Informatyka Stosowana

Wydział Fizyki i Informatyki Stosowanej

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie



Inżynierska praca dyplomowa

Wykorzystanie technologii WebGL do utworzenia prostej gry webowej i elementów serwisu internetowego.

Michał Kalita

Dawid Stankiewicz

Opiekun pracy:

**dr inż. Barbara Kawecka-Magiera**

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

........................................

(czytelny podpis)

Spis treści

[1. Wstęp 5](#_Toc439152168)

[1.1 WebGL 5](#_Toc439152169)

[1.2 Cel pracy 5](#_Toc439152170)

[1.3 Użyte oprogramowanie i biblioteki 5](#_Toc439152171)

[2. WebGL i THREE.JS 7](#_Toc439152172)

[2.1 Podrozdział 8](#_Toc439152173)

[3. THREE.JS 9](#_Toc439152174)

[3.1 Importowanie, hierarchia sceny 9](#_Toc439152175)

[4. Opis projektu 11](#_Toc439152176)

[4.1 Podstawowa funkcjonalność 11](#_Toc439152177)

[4.2 Wykorzystane technologie i programy 11](#_Toc439152178)

[4.3 Struktura projektu 11](#_Toc439152179)

[4.4 Blender – przygotowanie modeli 3D 11](#_Toc439152180)

[4.4.1 Przygotowanie modelu samochodu 11](#_Toc439152181)

[4.4.2 Przygotowanie sceny „garaż” 16](#_Toc439152182)

[5. Dalszy rozwój aplikacji 19](#_Toc439152183)

[6. Podsumowanie 20](#_Toc439152184)

# Wstęp

Świat który nas otacza jest trójwymiarowy. Żyjemy , poruszamy się, a nawet myślimy w trzech wymiarach. To przyzwyczajenie do przestrzeni trójwymiarowej sprawiło, że nauczyliśmy się odzwierciedlać ją najpierw na kartce papieru, a następnie w jeszcze większym stopniu na płaskim ekranie coraz szerszej gamy urządzeń.

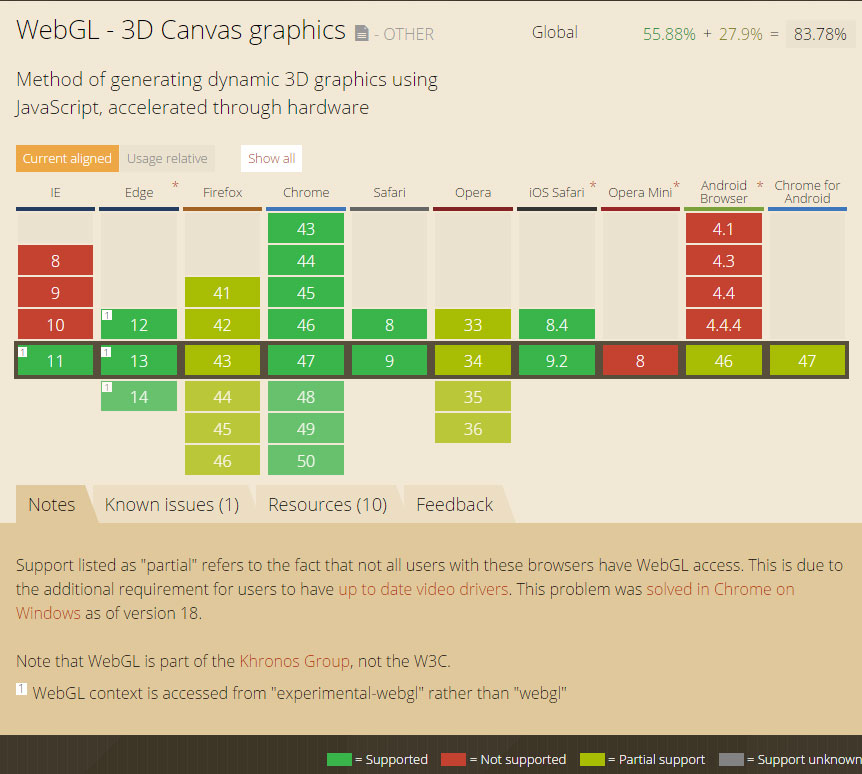
Dla większości ludzi trójwymiarowa przestrzeń w połączeniu z urządzeniami takimi jak komputery, laptopy, konsole, itp. kojarzy się głównie z grami. Jest to skojarzenie jak najbardziej trafne. Gry 3D zachwycają graczy swoim przestrzennym światem już od dość dawna. Technologie 3D towarzyszą jednak nie tylko graczom, spotykamy się z nimi znacznie częściej. Animowane bajki, efekty specjalne w filmach tworzone są z wykorzystanie grafiki trójwymiarowej. W nauce trójwymiarowe wizualizacje pozwalają nam lepiej zrozumieć zagadnienia teoretyczne, lub spojrzeć na rzeczy, które w otaczającym nas świcie są zbyt małe lub zbyt przestronne aby móc na nie spojrzeć z naszej perspektywy. Samochodowe nawigacje dają możliwość lepszego odnalezienia się w terenie właśnie poprzez trójwymiarowe przedstawianie mapy.

