WebGL (three.js)

Standardisation de la 3D sur le web

Christian Nguyen

Département d'informatique
Université de Toulon

Plan

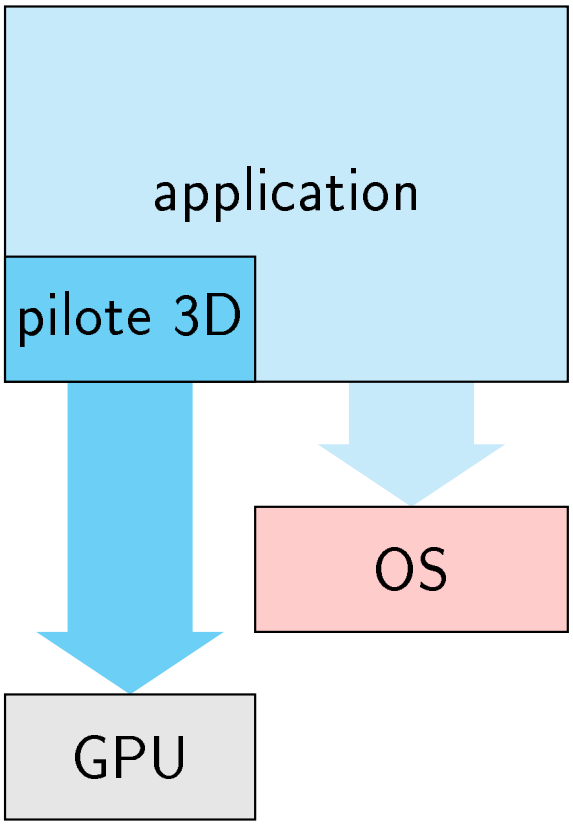
Introduction
L'API WebGL

Une API qui permet :

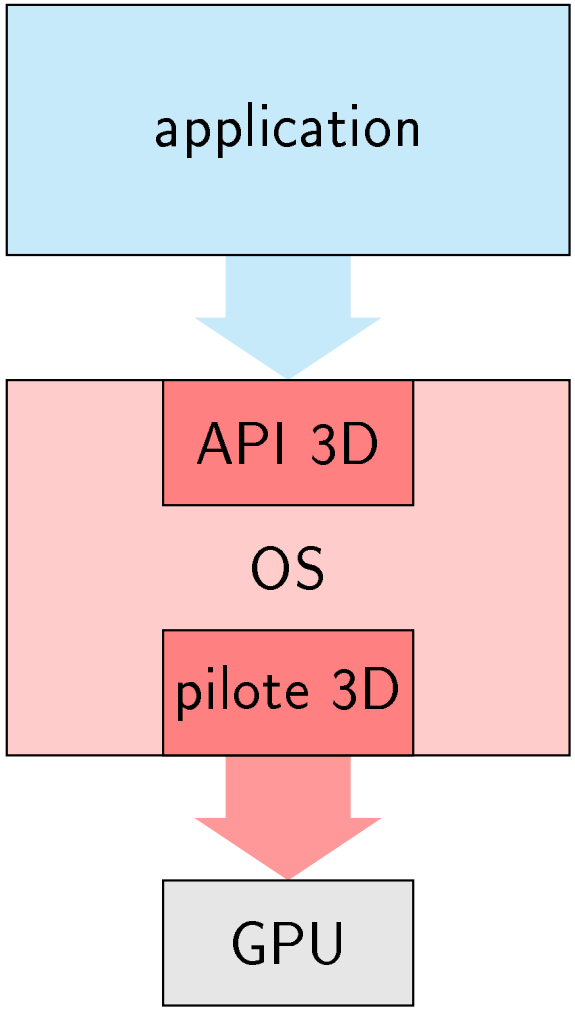
- de bénéficier de l'accélération graphique 3D
- dans un navigateur (PC) ou un appareil mobile,
- via le langage JavaScript et HTML5.

API orientée uniquement vers la *visualisation* et la *transformation* d'objets 3D.

