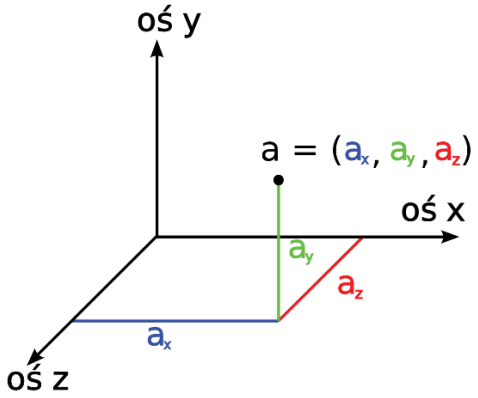
Cel i wizja projektu

Celem projektu jest implementacja wirtualnej kamery posiadającej możliwość poruszania się w przestrzeni trójwymiarowej, równolegle do osi XYZ własnego układu współrzędnych oraz obrotu wokół tych osi. Kamera ma posiadać możliwość przybliżania obrazu za pomocą zoom-u oraz robienia zdjęć obiektom. Przedstawionym obiektem w przestrzeni będzie osiem sześcianów ustawionych na kształt kostki Rubika 2x2 (tylko z większym odstępem między sześcianami). Każdy z sześcianów będzie się wyróżniał kolorem, dzięki czemu możliwe będzie sprawdzenie poprawności wykonywanych operacji. Zaplanowaną właściwością jest również ukrywanie krawędzi które znalazły by się za kamerą w momencie zbyt dużej translacji w stronę obiektu.

Podstawy teoretyczne projektu

Translacja – to po prostu dodanie do każdego punktu obiektu odpowiedniej wartości zależnej od kierunku w którym chcemy poruszyć obiekt (w rzeczywistości poruszamy obiektem, jednak wykorzystanie układu kamery i odwrócenia sterownia, sprawia wrażenie poruszania kamerą).

Układ współrzędnych prawoskrętny wykorzystany w projekcie wygląda tak:



Rys. 1 Układ opisujący położenie obiektów w przestrzeni

Obrót – wykonywany jest przez przemnożenie każdego punktu przez odpowiednią macierz obrotu. Macierze obrotu wyglądają tak jak poniżej:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

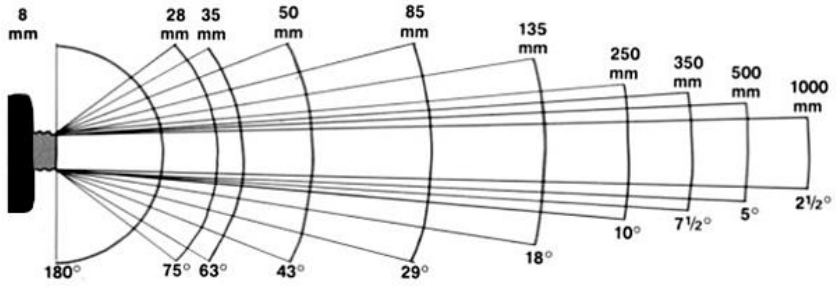
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rys. 2

Kąt fi to kąt o jaki chcemy wykonać obrót wokół danej osi, kąt może być również ujemny w przypadku obrotu w drugą stronę.

Zoom – to po prostu zwiększenie ogniskowej kamery, kiedy zmniejszymy ogniskową nasz obiektyw stanie się bardziej szerokokątny i będzie „widział” w szerszym zakresie, natomiast po zwiększeniu ogniskowej otrzymamy coś w rodzaju teleobiektywu, który bezstratnie przybliża obraz. Różnica między zoomem, a translacją wzdłuż osi Z jest oczywista, zoom przybliża obiekt, ale kamera nigdy nie zbliży się do niego, natomiast wykonując translacje powodujemy zmianę odległości kamery od obiektu oraz możemy nawet „wlecieć” w obiekt, powodując ukrycie niewidocznych krawędzi.



Rys. 3 Wizualizacja pojęcia ogniskowej

Rzutowanie – to przekształcenie obrazu (zbioru punktów) z przestrzeni 3d na przestrzeń 2d. Wykonuje się je poprzez przemnożenie wektora współrzędnych punktu przez macierz rzutowania lub poprzez użycie wzoru:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Rys. 4

Gdzie x1, x2 oraz x3 oznaczają współrzędne punktu w przestrzeni 3d, a y1 i y2 współrzędne na płaszczyźnie rzutowania. Aby uzyskać pożądany efekt kamery należy przesunąć układ współrzędnych na środek płaszczyzny, czyli ekranu na którym wyświetlamy obraz.

