Three.js

Nowy wymiar 3D w przeglądarce

Michał Rygielski

2016-04-21

Czym jest Three.js?

Jest to **zaawansowana biblioteka stworzona w JavaScript**. Jej twórą jest Ricardo Cabello, który w 2010 upublicznił pierwszą wersję Three.js w Internecie.

Pomysł na bibliotekę wziął się z bardzo dużego zaangażowania Ricardo Cabello w demoscenę. **Pierwsza wersja biblioteki została napisana w ActionScript w 2009** roku a dopiero rok później przeniesiona do JavaScript. Pierwsza biblioteki opierała się o rendering CanvasRenderer oraz SVGRenderer.

Wraz z rozwojem WebGL w przeglądarkach internetowych biblioteka została zaktualizowana właśnie o tą technologię. Rozszerzenia dokonał Paul Brunt.

Three.js **dostępna jest za darmo** i jest w tej chwili rozwijana przez społeczność.

Na co tak na prawdę pozwala Three.js?

Biblioteka Three.js jest dedykowana środowisku 3D i pozwala w prosty i szybki sposób stworzyć wirtualny świat w okienku przeglądarki internetowej.

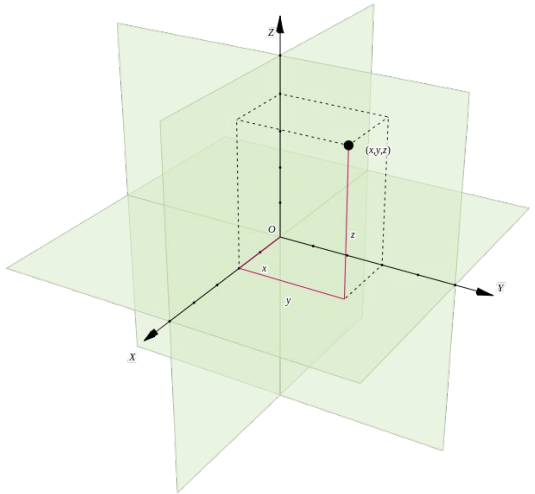
Dzięki Three.js możemy m.in.:

- Budować bryły
- Wczytywać modele 3D
- Odtwarzać animacje szkieletowe
- Zarządzać oświetleniem
- Ustawiać kamerę
- Konfigurować cieniowanie obiektów

Tylko wyobraźnia ogranicza programistę

Czym w ogóle jest 3D?

Jest to nic innego jak przestrzeń trójwymiarowa (euklidesowa). Znaczy to tyle, że każdemu punktowi w tej przestrzeni odpowiada trójka liczb rzeczywistych określanych jako X, Y i Z.



fot. Jorge Stolfii

3D jest niczym kosmos!

No prawie, lecz podobieństwo jest bardzo duże, bynajmniej jeśli chodzi o zilustrowanie jak to wszystko działa.

Przestrzeń kosmiczna jest niczym matrix, w której gwiazdy (punkty) posiadają swoje współrzędne określane jako X, Y, Z.

Mając statek kosmiczny, który ma się poruszać w tej przestrzeni wystarczy, że będziemy operować na wektorach konkretnych osi w układzie współrzędnych.

Przestrzeń 3D jest bardzo podobna do tej dwuwymiarowej, różni się tylko tym, że dochodzi nam oś Z, czyli ta odpowiadająca za określenie głębii.

Wyobraź sobie przestrzeń trójwymiarową jako kosmos

Robimy pierwszą aplikację w 3D - zmienne

Wszystkie aplikacje oparte o Three.js nie potrzebują ustawiania dedykowanych kontenerów w postaci czy <canvas></canvas>. Biblioteka sama dba o “wytworzenie” środowiska niezbędnego do renderingu obrazu.

W pierwszej kolejności musimy zadeklarować zmienne, które będą podstawowym budulcem naszej aplikacji 3D.

```
var camera,
    scene,
    renderer,
    mesh;
```

Można przejść zatem do zarządzania kamerą.

Robimy pierwszą aplikację w 3D - kamera

Następnie musimy zdefiniować kamerę oraz ustawić jej parametry.

```
camera = new THREE.PerspectiveCamera(70, window.innerWidth/window.innerHeight, 1, 1000);
camera.position.z = 350;
```

Pierwszy parametr oznacza tzw. fov czyli pionowe pole widzenia - większość gier fps pozwala na edycję tego parametru aby zwiększyć zasięg widzenia względem gracza. Następnie mamy proporcje ekranu (po to aby obraz był zawsze dobrze dopasowany do wymiarów ekranu).

