

# Sprawozdanie do zadania projektowego nr 1: WIRTUALNA KAMERA

## I. Wirtualna kamera - koncepcja ogólna rozwiązania:

- kamera jest definiowana przez swój stan: pozycję w świecie, lokalną orientację i parametr ogniskowej (focal length).
- aby wyświetlić obiekty tak, jak je widzi kamera, dokonujemy transformacji świat→kamera (przesunięcie i obrót), a następnie projekcji perspektywicznej.
- obraz 2D, który widzimy, jest efektem tych przeliczeń: to tak, jakbyśmy patrzyli kamerą w danym miejscu w konkretną stronę, pod odpowiednim kątem
- używamy tylko wbudowanej biblioteki Canvas2D do rysowania
- Canvas2D jest natywną funkcjonalnością przeglądarki (żadnej dodatkowej instalacji czy komplikacji nie potrzeba).
- wszystkie obliczenia graficzne (ruch kamery i rzutowanie) są napisane samodzielnie w języku JavaScript
- kluczowe jest to, że cały pipeline 3D, czyli:
  - obliczanie macierzy orientacji,
  - mnożenie macierzy,
  - transpozycja
  - rzutowanie perspektywiczne,
  - a także obsługa kamery (przesunięcia lokalne, obroty) zostały zaimplementowane w kodzie ręcznie, bez używania gotowych bibliotek 3D (np. trójwymiarowych silników graficznych)

## II. Struktura plików zadania:

- index.html - plik HTML odpowiedzialny za renderowanie i strukturę sceny w przeglądarce
- style.css - arkusz stylów, odpowiedzialny za formatowanie (kolorystyka, czcionka, przyciski etc.) poszczególnych elementów sceny
- kamera.js - właściwy program implementujący wirtualną kamerę, napisany w języku JavaScript

## III. Koncepcja rozwiązania problemu - krok po kroku:

### 1. Transformacja „Świat → Kamera”

Aby narysować coś z perspektywy kamery, musimy najpierw powiedzieć, gdzie dany punkt jest względem kamery, dlatego:

- kod najpierw „przesuwa” każdy punkt tak, by kamera znalazła się w punkcie (0,0,0).
- potem „obracia” ten punkt, tak by osi kamery była zgodna ze standardową osią (czyli kamera „patrzy” wzduż osi Z)

Matematycznie:

1. przesunięcie:  $(x, y, z) \mapsto (x - x_{cam}, y - y_{cam}, z - z_{cam})$
2. obrót przez odwrotność (transpozycję) macierzy kamery.

W efekcie dostajemy współrzędne każdego obiektu w układzie kamery, czyli tak, jak „widzi” go kamera.

## 2. Rzutowanie perspektywiczne

Po tej transformacji wiemy, gdzie punkt leży względem kamery, np. `camX`, `camY`, `camZ`:

- jeżeli `camZ` jest za kamerą (np.  $\leq 0$ ), to w prostym modelu nie rysujemy takiego punktu, bo go „nie widzimy”.
- jeżeli `camZ > 0`, to kod oblicza, w którym miejscu płaszczyzny rzutowania (ekranu) ma się on znaleźć.
- prosty wzór:

$$x_{\text{projekcja}} = \frac{\text{ogniskowa} \times x_{camera}}{z_{camera}}, \quad y_{\text{projekcja}} = \frac{\text{ogniskowa} \times y_{camera}}{z_{camera}}$$

- im większa ogniskowa, tym mniej widać perspektywę „rozszerzającą się” na boki; można to porównać do zmiany obiektywu w aparacie.

## 3. Zamiana współrzędnych „projekcji” na piksele

- współrzędna  $(0, 0)$  w układzie kamery to środek „ekranu”, więc przesuwamy tę projekcję, żeby faktycznie w środku okna graficznego (`canvasu`) mieć  $(0, 0)$ :

$$\begin{aligned} x_{\text{Ekran}} &= (\text{szerokość\_canvasu} / 2) + x_{\text{Projekcja}} \\ y_{\text{Ekran}} &= (\text{wysokość\_canvasu} / 2) - y_{\text{Projekcja}} \end{aligned}$$

(odejmujemy `yProjekcja`, bo w HTML5-Canvas2D osią `Y` rośnie w dół, a w matematyce rośnie w górę).

- w kodzie przekształcenie to jest realizowane przez funkcję: `naEkran(pt, canvas)`

## 4. Rysowanie linii

- każda linia sceny to krawędź rysowana pomiędzy dwoma punktami:  $(p1, p2)$ .
- kod oblicza transformację (świat  $\rightarrow$  kamera) i następnie rzutowanie perspektywiczne dla obu końców.
- jeżeli oba punkty wypadają przed kamerą (tj. mają  $z > 0$  po transformacji), to rysujemy linię między ich współrzędnymi na ekranie.

## 5. Sterowanie (klawiatura i mysz)

- `W, S, A, D, Q, E`  $\rightarrow$  przesuwają kamerę w jej lokalnym układzie. Przykładowo `W` to idź do przodu  $(0, 0, +\text{krok})$  względem aktualnego kierunku, w jakim patrzy kamera.
- strzałki w górę (), w dół () , w lewo () i w prawo () obracają kamerę wokół jej lokalnych osi (`X`, `Y`), klawisze `R, F`  $\rightarrow$  obracają kamerę wokół lokalnej osi `Z`
- `=, -`  $\rightarrow$  zmieniają ogniskową (zoom).

