Konstruowanie obiektów z użyciem Three.js

May 8, 2019

1 Introduction

Celem jest tworzenie obiektów za pomocą biblioteki Three.js

Zobaczymy, jak utworzyć nowe geometrie siatki od podstaw. Przyjrzymy się także niektórym innym wsparciem, które three.js zapewnia do pracy z obiektami i materiałami.

2 Indeksowane zestawy ścian

Siatka (mesh) w three.js jest indeksowanym zestawem ścian. W siatce three.js wszystkie wielokąty są trójkątami.

Geometria w three.js jest obiektem typu `THREE.Geometry`.

Dowolny obiekt geometrii zawiera tablicę wierzchołków, reprezentowanych jako obiekty typu `THREE.Vector3`.

Dla geometrii siatki zawiera ona również tablicę ścian, reprezentowanych jako obiekty typu `THREE.Face3`. Każdy obiekt typu `Face3` określa jedną z trójkątnych powierzchni geometrii. Trzy wierzchołki trójkąta są określone przez trzy liczby całkowite. Każda liczba całkowita jest indeksem w tablicy wierzchołków geometrii. Trzy liczby całkowite można określić jako parametry konstruktora `THREE.Face3`. Na przykład,

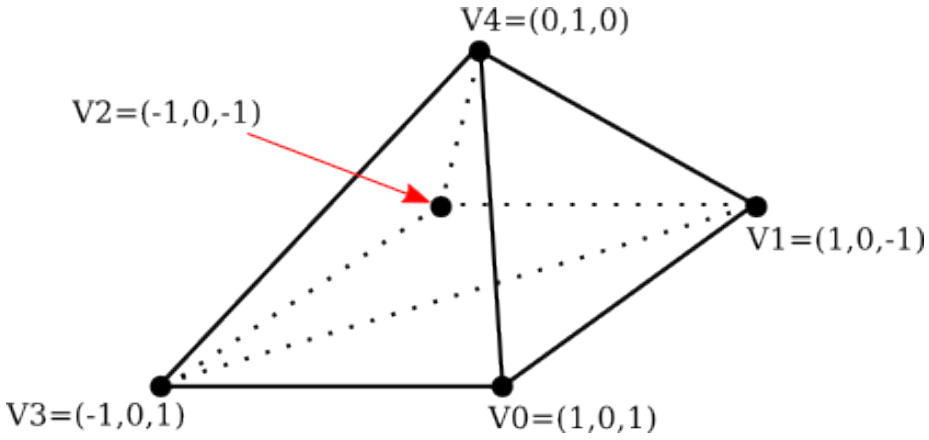
```
var f = new THREE.Face3(0, 7, 2);
```

Trzy indeksy są przechowywane jako właściwości `f.a, f.b` i `f.c` obiektu ściany. Jako przykład zobaczmy, jak bezpośrednio utworzyć geometrię three.js dla tej piramidy:

Należy zauważyć, że dolna powierzchnia piramidy, która jest kwadratem, musi być podzielona na dwa trójkąty, aby piramida była reprezentowana jako geometria siatki. Jeśli `pyramidGeom` jest obiektem geometrii dla tej piramidy, to `pyramidGeom.vertices` jest tablicą wierzchołków, a `pyramidGeom.faces` jest tablicą ścian. Mając to na uwadze, możemy zdefiniować:

```
var pyramidGeom = new THREE.Geometry();

pyramidGeom.vertices = [  // array of Vector3 giving vertex coordinates
```




```
        new THREE.Vector3( 1, 0, 1 ),    // vertex number 0
        new THREE.Vector3( 1, 0, -1 ),   // vertex number 1
        new THREE.Vector3( -1, 0, -1 ),  // vertex number 2
        new THREE.Vector3( -1, 0, 1 ),   // vertex number 3
        new THREE.Vector3( 0, 1, 0 )     // vertex number 4
    ];

pyramidGeom.faces = [  // array of Face3 giving the triangular faces
        new THREE.Face3( 3, 2, 1 ),  // first half of the bottom face
        new THREE.Face3 3, 1, 0 ),   // second half of the bottom face
        new THREE.Face3( 3, 0, 4 ),  // remaining faces are the four sides
        new THREE.Face3( 0, 1, 4 ),
        new THREE.Face3( 1, 2, 4 ),
        new THREE.Face3( 2, 3, 4 )
    ];
```

Należy zauważyć, że **kolejność wierzchołków** na powierzchni nie jest całkowicie dowolna: powinny być wymienione w *kolejności przeciwnej do ruchu wskazówek zegara*, jak widać z przodu ściany, to znaczy patrząc na ścianę z zewnątrz piramidy.

