Sprawozdanie – Algorytm Malarski z BSP

Adrian Rybaczuk 318483

18 maja 2025

Spis treści

1 Wstęp

Projekt polega na implementacji systemu wirtualnej kamery z wykorzystaniem algorytmu malarskiego (Painter's Algorithm) wspomaganego przez Binary Space Partitioning (BSP). Celem jest zapewnienie poprawnego renderowania sceny 3D z zachowaniem prawidłowej kolejności rysowania obiektów.

2 Algorytm Malarski z BSP

2.1 Reprezentacja ściany w przestrzeni 3D

Każda ściana w scenie jest reprezentowana jako zbiór wierzchołków w przestrzeni jednorodnej:

$$P_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}, \quad i \in \{1, 2, \ldots, n\}$$

Ściana jest zdefiniowana jako:

$$F = \{P_1, P_2, \ldots, P_n\}$$

2.2 Plaszczyzna dzieląca w BSP

Dla każdej ściany definiujemy płaszczyznę dzielącą w postaci:

$$ax + by + cz + d = 0$$

gdzie wektor normalny płaszczyzny:

$$\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

jest obliczany jako iloczyn wektorowy dwóch krawędzi ściany:

$$\mathbf{n} = (P_2 - P_1) \times (P_3 - P_1)$$

2.3 Podział przestrzeni

Dla każdej ściany F i punktu P w przestrzeni, możemy określić jego położenie względem płaszczyzny dzielącej:

$$\text{Pozycja}(P) = \begin{cases} \text{Przed} & \text{gdy } \mathbf{n} \cdot (P - P_1) > 0 \\ \text{Za} & \text{gdy } \mathbf{n} \cdot (P - P_1) < 0 \\ \text{Na} & \text{gdy } \mathbf{n} \cdot (P - P_1) = 0 \end{cases}$$

2.4 Struktura drzewa BSP

Drzewo BSP jest strukturą binarną, gdzie każdy węzeł zawiera:

- Ścianę dzielącą
- Poddrzewo przed ścianą
- Poddrzewo za ścianą

Formalnie, drzewo BSP T jest zdefiniowane jako:

$$T = \begin{cases} \emptyset & \text{dla pustego drzewa} \\ (F, T_{przed}, T_{za}) & \text{dla węzła z ścianą } F \end{cases}$$

2.5 Algorytm budowania drzewa BSP

Dla zbioru ścian $S = \{F_1, F_2, \ldots, F_n\}$, drzewo BSP jest budowane rekurencyjnie:

1. Wybierz ścianę F_i jako ścianę dzielącą
2. Podziel pozostałe ściany na zbiory:

$$S_{przed} = \{F_j | F_j \text{ jest przed } F_i\}$$

$$S_{za} = \{F_j | F_j \text{ jest za } F_i\}$$

$$S_{na} = \{F_j | F_j \text{ jest na } F_i\}$$

3. Rekurencyjnie zbuduj poddrzewa dla S_{przed} i S_{za}

2.6 Algorytm malarski z BSP

Kolejność rysowania jest określona przez przejście drzewa BSP w porządku in-order, z uwzględnieniem pozycji kamery:

$$\text{Rysuj}(T, P_{camera}) = \begin{cases} \emptyset & \text{dla pustego drzewa} \\ \text{Rysuj}(T_{za}, P_{camera}) \cup \{F\} \cup \text{Rysuj}(T_{przed}, P_{camera}) & \text{gdy } P_{camera} \text{ jest przed } F \\ \text{Rysuj}(T_{przed}, P_{camera}) \cup \{F\} \cup \text{Rysuj}(T_{za}, P_{camera}) & \text{gdy } P_{camera} \text{ jest za } F \end{cases}$$

3 Optymalizacje

3.1 Przyspieszenie obliczeń

Dla przyspieszenia obliczeń, wykorzystujemy:

- Normalizację wektorów normalnych
- Cachowanie wyników testów położenia
- Wczesne odrzucanie ścian za kamerą

3.2 Stabilizacja numeryczna

Aby zapewnić stabilność numeryczną, wprowadzamy epsilon ϵ:

$$\text{Pozycja}(P) = \begin{cases} \text{Przed} & \text{gdy } \mathbf{n} \cdot (P - P_1) > \epsilon \\ \text{Za} & \text{gdy } \mathbf{n} \cdot (P - P_1) < -\epsilon \\ \text{Na} & \text{gdy } |\mathbf{n} \cdot (P - P_1)| \leq \epsilon \end{cases}$$

4 Testy

4.1 Test 1: Poprawność kolejności rysowania

Test sprawdzający poprawność algorytmu BSP w przypadku złożonej sceny z wieloma obiektami. Scena zawiera:

- Duży sześcian
- Piramidę umieszczoną wewnątrz sześcianu
- Cylinder przecinający sześcian

Kolejność obiektów na scenie (od najbliższego do najdalszego):

1. Cylinder (przecina sześcian, część jest przed nim)
2. Sześcian (duży, zawiera piramidę)
3. Piramida (umieszczona wewnątrz sześcianu)

Zalecane ustawienie kamery:

- Pozycja: (5, 0, 5) - widok z boku
- Kierunek: (-1, 0, 0) - patrząc na scenę z boku
- Kąt widzenia: 60 stopni

Oczekiwany wynik: Obiekty są rysowane w prawidłowej kolejności, z zachowaniem poprawnej widoczności. Cylinder powinien być poprawnie renderowany względem sześcianu.

