

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

Potwierdzam złożenie pracy
dyplomowej

.....

data i podpis opiekuna pracy

.....

data i podpis pracownika dziekanatu

Spis treści

Wstęp	3
1 Podstawy Ray Tracingu	5
1.1 Definicje i oznaczenia	5
1.2 Algorytm ray tracingu	6
1.3 Badanie przecięcia promienia ze sferą	7
1.4 Badanie przecięcia promienia z trójkątem	8
2 Optymalizacja	11
2.1 Bounding Volume Hierarchy	11
2.2 Zasada działania BVH	11
2.3 Przechodzenie przez drzewo BVH	12
2.4 Surface Area Heuristic	14
3 Materiały i modele oświetlenia	15
3.1 Materiały w grafice komputerowej	15
3.2 Model oświetlenia lokalnego	16
3.3 Model oświetlenia globalnego	17
4 Prezentacja silnika	19
4.1 Biblioteki	19
4.2 Opis potoku graficznego	20
4.3 Implementacja algorytmów	20
Podsumowanie	21
4.4 Testy Wydajności	21

4.5 Zakończenie	21
Bibliografia	22

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 głównie 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 obiekta 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 i modeli oświetlenia wykorzystywanych w grafice komputerowej. Oba zagadnienia mają największy wpływ na aspekty wizualne. W czwartym rozdziale opisany będą

dzie 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.
- C - punkt w przestrzeni trójwymiarowej.
- v - wektor, w większości przypadków $v = (x, y, z)$.
- n - wektor normalny. Wektor prostopadły do danej powierzchni.
- $a \cdot b$ - iloczyn skalarny.
- $a \times b$ - 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 zauważyc, 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 rozdzielcości 2560x1440, mając ustawione 2 próbki na piksel i 5 odbiciach, program musiałby dla każdego piksela wykonać (w tej rozdzielcości 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 $C = (c_x, c_y, c_z)$.

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

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

$$(P - C)$$

Teraz równanie sfery można zapisać tak:

$$(P - C) \cdot (P - C) = r^2 \quad (1.2)$$

Każdy punkt 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 :

$$P(t) = O + tD \quad (1.3)$$

Wstawiając (1.3) do (1.2):

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

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

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

Jest to równanie kwadratowe o współczynnikach:

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

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:

$$f(u, v) = (1 - u - v)\mathbf{P}_0 + u\mathbf{P}_1 + v\mathbf{P}_2.$$

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 rozwiązyania układu równań:

$$\mathbf{O} + t\mathbf{D} = (1 - u - v)\mathbf{P}_0 + u\mathbf{P}_1 + v\mathbf{P}_2 \quad (1.6)$$

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

Po przepisanu na postać macierzową:

$$\begin{bmatrix} -\mathbf{D} & \mathbf{P}_1 - \mathbf{P}_0 & \mathbf{P}_2 - \mathbf{P}_0 \end{bmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \mathbf{O} - \mathbf{P}_0 \quad (1.7)$$

Pomocnicze oznaczenia: $\mathbf{e}_1 = \mathbf{P}_1 - \mathbf{P}_0$, $\mathbf{e}_2 = \mathbf{P}_2 - \mathbf{P}_0$, $\mathbf{s} = \mathbf{O} - \mathbf{P}_0$

Używając oznaczeń układ równań ma postać:

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

Wtedy macierz główna układu ma postać:

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

Do rozwiązyania układu równań można wykorzystać metodę Crame-

ra:

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

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

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

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

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

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: rozdzielcość (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.

Idea algorytmu jest prosta - pomijać te trójkąty których promień na pewno nie przetnie. Bez optymalizacji, aby sprawdzić czy promień przecina się z jakimś trójkątem, trzeba przejść przez całą tablicę trójkątów modelu i każdy przetestować, w przypadku użycia BVH większość pomijamy badając tylko te najbliższe punktowi przecięcia.

