



Institut Universitaire
de Technologie
Aix-Marseille Université

Imagerie Numérique

M4105Cin – Moteurs 3D avancés

2. WebGL

DUT INFO 2ème année 2019-2020

Sébastien THON

IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

INTRODUCTION

On peut afficher de la 3D interactive dans une page web au moyen des 3 technologies suivantes :

HTML : définit le contenu de pages web (fond).

CSS : définit la présentation de pages web (forme).

JavaScript : permet de programmer le comportement dynamique de pages web.

WebGL : interface JavaScript permettant d'utiliser OpenGL dans une page web.



HTML (Hyper Text Markup Language)

HTML est un langage à **balises** permettant de décrire des pages web, inventé en 1991 par Tim Berners-Lee, chercheur au CERN.

Ce code est écrit sous la forme de texte dans un fichier portant une extension **.htm** ou **.html**

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Titre de la page</title>
  </head>

  <body>
    <h1>Un premier titre</h1>
    <p>Un paragraphe de texte.</p>
  </body>
</html>
```

Navigateur web (« *browser* », en anglais)



Le rôle d'un navigateur web (Chrome, Firefox, Internet Explorer, Opera, Safari, etc.) est de charger des documents HTML, de les afficher et de permettre la navigation hypertexte.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Titre de la page</title>
  </head>

  <body>
    <h1>Un premier titre</h1>
    <p>Un paragraphe de texte.</p>
  </body>
</html>
```



Attention : une page web peut ne pas s'afficher de la même manière sur des navigateurs différents et sur des numéros de versions différentes → testez votre site web sur plusieurs navigateurs !

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<title>Titre de la page</title>
```

```
</head>
```

```
<body>
```

```
<h1>Un premier titre</h1>
```

```
<p>Un paragraphe de texte.</p>
```

```
</body>
```

```
</html>
```

Deux grandes parties :

L'en-tête : entre `<head>` et `</head>`

Définition du titre, de l'encodage, de mots clés, etc.

→ Informations destinées aux machines (navigateur, robots)

Le corps : entre `<body>` et `</body>`

Contenu affichable de la page web

→ Informations destinées à l'humain (et aux machines)



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Titre de la page</title>
  </head>
```

```
  <body>
    <h1>Un premier titre</h1>
    <p>Un paragraphe de texte.</p>
  </body>
```

```
</html>
```

Le navigateur web n'affiche pas les balises HTML mais s'en sert pour déterminer comment afficher le document.



2. Balises

2.1 Balises ouvrantes et fermantes

Une page HTML contient du texte délimité par des **balises** écrites entre chevrons « < » et « > ».

<balise>

Il existe des balises **ouvrantes** et des balises **fermantes** signalées par un slash « / » pour délimiter la partie du texte sur lequel agit la balise.

<balise> du texte **</balise>**

Il existe des dizaines de balises dans le langage HTML : <html>, <head>, <body>, <h1>, <h2>, <a>, <p>, , , <table>, <tr>, <td>, <form>, , <div>, ...

Exemples :

Dans votre fichier HTML, si vous écrivez :

Le texte suivant est écrit en **gras**.

Ce sera affiché ainsi dans votre navigateur web :

Le texte suivant est écrit en **gras**.

La balise **** permet de délimiter une zone de texte qui sera écrite en gras.

Exemples :

<h1>Titre le plus important</h1>

<h2>Titre moins important</h2>

<h6>Titre le moins important</h6>

<p>Paragraphe de texte</p>

<div>Bloc d'informations avec retours à la ligne</div>

Bloc d'information « inline » sans retour à la ligne

Listes non ordonnées

Une liste non ordonnée commence avec la balise `` (« *Unordered List* ») et se termine avec la balise ``.

Chaque élément de la liste commence avec la balise `` (« *List Item* ») et se termine avec la balise ``. Il est affiché précédé d'une puce.

Exemple :

``

``Premier élément``

``Deuxième élément``

``Troisième élément``

``



- Premier élément
- Deuxième élément
- Troisième élément

Listes ordonnées

Une liste ordonnée commence avec la balise `` (« *Ordered List* ») et se termine avec la balise ``.

Chaque élément de la liste commence avec la balise `` (« *List Item* ») et se termine avec la balise ``. Par défaut, il est affiché précédé d'un numéro.

Exemple :

``

`Premier élément`

`Deuxième élément`

`Troisième élément`

``



1.Premier élément

2.Deuxième élément

3.Troisième élément

Lien vers un autre site web

Lien vers un `moteur de recherche`

Dans cet exemple, on a spécifié une URL **absolue** (une adresse web complète) d'une page web. Elle peut donc se trouver sur un autre site web.

Lien vers une autre page du site (sans le http://)

Exemple, dans un fichier **page1.html** :

Lien vers `une autre page`.

Résultat :

Lien vers [une autre page](#).

Pour que cela fonctionne, il faut que le fichier **page2.html** se trouve dans un sous-répertoire **chemin** du répertoire où se trouve **page1.html**

La balise <canvas>

Ajouté en HTML5, l'élément <canvas> peut être utilisé pour dessiner des graphismes via des scripts JavaScript : dessiner des graphes, faire des compositions de photos, des animations, faire du traitement ou de l'affichage de vidéos en temps réel.

L'élément <canvas> est aussi utilisé par WebGL pour afficher des graphismes 3D avec accélération matérielle sur des pages web.

2.2 Balises auto-fermantes

Il existe aussi des balises **auto-fermantes** qui signalent un élément ponctuel dans le document HTML et qui n'ont pas besoin de fonctionner avec une paire ouvrante / fermante.

<balise />

Notez le slash « / » à la fin du nom de la balise. Il n'est pas obligatoire pour que votre page soit syntaxiquement correcte mais il est considéré comme une bonne pratique.

Exemple :

Une ligne.**
Une autre ligne
**.

La balise auto-fermante **
** permet de sauter une ligne.

D'autres balises auto-fermantes : ****, **<hr />**, etc.

2.3 Attributs

Les **balises** peuvent avoir des paramètres définits par des **attributs** selon la syntaxe suivante :

```
<balise attribut1="valeur" attribut2="valeur">
```

Exemple :

```

```

La balise `` qui permet d'insérer une image reçoit plusieurs attributs : le fichier source (src), le texte alternatif (alt), la largeur (width) et la hauteur (height) de l'image.

Seule une balise ouvrante ou auto-fermante peuvent avoir des attributs, pas les balises fermantes.

JavaScript

JavaScript permet de rendre une page web dynamique, par opposition à une page web statique créée avec du simple HTML :

- Mettre à jour des éléments de la page sans recharger la page.
- Demander au serveur un nouveau bout de page et l'insérer dans la page en cours, sans la recharger.
- Attendre que l'utilisateur fasse quelque chose (cliquer, taper au clavier, bouger la souris...) et réagir.
- Effectuer des tests
- Animer des éléments HTML.
- Afficher de la 3D dans une page web.
- etc.

1. Le langage JavaScript

Un programme JavaScript est composé d'instructions séparées par des point-virgule ';

```
<!DOCTYPE html>
<html>
  <body>
```

*La balise **** a ici un **identifiant** (« resultat »), qui permet de la repérer de manière unique*

Le calcul de 2 + 3 donne ``

```
<script>
  var x = 2;
  var y = 3;
  var z = x + y;
  document.getElementById("resultat").innerHTML = z;
</script>
```

```
</body>
</html>
```

Où écrire du code JavaScript

1) Dans le fichier HTML

On écrit le code dans la partie **<head></head>** ou dans la partie **<body></body>** du fichier HTML.

Le code JavaScript doit ensuite être écrit entre des balises **<script>** et **</script>**. Il peut y avoir plusieurs scripts dans un même fichier HTML.