Problématique du rendu 3D

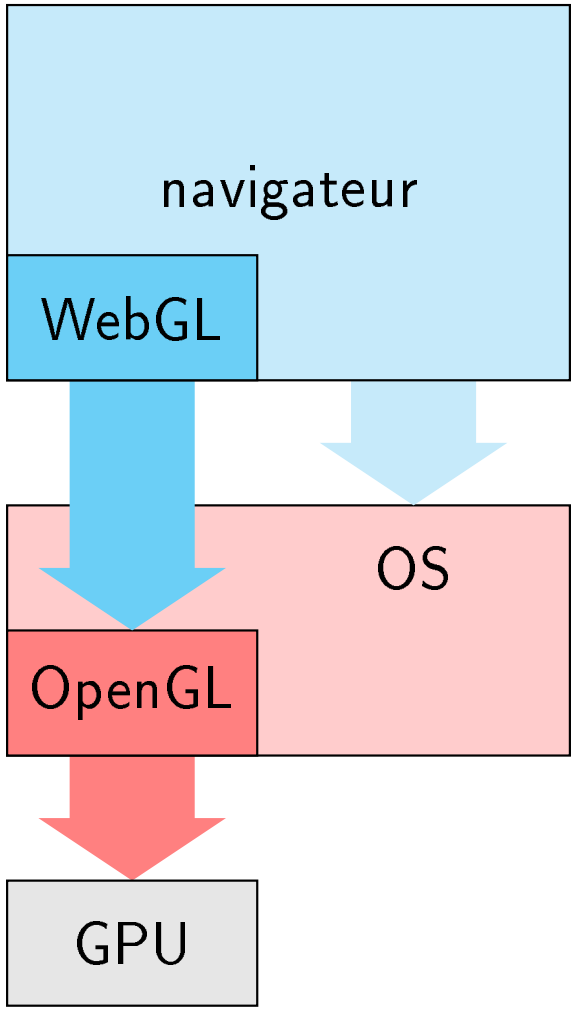


application avec pilote intégré



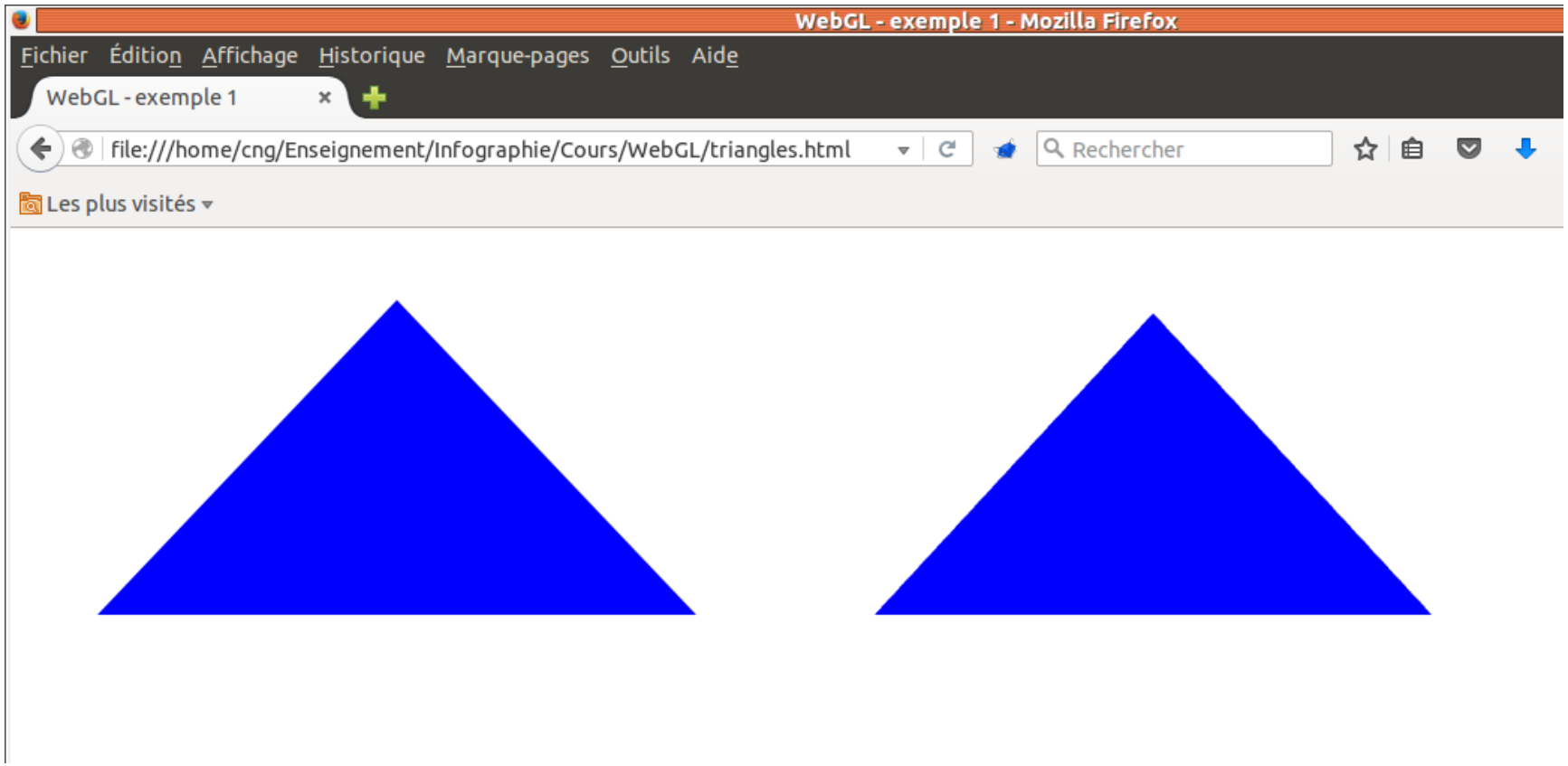
application utilisant une API

Problématique du rendu 3D
WebGL



application portable avec WebGL

Un exemple


WebGL - exemple 1 - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils Aide
WebGL - exemple 1
file:///home/cng/Enseignement/Infographie/Cours/WebGL/triangles.html
Rechercher
Les plus visités

Un exemple

Deux canvas en HTML5

```
<html>
<head>
...
</head>
  <body onload="draw2D();draw3D();">
    <canvas id="shapecanvas" class="front" width="500" height="500">
    </canvas>
    <canvas id="shapecanvas2" style="border: none;" width="500" height="500">
    </canvas>
  <br/>
  </body>
</html>
```

Un exemple

Deux canvas en HTML5

```
<html>
<head>
...
</head>
  <body onload="draw2D();draw3D();">
    <canvas id="shapecanvas" class="front" width="500" height="500">
    </canvas>
    <canvas id="shapecanvas2" style="border: none;" width="500" height="500">
    </canvas>
  <br/>
  </body>
</html>
```

event handlers

définition de deux canvas

Un exemple

Un triangle 2D (canvas HTML5)

```
<script type="text/javascript">
  function draw2D()  {
      var canvas = document.getElementById("shapecanvas");
      var c2dCtx = canvas.getContext('2d');
      c2dCtx.fillStyle = "#0000ff";
      c2dCtx.beginPath();
      c2dCtx.moveTo(250, 40);
      c2dCtx.lineTo(450, 250);
      c2dCtx.lineTo(50, 250);
      c2dCtx.closePath();
      c2dCtx.fill();
  }
```