Odpoczywany, uczymy się, czy nawet podróżujemy wykorzystując przy tym dobrodziejstwa grafiki trójwymiarowej. Trójwymiarowość wkrada się do coraz większej liczby, dotychczas dwuwymiarowych zagadnień. Osobiście uważamy, że jest to bardzo dobry kierunek rozwoju, dla tego zainteresowała nas możliwość wykorzystania grafiki trójwymiarowej bezpośrednio w przeglądarkach internetowych, z których tak często aktualne korzystamy przeglądając Internet, który staję się coraz bardziej nieodłączną częścią życia każdego z nas.

## Cel pracy

# WebGL i THREE.JS

Biblioteka WebGL nie należy do oficjalnej specyfikacji HTML5, jest jednak obsługiwana przez większość przeglądarek stosujących tą technologię. Poniżej przedstawiono graf ilustrujący wsparcie dla WebGL w najpopularniejszych przeglądarkach desktopowych oraz mobilnych.



WebGL to niskopoziomowe API graficzne. Przekazuje się do niego tablice danych i shader, aby coś narysować. Tym samym przykładowa prosta scena zawierająca rotujący sześcian zmusza programistę do wyprodukowania ponad 200 linii kodu.

Z pomocą przychodzi kilka otwartych zestawów narzędzi. Jedną z nich jest darmowa biblioteka THREE.JS przy wykorzystaniu której, przytoczony wcześniej przykład, można zaimplementować w zaledwie kilkunastu linijkach kodu. Łatwość użycia i aktualna popularność tej biblioteki przyczyniły się w głównej mierze do napisania aplikacji właśnie przy jej użyciu.

# THREE.JS

## Wprowadzenie

THREE.JS to oparta na JavaScript bibliotek 3D. Jej celem jest uproszczenie pracy z WebGL poprzez jej bardzo niską złożoność. [1]

Do rozpoczęcia pracy z THREE.JS potrzebujemy samej biblioteki oraz strony HTML na której będzie można wyświetlić efekty pracy. Poniżej przedstawiono kod przykładowej strony HTML z dodaną biblioteką THREE.JS.

<!DOCTYPE html>

<html>

<head>

<script src="three.min.js"></script>

</head>

<body>

<script>

*// Our JavaScript will go here*

</script>

</body>

</html>

## Pierwsza scena [2]

Stworzenie prostej sceny w THREE.JS wymaga zaledwie kilku linijek kodu. Do zbudowania sceny używamy trzech rodzajów obiektów: kamer, świateł i siatek.

W celu wyrenderowania sceny najpierw należy utworzyć obiekt renderujący WebGL. Poniższy kod tworzy nowy obiekt WebGLRenderer, ustawia rozmiar płótna wyjściowego HTML tego obiektu do 800 x 640 pikseli, dodaje obiekt do ciała dokumentu i inicjuje scenę THREE.JS.



Następnie zdefiniować kamerę. Posłuży ona do wyrenderowania sceny przez obiekt renderujący.



W definicji zmiennej camera pierwszy parametr określa szerokość pola widzenia. Drugi definiuję proporcję. Parametry trzeci i czwarty określają określają punkty odcięcia dla obiektów w widoku kamery. Jeśli odległość pomiędzy obiektem a kamerą nie mieści się w zakresie ustalonym przez parametry NEAR i FAR, to obiekt nie zostanie wyświetlony na scenie. W następnej linii ustawiono pozycję kamery odpowiadającą współrzędnym (X, Y, Z) = (-15, 10, 15). Ostatnia linia ustawia skierowanie kamery w kierunku obiektu sceny.

W celu utworzenia czerwonego sześcianu o boku długości 5 konieczne jest utworzenie odpowiedniej geometrii i materiału a następnie połączenie ich w obiekcie siatki.



