# Grafika Komputerowa

Projekt 1 Marcin Maciąg

## Wstęp

Celem projektu było stworzenie **wirtualnej kamery**, która pozwoli nam w miarę możliwości naśladowanie naszej "ludzkiej" percepcji. Kamera powinna również mieć oczywiście możliwości podstawowego poruszania się w dowolnym kierunku(prawo, lewo, góra, dół, przód, tył) oraz obracanie kamery o zadany kąt. Ostatnią opcją będzie jeszcze możliwość zooma. Damy także możliwość, użytkownikowi na wybór punktów w których będą umieszczone obiekty. Podczas tworzenia takiej kamery zapoznamy się z podstawowymi wzorami związanymi z Grafiką komputerowy np. **translacji** czy właśnie **obrotu**.

## Kod

Całość kodu znajduje się na repozytorium GitHub:

https://github.com/MarvelousMarcin/camera

## Technologie

Do stworzenia aplikacji został wykorzystany podstawowy JavaScript wraz z biblioteką graficzną 3D Three.js

## Transformacje

Aby uprościć sobie implementację mamy tutaj do czynienia z jednym układem odniesienia aby nie musieć wykonywać wielu operacji wielokrotnie i analizować często skomplikowane kierunki w zależności od punktu obserwatora. Aby zapewnić zmianę położenia naszego obrazu dokonujemy transformacji na wszystkich punktach

jednocześnie przy pomocy mnożenia macierzowego. Wszystkie punkty przechowujemy w znormalizowanych wektorach:

$$P = [x, y, z, 1]$$

Jeżeli użytkownik chce przesunąć kamerę **w prawą stronę** to wystarczy że przesuniemy wszystkie obiekty **w lewą stronę** o stałą wartość. W ten sposób zasymulujmy ruch kamery w odpowiednią stronę. To samo tyczy się każdej innej operacji włącznie np. z obrotami. Jeżeli chcemy kamerę obrócić o 15 radianów to obracamy wszystkie elementy o -15 radianów.

**Translacja**

Aby dokonać translacji o zadany wektor musimy w zależności od kierunku przesunąć wszystkie punkty o daną wartość. Nowe punkty wyznaczamy za pomocą znormalizowanej operacji translacji:

$$\begin{pmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

Wykorzystanie tego wzoru daje nam możliwość na wykonanie zaplanowanych 6 ruchów w dowolnym kierunku.

**Rotacja**

Aby wykonać translację również musimy każdy z punktów niejako przesunąć tym razem o zadany stały kąt. Do obliczenia tego wykorzystamy takie wzory:

***Obrót o kąt alfa wokół osi OX:***

$$\begin{pmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

***Obrót o kąt alfa wokół osi OY:***

$$\begin{pmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & 0 & sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

***Obrót o kąt alfa wokół osi OZ:***

$$\begin{pmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

Dzięki tym operacjom jesteśmy w stanie obracać kamerą wokół dowolnej osi. Co daje nam finalnie pełen zakres ruchów naszej kamery.

**Zoom**

Ostatnim funkcją naszej kamery będzie możliwość zooma. Jest to operacja bardzo prosta do zaimplementowania bo będziemy po prostu sterować ogniskową co da nam coś na wzór oddalania oraz przybliżania. Co warto zauważyć nie zoom nie zmienia naszego położenia co a tylko zmienia nasze przybliżenia do obiektów co sprawia że nie zmieniają się obiekty które widzimy.

**Rzutowanie punktów 3D na 2D**

Przy pomocy odpowiedniego algorytmu naszą płaszczyznę trójwymiarową będziemy rzutować na płaszczyznę 2D. Każdy punkt który mamy musimy zamienić używając do tego ogniskowej, szerokości i długości ekranu oraz 3 współrzędnych.

Wyliczenie x:

$$x = focal/y * x + width/2$$

Wyliczenie y:

$$y = height/2 - focal/y * z$$

W metodzie liczącej weryfikujemy także czy wartość y czyli naszego przód tył nie jest przypadkiem ujemna ani równa zero ponieważ w tedy mamy dzielenie przez zero lub wartość współrzędnej jest nieskończona co skutkuje błędem. Aby temu zapobiec sprawdzamy czy tak właśnie nie jest a jeśli jest to ustawiamy wartość na bardzo bliską zero co da nam wrażenie jakby element został ucięty na naszym ekranie.

```
if(y <= 0)  y = 0.0001
```

## Uruchomienie aplikacji

Aby uruchomić program będziemy potrzebowali **node.js.** Aby sprawdzić czy go mamy wystarczy w terminalu wpisać:

```
node -v
```

Jeśli zobaczmy wersję programu to możemy kontynuować.