Un exemple

Un triangle 2D (canvas HTML5)

```
<script type="text/javascript">
  function draw2D()  {
      var canvas = document.getElementById("shapecanvas");
      var c2dCtx = canvas.getContext('2d');
      c2dCtx.fillStyle = "#0000ff";          couleur bleue
      c2dCtx.beginPath();                    définition d'un chemin
      c2dCtx.moveTo(250, 40);
      c2dCtx.lineTo(450, 250);
      c2dCtx.lineTo(50, 250);
      c2dCtx.closePath();
      c2dCtx.fill();                         remplissage
  }
```

Un exemple

Un triangle 3D (WebGL)

```
<script type="text/javascript">
  //variables globales
  function draw3D()  {
        var canvas = document.getElementById("shapecanvas2");
        var glCtx = canvas.getContext("experimental-webgl");
        glCtx.viewport(0, 0, canvas.width, canvas.height);
        // vertex data ; color data
        // vertex shader ; fragment shader
        // projection
        update(glCtx);
  }

  function draw(ctx) { ... }

  function update(gl) { draw(gl); }
```

Un exemple

Un triangle 3D (WebGL)

```
<script type="text/javascript">
  //variables globales
  function draw3D()  {
        var canvas = document.getElementById("shapecanvas2");
        var glCtx = canvas.getContext("experimental-webgl");
        glCtx.viewport(0, 0, canvas.width, canvas.height);
        // vertex data ; color data
        // vertex shader ; fragment shader
        // projection
        update(glCtx);
  }

  function draw(ctx) { ... }

  function update(gl) { draw(gl); }
```

aire totale du canvas

rendu pipeline

Un exemple

WebGL - // vertex data

```
vertBuffer = glCtx.createBuffer();
glCtx.bindBuffer(glCtx.ARRAY_BUFFER, vertBuffer);
var verts = [
  0.0, 1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
  0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0,
  0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.0,
  0.0, 1.0, 0.0, 0.0, 0.0, -1.0, -1.0, 0.0, 0.0
];
glCtx.bufferData(glCtx.ARRAY_BUFFER, new Float32Array(verts),
                 glCtx.STATIC_DRAW);
```

Un exemple

WebGL - // vertex data

```
vertBuffer = glCtx.createBuffer();
glCtx.bindBuffer(glCtx.ARRAY_BUFFER, vertBuffer);
var verts = [
  0.0, 1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
  0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0,
  0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.0,
  0.0, 1.0, 0.0, 0.0, 0.0, -1.0, -1.0, 0.0, 0.0
];
glCtx.bufferData(glCtx.ARRAY_BUFFER, new Float32Array(verts),
                 glCtx.STATIC_DRAW);
```

pyramide sans base

Float32Array : buffer binaire

Un exemple

WebGL - // color data

```
colorBuffer = glCtx.createBuffer();
glCtx.bindBuffer(glCtx.ARRAY_BUFFER, colorBuffer);
var faceColors = [
    [0.0, 0.0, 1.0, 1.0], // Front  (blue)
    [1.0, 1.0, 0.0, 1.0], // right   (yellow)
    [0.0, 1.0, 0.0, 1.0], // back   (green)
    [1.0, 0.0, 0.0, 1.0], // left   (red)
];
var vertColors = [];
faceColors.forEach(function(color) {
    [0,1,2].forEach(function () {
      vertColors = vertColors.concat(color);
    });
});
glCtx.bufferData(glCtx.ARRAY_BUFFER, new Float32Array(vertColors),
                 glCtx.STATIC_DRAW);
```