Ta geometria piramidy, jak podano, będzie działać z `MeshBasicMaterial`, ale do pracy z podświetlonymi materiałami, takimi jak `MeshLambertMaterial` lub `MeshPhongMaterial`, geometria wymaga wektorów normalnych. Jeśli geometria nie ma wektorów normalnych, materiały `Lambert` i `Phong` będą czarne. Możliwe jest ręczne przypisanie wektorów normalnych, ale możesz także obliczyć je za pomocą metody three.js, wywołując metody w klasie geometrii. W przypadku piramidy byłoby to wykonywane przez wywołanie

```
pyramidGeom.computeFaceNormals();
```

Ta metoda oblicza jeden normalny wektor dla każdej ściany, gdzie wektor normalny jest prostopadły do ściany. Jest to wystarczające, jeśli materiał wyko-

rzystuje cieniowanie płaskie; to znaczy, jeśli właściwość `flatShading` materiału jest ustawiona na true.

Cieniowanie płaskie jest odpowiednie dla piramidy. Ale gdy obiekt ma wyglądać bardziej gładko niż płasko, potrzebujemy normalnego wektora dla każdego wierzchołka, a nie dla każdej ściany. `Face3` ma tablicę trzech normalnych wierzchołków. Można je ustawić ręcznie lub Three.js może obliczyć rozsądne normalne wektory wierzchołków dla gładkiej powierzchni poprzez uśrednienie normalnych wektorów ścian, które mają wspólny wierzchołek. Wywołujemy

```
geom.computeVertexNormals();
```

gdzie `geom` jest obiektem geometrii. Zwróć uwagę, że normalne wektory ścian muszą już istnieć przed wywołaniem `computeVertexNormals`, więc zwykle wywołujesz `geom.computeVertexNormals()` natychmiast po wywołaniu `geom.computeFaceNormals()`. Geometria, która ma normalne wektory ścian, ale nie ma normalnych wektorów wierzchołków, nie będzie działać z materiałem, którego właściwość `flatShading` ma wartość domyślną false. Aby umożliwić użycie gładkiego cieniowania na powierzchni takiej jak piramida, wszystkie normalne wektory wierzchołków każdej powierzchni powinny być ustawione na normalny wektor powierzchni. W takim przypadku, nawet przy gładkim cieniowaniu, strona piramidy będzie wyglądać płasko. Standardowe geometrie three.js, takie jak `BoxGeometry`, mają prawidłowe normalne wektory powierzchni i wierzchołków.

Wektor normalny ściany dla obiektu, `face`, typu `THREE.Face3` jest przechowywany w właściwości `face.normal`. Normalne wektory wierzchołków są przechowywane w `face.vertexNormals`, która jest tablicą trzech `Vector3`.

Dzięki pełnemu zestawowi normalnych wektorów piramida jest gotowa do użycia z dowolnymi materiałami siatkowymi, które ma na sobie, ale wygląda trochę nudno z jednym kolorem. Możliwe jest użycie kilku kolorów na jednej siatce. Aby to zrobić, możesz dostarczyć tablicę materiałów do konstruktora obiektów siatki, zamiast pojedynczego materiału. Umożliwia to stosowanie różnych materiałów do różnych ścian. Na przykład, aby zrobić sześcian z różnymi materiałami na sześciu ścianach:

```
var cubeGeom = new THREE.BoxGeometry(10,10,10);
var cubeMaterials =  [
    new THREE.MeshPhongMaterial( { color: "red" } ),     // for the +x face
    new THREE.MeshPhongMaterial( { color: "cyan" } ),    // for the -x face
    new THREE.MeshPhongMaterial( { color: "green" } ),   // for the +y face
    new THREE.MeshPhongMaterial( { color: "magenta" } ), // for the -y face
    new THREE.MeshPhongMaterial( { color: "blue" } ),    // for the +z face
    new THREE.MeshPhongMaterial( { color: "yellow" } )   // for the -z face
];
var cube = new THREE.Mesh( cubeGeom, cubeMaterials );
```

Istnieje kilka sposobów przypisywania koloru do ściany w siatce. Jednym z nich jest po prostu aby każda ściana miała inny jednolity kolor. Każdy obiekt ściany ma właściwość `color`, której można użyć do realizacji tego pomysłu.