Klonujemy repozytorium na nasz pulpit a następnie instalujemy wszystkie pakiety:

```
npm i
```

A na koniec uruchamiam serwer:

```
npm run dev
```

Teraz w naszej przeglądarce pod adresem: **http://localhost:5173/** będziemy mieli dostęp do aplikacji

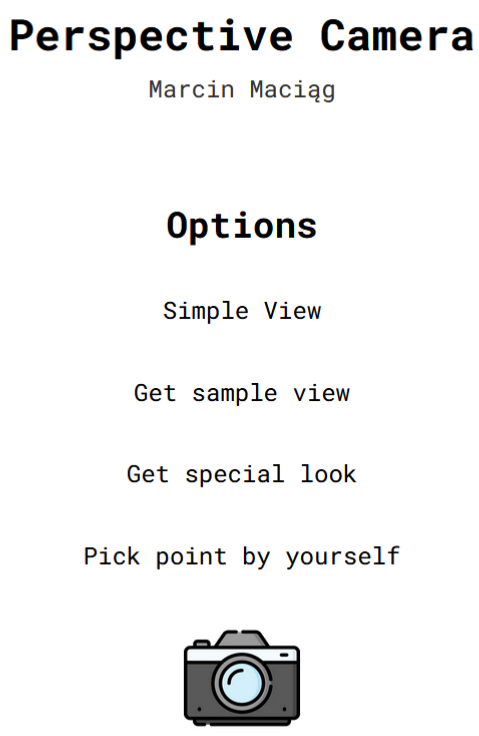
## Obsługa programu

- Po uruchomieniu wybieramy punkty w których chcielibyśmy umieścić elementy i klikamy przycisk **Start**
- Następnie przy pomocy odpowiednich klawiszy sterujemy kamerą

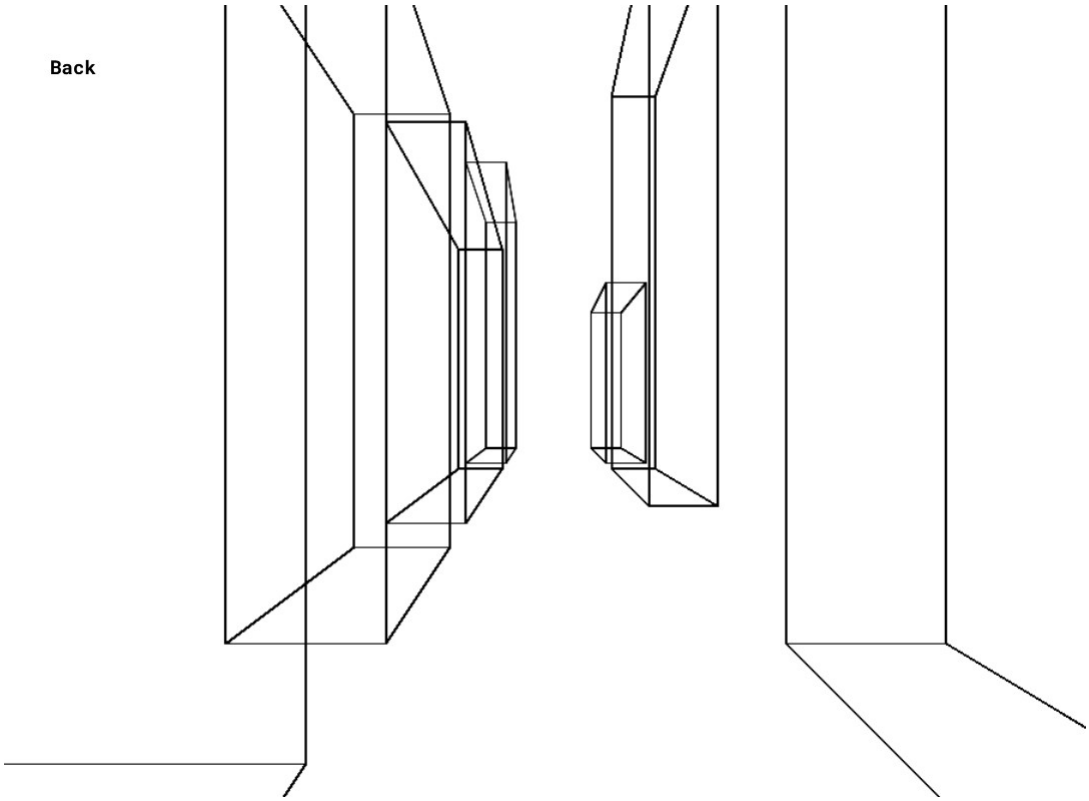
- **a,w,s,d** - translacja lewo, góra, dół, prawo
- **q,e** - translacja przód, tył
- **z, x** - zoom
- **j, l, j, k, o, u** - rotacje