Un exemple

WebGL - // color data

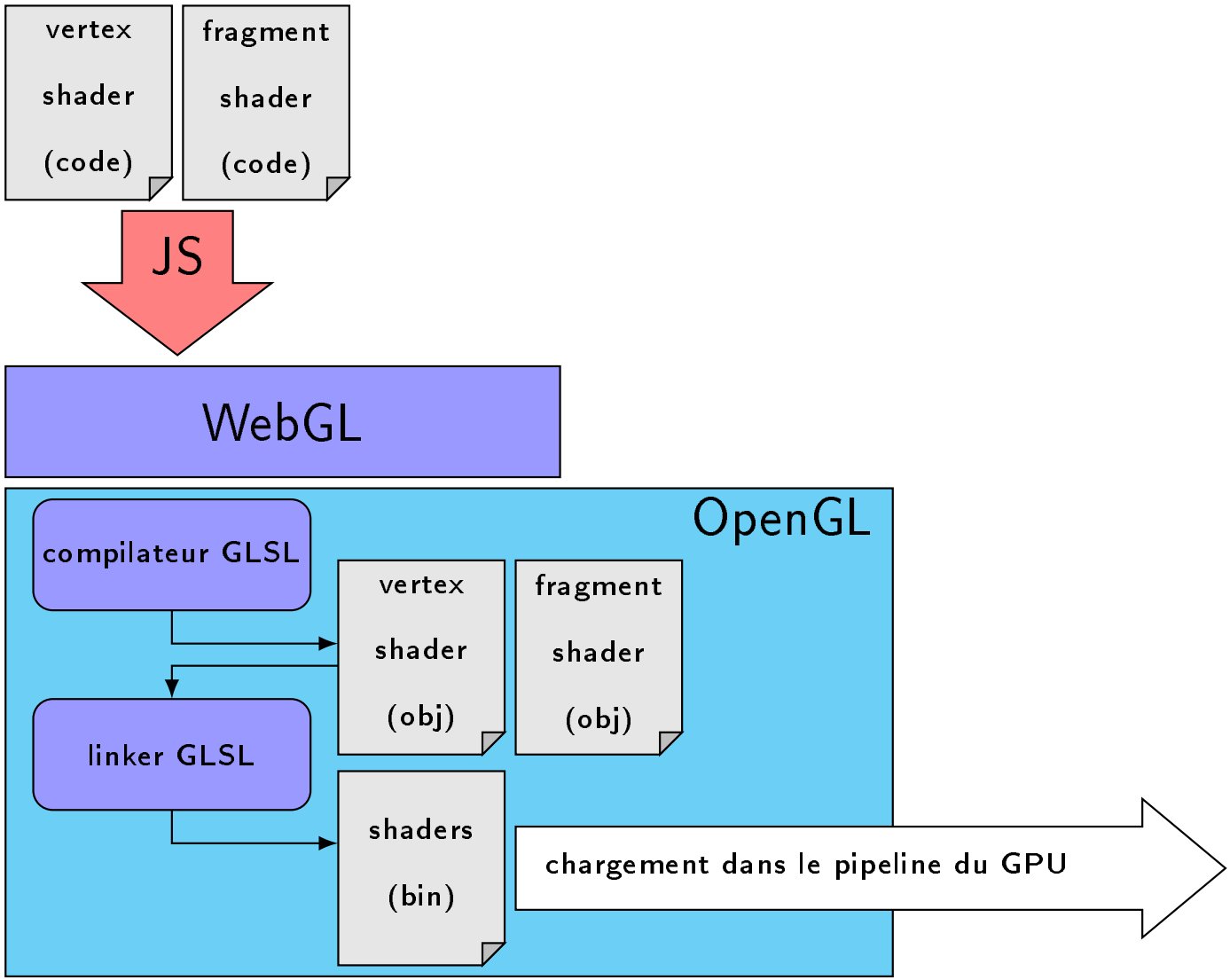
```
colorBuffer = glCtx.createBuffer();
glCtx.bindBuffer(glCtx.ARRAY_BUFFER, colorBuffer);
var faceColors = [
    [0.0, 0.0, 1.0, 1.0], // Front  (blue)
    [1.0, 1.0, 0.0, 1.0], // right   (yellow)
    [0.0, 1.0, 0.0, 1.0], // back   (green)
    [1.0, 0.0, 0.0, 1.0], // left   (red)
];
var vertColors = [];
faceColors.forEach(function(color) {
    [0,1,2].forEach(function () {
      vertColors = vertColors.concat(color);
    });
});
glCtx.bufferData(glCtx.ARRAY_BUFFER, new Float32Array(vertColors),
                 glCtx.STATIC_DRAW);
```

structure intermédiaire

12 entrées de couleur

m̂ couleur / 3 som.

GLSL


vertex
shader
(code)
fragment
shader
(code)
JS
WebGL
OpenGL
compilateur GLSL
vertex
shader
(obj)
fragment
shader
(obj)
linker GLSL
shaders
(bin)
chargement dans le pipeline du GPU

Un exemple

WebGL - // vertex shader

```
var vertShaderCode = document.getElementById("vertshader").textContent;
var vertShader = glCtx.createShader(glCtx.VERTEX_SHADER);

glCtx.shaderSource(vertShader, vertShaderCode);
glCtx.compileShader(vertShader);
```

WebGL - // fragment shader

```
var fragShaderCode = document.getElementById("fragshader").textContent;

var fragShader = glCtx.createShader(glCtx.FRAGMENT_SHADER);

glCtx.shaderSource(fragShader, fragShaderCode);
glCtx.compileShader(fragShader);
```

Un exemple

shaders en GLSL

WebGL - // vertex shader

pour chaque sommet

```
var vertShaderCode = document.getElementById("vertshader").textContent;
var vertShader = glCtx.createShader(glCtx.VERTEX_SHADER);

glCtx.shaderSource(vertShader, vertShaderCode);
glCtx.compileShader(vertShader);
```

chargement
compilation

WebGL - // fragment shader

pour chaque fragment

```
var fragShaderCode = document.getElementById("fragshader").textContent;

var fragShader = glCtx.createShader(glCtx.FRAGMENT_SHADER);

glCtx.shaderSource(fragShader, fragShaderCode);
glCtx.compileShader(fragShader);
```

Un exemple

WebGL - // projection

```
modelViewMatrix = mat4.create();  // global - used in animation
mat4.translate(modelViewMatrix, modelViewMatrix, [0, 0, -3]);

projectionMatrix = mat4.create();
mat4.perspective(projectionMatrix, Math.PI / 4,
                 canvas.width / canvas.height, 1, 100);

rotationAxis = vec3.create();   // global - used in animation
vec3.normalize(rotationAxis, [0, 1, 0]);
```

Un exemple

WebGL - // projection

```
modelViewMatrix = mat4.create();  // global - used in animation
mat4.translate(modelViewMatrix, modelViewMatrix, [0, 0, -3]);
                                                    placement caméra/objets
projectionMatrix = mat4.create();
mat4.perspective(projectionMatrix, Math.PI / 4,     type de projection
                 canvas.width / canvas.height, 1, 100);

rotationAxis = vec3.create();   // global - used in animation
vec3.normalize(rotationAxis, [0, 1, 0]);
```

