Wirtualna kamera

Część pierwsza projektu
Grafika Komputerowa

Łukasz Laskowski
295803

19.03.2021 r.

1. Wstęp

Pierwsza część projektu polegała na zaimplementowaniu kamery wirtualnej w dowolnym języku. Kamera nie musiała eliminować elementów przysłoniętych.

Zadanie rozwiązałem z wykorzystaniem języka Python, do którego przydatnymi okazały się dwie biblioteki: Tkinter (odpowiedzialny za rysowanie obiektów na utworzonym canvasie) oraz NumPy (przeprowadzający operacje na macierzach).

Rozwiązania poszczególnych etapów oparte są na zawartości książki *Fundamentals of Computer Graphics, Stephen R. Marschner, Peter Shirley, Michael Ashikhmin.*

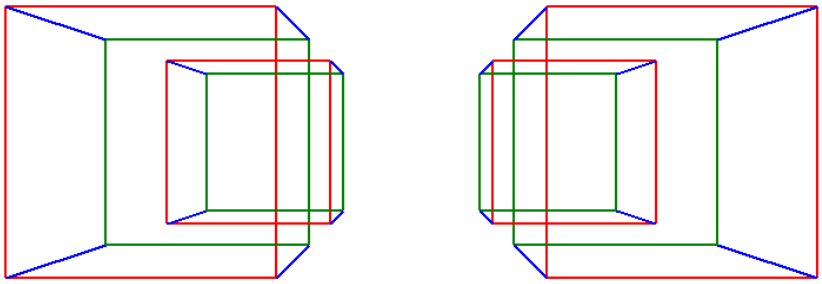
2. Opis obiektów

Obraz wyświetlany na ekranie przedstawia jedynie cztery prostopadłościany, składające się z poszczególnych krawędzi, których końce opisane zostały w pliku tekstowym. Plik ten zawiera linie, z której każda przechowuje początkowe koordynaty (x, y, z) punktów. Położenia te są wczytywane do programu oraz natychmiastowo konwertowane do współrzędnych jednorodnych (x, y, z, 1).

3. Opis operacji

Program umożliwia wykonanie operacji translacji i rotacji w trzech osiach oraz zoomu. Wszystkie transformacje opierały się o poniższy schemat:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_x * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



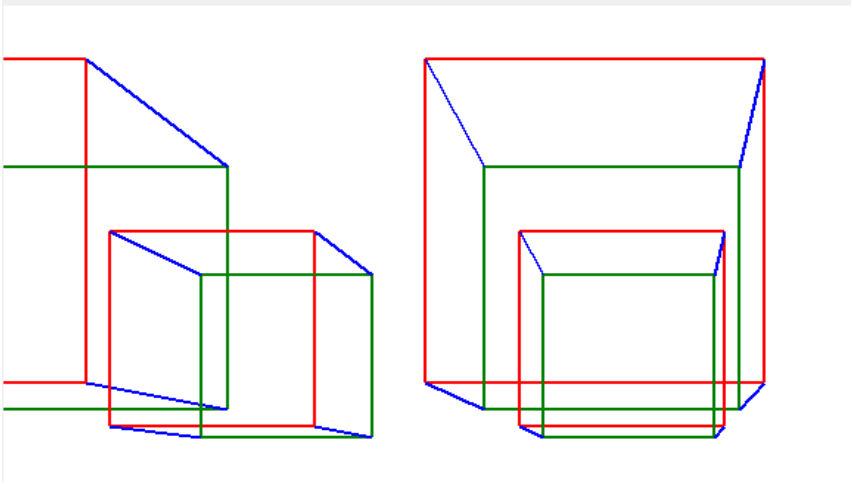
Rys. 1 Początkowy widok

a. Translacja

Każdy ruch kamery wymagał wykorzystania podstawowej macierzy translacji, w której umieszczane były odpowiednie współczynniki.

$$M_{trans} = \begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

gdzie x_t, y_t i z_t to przesunięcia względem odpowiednich osi.