Utworzenie obiektu światła i dodanie go do sceny jest ostatnim elementem tworzenia sceny.



Aby dostrzec zacienione krawędzie kostki na tle czarnego tła zmieniono kolor otoczenia.



Pozostało już tylko wyrenderować sceną, aby wyświetlić ją uchwyconą okiem kamery.



Tak wygląda powyższy przykład zebrany w całość i umieszczony w prostym szablonie HTML.

<!DOCTYPE html>

<html>

<head>

<script src="three.min.js"></script>

</head>

<body>

<script>

window.onload = function() {

var renderer = new THREE.WebGLRenderer();

renderer.setSize( 800, 600 );

document.body.appendChild( renderer.domElement );

var scene = new THREE.Scene();

var camera = new THREE.PerspectiveCamera(

35, *// Field of view*

800 / 600, *// Aspect ratio*

0.1, *// Near plane*

10000 *// Far plane*

);

camera.position.set( -15, 10, 10 );

camera.lookAt( scene.position );

var geometry = new THREE.BoxGeometry( 5, 5, 5 );

var material = new THREE.MeshLambertMaterial({color:0xFF0000});

var mesh = new THREE.Mesh( geometry, material );

scene.add( mesh );

var light = new THREE.PointLight( 0xFFFF00 );

light.position.set( 10, 0, 10 );

scene.add( light );

renderer.setClearColor( 0xdddddd, 1);

renderer.render( scene, camera );

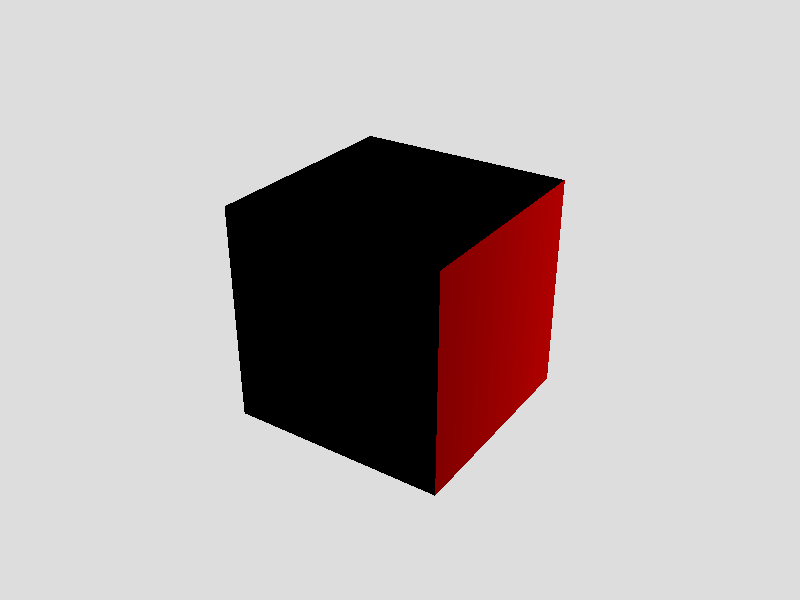
};

</script>

</body>

</html>

Tak prezentuje się powyższy kod uruchomiony w przeglądarce.



## Importowanie, hierarchia sceny

Importowanie i hierarchię sceny

Importowanie przygotowanych w Blenderze modeli zrealizowano za pomocą biblioteki ColladaLoader.js będącej częścią THREE.JS.

W celu załadowania modelu tworzymy obiekt klasy ColladaLoader

var loader = new THREE.ColladaLoader();

Powyższa linia kodu występuje w 3 miejscach projektu:

* Wewnątrz klasy CarBasic, gdzie obiekt loader odpowiedzialny jest za załadowanie modelu samochodu
* W pliku main.js we fragmencie kodu odpowiedzialnego za wczytanie modelu garażu
* W pliku klasy world, w miejscu ładowania mapy

W każdym z tych miejsc wykonywane są następujące czynności:

1. Ustawienie wartości convertUpAxis na true



W Blenderze osią określającą „wysokość” jest oś Z, w WebGL jest to natomiast oś Y. Ustawienie tej wartości na true sprawi, że wszystkie koniecznie przekształcenia ,wynikające z twego faktu, zostaną wykonane automatycznie.

