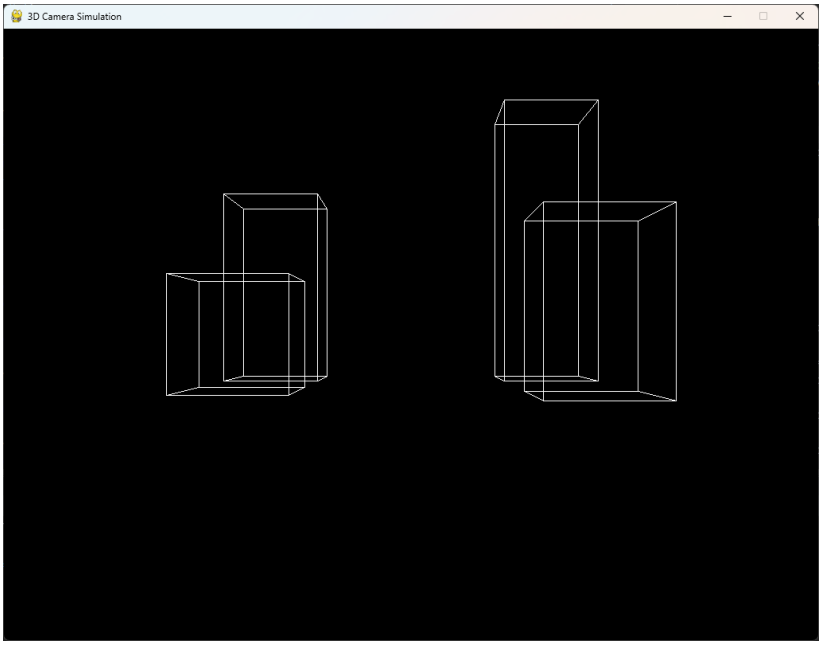
Sprawozdanie – ćw 1

Tomasz Tkaczyk, 319119



1. Technologia:

   Ćwiczenie zostało zrealizowane w języku *python* oraz przy użyciu bibliotek *pygame*, do rysowania kształtów 2d oraz *numpy* do obliczeń na macierzach.

2. Struktury danych:
   a. Obiekty przechowywane są jako lista punktów w pliku

```
"points_1": [
    [100, 0, 0],
    [200, 0, 0],
    [200, 100, 0],
    [100, 100, 0],
    [100, 0, 100],
    [200, 0, 100],
    [200, 100, 100],
    [100, 100, 100]
],
"points_2": [
    [100, 0, -100],
    [200, 0, -100],
    [200, 200, -100],
    [100, 200, -100],
    [100, 0, -200],
    [200, 0, -200],
    [200, 200, -200],
    [100, 200, -200]
],
```

   b. Krawędzie zdefiniowane jako połączenia między punktami

3. Pozycja oraz rotacja kamery zdefiniowane jako wektory 3 współrzędnych:

```
self.camera_position = np.array([0.0, 50, 750])
self.camera_rotation = np.array([0.0, 0.0, 0.0])
```

4. Ruch kamery realizowany jest poprzez wektory ruchu, które dodawane są do wektora pozycji:

```
self.move_vectors = {
    "forward": np.array([0.0, 0.0, -0.5]),
    "backward": np.array([0.0, 0.0, 0.5]),
    "left": np.array([0.5, 0.0, 0.0]),
    "right": np.array([-0.5, 0.0, 0.0]),
    "up": np.array([0.0, -0.5, 0.0]),
    "down": np.array([0.0, 0.5, 0.0]),
}
```

W celu „przywiązania” sterowania do kamery, wektory są obracane względem odpowiednich osi, zanim zostaną dodane do współrzędnych kamery:

```
def rotate_vector(vector, angles):
    x_angle, y_angle, z_angle = angles

    rotated_vector = vector
    rotated_vector = rotate_y_scene(rotated_vector, x_angle)
    rotated_vector = rotate_x_scene(rotated_vector, y_angle)
    rotated_vector = rotate_z_scene(rotated_vector, z_angle)
    return rotated_vector
```

5. Obroty kamery:
Obroty kamery realizowane są poprzez zwiększanie oraz zmniejszanie wartości kątów przechowywanych w macierzy rotacji kamery. Następnie kąty używane są podczas transformacji punktów w odpowiednich macierzach:

```
2 usages  Tomasz Tkaczyk
def rotate_x_scene(point, k):
    """Rotation matrix around the X-axis."""
    cos_k = np.cos(k)
    sin_k = np.sin(k)
    rotation_matrix = np.array([[1, 0, 0], [0, cos_k, -sin_k], [0, sin_k, cos_k]])
    return np.dot(rotation_matrix, point)


2 usages  Tomasz Tkaczyk
def rotate_y_scene(point, k):
    """Rotation matrix around the Y-axis."""
    cos_k = np.cos(k)
    sin_k = np.sin(k)
    rotation_matrix = np.array([[cos_k, 0, sin_k], [0, 1, 0], [-sin_k, 0, cos_k]])
    return np.dot(rotation_matrix, point)


2 usages  Tomasz Tkaczyk
def rotate_z_scene(point, k):
    """Rotation matrix around the Z-axis."""
    cos_k = np.cos(k)
    sin_k = np.sin(k)
    rotation_matrix = np.array([[cos_k, -sin_k, 0], [sin_k, cos_k, 0], [0, 0, 1]])
    return np.dot(rotation_matrix, point)
```

6. Transformacje punktów:
   Każdy punkt w celu narysowania na ekranie przechodzi następującą transformację:
   a. Odjęcie współrzędnych kamery od punkt
   b. Obrót punktu względem rotacji kamery
   c. Projekcja punktu przy użyciu macierzy projekcji oraz zastosowanie przybliżenia
   d. Centrowanie 2-wymiarowego punktu w celu rysowania obrazu na środku ekranu

```python
def transform_point(point, camera_position, camera_rotation, f, zoom=1):
    """Transform a 3D point to a 2D point."""
    translated_point = point - camera_position
    rotated_point = rotate_point(translated_point, camera_rotation)
    projected_point = project_point(rotated_point, f, zoom)
    centered_point = center_point(projected_point)

    return centered_point
```

7. Po transformacji punktów, rysowane są odpowiednie krawędzie:

```python
def draw_figure(self, figure_points):
    transformed_points = []
    for point in figure_points:
        transformed_point = transform_point(point, self.camera_position, self.camera_rotation, self.f, self.zoom)
        transformed_points.append(transformed_point)
    for i, edge in enumerate(edges):
        self.draw_edge(edge, transformed_points, Colors.WHITE)
```