Un exemple

WebGL - draw(ctx)

```
ctx.clearColor(1.0, 1.0, 1.0, 1.0);
ctx.enable(ctx.DEPTH_TEST);
ctx.clear(ctx.COLOR_BUFFER_BIT  | ctx.DEPTH_BUFFER_BIT);

ctx.useProgram(shaderProg);
ctx.bindBuffer(ctx.ARRAY_BUFFER, vertBuffer);
ctx.vertexAttribPointer(shaderVertexPositionAttribute, 3,
                        ctx.FLOAT, false, 0, 0);
ctx.bindBuffer(ctx.ARRAY_BUFFER, colorBuffer);
ctx.vertexAttribPointer(shaderVertexColorAttribute, 4,
                        ctx.FLOAT, false, 0, 0);
ctx.uniformMatrix4fv(shaderProjectionMatrixUniform, false,
                     projectionMatrix);
mat4.rotate(modelViewMatrix, modelViewMatrix, Math.PI/4, rotationAxis);
ctx.uniformMatrix4fv(shaderModelViewMatrixUniform, false,
                     modelViewMatrix);

ctx.drawArrays(ctx.TRIANGLES, 0, 12);
```

Un exemple

WebGL - draw(ctx)

appelé depuis le wrapper update()

```
ctx.clearColor(1.0, 1.0, 1.0, 1.0);
ctx.enable(ctx.DEPTH_TEST);
ctx.clear(ctx.COLOR_BUFFER_BIT  | ctx.DEPTH_BUFFER_BIT);

ctx.useProgram(shaderProg);
ctx.bindBuffer(ctx.ARRAY_BUFFER, vertBuffer);
ctx.vertexAttribPointer(shaderVertexPositionAttribute, 3,
                        ctx.FLOAT, false, 0, 0);
ctx.bindBuffer(ctx.ARRAY_BUFFER, colorBuffer);
ctx.vertexAttribPointer(shaderVertexColorAttribute, 4,
                        ctx.FLOAT, false, 0, 0);
ctx.uniformMatrix4fv(shaderProjectionMatrixUniform, false,
                     projectionMatrix);
mat4.rotate(modelViewMatrix, modelViewMatrix, Math.PI/4, rotationAxis);
ctx.uniformMatrix4fv(shaderModelViewMatrixUniform, false,
                     modelViewMatrix);

ctx.drawArrays(ctx.TRIANGLES, 0, 12);
```

Un exemple

GLSL - scripts shaders

```
<script id="vertshader" type="x-shader/vertshader">
      attribute vec3 vertPos;
      attribute vec4 vertColor;
      uniform mat4 mvMatrix;
      uniform mat4 pjMatrix;
      varying lowp vec4 vColor;
      void main(void) {
        gl_Position = pjMatrix * mvMatrix * vec4(vertPos, 1.0);
        vColor = vertColor;
      }
 </script>

 <script id="fragshader" type="x-shader/fragshader">
      varying lowp vec4 vColor;
      void main(void) {
        gl_FragColor = vColor;
      }
</script>
```

Un exemple

GLSL - scripts shaders

```
<script id="vertshader" type="x-shader/vertshader">
      attribute vec3 vertPos;
      attribute vec4 vertColor;
      uniform mat4 mvMatrix;
      uniform mat4 pjMatrix;
      varying lowp vec4 vColor;
      void main(void) {
        gl_Position = pjMatrix * mvMatrix * vec4(vertPos, 1.0);
        vColor = vertColor;
      }
 </script>

 <script id="fragshader" type="x-shader/fragshader">
      varying lowp vec4 vColor;
      void main(void) {
        gl_FragColor = vColor;
      }
</script>
```

attribute = lien avec WebGL

uniform = lien avec JS (ro)

modifié par CPU (pas par GPU)

varying = lien entre shaders

Un exemple

Animation (Firefox, Chrome)

```
function update(gl) {
    requestAnimationFrame(function() { update(gl); });
    draw(gl);
}
```

WebGL ne dispose d'aucune fonction d'animation.
Cette fonctionnalité est offerte par la plupart des navigateurs.
Le callback se fait environ 60 fois / s.

Un exemple

WebGL - draw(ctx) 2ème version

```
var onerev = 10000; // ms
var exTime = Date.now();
...
ctx.uniformMatrix4fv(shaderProjectionMatrixUniform, false,
                     projectionMatrix);

var now = Date.now();
var elapsed = now - exTime;
exTime = now;
var angle = Math.PI * 2 * elapsed/onerev;
mat4.rotate(modelViewMatrix, modelViewMatrix, angle, rotationAxis);

ctx.uniformMatrix4fv(shaderModelViewMatrixUniform, false,
                     modelViewMatrix);

ctx.drawArrays(ctx.TRIANGLES, 0, 12);
```

Un exemple

WebGL - draw(ctx) 2ème version

```
var onerev = 10000; // ms
var exTime = Date.now();
...
ctx.uniformMatrix4fv(shaderProjectionMatrixUniform, false,
                     projectionMatrix);

var now = Date.now();
var elapsed = now - exTime;
exTime = now;
var angle = Math.PI * 2 * elapsed/onerev;
mat4.rotate(modelViewMatrix, modelViewMatrix, angle, rotationAxis);

ctx.uniformMatrix4fv(shaderModelViewMatrixUniform, false,
                     modelViewMatrix);

ctx.drawArrays(ctx.TRIANGLES, 0, 12);
```

un tour en 10 s

Pour aller plus loin
Unreal Engine 4

Conclusion

WebGL permet d'accéder à l'accélération 3D via JS mais reste de bas niveau :

- aucun outil de modélisation de la géométrie,
- gère uniquement le pipeline graphique 3D $\rightarrow$ 2D pour une image, l'animation nécessite du code supplémentaire,
- les mouvements et les interactions entre les objets et l'environnement nécessite du code supplémentaire,
- la gestion des événements (entrées utilisateur), la sélection d'objet, etc. nécessite du code supplémentaire.