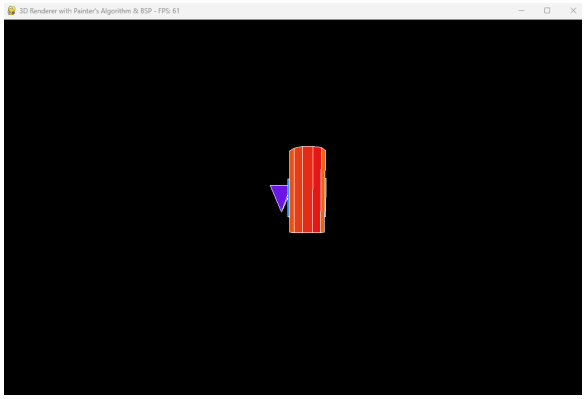
4.2 Test 2: Zachowanie przy zmianie pozycji kamery

Test sprawdzający zachowanie algorytmu przy dynamicznej zmianie pozycji kamery:

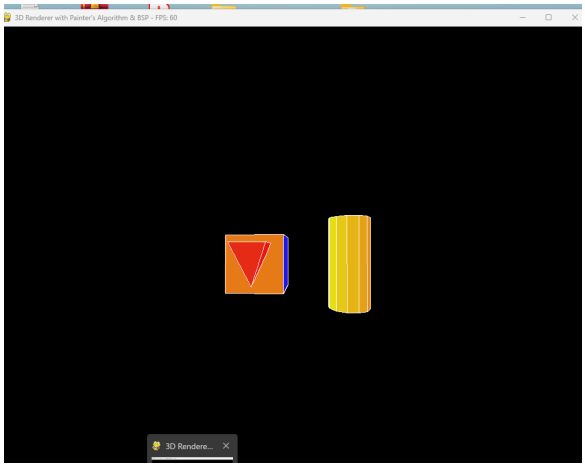
- Kamera porusza się wokół złożonej sceny
- Sprawdzenie poprawności kolejności rysowania z różnych perspektyw
- Weryfikacja płynności przejść między różnymi widokami

Kolejność obiektów na scenie:

1. Cylinder (na platformie, z prawej strony)
2. Platforma (pod wszystkimi obiektami)



Rysunek 1: Scena testowa - widok z boku



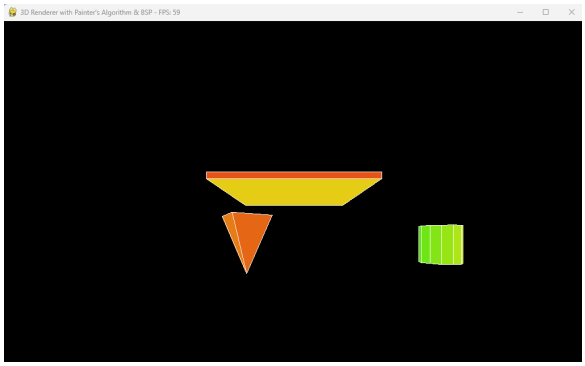
Rysunek 2: Wynik renderowania

3. Piramida (na platformie, z lewej strony)

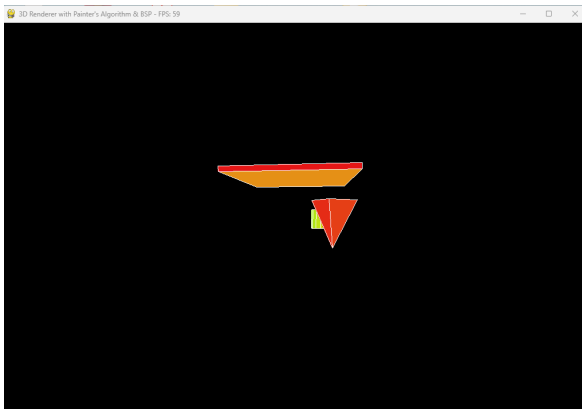
Zalecane pozycje kamery:

1. Widok 1: (0, 0, 10) - widok z przodu
2. Widok 2: (5, 0, 5) - widok z boku

Oczekiwany wynik: Płynna zmiana kolejności rysowania przy zachowaniu poprawności widoczności.