Exemple de JavaScript dans le <head>

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <script>
```

```
    function maFonction()
```

```
    {
```

```
      document.getElementById("demo").innerHTML = "Bonsoir";
```

```
    }
```

```
  </script>
```

```
</head>
```

```
<body>
```

```
  <p id="demo">Bonjour</p>
```

```
  <button type="button" onclick="maFonction()">Test</button>
```

```
</body>
```

```
</html>
```

Exemple de JavaScript dans le <body>

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
  <p id="demo">Bonjour</p>
```

```
  <button type="button" onclick="maFonction()">Test</button>
```

```
  <script>
```

```
    function maFonction()
```

```
    {
```

```
      document.getElementById("demo").innerHTML = "Bonsoir";
```

```
    }
```

```
  </script>
```

```
</body>
```

```
</html>
```

Remarque :

Il est préférable de placer les scripts **à la fin du <body>**.

En effet, cela peut améliorer le chargement de la page web, car la compilation des scripts peut ralentir l'affichage.

2) Dans un fichier .js séparé

On peut écrire du code JavaScript dans un fichier texte auquel on donne l'extension **.js** :

scriptBonsoir.js

```
function maFonction()  
{  
    document.getElementById("demo").innerHTML = "Bonsoir";  
}
```

Pour utiliser ce script dans le document HTML, on donne son nom à l'attribut **src** de la balise **<script>** :

```
<!DOCTYPE html>  
<html>  
  <body>  
    <p id="demo">Bonjour</p>  
    <button type="button" onclick="maFonction()">Test</button>  
    <script src="scriptBonsoir.js"></script>  
  </body>  
</html>
```

console.log()

```
<!DOCTYPE html>
```

```
<html>
```

```
  <meta charset="utf-8" />
```

```
    <title>Utilisation de la console</title>
```

```
  </head>
```

```
  <body>
```

```
    <p>Appuyez sur F12 pour afficher la console du navigateur  
    et lire ce qu'affiche le code JavaScript suivant :</p>
```

```
  <script>
```

```
    console.log("Du texte");
```

```
    console.log(5 + 6);
```

```
  </script>
```

```
</body>
```

```
</html>
```



Modification du contenu d'un élément HTML

getElementById()

Fonction JavaScript permettant de trouver un élément HTML dont on fournit l'id (identifiant, qui doit être unique).

innerHTML

Permet de modifier le contenu d'un élément HTML.

Exemple :

```
document.getElementById("demo").innerHTML = "Bonjour";
```


Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
  <p id="demo">Bonjour</p>
```

```
  <button type="button" onclick="maFonction()">Test</button>
```

```
  <script>
```

```
    function maFonction()
```

```
    {
```

```
      document.getElementById("demo").innerHTML = "Bonsoir";
```

```
    }
```

```
  </script>
```

```
</body>
```

```
</html>
```

Variables

Déclaration de variable, avec le mot clé **var** :

```
var annee ;
```

Affectation de valeur à une variable :

```
annee = 2016 ;
```

On peut déclarer une variable en lui affectant une valeur :

```
var annee = 2016 ;
```

On peut déclarer plusieurs variables en une seule instruction :

```
var annee = 2016, mois = 11, jour = 5 ;
```

Identificateurs

JavaScript est sensible à la casse (distinction minuscules/majuscules) :

```
var nomVariable ;  
var nomvariable ;
```

nomVariable et **nomvariable** sont deux variables différentes.

Types de données

On peut mettre différents types de données dans une variable JavaScript :

<code>var annee = 2016;</code>	<code>// Nombre entier</code>
<code>var temperature = 37.2</code>	<code>// Nombre réel</code>
<code>var majeur = true, accord = false ;</code>	<code>// Booléen</code>
<code>var prenom = "Jean";</code>	<code>// Chaîne de caractères</code>
<code>var marque = ["Renault", "Peugeot", "BMW"];</code>	<code>// Tableau</code>
<code>var personne = {prenom : "Jean", nom : "Martin", age : 42};</code>	<code>// Objet</code>

Une variable déclarée à laquelle on n'a pas affecté de valeur a la valeur **undefined**.

On peut connaître le type d'une variable avec l'opérateur **typeof** :

```
var mot = "du texte"  
var nombre = 432 ;  
var majeur = true ;  
var marque = ["Renault", "Peugeot", "BMW"];  
var personne = {prenom : "Jean", nom : "Martin", age : 42};  
var variable ;
```

typeof mot	→ "string"
typeof nombre	→ "number"
typeof majeur	→ "boolean"
typeof marque	→ "object", car en JS les tableaux sont des objets
typeof personne	→ "object"
typeof variable	→ "undefined"

On peut concaténer des chaînes de caractères avec l'opérateur + :

```
var nom = "Martin";  
var prenom = "Jean" ;  
var nomComplet = prenom + " " + nom ;           → "Jean Martin"
```

Si on additionne une valeur numérique à une chaîne de caractères, on obtient une chaîne :

```
var mot = "truc", nombre = 42 ;  
var resultat = mot + nombre ;                   → "truc42"
```

JavaScript évalue les expressions de la gauche vers la droite, ce qui peut produire des résultats différents :

```
var a = 2, b = 3, mot = "truc" ;  
resultat = a + b + mot;                        → "5truc"  
resultat = mot + a + b;                        → "truc23"
```

Commentaires

On peut insérer des commentaires de deux manières :

// Un commentaire

var annee ; // Un autre commentaire

/* Un commentaire

Sur plusieurs lignes */

Fonctions

Une fonction est un bloc d'instructions destiné à effectuer une tâche particulière.

Elle peut recevoir des paramètres et elle peut retourner une valeur.

Une fonction est exécutée lorsqu'elle est appelée.

// Fonction retournant le produit de ses deux paramètres :

```
function produit (a, b) {  
    return a * b;  
}
```

// Exemple d'appel :

```
var resultat = produit (4, 3) ;
```


Exemple complet :

```
<!DOCTYPE html>
```

```
<html>
```

```
  <body>
```

```
    <p>77 degrés Fahrenheit = <span id="demo"></span> degrés Celsius</p>
```

```
    <script>
```

```
      function versCelsius(fahrenheit) {  
        var celcius = (5/9) * (fahrenheit-32);  
        return celcius;  
      }
```

```
      var resultat = versCelsius(77) ;
```

```
      document.getElementById("demo").innerHTML = resultat;  // → 25
```

```
    </script>
```

```
  </body>
```

```
</html>
```

Portée des variables

1) Variables locales

Les variables déclarées dans une fonction avec le mot clé **var** ont une portée locale, elles ne peuvent être utilisées que dans cette fonction :

```
function uneFonction()  
{  
    var nom = "Martin";           // Variable locale  
}
```

Exemple

```
<!DOCTYPE html>
```

```
<html>
```

```
  <body>
```

```
    Dans la fonction : <span id="nom1"></span><br/>
```

```
    En dehors de la fonction : <span id="nom2"></span>
```

```
  <script>
```

```
    uneFunction();
```

```
    document.getElementById("nom2").innerHTML = nom;
```

```
    function uneFunction()
```

```
    {
```

```
      var nom = "Martin";
```

```
      document.getElementById("nom1").innerHTML = nom;
```

```
    }
```

```
  </script>
```

```
</body>
```

```
</html>
```

Dans la fonction : Martin
En dehors de la fonction :

Erreur
« Nom is not
defined »

2) Variables globales

Les variables déclarées en-dehors de fonctions ont une portée globale, tous les scripts et fonctions d'une page web peuvent y accéder.

À ÉVITER !

```
var nom = "Martin";           // Variable globale
```

```
function uneFonction()  
{  
  
}
```

Exemple

```
<!DOCTYPE html>
```

```
<html>
```

```
  <body>
```

```
    Dans la fonction : <span id="nom1"></span><br/>
```

```
    En dehors de la fonction : <span id="nom2"></span>
```

```
  <script>
```

```
    var nom = "Martin";
```

```
    uneFunction();
```

```
    document.getElementById("nom2").innerHTML = nom;
```

```
    function uneFunction()
```

```
    {
```

```
      document.getElementById("nom1").innerHTML = nom;
```

```
    }
```

```
  </script>
```

```
  </body>
```

```
</html>
```

Dans la fonction : Martin

En dehors de la fonction : Martin



3) Variables de bloc

Les variables peuvent aussi être déclarées avec le mot clé **let**.

Il s'agit alors de variable de bloc, qui n'ont d'existence que dans le bloc (par exemple une condition, une boucle), alors que **var** déclare une variable globale ou locale dans toute une fonction.

