**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 malarza** w uproszczonej wersji – obliczam.

**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, zamiast par punktów tworzących odcinek, generujemy zbiory 4 punktów tworzące ściany. Wymagało to też aktualizacji wczytywania danych z pliku (wczytujemy teraz punkty w grupach po 4, a nie jak wcześniej po 2). Podobnie jak w poprzednim projekcie, generuję duży sześcian złożony z mniejszych sześcianów o modyfikowalnych parametrach.

**3.2 Zaktualizowane renderowanie sceny i implementacja algorytmu malarza**

Algorytm malarza zaimplementowałem wewnątrz funkcji renderującej scenę, co znacznie zwiększyło jej złożoność. Pierwotnie, funkcja ta pobierała pary punktów z pliku, a następnie renderowała linie pomiędzy nimi. Obecnie funkcja renderująca połączona z algorytmem malarza cyklicznie wykonuje następującą linię kroków:

1. Zbierz wszystkie wierzchołki
2. Dla każdego wierzchołka danej ściany:
   1. Odejmij położenie kamery od wierzchołka, żeby kamera była punktem odniesienia (translacja)
   2. Przekształć punkt przez transponowaną macierz obrotu kamery, by wyrównać go z kierunkiem patrzenia. (rotacja)
   3. Jeżeli dowolny punkt ściany znajduje się za kamerą (z <= 0), nie renderuj jej i **przejdź do kolejnej ściany**
   4. Dodaj perspektywę i zoom
   5. Oblicz średnią głębokość punktów danej ściany
   6. Renderuj ściany malejąco względem średniej głębokości (od najdalszych do najbliższych)

**4. Testy**

Zrzuty ekranu przedstawiające działanie programu:

Obraz zawierający tekst, zrzut ekranu, krzesło, diagram

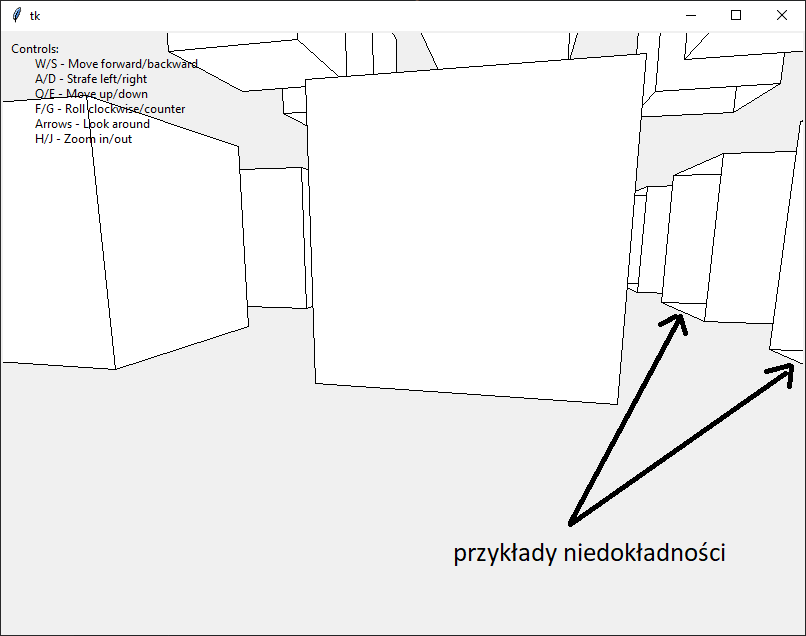
Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

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

Obraz zawierający diagram, szkic, tekst, Plan

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

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

****

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

**5. Podsumowanie**

Żeby algorytm działał bardziej precyzyjnie (szczególnie dla bardziej skomplikowanych brył) moglibyśmy śledzić każdy wierzchołek oddzielnie (a nie obliczać średnią głębokość całej ściany) np. przy pomocy Z-buffera.

Warto jednak podkreślić, że zastosowane uproszczenie zapewnia wystarczającą wydajność dla prostych scen, zachowując przejrzystość kodu. Główną zaletą algorytmu malarza w tej wersji jest minimalny narzut obliczeniowy, co pozwala na płynną interakcję nawet na dużych siatkach sześcianów. Jednocześnie, w scenach z wieloma przecinającymi się powierzchniami o zbliżonych głębokościach, błędy kolejności renderowania stają się bardziej widoczne – w takich przypadkach warto rozważyć hybrydowe podejście, łączące sortowanie ścian z dodatkowym sprawdzaniem głębi dla kluczowych fragmentów.