbol{O} - \boldsymbol{C}), c = (\boldsymbol{O} - \boldsymbol{C}) \cdot (\boldsymbol{O} - \boldsymbol{C}) - r^2$$

Wzór na paramter t:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Równanie (1.5) może mieć rozwiązania dodatnie jak i ujemne. Silnik, aby poprawnie generować obrazy dla wielu sfer w scenie, potrzebuje wartości najbliższej i dodatniej (wartość ujemna świadczy o tym, że sfera jest za kamerą). W ramach oznaczenia przypadku braku rozwiązania równania (1.5) z funkcji zwracana jest wartość -1.0.

1.4 Badanie przecięcia promienia z trójkątem

Autorami przedstawionego poniżej algorytmu są Tomas Möller i Ben Trumbore [1]. W odróżnieniu od innych popularnych algorytmów szukających punkt przecięcia promienia z trójkątem, ten nie oblicza równania płaszczyzny wyznaczanej przez trójkąt, tylko opiera się na samych wierzchołkach trójkąta. Istnieją szybsze algorytmy np. algorytm Douga Baldwina i Michaela Webera [2], ale wymagają one dodatkowych informacji razem z trójkątami.

Definicja 1.4.1. *Współrzędne barycentryczne na trójkącie o wierzchołkach w punktach* P_0, P_1, P_2 *to trójka liczb* $(w, u, v) \in \mathbb{R}$*, dzięki którym możemy*

przedstawić każdy punkt na trójkącie w formie:

$$f(u,v) = (1-u-v)\boldsymbol{P_0} + u\boldsymbol{P_1} + v\boldsymbol{P_2}.$$

Liczby (w,u,v) *spełniają następujące warunki:*

$$w+u+v=1, \quad w \geq 0, \quad u \geq 0, \quad v \geq 0.$$

[DODAC RYSUNEK Z REAL TIME RENDERING]

Znalezienie punktu przecięcia sprowadza się do rozwiązania układu równań:

$$\boldsymbol{O} + t\boldsymbol{D} = (1-u-v)\boldsymbol{P_0} + u\boldsymbol{P_1} + v\boldsymbol{P_2} \tag{1.6}$$

Gdzie $\boldsymbol{O} = (o_x, o_y, o_z)$ to punkt początkowy promienia, a $\boldsymbol{D} = (d_x, d_y, d_z)$ to kierunek.

Po przepisanu na postać macierzową:

$$\begin{bmatrix} -\boldsymbol{D} & \boldsymbol{P_1} - \boldsymbol{P_0} & \boldsymbol{P_2} - \boldsymbol{P_0} \end{bmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \boldsymbol{O} - \boldsymbol{P_0} \tag{1.7}$$

Pomocnicze oznaczenia: $\boldsymbol{e_1} = \boldsymbol{P_1} - \boldsymbol{P_0}, \boldsymbol{e_2} = \boldsymbol{P_2} - \boldsymbol{P_0}, \boldsymbol{s} = \boldsymbol{O} - \boldsymbol{P_0}$
Używając oznaczeń układ równań ma postać:

$$\begin{aligned} -d_x t + e_{1x} u + e_{2x} v &= s_x \\ -d_y t + e_{1y} u + e_{2y} v &= s_y \\ -d_z t + e_{1z} u + e_{2z} v &= s_z \end{aligned} \tag{1.8}$$

Wtedy macierz główna układu ma postać:

$$M = \begin{bmatrix} -d_x & e_{1x} & e_{2x} \\ -d_y & e_{1y} & e_{2y} \\ -d_z & e_{1z} & e_{2z} \end{bmatrix} \tag{1.9}$$

Do rozwiązania układu równań można wykorzystać metodę Crame-

ra:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\det(-\boldsymbol{D}, \boldsymbol{e_1}, \boldsymbol{e_2})} \begin{pmatrix} \det(\boldsymbol{s}, \boldsymbol{e_1}, \boldsymbol{e_2}) \\ \det(-\boldsymbol{D}, \boldsymbol{s}, \boldsymbol{e_2}) \\ \det(-\boldsymbol{D}, \boldsymbol{e_1}, \boldsymbol{s}) \end{pmatrix} \tag{1.10}$$

Z własności iloczynu mieszanego wiadomo, że: $(\boldsymbol{a} \times \boldsymbol{b}) \cdot \boldsymbol{c} = \det(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ Wtedy równanie można zapisać tak (zmieniając kolumny macierzy wyznacznik zmienia znak na przeciwny):

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(\boldsymbol{B} \times \boldsymbol{e_2}) \cdot \boldsymbol{e_1}} \begin{pmatrix} (\boldsymbol{s} \times \boldsymbol{e_1}) \cdot \boldsymbol{e_2} \\ (\boldsymbol{D} \times \boldsymbol{e_2}) \cdot \boldsymbol{s} \\ (\boldsymbol{s} \times \boldsymbol{e_1}) \cdot \boldsymbol{D} \end{pmatrix} \tag{1.11}$$

Wprowadzając kolejne oznaczenia, dla uproszczenia: $\boldsymbol{q} = \boldsymbol{D} \times \boldsymbol{e_2}$ $\boldsymbol{r} = \boldsymbol{s} \times \boldsymbol{e_1}$ Otrzymane równanie:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\boldsymbol{q} \cdot \boldsymbol{e_1}} \begin{pmatrix} \boldsymbol{r} \cdot \boldsymbol{e_2} \\ \boldsymbol{q} \cdot \boldsymbol{s} \\ \boldsymbol{r} \cdot \boldsymbol{D} \end{pmatrix} \tag{1.12}$$

W implementacji funkcja zwraca parametr t, dzięki niemu znany jest punkt przecięcia.

Rozdział 2

Optymalizacja

2.1 Bounding Volume Hierarchy

W stworzonym silniku graficznym w danej chwili jest kilka stałych wartości: rozdzielczość (np. 1920x1080 pikseli), liczba odbić promienia, liczba promieni na piksel i liczba trójkątów w scenie. Jedyną zmienną, którą można optymalizować jest liczba testów przecięcia promień-trójkąt (dla uproszczenia algorytmu sfery są pominięte). Do takiej optymalizacji można posłużyć się tzw. drzewami BVH - Bounding Volume Hierarchy.

Bibliografia

[1] Tomas Möller i Ben Trumbore. “Fast Minimum Storage Ray-Triangle Intersection”. W: *Journal of Graphics Tools* (1997).

[2] Doug Baldwin i Michael Weber. “Fast Ray-Triangle Intersections by Coordinate Transformation”. W: *Journal of Computer Graphics Techniques* (2016).