2.2 Zasada działania BVH

W BVH dzielimy dany model na prostopadłościany zwane "bounding box". Proces podziału rozpoczyna się od korzenia (ang. root) jako prostopadłościan okalający cały obiekt (lub całą scenę), następnie rekurencyjnie dzielimy model na coraz to mniejsze prostopadłościany, aż do zadanej granicy np. 2 trójkątów w jednym boxie. BVH ma strukturę drzewa binarnego

(tak jest w stworzonym silniku), gdzie w wierzchołkach mieszą się kolejne prostopadłościany z podziału, a w liściach znajdują się trójkąty [3].
[OBRAZEK POKAZUJĄCY BVH]

W strukturze BVH używa się różnych typów "bounding box", w ray tracingu popularnym wyborem są "axis-aligned bounding box" (AABB). AABB tworzone są poprzez wyznaczenie dwóch punktów: prawego górnego punktu i lewego dolnego punktu. Maksymalny punkt AABB wyznaczany jest poprzez wyszukanie największych wartości na każdej osi ze zbioru trójkątów, każdy trójkąt ma 3 punkty. Punkt minimalny wyznaczany jest analogicznie poprzez szukanie najmniejszej wartości.

Można zapisać to tak:

Algorithm 1 Wyznaczanie AABB

```
Triangles ← [ (v1, v2, v3), ... ]  
for each triangle t in Triangles do  
    pmin ← ( min(Pmin, t.v1) )  
    pmin ← ( min(Pmin, t.v2) )  
    pmin ← ( min(Pmin, t.v3) )  
    pmax ← ( max(Pmax, t.v1) )  
    pmax ← ( max(Pmax, t.v2) )  
    pmax ← ( max(Pmax, t.v3) )  
end for
```

Aby stworzyć strukturę drzewa należy teraz podjąć decyzję w jaki sposób dzielić model (AABB boxy). Dla prostoty algorytmu dzielić można wzduż najdłuższej osi [3]. Następnie dzielimy trójkąty na dwie grupy poprzez sprawdzanie czy jego centroid¹ leży w jednym boxie, czy w drugim.

2.3 Przechodzenie przez drzewo BVH

Najważniejszym punktem w procedurze przechodzenia przez drzewo BVH jest testowanie przecięcia promienia z AABB. W tym przypadku test nie musi zwracać dokładnego punktu przecięcia, ale samą informację,

¹Centroid to punkt przecięcia median trójkąta (mediana to odcinek łączący wierzchołek ze środkiem przeciwnego boku) [4]

czy do przecięcia doszło lub (jeśli potrzebna) odległość od punktu początkowego promienia do przecięcia. Jedną z metod jest tzw. slab test.

Slab test polega na sprawdzaniu czy promień przecina wszystkie płaszczyzny wyznaczane przez osie x, y, z. Dla każdej osi należy obliczyć, AABB box zdefinowany jest jako dwa punkty $P_{min} = (x_{min}, y_{min}, z_{min})$, $P_{max} = (x_{max}, y_{max}, z_{max})$, parametr t , który można użyć do obliczenia punktu przecięcia używając równanie promienia (1.3):

$$t_1 = \frac{P_{min} - O}{D} \quad (2.1)$$

$$t_2 = \frac{P_{max} - O}{D} \quad (2.2)$$

Następnym krokiem jest wybranie wartości największej i najmniejszej:

$$t_{min} = \min(t_1, t_2) \quad (2.3)$$

$$t_{max} = \max(t_1, t_2) \quad (2.4)$$

Następnie należy obliczyć wartości:

$$t_{near} = \max(t_{min_x}, t_{min_y}, t_{min_z}) \quad (2.5)$$

$$t_{far} = \min(t_{max_x}, t_{max_y}, t_{max_z}) \quad (2.6)$$

Jeśli $t_{near} \leq t_{far}$ to promień jest na części wspólnej wyznaczonej przez płaszczyzny, czyli przecina AABB box, jeśli $t_{near} \geq t_{far}$ lub $t_{far} < 0$ promień nie trafił w box.