1. Wywołanie funkcji load(), której parametrami są: ścieżka dostępu do pliku modelu, funkcja „onLoad” wywoływana po załadowaniu obiektu, której parametr stanowił będzie załadowany obiekt, funkcja „onProgress” wywoływana podczas ładowania, której argument stanowił będzie obiekt zawierający parametry .loaded i .total.

Wewnątrz funkcji onLoad:

1. Przypisanie zarówno całej sceny jak i poszczególnych jej elementów do zmiennych zadeklarowanych w kodzie programu. Do przeszukania zaimportowanej sceny w głąb je struktury użyto specjalnej funkcji traverse(). Poszczególne elementy sceny rozróżniono na podstawie parametru .colladaId zawierającego nazwę elementu ustawioną w Blenderze.
2. Zmiana lub ustawienia właściwości elementów, takich jak np. materiał, skala, cienie.
3. Wywołanie funkcji updateMatrix() na wczytanym modelu.
4. Dodanie modelu do sceny.

Obiekt do którego zostaje przypisany zaimportowany model jest typu THREE.Group. Poniżej przedstawiono uproszczony schemat hierarchii obiektu sceny z rozwiniętym drzewem obiektu odnoszącego się do zaimportowanego modelu samochodu. Cała scena odnosi się do części aplikacji dającej możliwość modyfikacji parametrów samochodu.



## Materiały

Materiały to bez wątpienia jeden z najprzydatniejszych elementów THREE.JS. Zaoszczędzają one mnóstwo pracy poświęconej na pisanie shaderów koniecznych do stworzenia w przypadku użycie czystego WebGL.

W projekcie wykorzystano 3 podstawowe materiały:

* Basic
* Lamber
* Phong

Basic to materiał przedstawiający obiekt w sposób płaski, pozbawiony cieni.

# Opis projektu

## Podstawowa funkcjonalność

Aplikacja została podzielona na dwie zasadnicze części: grę właściwą będącą prostym wyścigiem samochodowym, oraz aplikację pozwalającą na personalizację i modyfikacje samochodu przed przystąpieniem do rozgrywki. Całość umieszczona została wewnątrz specjalnie przygotowanej strony, napisanej w języku HTML5, w celu lepszej prezentacji przygotowanej aplikacji, jak również zaprezentowaniu możliwości obustronnej interakcji aplikacji z elementami strony.

## Wykorzystane technologie i programy

## Struktura projektu

## Blender – przygotowanie modeli 3D

Do pracy nad wszystkimi modelami użytymi w projekcie użyto programu Blender v2.76.

### Przygotowanie modelu samochodu

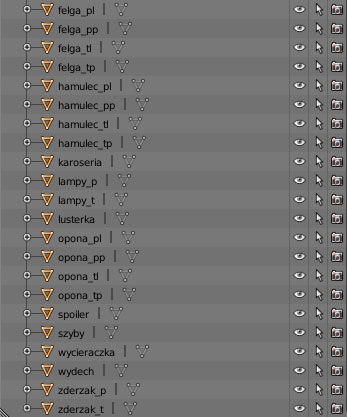
Wykorzystany w niniejszej aplikacji model samochodu pochodzi ze strony <http://www.turbosquid.com/> i jest bardzo prostym i jednocześnie darmowym modelem co idealnie wpasowało się w założenia projektu.

Przed wdrożeniem modelu do gry konieczne były pewnie modyfikacje dokonane w Blenderze w celu maksymalnego uproszczenia pracy z modelem w środowisku THREE.JS.

Model w swoim początkowym stanie składał się z 81 elementów.

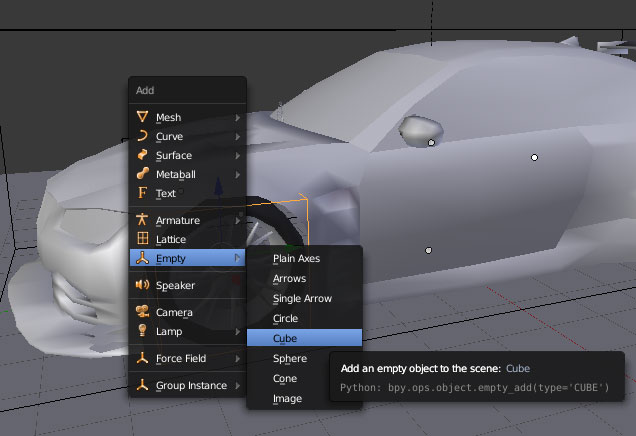


Zarządzanie taką ilością obiektów w aplikacji było by niewygodne, ponadto nie zakładaliśmy indywidualnej kontroli nad tak szczegółowymi elementami modelu. Z tego względu liczba elementów została ograniczona poprzez ich łączenie. Połączone elementy zostały następnie odpowiednio nazwane.

