**Sprawozdanie z projektu 2  
„Wykorzystanie i implementacja algorytmu eliminacji powierzchni zasłoniętych”  
w ramach zajęć z Grafiki Komputerowej**

Maciej Karabin, 325483

**1. Wstęp i cel projektu**

Projekt polegał na implementacji algorytmu eliminacji powierzchni zasłoniętych oraz połączeniu go z wirtualną kamerą wykonaną w ramach projektu nr 1.

Cel został osiągnięty przy użyciu wyłącznie podstawowych metod graficznych, bez użycia OpenGL, a do eliminacji powierzchni zasłoniętych użyłem **algorytmu podziału binarnego przestrzeni (drzewo BSP).**

**2. Technologie i biblioteki wykorzystane w projekcie**

Projekt został wykonany jako rozwinięcie projektu nr 1 w języku **Python** z użyciem tych samych bibliotek:

* **Tkinter** – graficzna biblioteka do stworzenia interfejsu użytkownika oraz renderowania na ekranie.
* **numpy** – biblioteka matematyczna użyta do operacji geometrycznych, transformacji punktów, manipulacji macierzami rotacji oraz wektorami (translacje, obroty, rzutowanie perspektywiczne).

**3. Logiczny opis struktury projektu**

W porównaniu do pierwszego projektu, wprowadzono dwie zmiany:

**3.1 Nowy sposób generowania i wczytywania punktów**

W ramach pierwszego projektu generowane były pary punktów, które potem były wyświetlane jako odcinki. Na potrzebę tego projektu, poza parami punktów tworzących odcinek, generujemy również zbiory 4 punktów tworzące ściany, aby umożliwić przełączanie między trybami **BSP** oraz **wireframe.** Wymagało to aktualizacji wczytywania danych z pliku. Podobnie jak w poprzednim projekcie, generuję duży sześcian złożony z mniejszych sześcianów o modyfikowalnych parametrach.

**3.2 Implementacja algorytmu BSP**

Główną nowością jest implementacja algorytmu BSP. Działaon następująco:

**1. Tworzenie drzewa BSP:**

* Tworzone jest drzewo, które dzieli przestrzeń na półprzestrzenie
* Każdy węzeł zawiera płaszczyznę podziału oraz listy wielokątów z przodu i z tyłu
* Rekurencyjnie budowane są pod-drzewa dla wielokątów leżących po obu stronach

**2. Klasa Plane:**

* Reprezentuje płaszczyznę podziału w przestrzeni 3D
* Posiada metody do klasyfikacji punktów i wielokątów względem płaszczyzny
* Umożliwia podział wielokątów przecinających płaszczyznę

**3. Klasa BSPNode:**

* Reprezentuje węzeł drzewa BSP
* Zawiera płaszczyznę podziału, listy wielokątów oraz odniesienia do podrzewa
* Implementuje rekurencyjne budowanie struktury drzewa
* Zawiera metodę do pobierania widocznych wielokątów względem pozycji kamery

**3.3 Zaktualizowane renderowanie sceny**

Proces renderowania z wykorzystaniem drzewa BSP wygląda następująco:

**1. Inicjalizacja:** Budowa drzewa BSP z wielokątów sceny (podczas inicjalizacji aplikacji)

**2. Podczas renderowania każdej klatki:**

* Pobieranie widocznych wielokątów na podstawie aktualnej pozycji kamery
* Sprawdzanie, czy wielokąty są zwrócone w kierunku kamery
* Rzutowanie widocznych wielokątów na płaszczyznę ekranu
* Renderowanie wielokątów w określonym kolorze

**3. Dodatkowe funkcje:**

* Możliwość przełączania między renderowaniem z użyciem BSP a prostym renderowaniem krawędzi

**4. Testy**

Zrzuty ekranu przedstawiające działanie programu:

Obraz zawierający tekst, zrzut ekranu, diagram, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

***Rys 1 – widok z przodu dla obiektu 3x3x3***

Obraz zawierający tekst, diagram, zrzut ekranu, Plan

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

***Rys 2 – widok z bliska dla obiektu 3x3x3***

Obraz zawierający diagram, origami

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

***Rys 3 – przykład niedokładności uproszczonego algorytmu malarza***

Obraz zawierający tekst, diagram, Plan, linia

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

***Rys 4 – widok z wnętrzna kostki dla obiektu 5x5x5***

**5. Podsumowanie**

Implementacja algorytmu BSP stanowi znaczącą poprawę w stosunku do prostszego algorytmu malarza z użyciem środków ciężkości, który początkowo wykonałem.

Próbowałem wykonać również implementację za pomocą uproszczonego Z-bufora, w której mielibyśmy używać wyłącznie wierzchołków ścian, zamiast wszystkich pikseli. Niestety moja implementacja nie dość, że była bardzo wadliwa i zasłaniała powierzchnie gorzej od wcześniej wspomnianego algorytmu malarza, to dodatkowo była na tyle źle zoptymalizowana, że finalna projekcja dawała mniej więcej jedną klatkę na 5 sekund.

Dlatego też zdecydowałem się na porzucenie mojego pierwotnego kodu i skupienie się na implementacji drzewa BSP od zera.

BSP rozwiązuje wiele problemów:

* Eliminuje błędy sortowania występujące w algorytmie malarza dla przecinających się wielokątów
* Zapewnia poprawną widoczność niezależnie od złożoności sceny
* Działa efektywnie dla dynamicznych scen z poruszającą się kamerą

Należy jednak zauważyć, że drzewo BSP wymaga większych nakładów obliczeniowych podczas jego budowania, ale za to zapewnia znacznie lepszą jakość eliminacji powierzchni zasłoniętych. W przypadku statycznych scen, drzewo wystarczy zbudować raz, co czyni ten algorytm bardzo efektywnym.

Możliwość przełączania między widokiem BSP a prostym rysowaniem krawędzi pozwala na porównanie obu metod i wyraźnie pokazuje korzyści płynące z zastosowania zaawansowanego algorytmu eliminacji powierzchni zasłoniętych.