Rysunek 3: Widok 1 - kamera z przodu



Rysunek 4: Widok 2 - kamera z boku

4.3 Test 3: Obsługa przecinających się obiektów

Test sprawdzający zachowanie algorytmu w przypadku przecinających się obiektów:

- Dwa sześciany przecinające się pod kątem 45 stopni
- Piramida przecinająca cylinder
- Złożony obiekt z wieloma przecinającymi się ścianami

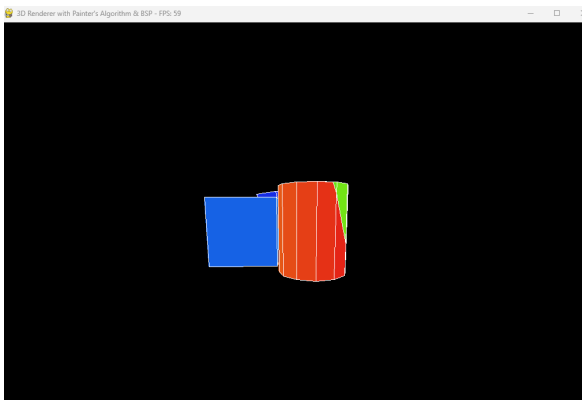
Kolejność obiektów na scenie:

1. Pierwszy sześcian (przecina się z drugim)
2. Drugi sześcian (przecina się z pierwszym)
3. Piramida (przecina cylinder)
4. Cylinder (przecinany przez piramidę)

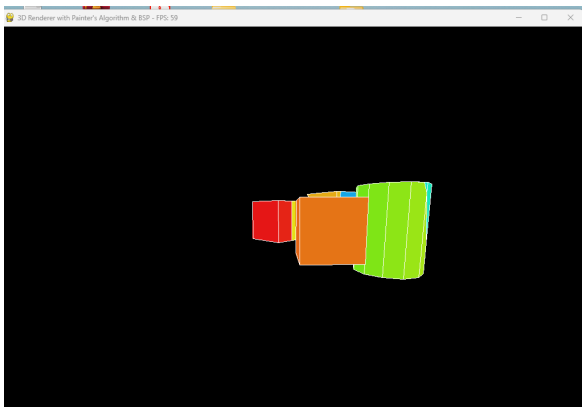
Zalecane ustawienie kamery:

- Pozycja: (3, 3, 3) - widok ukośny
- Kierunek: (-1, -1, -1) - patrząc na przecinające się obiekty
- Kąt widzenia: 60 stopni

Oczekiwany wynik: Poprawne rozdzielenie przecinających się obiektów i zachowanie prawidłowej kolejności rysowania.



Rysunek 5: Scena z przecinającymi się obiektami



Rysunek 6: Wynik renderowania z poprawną kolejnością

4.4 Test 4: Wydajność przy złożonej scenie

Test sprawdzający wydajność algorytmu przy dużej liczbie obiektów:

- Scena zawierająca więcej obiektów
- Różne typy obiektów (sześciany, piramidy, cylindry)
- Obiekty umieszczone w różnych odległościach od kamery

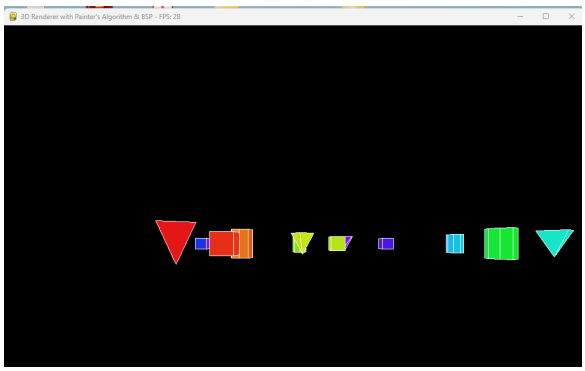
Układ obiektów na scenie:

1. Pierwszy rząd: sześciany (z = 3)
2. Drugi rząd: piramidy (z = 5)
3. Trzeci rząd: cylindry (z = 7)

Zalecane ustawienia kamery:

1. Widok ogólny: (0, 0, 15) - widok z góry
2. Widok szczegółowy: (2, 2, 5) - widok z bliska

Oczekiwany wynik: Płynne renderowanie sceny z zachowaniem poprawności kolejności rysowania.



Rysunek 7: Widok z przodu ogólny złożonej sceny

Rysunek 8: Widok z góry renderwoania

5 Podsumowanie

Implementacja algorytmu malarskiego z BSP zapewnia:

- Poprawną kolejność rysowania obiektów
- Efektywne zarządzanie złożonymi scenami
- Stabilność numeryczną
- Możliwość optymalizacji wydajności

6 Literatura

1. Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1990). *Computer Graphics: Principles and Practice* (2nd ed.). Addison-Wesley.
2. BSP Tree FAQ. `http://www.faqs.org/faqs/graphics/bsptree-faq/`
3. Binary Space Partitioning. `https://en.wikipedia.org/wiki/Binary_space_partitioning`
4. Tutorials Point. Binary Space Partitioning Trees in Computer Graphics. `https://www.tutorialspoint.com/computer_graphics/computer_graphics_binary_space_partitioning.htm`
5. Kiciak, P. (2010). Grafika komputerowa I. Uniwersytet Warszawski. `https://mst.mimuw.edu.pl/lecture.php?lecture=gk1&part=Ch9`