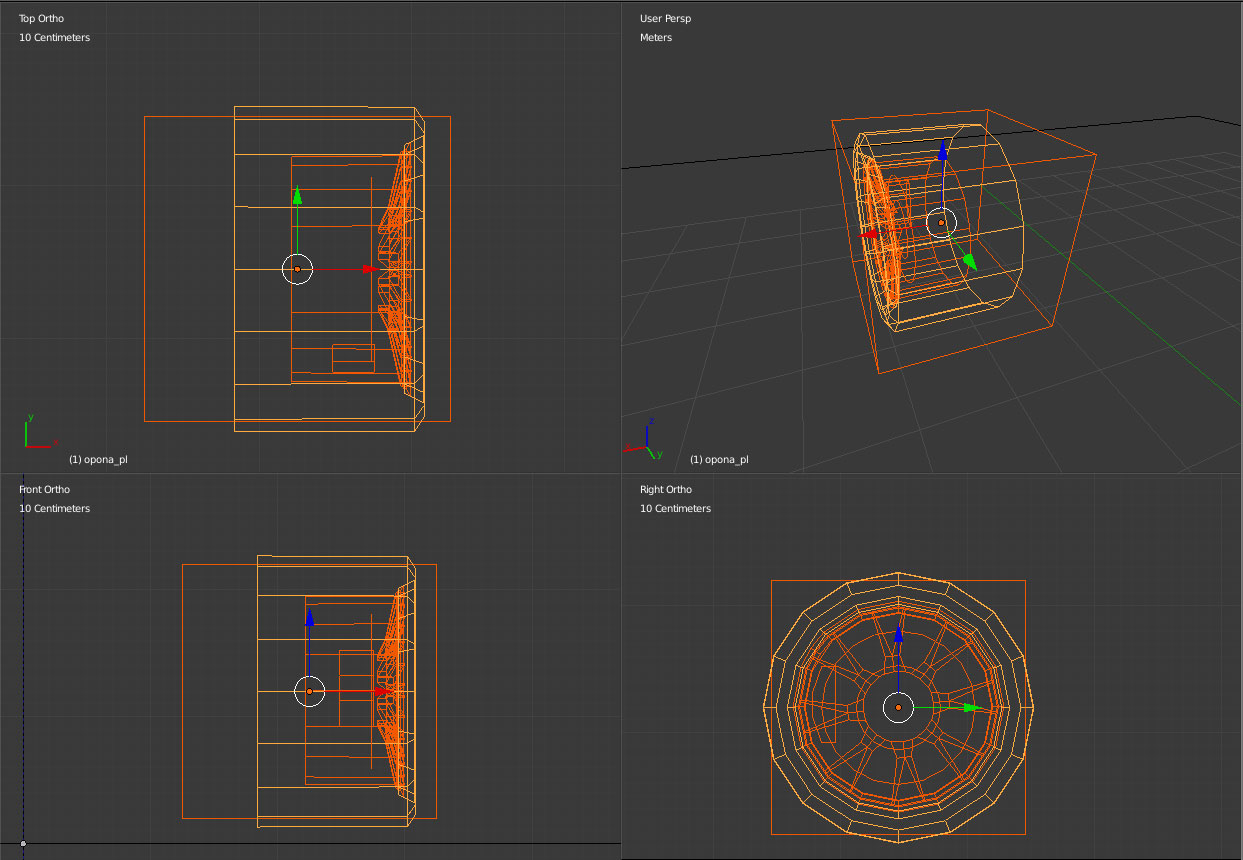


Dokładne nazewnictwo w Blenderze było bardzo ważne, ponieważ na podstawie nazw elementów rozpoznajemy później odpowiadające im obiekty wewnątrz aplikacji. (Pobieranie nazw opisane w punkcie Integracja/Import).

Ostatnią akcją podjętą na wszystkich elementach modelu było ich odpowiednie pogrupowanie. Zabieg ten miał na celu uproszczenie późniejszego sterowania modelem wewnątrz aplikacji. Grupowanie elementów z wykorzystaniem wbudowanych narzędzi Blendera ogranicza się jedynie do programu i nie dawało żadnych efektów po wyeksportowaniu. Problem ten rozwiązano poprzez utworzenie pustego sześciany („Empty Cube”).



