

# Grafika komputerowa. Laboratorium 2. Oświetlenie w Three.js (release 116)

---

(październik 2024)

## Elementy biblioteki three.js (krótkie przypomnienie)

W bibliotece Three.js scena jest budowana z obiektów. Sama scena jest również obiektem – kontenerem – w którym umieszczane są inne obiekty, zarówno te geometryczne jak i pozostałe charakteryzujące scenę.

Podstawowe obiekty, które tworzymy na podstawie wbudowanych w bibliotekę klas, to:

- Renderery (są przynajmniej dwa: Canvas renderer i WebGL renderer, ten ostatni z większymi możliwościami, ale też i potrzebami odnośnie sprzętu)
- Formy geometryczne, gotowe, zapewnione przez bibliotekę (takie jak sześciian, sfera, itp.) oraz tworzone ręcznie jako siatka wielokątów.
- Obiekty przedstawiające obserwatora sceny (*camera*), jego położenie, punkt, na który patrzy, kąt widzenie, etc.
- Źródła światła, które oświetlają scenę.
- Materiały przypisane formom geometrycznym, określające ich kolor i sposób odbijania światła.

Obiekty three.js mają swoje zmienne składowe, którym należy nadać określone wartości oraz metody. **Dobrą dokumentacją opisującą między innymi strukturę obiektów jest ta na stronie [Threejs.org](https://threejs.org).** Będziemy z niej korzystać przy wykonaniu ćwiczeń.

W ćwiczeniach korzystamy z gotowych kształtów zdefiniowanych w bibliotece. W dokumentacji na stronie threejs.org można je znaleźć w grupie Geometries (dawniej Extras/Geometries).

Dostępne są następujące obiekty:

- 2D:
  - PlaneGeometry
  - CircleGeometry
  - ShapeGeometry
- 3D:
  - BoxGeometry
  - SphereGeometry
  - CylinderGeometry
  - TorusGeometry
  - TorusKnotGeometry
  - PolyhedronGeometry
  - IcosahedronGeometry
  - OctahedronGeometry
  - TetraHedronGeometry

Każdy z kształtów ma właściwe sobie parametry – do odczytania w dokumentacji.

Umieszczane na scenie obiekty są obiektami klasy *Mesh* i składają się z kształtu (geometrii) oraz materiału np. w taki sposób:

```
const cubeGeometry = new THREE.BoxGeometry(4,4,4);
const cubeMaterial= new THREE.MeshLambertMaterial({color:0xff0000});
const cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
```

Materiały, które charakteryzują powierzchnię obiektu decydują o kolorze, sposobie odbijania światła, połyskliwości, przezroczystości, itp. Ich lista (do przeglądu w threejs.org) obejmuje ponad 10 pozycji (w r97 jest ich już 16, w r126 – 18 i na razie nie rośnie). Najważniejsze dla nas w tej chwili są *MeshLambertMaterial* i *MeshPhongMaterial*, ewentualnie jeszcze *MeshBasicMaterial*.

*MeshBasicMaterial* jest najprostszy i nie pozwala na wykorzystywanie źródeł światła. Jednak pozwala na nadawanie materiałowi koloru i to w dość złożony sposób, np. poprzez użycie map barwnych.

*MeshLambertMaterial* oprócz możliwości jakie daje *MeshBasicMaterial*, odbija światło w sposób rozproszony, a *MeshPhongMaterial* dodatkowo jeszcze w sposób połyskliwy. Trzeba jednak zauważyć, że te dwa ostatnie materiały w inny sposób traktują obliczanie oświetlenia, również tego rozprozonego. *MeshLambertMaterial* oblicza odbicia w węzłach siatki i je interpoluje liniowo pomiędzy nimi (wykonuje interpolację Gourauda), *MeshPhongMaterial* oblicza oświetlenie w pikselach na podstawie interpolowanych liniowo normalnych (wykonuje interpolację Phonga, której nie należy mylić z modelem oświetlenia połyskliwego Phonga; ale i tak większość osób myli 😊).

Obiekty klasy *Mesh* możemy modyfikować stosując właściwe dla nich atrybuty:

Atrybut	Opis
position	Okręsła położenie x,y,z obiektu, względem obiektu – rodzica. Zazwyczaj obiektem odniesienia jest <i>Scene</i> .
rotation	Okręsła obrót obiektu wokół każdej z osi.
scale	Okręsła skalowanie obiektu wzduż każdej z osi.
translateX(amount)	Przesuwa obiekt wzduż osi x o zadaną wielkość.
translateY(amount)	Przesuwa obiekt wzduż osi y o zadaną wielkość
translateZ(amount)	Przesuwa obiekt wzduż osi z o zadaną wielkość

Położenie obiektu można ustawić na trzy sposoby.