Jeśli promień przetnie box AABB, wtedy trzeba sprawdzić czy wierzchołek ze struktury BVH jest liściem. Jeśli tak (liść ma w sobie trójkąty), to uruchamia się test przecięcia promień-trójkąt. W przypadku wierzchołków, które nie są liśćmi, funkcja wchodzi w rekurencję dla lewego potomka i prawnego potomka.

2.4 Surface Area Heuristic

Istnieją też inne metody na budowę BVH, jedną z tych metod jest SAH "Surface Area Heuristic". W tej metodzie przy każdym podziale oblicza się koszt przeprowadzania testów przecięcia promienia z danym obiektem. Wzór na koszt [5]:

$$c(A, B) = t_{trav} + p_A \sum_{i=1}^{N_A} t_{isect}(a_i) + p_B \sum_{i=1}^{N_B} t_{isect}(b_i) \quad (2.7)$$

- t_{trav} - czas potrzebny na ustalenie przez które dzieci przechodzi promień.
- p_A - to prawdopodobieństwo, że przez wierzchołek A przejdzie promień (odpowiednio p_B).
- $\sum_{i=1}^{N_A} t_{isect}(a_i)$ - Czas potrzebny na przeprowadzenie testu przecięcia promienia z obiektem dla każdego obiektu w danym AABB.

$$p_A = \frac{s_a}{s_b} \quad (2.8)$$

- s_a - powierzchnia danego AABB ($s_a > s_b$)

Rozdział 3

Materiały i modele oświetlenia

3.1 Materiały w grafice komputerowej

W rzeczywistości zachowanie światła padającego na dany obiekt będzie zależało od materiału z którego ten obiekt jest stworzony np. światło padające na metal będzie odbijało się inaczej od światła padającego na plastik. W grafice komputerowej chcemy symulować właściwości różnych materiałów za pomocą tzw. modeli oświetlenia.

Model oświetlenia, to matematyczny opis zachowania światła w scenie. Określa w jaki sposób obiekty powinny być renderowane tj. w jaki sposób światło odbija się od powierzchni obiektów. Modele oświetlenia możemy podzielić na:

- Modele oświetlenia lokalnego (local illumination) - kolor danej powierzchni zależy tylko od materiału, z którego powierzchnia jest zrobiona i źródła światła.
- Modele oświetlenia globalnego (global illumination) - kolor danej powierzchni zależy od materiału, z którego powierzchnia jest zrobiona, źródła światła i światła odbitego od innych obiektów. [3]

3.2 Model oświetlenia lokalnego

[TU JAKIEŚ SKRINY ZE SWOJEGO SILNIKA]

Jednym z najpopularniejszych modeli oświetlenia lokalnego jest model Phonga (ang. Phong reflection model). Model Phonga składa się z trzech komponentów:

- Ambient - światło otoczenia. W tym algorytmie wpływ innych obiektów na jasność i kolor danego modelu (global illumination) jest symulowane poprzez dodanie pewnej stałej wartości do jasności i koloru obiektu.
- Diffuse - światło rozproszone. Najważniejszy komponent oświetlenia, światło padające na powierzchnię odbija się we wszystkich kierunkach równomiernie [5]
- Specular - światło zwierciadlane, odblask.

Wzór przedstawiający ostateczne oświetlenie danego punktu [6]:

$$I = I_a k_a + k_d I_i (\mathbf{n} \cdot \mathbf{l}) + k_s I_i (\mathbf{r} \cdot \mathbf{v})^s \quad (3.1)$$

Gdzie:

- I_a - natężenie światła otoczenia.
- I_i - natężenie światła padającego.
- k_a - współczynnik odbicia światła otoczenia. Własność materiału.
- k_d - współczynnik odbicia rozproszonego. Określa jak bardzo obiekt jest matowy. Własność materiału.
- k_s - współczynnik odbicia zwierciadlanego. Określa jak bardzo obiekt jest błyszczący. Własność materiału.
- \mathbf{n} - wektor normalny do powierzchni.