```
function uneFonction()  
{  
  if(condition)  
  {  
    let var1 = 100 ;  
    var var2 = 200 ;  
  }  
  
  console.log(var1) ;           // erreur, var1 n'existe plus  
  console.log(var2) ;           // affiche 200, var2 existe encore  
}
```

Conditions

Les structures conditionnelles permettent d'effectuer différentes actions en fonction de conditions.

Condition : if

```
if (age < 18)
{
    texte = "Vous êtes trop jeune" ;
}
```

Opérateurs de comparaison

Opérateur	Description
==	Égal à
===	Même valeur et même type
!=	Différent de
!==	Valeur et type différents
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à

Opérateurs logiques

Les opérateurs logiques permettent de déterminer la logique entre des variables et des valeurs :

Opérateur	Description	Exemple
&&	et	(x>0 && x<400)
	ou	(age>=18 autorisation==true)
!	non	!(x == y)

Condition : if ... else

```
if (age < 18)
{
    texte = "Vous êtes mineur" ;
}
else
{
    texte = "Vous êtes majeur" ;
}
```

Condition : if ... else if

```
if (moyenne >= 16)
{
    mention = "Très bien" ;
}
else if (moyenne >= 14 && moyenne < 16)
{
    mention = "Bien" ;
}
else if (moyenne >= 12 && moyenne < 14)
{
    mention = "Assez Bien" ;
}
else
{
    mention = "" ;
}
```

Boucles

Les boucles permettent de répéter un bloc d'instructions.

JavaScript permet 3 types de boucles :

- **for** : répète un bloc de code un certain nombre de fois.
- **while** : répète un bloc de code tant qu'une condition est vraie, en évaluant la condition au début.
- **do ... while** : répète un bloc de code tant qu'une condition est vraie, en évaluant la condition à la fin.

1) Boucle for


Syntaxe :

```
for (Instruction 1 ; Instruction 2 ; Instruction 3)
{
    Bloc d'instructions à exécuter
}
```

- **Instruction 1** : exécuté avant le début de la boucle, on y met des initialisations.
- **Instruction 2** : définit la condition pour que la boucle soit exécutée.
- **Instruction 3** : exécuté à chaque fois après que le bloc d'instructions de la boucle ait été exécuté.

Exemple :

```
var texte = "" ;
for (i = 0; i < 5; i++) {
    texte += "Tour de boucle N°" + i + "<br>" ;
}
```



Tour de boucle N°0
Tour de boucle N°1
Tour de boucle N°2
Tour de boucle N°3
Tour de boucle N°4

2) Boucle while

Répète un bloc d'instructions tant qu'une condition est vraie.

Syntaxe :

```
while (condition)
{
    Bloc d'instructions à exécuter
}
```

Exemple :

```
var i = 3, decompte = "Mise à feu : ";
```

```
while (i >= 0)
{
    decompte += i + " ";
    i--;
}
```



Mise à feu : 3 2 1 0

3) Boucle do ... while


La boucle exécute un bloc d'instructions une fois, avant de vérifier que la condition est vraie, puis répète la boucle tant que la condition est vraie.

Syntaxe :

```
do
{
    Bloc d'instructions à exécuter
} while (condition) ;
```

Exemple :

```
var i = 0 ;
do
{
    texte += "Nombre : " + i;
    i++;
} while (i < 5);
```



```
Nombre : 0
Nombre : 1
Nombre : 2
Nombre : 3
Nombre : 4
```

Tableaux

http://www.w3schools.com/jsref/jsref_obj_array.asp

Les tableaux permettent de stocker plusieurs valeurs dans une variable :

```
var etudiants = ["Thomas", "Clément", "Quentin"];
```

Lire un élément du tableau :

```
var nom = etudiants[0]; // → "Thomas"
```

Modifier un élément du tableau :

```
etudiants[0] = "Lucas"; // Remplace "Thomas" par "Lucas"
```

Connaître le nombre d'éléments d'un tableau :

```
var nb = etudiants.length; // → 3
```


Ajouter un nouvel élément à la fin du tableau :

```
var etudiants = ["Thomas", "Clément", "Quentin"];  
etudiants.push("Julien");           // → "Thomas", "Clément", "Quentin", "Julien"
```

Ou bien :

```
etudiants[3] = "Julien" ;
```

Ajouter un nouvel élément au début du tableau (et décale les suivants) :

```
var etudiants = ["Thomas", "Clément", "Quentin"];  
etudiants.unshift("Anthony");      // → "Anthony", "Thomas", "Clément", "Quentin"
```

Supprimer le dernier élément du tableau :

```
var etudiants = ["Thomas", "Clément", "Quentin"];  
etudiants.pop () ;           // → "Thomas", "Clément"
```

Supprimer le premier élément du tableau (et décale les suivants) :

```
var etudiants = ["Thomas", "Clément", "Quentin"];  
etudiants.shift () ;         // → "Clément", "Quentin"
```

Ajouter et supprimer des éléments dans un tableau :

```
var fruits = ["Banane", "Orange", "Pomme", "Poire"];  
fruits.splice(2, 1, "Citron", "Kiwi");
```

Où les nouveaux éléments
doivent être ajoutés

Combien d'éléments
doivent être supprimés

Nouveaux éléments
à ajouter

// → "Banane", "Orange", "Citron", "Kiwi", "Poire"

Supprimer des éléments dans un tableau :

```
var fruits = ["Banane", "Orange", "Pomme", "Poire", "Citron", "Kiwi"];  
fruits.splice(1, 3);
```

// À partir de la position 1, supprime 3 éléments

// → "Banane", "Citron", "Kiwi"

Trier un tableau par ordre alphabétique :

```
var fruits = ["Citron", "Orange", "Banane", "Kiwi"];  
fruits.sort() ; // → "Banane", "Citron", "Kiwi", "Orange"
```

Renverser l'ordre des éléments d'un tableau :

```
var fruits = ["Citron", "Orange", "Banane", "Kiwi"];  
fruits.reverse() ; // → "Kiwi", "Banane", "Orange", "Citron"
```

Parcourir les éléments d'un tableau :

```
var fruits = ["Banane", "Orange", "Pomme", "Poire"];
```

```
var nb = fruits.length;
```

```
var texte = "<ul>";
```

```
var i ;
```

```
for (i = 0; i < nb; i++)
```

```
{
```

```
    texte += "<li>" + fruits[i] + "</li>";
```

```
}
```

```
texte += "</ul>";
```

Autre syntaxe de la boucle for :

```
var fruits = ["Banane", "Orange", "Pomme", "Poire"];
```

```
var texte = "<ul>";
```

```
var i ;
```

```
for (i in fruits)
```

```
{
```

```
    texte += "<li>" + fruits[i] + "</li>";
```

```
}
```

```
texte += "</ul>";
```

Exemple :

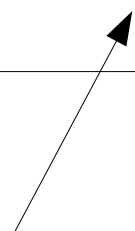
```
<!DOCTYPE html>
<html>
  <body>
    <p>Liste de fruits :</p>
    <p id="demo"></p>

    <script>
      var fruits = ["Banane", "Orange", "Pomme", "Poire"];
      var texte = "<ul>";

      for (i in fruits) {
        texte += "<li>" + fruits[i] + "</li>";
      }
      texte += "</ul>";
      document.getElementById("demo").innerHTML = texte;
    </script>

  </body>
</html>
```

Liste de fruits :

- Banane
 - Orange
 - Pomme
 - Poire
- 

Fonctions mathématiques

http://www.w3schools.com/js/js_math.asp

Nombres aléatoires

Fonction retournant un nombre réel aléatoire entre 0 et 1 (exclu) :

```
Math.random();
```

Fonction retournant un nombre entier aléatoire entre 0 et 99 :

```
Math.floor(Math.random() * 100);
```

Fonction retournant un nombre entier aléatoire entre min et max (inclus) :

```
function nombreAleatoire(min, max)
{
    return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

Les objets

Un objet a des propriétés et des méthodes.