Pierwszy:

```
cube.position.x=10;
cube.position.y=3;
cube.position.z=1;
```

Drugi:

```
cube.position.set(10,3,1);
```

Trzeci

```
cube.postion=new THREE.Vector3(10,3,1);
```

## Narzędzia pomocnicze: dat.GUI.js, stats.js

W przykładach wykorzystywane są mikro-biblioteki, które nieco poprawiają interfejs użytkownika. Pierwszą z nich jest dat.GUI.js. Biblioteka umożliwia utworzenie okien dialogowych, w których można zmieniać wartości wybranych zmiennych – przez podanie nowej wartości, za pomocą suwaka lub przez wybór z listy.

stats.js jest z kolei niewielką klasą, która służy do wyświetlania na ekranie okienka z informacją o wydajności aplikacji wyrażonej w FPS (*frames per second*) lub w ms na klatkę.

W śladowych ilościach, w niektórych przykładach wykorzystany jest popularny framework JavaScriptu – jQuery.js

## Modele oświetlenia

Zasadniczym elementem tego laboratorium jest zbadanie możliwości oświetlenia dostępnego w three.js. Zwróćmy też uwagę, że ostateczny efekt oświetlenia zależy od zarówno od charakterystyki źródeł światła jak i charakterystyki odbicia od obiektów, czyli parametrów materiałowych.

Wypada zwrócić uwagę, że modele oświetlenia w bibliotece Three.js zmieniają się częściej niż inne moduły, nie zawsze zachowując kompatybilność z poprzednimi wersjami. W tym przykładzie odwołuję się do wersji 97 Three.js, aktualnej w październiku 2017.

Tradycyjnie oświetlenie w Three.js ogranicza się do modelu lokalnego. Jednak w porównaniu z podstawowym modelem ADS (*Ambient, Diffuse, Specular*) pozwala na prawie automatyczne rzucanie cieni, uwzględnianie załamania światła w obiektach przejrzystych, a także na użycie płaskich źródeł światła o skończonej powierzchni. Są to dość istotne i praktyczne rozszerzenia.

### Rodzaje źródeł światła: *AmbientLight*

Jest to światło otoczenia, które jest formą poświaty, które ma źródło w każdym punkcie przestrzeni i świeci jednakowo w każdym kierunku.

Dziedziczenie klas jest następujące:

Object3D → Light → AmbientLight

Przykład konstruktora AmbientLight:

```
const light = new THREE.AmbientLight( 0xff8080, 1 );
scene.add( light );
```

Parametry konstruktora są następujące:

AmbientLight( color : Integer, intensity : Float)

gdzie:

Atrybut	Opis
color	Kolor światła RGB, domyślnie biały 0xFFFFFFFF
intensity	Intensywność światła, domyślnie 1.0

## Rodzaje źródeł światła: *PointLight*

Jest to światło punktowe, które świeci we wszystkie strony

Dziedziczenie klas jest następujące:

Object3D → Light → PointLight

Przykład konstruktora PointLight:

```
var light = new THREE.PointLight( 0xff0000, 1, 100 );
light.position.set( 50, 50, 50 );
scene.add( light );
```

Parametry konstruktora są następujące:

```
PointLight( color : Integer, intensity : Float, distance
: Number, decay : Float )
```

gdzie:

Atrybut	Opis
color	Kolor światła RGB, domyślnie biały 0xFFFFFFFF
intensity	Intensywność światła, domyślnie 1.0
distance	Odległość, przy której intensywność spada do 0. Jeśli <code>distance=0</code> , to światło świeci nieskończoność daleko.
decay	Sposób w jaki intensywność światła maleje z odlegością: domyślnie liniowo (1), alternatywnie kwadratowo (2)
position	Położenie źródła światła.

## Rodzaje źródeł światła: *SpotLight*

Szczególnie ciekawe dla SpotLight są poniższe atrybuty:

Atrybut	Opis
castShadow	Jeśli <code>true</code> – światło rzuca cienie
shadow.camera.near	Od jakiej odległości od źródła światła wyliczane są cienie
shadow.camera.far	Do jakiej odległości od źródła światła wyliczane są cienie
shadow.camera.fov	W jakim kącie wyliczane są cienie
target	Kierunek, w którym świeci SpotLight (target musi być obiektem)

angle	Kąt stożka światła w radianach(domyślnie $\pi/3$ )
penumbra	Osłabienie intensywności oświetlenia przy oddalaniu się od środka
shadow.camera	Jako obiekt CameraHelper. Wyświetlają się linie pomocnicze pokazujące ostrosłup światła (patrz przykład 1)
shadow.mapSize.width	Liczba pikseli użytych do tworzenia cienia
shadow.mapSize.height	Liczba pikseli użytych do tworzenia cienia

---

## Przykłady do zmodyfikowania

W ramach ćwiczenia ograniczymy się do modyfikacji tylko jednego (pierwszego) przykładu. Pozostałe przykłady zostawiamy jedynie do przejrzenia. Pewne zmiany zaproponowane do zrobienia w przykładzie pierwszym zostały zrealizowane w następnych.