- \mathbf{l} - znormalizowany wektor kierunku do źródła światła od badanego punktu.
- \mathbf{r} - wektor kierunku odbicia światła obliczany według wzoru:

$$\mathbf{r} = 2.0(\mathbf{l} \cdot \mathbf{n}) \cdot \mathbf{n} - \mathbf{l}$$

- \mathbf{v} - znormalizowany wektor kierunku do kamery.
- s - współczynnik przedstawiający rozmiar odblasku.

W przypadku wielu źródeł światła w scenie wzór przyjmuje postać:

$$I = I_a k_a + \sum_i^{lights} k_d I_i (\mathbf{n} \cdot \mathbf{l}_i) + \sum_i^{lights} k_s I_i (\mathbf{r}_i \cdot \mathbf{v})^s \quad (3.2)$$

Ulepszeniem tego modelu jest Model Blinna-Phonga w którym oblicza się tzw. halfway vector $\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{\|\mathbf{l}+\mathbf{v}\|}$ i zamienia się $\mathbf{r} \cdot \mathbf{v}$ na $\mathbf{n} \cdot \mathbf{h}$

3.3 Model oświetlenia globalnego

[WPROWADZIĆ RADIOMETRIĘ?] [TU JAKIEŚ SKRINY ZE SWOJEGO SILNIKA]

W modelach oświetlenia globalnego (ang. *global illumination*) do obliczenia natężenia światła (koloru piksela) w danym punkcie bierze się pod uwagę źródło światła jak i otoczenie - światło odbite od innych powierzchni. Efekty takie jak: realistyczne cienie, odbicia i obiekty transparentne są implementowane za pomocą algorytmów globalnego oświetlenia.

Algorytmy global illumination oparte są na tzw. *rendering equation* [3]:

$$L_o(\mathbf{p}, \mathbf{v}) = L_e(\mathbf{p}, \mathbf{v}) + \int_{\Omega} f(\mathbf{l}, \mathbf{v}) \mathbf{L}_o(r(\mathbf{p}, \mathbf{l}), -\mathbf{l})(\mathbf{n} \cdot \mathbf{l})^+ d\mathbf{l} \quad (3.3)$$

gdzie:

- $\mathbf{L}_o(\mathbf{p}, \mathbf{v})$ - światło wychodzące od punktu \mathbf{p} w kierunku \mathbf{v} (kierunek kamery).

- $L_e(p, v)$ - światło emitowane przez powierzchnię w punkcie p w kierunku v .
- Ω - to powierzchnia półsfery, która znajduje się nad punktem dla którego oblicza się wychodzące światło. Całkowanie po tej półferze odpowiada sumowaniu wkładu światła docierającego ze wszystkich możliwych kierunków.
- $f(l, v)$ - to tzw. *bidirectional reflectance distribution function* (dalej BRDF). BRDF to funkcja opisująca w jaki sposób światło nadchodzące z kierunku l jest odbijane w kierunku v przez dany materiał.
- $L_o(r(p, l), -l)$ - przychodzące światło. p to punkt na powierzchni, a l to kierunek. Funkcja $r(p, l)$ zwraca punkt przecięcia się promienia, którego p to punkt początkowy, a l kierunek. Komponent ten oznacza, że światło przychodzące do punktu p jest światłem wychodzącym od innego punktu.
- $(n \cdot l)^+$ - iloczyn skalarny wektora normalnego powierzchni z wektorem kierunku światła, ponieważ oba wektory są znormalizowane wynikiem jest cosinus kąta pomiędzy tymi wektorami. $+$ oznacza branie tylko dodatnich wyników.

Analitycznie równanie 3.3 jest niemożliwe do rozwiązania (poza bardzo prostymi scenami), ponieważ aby rozwiązać równanie dla punktu p trzeba znać wynik dla punktu wcześniejszego p' , powstaje nieskończona rekurencja na nieskończonej liczbie kierunków z których światło dochodzi do punktu p .

