

Grafika komputerowa. Laboratorium 5.

Fizyka. Kolizje

Obliczanie kolizji z użyciem biblioteki **physijs**

W tym ćwiczeniu zajmujemy się wykorzystaniem biblioteki **Physijs**, jednej z popularniejszych bibliotek pozwalających na obliczenia/symulacje fizyczne. Alternatywne biblioteki do współpracy z **WebGL/Three.js** to między innymi: **cannon.js** i w pewnym sensie **babylon.js** (którego możliwości są znacznie większe).

Przyznaję, że **Physijs** i **cannon.js** nie są już obecnie (2022) rozwijane. Należałoby w ćwiczeniu zastosować raczej bibliotekę **cannon-es** (następcę **cannon.js**) i **enable3d**. Nie przygotowałem tego jeszcze, jednak zachęcam wszystkich do spróbowania użycia nowych bibliotek.

Nie jest to temat ściśle graficzny, ale ważny w animacji i grach komputerowych.

Tworzenie sceny **three.js** z wykorzystaniem **Physijs**

Tworzenie sceny z wykorzystaniem **Physijs** jest bardzo proste i składa się zazwyczaj z kilku tylko kroków.

Samą bibliotekę i towarzyszące jej pliki pomocnicze można pobrać z

<http://chandlerprall.github.io/Physijs/>.

Krótki opis biblioteki jest dostępny na stronie: <https://github.com/chandlerprall/Physijs/wiki>

Poszczególne etapy użycia **Physijs** mogą być następujące:

1. Importujemy bibliotekę z pomocą:

```
<script type="text/javascript" src="physijs.js"></script>
```

2. Uruchamiamy tryb *worker* dla symulacji.

Symulacje fizyczne obejmujące kolizje są dość wymagające obliczeniowo, a takie obciążenie procesora może niekorzystnie wpłynąć na szybkość renderowania. W **Physijs** obliczenia wykonują się w tle dzięki zastosowaniu mechanizmu **Web workers** dostępnego w większości przeglądarek.

Informację na temat **Web workers** można znaleźć w

<http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html>

W **Physijs** musimy wskazać co chcemy wykonywać w tle (czyli w trybie *worker*) i jakiego silnika fizyki używamy (w naszym przypadku jest to **ammo.js**. **Physijs** jest tylko wrapperem do współpracy z **Three.js**). Sam silnik **ammo.js** można znaleźć na <https://github.com/kripken/ammo.js/>

W kodzie wpisujemy:

```
Physijs.scripts.worker = 'physijs_worker.js';  
Physijs.scripts.ammo = 'ammo.js';
```

3. Zastępujemy `THREE.Scene` przez `Physijs.Scene`. Dzięki temu scena zostaje wyposażona w dodatkowe atrybuty reprezentujące fizykę.

Jako jeden z atrybutów tej sceny dokładamy grawitację:

```
var scene = new Physijs.Scene();
scene.setGravity(new THREE.Vector3(0, -10, 0));
```

Więcej atrybutów sceny można znaleźć na wiki <https://github.com/chandlerprall/Physijs/wiki>

4. Następnie dodajemy obiekty, które będą podlegać symulacji. Są to zwykłe obiekty Three.js 'owinięte' własnościami dostarczonymi przez Physijs.

```
var stoneGeom = new THREE.CubeGeometry(0.6, 6, 2);
var stone = new Physijs.BoxMesh(stoneGeom,
new THREE.MeshPhongMaterial({color: 0xff0000}));
scene.add(stone);
```

Mesh

W powyższym fragmencie zamiast obiektu typu `THREE.Mesh` tworzymy obiekt `Physijs.BoxMesh`, który zapewnia obsługę kolizji. Zauważamy, że w miejsce jednej uniwersalnej funkcji `Mesh`, `Physijs` dostarcza nam kilku funkcji dopasowanych do różnych kształtów. Dzieje się tak dlatego, że generowane są niejako dwa obiekty: jeden, który zawiera informacje o geometrii do wyświetlenia, i drugi, który zawiera informacje o geometrii do obliczeń kolizji. Przy tym oba te kształty nie muszą być dokładnie jednakowe. Funkcje zebrane są w poniższej tabelce.