Par exemple, pour représenter une personne :

```
var personne = {  
  prenom : "Sabine",  
  nom    : "Durand",  
  age    : 48,  
  nomComplet : function() {  
    return this.prenom + " " + this.nom;  
  }  
};
```

Propriétés

Méthode

Le mot clé **this** désigne l'objet lui-même.

On peut utiliser cet objet ainsi :

```
personne.nom = "Martin" ;  
document.getElementById("demo").innerHTML = personne.nomComplet() ;
```

Création de plusieurs objets

La syntaxe précédente ne permet de créer qu'un seul objet.
On veut parfois créer plusieurs objets du même type. Pour ce faire, on peut utiliser une fonction de construction d'objet et écrire des méthodes permettant de manipuler cet objet :

```
function Personne (prenom, nom, age) {  
    this.prenom = prenom;  
    this.nom = nom;  
    this.age = age;  
}
```

```
Personne.prototype.nomComplet = function()  
{  
    return this.prenom + " " + this.nom;  
}
```

```
var papa = new Personne("Jean", "Martin", 50);  
var maman = new Personne("Sabine", "Durand", 48);  
console.log(papa.nomComplet());
```

JSON

JSON= **J**ava**S**cript **O**bject **N**otation

C'est un format d'échange de données très léger. Sa syntaxe dérive de celle de la notation des objets de JavaScript :

- Les accolades { } définissent des objets

- Les données sont en paires nom / valeur

- Les données sont séparées par des virgules

- Les crochets [] définissent des tableaux

Exemple :

```
{"employees":  
  {"prenom":"John", "nom":"Doe"},  
  {"prenom":"Anna", "nom":"Smith"},  
  {"prenom":"Peter", "nom":"Jones"}  
}]
```

Remarque :

Bien qu'il soit très adapté au langage JavaScript dont il utilise la syntaxe, JSON est avant tout un simple format texte qui peut être utilisé avec d'autres langages de programmation.

http://www.w3schools.com/js/js_json_intro.asp

Utilisation de JSON avec JavaScript

Le format JSON est syntaxiquement identique au code que l'on écrit pour créer des objets JavaScript, ce qui rend direct l'accès à ces données avec la notation objet.

1) Cas où les données au format JSON sont stockées dans une chaîne de caractères :

```
<script>
```

```
var texte = '{"prenom": "Sébastien", "nom": "Thon", "qualite": "Mr"}';
```

```
var personne = JSON.parse(texte);  
console.log(personne.qualite) ;  
console.log(personne.prenom) ;  
console.log(personne.nom) ;
```

```
</script>
```

2) Cas où les données au format JSON sont stockées dans un fichier :

fichier.json :

```
{"prenom": "Sébastien", "nom": "Thon", "qualite": "Mr"}
```

code JavaScript :

```
function handler() {  
    if(this.status == 200 && this.readyState == 4 ) {  
        var json = this.response;  
        console.log(json.qualite, json.prenom, json.nom);  
    } else {  
        console.log("Erreur de lecture");  
    }  
}
```

```
var client = new XMLHttpRequest();  
client.onload = handler;  
client.responseType = 'json';  
client.open("GET", "fichier.json");  
client.send();
```

Comparaison entre JSON et XML

Pas besoin d'utiliser un parseur pour parcourir les données JSON comme c'est le cas par exemple avec le format XML, l'accès est direct avec la notation objet.

Le code JSON prend moins de place que son équivalent en XML.

N'ayant pas besoin d'être parsé et étant plus compact, un code JSON est plus rapide à charger.

XML

```
<employees>
  <employee>
    <prenom>John</prenom> <nom>Doe</nom>
  </employee>
  <employee>
    <prenom>Anna</prenom> <nom>Smith</nom>
  </employee>
  <employee>
    <prenom>Peter</prenom> <nom>Jones</nom>
  </employee>
</employees>
```

JSON

```
{"employees":[
  {"prenom":"John", "nom":"Doe"},
  {"prenom":"Anna", "nom":"Smith"},
  {"prenom":"Peter", "nom":"Jones"}
]}
```


Événements

Un événement HTML est quelque chose qui arrive à un élément HTML, du fait du navigateur ou du fait de l'utilisateur :

- Une page HTML a fini de se charger.
- La fenêtre a été redimensionnée.
- La souris a été déplacée.
- On a cliqué sur un bouton de la souris.
- On a appuyé ou relâché une touche du clavier.
- Le champ d'un formulaire a été changé.
- On a cliqué sur un élément HTML, par exemple un bouton.
- etc.

Une liste complète d'événements HTML :

http://www.w3schools.com/jsref/dom_obj_event.asp

On peut écrire du code JavaScript pour réagir à ces événements.

Événement de clic sur un élément html

```
<!DOCTYPE html>
```

```
<html>
```

```
  <body>
```

```
    <p>Cliquez sur le bouton pour afficher la date.</p>
```

```
    <button onclick="afficheDate()">Quelle heure est-il ?</button>
```

```
    <p id="demo"></p>
```

```
    <script>
```

```
      function afficheDate() {
```

```
        document.getElementById("demo").innerHTML = Date();
```

```
      }
```

```
    </script>
```

```
  </body>
```

```
</html>
```

Événement d'appui sur une touche du clavier

```
window.addEventListener("keydown", function (event) {  
    switch (event.key) {  
        case "a":  
            console.log("a");  
            break;  
        case "ArrowDown":  
            console.log("down");  
            break;  
        case "Escape":  
            console.log("esc");  
            break;  
        default:  
            return;  
    }  
    event.preventDefault(); // Pour que d'autres fonctions puissent traiter la touche  
}, true);
```

Liste de codes de touches :

<https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode>

L'API Canvas

<https://developer.mozilla.org/fr/docs/Web/HTML/Canvas>

Fonctions JavaScript permettant de tracer des éléments graphiques (des lignes, des rectangles, du texte, des images, etc.) dans une balise **<canvas>** d'un document HTML.

```
<!DOCTYPE html>
<html>
  <body>
    <canvas id="monCanvas" width="400" height="300"></canvas>

    <script>
      var canvas = document.getElementById('monCanvas');
      var ctx = canvas.getContext('2d');

      ctx.fillStyle = 'rgb(255,0,0)';
      ctx.fillRect(0, 0, canvas.width, canvas.height);
    </script>
  </body>
</html>
```

WebGL

<https://www.khronos.org/webgl/>



Depuis 2011, interface JavaScript permettant d'utiliser OpenGL dans une page web.

Des bibliothèques facilitent le développement en WebGL : Babylon.js, Blend4Web, GLGE, CopperLicht, C3DL, SceneJS, SpiderGL et OSGJS, Three.js, XTK, ...

three.js

<https://threejs.org>

Facilite l'utilisation de WebGL en apportant des fonctionnalités de haut niveau : représentation de la scène sous la forme d'un graphe, chargement d'objets, primitives graphiques, sources de lumière, matériaux, ombres portées, animation, gestion aisée de la réalité virtuelle stéréoscopique, ...