Wartość właściwości `color` jest obiektem typu `THREE.Color`, reprezentującym kolor dla całej ściany. Na przykład możemy ustawić kolory ścian piramidy za pomocą

```
pyramidGeom.faces[0].color = new THREE.Color(0xCCCCCC);
pyramidGeom.faces[1].color = new THREE.Color(0xCCCCCC);
pyramidGeom.faces[2].color = new THREE.Color("green");
pyramidGeom.faces[3].color = new THREE.Color("blue");
pyramidGeom.faces[4].color = new THREE.Color("yellow");
pyramidGeom.faces[5].color = new THREE.Color("red");
```

Aby użyć tych kolorów, właściwość `vertexColors` materiału musi być ustawiona na wartość `THREE.FaceColors`; na przykład:

```
material = new THREE.MeshLambertMaterial({
        vertexColors: THREE.FaceColors,
        shading: THREE.FlatShading
    });
```

Wartością domyślną właściwości jest THREE.NoColors, która mówi rendererowi, aby używał właściwości koloru materiału dla każdej twarzy.

Drugim sposobem na zastosowanie koloru do ścian jest zastosowanie innego koloru do każdego wierzchołka ściany. WebGL następnie interpoluje kolory wierzchołków, aby obliczyć kolory pikseli wewnątrz ściany. Każdy obiekt ściany ma właściwość o nazwie `vertexColors`, której wartość powinna być tablicą trzech obiektów `THREE.Color`, po jednym dla każdego wierzchołka ściany. Aby użyć tych kolorów, właściwość `vertexColors` materiału musi być ustawiona na `THREE.VertexColors`.

3 Krzywe i powierzchnie

Oprócz umożliwienia tworzenia indeksowanych zestawów ścian, three.js obsługuje także krzywe i powierzchnie zdefiniowane matematycznie.

Powierzchnie parametryczne są najłatwiejsze do pracy. Powierzchnia parametryczna jest zdefiniowana przez funkcję matematyczną $f(u, v)$, gdzie u i v są liczbami, a każda wartość funkcji jest punktem w przestrzeni. Powierzchnia składa się ze wszystkich punktów, które są wartościami funkcji dla u i v w niektórych określonych zakresach. Dla three.js funkcja jest zwykłą funkcją JavaScript, która zwraca wartości typu `THREE.Vector3`. Parametryczna geometria powierzchni jest tworzona przez ocenę funkcji w siatce punktów u i v. Daje to siatkę punktów na powierzchni, które są następnie łączone, aby uzyskać wielokątne przybliżenie powierzchni. W three.js wartości u i v są zawsze w zakresie od 0,0 do 1,0. Geometria jest tworzona przez konstruktora

```
new THREE.ParametricGeometry( func, slices, stacks )
```

gdzie `func` jest funkcją JavaScript, a `slices` i `stacks` określają liczbę punktów w siatce; `slices` dają liczbę podpodziałów przedziału od 0 do 1 w kierunku u

i `stacks` w kierunku v. Po uzyskaniu geometrii możesz użyć jej do utworzenia siatki w zwykły sposób. Oto przykład z przykładowego programu. Ta powierzchnia jest zdefiniowana przez funkcję

```
function surfaceFunction( u, v ) {
    var x,y,z;  // A point on the surface, calculated from u,v.
                // u  and v range from 0 to 1.
    x = 20 * (u - 0.5);  // x and z range from -10 to 10
    z = 20 * (v - 0.5);
    y = 2*(Math.sin(x/2) * Math.cos(z));
    return new THREE.Vector3( x, y, z );
}
```

a siatka three.js reprezentująca powierzchnię jest tworzona za pomocą

```
var surfaceGeometry = new THREE.ParametricGeometry(surfaceFunction, 64, 64);
var surface = new THREE.Mesh( surfaceGeometry, material );
```

