Wprowadzenie do grafiki komputerowej

Podstawy OpenGL/OpenGL ES/WebGL

Czym jest OpenGL?

- Jest to przenośna biblioteka graficzna 3D typu API (Application Programming Interface) .
- Jest w założeniu niezależna od sprzętu, ale wykorzystuje akceleracje sprzętową.
- Jest biblioteką niskopoziomową („asembler” w oprogramowaniu graficznym).

Najkrótsza charakterystyka OpenGL

- Stanowi podstawę dla bibliotek i pakietów narzędziowych wysokiego poziomu (np. *Open Inventor*, *Open Scene Graph* i innych)
- Stanowi podstawę licznych aplikacji graficznych (pakietów wizualizacyjnych i gier).
- W roku 1995 Microsoft wprowadził bibliotekę Direct3D (razem z całym pakietem DirectX) – głównego konkurenta OpenGL.

OpenGL jako narzędzie do renderowania

- OpenGL jest przede wszystkim narzędziem do renderowania
- Czym jest renderowanie? – najkrócej:
  Tworzenie dwuwymiarowego obrazu rastrowego, uwzględniające różne efekty optyczne występujące w scenie trójwymiarowej

- OpenGL nie ułatwia użytkownikowi modelowania obiektów geometrycznych. Zwykle do modelowania używamy aplikacji lub bibliotek wysokopoziomowych

- Nowe wersje OpenGL jeszcze bardziej kierują się w stronę renderowania i przenoszą ciężar programowania na karty graficzne.

Co może OpenGL?

- W skrócie OpenGL:
  - tworzy elementarne obiekty geometryczne (primitives): punkty, linie, wielokąty
  - pozwala z nich składać obiekty bardziej skomplikowane: siatki wielokątów (zwykle trójkątów), bryły,…
  - pozwala ustawiać je na scenie i poddawać transformacjom geometrycznym
  - renderuje sceny zależnie od stanu (kolorów, materiałów, źródeł światła, etc.)

Najkrótsza historia OpenGL 1/3

- Utworzona z biblioteki IRIS GL (Graphics Library), firmy Silicon Graphics (później SGI), z 1982.
- Silicon Graphics jest legendą grafiki komputerowej, od kilku lat już nie istnieje (szczegóły w wikipedii).

- Od 1992 rozwijana przez Architecture Review Board (ARB) już jako OpenGL – otwarta biblioteka pracująca na różnych platformach (wersję 1.0 ogłoszono w 1992).
  - Kluczowi uczestnicy ARB w różnych okresach: 3Dlabs, Apple, ATI, Dell, IBM, Intel, Nvidia, SGI, Sun Microsystems, Microsoft (do 2003).

Najkrótsza historia OpenGL 2/3

- W kolejnych latach ogłaszane są oficjalne wersje: 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5.

- Ogłoszona w 2004 r. wersja 2.0 włączyła oficjalnie język shaderów (OpenGL Shading Language, GLSL) do realizacji programów na procesorach graficznych.

- Wersja 2.1 kończy serię bibliotek kompatybilnych wstecz. W dalszym ciągu może być i jest używana – choć pewnie coraz rzadziej.

Najkrótsza historia OpenGL 3/3

- Od września 2006 nadzór na rozwojem OpenGL sprawuje Khronos Group.
- W sierpniu 2008 – OpenGL 3.0 + GLSL 1.30, kompatybilny z wcześniejszymi wersjami, ale zapowiadający usuwanie starszych rozwiązań w kolejnych wersjach.
- W marcu 2009 – OpenGL 3.1 + GLSL 1.40.
- W sierpniu 2009 – OpenGL 3.2 + GLSL 1.50.
- W marcu 2010 – OpenGL 3.3 i 4.0
- W lipcu 2010 – OpenGL 4.1
- Kolejne wersje – co rok, sierpień 2014 OpenGL 4.5
- W lipcu 2017 – OpenGL 4.6 (po trzyletniej przerwie)

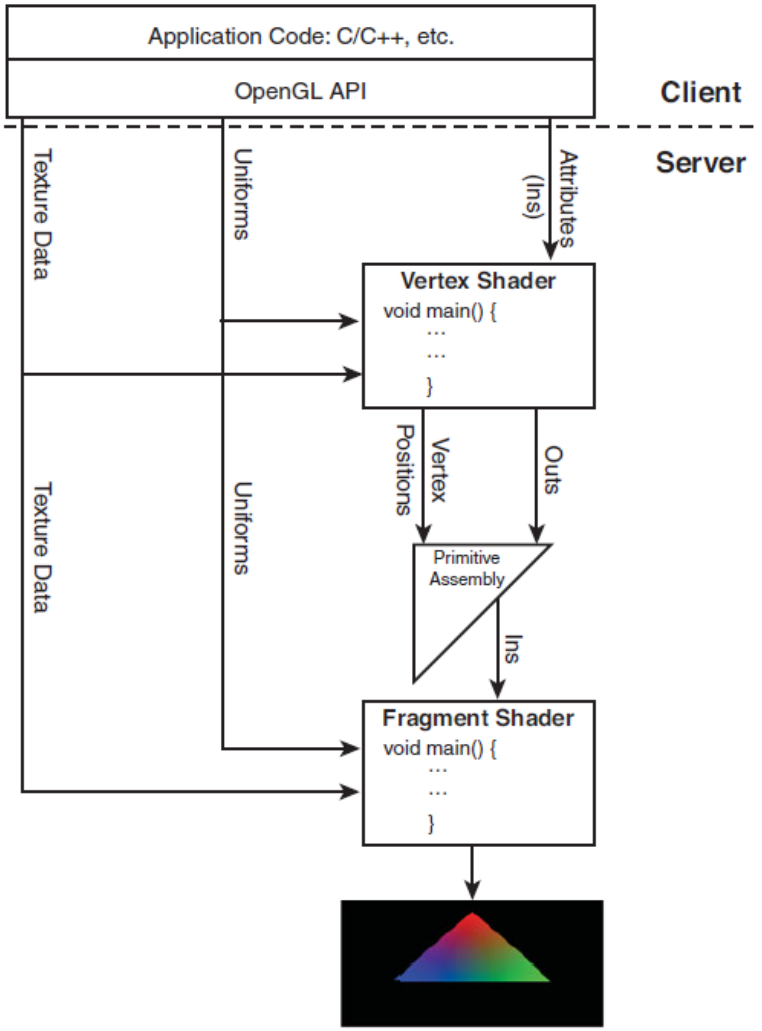
Na czym polega przełom pomiędzy wersją 2.1, a późniejszymi?