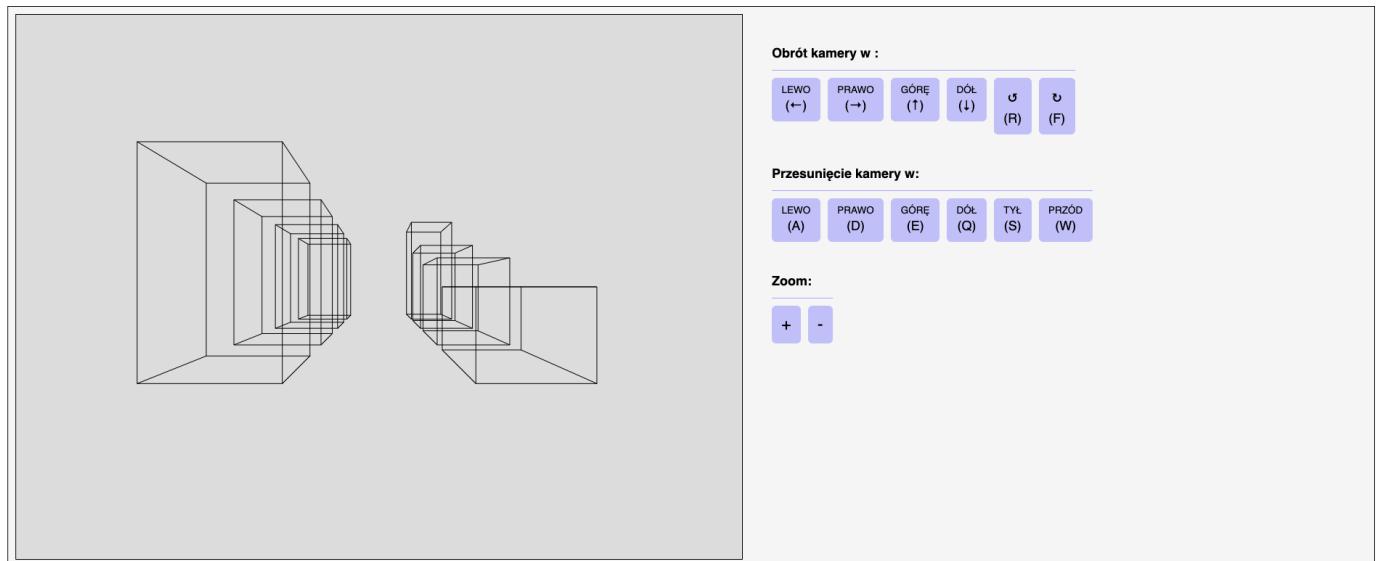
Za każdym razem, gdy zmieniamy położenie lub orientację kamery, wywołujemy `rysujScenę()`, żeby na nowo przeliczyć pozycje punktów i narysować scenę z nowej perspektywy.

Dodatkowo interfejs programu udostępnia przyciski, zlokalizowane po prawej stronie ekranu, reagujące na kliknięcia myszą, które umożliwiają wymienione wyżej operacje kamery za pomocą myszy.

## VI. Analiza rezultatów - testowanie kamery

Do przeprowadzenie testów poprawności działania kamery, zaprojektowałem scenę, która składa się z 8 prostopadłościanów. Dla lepszego zobrazowania efektów, cztery bryły mają jednakową wielkość i są umieszczone z lewej strony osi Z kamery. Pozostałe zaś, znajdujące się po jej prawej stronie, różnią się od siebie wysokością i są uszeregowane od najmniejszego do największego (zgodnie z kierunkiem patrzenia)[Rys1]

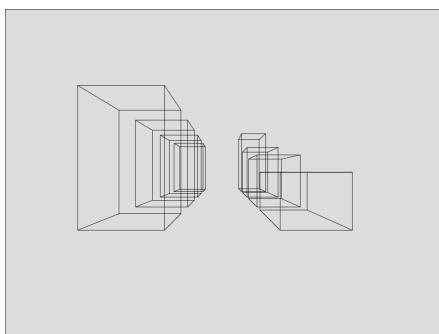
Zadanie 1: Wirtualna Kamera – Grafika Komputerowa i Wizualizacja (OKNO)



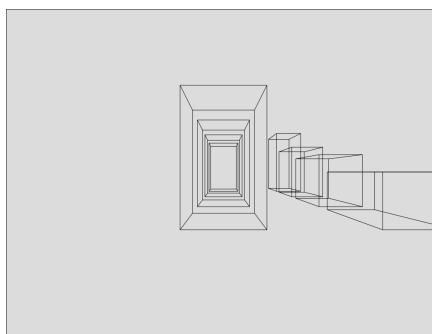
Rys1. Główne okno programu

### 1. Test translacji kamery

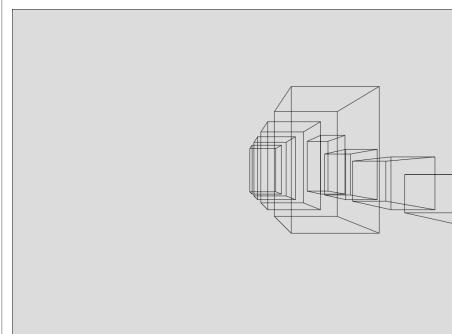
- ustawiam kamerę naprzeciw budynków i wciskam kilkukrotnie odpowiednie klawisze sterowania (A, D, Q, E, W, S):
  - translacja kamery w **lewą** stronę (dwukrotnie wciśnięty klawisz A):



ekran początkowy

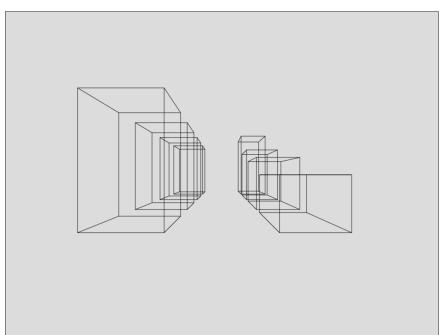


translacja w lewo (A)

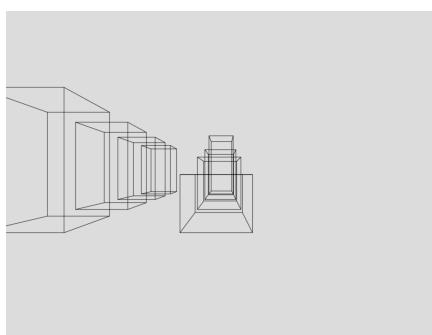


translacja w lewo (A)

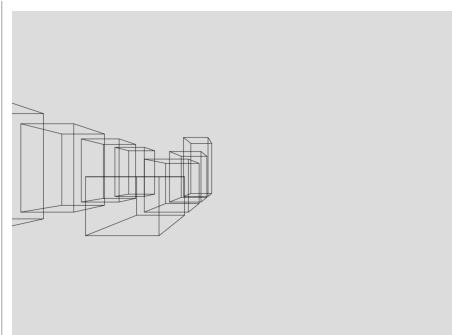
- translacja kamery w **prawą** stronę (dwukrotnie wcisnięty klawisz **D**):



ekran początkowy

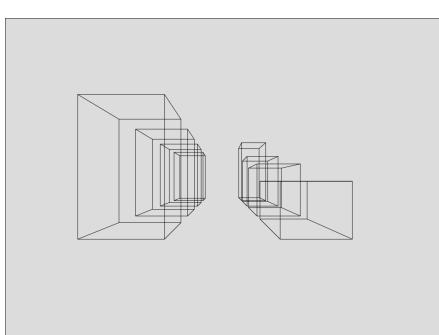


translacja w prawo (D)