Krzywe są bardziej skomplikowane w three.js (i niestety API do pracy z krzywymi nie jest bardzo spójne). Klasa `THREE.Curve` reprezentuje abstrakcyjną koncepcję krzywej parametrycznej w dwóch lub trzech wymiarach. (Nie reprezentuje geometrii three.js.) Krzywa parametryczna jest zdefiniowana przez funkcję jednej zmiennej numerycznej t. Wartość zwracana przez funkcję jest typu `THREE.Vector2` dla krzywej 2D lub `THREE.Vector3` dla krzywej 3D. Dla obiektu `curve` typu `THREE.Curve` metoda `curve.getPoint(t)` powinna zwrócić punkt na krzywej odpowiadający wartości parametru `t`. Jednak w samej klasie `Curve` ta funkcja jest niezdefiniowana. Aby uzyskać rzeczywistą krzywą, musisz ją zdefiniować. Na przykład,

```
var helix = new THREE.Curve();
helix.getPoint = function(t) {
   var s = (t - 0.5) * 12*Math.PI;
         // As t ranges from 0 to 1, s ranges from -6*PI to 6*PI
   return new THREE.Vector3(
        5*Math.cos(s),
        s,
        5*Math.sin(s)
   );
}
```

Po zdefiniowaniu `getPoint` masz użyteczną krzywą. Jedną z rzeczy, które można z tym zrobić, jest utworzenie geometrii rury, która definiuje powierzchnię, która jest rurą o okrągłym przekroju poprzecznym, a krzywa biegnie wzdłuż środka rury. Przykładowy program używa zdefiniowanej powyżej krzywej `helix` do utworzenia dwóch rur: Geometria szerszej rury jest tworzona za pomocą

```
tubeGeometry1 = new THREE.TubeGeometry( helix, 128, 2.5, 32 );
```

