Zadanie 1 - Grafika Komputerowa

Witold Winiarski 325523

Wprowadzenie

Poniższy raport szczegółowo analizuje techniki i struktury danych wykorzystane w implementacji renderowania krawędziowego (wireframe) w dostarczonym kodzie Pygame. Skupia się na reprezentacji obiektów 3D, zastosowanych transformacjach wektorowych oraz kluczowych wzorach matematycznych rządzących procesem wizualizacji.

1. Reprezentacja Prostopadłościanów i Sceny

Podstawą wizualizacji są dane geometryczne opisujące obiekty na scenie. W tym przypadku są to prostopadłościany.

- **Definicja Wierzchołków (stworz_wierzcholki_prostopadloscianu):**
  - Funkcja ta generuje 8 wierzchołków prostopadłościanu na podstawie jego środka (sx, sy, sz) oraz wymiarów (szerokosc, wysokosc, glebokosc) w globalnym układzie współrzędnych świata.
  - Obliczane są połowy wymiarów: pol_szer = szerokosc / 2, pol_wys = wysokosc / 2, pol_gleb = glebokosc / 2.
  - Każdy z 8 wierzchołków jest tworzony przez kombinację dodawania lub odejmowania tych połówek od współrzędnych środka. Na przykład:
    - Wierzchołek 0: (sx - pol_szer, sy - pol_wys, sz - pol_gleb)
    - Wierzchołek 1: (sx + pol_szer, sy - pol_wys, sz - pol_gleb)
    - Wierzchołek 6: (sx + pol_szer, sy + pol_wys, sz + pol_gleb)
    - ... i tak dalej dla wszystkich 8 narożników.
  - Każdy wierzchołek jest przechowywany jako słownik Pythona: {'x': wartosc_x, 'y': wartosc_y, 'z': wartosc_z}.
- **Definicja Krawędzi (krawedzie_prostopadloscianu):**
  - Jest to *stała* lista definiująca topologię połączeń wierzchołków dla *każdego* prostopadłościanu. Zawiera 12 krotek, gdzie każda krotka (i, j) oznacza, że istnieje krawędź pomiędzy wierzchołkiem o indeksie i a wierzchołkiem o indeksie j (z listy wygenerowanej przez stworz_wierzcholki_prostopadloscianu).
  - Przykłady: (0, 1) łączy dolny-lewy-przedni z dolnym-prawym-przednim; (0, 4) łączy dolny-lewy-przedni z górnym-lewym-przednim.
  - Ta lista jest wykorzystywana bezpośrednio w pętli renderującej tryb wireframe do określenia, które pary wierzchołków należy połączyć linią.

- **Definicja Ścian (sciany_prostopadloscianu):**
  - Podobnie jak krawędzie, jest to stała lista definiująca 6 ścian prostopadłościanu. Każda ściana jest reprezentowana przez krotkę 4 indeksów wierzchołków, zazwyczaj w kolejności przeciwnej do ruchu wskazówek zegara, patrząc z zewnątrz obiektu.
  - Choć ściany nie są *rysowane* w trybie wireframe, są one kluczowe dla trybu wypełnienia (np. do określenia widoczności ściany i rysowania wielokątów). Definiują one powierzchnie bryły.
- **Struktura Sceny (dane_sceny):**
  - Jest to lista, gdzie każdy element reprezentuje jeden obiekt (prostopadłościan) na scenie.
  - Każdy element to słownik zawierający:
    - 'wierzcholki': Lista 8 wierzchołków wygenerowana dla tego konkretnego obiektu.
    - 'krawedzie': Odniesienie do stałej listy krawedzie_prostopadloscianu.
    - 'sciany': Odniesienie do stałej listy sciany_prostopadloscianu.
    - 'kolor': Kolor używany do rysowania krawędzi w trybie wireframe.
    - 'kolory_scian': Lista kolorów dla każdej ściany, używana w trybie wypełnienia.

2. Operacje Wektorowe i Transformacje Kamery

Rdzeniem silnika 3D są operacje na wektorach 3D, używane do transformacji punktów i orientacji kamery.

