Witold Alda

Grafika komputerowa. Laboratorium 6. Importowanie obiektów.

Ładowanie obiektów z zewnętrznych plików FBX – ćwiczenie do wykonania

Budowanie, zwłaszcza złożonych, obiektów geometrycznych w Three.js (nie mówiąc już o WebGL) jest bardzo uciążliwe.
Co prawda Three.js i pokrewne biblioteki zapewniają grupę narzędzi pozwalające na pewną automatyzację i wyjście ponad ręczne składanie obiektów z trójkątów, jednak i one nie są wygodne w stosowaniu. Raczej nadają się do budowy własnych aplikacji do modelowania. Do tych narzędzi należy między innymi:

klasa `THREE.ExtrudeGeometry,` do wyciągania powierzchni w 3D, https://threejs.org/docs/#api/en/geometries/ExtrudeGeometry .

wtyczka ThreeCSG.js do operacji boolowskich na obiektach: https://github.com/chandlerprall/ThreeCSG . ,

zagęszczanie siatki na podstawie `THREE.SubdivisionModifier :` https://gist.github.com/jackrugile/b40a07d6f6b5bc202b9d587aee14ce01

Podejście tu zaproponowane jest mniej ambitne i opiera się na importowaniu gotowych obiektów 3D, przygotowanych za pomocą aplikacji do modelowania, takich jak 3DS max, Blender, Maya, czy jakichś innych. Aplikacji takich jest sporo, a orientacyjną ich listę można znaleźć na `https://en.wikipedia.org/wiki/List_of_3D_modeling_software` .

Obiekty możemy modelować sami lub ściągnąć je z jednego z licznych repozytoriów internetowych.

W ramach ćwiczenia wykonamy jeden przykład, który ładuje animowany obiekt 3D w formacie FBX, za pomocą funkcji FBXLoader z biblioteki FBXLoader.js. Samym formatem FBX się nie zajmujemy, zwróćmy jednak uwagę, że three.js może importować wiele formatów 3D. Na stronie domowej three.js, w dziale Examples można znaleźć z piętnaście przykładów wczytywania różnych formatów modeli 3D.

W przykładzie, który zmieniamy jest wykorzystana struktura katalogów, która jest narzucana od wersji r116.

```
Nasz_folder
 |
 +- build
 |   |+-three.module.js
 |   |+-three.js
 |
 +-examples
   |
   +-jsm
   |
   +-controls
   | |
   | +-OrbitControls.js
   | +-...
   |
   +-loaders
   | |
   | +-FBXloader.js
   | +-...
   |
   ...
```

Przykład jest następujący.

W rozpakowanym katalogu `Lab06_Importowanie_fbxloader_wszib`, w podkatalogu `examples` uruchamiamy skrypt `webgl_loader_fbx.html`, który automatycznie ładuje tańczącego robota.
Robot pochodzi z sieciowej aplikacji Mixamo firmy Adobe.
Proszę wejść do Mixamo na http://mixamo.com. W tym celu trzeba się zarejestrować w aplikacji – ja to robię przez konto googlowe.
W Mixamo mamy do dyspozycji zestaw postaci humanoidalnych (około 100) i zestaw krótkich sekwencji ruchów (ponad 2000) o różnym charakterze (taniec, sport, walka). Postaci i sekwencje można dowolnie łączyć.

Proszę jedną lub dwie lub więcej postaci z jedną lub kilkoma sekwencjami ruchu, ściągnąć na swój komputer, w formacie FBX. Następnie proszę umieścić postaci na scenie, tak żeby:

- Wydłużyć sekwencję ruchu lub
- Ustawić kilka postaci, żeby synchronicznie tańczyły, lub
- Zamodelować scenę walki lub
- Wymyślić coś innego, ale fajnego
- Dobrze jest dodać proste otoczenie, żeby postaci nie poruszały się w pustce (podłogę, ścianę, sky-box).

W kodzie przykładu warto zwrócić uwagę na następujące elementy:

- Obiekt `mixer = new THREE.AnimationMixer(object);` `mixer` kontroluje animację naszej postaci. Jeśli postaci jest więcej należy użyć dla każdej z nich oddzielnego obiektu `mixer`, np. `mixer1`, `mixer 2`, …
- Postaci mogą być pozycjonowane na scenie, obracane i skalowane, jak wszystkie inne obiekty.
- W funkcji `animate();` wykorzystywany jest czas (a dokładnie przyrost czasu) z zegara systemowego w wywołaniu `mixer.update(delta);` który powoduje ustawienie i wyświetlenie postaci w stanie odpowiadającym danej chwili od początku obliczeń. Możemy manipulować parametrem delta w celu przyspieszenia/zwolnienia animacji postaci albo w

celu przesunięcia w czasie ruchu jednej postaci względem innej. Ten ostatni przypadek może być wykorzystany np. w scenie walki dwóch bohaterów.

Tak zmodyfikowany przykład jest celem naszego ćwiczenia. Proszę o wysłanie pliku html wraz z postaciami w formacie fbx do zadania z Laboratorium 06.

UWAGA: Może się zdarzyć, że postaci w formacie fbx zajmą zbyt wiele miejsca – zwłaszcza jeśli użyjemy ich kilka - i nie da się spakowanego pliku załadować do UPELa (jedna postać fbx zajmuje od kilku do kilkudziesięciu MB). W takiej sytuacji proszę przesłać swój przykład w jakieś wygodne dla siebie miejsce w chmurze, a do zadania w UPELU wysłać tylko link.

