Katarzyna Łuszczewska
Sebastian Pilch

Animacja w THREE.js

Konspekt laboratorium:

1. Przedstawienie możliwości tworzenia animacji wraz z przykładami (całość będzie znajdować się w prezentacji)
2. Przekazanie pozostałym studentom prostych zadań do zrealizowania w oparciu w wiedzę teoretyczną z wykładu.
   Zadanie będzie zawierać scenę wraz z kilkoma bryłami, stworzą w THREE.js, którą będzie można uruchomić w przeglądarce. Każda z przedstawionych brył będzie wymagała zaimplementowania jednej z omawianych animacji:
   a) Zmiana koloru bryły poprzez kliknięcie
   b) Poruszanie się bryły po planszy
   c) Odbijanie się bryły
3. Zaprezentowanie rozwiązań

Do realizacji zadań będzie wykorzystane:

- Java Script wraz z biblioteką THREE.js

- HTML, do uruchomienia sceny w przeglądarce

Draft prezentacji:

Wstęp

Animacja w THREE.js polega na wykorzystywaniu gotowych metod z języka Java Script i zastosowania ich w obiektach 3D, co związane jest głównie z wykorzystywaniem możliwości obiektów i otoczenia na trzech płaszczyznach (x, y, z).

Do stworzenia każdej animacji konieczne jest utworzenie obiektu 3D wraz z materiałem, wyznaczanie pozycji kamery, ustawienie światła oraz rendera (mechanizmu renderującego) . Taka baza umożliwia stworzenie animacji.

Sposoby wywoływania animacji

1. RequestAnimationFrame
   Jest to często stosowana metoda, która pozwala na wykonanie animacji. Metoda ta wymaga od przeglądarki, aby wywoływała określoną funkcję przed kolejną zmianą położenia obiektu

poprzez animację. Metodę można wywoływać korzystając z rekursji, do momentu spełnienia określonych warunków, lub w nieskończoność.

```
requestAnimationFrame(callback)
```

callback - funkcja, którą należy wywołać, gdy nadejdzie czas na aktualizację animacji przed kolejnym odświeżeniem. Posiada jeden argument timestamp, który wskazuje na czas zakończenia renderowania poprzedniej klatki. Oparty jest na liczbie milisekund od początku działania)

2. AddEventListener
   Jest to funkcja, która wykorzystuje uchwyt (handler) i w zależności do niego wykonuje kod w momencie wystąpienia wydarzenia.

```
addEventListener(type, listener)
addEventListener(type, listener, options)
```

type – reprezentuje ciąg znaków odpowiadający typu eventu, którego zasłuchuje
listener – otrzymuje powiadomienie, które mówi o typie wydarzenia, która ma się zadziać
options – są to opcje true/false, które określają:
a) czy zdarzenie będzie obsługiwane podczas fazy przechwytywania (capture),
b) czy listener będzie usuwany po pierwszym nasłuchaniu (once)

Istnieje wiele eventów, które stosuje się dla tego uchwytu jak np.:
- „click” – po kliknięciu przyciskiem myszy

```
window.addEventListener('click', event =>
this.handle_mouse_click(event));
```

- „mousemove” – po najechaniu na element

```
window.addEventListener('mousemove', event =>
this.handle_mouse_move(event));
```

- „mouseup” – przeciągając jakiś element

```
addEventListener("mouseup", (event) => {});
```

- „keydown” – kliknięcie w wybrany klawisz

```
addEventListener("keydown", (event) => {});
```

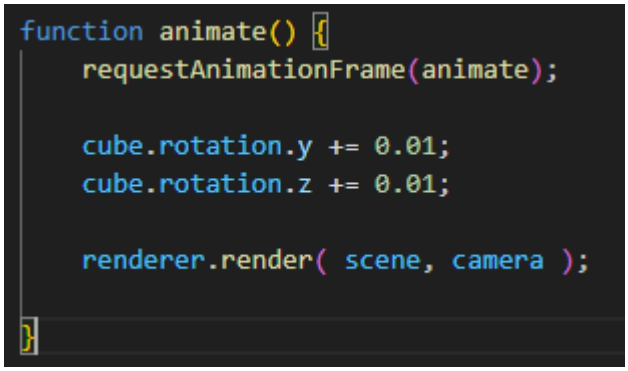
3. SetAnimationLoop

Jest to inna funkcja, która może być używana zamiast RequestAnimationFrame i wymaga również jedynie funkcji zwrotnej. Wykorzystuje się są zarówno do projektów VR i nie VR dla WebXR. Używana jest w przypadku, gdy render ma być płynny, czy ma trwać w nieskończonej pętli.

```
import * as THREE from 'three';

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setAnimationLoop( animate );
document.body.appendChild( renderer.domElement );

const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;

function animate() {

    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;

    renderer.render( scene, camera );

}
```

