Witold Alda

Grafika komputerowa. Laboratorium 3.

Mapowanie tekstur w three.js

Mapowanie tekstur na obiekty jest najważniejszym sposobem poprawiania realizmu sceny. Tekstury w bibliotece *three.js* mogą być używane w różny sposób. Podstawowym jest nakładanie tablicy zawierającej wczytany obraz na siatkę tworzącą powierzchnię obiektu

Ładowanie tekstur z użyciem *three.js* może być zrealizowane za pomocą funkcji `loadTexture()`, (to jest starsza wersja, *deprecated*, ale ciągle w użyciu) lub `TextureLoader().load`. Sam obiekt tekstury jest dołączany jako atrybut materiału parametrem `map`:

```
function createMesh(geom, imageFile) {
      var texture = THREE.ImageUtils.loadTexture
            ("textures/" + imageFile)
      var mat = new THREE.MeshPhongMaterial();
      mat.map = texture;
      var mesh = new THREE.Mesh(geom, mat);
return mesh;
}
```

Funkcja `loadTexture()` pozwala na czytanie i ładowanie plików w formacie JPG, GIF i PNG.
Analogicznie możemy napisać:

```
function createMesh(geom, imageFile) {
      var texture = THREE.Texture.Load().load
            ("textures/" + imageFile)
      var mat = new THREE.MeshPhongMaterial();
      mat.map = texture;
      var mesh = new THREE.Mesh(geom, mat);
return mesh;
}
```

Przykład 01 – Basic Texture

Przykład zawiera podstawowe mapowanie tekstur z domyślnymi wartościami parametrów. Warto może zwrócić uwagę, że sposób mapowania jest określony odrębnie dla różnych standardowych obiektów three.js. W przypadku sześcianu cała tekstura mapuje się na całą ścianę, w przypadku sfery występuje mapowanie sferyczne. Dla dwudziestościanu automatyczne mapowanie nie zawsze wydaje się naturalne.
Drobiazgi do zrobienia:

- Można przeskalować obiekty i zobaczyć jak zachowują się mapowanie na nie tekstury
- Można zmienić wartości w ustawieniu powtórzeń tekstury `texture.repeat.set( 1, 1 );` i uaktywnić atrybuty `wrapS` i `wrapT` dla wartości `RepeatWrapping` i `ClampToEdgeWrapping`.

Przykład 02 – Bump mapping (mapowanie wypukłości)

Drugi przykład pozwala na modyfikowanie normalnych w jednym z obiektów.

Funkcja ładowania tekstur wywołana jest dwukrotnie:

1. Najpierw wczytywana jest podstawowa tekstura i dołączana parametrem `map`.
2. Następnie doładowana tekstura z wypukłościami i dołączana parametrem `bumpMap`.

```
function createMesh(geom, imageFile, bump) {
      var texture = THREE.ImageUtils.loadTexture(
      "textures/" + imageFile)
      var mat = new THREE.MeshPhongMaterial();
      mat.map = texture;
      var bump = THREE.ImageUtils.loadTexture(
      "textures/" + bump)
      mat.bumpMap = bump;
```

Proszę zobaczyć jak wygląda tekstura bump i zmienić ją na inną (niekoniecznie przeznaczoną do bump mappingu). Przypomnijmy, że tekstura do bump mappingu jest monochromatyczna i reprezentuje mapę wysokości. Służy ona do modyfikowania wektorów normalnych na powierzchni obiektów w następujący sposób:

- Odczytuje wysokość z mapy wysokości w punkcie odpowiadającym położeniu na powierzchni.
- Oblicza wektor normalny do mapy wysokości (zwykle używa się metody różnic skończonych dla sąsiednich wysokości).
- Dodaje obie normalne – tę z mapy wysokości i z rzeczywistej powierzchni. Wynik znormalizuje, tak żeby nie wpływał na jasność.
- Oblicza oświetlenie według nowych normalnych.