Utilisation :

Télécharger three.js ici :

<https://github.com/mrdoob/three.js/archive/master.zip>

Dans l'archive, prenez le fichier **three.js** qui se trouve dans le répertoire **build**, vous aurez ainsi accès aux fonctionnalités de base. D'autres fonctionnalités se trouvent dans les fichiers **.js** du répertoire **examples/js**

Exemple :

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta charset="utf-8" />
```

```
    <title>Premier programme WebGL</title>
```

```
  </head>
```

```
  <body>
```

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

```
    <script>
```

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

```
      scene.background = new THREE.Color("rgb(0, 0, 0)");
```

```
      var camera = new THREE.PerspectiveCamera( 45,  
                                                window.innerWidth/window.innerHeight, 0.1, 10000 );
```

```
      var renderer = new THREE.WebGLRenderer( {antialias: true} );  
      renderer.setSize( window.innerWidth, window.innerHeight );  
      document.body.appendChild( renderer.domElement );
```

*Placez le fichier **three.js** dans le même répertoire que votre fichier .html*

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

```
camera.position.set( 0, 0, 8 );
```

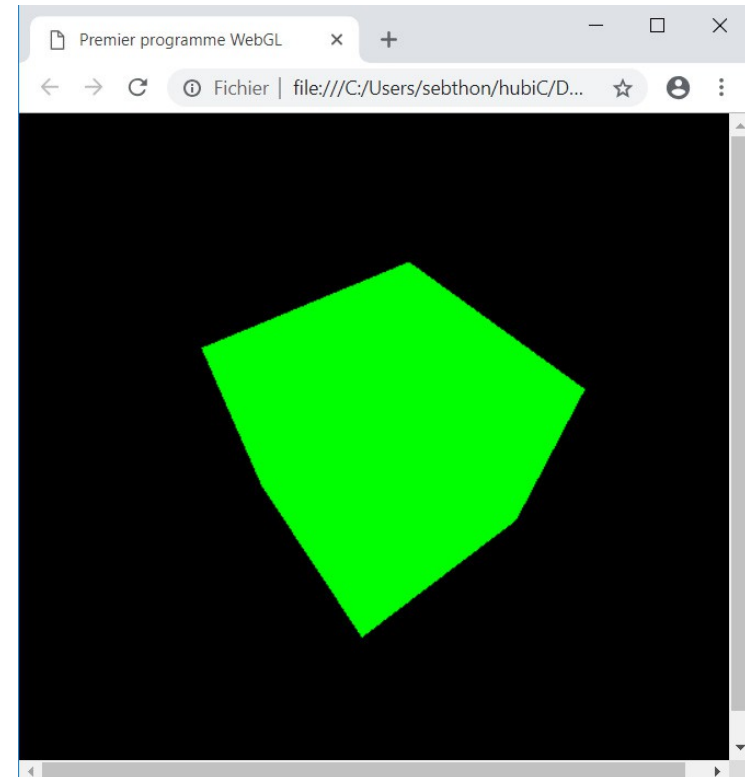
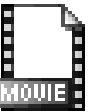
```
var animate = function () {  
    requestAnimationFrame( animate );  
  
    cube.rotation.x += 0.01;  
    cube.rotation.y += 0.01;  
  
    renderer.render( scene, camera );  
};
```

```
    animate();
```

```
</script>
```

```
</body>
```

```
</html>
```



Remarque :

```
var renderer = new THREE.WebGLRenderer( {antialias: true} );  
renderer.setSize( window.innerWidth, window.innerHeight );  
document.body.appendChild( renderer.domElement );
```

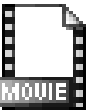
→ Ajoute au document HTML une balise **<canvas>** dans laquelle on peut dessiner (ici en 3D) et qui occupe toute la fenêtre.

On peut aussi placer explicitement une balise `<canvas>` dans le code HTML, dont on règle la taille, et la passer en paramètre au `renderer` lors de sa construction :

```
<canvas id="monCanvas" width="400" height="300"></canvas>
```

...

```
var renderer = new THREE.WebGLRenderer( {canvas: monCanvas,  
                                         antialias: true} );  
// document.body.appendChild( renderer.domElement );
```



Examples

<https://threejs.org/examples>

three.js / examples

three.js / examples

Type to filter

webgl

animation / cloth

animation / keyframes

animation / skinning / blending

animation / skinning / morph

camera

camera / array

camera / cinematic

camera / logarithmicdepthbuffer

clipping

clipping / advanced

clipping / intersection

decals

depth / texture

effects / anaglyph

effects / ascii

effects / parallaxbarrier

effects / peppersghost

effects / stereo

framebuffer / texture

geometries

geometries / parametric

geometry / colors

geometry / colors / lookuptable

geometry / convex

geometry / cube

geometry / dynamic

geometry / extrude / shapes

geometry / extrude / shapes2

geometry / extrude / splines

geometry / hierarchy

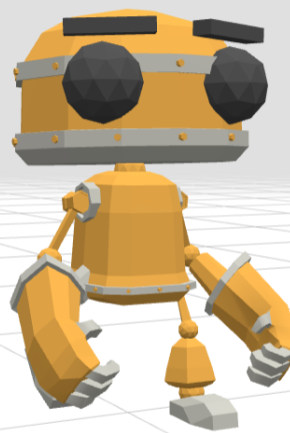
geometry / hierarchy2

60 FPS (9-60)

three.js - webgl - skinning and morphing

The animation system allows clips to be played individually, looped, or crossfaded with other clips. This example shows a character looping in one of several base animation states, then transitioning smoothly to one-time actions. Facial expressions are controlled independently with morph targets.

Model by [Tomás Lauh ](#), modifications by [Don McCurdy](#). CC0.



States

state

Walking

Emotes

Jump

Yes

No

Wave

Punch

ThumbsUp

Expressions

Angry

0

Surprised

0

Sad

0

Close Controls

View source

Géométrie

BoxGeometry

<https://threejs.org/docs/#api/en/geometries/BoxGeometry>

```
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 );
```

SphereGeometry

<https://threejs.org/docs/#api/en/geometries/SphereGeometry>

```
var geometry = new THREE.SphereGeometry( 5, 32, 32 );  
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );  
var sphere = new THREE.Mesh( geometry, material );  
scene.add( sphere );
```

CylinderGeometry

<https://threejs.org/docs/#api/en/geometries/CylinderGeometry>

```
var geometry = new THREE.CylinderGeometry( 5, 5, 20, 32 );  
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );  
var cylinder = new THREE.Mesh( geometry, material );  
scene.add( cylinder );
```

ConeGeometry

<https://threejs.org/docs/#api/en/geometries/ConeGeometry>

```
var geometry = new THREE.ConeGeometry( 5, 20, 32 );  
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );  
var cone = new THREE.Mesh( geometry, material );  
scene.add( cone );
```

PlaneGeometry

<https://threejs.org/docs/#api/en/geometries/PlaneGeometry>

```
var geometry = new THREE.PlaneGeometry( 20, 20, 32 );  
var material = new THREE.MeshBasicMaterial( {color: 0xffff00,  
                                              side: THREE.DoubleSide} );  
var plane = new THREE.Mesh( geometry, material );  
scene.add( plane );
```

TorusGeometry

<https://threejs.org/docs/#api/en/geometries/TorusGeometry>

```
var geometry = new THREE.TorusGeometry( 10, 3, 16, 100 );  
var material = new THREE.MeshBasicMaterial( { color: 0xffff00 } );  
var torus = new THREE.Mesh( geometry, material );  
scene.add( torus );
```

...

Transformations

Position

```
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 );  
cube.position.set(10,0,0);
```

Rotation

```
var geometry = new THREE.PlaneGeometry( 5, 20, 32 );  
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );  
var plane = new THREE.Mesh( geometry, material );  
scene.add( plane );  
plane.rotation.set(-Math.PI / 2, 0, 0);
```

Caméra

Projection perspective

<https://threejs.org/docs/#api/en/cameras/PerspectiveCamera>

```
var camera = new THREE.PerspectiveCamera( 45, width / height, 1, 1000 );  
scene.add( camera );
```

Projection orthographique

<https://threejs.org/docs/#api/en/cameras/OrthographicCamera>

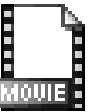
```
var camera = new THREE.OrthographicCamera( -width / 2, width / 2,  
                                            height / 2, -height / 2, 1, 1000 );  
scene.add( camera );
```

Controls : contrôle de la caméra

OrbitControls : permet de faire tourner la caméra autour d'une cible

<https://threejs.org/docs/#examples/controls/OrbitControls>

```
<script src="js/OrbitControls.js"></script>
```



```
var camera = new THREE.PerspectiveCamera( 45, window.innerWidth /  
                                         window.innerHeight, 1, 10000 );  
camera.position.set( 0, 20, 100 );
```

```
var controls = new THREE.OrbitControls( camera, renderer.domElement );
```

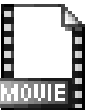
```
function animate() {  
    requestAnimationFrame( animate );  
    controls.update();  
    renderer.render( scene, camera );  
}
```