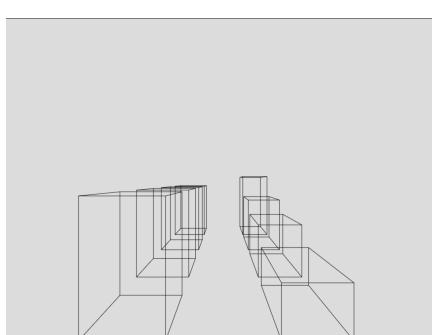


translacja w prawo (D)

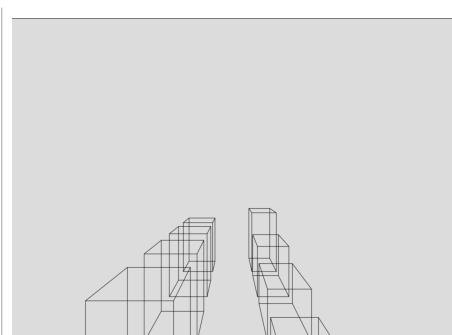
- translacja kamery w **góre** (dwukrotne wcisnięcie klawisza **Q**)



ekran początkowy

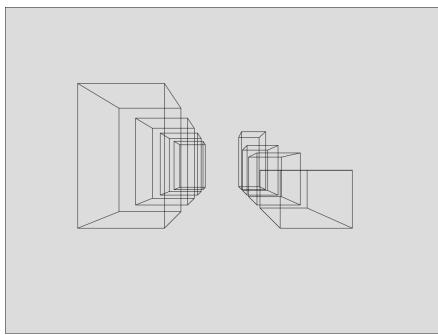


translacja do góry (Q)

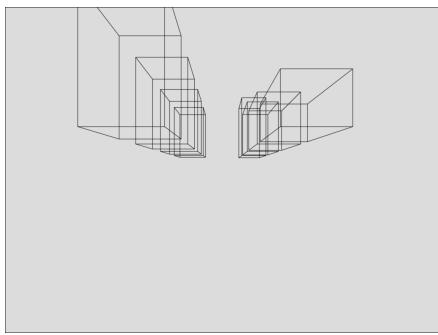


translacja do góry (Q)

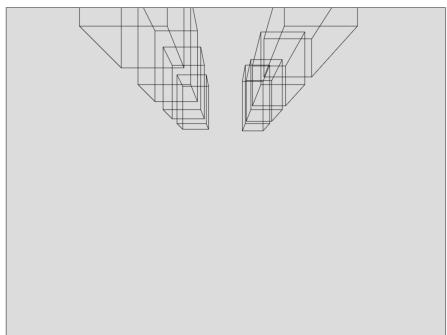
- translacja kamery w **dół** (dwukrotne naciśnięcie klawisza **E**):



ekran początkowy

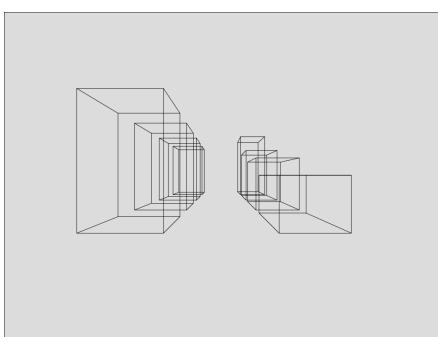


translacja do dołu (E)

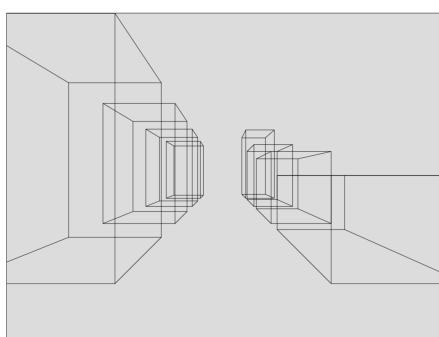


translacja do dołu (E)

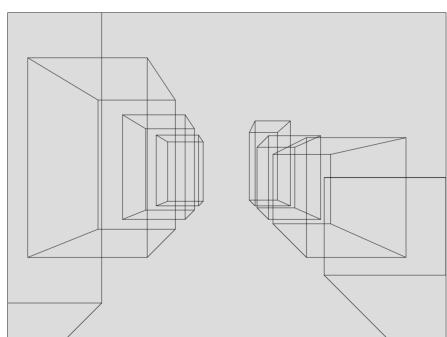
- translacja kamery do **przodu** (dwukrotne naciśnięcie klawisza **W**):



ekran początkowy

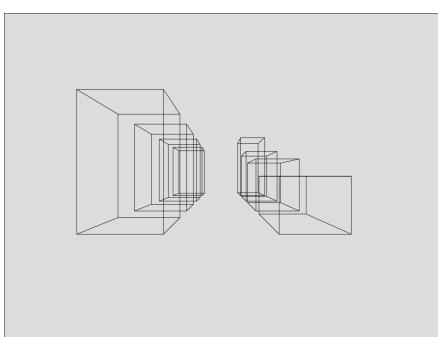


translacja do przodu (W)

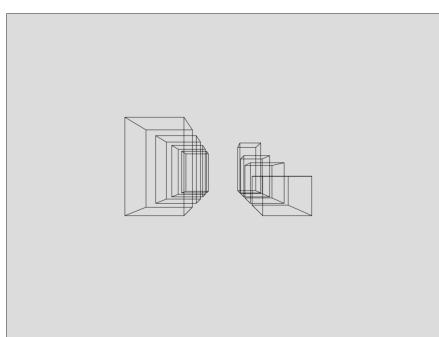


translacja do przodu (W)

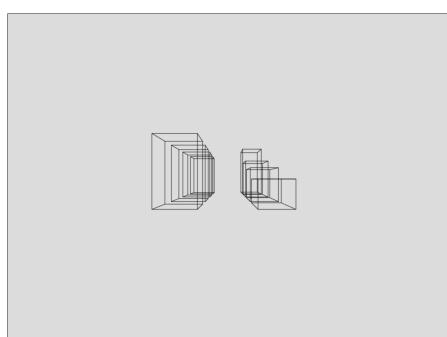
- translacja kamery do **tyłu** (dwukrotne naciśnięcie klawisza **S**):



ekran początkowy



translacja do tyłu (S)



translacja do tyłu (S)

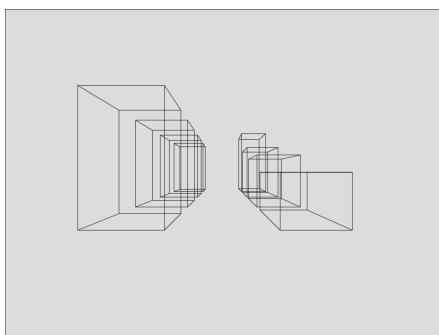
Oczekiwane wyniki przeprowadzonych operacji translacji:

Oczekiwanie	Rezultat
obraz powinien przesuwać się w odpowiednim kierunku	✓
budynki bliższe powinny przemieszczać się szybciej niż dalsze (efekt paralaksy)	✓
poruszenie się do przodu powinno powodować powiększanie się budynków (wizualne)	✓
poruszanie się do tyłu powinno powodować zmniejszanie się budynków (wizualnie)	✓

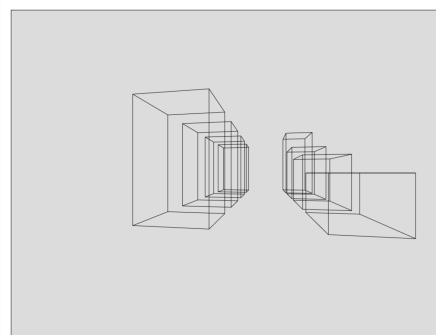
**Wniosek:** operacje translacji kamery generują obraz zgodnie z założeniami.

## 2. Test rotacji kamery

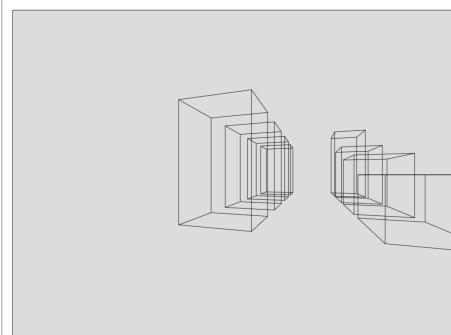
- ustawiam kamerę naprzeciw budynków i wciskam kilkukrotnie odpowiednie klawisze sterowania (, , , , R, F):
- rotacja kamery w **lewo** tzn. wokół osi Y (dwukrotne naciśnięcie klawisza ):



ekran początkowy

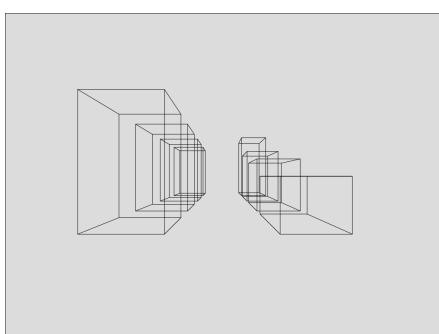


rotacja w lewo

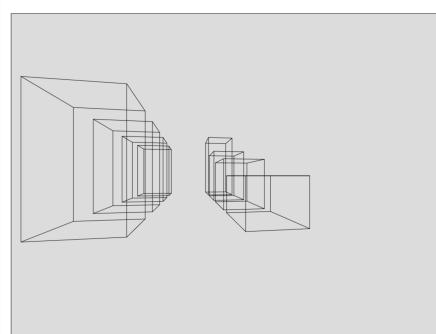


rotacja w lewo

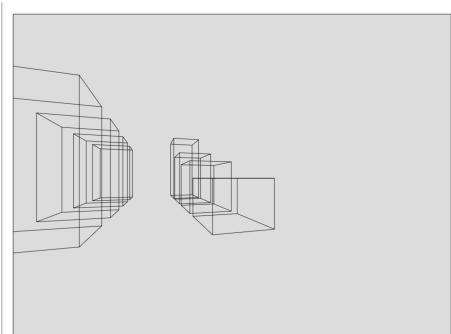
- rotacja kamery w **prawo** tzn. wokół osi Y (dwukrotne naciśnięcie klawisza ):



ekran początkowy



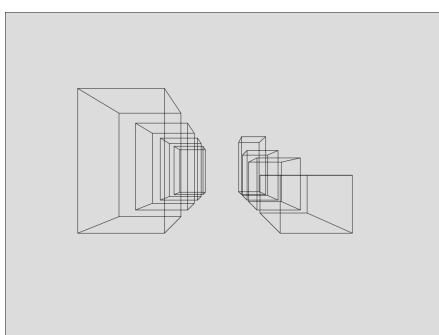
rotacja w prawo



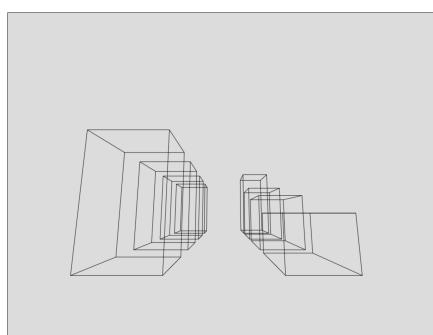
rotacja w prawo

- rotacja kamery do **góry** tzn. w lewo wokół osi X (dwukrotne naciśnięcie klawisza ):

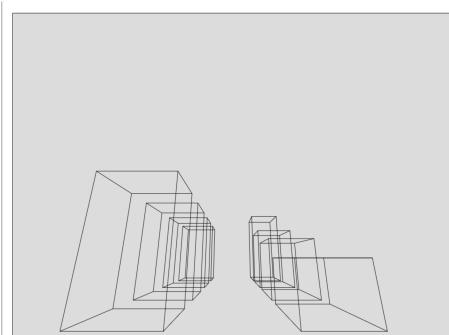
- 



ekran początkowy

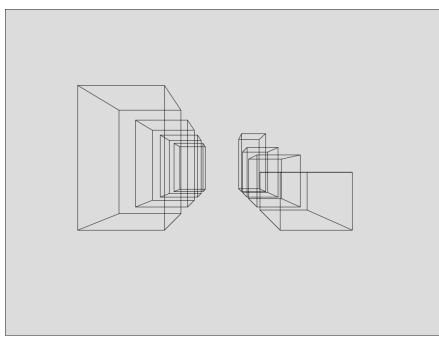


rotacja w góre

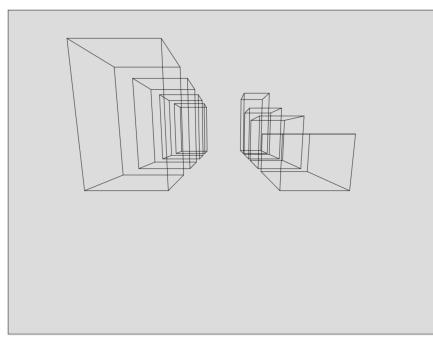


rotacja w góre

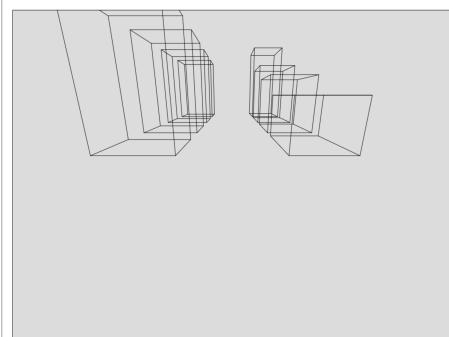
- rotacja kamery do **dół** tzn. w prawo wokół osi X (dwukrotne naciśnięcie klawisza ):