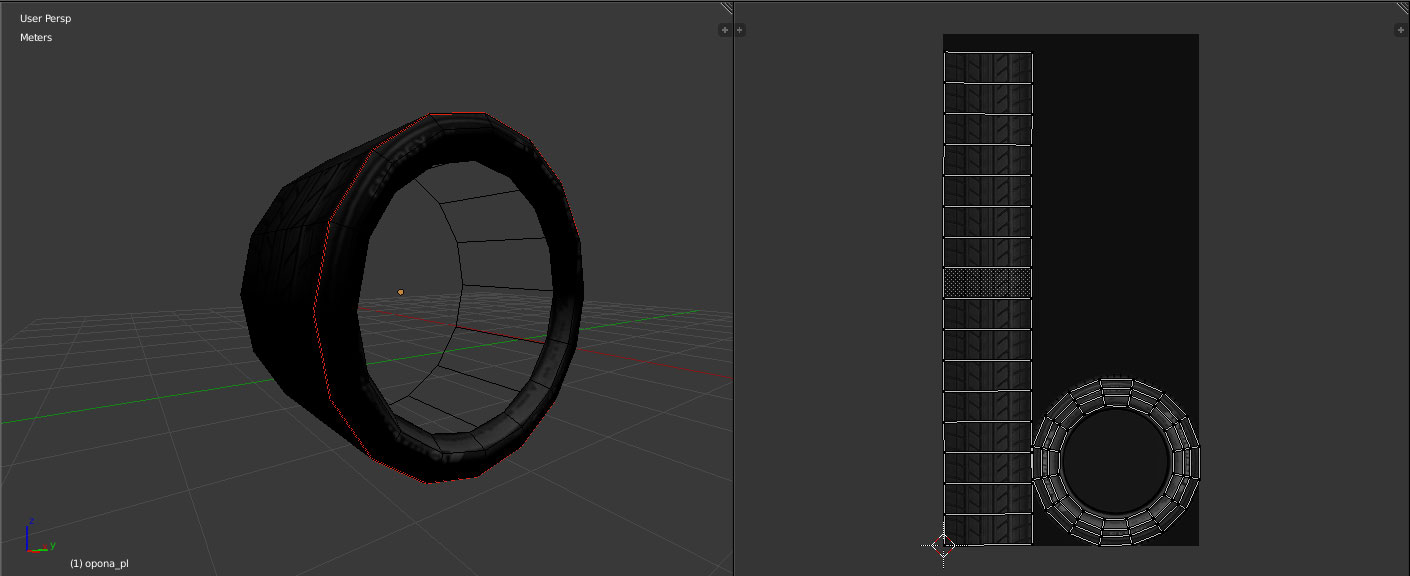
Dla uproszczenia jego rozmiar ustawiono tak aby otaczał elementy mające wchodzić w jego skład. Grupowanie zrealizowano poprzez ustawienie odpowiednich elementów jako potomków nowoutworzonego pustego sześcianu. Bardzo ważne okazało się również ustawienie wspólnej pozycji origin dla rodzica i wszystkich potomków.



Finalnie utworzonym w ten sposób grupom nadano odpowiednie nazwy również wykorzystane do identyfikacji grup wewnątrz aplikacji.

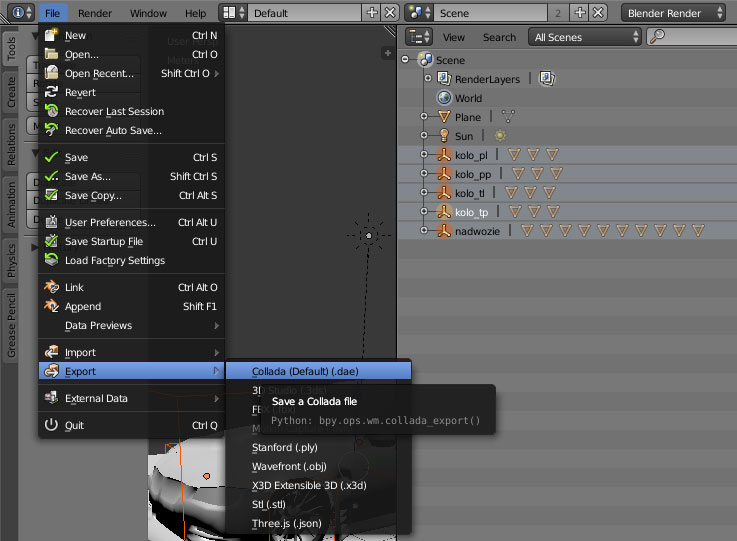


Założenia projektu przewidywały możliwość edycji materiałów, użytych w modelu samochodu, z poziomu aplikacji. Z tego w Blenderze przygotowano jedynie mapę tekstury dla opon.