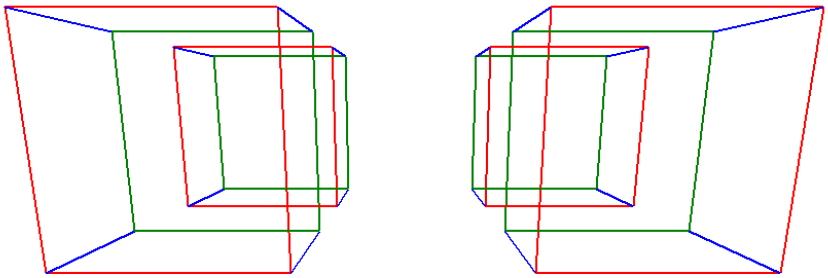


Rys. 2 Widok po przejściu w prawo i w dół

b. Rotacja

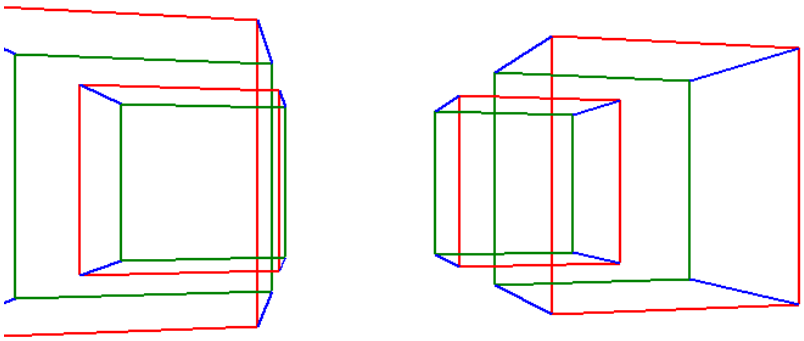
Obroty możliwe są w trzech osiach, przy czym dla każdej osi macierz transformacji zdefiniowana jest nieco inaczej:

$$M_{rotX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(x) & \sin(x) & 0 \\ 0 & -\sin(x) & \cos(x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$



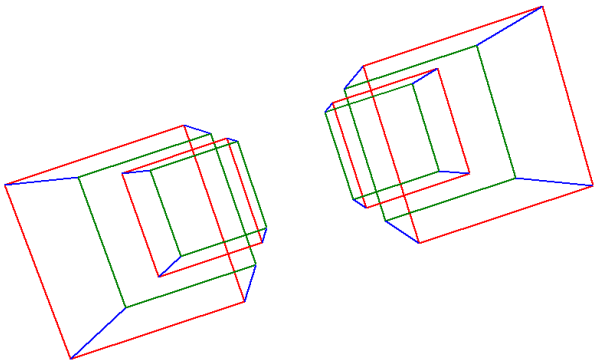
Rys. 3 Obraz po obrocie w osi X

$$M_{rotY} = \begin{bmatrix} \cos(x) & 0 & -\sin(x) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(x) & 0 & \cos(x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$



Rys. 4 Obraz po obrocie w osi Y

$$M_{rotZ} = \begin{bmatrix} \cos(x) & \sin(x) & 0 & 0 \\ -\sin(x) & \cos(x) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$



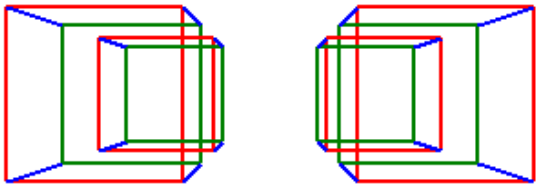
Rys. 5 Obraz po obrocie w osi Z

gdzie x jest miarą kąta obrotu w radianach.

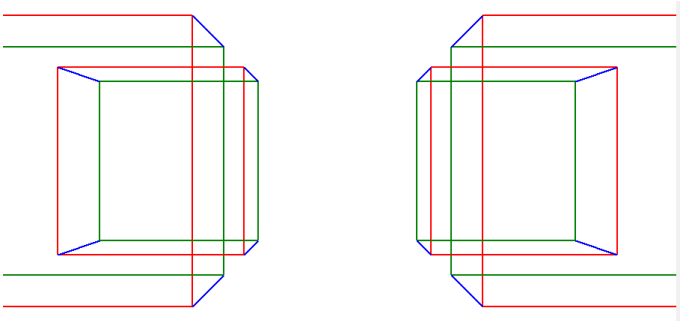
c. Zoom

Zoomowanie jest prostą operacją, skalującą wyświetlane obiekty. Do jej wykonania potrzebna była macierz:

$$M_{zoom} = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$



Rys. 6 Obraz po przeskalowaniu (0 < z < 1)



Rys. 7 Obraz po przeskalowaniu (z > 1)

a s jest tu współczynnikiem skalowania.