Plan

Introduction

Une biliothèque de haut niveau, l'API la plus utilisée pour développer et maintenir plus aisément un projet WebGL.

[Wikipédia] Elle permet des rendus en webGL, CSS3D et SVG.

Elle contient par exemple les fonctionnalités suivantes :

- animation par squelette,
- LOD (niveau de détails pour les objets),
- chargement de fichiers au formats `.OBJ`, `.JSON`, `.FBX`,
- système de particules.

La pyramide avec three.js

three.js - code complet

```
function draw3D() {

  function animate() {
    requestAnimationFrame(animate);
    pyramid.rotateY(Math.PI / 180);
    renderer.render(scene, camera);
  }

  var geo = new THREE.CylinderGeometry(0,2,2,4,1, true);

  var faceColors = [0xff0000, 0x00ff00, 0x0000ff, 0xffff00];
  faceColors.forEach( function(color, idx) { geo.faces[2 * idx + 1].color.setHex(color);});

  var pyramid = new THREE.Mesh(geo, new THREE.MeshBasicMaterial({vertexColors: THREE.FaceColors}));

  var camera = new THREE.PerspectiveCamera(45, 1,0.1, 100);
  pyramid.position.y = 1; camera.position.z = 6;

  var scene = new THREE.Scene();
  scene.add(pyramid);

  var renderer = new THREE.WebGLRenderer();
  renderer.setSize(500,500);
  var span = document.getElementById("shapecanvas2");
  span.appendChild( renderer.domElement );

  animate();
}
```