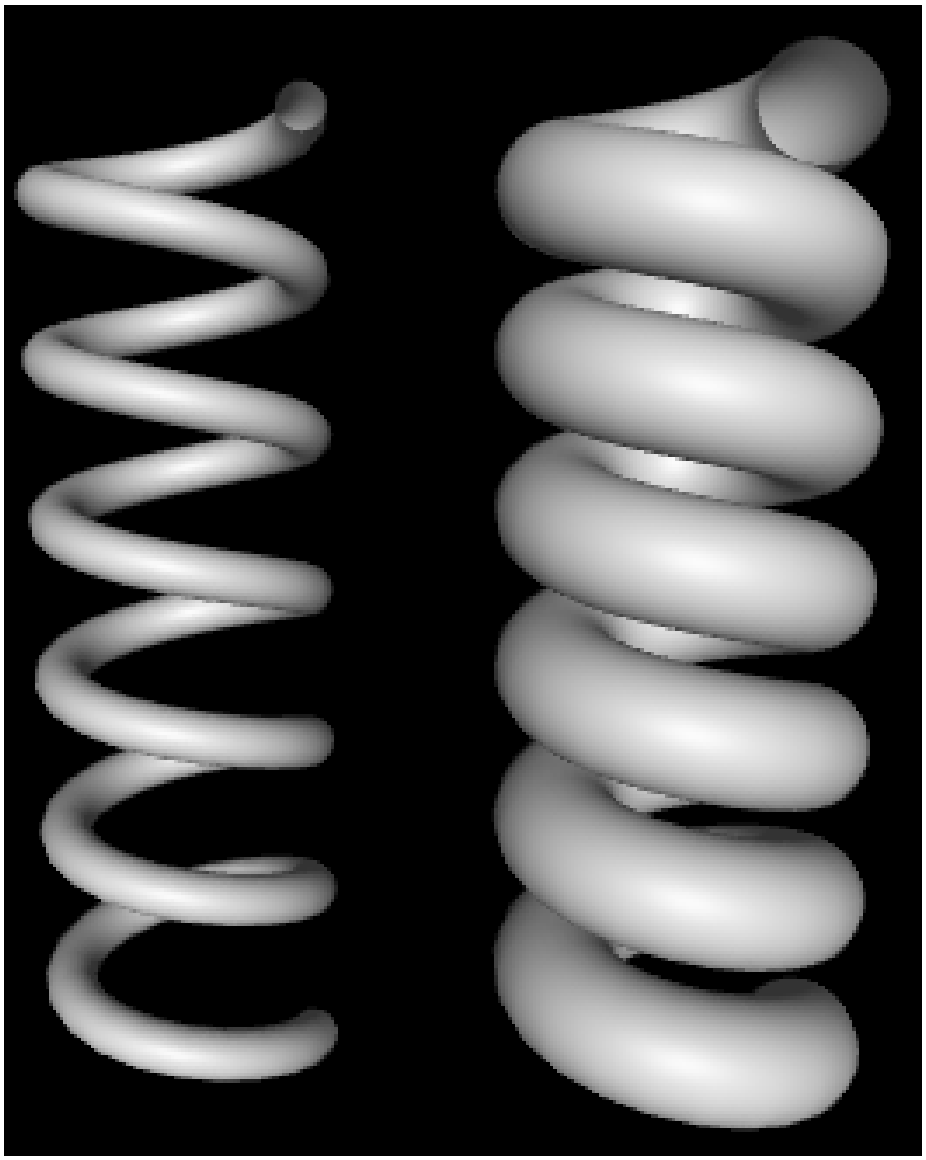


Figure 1: THREE.TubeGeometry

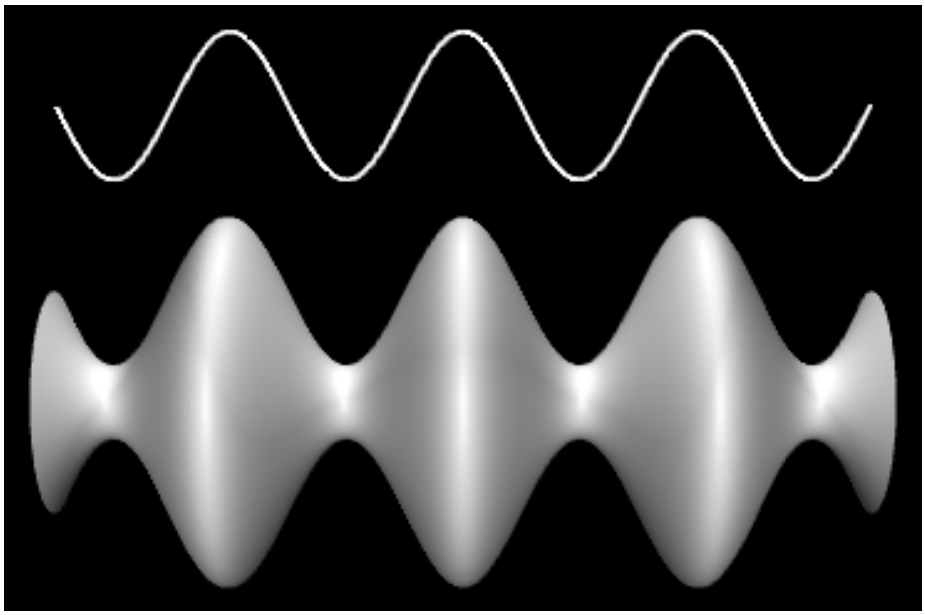


Figure 2: THREE.LatheGeometry

Drugim parametrem konstruktora jest liczba podziałów powierzchni wzdłuż długości krzywej. Trzeci jest promieniem kołowego przekroju poprzecznego rury, a czwarty jest liczbą podziałów na obwodzie przekroju poprzecznego.

Istnieje również kilka sposobów tworzenia powierzchni z krzywej 2D. Jednym ze sposobów jest obrócenie krzywej wokół linii, generowanie powierzchni obrotu. Powierzchnia składa się ze wszystkich punktów, przez które przechodzi krzywa, gdy się obraca. Jest to tak zwane toczenie (**lathing**). Ten obraz z przykładowego programu pokazuje powierzchnię wygenerowaną przez toczenie krzywej kosinusoidalnej. (Obraz jest obrócony o 90 stopni, tak że oś Y jest pozioma.) Krzywa jest pokazana powyżej powierzchni:

Powierzchnia jest tworzona w three.js za pomocą obiektu `THREE.LatheGeometry`. Geometria `LatheGeometry` jest zbudowana nie z krzywej, ale z tablicy punktów leżących na krzywej. Punkty są obiektami typu `Vector2`, a krzywa leży w płaszczyźnie xy. Powierzchnia jest generowana przez obrót krzywej wokół osi y. Konstruktor `LatheGeometry` przyjmuje postać

```
new THREE.LatheGeometry( points, slices )
```

Pierwszym parametrem jest tablica `Vector2`. Druga to liczba podziałów powierzchni wzdłuż okręgu generowanego, gdy punkt jest obracany wokół osi. (Liczba „stosów" powierzchni jest określona przez długość tablicy punktów.)