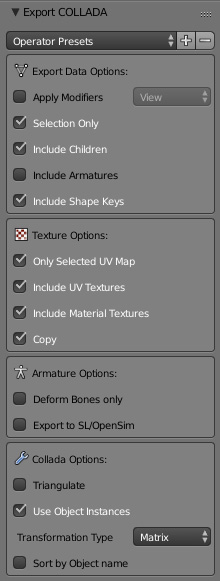


Warto jednak zwrócić uwagę na fakt iż Blender potrafi wyeksportować zarówno elementy sceny jak i użyte materiały. Tym samym edycja materiału jednego obiektu z poziomu THREE.JS poskutkuje zmianą materiału we wszystkich elementach do których materiał ten był połączony wewnątrz projektu Blendera.

Tak przygotowany model zostało jedynie wyeksportować. Biblioteka THREE.JS daje możliwość importowania wielu formatów plików zawierających modele. W projekcie wykorzystano domyślny dla Blendera format „collada Export”.



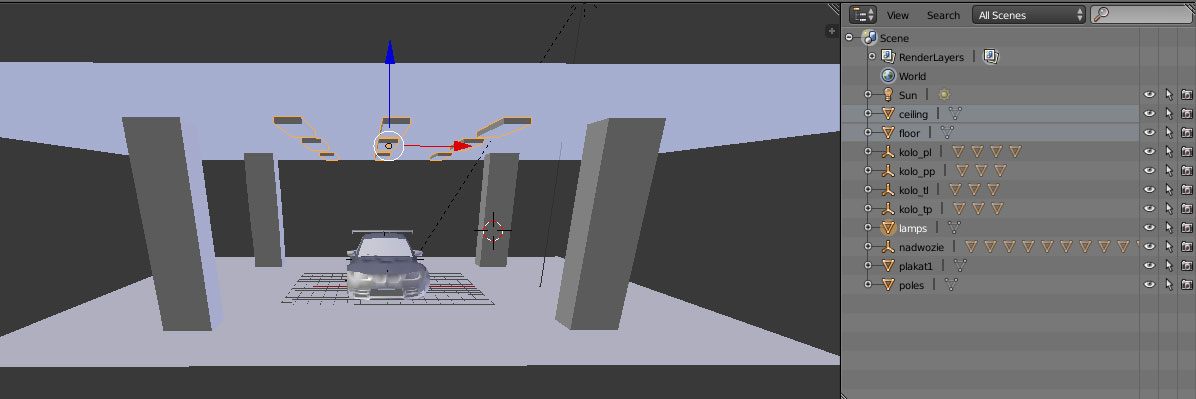
Korzystając przy tym z następujących ustawień eksporty.



### Przygotowanie sceny „garaż”

Scena przedstawiająca podziemny garaż została zaprojektowana od podstaw w Blenderze bez wykorzystania gotowych modeli. Ze względu na założoną wcześniej statyczność sceny, (brak poruszających się elementów), do stworzenia rozkładu świateł i cieni na zawartych elementach wykorzystano silnik renderujący „Cycles Render” w Blenderze, celem poprawienia wyglądu sceny.

Widok wymodelowanej sceny przedstawiono poniżej.



