# Sprawozdanie z projektu 1

Krzysztof Jurkowski 319049

## Cel projektu

Projekt polegał na wykonaniu kamery wirtualnej, która wyświetlałaby wybraną przez nas scenę i posiadała funkcjonalności translacji, obrotów wokół każdej osi oraz zooma.

## Sposób wykonania

Projekt został wykonany w języku python z wykorzystaniem bibliotek numpy (działania na macierzach) oraz pygame (wyświetlanie okienka i rysowanie na nim).

### Reprezentacja sceny

Scena w programie jest zbiorem dwóch list – listy wierzchołków oraz listy krawędzi. Każdy wierzchołek zawiera cztery liczby: *x, y, z, w*, gdzie jego realne koordynaty w świecie to *x/w, y/w, z/w.*

Krawędzie natomiast to zbiór numerów wierzchołków z którymi połączony jest dany wierzchołek.

### Rzutowanie

W moim programie kamera znajduje się zawsze na początku układu współrzędnych, zatem macierz, za pomocą której rzutuję prezentuje się następująco:

$$\begin{bmatrix} \frac{1}{\frac{w}{h}\cdot\tan\frac{fov}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\frac{fov}{2}} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-f\cdot n}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Gdzie w – szerokość ekranu, h – wysokość ekranu, n = 0, f = 1.

### Operacje translacji

Jako że nasza kamera tak naprawdę się nie rusza, przy translacji odpowiednio przesuwamy wszystkie punkty korzystając z macierzy:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Przez tą macierz mnożymy wszystkie punkty, podstawiając pożądane przez nas wartości pod x, y, z, mając na uwadze to, że przesuwamy tak naprawdę punkty, czyli trzeba odwrócić tą wartość (jeśli chcemy przejść kamerą w prawo musimy przesunąć punkty w lewo itp.)

## Operacje obrotu

Tak samo jak przy translacji, operacje wykonujemy na punktach. W zależności od osi przez którą wykonujemy obrót mamy różne macierze:

$$\text{Oś } x: \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Oś } y: \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

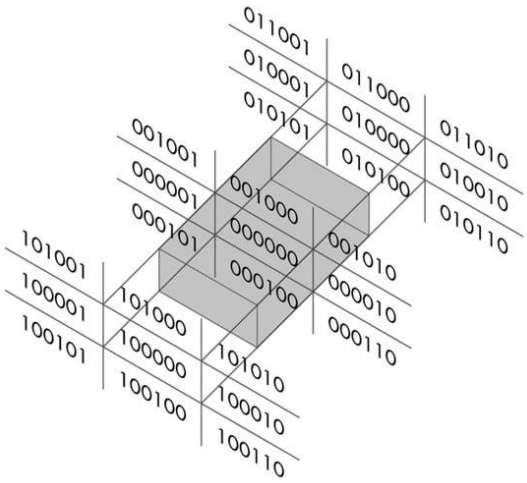
$$\text{Oś } z: \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Zoom

Zoom działa za pomocą manipulowania wartością fov. Przy zbliżaniu zmniejszam jej wartość, a przy oddalaniu zwiększam, pamiętając o tym żeby jej wartość znajdowała się w przedziale (0, 180).

## Przycinanie krawędzi

Przycinanie krawędzi wykonane jest za pomocą algorytm zbliżonego do algorytmu Cohena-Sutherlanda. Obsługuje on różne scenariusze korzystając z 6 cyfrowych kodów:



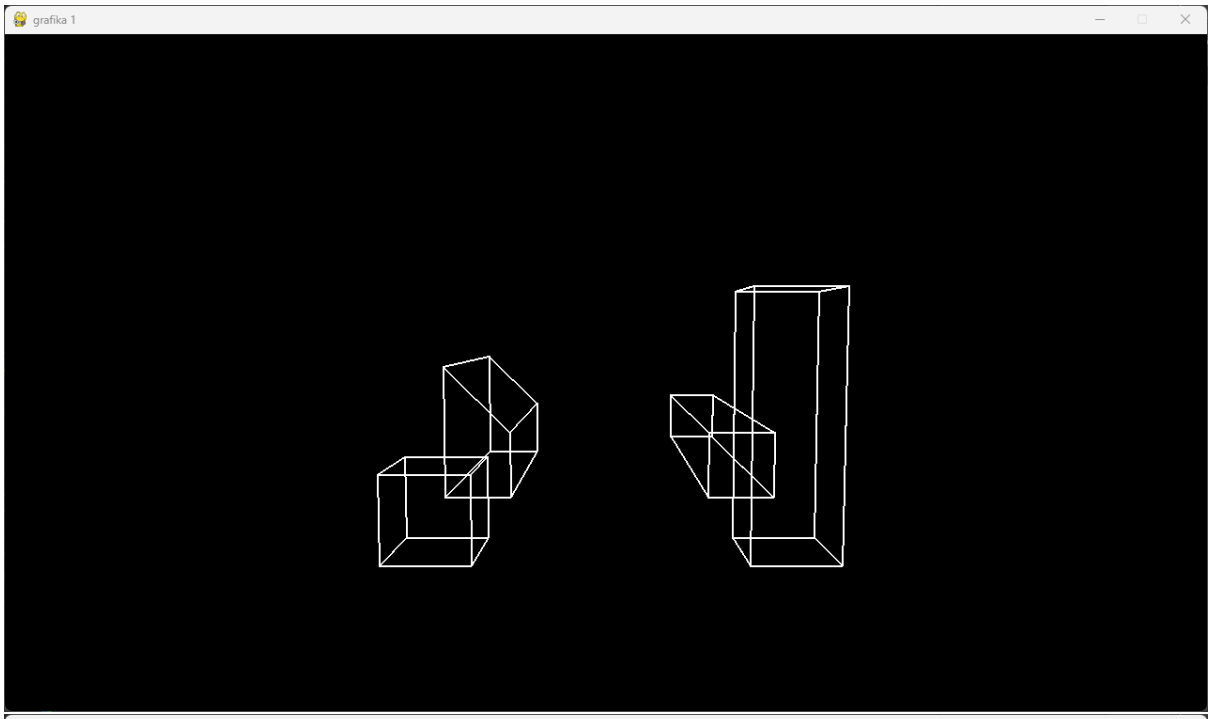

*https://www.mdpi.com/1999-4893/16/4/201*

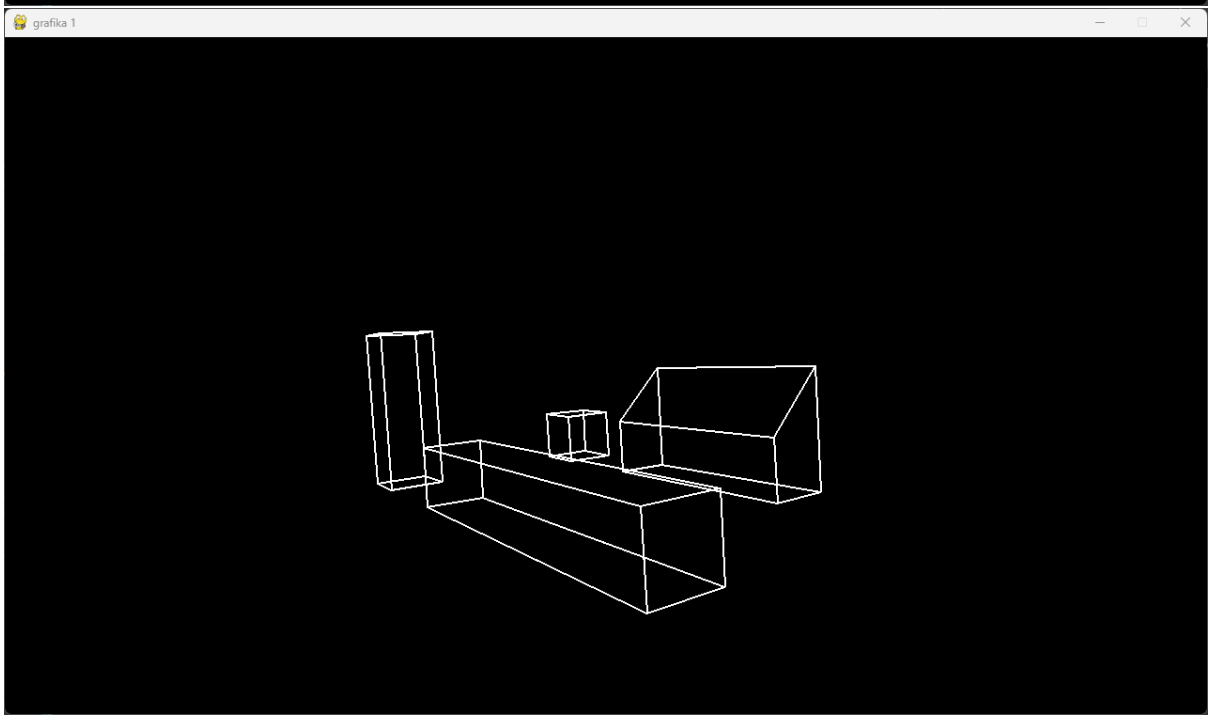
Dzięki temu możemy możemy w prosty sposób określić gdzie znajdują się punkty w relacji do kamery i czy trzeba przycinać krawędź. Są trzy podstawowe scenariusze:

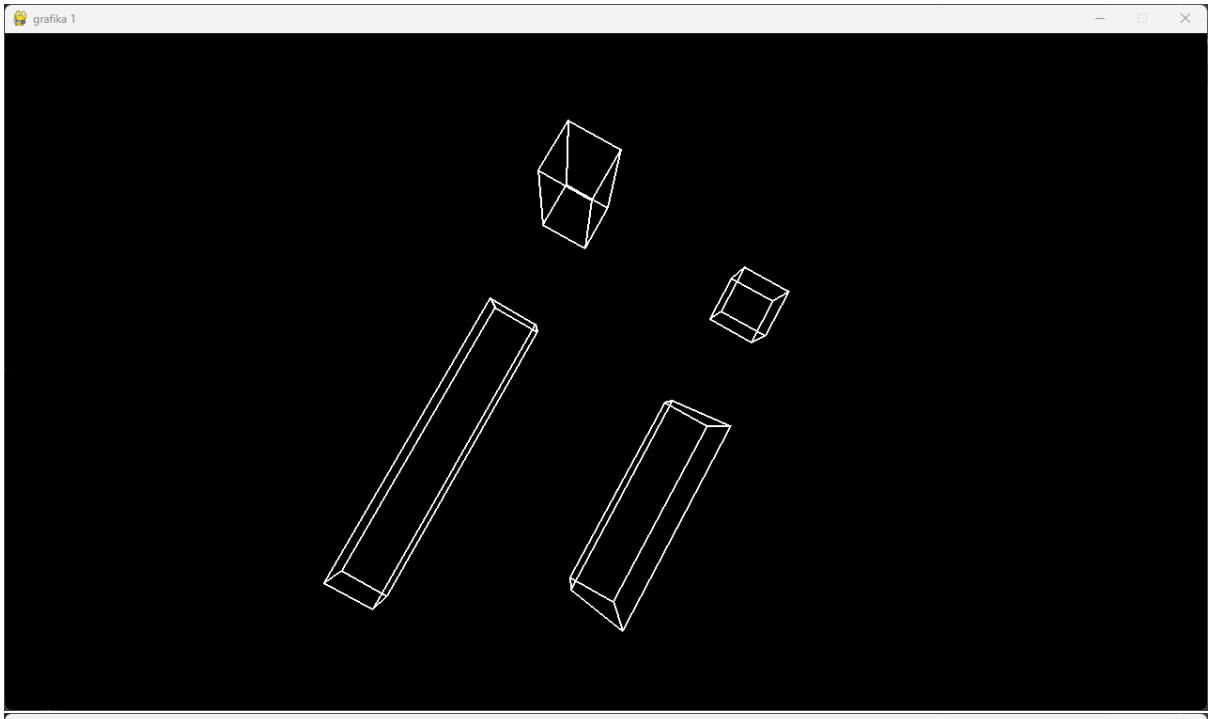
- Cała krawędź jest widoczna – jeśli oba punkty mają kod 000000
- Cała krawędź jest niewidoczna – jeśli AND kodów punktów nie jest równy 000000
- Krawędź wymaga przycięcia – jeśli AND kodów punktów wynosi 000000