Przykłady animacji

1. Rotacja
   Jest to najprostsza możliwa animacja do wykonania w JavaScripcie, ponieważ wymaga wykorzystanie wbudowanej metody rotate



   Po zdefiniowaniu obiektu, który ma być obracany, wystarczy do jego odpowiednich osi przyporządkować wartości, co ile wartości mają się zmieniać, co umożliwia ciągłą rotację w wyżej pokazanym przypadku.

2. Odbijanie się obiektu

Wykonanie tej animacji wymaga wykorzystania requestAnimationFrame oraz znajomości matematyki. Funkcja umożliwia aktualizowanie położenia obiektu za każdym razem w przypadku zmiany pozycji, z kolei matematyczne wzory związane z zmianą amplitudy i oscylacją umożliwiają poruszanie się obiektu na osi Y.

```
static bounce(object, amplitude, frequency, damping, starting_position) {
    const animate = (time) => {
        requestAnimationFrame(animate);
        if (!this.startTime) {
            this.startTime = time;
        }

        const elapsed = (time - this.startTime) / 1000;
        const current_amplitude = amplitude * Math.exp(-damping * elapsed);
        if (current_amplitude < 0.05) {
            object.position.y = starting_position;
            return;
        }
        object.position.y = starting_position + current_amplitude * Math.sin(2 * Math.PI * frequency * elapsed);
    };
    requestAnimationFrame(animate);
}
}
```

3. Zmiana obiektu

   Poniższe zmiany obiektu są możliwe dzięki wykorzystaniu addEventListener. Można takiej zmiany dokonać w sposób, który jest właściwie jedynie ograniczony przez uchwyty.

   a) koloru

      Obiekt może zmienić kolor po najechaniu na niego lub po kliknięciu na niego. Daje to możliwość interaktywności użytkownika z obiektem oraz ułatwia nawigację.

```
window.addEventListener('click', event => this.handle_mouse_click(event));
```

```
handle_mouse_click(event) {
    if (this.draggable) {

        console.log(`Drop draggable: ${this.draggable.userData.name}`);

        const target_pos_x = Math.floor(this.draggable.position.x) + 0.5;
        const target_pos_z = Math.floor(this.draggable.position.z) + 0.5;

        this.draggable.position.set(target_pos_x, 0.5, target_pos_z);
        this.draggable.material.emissive.set(0x000000);
        this.draggable = null;
        this.is_draggable = false;
        console.log(`Dropped at: ${target_pos_x}, ${target_pos_z}`);
        return;
    }

    this.click_mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
    this.click_mouse.y = - (event.clientY / window.innerHeight) * 2 + 1;

    this.raycaster.setFromCamera(this.click_mouse, this.camera);
    const intersects = this.raycaster.intersectObjects(this.scene.children);

    if (intersects.length > 0) {
        const intersectedObject = intersects[0].object;
        if (intersectedObject.userData && intersectedObject.userData.draggable) {

            this.draggable = intersectedObject;
            console.log(`Found draggable: ${this.draggable.userData.name}`);
            this.is_draggable = true;
            this.draggable.material.emissive.set(0xff0000);
        }
    } else {
        console.log('Nothing found');
    }
}
```

b) Zmiana pozycji przez kopiowanie pozycji kursora

Taka opcja jest szczególnie przydatna w momencie przesunięcia czegoś. Pozwala w grach ustawiać pionki czy tworzyć sceny.

```
window.addEventListener('click', event => this.handle_mouse_click(event));
```

```
drag_object() {
    if (this.is_draggable && this.draggable) {
        this.raycaster.setFromCamera(this.move_mouse, this.camera);
        const intersects = this.raycaster.intersectObjects(this.board.children);

        if (intersects.length > 0) {
            for (let obj of intersects) {
                if (!obj.object.userData.ground) continue;

                this.draggable.position.x = obj.point.x
                this.draggable.position.z = obj.point.z

            }
        }
    }
}
```