- Wersje =< 2.1 pozwalały na ukrycie karty graficznej przed użytkownikiem.
- W wersji 2.1 można jawnie programować kartę graficzną za pomocą języka GLSL (jawnie korzystać z shaderów), można jednak korzystać jedynie z mechanizmów OpenGL.
- W wersji 3.3 istnieje tryb tradycyjny (*Compatibility Mode* – opowiadający 2.1) i tryb nowy (*Core Mode*) pozwalający na efektywne wykorzystanie procesorów graficznych.
- W wersji 3.3 (*Core Mode*) jawne korzystanie z shaderów jest konieczne.

Podsumowanie

- Przygotowując program w OpenGL, w rzeczywistości przygotowujemy DWA programy:
  - pierwszy, opisany przy pomocy funkcji OpenGL realizowany na CPU (przygotowuje scenę, buduje obiekty)
  - drugi opisany przez język shaderów GLSL, realizowany na GPU (dokonuje przekształceń obiektów i renderowania)

Potok renderowania w uproszczeniu; model klient-serwer



Klient – kod pracujący na CPU

Serwer – kod pracujący na GPU

OpenGL ES

- OpenGL ES (embedded systems) jest wersją na smartfony, tablety, konsole gier, i inne urządzenia przenośne.
- Kolejne wersje powstawały równolegle z OpenGL, różniły się jednak numeracją:
  - OpenGL ES 1.0 $\rightarrow$ OpenGL 1.3
  - OpenGL ES 1.1 $\rightarrow$ OpenGL 1.5
  - OpenGL ES 2.0 $\cong$ OpenGL 2.0 (rok 2007)
  - OpenGL ES 3.0 $\rightarrow$ OpenGL 4.3
  - OpenGL ES 3.1 (marzec 2014)
  - OpenGL ES 3.2 (styczeń 2015?)

WebGL 1/2

- WebGL jest implementacją OpenGL ES dla przeglądarek
- WebGL 1.0 odpowiada OpenGL ES 2.0
- W styczniu 2017 zakończono specyfikację WebGL 2.0 odpowiadającego OpenGL ES 3.0
- W bieżącym kursie zasadniczo ograniczamy się do WebGL 1.0
- Różnice dla programisty pojawiają się przy bardziej zaawansowanym wykorzystywaniu shaderów.
- Zestawienie różnic, np. https://webgl2fundamentals.org/webgl/lessons/webgl1-to-webgl2.html

WebGL 2/2

- Użycie WebGL wymaga jakiegokolwiek graficznego wsparcia sprzętowego

- Wykorzystuje Javascript, wywoływany z HTML 5
- Współpracuje z większością przeglądarek
- http://get.webgl.org/ pozwala sprawdzić czy nasza przeglądarka obsługuje WebGL (obecnie na pewno tak)
- http://doesmybrowsersupportwebgl.com/ zawiera bardziej szczegółowe informacje.

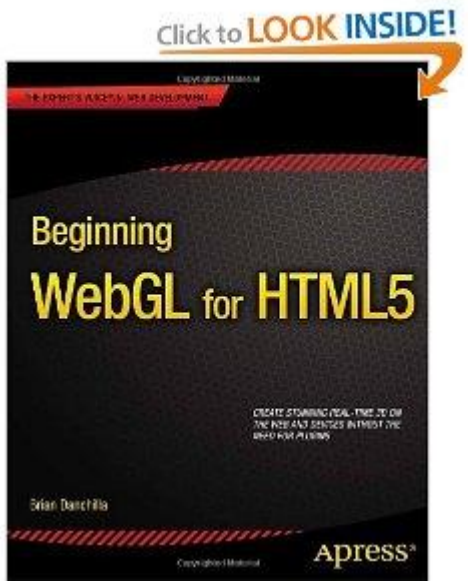
Nowość, o której należy wspomnieć

- Vulkan – nowy standard API, wspólny dla wszystkich platform.
- Specyfikacja wersji 1.0 ogłoszona w lutym 2016
- Na razie nie zyskuje wielkiej popularności, ale przyhamował rozwój OpenGL (wersja 4.5 w 2014 r., wersja 4.6 w 2017… i tyle)
- W marcu 2018 ogłoszono specyfikację Vulkan1.1 – należy się spodziewać wzrostu popularności.

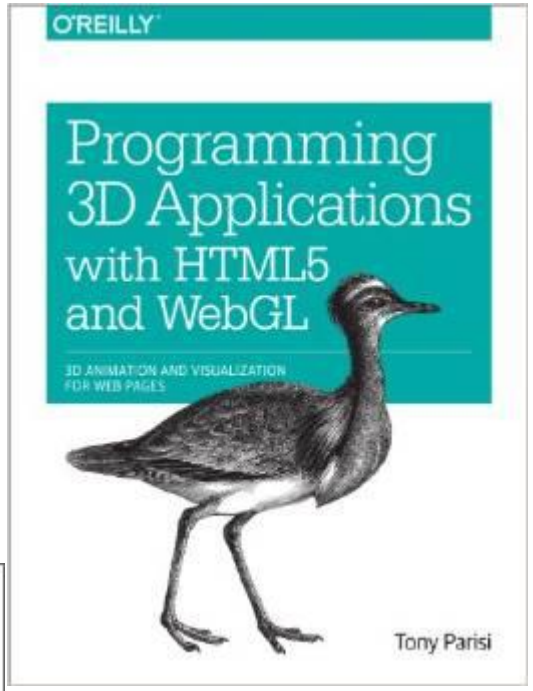
Przydatne podręczniki do WebGL

- Tony Parisi
  WebGL Up and Running, wyd O'Reilly 2012

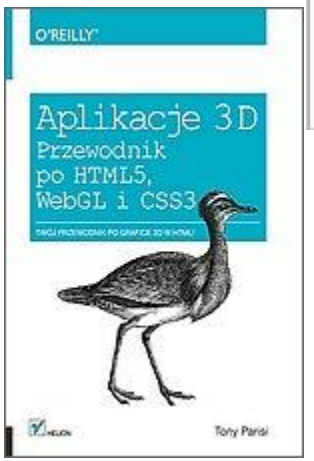
- Brian Danchilla
  Beginning WebGL for HTML 5, Apress 2013



