Szymon Dyszewski

Numer albumu: 310625

Sprawozdanie z laboratorium nr 8

z przedmiotu WMM

Składanie transformacji

Początkowo wczytujemy dostarczone obiekty tj. Cube i Sphere.

```
def model_load(self):
    self.box = self.load_scene('cube.obj')
    self.box = self.box.root_nodes[0].mesh.vao.instance(self.program)

    self.sphere = self.load_scene('sphere.obj')
    self.sphere = self.sphere.root_nodes[0].mesh.vao.instance(self.program)
```

Tworzymy obiekt Sphere, który posłuży nam za głowę robota, ustawiamy jej kolor i przekształcenie o wektor (0, 0, 5) - zgodnie z poleceniem. Następnie renderujemy obiekt do wyświetlenia.

```
##############################################################################

        # Head

        # Color
        self.color.value = (0.0, 1.0, 0.0)

        # Translation by vector(0, 0, 5)
        move = Matrix44.from_translation((0.0, 0.0, 5.0))

        self.pvm_matrix.write((projection * lookat * move).astype('f4'))
        self.sphere.render()

##############################################################################
```

Analogicznie tworzone jest ciało robota (obiekt Cube). Poza ustawieniem koloru i translacji, obiekt musi zostać przeskalowany (wydłużony dwukrotnie w osi Z), by posłużył nam za tułów.

```
#############################################################################

        # Body

        # Color
        self.color.value = (0.0, 1.0, 1.0)

        # Axis scale by (1, 1, 2)
        scale = Matrix44.from_scale((1.0, 1.0, 2.0))

        # Translation by vector(0, 0, 2)

        move = Matrix44.from_translation((0.0, 0.0, 2.0))

        self.pvm_matrix.write((projection * lookat * move * scale).astype('f4'))
        self.box.render()

#############################################################################
```

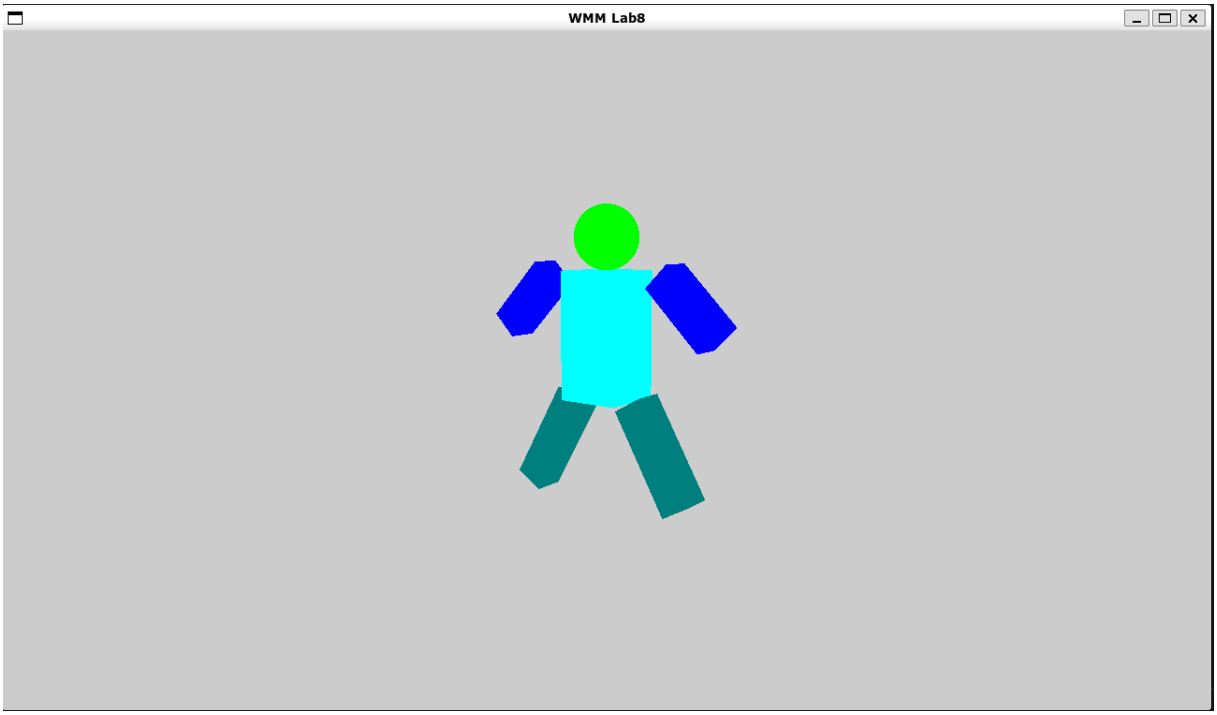
Ręce wymagają utworzenia obu obiektów Cube i odpowiedniego przeskalowania ich. Ponadto ręce muszą zostać obrócone o +/- pi/4 względem osi X, by odpowiadać zadanemu projektowi.

```
#############################################################################

        # Arms

        # Color
        self.color.value = (0.0, 0.0, 1.0)

        # Axis scale by (0.5, 0.5, 1.25)
        scale = Matrix44.from_scale(Vector3([0.5, 0.5, 1.25]))

        # 45 degrees rotation on the x axis
        rotate = Matrix44.from_x_rotation(-0.25 * np.pi)

        # Translation by vector(0, 3, 3)
        move = Matrix44.from_translation(Vector3([0.0, 3.0, 3.0]))

        self.pvm_matrix.write((projection * lookat * move * rotate * scale).astype('f4'))
        self.box.render()

        # 45 degrees rotation on the x axis
        rotate = Matrix44.from_x_rotation(0.25 * np.pi)

        # Translation by vector(0, -3, 3)
        move = Matrix44.from_translation(Vector3([0.0, -3.0, 3.0]))

        self.pvm_matrix.write((projection * lookat * move * rotate * scale).astype('f4'))
        self.box.render()

#############################################################################
```