Kształty obsługiwane przez Physijs

Atrybut	Opis
<code>Physijs.PlaneMesh</code>	Wygeneruje obiekt o zerowej grubości. Można również użyć <code>BoxMesh</code> o niewielkiej wysokości. Jednak <code>PlaneMesh</code> standardowo nie podlega grawitacji i nie porusza się pod wpływem zderzeń
<code>Physijs.BoxMesh</code>	Prostopadłościan. Można łączyć z geometrią <code>THREE.CubeGeometry</code>
<code>Physijs.SphereMesh</code>	Sfera. Łączy się z <code>THREE.SphereGeometry</code>
<code>Physijs.CylinderMesh</code>	Cylinder. Łączy się z <code>THREE.Cylinder</code> .
<code>Physijs.ConeMesh</code>	Stożek. Łączy się z geometrią <code>THREE.Cylinder</code> z górnym promieniem równym 0.
<code>Physijs.CapsuleMesh</code>	Cylinder z zaokrąglonymi końcami.
<code>Physijs.ConvexMesh</code>	Złożony kształt wypukły. Może się łączyć z geometrią <code>THREE.ConvexGeometry</code> , którą aproksymuje. <code>ConvexGeometry</code> nie jest opisana w <code>threejs.org</code>
<code>Physijs.ConcaveMesh</code>	
<code>Physijs.HeightfieldMesh</code>	Z pomocą tej siatki można wygenerować pole wysokości na podstawie <code>THREE.PlaneGeometry</code>

Obiekt zdefiniowany przez jedną z powyższych siatek może być uzupełniony o parametr 0 lub 1 wskazujący na obecność grawitacji (przykład poniżej).

Materiał zdefiniowany z użyciem Three.js może być 'owinięty' przez `Physijs.createMaterial`, który uzupełnia materiał o własności mechaniczne powierzchni: tarcie (friction) i sprężystość (restitution).

5. Symulację wprowadzonych obiektów możemy zapoczątkować przez:

```
render = function() {
  requestAnimationFrame(render);
  renderer.render(scene, camera);
  scene.simulate();
}
```

Przykłady

Do eksperymentów z kolizjami dołączony jest zestaw z trzema przykładami. Zajmiemy się modyfikacją tylko jednego z nich (balls.html).

Dwa pozostałe, które pochodzą ze strony `physijs` (`collisions.html` i `shapes.html`) tylko oglądamy.

Proszę zwrócić uwagę na parametry, które pojawiają się w definicji obiektów za pomocą metod: `Physijs.BoxMesh` i `Physijs.createMaterial`.

Pojawiają się tam parametry `.4`, `.99` i `0` związane ze sprężystością odbicia, tarciem i siłą ciężkości. Proszę poeksperymentować ze zmianą tych parametrów.

```
plane = new Physijs.BoxMesh(
  new THREE.CubeGeometry(100, 100, 2, 10, 10),
  Physijs.createMaterial(
    new THREE.MeshLambertMaterial({
      color: 0xeeeeee
    }),
    .4,           // friction
    .99          // restitution
  ),
  1              // gravity
);
```

Uzupełnienie 7.12.2021

Uruchomienie symulacji obiektów `scene.simulate()`; obejmuje tylko te obiekty, które poruszają się pod wpływem grawitacji lub na skutek kolizji. `Physijs` automatycznie rejestruje zmiany ich położenia i orientacji. Natomiast te obiekty, które są poruszane lub obracane niezależnie, np. za pomocą atrybutów `.position` lub `.rotation` – nie informują o tym `Physijs`, który uznaje, że nic się w otoczeniu nie zmieniło. Można tę informację przekazać ręcznie poprzez dodanie do przemieszczanego/obracanego obiektu atrybutu odpowiednio `__dirtyPosition` i `__dirtyRotation` w formie:

```
object.__dirtyPosition = true;  
object.__dirtyRotation = true;
```

Zostało to pokazane w przykładzie `ball_single`.

Do zrobienia:

Proszę zmodyfikować układ z kulkami np. poprzez: dołożenie przeszkód na podłożu, od którego odbijają się kulki, huśtanie się podłoża, zmianę częstotliwości generowania kulek, lub jeszcze coś innego. Można wygenerować pudło, które wypełnia się kulkami, aż się z niego wysypują. Kulki mogą się przesypywać z jednego pudła do drugiego, ... i co jeszcze przyjdzie Państwu do głowy.