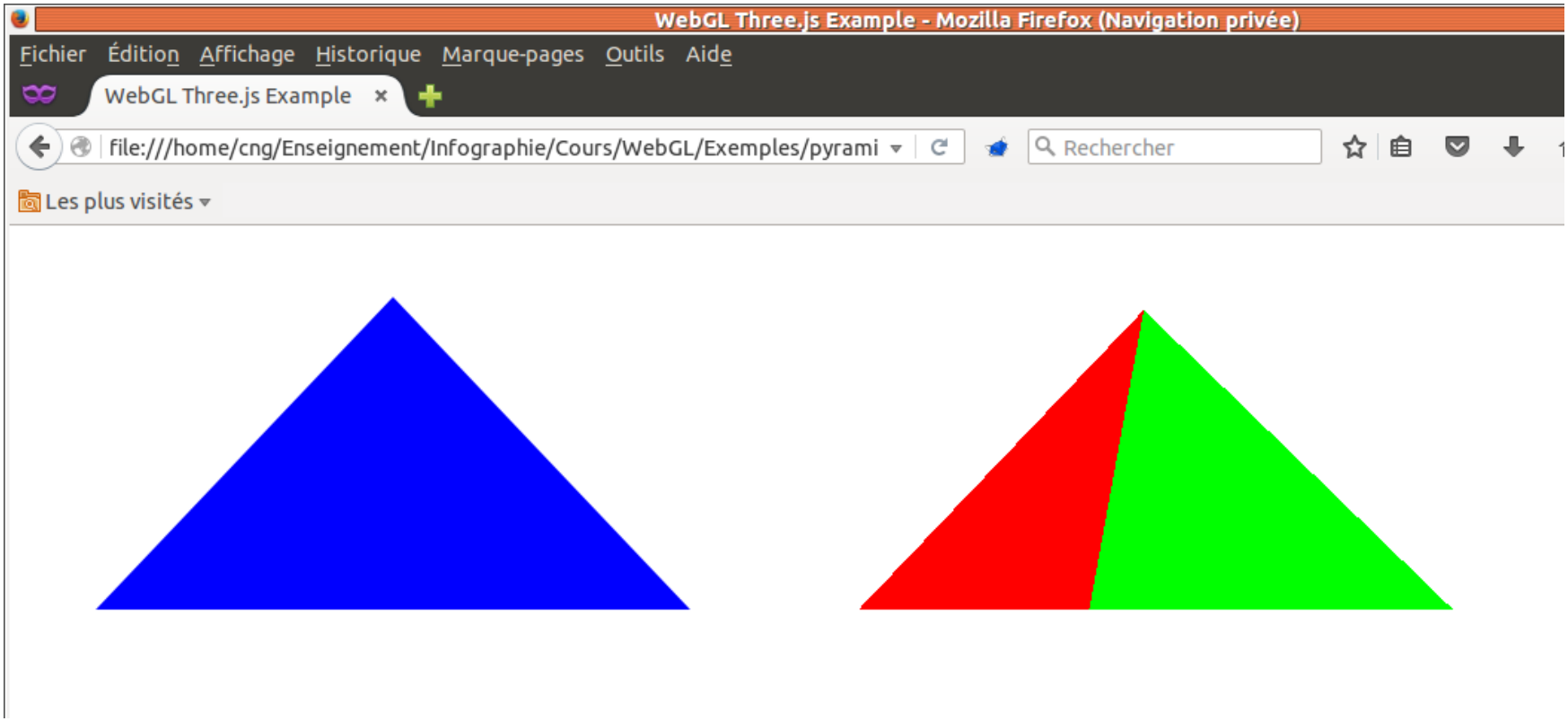
La pyramide avec three.js

three.js - code complet

```
function draw3D() {

  function animate() {
    requestAnimationFrame(animate);
    pyramid.rotateY(Math.PI / 180);
    renderer.render(scene, camera);
  }

  var geo = new THREE.CylinderGeometry(0,2,2,4,1, true);

  var faceColors = [0xff0000, 0x00ff00, 0x0000ff, 0xffff00];
  faceColors.forEach( function(color, idx) { geo.faces[2 * idx + 1].color.setHex(color);});

  var pyramid = new THREE.Mesh(geo, new THREE.MeshBasicMaterial({vertexColors: THREE.FaceColors}));

  var camera = new THREE.PerspectiveCamera(45, 1,0.1, 100);
  pyramid.position.y = 1; camera.position.z = 6;

  var scene = new THREE.Scene();
  scene.add(pyramid);

  var renderer = new THREE.WebGLRenderer();
  renderer.setSize(500,500);
  var span = document.getElementById("shapecanvas2");
  span.appendChild( renderer.domElement );

  animate();
}
```

span (et non canvas)
car 3js l'implante

La pyramide avec three.js



Géométrie et couleurs

Three.js inclut des fonctions de génération de primitives 3D (cube, sphere, cylinder, torus, tetrahedron, icosahedron, octahedron, plane, tube, text, etc.).

Dans l'exemple, on utilise la primitive cylindre (!) avec un rayon supérieur à 0 et seulement 4 faces pour le pourtour (avec *open-ended* à vrai).

Les couleurs, codées en hexadécimal, sont associées aux faces via `geo.faces`. Avec un rayon supérieur de 0, seules les faces impaires sont traitées (chaque face ne comporte qu'un triangle sur deux).

Propriétés de la matière

Le mot-clé `material` détermine le rendu de la surface d'un objet ainsi que son interaction avec la lumière (comme en WebGL).

Un `mesh` permet d'associer une géométrie à un `material`. Un `MeshBasicMaterial` ne nécessite pas de lumière pour le rendu (autres possibilités : `MeshLambertMaterial` et `MeshPhongMaterial`).
Remarque : Le rendu « fil-de-fer », conseillé pour le debug, est possible par l'option `wireframe: true`.

Three.js prend en charge l'allocation du tableau des couleurs et son chargement dans le buffer approprié. De plus, le code GLSL correspondant, sa compilation et son édition de lien sont masqués.

Placement

Three.js évite la manipulation directe des matrices 4×4 en proposant une interface plus intuitive via les propriétés `position`, `rotation` et `scale` des objets.

La position par défaut est $(0, 0, 0)$. Pour placer la pyramide à la position $(0, 1, 0)$ dans la scène il faut simplement écrire :

```
pyramid.position.y = 1;
```

Éloigner la caméra de l'origine de la scène se fait par :

```
camera.position.z = 6;
```

Mise en scène

Au lieu de définir une matrice de projection, comme en WebGL, on spécifie plus naturellement une caméra « perspective » par :

```
var camera = new THREE.PerspectiveCamera(45, 1, 0.1, 100);
```

Puis on place les objets dans la scène :

```
var scene = new THREE.Scene();
scene.add(pyramid);
```

Enfin, on produit l'image finale par :

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

L'animation se fera simplement par l'appel de la méthode `rotateY`.

Graphe de scène

Pour obtenir une scène comportant deux pyramides pivotant en sens opposé, il suffit :

1. de créer et de positionner une autre pyramide :

```
var pyramid2 = pyramid1.clone();
pyramid2.position.set(2.5, 1, 0);
```

2. de l'ajouter au graphe de scène :

```
scene.add(pyramid2);
```

3. de l'animer :

```
pyramid2.rotateY(-(Math.PI/ 180));
```

Un graphe de scène est nécessaire dès qu'une scène comporte plus d'un objet, ce qui est le cas avec un objet et une caméra.

Relations d'héritage

Une relation de dépendance couplée à celle d'héritage facilitent la manipulation ou la transformation d'un groupe d'objets. Vouloir faire tourner tous les objets de la scène autour de l'axe Y nécessite :

- de créer un parent commun à tous les objets :

```
var multi = new THREE.Object3D();
multi.add(pyramid1);
multi.add(pyramid2);
```

- de positionner et d'ajouter ce parent commun à la scène :

```
multi.position.z = 0;
var scene = new THREE.Scene();
scene.add(multi);
```

- de l'animer :

```
function animate() {
  ...
  multi.rotateY(Math.PI/360);
```

De l'ombre à la lumière

On rappelle que le `THREE.MeshBasicMaterial` ne nécessite aucune source lumineuse pour le rendu.

Pour augmenter le réalisme, on doit ajouter des sources lumineuses :

```
var light = new THREE.DirectionalLight(0xe0e0e0);
light.position.set(5,2,5).normalize();
light.castShadow = true;
light.shadowDarkness = 0.5;
light.shadowCameraRight = 5;
light.shadowCameraLeft = -5;
light.shadowCameraTop = 5;
light.shadowCameraBottom = -5;
light.shadowCameraNear = 2;
light.shadowCameraFar = 100;
scene.add(light);
scene.add(new THREE.AmbientLight(0x101010));
```

Pour éviter des calculs inutiles, on spécifie quels objets projettent une ombre (par `<obj>.castShadow = true;`) et quelles objets les reçoivent (par `<obj>.receiveShadow = true;`).

De l'ombre à la lumière

On rappelle que le `THREE.MeshBasicMaterial` ne nécessite aucune source lumineuse pour le rendu.