Ostatnim etapem jest stworzenie nóg przy pomocy dwóch obiektów Cube. Wszystkie kroki wykonano analogicznie do kroków z rąk.

```
########################################################################

        # Legs

        # Color
        self.color.value = (0.0, 0.5, 0.5)

        # Axis scale by (0.5, 0.5, 1.75)
        scale = Matrix44.from_scale(Vector3([0.5, 0.5, 1.75]))

        # 30 degrees rotation on the x axis
        rotate = Matrix44.from_x_rotation(-np.pi/6)

        # Translation by vector(0, 2, -1.5)
        move = Matrix44.from_translation(Vector3([0.0, 2.0, -1.5]))

        self.pvm_matrix.write((projection * lookat * move * rotate * scale).astype('f4'))
        self.box.render()

        # 30 degrees rotation on the x axis
        rotate = Matrix44.from_x_rotation(np.pi/6)

        # Translation by vector(0, -2, -1.5)
        move = Matrix44.from_translation(Vector3([0.0, -2.0, -1.5]))

        self.pvm_matrix.write((projection * lookat * move * rotate * scale).astype('f4'))
        self.box.render()
```

Efekt końcowy po wywołaniu wszystkich poleceń to:



Dodanie wektora kolorów dla wszystkich fragmentów:

resources > shaders > passthrough > robot > robot.frag

```
1   #version 330
2
3   uniform vec3 color;
4
5   out vec4 f_color;
6
7   void main()
8   {
9       f_color = vec4(color, 1.0);
10  }
11
```

Dodanie macierzy transformacji dla wszystkich wierzchołków:

resources > shaders > passthrough > robot > robot.vert

```
1   #version 330
2
3   uniform mat4 pvm_matrix;
4
5   in vec3 in_position;
6
7   void main()
8   {
9       gl_Position = pvm_matrix * vec4(in_position, 1.0);
10  }
11
```