- Tony Parisi Programming 3D Applications with HTML5 and WebGL O'Reilly 2014

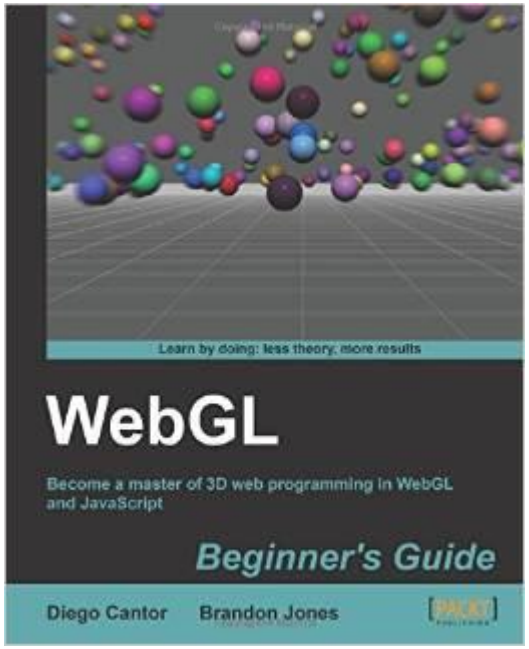


- Jest też polskie wydanie



- Diego Cantor, Brandon Jones
  WebGL – Beginners Guide,
  PACKT Publishing 2012

  Czysty WebGL z shaderami (bez Three.js)



Bardziej szczegółowa bibliografia WebGL i Three.js

- Nie jest oczywiście kompletna, ale mam nadzieję ją uzupełniać.
  Niektóre pozycje są orientowane na programowanie niskopoziomowe w czystym WebGL z wykorzystaniem shaderów pisanych w GLSL, niektóre opierają wyłącznie na wysokopoziomowej bibliotece Three.js (tak, że biblioteki WebGL praktycznie nie widać), a inne starają się łączyć oba podejścia. Dwa podręczniki, przynajmniej w tytule, zapowiadają, że zajmują się budową gier.

Bibliografia c.d.

1. Brian Danchilla, Beginning WebGL for HTML5, Apress, 2012. Czysty WebGL, shadery, Three.js, trochę fizyki.
2. Diego Cantor, Brandon Jones, WebGL Beginner's Guide, Packt, 2012. Czysty WebGL z shaderami.
3. Andreas Anyuru, Professional WebGL Programming, wrox, ??. Czysty webGL, podstawy matematyczne, macierze, etc.
4. Tony Parisi, WebGL Up and Running, O'Reilly, 2012. Czysty WebGL + Three.js. Klasyczna pozycja.
5. Tony Parisi, Programming 3D Applications with HTML5 and WebGL, O'Reilly 2014. Podobna do poprzedniej książki, ale poszerzona. Jest polskie wydanie.
6. Sumeet Arora, WebGL Game Development, Packt, 2014. Czysty WebGL z shaderami + Three.js
7. Jos Dirksen, Learning Three.js: The JavaScript 3D Library for WebGL, Packt, 2013. oparta na przykładach. Jest drugie wydanie z 2015 – zmiany są, ale niezbyt wielkie.
8. Jos Dirksen, Three.js Essentials, Packt, 2014. W stylu poprzedniej książki Dirksena, ale o rok nowsza. Można potraktować jak rozszerzenia, ale niezależnie też.
9. Isaak Sukin, Game Development with Three.js, Packt, 2013. Tytuł mówi wszystko, nieduża książka (118 str), dość elementarna, ale fajna.

Pożyteczne linki (WebGL)

- http://learningwebgl.com/blog/ 15 podstawowych lekcji – przykładów; kurs zakończony w marcu 2015
- https://www.khronos.org/webgl/wiki/Main_Page dokumentacja, przykłady, jedna z podstawowych stron referencyjnych
- https://stemkoski.github.io/Three.js/ sympatyczne przykłady w three.js
- https://threejs.org/ strona domowa biblioteki three.js, dokumentacja, przykłady
- https://www.chromeexperiments.com/
- i wiele, wiele innych….

Dwie pozycje z OpenGL ES 2

- OpenGL ES 2 for Android A Quick - Start Guide (2013)

- Pro OpenGL ES for Android (2012)

Klasyczne pozycje OpenGL i GLSL.

- R.S.Wright, et al.. **OpenGL-Superbible,** 7th edition, Addison-Wesley, 2016 (polskie wydanie **OpenGL – Księga eksperta**, wyd 7, Helion, 2016)
- Randi J. Rost, **OpenGL Shading Language,** 3rd edition, 2009
- M. Bailey i S. Cunnigham, **Graphics shaders**, 2nd edition CRC Press, 2012
- M.Woo, J.Neider, T.Davies, **OpenGL – Programming Guide**, Addison-Wesley, 8th edition, 2013
  (są dostępne wersje elektroniczne starszych wersji pdf)

Wprowadzenie do WebGL

- Prezentacja z roku 2011
  http://www.khronos.org/assets/uploads/developers/library/2011-siggraph-mobile/Khronos-and-the-Mobile-Ecosystem_Aug-11.pdf
  Proszę obejrzeć 21 slajdów z tego linku i potraktować je jako część wykładu.

WebGL vs OpenGL. Ograniczenia w stosunku do ostatnich wersji OpenGL. Podane dla porządku - dla nas niezbyt istotne. Część ograniczeń została usunięta w WebGL 2

- WebGL obejmuje
  - Vertex shaders
  - Fragment shaders
  - Vertex buffers
  - Textures
  - Framebuffers
  - Render states
  - …

- WebGL nie obejmuje
  - Geometry shaders
  - Tessellation shaders
  - Vertex Array Objects
  - Multiple render targets
  - Floating-point textures
  - Compressed textures
  - FS depth writes
  - …

Patrz http://www.khronos.org/registry/webgl/specs/latest/

Alternatywy dla WebGL?

