4D-GML

Marek Krótkiewicz, Marcin Jodłowiec

10 czerwca 2020

1 Cel

Opracowanie składni abstrakcyjnej i semantyki dla języka operowania w przestrzeni trójwymiarowej z uwzględnieniem osi czasu na obiektach reprezentowanych przez grafy.

2 Przestrzeń

Przestrzeń jest trójwymiarowa (X, Y, Z), w której określono kartezjański układ współrzędnych. Dodatkowo zakłada się dyskretną oś czasu.

3 Obiekt

$$G = (N, E) \tag{1}$$

gdzie:

N – zbiór węzłów,

E – zbiór krawędzi.

Wierzchołek:

$$n \in N \tag{2}$$

$$n = (point, vpoint, apoint) \tag{3}$$

$$point = (t, x, y, z, \theta, \phi, \psi) \tag{4}$$

$$vpoint = (v_x, v_y, v_z, \omega_\theta, \omega_\phi, \omega_\psi) \tag{5}$$

$$apoint = (a_x, a_y, a_z, \alpha_\theta, \alpha_\phi, \alpha_\psi) \tag{6}$$

$$v_x = \frac{dx}{dt}; v_y = \frac{dy}{dt}; v_z = \frac{dz}{dt} \tag{7}$$

$$\omega_\theta = \frac{d\theta}{dt}; \omega_\phi = \frac{d\phi}{dt}; \omega_\psi = \frac{d\psi}{dt} \tag{8}$$

$$a_x = \frac{dv_x}{dt} = \frac{d^2x}{dt^2}; a_y = \frac{dv_y}{dt} = \frac{d^2y}{dt^2}; a_z = \frac{dv_z}{dt} = \frac{d^2z}{dt^2} \tag{9}$$

$$\alpha_\theta = \frac{d\omega_\theta}{dt} = \frac{d^2\alpha_\theta}{dt^2}; \alpha_\phi = \frac{d\omega_\phi}{dt} = \frac{d^2\alpha_\phi}{dt^2}; \alpha_\psi = \frac{d\omega_\psi}{dt} = \frac{d^2\alpha_\psi}{dt^2} \tag{10}$$

Krawędź:

$$e \in E \tag{11}$$

$$e = (startnode, endnode) \tag{12}$$

$$startnode, endnode \in N \tag{13}$$

gdzie:

t – chwila dla której określono x, y, z oraz θ, ϕ, ψ;

x, y, z – współrzędne aktualnego położenia węzła w chwili t (w kartezjańskim trójwymiarowym układzie współrzędnych), tj. $x(t), y(t), z(t)$;

θ, ϕ, ψ – orientacja w przestrzeni trójwymiarowej danego węzła, tj. $\theta(t), \phi(t), \psi(t)$;

v_x, v_y, v_z – prędkości liniowe w określonej osi: x, y lub z danego węzła w chwili t, tj. $v_x(t), v_y(t), v_z(t)$;

$\omega_\theta, \omega_\phi, \omega_\psi$ – prędkości kątowe danego węzła, tj. $\omega_\theta(t), \omega_\phi(t), \omega_\psi(t)$;

a_x, a_y, a_z – przyśpieszenia liniowe w określonej osi: x, y lub z danego węzła w chwili t, tj. $a_x(t), a_y(t), a_z(t)$;

$\alpha_\theta, \alpha_\phi, \alpha_\psi$ – prędkości kątowe danego węzła, tj. $\alpha_\theta(t), \alpha_\phi(t), \alpha_\psi(t)$.

Przykład:

W określonym momencie t, dla każdego węzła można ustalić położenie początkowe wprowadzając np.:

$$\begin{aligned} n_1.point.x &= 14.53, \\ n_1.point.y &= 4.56, \\ n_1.point.z &= 828. \end{aligned}$$

Jeżeli wprowadzono jednocześnie:

$$n_1.point.t = 0,$$

to oznacza, że współrzędne obowiązują od momentu wprowadzenia. Dla innej wartości np. $n_1.point.t = 3.4$, oznaczałoby to, że wartości te obowiązują (należy zrealizować) za 3.4 sekundy. Analogicznie dla prędkości i przyśpieszeń liniowych i kątowych:

$$\begin{aligned} n_1.point.t &= 5.73, \\ n_1.point.v_x &= 21.5, \\ n_1.point.\alpha_\phi &= 2.94, \end{aligned}$$

oznacza, że za 5.73 sekundy należy zmienić prędkość liniową w osi x węzła n_1 na wartość $21.5\ {}^{m}/_{s}$ oraz przyśpieszenie kątowe w płaszczyźnie ϕ na wartość $2.94\ {}^{rad}/_{s^2}$.

Każde wykonanie polecenia tj. `EXECUTEMOVEMENT()` powoduje iż wątek obliczający wartości położenia dla poszczególnych węzłów przelicza ich aktualne położenie oraz orientację w przestrzeni zgodnie z zadanymi aktualnie wartościami.

4 Przykładowe wyrażenia

4.1 CREATE

```
graph = GRAPH()
node = NODE(x, y, z, theta, phi, psi)
edge = EDGE(node1, node2)
```

4.2 READ

```
valx = graph.nodes["n1"].point.x
valatheta = graph.nodes["n1"].apoint.alpha_theta
```

4.3 UPDATE

```
graph.ADDNODE(node)
graph.ADDEDGE(edge)
graph.UPDATENODE(node1, node2)
graph.UPDATEEDGE(edge1, edge2)
graph.MERGEGRAPH(graph1)

//------------------------//
graph.nodes["n1"].point.x = valx
graph.nodes["n1"].vpoint.vy = valy
```

4.4 DELETE

```
graph.DELETE()
graph.DELETENODE(node)
graph.DELETEEDGE(edge)
```

4.5 EXECUTEMOVEMENT

```
graph.EXECUTEMOVEMENT()
```

5 Struktury danych

(pseudo-python)

5.1 Graf

```
class GRAPH:

    def __init__(self):
        self.nodes = {}
        self.edges = {}
```

5.2 Węzeł

```
class NODE:
    class POINT:
        def __init__(self, x, y, z, alpha, beta, gamma):
            self.x = x
            self.y = y
            self.z = z
            self.alpha = alpha
            self.beta = beta
            self.gamma = gamma

    class VPOINT:
        def __init__(self, vx,vy,vz,om_theta,om_phi,om_psi):
            self.vx = vx
            self.vy = vy
            self.vz = vz
            self.om_theta = om_theta
            self.om_phi = om_phi
            self.om_psi = om_psi

    class APOINT:
        def __init__(self, ax,ay,az,alpha_theta,alpha_phi,alpha_psi):
            self.ax = ax
            self.ay = ay
            self.az = az
            self.alpha_theta = alpha_theta
            self.alpha_phi = alpha_phi
            self.alpha_psi = alpha_psi


     def __init__(self, x, y, z, alpha, beta, gamma):
        self.point = POINT(x, y, z, alpha, beta, gamma)
        self.vpoint = VPOINT(0,0,0,0,0,0)
        self.apoint = APOINT(0,0,0,0,0,0)
```

5.3 Krawędź

```
class EDGE:

    def __init__(self, node1, node2):
        self.node1 = node1
        self.node2 = node2
```