Witold Alda

Grafika komputerowa. Laboratorium 4.

Nawigacja

W tym zestawie przykładów patrzymy na funkcje pozwalające na nawigowanie na scenie. Dostępnych jest sporo modułów obsługujących nawigowanie za pomocą klawiatury i myszki.
Nazwy niektórych z nich wraz informacją o podstawowej funkcjonalności umieszczono poniżej. Chwilowo opisy skopiowane po angielsku.

Name	Description
`FirstPersonControls`	Controls that behave like those in first person shooters. Move around with the keyboard, and look around with the mouse.
`FlyControls`	Flight simulator like controls. Move and steer with the keyboard and the mouse.
`RollControls`	A simpler version of the `FlyControls`. Allows you to move around and roll around the z-axis.
`TrackBallControls`	Most used controls, allow you to use the mouse (or the trackball) to easily move, pan, and zoom around the scene.
`OrbitControls`	Simulates a satellite in orbit around a specific scene. Allows you to move around with the mouse and keyboard.
`PathControls`	With this control the camera's position moves around a predefined path. You can compare it with a rollercoaster ride where you can look around you, but have no influence on your position.

W trzech przykładach wykorzystujemy kontrolery:

- TrackBallControls
- OrbitControls
- PointerLockControls

Przykłady pochodzą z zestawu załączonego na stronie threejs.org.
Dwa pierwsze przykłady `01_controls_trackball` oraz `02_controls_orbit` są bardzo podobne choć wykorzystują inne moduły. Oglądamy je tylko i nie modyfikujemy.

Najciekawszy jest przykład `03_controls_pointerlock`. Sama scena też zawiera kilka ciekawych elementów.

Przykład 01_controls-trackball

Przykład wykorzystuje biblioteczkę `TrackBallControls.js`. Sterowanie odbywa się za pomocą myszki, dość intuicyjnie.
Na początek należy ją dołączyć do skryptu:

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

a następnie skojarzyć z obserwatorem:

```
var trackballControls = new THREE.TrackballControls(camera);
```

Przy okazji można ustawić parametry związane z szybkością obrotów, zbliżania/oddalania i przesuwania:

```
trackballControls.rotateSpeed = 1.0;
```

```
trackballControls.zoomSpeed = 1.0;
trackballControls.panSpeed = 1.0;
```
Uaktualnienie położenia kamery może być zrobione w pętli renderowania, np w taki sposób:
```
var clock = new THREE.Clock();
function render() {
var delta = clock.getDelta();
trackballControls.update(delta);
requestAnimationFrame(render);
webGLRenderer.render(scene, camera);
}
```

Przykład 02_controls-orbit

Przykład ten jest praktycznie identyczny jak poprzedni, korzysta jedynie z innej biblioteki do nawigacji. Nawigować tu można i myszką, i strzałkami. Parametry dotyczące szybkości obrotów, przesuwania i przybliżania/oddalania i wartości domyślnych, najlepiej odczytać z pliku biblioteczki. Parametry są odrobinę inne niż w poprzednim przykładzie, ale różnice są kosmetyczne.

```
<script type="text/javascript"
src="OrbitControls.js"></script>
...
var orbitControls = new THREE.OrbitControls(camera);
orbitControls.autoRotate = true;
var clock = new THREE.Clock();
...
var delta = clock.getDelta();
orbitControls.update(delta);
```

Przykład 03_controls_pointerlock

Jest to najciekawszy przykład w zestawie. Nawigacja nawiązuje do gier First Person Shooter i pozwala na sterowanie kamerą w typowy dla nich sposób. Ruch myszką pozwala na rozglądanie się na boki oraz w górę i w dół. Klawiatura pozwala na przemieszczanie się kamery.

Opis modułu można znaleźć na stronie
http://www.html5rocks.com/en/tutorials/pointerlock/intro/

Ponadto w przykładzie można zwrócić uwagę na dodatkowe elementy graficzne:

- Podłoże jest zbudowane na siatce trójkątów, która została zdeformowana w każdym kierunku. Mamy więc nierówną powierzchnię z różnych trójkątów.
- Kolor każdego trójkąta jest interpolowany na podstawie kolorów wierzchołków, a te z kolei są kolorowane losowo – jednak, jak widać z pewnego zakresu. W przykładzie jest to zakres rozbielonej barwy niebieskawej – co zapewne ma przypominać powierzchnię śnieżną lub lodową. Wygenerowanie określonej barwy ułatwia użyty model HSL (zamiast RGB), który posługuje się parametrami Hue, Saturation, Light. Proponuję popatrzeć na opis modelu HSL w Wikipedii.

Do zrobienia.

- Proszę zmodyfikować przykład 03 przez zmianę kolorów, oświetlenia i ułożenia elementów sceny.
- Przede wszystkim proszę umieścić obiekt, który byłby przyklejony do kamery, znajdował się przed nią i udawał celownik, broń, deskę rozdzielczą samochodu, lub coś podobnego. Można to zrobić tak, że przed kamerą ustawiamy obiekt, np. kwadrat albo sześcian, odpowiednio go pozycjonujemy (w jakiej odległości powinien być przed kamerą), a następnie dodajemy go do obiektu `camera` analogicznie jak dodajemy obiekty do sceny: `camera.add(object)`. W ten sposób obiekt będzie poruszał się zgodnie z kamerą. Następnie na obiekt należy nałożyć wspomnianą teksturę celownika, pistoletu, kokpitu, samochodu od tyłu, etc. Tekstury proszę znaleźć w Internecie i ściągnąć je (warto wybrać tekstury z przezroczystym tłem).
  W jednej z modyfikacji, jako obiekt przed kamerą, można dodać kwadrat, a na niego nałożyć załączoną teksturę *maska.png.* Tekstura ma przezroczystości - wycięcia (czego przy podglądzie w niektórych aplikacjach nie widać), tak że po dołączeniu jej do sceny i odpowiednim przeskalowaniu, mamy wrażenie jakbyśmy obserwowali scenę przez otwory w masce (Batmana dajmy na to). Proszę pamiętać, żeby w materiale związanym z teksturą był ustawiony parametr `transparent:true`
- Ostatecznie proszę przygotować scenę – może być całkiem zmieniona – po której poruszamy się z jakimś efektownym obiektem przed kamerą.
- Wynik zawierający plik html i dodaną/dodane tekstury proszę przesłać do moodla do zadania Laboratorium 4.