```
animate();
```


Paramètres optionnels de OrbitControls :

```
var controls = new THREE.OrbitControls( camera );
```

```
controls.enablePan = false;           // translation clic droit true/false  
controls.enableZoom = true;           // zoom actif true/false  
controls.zoomSpeed = 1.0;             // vitesse du zoom à la molette  
controls.enableDamping = true;        // inertie de rotation active true/false  
controls.dampingFactor = 0.2;         // facteur d'inertie  
controls.minDistance = 1;             // distance min  
controls.maxDistance = 100;          // distance max
```



Autres modes de contrôle de la caméra :

Fly

https://threejs.org/examples/#misc_controls_fly

Lookat

https://threejs.org/examples/#misc_lookat

Map

https://threejs.org/examples/#misc_controls_map

PointerLock

https://threejs.org/examples/#misc_controls_pointerlock

Trackball

https://threejs.org/examples/#misc_controls_trackball

Ajouter une source de lumière

Source ambiante :

<https://threejs.org/docs/#api/en/lights/AmbientLight>

```
var lumiere = new THREE.AmbientLight( 0x555555 );  
scene.add( lumiere );
```

Source ponctuelle :

<https://threejs.org/docs/#api/en/lights/PointLight>

```
var lumiere = new THREE.PointLight( 0xff0000,           // couleur  
                                     1,                  // intensité  
                                     10 );                // distance (0 : pas de limite)  
  
lumiere.position.set( 50, 50, 50 );  
scene.add( lumiere );
```

Source directionnelle :

<https://threejs.org/docs/#api/en/lights/DirectionalLight>

```
var lumiere = new THREE.DirectionalLight( 0xffffff );  
lumiere.position.set( 1, 1, 1 );  
scene.add( lumiere );
```

Source ambiante hémisphérique de ciel :

<https://threejs.org/docs/#api/en/lights/HemisphereLight>

```
var lumiere = new THREE.HemisphereLight(  
    0xddeeff,          // couleur du ciel  
    0x202020,          // couleur du sol  
    1                  // intensité  
);  
scene.add( lumiere );
```

Matériaux

MeshBasicMaterial

Matériau simple, couleur unique en tout point de l'objet, qui n'est pas affecté par les sources de lumière.

<https://threejs.org/docs/#api/en/materials/MeshBasicMaterial>

MeshLambertMaterial

Matériau diffus utilisant l'équation d'illumination de Lambert, sans reflet spéculaire.

<https://threejs.org/docs/#api/en/materials/MeshLambertMaterial>

MeshPhongMaterial

Matériau utilisant l'équation d'illumination de Phong (ambient + diffus + spéculaire).

<https://threejs.org/docs/#api/en/materials/MeshPhongMaterial>

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );  
var material = new THREE.MeshPhongMaterial( { color: 0xff0000,  
                                              specular: 0xffffff,  
                                              shininess: 40 } );  
var cube = new THREE.Mesh( geometry, material );  
scene.add( cube );
```

MeshStandardMaterial

<https://threejs.org/docs/#api/en/materials/MeshStandardMaterial>

MeshDepthMaterial

<https://threejs.org/docs/#api/en/materials/MeshDepthMaterial>

MeshNormalMaterial

<https://threejs.org/docs/#api/en/materials/MeshNormalMaterial>

MeshToonMaterial

<https://threejs.org/docs/#api/en/materials/MeshToonMaterial>

MeshPhysicalMaterial

<https://threejs.org/docs/#api/en/materials/MeshPhysicalMaterial>

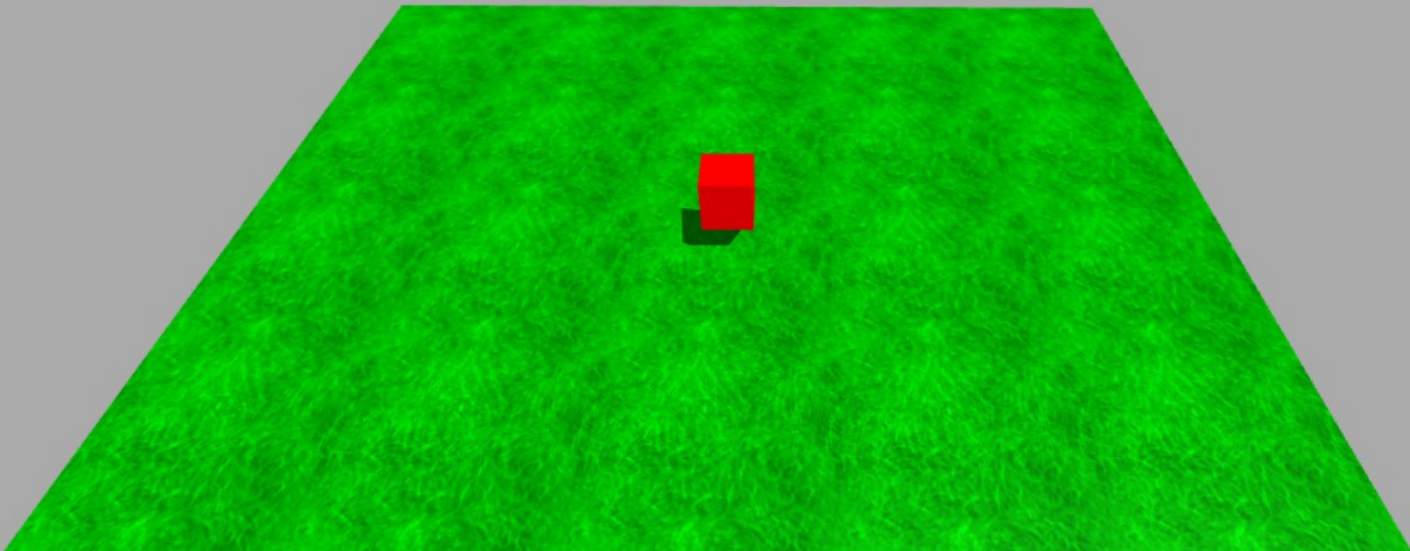
...

Texture

<https://threejs.org/docs/#api/en/loaders/TextureLoader>

<https://threejs.org/docs/#api/en/textures/Texture.id>

```
var texture = new THREE.TextureLoader().load( 'sol.png' );  
texture.wrapS = THREE.RepeatWrapping;  
texture.wrapT = THREE.RepeatWrapping;  
texture.repeat.set( 5, 5 );  
var geometry = new THREE.PlaneGeometry( 20, 20, 32 );  
var material = new THREE.MeshPhongMaterial( { color: 0x00ff00, map: texture } );  
var plane = new THREE.Mesh( geometry, material );  
scene.add( plane );
```



Charger un objet 3D

Three.js permet de charger des objets 3D à plusieurs formats : 3ds, collada, GLB, GLTF, obj, ...