- Alternatywa rozumiana jako możliwość grafiki 3D w przeglądarkach
  - Flash
  - Silverlight
  - Java Applets
  - Unity

Elementy WebGL

- Tworzenie kontekstu:

```
// HTML:
<canvas id "glCanvas" width "1024"
  height "768"  /canvas>

// JavaScript:
var canvas =
  document.getElementById('glCanvas');
var gl=canvas.getContext('webgl');
```

Elementy WebGL

- Podstawowe funkcje WebGL są bardzo zbliżone do OpenGL:

```
// ...
gl.bindBuffer(/* ... */);
gl.vertexAttribPointer(/* ... */);
gl.useProgram(/* ... */);
gl.drawArrays(/* ... */);
```

- Więcej informacji w lekcjach http://learningwebgl.com/

WebGL

- Tworzenie pętli animacji:

```
function tick(){
  // ... GL calls to draw scene
  window.requestAnimationFrame(tick);
})();
```

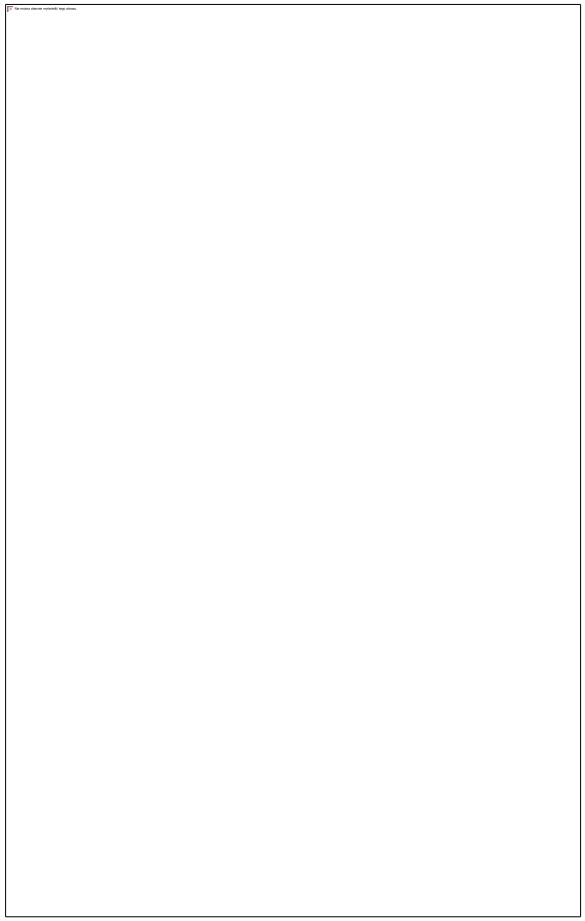
Patrz http://paulirish.com/2011/requestanimationframe-for-smart-animating/

Wydajność WebGL

- Wysoka. Dlaczego?
  - Renderowanie na GPU
  - Rysowanie wielu obiektów jednym wywołaniem
  - Użycie shaderów

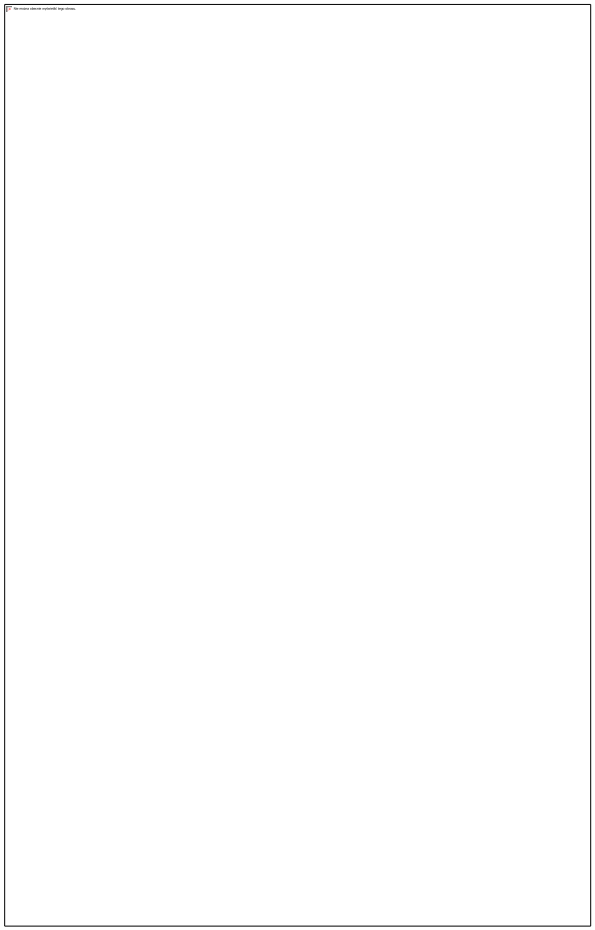
- Solidny wykład z 2011 roku na temat wydajności w WebGL: http://www.youtube.com/watch?v=rfQ8rKGTVlg

Zaawansowane demonstracje



WebGL Skin

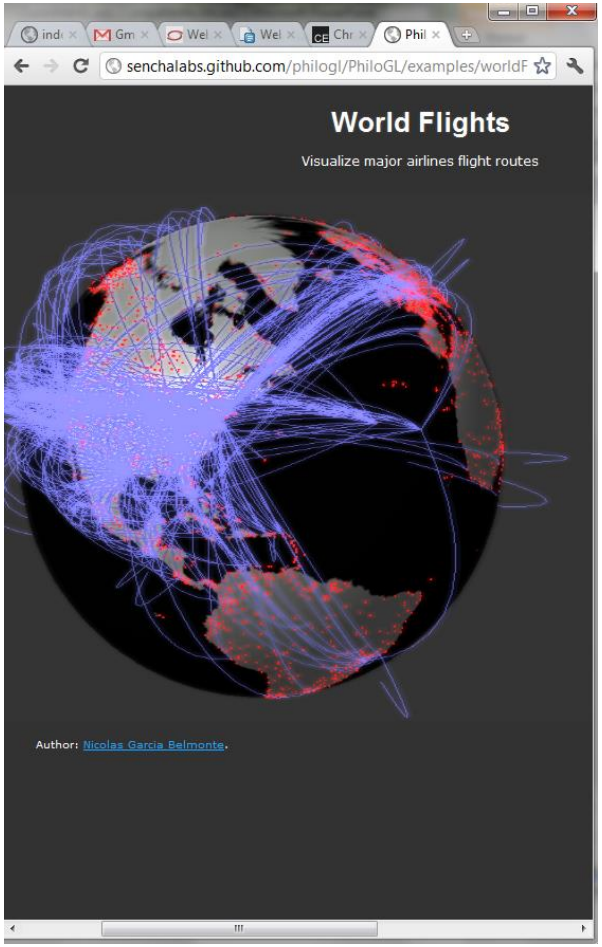
http://alteredqualia.com/three/examples/webgl_materials_skin.html