- **Podstawowe Operacje Wektorowe:**
  - dodaj_wektory(v1, v2): Suma dwóch wektorów (dodawanie komponentów).
  - odejmij_wektory(v1, v2): Różnica dwóch wektorów (odejmowanie komponentów).
  - mnoz_wektor_przez_skalar(v, s): Mnożenie wektora przez liczbę (skalowanie komponentów).
  - iloczyn_skalarny(v1, v2): Wynikiem jest skalar: v1.x*v2.x + v1.y*v2.y + v1.z*v2.z. Używany m.in. do projekcji jednego wektora na drugi.
  - iloczyn_wektorowy(v1, v2): Wynikiem jest wektor prostopadły do obu wektorów wejściowych. Używany m.in. do znajdowania wektorów prostopadłych (np. przy ortonormalizacji).
  - normalizuj(v): Zwraca wektor jednostkowy (o długości 1) wskazujący w tym samym kierunku co v. Oblicza długość $||v|| = \text{sqrt}(v.x^2 + v.y^2 + v.z^2)$ i dzieli każdy komponent przez tę długość.
- **Baza Orthonormalna Kamery:**

- Orientacja kamery jest reprezentowana przez trzy wzajemnie prostopadłe wektory jednostkowe: kam_do_przodu, kam_prawo, kam_gora. Tworzą one lokalny układ współrzędnych kamery.
- **Rotacja Kamery (obroc_wektor):** Zamiast manipulować kątami Eulera (co może prowadzić do problemu blokady gimbala - gimbal lock), rotacje są stosowane bezpośrednio do wektorów bazy kamery za pomocą funkcji obroc_wektor. Implementuje ona **wzór Rodrigueza na obrót wektora**: v_rotated = v * cos(θ) + (k × v) * sin(θ) + k * (k · v) * (1 - cos(θ)) gdzie v to obracany wektor, k to znormalizowany wektor osi obrotu, θ to kąt obrotu, × oznacza iloczyn wektorowy, a · oznacza iloczyn skalarny. Funkcja obroc_wektor dokładnie implementuje ten wzór, używając podstawowych operacji wektorowych.
- **Aktualizacja Bazy:** W pętli głównej, na podstawie wejścia użytkownika, obliczane są małe kąty obrotu (delta_yaw_angle, delta_pitch_angle, delta_roll_angle). Następnie odpowiednie wektory bazy są obracane wokół odpowiednich osi (np. dla odchylenia (yaw) obraca się kam_do_przodu i kam_prawo wokół kam_gora).
- **Ortonormalizacja (ortonormalizuj):** Po wykonaniu obrotów, z powodu błędów numerycznych operacji zmiennoprzecinkowych, wektory bazy mogą nie być już idealnie prostopadłe lub jednostkowe. Funkcja ortonormalizuj przywraca te własności:
  1. Normalizuje wektor do_przodu.
  2. Oblicza nowy wektor prawo jako znormalizowany iloczyn wektorowy do_przodu × gora. Gwarantuje to prostopadłość prawo do do_przodu.
  3. Oblicza nowy wektor gora jako znormalizowany iloczyn wektorowy prawo × do_przodu. Gwarantuje to prostopadłość gora do obu pozostałych wektorów. Proces ten jest podobny do ortogonalizacji Grama-Schmidta.

- **Transformacja Świat -> Kamera (swiat_do_kamery):**
  - Celem jest wyrażenie pozycji wierzchołka W (współrzędne świata) względem układu współrzędnych kamery, mając jej pozycję C i bazę (P=kam_do_przodu, R=kam_prawo, U=kam_gora).
  - **Krok 1: Wektor Względny:** Oblicz wektor od kamery do wierzchołka: V = W - C (używając odejmij_wektory).
  - **Krok 2: Projekcja na Osie Kamery:** Oblicz współrzędne (x', y', z') w układzie kamery, rzutując wektor V na osie bazy kamery za pomocą iloczynu skalarnego:
    - x' = V · R (projekcja na oś "prawo" kamery)
    - y' = V · U (projekcja na oś "góra" kamery)
    - z' = V · P (projekcja na oś "przód" kamery - głębokość)

- Funkcja zwraca słownik {'x': x', 'y': y', 'z': z'}.