4. Odwzorowanie 3D na 2D

Odwzorowanie sześcianów z figur trójwymiarowych na obiekty wyświetlane w 2D odbywa się poprzez przeprowadzenie dwóch dodatkowych operacji na wszystkich punktach:

a. Zmiana układu współrzędnych kamery z wykorzystaniem transformacji perspektywicznej:

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

po czym obiekty rysowały się już w 2D, natomiast nie były jeszcze dopasowane do wyświetlania na naszej wielkości *canvasie*.
l, r, b, t, n i *f* to wymiary *orthographic view volume* (left, right, bottom, top, near i far).

b. Przejście z *canonical view volume* na widok ekranu z dopasowaniem do wymiarów canvasa:

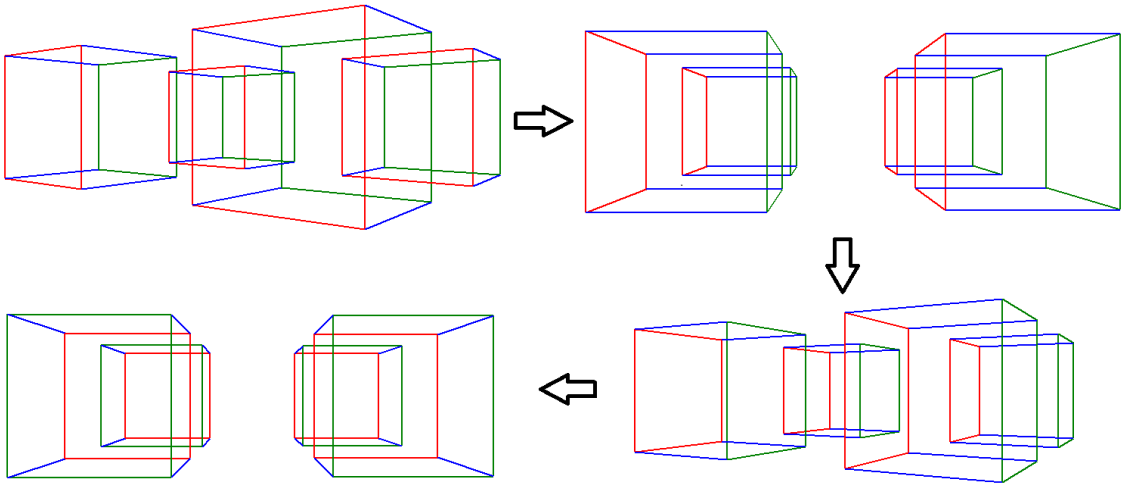
$$M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

gdzie n_x i n_y to wymiary canvasa.

5. Testy

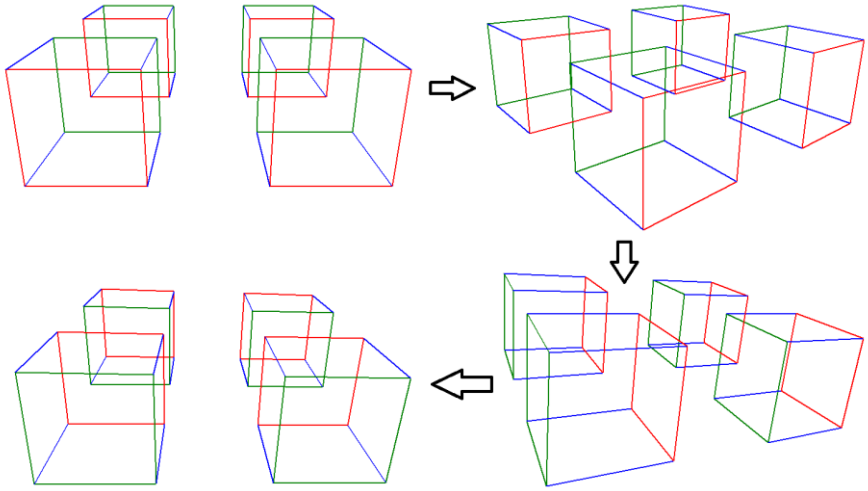
Po wykonaniu zadania przeprowadziłem kilka podstawowych testów, aby zweryfikować poprawność przeprowadzanych operacji.