WebGL Water

http://madebyevan.com/webgl-water/

Demonstracje w WebGL (wybrane arbitralnie)



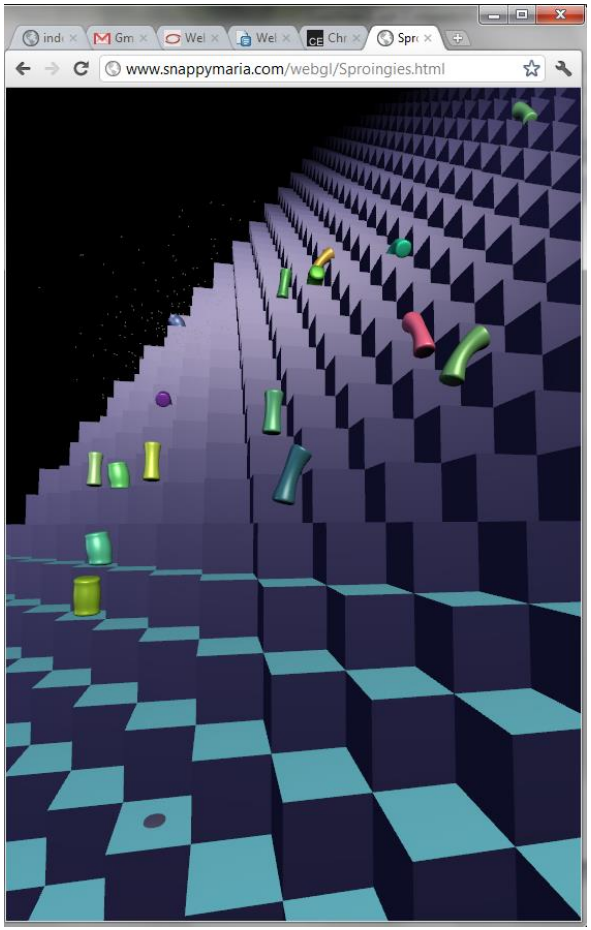
World Flights

http://senchalabs.github.com/philogl/PhiloGL/examples/worldFlights/

WebGL Jellyfish

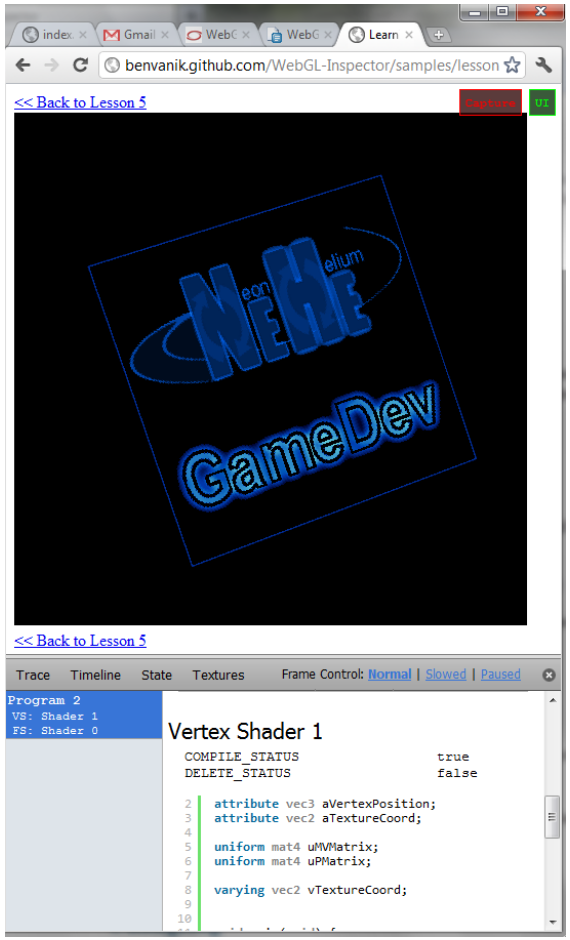
https://arodic.github.io/p/jellyfish/

Kolejne demonstracje



The Sproingies

http://www.snappymaria.com/webgl/Sproingies.html



WebGL Inspector

http://benvanik.github.com/WebGL-Inspector/samples/lesson05/embedded.html

Biblioteki wysokopoziomowe dla WebGL

- Three.js: https://github.com/mrdoob/three.js/
- SceneJS: http://scenejs.org/
- PhiloGL: http://senchalabs.org/philogl/
- SpiderGL: http://spidergl.sourceforge.net/
- Cenne źródło narzędzi wspomagających WebGL: http://www.khronos.org/webgl/wiki/User_Contributions
- List of WebGL Frameworks: https://en.wikipedia.org/wiki/List_of_WebGL_frameworks

- Obecnie Three.js wydaje się najpopularniejsza, ale to się może zmienić w każdej chwili.

WebGL, a Three.js

- WebGL jest biblioteką niskopoziomową uciążliwą w budowaniu scen
- Potrzebne są biblioteki wysokopoziomowe, bardziej intuicyjne, zawierające
  - Obiekty graficzne
  - Grafy sceny
  - „display lists”
- Pouczające jest zacząć od przykładu w czystym WebGL, jednak bardzo wygodnie jest przejść natychmiast do Three.js