Pozycję (w tym przykładzie głębię) okresla się przy pomocy modyfikacji osi z. Jeśli byśmy chcieli przesunąc kamerę w prawo należałoby zmodyfikować oś x.

Robimy pierwszą aplikację w 3D - scena

Co jednak nam po kamerze gdy nie będzie miała co “wyświetlać”. Kamera przedstawia to co znajduje się na scenie, trzeba zatem taką dodać.

```
scene = new THREE.Scene();
```

Posiadając już zadeklarowaną przestrzeń 3D możemy dodawać do niej obiekty.

```
scene.add(mesh);
```

Obiekt mesh jest naszym zbiorem (siatką) punktów, które mają być wyświetlane w przestrzeni twójwymiarowej.

Grupy takich punktów tworza polygony, które następnie mogą być kolorowane lub teksturowane (nakładanie obrazu).

Robimy pierwszą aplikację w 3D - tekstury cz. 1

Chcąc rozmawiać o teksturach należy zwrócić uawgę na nowe pojęcie czyli **tekstel**. Jest to najmniejszy element tekstury czyli **“piksel” tekstury**. Każda tekstura jest poddawana procesowi mipmappingu, który polega na wygenerowaniu z wejściowej tekstury mniejszych tekstur, czyli przeprowadzeniu operacji skalowania. Mipmapping możliwy jest tylko wtedy, gdy tekstura wejściowa ma rozmiar będący potęgą dwójki, np. 256×256, 64×64. Przykładowo, mając teksturę o wymiarach 128×128, wygenerowane zostaną tekstury 64×64, 32×32, 16×16, 8×8, 4×4, 2×2, 1×1, bo:

$128 = 2^7,\ 64 = 2^6,\ 32 = 2^5,\ 16 = 2^4,\ 8 = 2^3,\ 4 = 2^2,\ 2 = 2^1,\ 1 = 2^0$

Deklarując teksturę należy zwracać zatem uwagę na wymiary, jest to bardzo ważne.

Robimy pierwszą aplikację w 3D - tekstury cz. 2

Kolejną rzeczą jaką należy pamiętać to to, że 3D a 2D różni się trochę od siebie chociażby pod kątem transparentności. W grafice webowej 2D obraz PNG z transparentnością będzie wyświetlał się prawidłowo. W grafice 3D transparentność zostanie zastąpiona białym kolorem. Aby stworzyć teksturę z tzw. alphą należy utworzyć teksturę zawierającą maskę (format TGA).

Utwórzmy zatem testurę formatu JPG i nałóżmy ją na materiał.

```
var texture = new THREE.TextureLoader().load('textures/crate.jpg');
var material = new THREE.MeshBasicMaterial({ map: texture });
```

To jednak nie wszystko, aby móc nałożyć materiał z tekstują potrzebujemy obiekt.

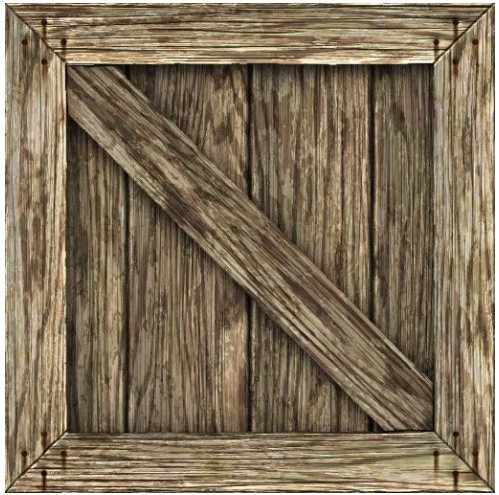
Robimy pierwszą aplikację w 3D - tekstury cz. 3

Na potrzeby przykładowej aplikacji stworzymy podstawową bryłę geometryczną (sześcian), która jest już zadeklarowana w bibliotece Three.js

```
var geometry = new THREE.BoxBufferGeometry(200, 200, 200);
mesh = new THREE.Mesh(geometry, material);
```

Mając bryłę możemy połączyć ją z naszym materiałem zawierającym teksturę. Sam materiał należy traktować jako tkaninę, która może mieć różny wzór, i która można następnie nałożyć na jakiś obiekt.

W naszym przykładzie naszym obiektem będzie sześcian dlatego też tekstura będzie w formie skrzynki.