Równanie renderingu rozwiązuje się metodami numerycznymi np. metodą monte-carlo. [DALEJ COOK TORRANCE]

Rozdział 4

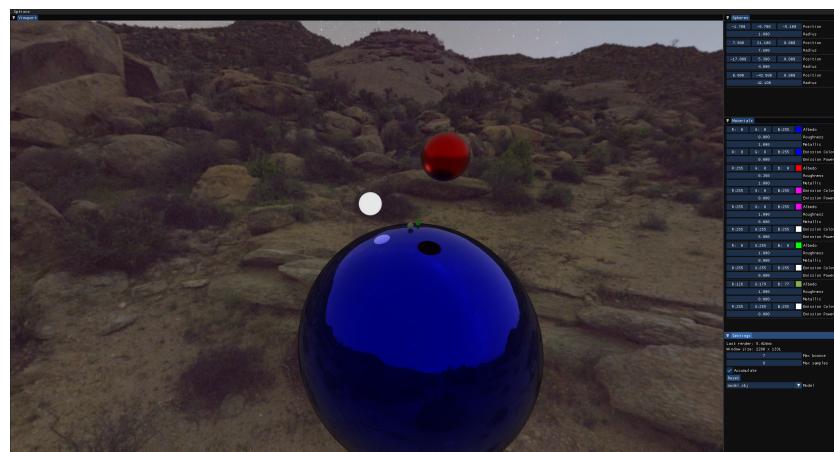
Prezentacja silnika

4.1 Biblioteki

- **DirectX 11** jest to biblioteka graficzna wykorzystywana głównie do tworzenia gier komputerowych (np. Baldur's Gate 3, Wiedźmin 3) lub innych aplikacji graficznych. Biblioteka została stworzona w 2009 roku przez firmę Microsoft. Dostępna jest tylko na komputerach z systemem windows i konsolach Xbox. W silniku wykorzystywana jest do grafiki poprzez shadery¹ takie jak: compute shader, pixel shader i vertex shader.
- **Win32** jest to interfejs programistyczny systemu Windows. Zawiera w sobie funkcje umożliwiające działanie programów w systemie. Okno prezentowanego silnika zostało stworzone za pomocą tej biblioteki.
- **ImGui** biblioteka wykorzystywana do tworzenia interfejsu użytkownika. Napisana jest w myśl paradygmatu "Immediate Mode GUI" (IM-GUI), który głównie polega na tym, że interfejs jest cały czas odświeżany, nie przechowuje on stanu między klatkami przez co jest zawsze zsynchronizowany z danymi. Biblioteka ta jest również łatwiejsza w obsłudze niż np. biblioteka QT.

¹Shader to program działający na karcie graficznej

- **Assimp** Open Asset Import Library jest to biblioteka umożliwiająca łatwe ładowanie modeli 3D w różnych formatach np. obj, glb. Napisana jest w języku C/C++.
- **Stb_Image** biblioteka wykorzystywana do ładowania tekstur w różnych formatach.



Rysunek 4.1: Zdjęcie przedstawia interfejs silnika.

4.2 Opis potoku graficznego

4.3 Implementacja algorytmów

Podsumowanie

4.4 Testy Wydajności

4.5 Zakończenie

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).
- [3] Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki i Sébastien Hillaire. *Real-Time Rendering*. 4 wyd. 2018.
- [4] *Wikipedia*. Dostęp 5.01.2026. URL: <https://en.wikipedia.org/wiki/Centroid>.
- [5] Matt Pharr, Jakob Wenzel i Greg Humphreys. *Physically Based Rendering. From Theory To Implementation*. 2023.
- [6] *Rodolphe Vaillant's homepage*. Dostęp 17.01.2026. URL: <https://rodolphe-vaillant.fr/entry/85/phong-illumination-model-cheat-sheet>.