ekran początkowy

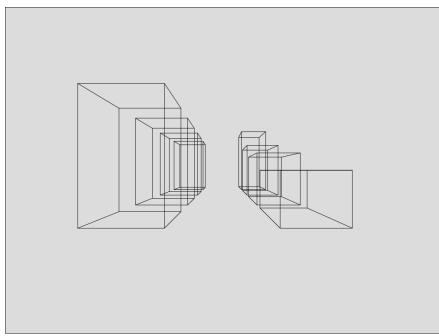


rotacja w dół

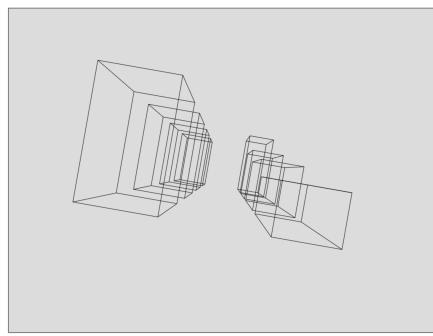


rotacja w dół

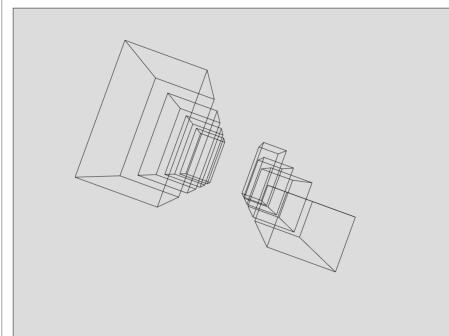
- rotacja kamery w kierunku **przeciwnym do ruchu wskazówek zegara** tzn. w lewo wokół osi Z (dwukrotne naciśnięcie klawisza **R**):



ekran początkowy

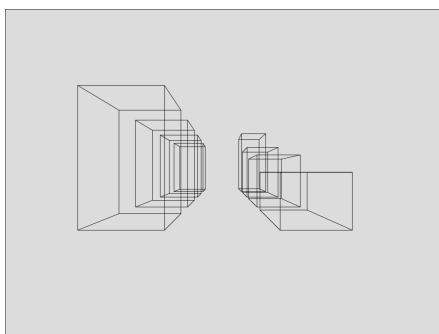


rotacja w lewo (R)

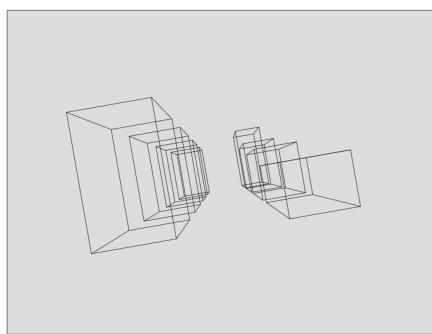


rotacja w lewo (R)

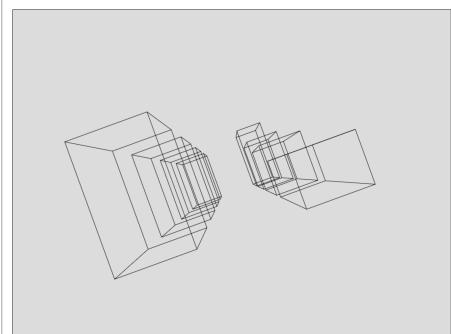
- rotacja kamery w kierunku **zgodnym do ruchu wskazówek zegara** tzn. w prawo wokół osi Z (dwukrotne naciśnięcie klawisza **F**):



ekran początkowy



rotacja w prawo (F)



rotacja w prawo (F)

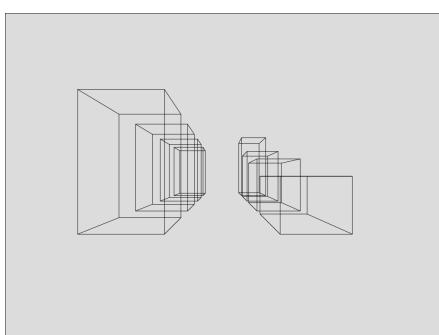
Oczekiwane wyniki przeprowadzonych operacji rotacji:

Oczekiwanie	Rezultat
obraz powinien przesuwać się w odpowiednim kierunku	✓
obrót w poziomie (osi Y) powinien zmieniać widok z lewej na prawą stronę ulicy	✓
obrót w pionie (osi X) powinien umożliwić patrzenie „do góry” i „w dół”	✓
obrót wokół osi Z (roll) powinien powodować przechylenie obrazu	✓

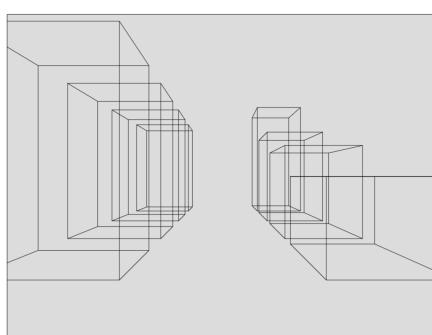
**Wnioski:** operacje rotacji kamery generują obraz zgodnie z założeniami; pewne zastrzeżenia budzi przechylanie widocznych krawędzi i ścian budynków podczas rotacji wokół osi Y i X