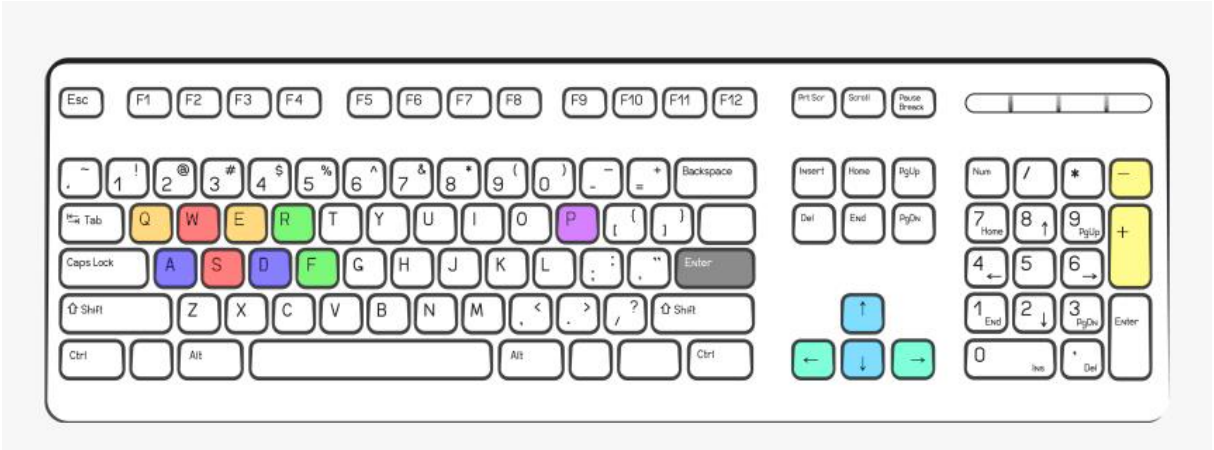
Środowisko i implementacja

Projekt został wykonany w języku Python z wykorzystaniem bibliotek:

- Pyglet – do rysowania kształtów i ich wyświetlania
- Numpy – do mnożenia macierzy
- Math – do operacji matematycznych

Do przechowywania danych zaimplementowane zostały 4 struktury danych, Point, Point2d, Line, Line2d. Służą do organizacji punktów w przestrzeniach 2d i 3d. Utworzone zostają punkty które odpowiadają wierzchołkom sześcianu, następnie poprzez analogie utworzone zostają pozostałe punkty, pozostałych sześcianów. Na podstawie tych punktów utworzone zostają linie. Każda operacja jest wyrażona odpowiednią funkcją. Macierze obrotów można obliczyć wcześniej ponieważ znamy z góry założone kąty pojedynczego obrotu. Podczas rzutowania nie uwzględniamy linii niewidocznych, czyli tych z tyłu kamery. Po zrzutowaniu linii 3d na 2d, są one rysowane na rzutnie czyli na okno programu. Obraz odświeżany jest 60 razy na sekundę aby zapewnić pełną płynność. Sterowanie odbywa się za pomocą klawiszy na klawiaturze.

Sterowanie



Translacje wzdłuż osi:

X: A/D Y: R/F Z: W/S

Obroty wokół osi:

X: UP/DOWN Y: LEFT/RIGHT Z: Q/E

Zoom:

NUM_PLUS/NUM_MINUS

Robienie zdjęcia:

P

Wnioski i spostrzeżenia

Planując oraz implementując zagadnienie wirtualnej kamery, popełniłem kilka błędów z których można wyciągnąć ciekawe wnioski. Pierwszym z nich było korzystanie tylko z punktów jako podstawa do dalszego rysowania modelu. Co prawda w przypadku braku funkcjonalności usuwania niewidocznych krawędzi (poza piramidą widzenia), przy użyciu skomplikowanej pętli, dało się narysować pożądany kształt, jednak jeśli chcemy taką funkcjonalność zaimplementować, potrzebujemy linie jako główny sposób przechowywania danych obiektu, ponieważ przy usunięciu kilku z nich, nadal bezproblemowo możemy je narysować nie parząc na kolejność punktów w liście.

Kolejnym błędem było przemnażanie punktów startowych przez zmienioną macierz obrotu, powodowało to obrót wokół osi świata, a nie osi kamery, wnioskiem jest to, że należy przemnożyć już wcześniej zmienione punkty, przez tą samą macierz obrotu (gdyż kąt pojedynczego obrotu mamy ustalony z góry). Początkową pomyłką było również nie uwzględnienie środka ekranu (rzutni), jako środka układu współrzędnych kamery, przez co translacje wzdłuż osi X oraz Y nie wyglądały dobrze, a obraz nie był w centrum ekranu, gdyż punkt rzutowania był na początku układu współrzędnych rzutni.

Ciekawą obserwacją jest przecinanie się linii które zależne jest od tego który sześcian został narysowany pierwszy. Jeżeli pierwszy został narysowany obiekt bliżej kamery wg. Osi Z, to obiekt który jest dalej i został później narysowany, będzie przecinał obiekt pierwszy od przodu, w miejscach przecięć będzie widoczna jego linia, niestety nie znam sposoby eliminacji tego problemu, sortowanie wg. Odległości Z działa tylko jednostronnie, być może należało by uwzględnić dotychczasowy obrót i odpowiednio posortować kolejność rysowania linii.