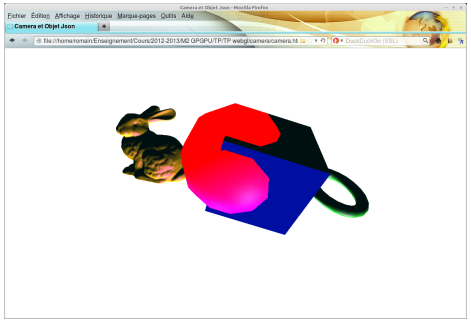
Sommaire intermédiaire

WebGL

Depuis le standard HTML5 [28] il devient possible de spécifier des canvas dans lesquels le navigateur a accès directement au matériel (son, video). On peut aussi créer un canvas pour mettre de l'OpenGL : c'est la partie webGL. La programmation se fait en javascript (interprété par le navigateur). On a accès aux contrôles, document, transmission via javascript (DOM, Ajax).



28. spéc. sur `http://www.w3.org/TR/html5/`

Qu'est-ce qu'il y a dedans ?

On retrouve presque les mêmes fonctions (ou absences de) qu'avec OpenGL ES. Pas de mode différé :

```
glBegin(GL_TRIANGLES);
  glVertex3f(1.0, 2.0f, 3.0f);
  glVertex3f(2.0, 3.0f, 4.0f);
  glVertex3f(3.0, 4.0f, 5.0f);
glEnd();
```

→ on passe toute les données via VBO et textures (cf. cours précédent). Pas de mode deprecated, cela n'existe plus du tout.

Pour éviter le code 'OpenGLES'

On utilise des bibliothèques, comme Glut avec les applications lourdes, pour accélérer l'écriture du code Javascript (contrôles, caméras, chargement d'objets, effets, ...). Il n'y en a pas d'officielles mais on peut citer les plus connues :

- Copperlicht, payant, `http://www.ambiera.com/copperlicht/`,
- GLGLE, gratuit, `http://www.glge.org/`
- CreativeJS, gratuit, plutôt orienté prototypage, `http://creativejs.com/`
- Three.js (celle que l'on utilisera), gratuite. Lancée à partir d'un Google SoC., `http://threejs.org/`.

Comment cela s'écrit ?

Les balises HTML restent :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ... />
<html xmlns="http://www.w3.org/1999/xhtml" ... />

<head>
   <title>Mon premier WebGL</title>
   <meta http-equiv="content-type" content="text/html ... />
   <style>canvas { width: 100%; height: 100% }</style>
</head>

<body>
   ...
<body>
```


Et le corps WebGL

On écrit ensuite les fonctions dans la partie <body> ... </body> :

```
<body>
  //on charge une librairie cela rendra le travail plus facile
  <script src="three.js"></script>

  <script> //on décrit la scène, la caméra

  var scene = new THREE.Scene();

  var camera = new THREE.PerspectiveCamera(75, \\
    window.innerWidth/window.innerHeight, 0.1, 1000);

  var renderer = new THREE.WebGLRenderer();
  renderer.setSize(window.innerWidth, window.innerHeight);

  document.body.appendChild(renderer.domElement);
```


...

```
//enfin on décrit une gométrie et on l'ajouter à la scène
 var geometry = new THREE.CubeGeometry(1,1,1);
 var material = new THREE.MeshBasicMaterial({color: 0x0ffff0});

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

  scene.add(cube);

  camera.position.z = 5;
```

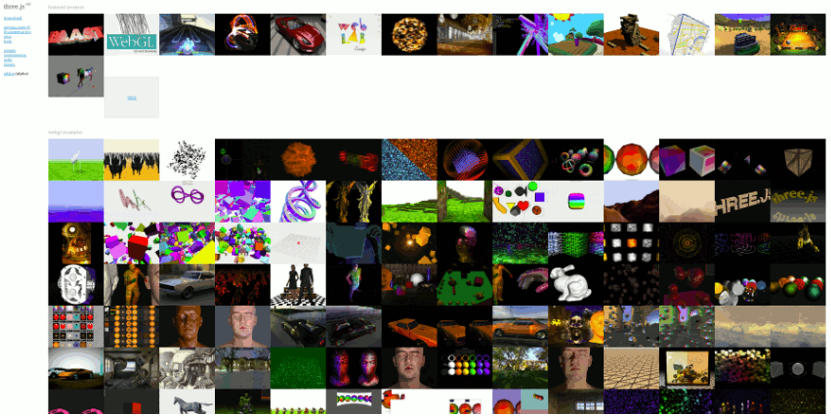

Enfin on demande l'affichage :

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

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

   renderer.render(scene, camera);
   }

render();

</script>
</body>
```

Quelques liens :

- https://www.khronos.org/webgl/
- http://learningwebgl.com/cookbook/index.php/Main_Page

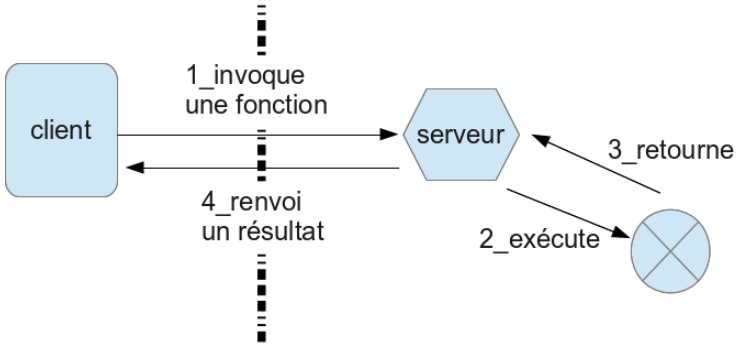
Three.js



Three.js

Des démos ?

Tout cela ne vous rappelle pas quelque chose ?



Convertir des objets

Il serait très consommateur de bande passante de transférer les objets du serveur Web vers le navigateur client en texte simple (cf. les formats .OFF). On utilise donc des compressions binaires, avec ou sans pertes :

- de .OBJ en .JS binaire

```
python convert_obj_three.py
  -i StTrophime-3_7.obj -o StTrophime3-7.js -a center
  -s smooth -t binary -d normal
```

- de XXX en .CTM [29]

```
ctmconv.exe stanford_bunny.off bunny.ctm --calc-normals
```

29. http://openctm.sourceforge.net/

Au fait, vous avez essayé ?

La 3D avec lunette « made easy », avec ThreeJS [30].

```
https://threejs.org/examples/#webgl_effects_anaglyph
```

30. OpenSceneGraph le fait également nativement pour les applications lourdes