### 01 ambient and spotlight

Scena przedstawia 3 obiekty geometryczne: płaszczyznę z klasy *PlaneGeometry*, sześcian z klasy *BoxGeometry*, sferę z klasy *SphereGeometry*.

W przykładzie – wbrew nazwie, oprócz światła ambient, użyto również źródła światła typu „reflektor” *SpotLight* oraz zastosowano materiał, który odbija światło w sposób rozproszony, *MeshLambertMaterial*. Zmiany właściwie wyłącznie dotyczą oświetlenia *SpotLight*.

Wprowadzono również prostą animację. W przykładzie, zmieniające się położenia, obroty, etc. są przypisane obiektom w funkcji *render()*, a z kolei ciąg tych zmian (animacja) jest zrealizowany wywołaniem:

```
requestAnimationFrame(render);
```

umieszczonego wewnątrz funkcji *render()*. Użycie *requestAnimationFrame(render)* uruchamia animację z prędkością 60 fps (jeśli komputer daje radę).

#### Do zrobienia:

W ramach ćwiczenia proszę (korzystając z dokumentacji na stronie [threejs.org](http://threejs.org) i uwag powyżej) zmienić różne parametry oświetlenia. W szczególności:

- Proszę zmienić parametry źródła światła *SpotLight*: jego położenie i kąt stożka światła na mniejszy, np. 0.2 do 0.25. Pomoce w śledzeniu stożka światła może być ustawienie parametru *shadow.camera* jako obiekt *CameraHelper*, co pozwala na wyświetlenie linii w przybliżeniu pokazujących stożek (bardziej ostrosłup), wewnątrz którego obiekty mogą rzucić cień.
- **Proszę uzyskać ślad kręgu światła na płaszczyźnie i obiektach.** Należy zrobić to dla obu rodzajów materiałów, tzn. *MeshLambertMaterial* i *MeshPhongMaterial*. Warto przy tym zwrócić uwagę na to, że zachowują się one odmiennie. Dla *MeshLambertMaterial* warunkiem koniecznym uzyskania śladu kręgu światła jest nałożenie na płaszczyznę siatki o większej gęstości (domyślnie *PlaneGeometry* ma tylko cztery węzły, trzeba sięgnąć do referencji w [threejs.org](http://threejs.org) i doczytać resztę parametrów). *MeshLambertMaterial* interpoluje oświetlenie między węzłami., *MeshPhongMaterial* odwołuje się do pikseli i nie wymaga gęstszej siatki w

*PlaneGeometry*. Proszę poeksperymentować i porównać wyniki uzyskiwane przy obu materiałach. Na koniec można wybrać ten, który nam się bardziej podoba.

- Proszę zmienić obiekt, na który świeci **SpotLight** (przez parametr **target**). Na przykład proszę wybrać sferę
- Proszę zobaczyć jakie parametry sterują rzucaniem cienia (jest ich kilka). Czy można rzucić np. cień sześcianu na sferę?
- Proszę zmienić materiał z **MeshLambertMaterial** na **MeshPhongMaterial** i uzyskać odblaski na sześcianie i sferze (pomocne będą parametry **specular** i **shininess**).
- Jeżeli cienie wyglądają zbyt topornie można spróbować zwiększyć parametry **shadow.mapSize.width** i **shadow.mapSize.height**. Należy też zwrócić uwagę, żeby obszar cienia był objęty przez źródło światła, w czym mogą pomóc parametry **shadow.camera.near**, **shadow.camera.far** i **shadow.camera.fov**. Proszę poeksperymentować z parametrami, tak żeby cienie wyglądały ładnie i naturalnie.
- Proszę do stożkowego światła **SpotLight** dodać transparentny stożek, który udawałby snop światła jaki czasem widać przy rozpraszaniu na różnych drobnych obiektach (kurzu, kropelkach) Przejrzystość częściową można uzyskać za pomocą parametrów **transparent** i **opacity** przy definiowaniu materiału, np. w sposób 

```
new THREE.MeshLambertMaterial( { opacity:0.6, color: 0x44ff44, transparent:true } );
```

 Proszę spowodować, żeby źródło światła poruszało się po okręgu razem z dołączonym stożkiem – snopem światła. Ten punkt jest dosyć uciążliwy i niewdzięczny w realizacji. Traktuję go jako nadobowiązkowy
- **Proszę dodać więcej ruchomych obiektów i ruchomych, kolorowych źródeł światła. Taką rozbudowaną scenę proszę przysłać jako zadanie z tego laboratorium.** Nadprogramowo może być w niej snop światła wspomniany w poprzednim punkcie. Ciekawie mogłaby wyglądać przy tym scena nocna.