### 3. Test ogniskowej (zoom)

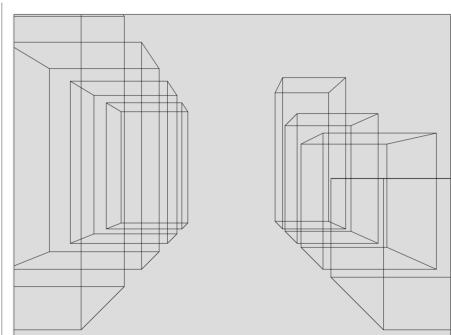
- ustawiam kamerę skierowaną w stronę budynków
- zmieniam ogniskową klawiszami +, -
- zwiększam** ogniskową (osmiokrotne naciśnięcie klawisza +):



ekran początkowy

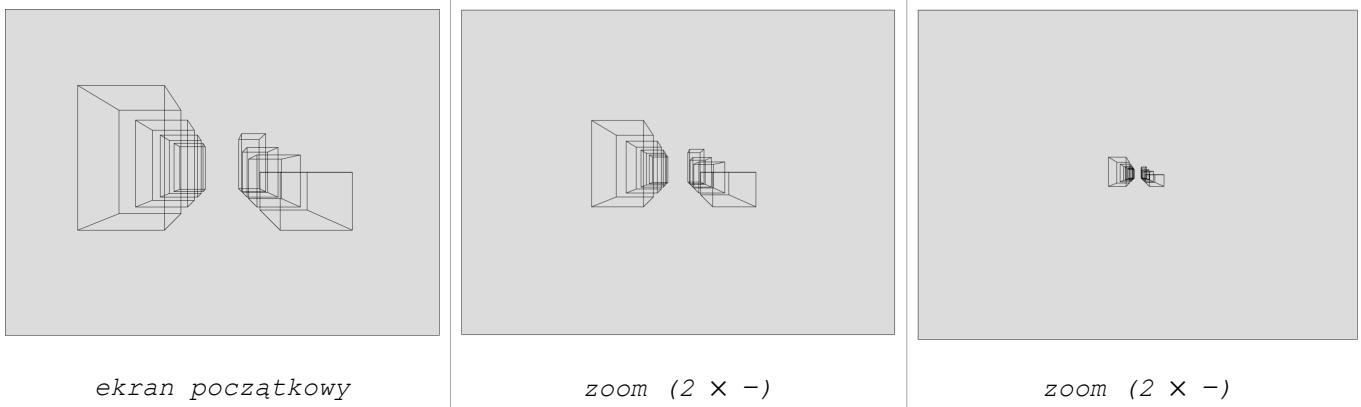


zoom (4 x +)



zoom (4 x +)

- **zmniejszam** ogniskową kamery (naciskając czterokrotnie klawisz -):



Oczekiwane wyniki testów zmiany ogniskowej:

Oczekiwanie	Rezultat
zwiększanie ogniskowej powinno dawać efekt przybliżenia (jak lornetka lub teleobiektyw).	✓
zmniejszanie ogniskowej powinno poszerzać pole widzenia, dając efekt obiektywu szerokokątnego.	✓

**Wnioski:** kamera zachowuje się prawidłowo, zgodnie z założeniami; efekt zmiany ogniskowej jest wizualnie bardzo podobny do zmiany położenia kamery poprzez jej translację w tył (S) i przód (W), ale dostrzec można subtelne lecz istotne różnice:

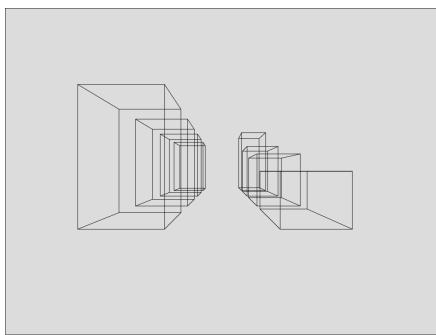
Cecha	Translacja (W/S)	Zmiana ogniskowej
kamera się porusza?	✓ (tak)	✗ (nie)
zmienia się perspektywa?	✓ (tak)	✗ (nie)
efekt paralaksy (względny ruch obiektów)?	✓ (tak)	✗ (nie)
obiekty zbliżają się równo?	✗ zależy od głębokości	✓ równomiernie
punkt zbiegu się zmienia?	✓ (tak)	✗ (nie)

#### 4. Test transformacji lokalnej vs globalnej (zwany także testem prof. Sawickiego)

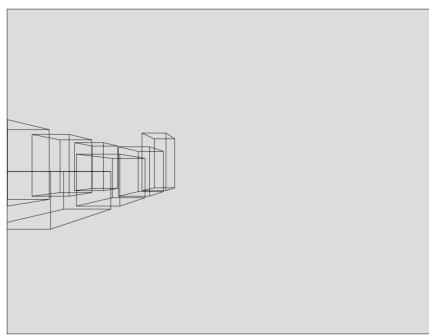
Test pozwala sprawdzić, czy kamera pomimo jej przesuwanie w przestrzeni świata (transformacja globalna) zachowuje się tak, jakby użytkownik był w niej osadzony (zawsze porusza się względem aktualnego kierunku widzenia). Badanie to przeprowadza się, wykonując wirtualny spacer po scenie w taki sposób, aby można było obejść widoczne na scenie obiekty dookoła i w wybranych momentach sprawdzić, czy kamera umożliwia niezmiennie dokonywanie

dowolnej operacji przekształcenia zachowując naturalne wrażenia ruchu z punktu widzenia użytkownika:

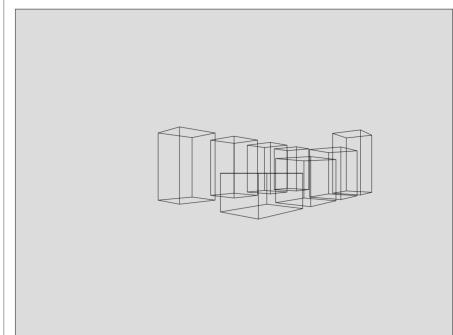
- umieszczać kamerę w punkcie startowym i dokonujemy potrójnej translacji kamery w **prawą** stronę (3 razy wciskamy klawisz **D**) a następnie wykonujemy trzykrotnie rotację kamery w **lewą** stronę (trzykrotne wcisnięcie klawisza to obrót o kąt 45 stopni wokół osi **Y**):



ekran początkowy



translacja (3 × D)

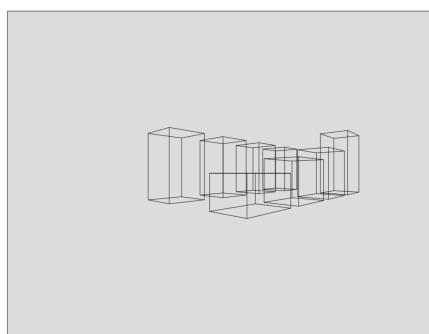


rotacja (3 × ←)