Inne formaty obiektów

Poniżej przytaczam informację o innych popularnych formatach modeli 3D (nieanimowanych). Nie wykorzystujemy ich w tym ćwiczeniu, a opis pochodzi z wcześniejszych wersji laboratorium. Zostawiam je bez wyraźnego celu, ale nie zajmują wiele miejsca i są *à propos* .

Z biblioteką three.js stowarzyszonych jest kilka dodatkowych funkcji, które obsługują te formaty 3D. Są one umieszczone w plikach, które mają w swojej nazwie człon *Loader* (np. OBJLoader.js, MTLLoader.js, etc) i służą do ładowania obiektów, a także w plikach z członem *Exporter* (np. OBJExporter.js) służących oczywiście do zapisywaniu obiektów na dysku.

W WebGL z biblioteką Three.js mamy możliwość obsłużenia różnych formatów 3D. Poniżej wymienione są tylko niektóre:

Nazwa formatu	Opis
JSON	Natywny format Three.js
OBJ/MTL	Jeden z najpopularniejszych formatów 3D utworzony przez firmę Wavefront Technologies. Pliki OBJ opisują geometrię, a pliki MTL – parametry materiałowe.
Collada	Popularny format zapisujący dane w XML
STL	STereoLithography, znajduje zastosowanie m. in. w opisie modeli dla drukarek 3D.
CTM	Format zdefiniowany dla biblioteki openCTM (Open Compressed Triangle Mesh). Nie jestem pewien czy jeszcze działa obecnie w Three.js (nie ma obecnie przykładu na stronie), ale działał: http://www.yanhuangxueyuan.com/threejs/examples/webgl_loader_ctm.html
VTK	Format zdefiniowany dla biblioteki Visualization Toolkit, specyfikuje wierzchołki i ściany. Three.js obsługuje starszą wersję formatu w ASCII.
PDB	Protein Databank, format zaprojektowany do opisu złożonych molekuł, zwłaszcza białek.
PLY	Polygon file format. Najczęściej jest używany do przechowywania danych ze skanerów 3D.

Poniżej opisany jest tylko jeden, być może najpopularniejszy format . Pozostałe niestety nie.

Format OBJ/MTL

OBJ i MTL są stowarzyszonymi formatami zwykle używanymi wspólnie. Plik OBJ opisuje geometrię, a MTL definiuje użyte materiały. Oba formaty zapisywane są tekstowo. Przykładowy fragment OBJ umieszczony jest poniżej. Dokładniejszą specyfikację format można znaleźć np. w http://en.wikipedia.org/wiki/Wavefront_.obj_file

```
v -0.032442 0.010796 0.025935
v -0.028519 0.013697 0.026201
v -0.029086 0.014533 0.021409
usemtl Material
s 1
f 2731 2735 2736 2732
```

```
f 2732 2736 3043 3044
```

Wiersze z identyfikatorem ‘v’ oznaczają wierzchołki i ich współrzędne, a wiersze z identyfikatorem ‘f’, ścianki (faces) z numerami wierzchołków, które je tworzą. Z kolei plik MTL definiuje materiał np. w następujący sposób:

```
newmtl Material
Ns 56.862745
Ka 0.000000 0.000000 0.000000
Kd 0.360725 0.227524 0.127497
Ks 0.010000 0.010000 0.010000
Ni 1.000000
d 1.000000
illum 2
```

Specyfikację pliku MTL można obejrzeć np. na stronie Paule Bourke: http://paulbourke.net/dataformats/mtl/

OBJ i MTL są odczytywane przez Three.js. Jednocześnie mogą być tworzone przez wiele programów do modelowania 3d, m. in. przez Blendera i Art of Illusion. Three.js ma dwie oddzielne funkcje ładujące obiekty w tych formatach. Jeżeli chcemy użyć tylko pliku OBJ z geometrią, należy użyć OBJLoader i dodać go do naszego kodu:

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

Samo importowanie może wykorzystać następujący schemat:

```
var loader = new THREE.OBJLoader();
loader.load('pinecone.obj', function(geometry) {
var material = new THREE.MeshLambertMaterial({
color: 0x5C3A21
});
// geometry is a group of children.
// If a child has one additional child it's probably a mesh
geometry.children.forEach(function(child) {
if (child.children.length == 1) {
if (child.children[0] instanceof THREE.Mesh) {
child.children[0].material = material;
}
}
});
geometry.scale.set(100, 100, 100);
geometry.rotation.x = -0.3;
scene.add(geometry);
});
```

Użycie klasy OBJMTLLoader służy do załadowania modelu oraz dołączonego materiału i tekstury.

Musimy dołożyć funkcje ładujące:

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

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

A kod może wyglądać następująco:

```
var loader = new THREE.OBJMTLLoader();
loader.addEventListener( 'load', function ( event ) {
var object = event.content;
var wing2 = object.children[5].children[0];
var wing1 = object.children[4].children[0];
wing1.material.alphaTest = 0.5;
wing1.material.opacity = 0.6;
wing1.material.transparent = true;
wing2.material.alphaTest = 0.5;
wing2.material.opacity = 0.6;
wing2.material.transparent = true;
object.scale.set(140, 140, 140);
object.rotation.x = 0.2;
object.rotation.y = -1.3;
scene.add(object);
});
loader.load('butterfly.obj',
'butterfly.mtl');
```