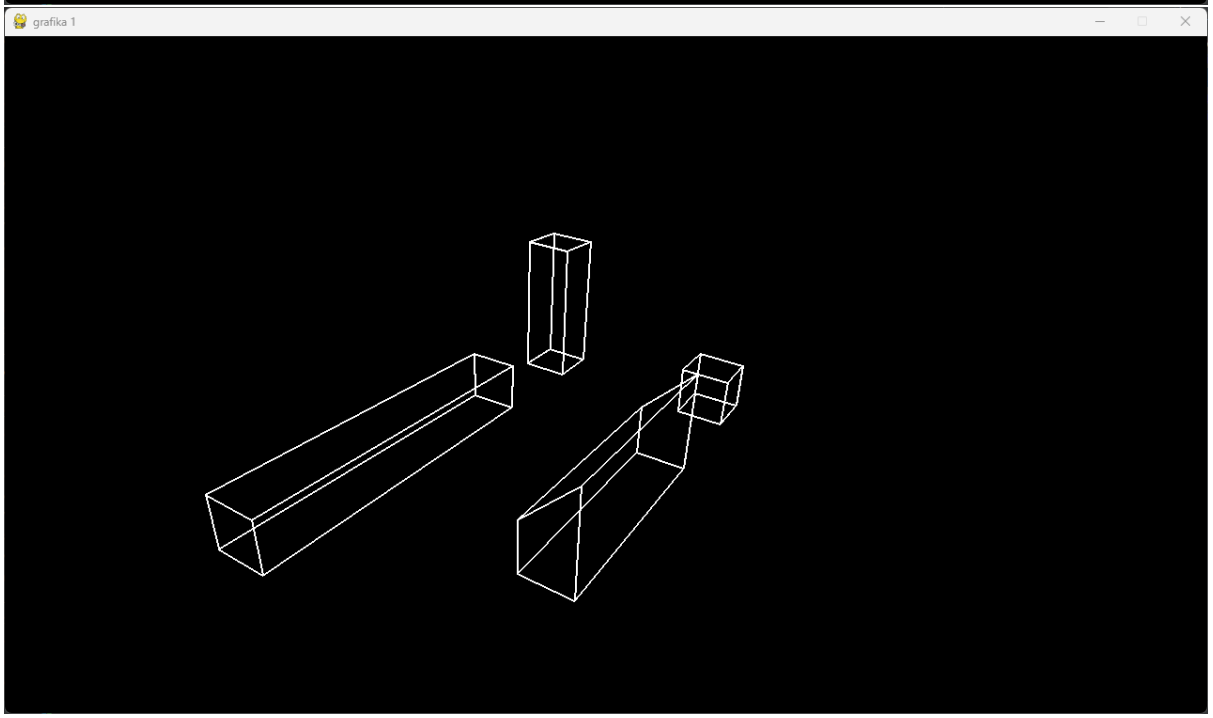
# Efekt końcowy i testy