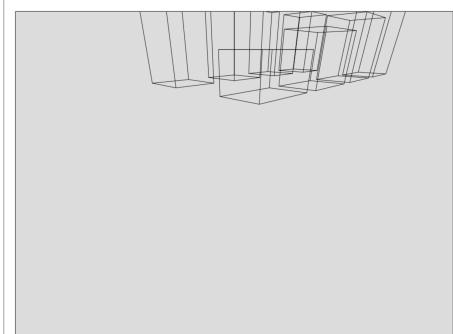
- sprawdzam poprawność zachowania kamery w transformacji lokalnej poprzez wykonanie rotacji kamery w **górę** i w **dół** (czyli wokół osi X kamery):



rotacja w góre

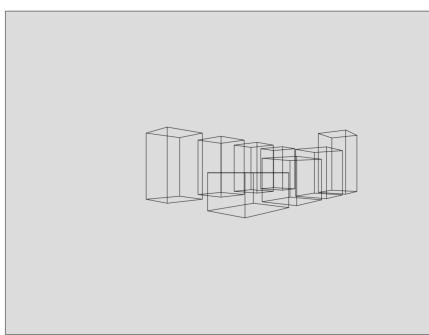


klatka wyjściowa

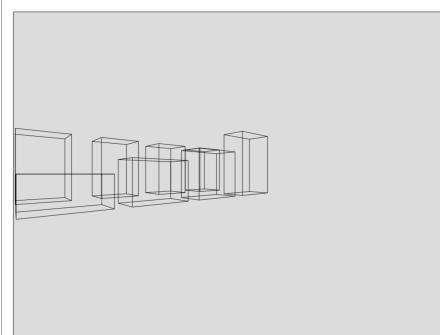
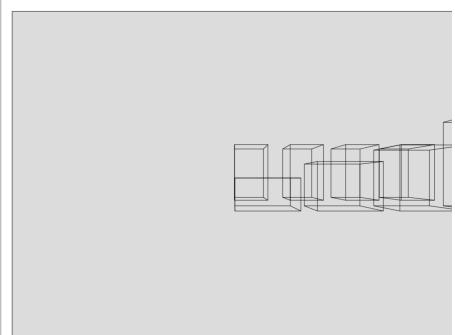


rotacja w dół

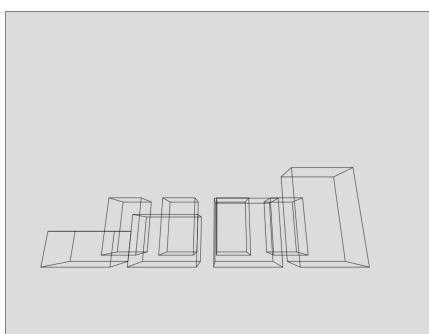
- kontynuuję wędrówkę po scenie - translacja kamery w **prawo** (3 × **D**) i rotacja kamery w **lewo** wokół osi **Y** (3 × ):



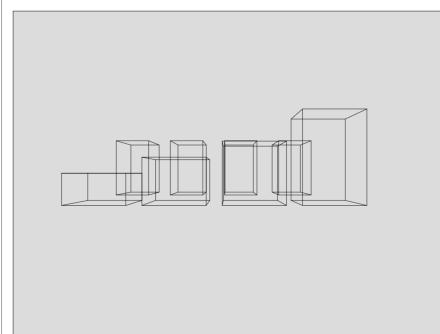
klatka wyjściowa

translacja ( $3 \times D$ )rotacja ( $3 \times \leftarrow$ )

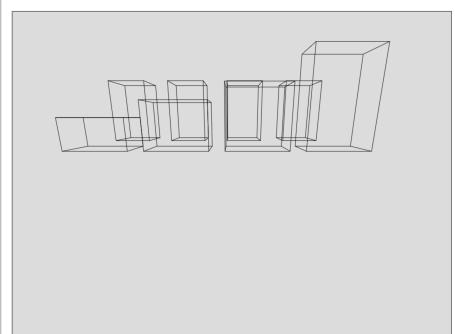
- po wykonanych operacjach kamerą, znajdujemy się z boku sceny i sprawdzamy jak zachowią się kamery, gdy zechcemy dokonać rotacji w **góre** i w **dół** (sprawdzamy czy zachowią lokalną swobodę ruchu), przesuniemy się jeszcze w **prawą** stronę ( $2 \times D$ ) aby wyśrodkować nieco obraz (klatka wyjściowa na poniższym rysunku):



rotacja w góre

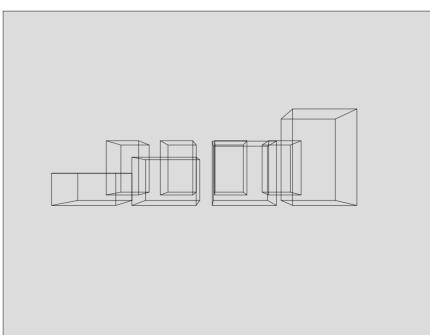


klatka wyjściowa

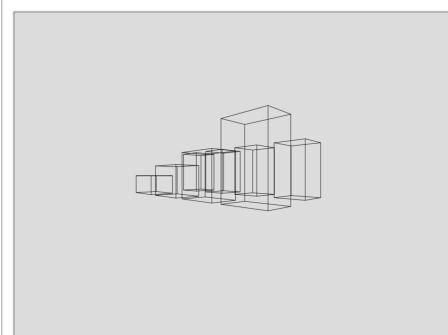


rotacja w dół

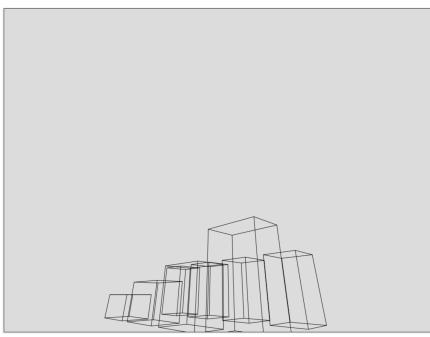
- aby przemieścić się do kolejnego punktu kontrolnego dokonujemy kolejnych serii translacji w **prawo** ( $3 \times D$ ), rotacji kamery **lewo** ( $3 \times \rightarrow$ ) na koniec translacji w **prawo** ( $2 \times D$ ), żeby wyśrodkować scenę:



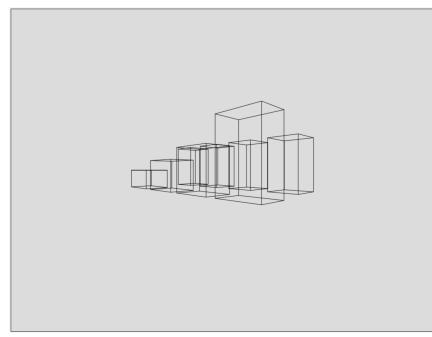
klatka wyjściowa

translacja  $3 \times D$   
rotacja ( $3 \times \leftarrow$ )translacja ( $2 \times D$ )