Pour cela, il faut inclure le script correspondant au format souhaité. Vous les trouverez dans le répertoire `examples/js/loaders`

Note :

Si vous cherchez à charger un fichier d'objet en local (pas sur un serveur), cela ne fonctionnera pas pour des raisons de sécurité et vous aurez une erreur de CORS (Cross Origin Resource Sharing). Vous trouverez une solution ici, consistant à modifier les paramètres de sécurité de votre navigateur ou à installer un serveur local :

<https://threejs.org/docs/index.html#manual/en/introduction/How-to-run-things-locally>

Charger un objet au format GLB ou GLTF

Ajoutez le chargement du script **GLTFLoader.js** que vous trouverez dans **examples/js/loaders/** :

```
<script src="GLTFLoader.js"></script>
```

Et le code JavaScript suivant pour charger un fichier au format glb et l'ajouter à la scène :

```
var mon_objet;  
var loader = new THREE.GLTFLoader();  
  
loader.load( 'Flamingo.glb', function ( objet ) {  
    mon_objet = objet.scene;  
    scene.add( mon_objet );  
    console.log( "objet chargé" );  
}, undefined, function ( error ) {  
    console.error( error );  
} );
```

Charger un objet au format OBJ

Ajoutez le chargement du script **OBJLoader.js** que vous trouverez dans **examples/js/loaders/** :

```
<script src="OBJLoader.js"></script>
```

Et le code JavaScript suivant pour charger un fichier au format obj et l'ajouter à la scène :

```
var mon_objet;  
var loader = new THREE.OBJLoader();  
  
loader.load( 'cow.obj', function ( objet ) {  
    scene.add( objet );  
},  
function ( xhr ) {  
    console.log( ( xhr.loaded / xhr.total * 100 ) + '% loaded' );  
},  
function ( error ) {  
    console.error( 'Erreur de chargement' );  
} );
```

Les matériaux d'un objet au format OBJ sont décrits dans un fichier d'extension MTL portant le même nom. Pour le charger :

```
<script src="OBJLoader.js"></script>  
<script src="MTLLoader.js"></script>
```

```
var mtlLoader = new THREE.MTLLoader();  
mtlLoader.setPath( 'objets/vache/' );  
mtlLoader.load( "vache.mtl", function( materials ) {  
    materials.preload();  
  
    var objLoader = new THREE.OBJLoader();  
    objLoader.setMaterials( materials );  
    objLoader.setPath( 'objets/vache/' );  
    objLoader.load( 'vache.obj', function ( object ) {  
        scene.add( object );  
    },function ( xhr ) {  
        console.log( ( xhr.loaded / xhr.total * 100 ) + '% loaded' );  
    },  
    function ( error ) {  
        console.error( "Erreur de chargement" );  
    } );  
});
```

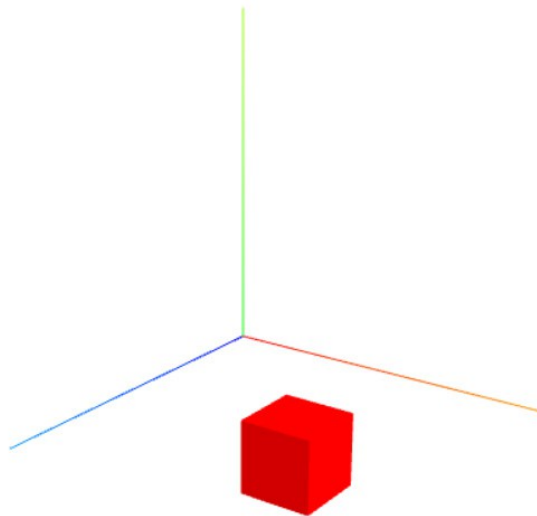
Helpers

Classes permettant d'afficher facilement des objets 3D servant d'aides dans la scène :

Axes

<https://threejs.org/docs/#api/en/helpers/AxesHelper>

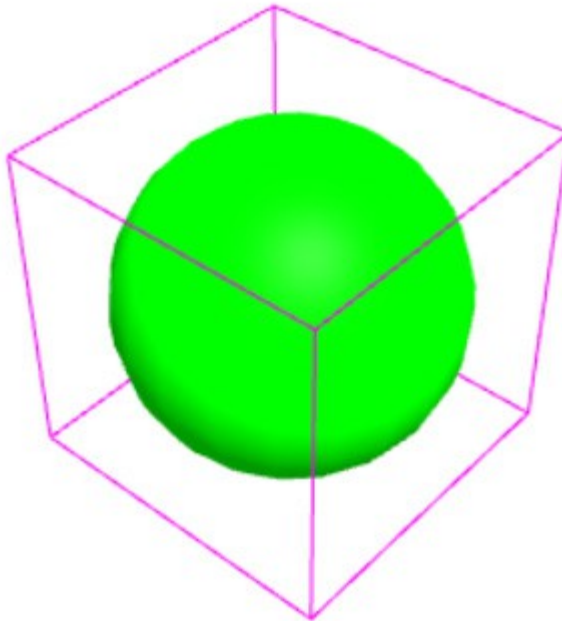
```
var axesHelper = new THREE.AxesHelper( 5 );  
scene.add( axesHelper );
```



Boîtes englobantes

<https://threejs.org/docs/#api/en/helpers/BoxHelper>

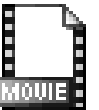
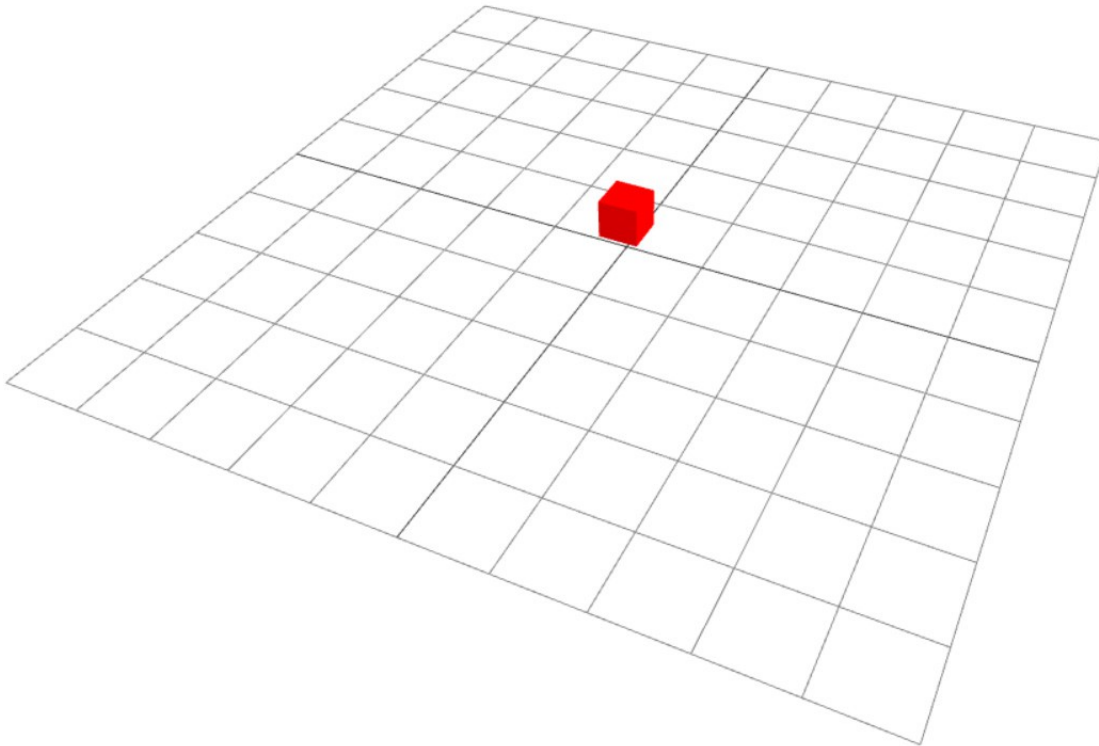
```
var sphere = new THREE.SphereGeometry();  
var objet_sphere = new THREE.Mesh( sphere,  
                                     new THREE.MeshPhongMaterial( { color: 0x00ff00 } ));  
scene.add( objet_sphere );  
  
var boite = new THREE.BoxHelper( objet_sphere, 0xff00ff );  
scene.add( boite );
```



Grille

<https://threejs.org/docs/#api/en/helpers/GridHelper>

```
var size = 20;  
var divisions = 10;  
var gridHelper = new THREE.GridHelper( size, divisions );  
scene.add( gridHelper );
```



Visibilité

Tous les objets (helpers, objets 3D, ...) de ThreeJS ont une propriété **visible** qui peut être mise à **true** ou **false** pour les afficher ou non.