a. obejście obiektów (ruch w prawo, obrót, ruch, obrót, ruch)



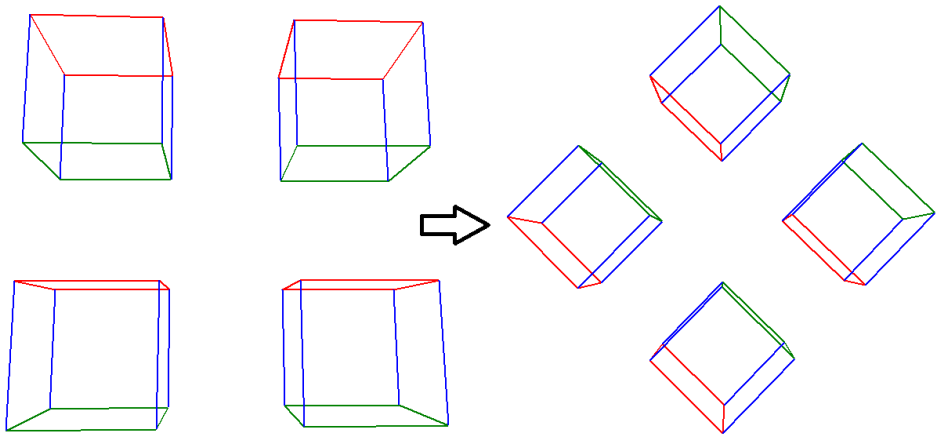
Rys. 8 Kolejne fazy przejścia wokół obiektów na ich wysokości

b. obejście obiektów, patrząc na nie pod kątem z góry



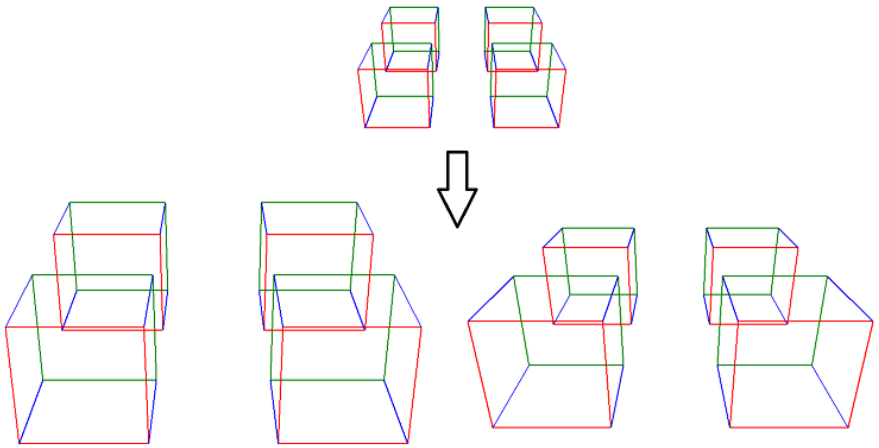
Rys. 9 Kolejne fazy przejścia wokół obiektów nad nimi

c. oglądanie obiektów z góry



Rys. 10 Kolejne fazy przejścia wokół obiektów na ich wysokości

d. porównanie operacji zoomowania i podejścia do obiektu



Rys. 11 Widok początkowy (na górze) oraz zoom (po lewej) i po podejściu (po prawej)

6. Wnioski

Okazuje się, że wykorzystanie *orthographic view volume* nie było warunkiem koniecznym do wykonania tego zadania, jednak prawdopodobnie przyda się to przy implementowaniu części drugiej projektu, w której dodawane będzie eliminowanie elementów przysłoniętych.

7. Repozytorium

Cały kod programu umieszczony jest na repozytorium:
https://github.com/LuckieLuke/Virtual-Camera