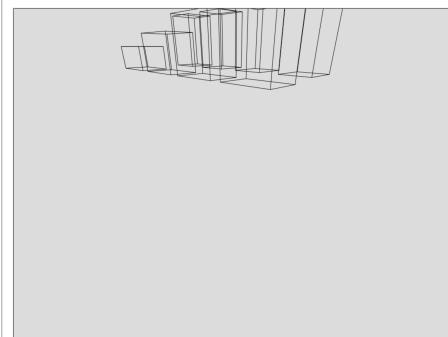
- ponownie wykonujemy rotację w **górę** i w **dół** (oś X kamery) dla sprawdzenia poprawności zachowania lokalności ruchu kamery:



rotacja w góre

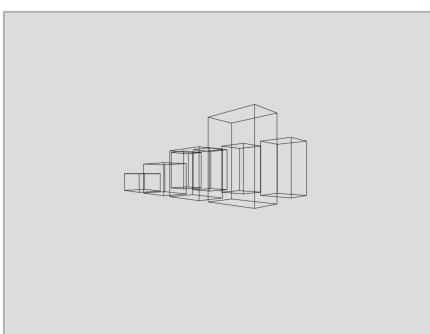


klatka wyjściowa

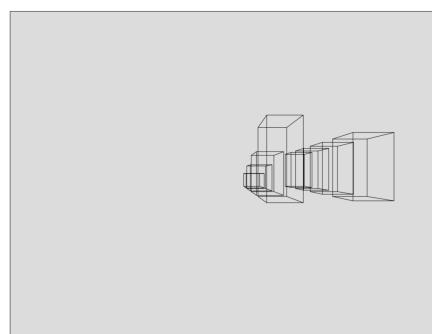
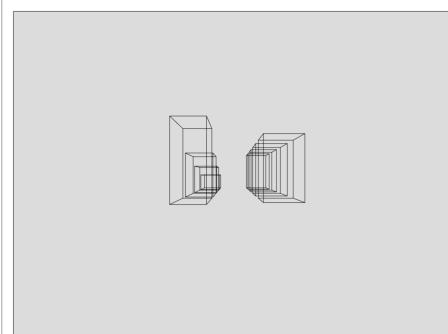


rotacja w dół

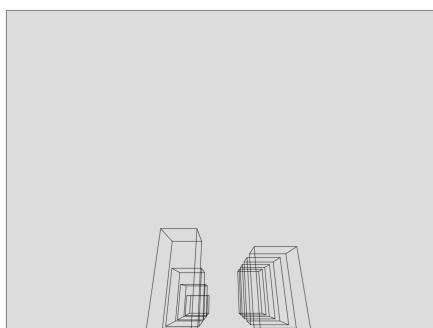
- aby przemieścić się do kolejnego punktu kontrolnego dokonujemy kolejnych serii translacji w **prawo** ( $3 \times D$ ), rotacji kamery **lewo** ( $3 \times \rightarrow$ ) na koniec translacji w **prawo** żeby wyśrodkować scenę ( $2 \times D$ ); znajdujemy się teraz “z tyłu” sceny, kaskadowe budynki widzimy teraz po lewej stronie:



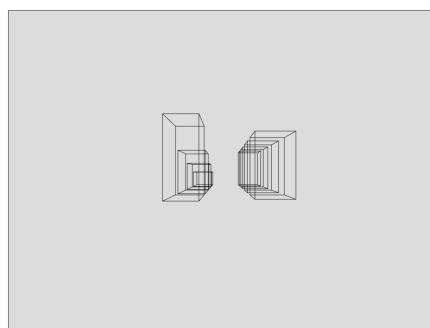
klatka wyjściowa

translacja  $3 \times D$   
rotacja ( $3 \times \rightarrow$ )translacja ( $2 \times D$ )

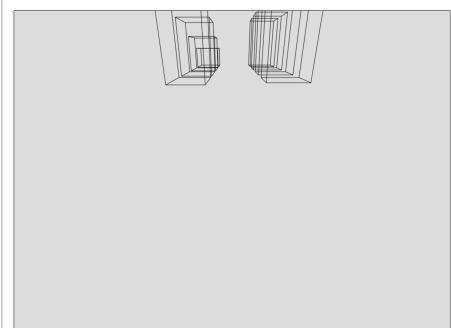
- ponownie wykonujemy rotację w **górę** i w **dół** (osi x kamery) dla sprawdzenia poprawności zachowania lokalności ruchu kamery:



rotacja w góre

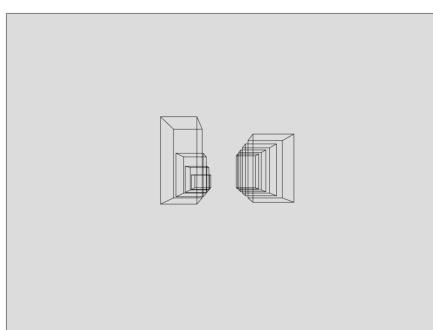


klatka wyjściowa

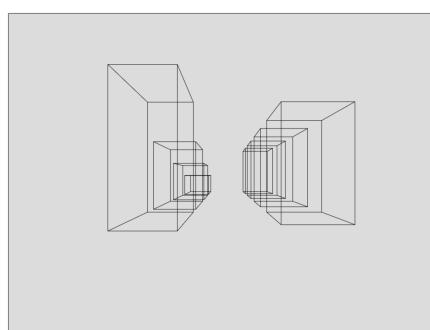
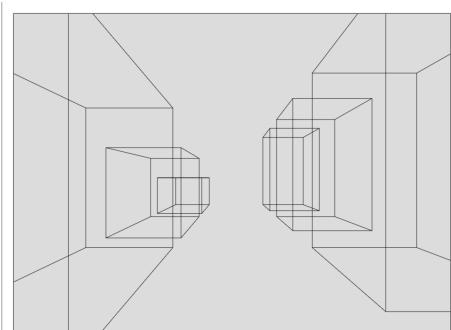


rotacja w dół

- na koniec, przesuwając kamerę do przodu (kilukrotne wcisnięcie klawisza W) sprawdę, czy kamera porusza się zgodnie z aktualnym kierunkiem swojej osi Z (czyli w kierunku patrzenia):



klatka wyjściowa

translacja  $2 \times W$ translacja  $2 \times W$ 

Podsumowując zachowanie kamery w teście:

Oczekiwanie	Rezultat
obraz powinien przesuwać się w odpowiednim kierunku	✓
kierunek "do przodu" (W) zależy od aktualnego kierunku patrzenia	✓
po obrocie (rotacji  ) kamera działa zgodnie z kierunkiem patrzenia	✓