Robimy pierwszą aplikację w 3D - renderowanie

Mamy już wszystko czas więc wyświetlić, a raczej poprawniej mówiąc wyrenderować naszą scenę z obiektem.

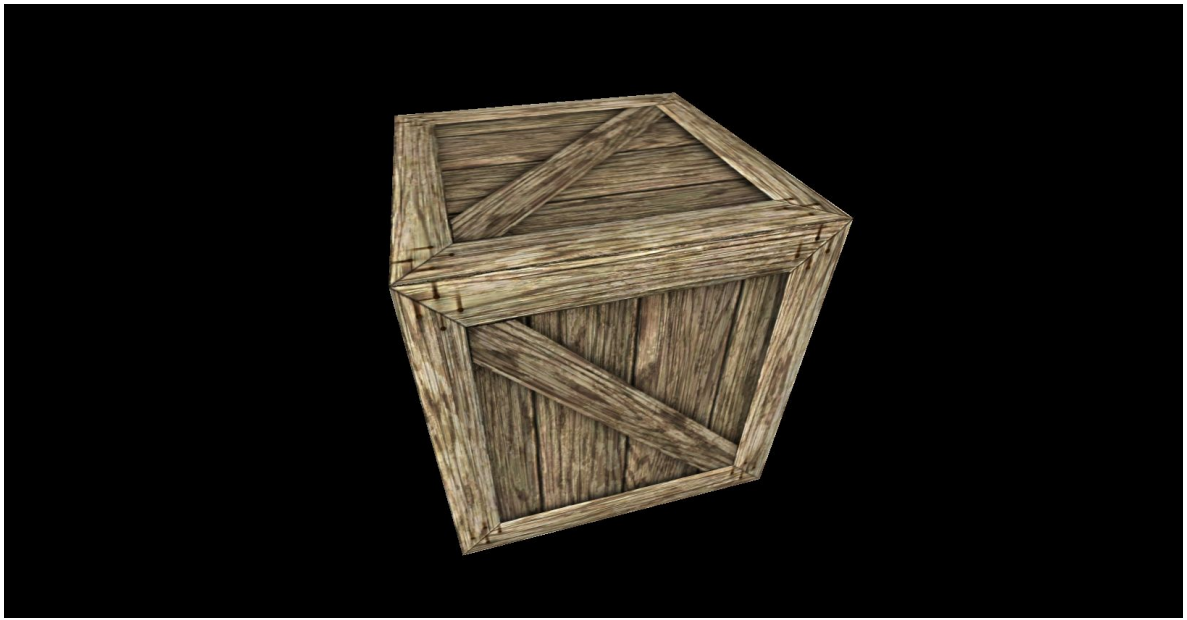
```
renderer = new THREE.WebGLRenderer();
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(window.innerWidth, window.innerHeight);
```

Określamy nasz typ renderowania (w tym przypadku jest to WebGL) następnie określamy proporcje ekranu oraz wymiary i to wszystko.

Sam WebGL odpowiada już za to aby aplikacja była odświeżana automatycznie, nie potrzeba zatem tworzyć żadnych interwałów czasowych, które będą odpowiedzialne za odświeżanie renderingu.

Robimy pierwszą aplikację w 3D - gotowy efekt

Tak oto prezentuje się gotowa wyrenderowana aplikacja.



Głębia dodaje możliwości, głębia zwiększa zyski

Czy warto interesować się 3D?

Wszystko tak na prawdę zależy od tego czym programista chce się zajmować. Warto jednak mieć wiedzę, nawet podstawową czym ów świat 3D jest.

Strony internetowe bardzo szybko ewoluują, co skutkuje tym, że wielkie korporacje mogą chcieć wykorzystywać elementy 3D lub nawet całe sceny do kreowania swoich produktów i ich promocji.

Kolejnym krokiem są gry. Wszystko zmieża w tym kierunku, że to przeglądarka internetowa będzie swoistym hubem pozwalającym na granie na zasadzie “click&play” bezpośrednio w okienku FireFoxa lub Chrome.

Dlatego tak! Warto interesować się 3D zwłaszcza tym webowym.

Dziękuję za uwagę

Omówiony przykład jak także inne są dostępne pod adresem:

https://github.com/mrygielski/ThreeJS/

Dokumentacja oraz główne repozytorium Three.js dostępne jest pod adresem:

https://github.com/mrdoob/three.js/

Dzięki za uwagę! (͡° ͜ʖ ͡°)