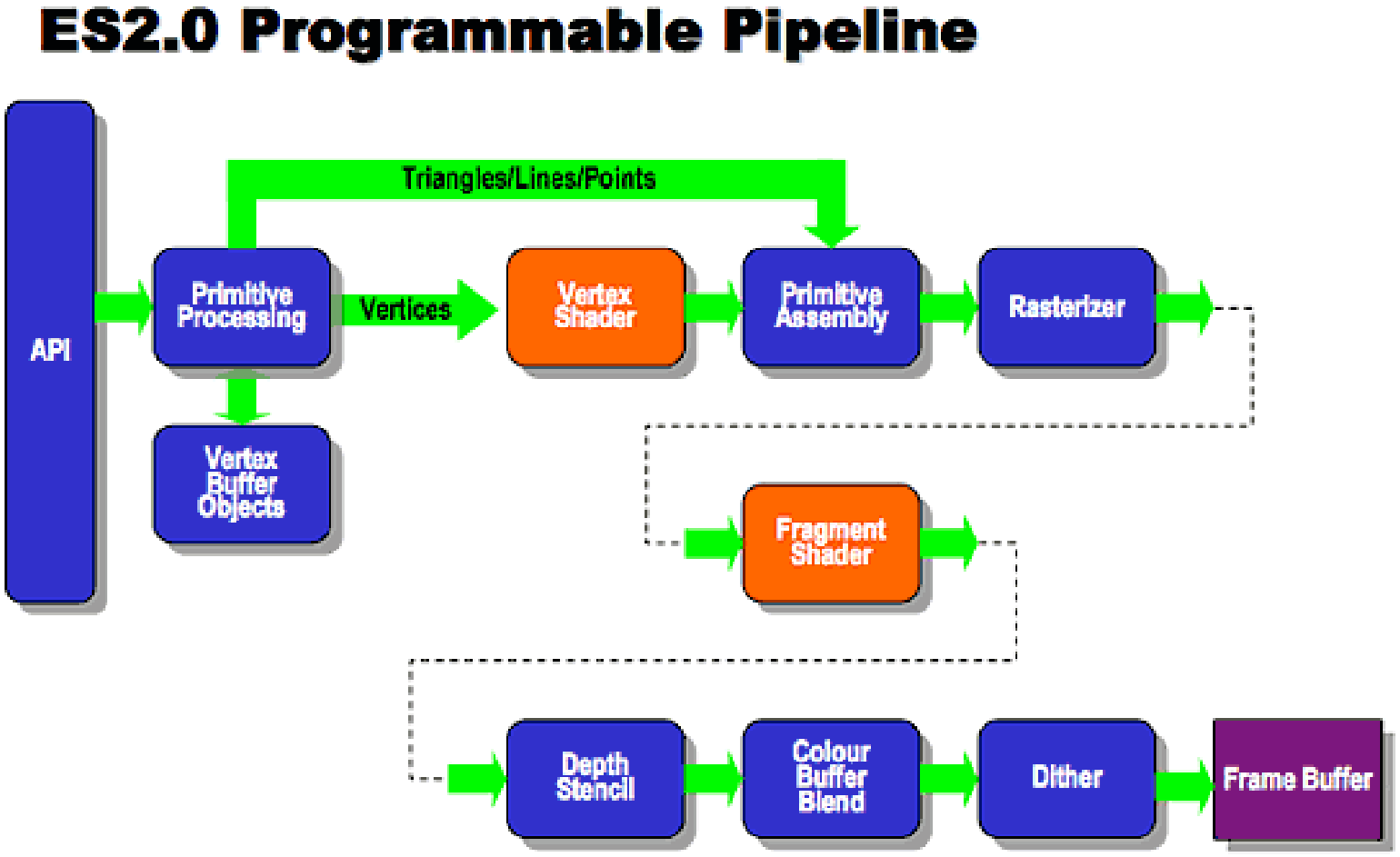
Pojęcia WebGL – głównie dotyczą buforów, przesyłania CPU-GPU, danych dla shaderów

- Buffers
- RenderBuffer
- FrameBuffer
- Textures
- Blending
- Depth buffer
- Stencil buffer
- Uniform variables
- Attribute variables
- (w sumie WebGL liczy ok 130 funkcji)

Typowa struktura kodu w czystym WebGL

- Kroki postępowania:
  - Utwórz element canvas w HTML5
  - Określ kontekst rysowania
  - Zainicjuj viewport
  - Wygeneruj bufory wierzchołków
  - Utwórz macierze transformacji
  - Utwórz shadery
  - Rysuj

Programowany potok graficzny (nie analizujemy go głęboko)



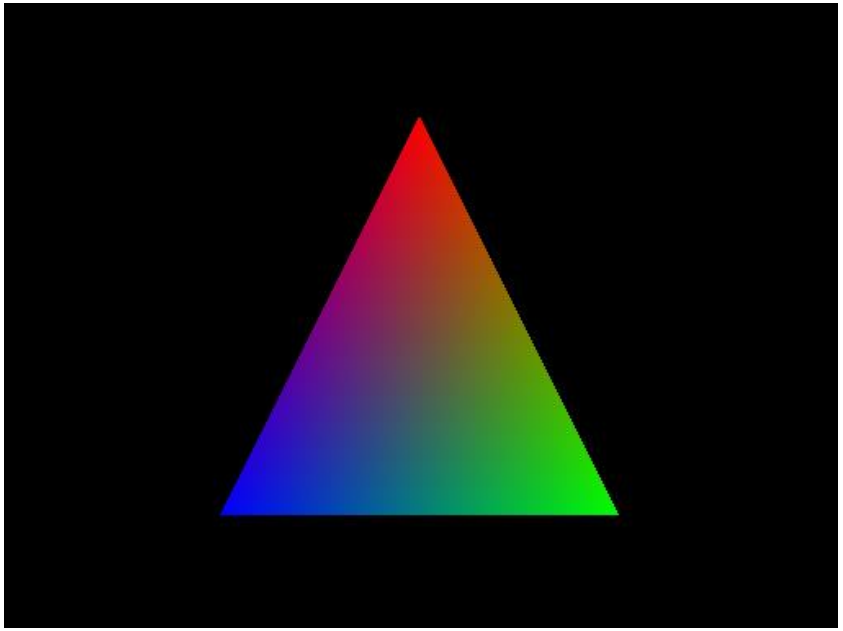
Shadery

- GLSL: GL Shader Language
- Składnia zbliżona do C
- Vertex shaders: obliczenia *per-vertex*
- Fragment shaders: obliczenia *per-pixel*
- architektura typu SIMD

- Przykłady różnych shaderów można znaleźć np. na stronie https://www.shadertoy.com/ w zakładce *browse*
- Mamy oddzielny wykład poświęcony shaderom

Hello WebGL

- Sporo pracy, żeby narysować trójkąt
- Jeśli jednak framework jest gotowy, reszta jest nie tak złożona



- Dobrym źródłem przykładów w czystym WebGL jest http://learningwebgl.com/

Przykład kodu

- Elementarny kod w czystym WebGL i z użyciem biblioteki Three.js

- Źródło: Tony Parisi, WebGL Up and Running

- Przykład 1 (179 wierszy)
- Przykład 2 (42 wiersze)

Three.js silnik 3D dla WebGL

- Stworzony przez Miguela Cabello z Hiszpanii (znanego jako Mr. Doob)
- Wysokopoziomowy framework dla WebGL
- Dokumentacja na stronie threejs.org nie jest kompletna (na razie, ale ciągle się rozbudowuje)
- Na stronie jest blisko 400 przykładów edukacyjnych
- Dodatkowo mamy kilkadziesiąt projektów nadesłanych przez zewnętrznych autorów.