Pour augmenter le réalisme, on doit ajouter des sources lumineuses :

```
var light = new THREE.DirectionalLight(0xe0e0e0);
light.position.set(5,2,5).normalize();
light.castShadow = true;
light.shadowDarkness = 0.5;
light.shadowCameraRight = 5;
light.shadowCameraLeft = -5;
light.shadowCameraTop = 5;
light.shadowCameraBottom = -5;
light.shadowCameraNear = 2;
light.shadowCameraFar = 100;
scene.add(light);
scene.add(new THREE.AmbientLight(0x101010));
```

z-buffer shadow mapping

définition d'une shadow camera

on débouche les ombres

Pour éviter des calculs inutiles, on spécifie quels objets projettent une ombre (par `<obj>.castShadow = true;`) et quelles objets les reçoivent (par `<obj>.receiveShadow = true;`).

Animation par interpolation (tweening)

La bibliothèque `tween.js` permet la création et l'enchaînement de séquences d'animation.

L'interpolation se fait par défaut de façon linéaire entre des points définis par l'utilisateur, mais elle peut aussi suivre une courbe (*easing curves*).

L'enchaînement des mouvements de caméra et d'objets d'une scène « complexe » est défini préalablement.

Certaines animations nécessitent des changements de repère (cf. `applyMatrix`).

Mise en scène d'un mouvement de caméra vers l'avant de 8 s, suivi de l'ouverture d'une porte durant 3 s et conclu par l'entrée dans une pièce en 5 s.

```
function setup() {
  var tweenOpenDoor = new TWEEN.Tween(door.rotation)
      .to({y: door.rotation.y - Math.PI}, 3000);
  var tweenWalkUp = new TWEEN.Tween(camera.position)
      .to({z: camera.position.z - 25}, 8000);
  var tweenWalkIn = new TWEEN.Tween(camera.position)
      .to({z: camera.position.z - 32}, 5000);

  tweenOpenDoor.chain(tweenWalkIn);
  tweenWalkUp.chain(tweenOpenDoor);
  tweenWalkUp.start();
}
```

Prise en compte des calculs d'interpolation :

```
function animate() {
  ...
  TWEEN.update();
```

Modélisation par définition d'un profil

Un mur peut être créé en utilisant les API `ShapeGeometry` ou `ExtrudeGeometry` de Three.js.

Une forme est définie par des instructions 2D classiques du canvas (`moveto()`, `lineto`, ...) puis convertie en un objet 3D.

```
var doorWall = new THREE.Shape();
doorWall.moveTo( 0, 0 );
doorWall.lineTo( 23, 0 );
...
geo = new THREE.ShapeGeometry(doorWall);
var dWall = new THREE.Mesh(geo, new THREE.MeshBasicMaterial({color: 0xff0000}));
```

Ajout de textures

Three.js autorise non seulement le *texture mapping* mais aussi le *light mapping* et le *bump mapping*.

L'interpolation d'une texture sur une surface se fait en plusieurs étapes :

1. chargement d'un fichier image (dimensions : 2^n) :

```
floortex = THREE.ImageUtils.loadTexture('floor.png');
```

2. spécification des propriétés de répétition :

```
floortex.wrapS = THREE.RepeatWrapping;
floortex.wrapT = THREE.RepeatWrapping;
floortex.repeat.x = 10;
floortex.repeat.y = 10;
```

3. association de la texture à l'objet (2D ou 3D) :

```
geo = new THREE.PlaneGeometry(20, 50);
floor = new THREE.Mesh(geo, new THREE.MeshBasicMaterial({ map: floortex }));
```

Modèles 3D prédéfinis

Three.js intègre un certain nombre de chargeurs (*loader*) de format de fichier d'objets 3D, citons :

- `OBJ` (`MTL` : *material* associé) : format de description d'une géométrie 3D défini par Wavefront et adopté par de nombreux logiciels 3D (Maya, Blender, 3DS Max, Lightwave, ...).
- `Collada` : format exporté par Trimble SketchUp notamment.

Exemple d'un sofa au format Collada :

```
var cloader = new THREE.ColladaLoader();
cloader.load( './sofawork.dae', function ( collada ) {
    sofa = collada.scene;
    sofa.position.set(5, -2, 16);
    scene.add(sofa);
    ...
```

Gestion de la caméra

Three.js fournit un module `OrbitControls.js` qui facilite grandement ce mécanisme. Exemple :

```
<script src="js/controls/OrbitControls.js"></script>
...
function draw3D()
  var controls;
  ...
  function updateControls() {
    controls.update();  // screen refresh via rAF
  }
  ...
  controls = new THREE.OrbitControls( camera );
  controls.addEventListener( 'change', updateControls );
```

Interaction

Il existe de multiples façons d'interagir avec une scène 3D c'est pourquoi Three.js intègre plusieurs API :

- `TrackballControls.js` simule l'utilisation d'un trackball,
- `FirstPersonControl.js` fournit le standard d'interaction des FPS,
- `FlyControl.js` intègre un contrôleur de vol avec roulis et tangage,
- `OculusControls.js` supporte l'interaction offerte par l'Occulus Rift.

Picking

Une forme d'interaction fondamentale est la sélection d'un objet dans une scène 3D. La possibilité de changer le point de vue complique singulièrement la problématique.

La classe `RayCaster` permet de lancer une demi-droite dans une scène 3D et de déterminer les objets intersectés.

```
var raycaster = new THREE.Raycaster();
...
// update the picking ray with the camera and mouse position
raycaster.setFromCamera( mouse, camera );

// calculate objects intersecting the picking ray
var intersects = raycaster.intersectObjects( scene.children );
```