## Przykłady użycia - Galeria

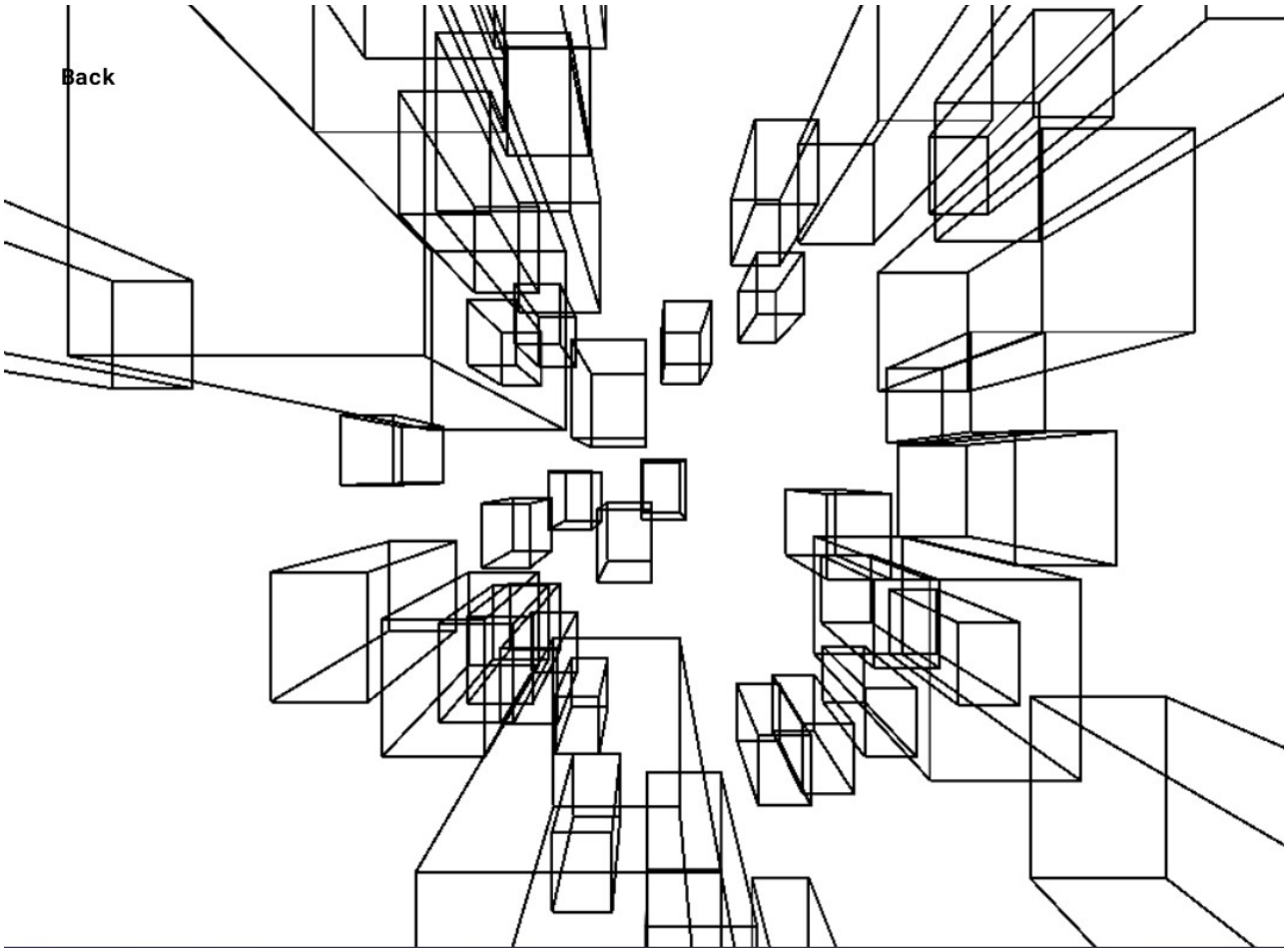
### Menu Główne



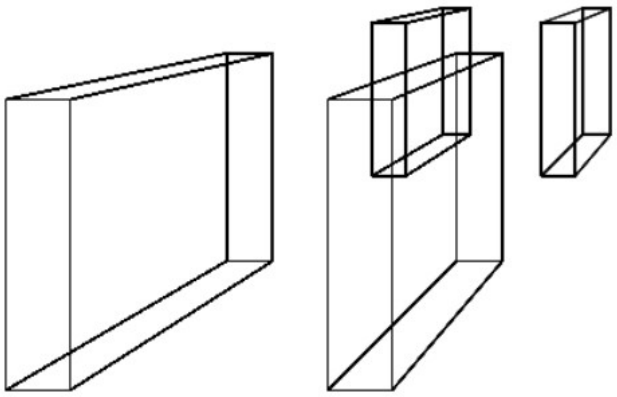

### Widok Miasta



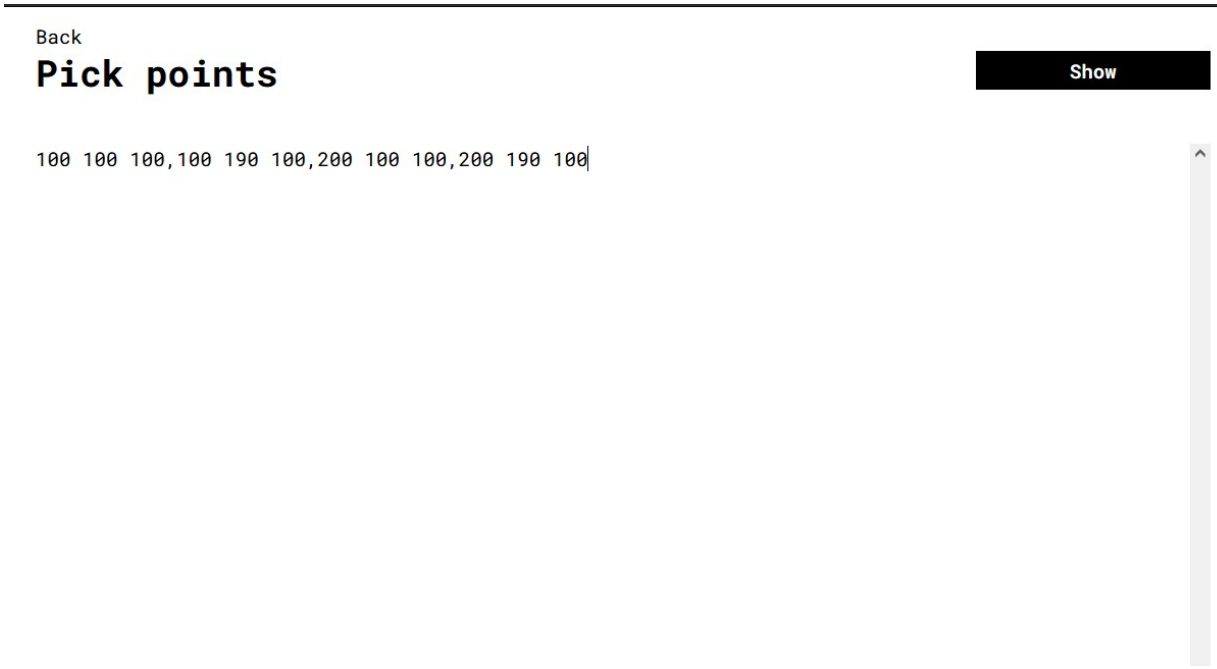

## Losowe Elementy




**Wygenerowane przez punkty użytkownika**



**Wybór punktów**




## Wnioski

Tworząc wirtualną kamerę poznałem podstawowe metody obliczeń grafiki komputerowej, umożliwiające tworzenie naprawdę ciekawych efektów. Translacja, obroty, zoom czy rzutowanie 3D na 2D, umożliwiając tworzenie na ekranie wszystkie co chcemy. Dodatkowo sposób jaki wybrałem na renderowanie okazał się być niestety mało wydajny, przez co ruch jest bardzo opóźniony i musimy czasami poczekać aż aplikacji dokona wszystkich obliczeń. Spowodowane jest to tym że musimy każdy punkt wymnażać przez odpowiednią macierz i jeszcze dodatkowo rzutować to wszystko na płaszczyznę, co przy większej ilości elementów sprawia że nawet nasze obecne pamięci obliczeniowe nie dają sobie z tym rady. Podsumowując mając do dyspozycji najprostsze płótno, technologię  oraz podstawowe algorytmt jesteśmy w stanie stworzyć prawdziwą wirtualną kamerę odzwierciedlającą naszą ludzką percepcję.

Marcin Maciąg