Kolejną rzeczą, którą możesz zrobić za pomocą krzywej 2D, jest wypełnienie wnętrza krzywej, dając kształt wypełniony 2D. Aby to zrobić w three.js, możesz użyć obiektu typu `THREE.Shape`, który jest podklasą `THREE.Curve`. Kształt

można zdefiniować w taki sam sposób, jak ścieżkę w interfejsie API 2D *Canvas*. Oznacza to, że kształt obiektu typu `THREE.Shape` ma metody `shape.moveTo`, `shape.lineTo`, `shape.quadraticCurveTo` i `shape.bezierCurveTo`, których można użyć do zdefiniowania ścieżki. Jako przykład możemy utworzyć kształt łzy:

```
var path = new THREE.Shape();
path.moveTo(0,10);
path.bezierCurveTo( 0,5, 20,-10, 0,-10 );
path.bezierCurveTo( -20,-10, 0,5, 0,10 );
```

Aby użyć ścieżki do utworzenia wypełnionego kształtu w three.js, potrzebujemy obiektu `ShapeGeometry`:

```
var shapeGeom = new THREE.ShapeGeometry( path );
```

4 Ładowanie modelu JSON

Three.js ma swój własny format pliku, w którym modele są określane za pomocą JSON, wspólnego formatu do reprezentowania obiektów JavaScript. Jest to format pliku tworzony przez skrypty eksportu. Klasa `THREE.JSONLoader` może być używana do odczytywania opisów modeli z takich plików. Istnieje kilka innych ładowarek, które współpracują z innymi formatami plików, ale tutaj omówię tylko `JSONLoader`.

Jeśli `loader` jest obiektem typu `THREE.JSONLoader`, możesz użyć jego metody `load ()`, aby rozpocząć proces ładowania modelu:

```
loader.load( url, callback );
```

Pierwszy parametr to adres URL pliku zawierającego model. Modele JSON są przechowywane jako rzeczywisty kod JavaScript, więc plik zwykle ma nazwę kończącą się na „.js". Drugi parametr, `callback`, to funkcja, która zostanie wywołana po zakończeniu ładowania. Ładowanie jest asynchroniczne; `loader.load ()` uruchamia proces i natychmiast powraca. Obowiązkiem funkcji zwrotnej jest wykorzystanie danych z pliku do utworzenia three.js Object3D i dodanie go do sceny. Funkcja zwrotna przyjmuje dwa parametry, `geometry` i `materials`, które zawierają informacje potrzebne do utworzenia obiektu; parametry reprezentują dane, które zostały odczytane z pliku. Parametr `materials` to materiał lub tablica materiałów, które mogą być użyte jako drugi parametr w konstruktorze obiektów siatki. (Oczywiście możesz również użyć własnego materiału zamiast materiału z pliku.)

Tutaj jest para funkcji, które mogą być użyte do załadowania modelu JSON z określonego adresu URL i dodania go do sceny (chociaż ogólnie, prawdopodobnie chcesz zrobić coś bardziej skomplikowanego z obiektem):

```
function loadModel( url ) {  // Call this function to load the model.
    var loader = new THREE.JSONLoader();
    loader.load( url, modelLoaded ); // Start load, call modelLoaded when done.
```

Figure 3: Warianty zadania

```
}

function modelLoaded( geometry, material ) { // callback function for loader
    var object = new THREE.Mesh( geometry, material );
    scene.add(object);
    render(); // (only need this if there is no animation running)
}
```

Literatura

Uwaga!

Tworzenie aplikacji z wykorzystaniem three.js

`https://docplayer.pl/20001297-Pisanie-aplikacji-z-wykorzystaniem-three-js.html`

W języku angielskim

- książka interakcyjna: Building Objects `http://math.hws.edu/graphicsbook/c5/s2.html` (rozdziały 5.2-5.3)

5 Zadanie

Celem jest konstruowanie modelu figury szachowej zgodnie z wariantem zadania (patrz fig. 3) uywając three.js w oparciu na omówione na zajęcie metody konstruowania obiektów