- Bieżąca wersja three.js (w marcu 2018): **r90**

Szkielet programu z Three.js

```
<html>
<head>
<title> Pierwsza aplikacja Three.js </title>
<style> canvas {width: 100%; height: 100%} </style>
</head>
<body>
<script
src = ”three.js” >
</script>
<script>
// Tutaj nasz program w Javascript
</script>
</body>
</html>
```

Podstawowe pojęcia Three.js

- Żeby cokolwiek wyświetlić, potrzebujemy następujących obiektów:
  - scena
  - kamera (obserwator)
  - Renderer

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75,
width, height, 0.1, 1000);
scene.add(camera);

var renderer = new THREE.WebGLRenderer();
renderer.setSize(width, height);
document.body.appendChild(renderer.domElement);
```

Dodawanie obiektów geometrycznych do sceny

- Na przykład dodajemy sześcian

```
var geometry = new THREE.BoxGeometry(1,1,1);
var material = new THREE.MeshBasicMaterial({color:
0x00ff00});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);

camera.position.z = 5;
cube.rotation.x = .5;
cube.scale.x = 3.


camera.positon.set(0,4,5);
```

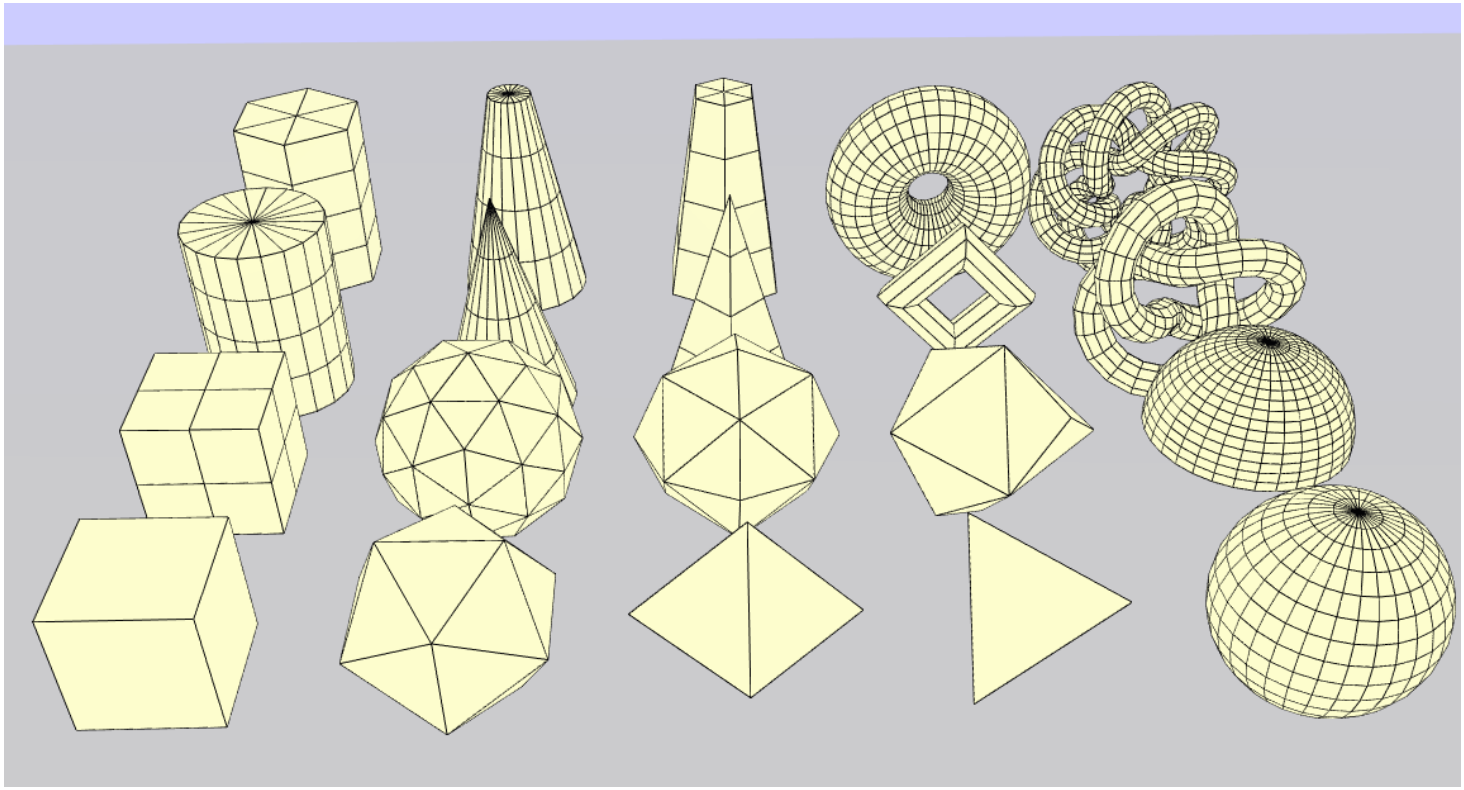
Renderowanie sceny