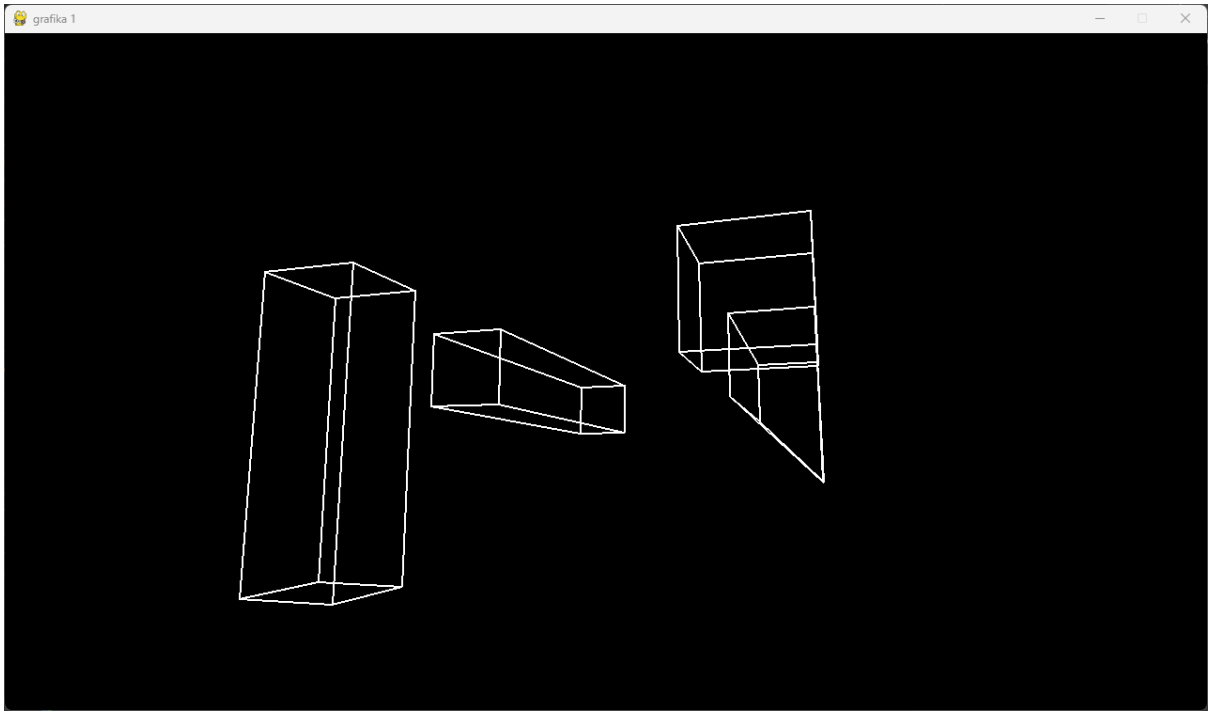
## Prezentacja podstawowych operacji translacji i obrotu:



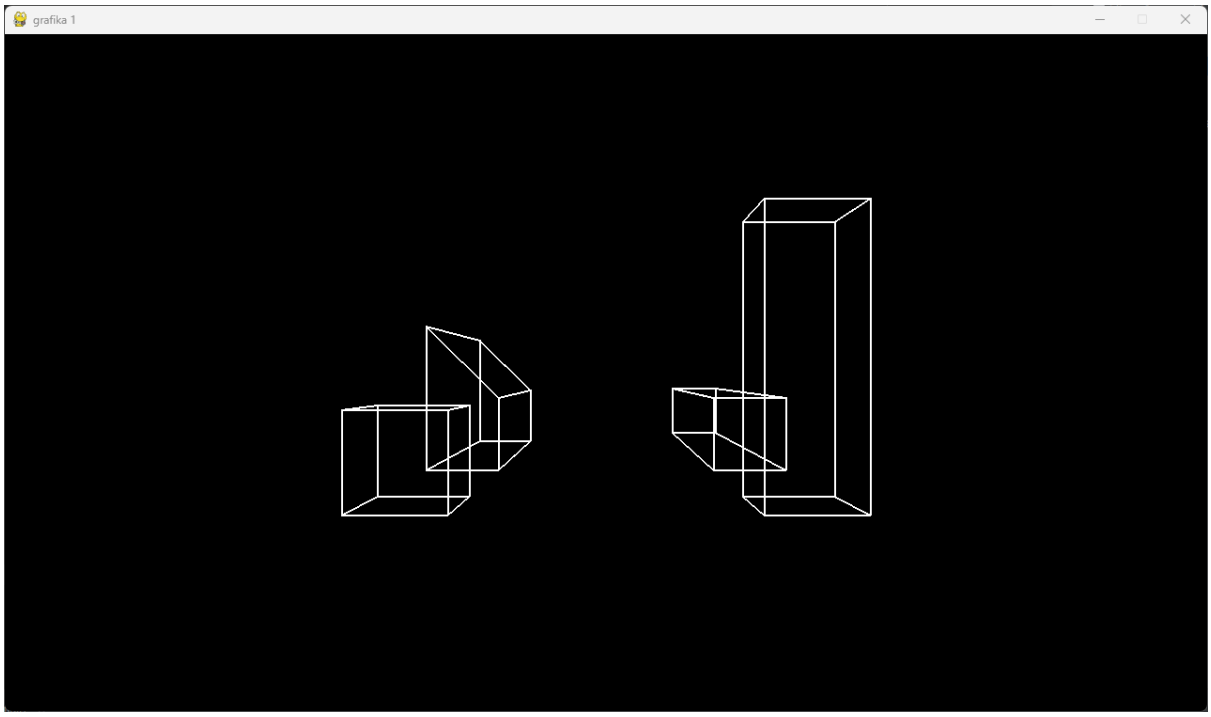

grafika 1


grafika 1

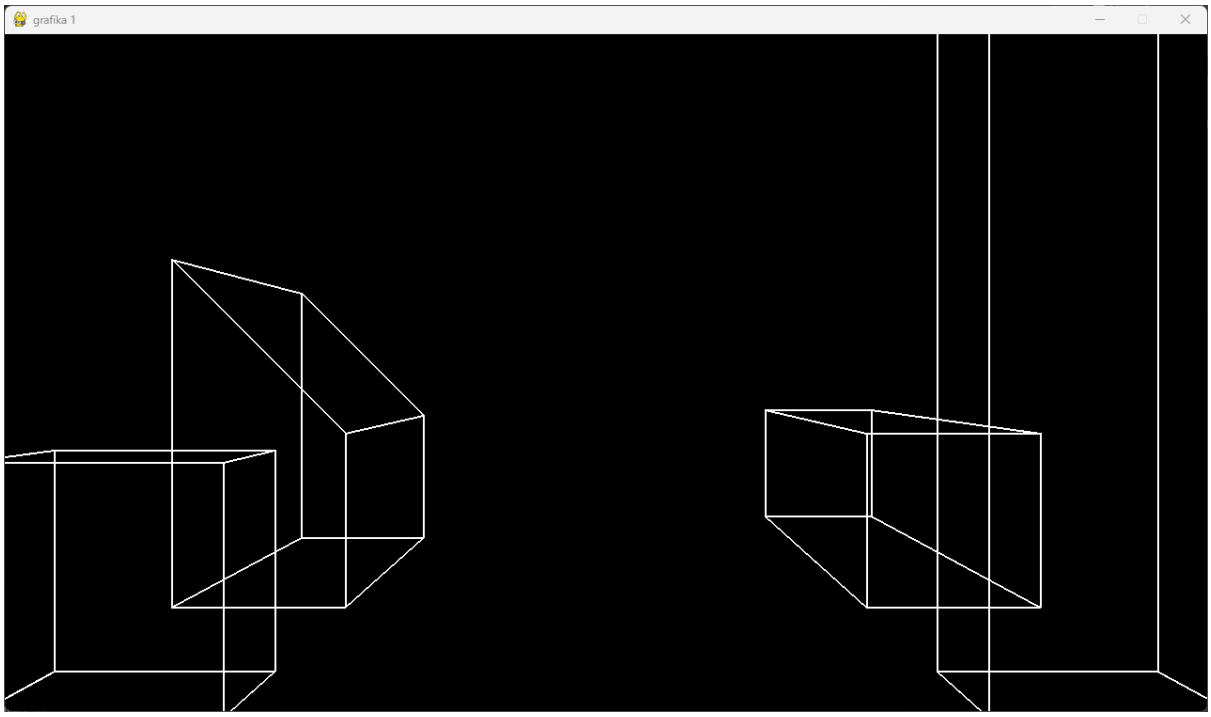



## Zoom

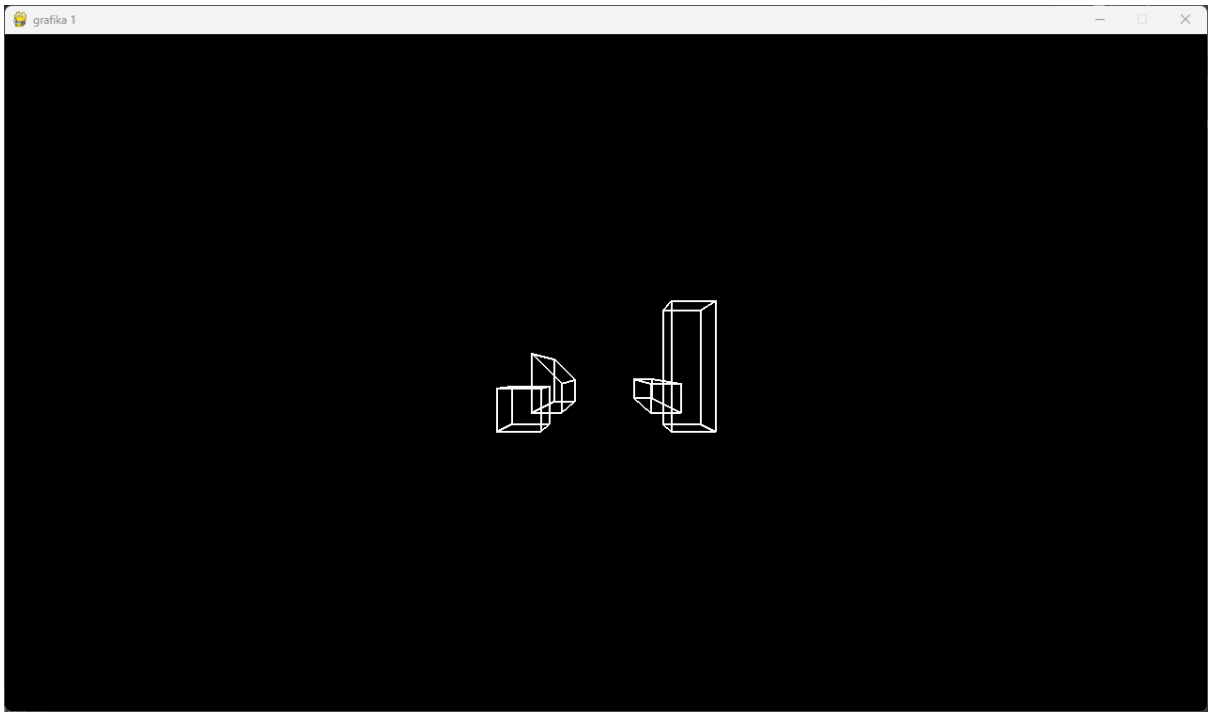
Bez zmienionej wartości (fov = 90):



Przybliżone (fov = 45):




Oddalone (fov = 135):



## Przycinanie krawędzi

Krawędzie rysowane są poprawnie, nawet w sytuacjach gdy „wlecimy do środka klocka” i widoczna na ekranie jest tylko część krawędzi.