3. Wzory Matematyczne: Rzutowanie i Przycinanie

- **Rzutowanie Perspektywiczne (rzutuj_wierzcholek):**
  - Transformuje punkt (x', y', z') z przestrzeni kamery na płaszczyznę ekranu (ekran_x, ekran_y).
  - **Kluczowy element:** Odwrotna proporcjonalność współrzędnych ekranowych do głębokości z'.
  - **Współczynnik Skalujący d:** Obliczenie tan_fov_half = tan(pole_widzenia / 2) oraz d = (WYSOKOSC_EKRANU / 2) / tan_fov_half wiąże pole widzenia z rozmiarem ekranu. d reprezentuje efektywną odległość płaszczyzny projekcji, na którą rzutowane są punkty, przeskalowaną do wymiarów ekranu.
  - **Wzory Projekcji:**
    - ekran_x = (x' / z') * d + SZEROKOSC_EKRANU / 2
    - ekran_y = -(y' / z') * d + WYSOKOSC_EKRANU / 2
    - x'/z' i y'/z' to współrzędne punktu rzutowanego na płaszczyznę w odległości 1 od kamery. Mnożenie przez d skaluje je do "ekranu", a dodanie połowy wymiarów ekranu centruje wynik. Ujemny znak przy y' dostosowuje do układu współrzędnych ekranu Pygame.
- **Przycinanie do Bliskiej Płaszczyzny (Interpolacja Liniowa):**
  - Gdy krawędź łączy wierzchołek widoczny wk1 (z głębokością z1 >= BLISKIE_CIECIE) i niewidoczny wk2 (z głębokością z2 < BLISKIE_CIECIE).
  - **Parametr Interpolacji t:** Chcemy znaleźć punkt na linii wk1-wk2, którego współrzędna z wynosi dokładnie BLISKIE_CIECIE. Punkt na linii można sparametryzować jako P(t) = wk1 + t * (wk2 - wk1) dla t od 0 do 1. Szukamy t takiego, że z-owa współrzędna P(t) jest równa BLISKIE_CIECIE: z_clip = z1 + t * (z2 - z1) Ustawiając z_clip = BLISKIE_CIECIE i rozwiązując dla t, otrzymujemy: t = (BLISKIE_CIECIE - z1) / (z2 - z1)
  - **Obliczenie Punktu Przecięcia w_clip:** Używając obliczonego parametru t, znajdujemy współrzędne x i y punktu przecięcia przez interpolację liniową:
    - x_clip = wk1['x'] + t * (wk2['x'] - wk1['x'])
    - y_clip = wk1['y'] + t * (wk2['y'] - wk1['y'])
    - z_clip = BLISKIE_CIECIE
  - Następnie rzutowany jest widoczny wierzchołek wk1 oraz obliczony punkt przecięcia w_clip, a linia rysowana jest pomiędzy ich rzutami na ekranie.

Podsumowanie

Implementacja renderowania krawędziowego w tym kodzie stanowi przykład

zastosowania fundamentalnych technik grafiki 3D. Reprezentacja obiektów opiera się na wierzchołkach i listach połączeń (krawędziach). Kluczowe są transformacje wektorowe: przeniesienie geometrii do układu kamery (poprzez odjęcie pozycji kamery i projekcję na jej osie za pomocą iloczynu skalarnego) oraz rzutowanie perspektywiczne (wykorzystujące dzielenie przez głębokość z' i skalowanie oparte na polu widzenia). Rotacja kamery jest realizowana za pomocą stabilnego numerycznie wzoru Rodrigueza, a utrzymanie ortonormalności bazy kamery zapewnia proces ortonormalizacji. Przycinanie do bliskiej płaszczyzny, wykorzystujące interpolację liniową, gwarantuje poprawne wyświetlanie krawędzi przecinających granicę widoczności.