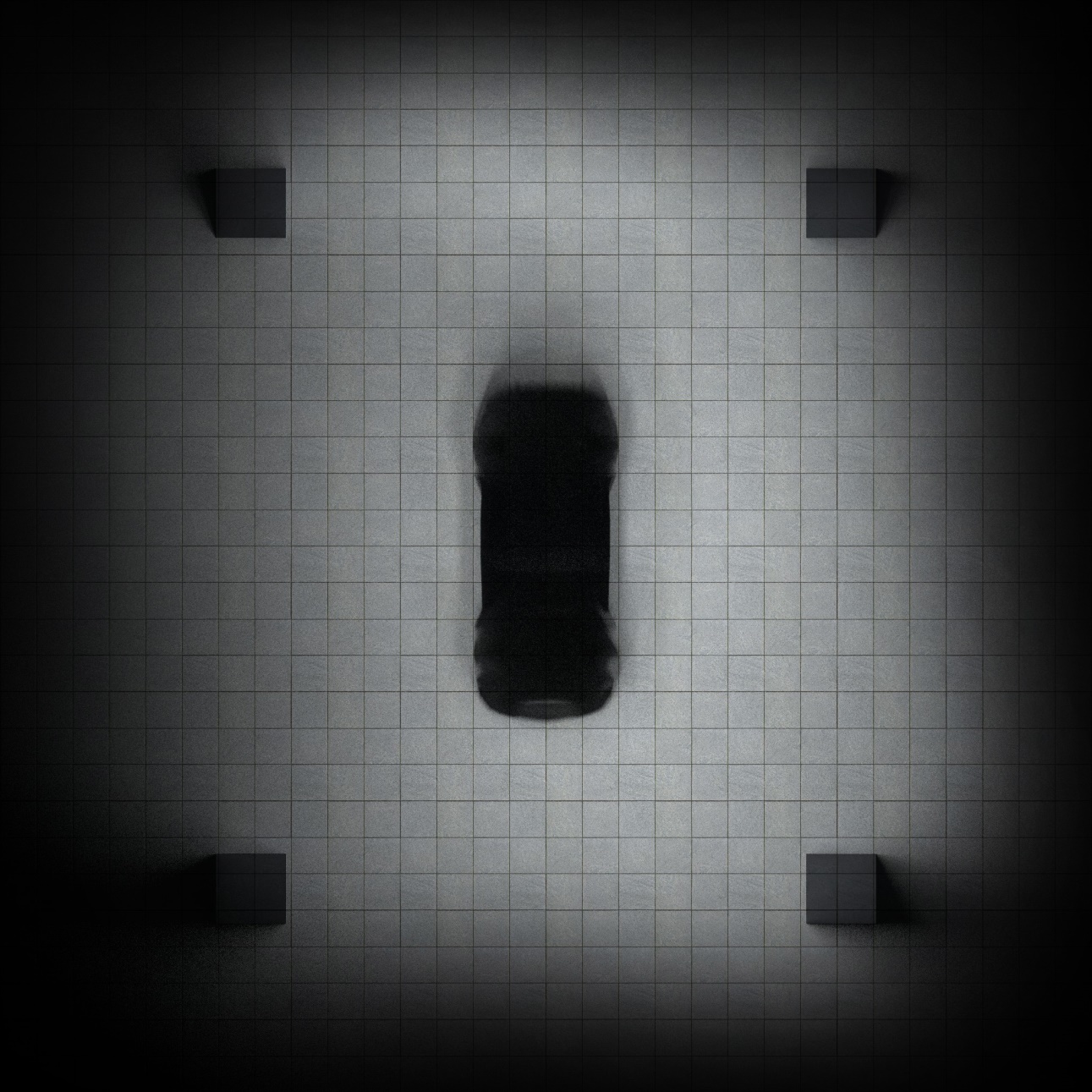
Do wymodelowanej sceny dołączono przygotowany wcześniej model samochodu. Znalazł się on tutaj wyłącznie w celu wygenerowania światłocieni, przy eksportowaniu wykluczono go ze sceny.

Do oświetlenia sceny wykorzystano wykreowane modele lamp umieszczone przy suficie, z wybranym emission shader ustawionym na kolor biały. Dodatkowo scenę doświetla standardowy element oświetlający Blendera – słońce („sun”). Po wygenerowaniu tekstur zredukowano oświetlenie do słońca, oraz po zaimportowaniu sceny, dodano światło otaczające.

Poniżej przedstawiono wyrenderowaną klatkę kompletnej sceny na której użyto darmowych tekstur pobranych ze strony http://www.textures.com/.



Przy renderowaniu tekstur właściwych użyto znacznie wyższego próbkowania niż na powyższej ilustracji, tym samym redukując znaczącą ilość szumów. Efekt osiągnięty przy pomocy opisanej techniki widać najwyraźniej na przykładzie tekstury podłogi, widoczniej poniżej.



# Dalszy rozwój aplikacji

# Podsumowanie

# Bibliografia

1. mrdoob / three.js. [Online] [Zacytowano: 29 Grudzień 2015.] https://github.com/mrdoob/three.js/.

2. Getting Started mrdoob/three.js. *mrdoob/three.js.* [Online] 29 Grudzień 2015. https://github.com/mrdoob/three.js/wiki/Getting-Started.