```
function render() {
renderer.render(scene, camera);
}
render();
```

```
function render() {
requestAnimationFrame(render);

cube.rotation.x += 0.1;
cube rotation.y += 0.1;

renderer.render(scene, camera);
}
render();
```

Geometria

- Jak stworzyć geometrię? Przykłady gotowych obiektów 3D z biblioteki Three.js



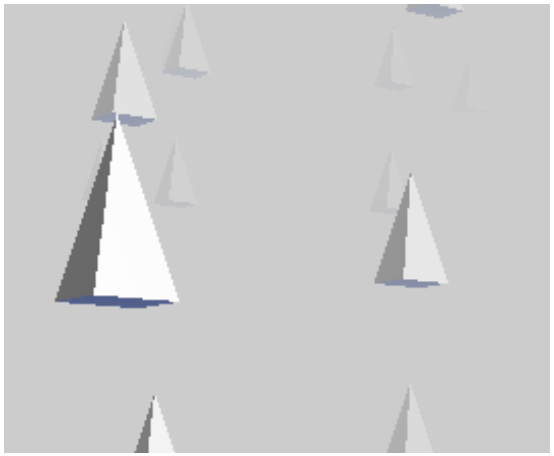
Tworzenie geometrii

- Używamy obiektów takich jak BoxGeometry, CylinderGeometry, PolyhedronGeometry, …
- (Mieszczą się one w grupie Geometries – gdy szukamy w dokumentacji threejs.org)

- Dodajemy je do sceny

Tworzenie geometrii. Generowanie losowo rozmieszczonych piramidek

```
scene = new THREE.Scene();
scene.fog = new THREE.FogExp2(0xcccccc, 0.002);

var geometry = new THREE.CylinderGeometry( 0, 10, 30, 4, 1 );
var material = new THREE.MeshLambertMaterial({ color:0xffffff,
shading:THREE.FlatShading});
for (var i=0; i<500; i++){
    var mesh = new THREE.Mesh(geometry, material);
    mesh.position.x=(Math.random()-0.5)*1000;
    mesh.position.x=(Math.random()-0.5)*1000;
    mesh.position.x=(Math.random()-0.5)*1000;
    mesh.updateMatrix();
    mesh.matrixAutoUpdate=false;
    scene.add(mesh);
}
```



Nawigowanie - Trackball

```
camera = new THREE.PerspectiveCamera( 60, window.innerW
camera.position.z = 500;

controls = new THREE.TrackballControls( camera );

controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;

controls.noZoom = false;
controls.noPan = false;

controls.staticMoving = true;
controls.dynamicDampingFactor = 0.3;

controls.keys = [ 65, 83, 68 ];

controls.addEventListener( 'change', render );
```

Oświetlenie?

- Podstawowe rodzaje oświetlenia:
- Rodzaje źródeł światła w Three.js:
  - AmbientLight,
  - DirectionalLight,
  - PointLight,
  - SpotLight
- Jest wystarczająca dokumentacja

Oświetlenie w Three.js

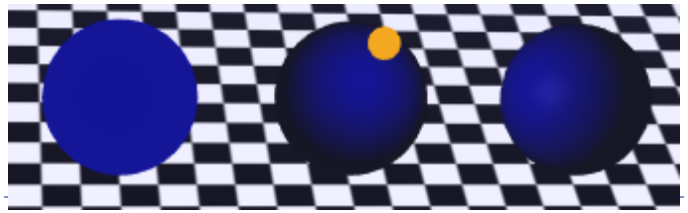
```
var light = new THREE.PointLight(0xff2200);
light.position.set(100, 100, 100);
scene.add( light);

var light2 = new THREE.AmbientLight(0x111111);
scene.add( light2 );


var geometry = new THREE.CubeGeometry( 100, 100, 100 );
var material = new THREE.MeshLambertMaterial({
color:0xff0000,});
```

Materiały

- Rodzaje materiałów:
  - MeshBasicMaterial
  - MeshLambertMaterial
  - MeshPhongMaterial

- Parametry/własności
  - Color, wireframe, shading, vertexColors, fog, lightMap, specularMap, envMap, skinning, morphTargets

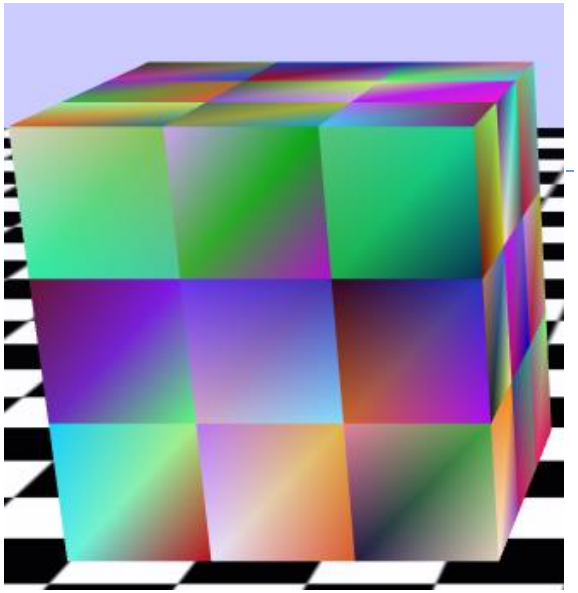
Materiały



```
// Sphere parameters: radius, segments along width, segments along height
var sphereGeom =  new THREE.SphereGeometry( 50, 32, 16 );

// Three types of materials, each reacts differently to light.
var darkMaterial = new THREE.MeshBasicMaterial( { color: 0x000088 } );
var darkMaterialL = new THREE.MeshLambertMaterial( { color: 0x000088 } );
var darkMaterialP = new THREE.MeshPhongMaterial( { color: 0x000088 } );

// Creating three spheres to illustrate the different materials.
// Note the clone() method used to create additional instances
//    of the geometry from above.
var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterial );
sphere.position.set(-150, 50, 0);
scene.add( sphere );

var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterialL );
sphere.position.set(0, 50, 0);
scene.add( sphere );

var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterialP );
sphere.position.set(150, 50, 0);
scene.add( sphere );
```

Kolorowanie gradientowe

- Na podstawie wierzchołków



```
face   = cubeGeometry.faces[ i ];
// determine if current face is a tri or a quad
numberOfSides = ( face instanceof THREE.Face3 ) ? 3 : 4;
// assign color to each vertex of current face
for( var j = 0; j < numberOfSides; j++ )
{
    vertexIndex = face[ faceIndices[ j ] ];
    // initialize color variable
    color = new THREE.Color( 0xffffff );
    color.setHex( Math.random() * 0xffffff );
    face.vertexColors[ j ] = color;
}
```