c) Zmiana pozycji model matematyczny

Opcja dostosowująca dynamikę zmian pozycji na podstawie równania różniczkowego II rzędu. Szczególnie przydatne w przypadku ruchów z opóźnieniem / gasnącymi oscylacjami itp.

```
drag_object() {
    if (this.is_draggable && this.draggable_obj) {
        this.raycaster.setFromCamera(this.move_mouse, this.camera);
        const intersects = this.raycaster.intersectObjects(this.board.children);

        if (intersects.length > 0) {
            for (let obj of intersects) {

                if (obj.object.userData.type != 'ground') continue;


                if (this.draggable_obj.userData.active)
                {
                    this.draggable_obj.userData.setPositionPrime = obj.point.clone().sub( this.draggable_obj.userData.setPosition.clone());
                    this.draggable_obj.userData.setPosition = obj.point.clone()
                }
                // var setPointPrime =obj.point.clone().sub( this.draggable_obj.position.clone());

                Animation.second_order_model(this.draggable_obj , this.params,  deltaTime: 0.05);


                // this.draggable_obj.position.x = obj.point.x
                // this.draggable_obj.position.z = obj.point.z

            }
        }
    }
}
```

```
handle_mouse_click(event) {

    if (this.draggable_obj && this.is_draggable) {

        // console.log(`Drop draggable: ${this.draggable.userData.name}`);
        // console.log(this.board);

        const target_pos_x = Math.floor(this.draggable_obj.userData.setPosition.x) + 0.5;
        const target_pos_z = Math.floor(this.draggable_obj.userData.setPosition.z) + 0.5;

        let new_setPoint = new THREE.Vector3(target_pos_x,  y: 0.5, target_pos_z)

        this.draggable_obj.userData.active = false;
        this.draggable_obj.userData.setPositionPrime =  new_setPoint.clone().sub(this.draggable_obj.userData.setPosition.clone())
        this.draggable_obj.userData.setPosition = new_setPoint.clone()

        // Animation.second_order_model(this.draggable_obj , this.params, 0.05);


        this.draggable_obj.userData.row = this.position_to_row(target_pos_z)
        this.draggable_obj.userData.column = this.position_to_row(target_pos_x)
        this.is_draggable = false;
        console.log(`Dropped at: ${target_pos_x}, ${target_pos_z}`);
        console.log(this.draggable_obj.userData.active);
        this.clear_board();
        this.change_emission(this.draggable_obj);
        this.draggable_obj = null;
        return;
    }
```

```
static second_order_model(object, params, deltaTime) {
    const { damping, frequency, response_factor } = params;
    let setpoint = object.userData.setPosition.clone();
    let setpointPrime = object.userData.setPositionPrime.clone();
    const omega = 2 * Math.PI * frequency;
    const damping_term = damping / (Math.PI * frequency);
    const stiffness_term = 1 / (omega ** 2);

    let Pos = object.position.clone();
    let PosPrime = new THREE.Vector3( x: 0,  y: 0,  z: 0);
    let PosDoublePrime = new THREE.Vector3( x: 0,  y: 0,  z: 0);

    const simulate = (time) => {
        requestAnimationFrame(simulate);

        deltaTime = deltaTime || 0.016;
        setpoint = object.userData.setPosition.clone();
        setpointPrime = object.userData.setPositionPrime.clone();
        Pos = object.position.clone();

        const rhs = setpoint.clone().add(
            setpointPrime
                .clone()
                .multiplyScalar(response_factor * damping / (2 * Math.PI * frequency))
        );

        PosDoublePrime = rhs
            .clone()
            .sub(Pos)
            .sub(PosPrime.clone().multiplyScalar(damping_term))
            .multiplyScalar(1 / stiffness_term);

        // Aktualizacja predkości i pozycji
        PosPrime.add(PosDoublePrime.clone().multiplyScalar(deltaTime));
        Pos.add(PosPrime.clone().multiplyScalar(deltaTime));

        object.position.copy(Pos);

        // if(!object.userData.active && object.position.clone().sub(setpoint.clone()).length() < 0.01)
        // {
        //
        //     // const snappedX = Math.floor(object.position.x) + 0.5;
        //     // const snappedZ = Math.floor(object.position.z) + 0.5;
        //     // object.position.set(snappedX, object.position.y, snappedZ);
        // //     return;
        // }

    };
    if(object.userData.active || object.position.clone().sub(setpoint.clone()).length() >= 0.001)
    {
        requestAnimationFrame( callback: () => simulate());
    }
}
```