Przykład 03 Normal mapping (mapowanie normalnych)

Mapowanie normalnych jest rozszerzeniem mapowania wypukłości, pozwalającym na dokładniejsze odwzorowanie szczegółów. Źródłem modyfikacji normalnych jest tu pełna mapa normalnych, z trzema współrzędnymi *x*, *y*, *z*, zapisanymi w obrazie RGB. Mapy normalnych często mają większą rozdzielczość niż obiekty, na które są mapowane (obiekty low-poly są wtedy wzbogacane szczegółami oświetlenia na podstawie mapy. Mapy normalnych dołącza się prawie identycznie jak mapy wypukłości – parametr **`bumpMap`** jest zastąpiony parametrem **`normalMap:`**

```
function createMesh(geom, imageFile, normal) {
      var t = THREE.ImageUtils.loadTexture("textures/" + imageFile);
      var m = THREE.ImageUtils.loadTexture("textures/" + normal);
      var mat2 = new THREE.MeshPhongMaterial({
                  map: t, normalMap: m
    });
      var mesh = new THREE.Mesh(geom, mat2);
      return mesh;
```

Przykłady 04, 05 – Sky box

Kolejne dwa przykłady pokazują użycie otoczenia typu Sky box. Pochodzą one z zestawu przykładów, których autorem jest Lee Stemkoski (z lekkimi modyfikacjami) i niestety widać w nich pewne przestarzałe rozwiązania.=.
W przykładzie 04 proszę zwrócić uwagę jak jest nakładanych 6 tekstur na obiekt typu `Cube` i która strona jest renderowana. 6 tekstur jest umieszczonych w tablicy `materialArray:`

```
var materialArray = [];
for (var i = 0; i < 6; i++)
          materialArray.push( new THREE.MeshBasicMaterial({
               map: THREE.ImageUtils.loadTexture( imagePrefix +
directions[i] + imageSuffix ),
               side: THREE.BackSide
          }));
```

Renderowanie wykonuje się na wewnętrznej stronie, bo znajdujemy się wewnątrz sześcianu. Tablica `materialArray` następnie jest łączona w jeden obiekt typu `Material` za pomocą metody `MeshFaceMaterial();`, obiekt ten jest następnie dodawany do sześcianu:

```
var skyMaterial = new THREE.MeshFaceMaterial( materialArray );
     var skyBox = new THREE.Mesh( skyGeometry, skyMaterial );
     scene.add( skyBox );
```

W przykładzie 05 mamy umieszczone obiekty zwierciadlane i przejrzyste. Proszę zauważyć jak definiowany jest materiał po to, żeby uzyskać pożądany efekt mapowania tekstury ze SkyBoxu na obiekty na scenie.
Zwróćmy uwagę na obiekt `CubeCamera` i własność `renderTarget` w parametrze `envMap`, które umożliwiają mapowanie tekstur SkyBox na obiekty, w taki sposób, że obiekty wyglądają jak lustrzane:

```
var sphereGeom =  new THREE.SphereGeometry( 50, 32, 16 );
mirrorSphereCamera = new THREE.CubeCamera( 0.1, 5000, 512 );
scene.add( mirrorSphereCamera );
var mirrorSphereMaterial = new THREE.MeshBasicMaterial( { envMap:
mirrorSphereCamera.renderTarget } );
mirrorSphere = new THREE.Mesh( sphereGeom, mirrorSphereMaterial );
mirrorSphere.position.set(75,50,0);
mirrorSphereCamera.position = mirrorSphere.position;
scene.add(mirrorSphere);
```

Do zrobienia

W ramach ćwiczenia proszę wypełnić jedną scenę otoczoną skyboxem, np. tę z przykładu 05, kilkoma obiektami o różnych własnościach materiałowych: ze zwykłą teksturą, z bump-mappingiem lub normal mappingiem, zwierciadlaną i przejrzystą. Proszę wprowadzić obroty lub poruszanie się obiektów, tak żeby można było zauważyć zmiany w oświetleniu oraz odbiciu światła pochodzącego od tekstur sky-boxa. Proszę uzyskac efekt, którym poruszające się lub obracające obiekty lustrzane odbijają się wzajemnie od siebie.

W prostopadłoscianie widać artefakty w mapowaniu otoczenia wynikające m.in. z tego, że ściana prostopadłościanu jest zbudowana z dwóch trójkątów. Proszę zmienić siatkę budującą prostopadłościan i zobaczyć jak to wpływa na odbicia.

Proszę sprawdzić w dokumentacji threejs.org co znaczą parametry obiektu CubeCamera, zmienić je i ocenić efekty zmian
Plik .html (bez bibliotek) z tej jednej sceny proszę podesłać na moodle'a.