Par exemple, pour masquer les axes et la grille définis ci-dessus :

```
axesHelper.visible = false ;  
gridHelper.visible = false ;
```

Pour les réafficher :

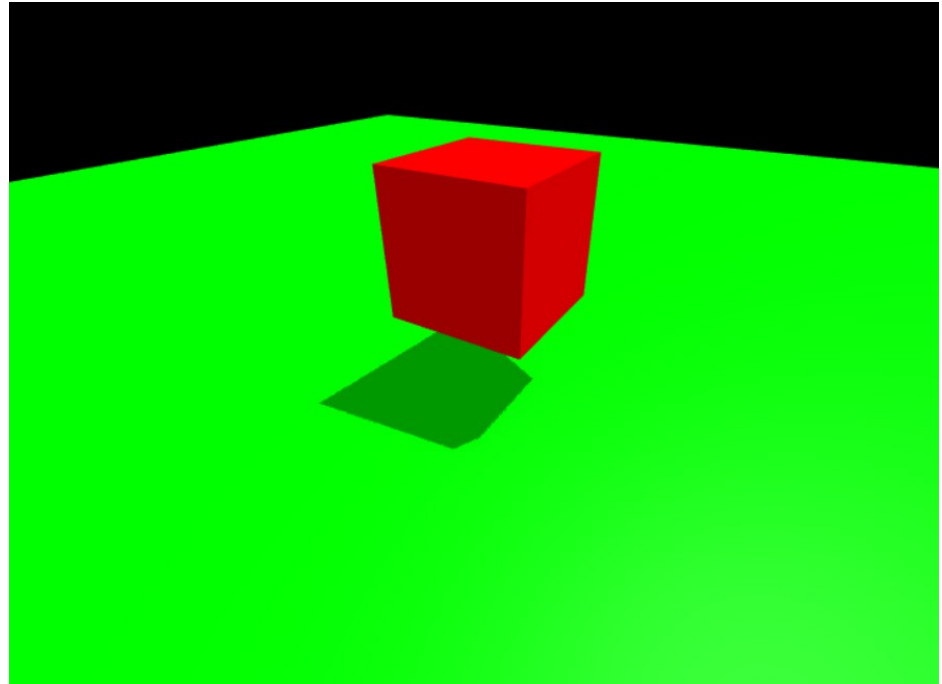
```
axesHelper.visible = true ;  
gridHelper.visible = true;
```


Ombres portées

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );  
var material = new THREE.MeshPhongMaterial( { color: 0xff0000 } );  
var cube = new THREE.Mesh( geometry, material );  
cube.position.set(0,1,0);  
cube.castShadow = true;  
cube.receiveShadow = true;  
scene.add( cube );
```

```
var geometry = new THREE.PlaneGeometry( 20, 20, 32 );  
var material = new THREE.MeshPhongMaterial( { color: 0x00ff00 } );  
var plane = new THREE.Mesh( geometry, material );  
plane.rotation.set(-Math.PI / 2,0,0);  
plane.castShadow = false;  
plane.receiveShadow = true;  
scene.add( plane );
```

```
var light = new THREE.DirectionalLight( 0xffffff );  
light.position.set( 0.5, 1, 0.25 );  
  
light.castShadow = true;  
light.shadow.width = 512;  
light.shadow.height = 512;  
light.shadow.camera.top = 2;  
light.shadow.camera.bottom = -2;  
light.shadow.camera.left = -2.5;  
light.shadow.camera.right = 2.5;  
light.shadow.camera.far = 5.75;  
light.shadow.bias = -0.025;  
  
scene.add( light );  
  
renderer.shadowMap.enabled = true;  
renderer.shadowMap.type = THREE.PCFShadowMap;
```



Brouillard

<https://threejs.org/docs/index.html#api/en/scenes/Fog>

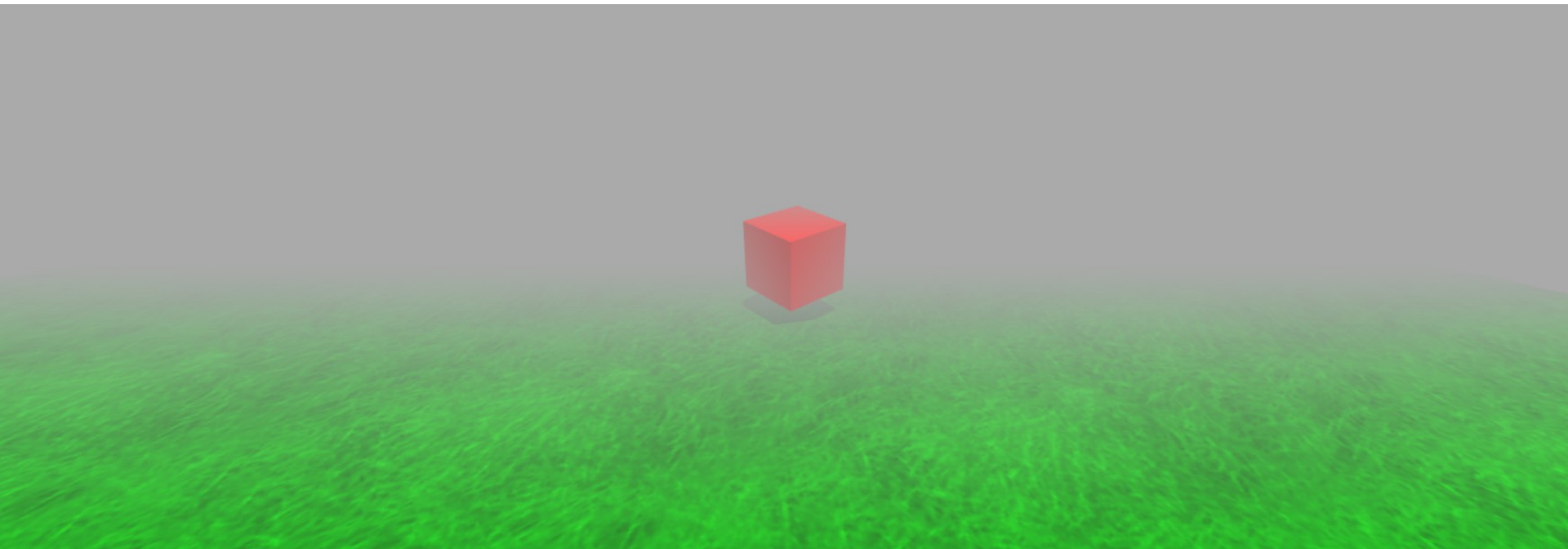
Atténuation des couleurs de la scène en fonction de celle d'un brouillard.

Constructeur :

Fog(color : Integer, near : Float, far : Float) ;

Exemple :

```
scene.fog = new THREE.Fog( 0xaaaaaa, 2, 10 );
```



Redimensionnement de la zone d'affichage 3D

Pour faire en sorte que la zone d'affichage 3D change de taille lorsqu'on modifie la taille de la fenêtre :

```
window.addEventListener( 'resize', onWindowResize, false );
```

```
...
```

```
function onWindowResize() {  
    camera.aspect = window.innerWidth / window.innerHeight;  
    camera.updateProjectionMatrix();  
    renderer.setSize( window.innerWidth, window.innerHeight );  
}
```

Un exemple complet qui pourra servir de base aux TP :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exemple ThreeJS</title>
  </head>

  <body>
    <script src="js/three.js"></script>
    <script src="js/controls/OrbitControls.js"></script>
    <script>
      function onWindowResize() {
        camera.aspect = window.innerWidth / window.innerHeight;
        camera.updateProjectionMatrix();
        renderer.setSize( window.innerWidth, window.innerHeight );
      }

      window.addEventListener( 'resize', onWindowResize, false );
```

```
var scene = new THREE.Scene();
scene.background = new THREE.Color(0xaaddff);

var camera = new THREE.PerspectiveCamera( 45,
                                         window.innerWidth/window.innerHeight, 0.1, 5000 );
camera.position.set( 0, 100, 100 );

var renderer = new THREE.WebGLRenderer( {antialias: true} );
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

var controls = new THREE.OrbitControls( camera, renderer.domElement );

var gridHelper = new THREE.GridHelper( 100, 10 );
scene.add( gridHelper );

var light1 = new THREE.DirectionalLight( 0xffffff );
light1.position.set( 1, 1, 1 );
scene.add( light1 );

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

```
var animate = function () {  
    requestAnimationFrame( animate );  
    controls.update();  
    renderer.render( scene, camera );  
};
```

```
    animate();
```

```
</script>
```

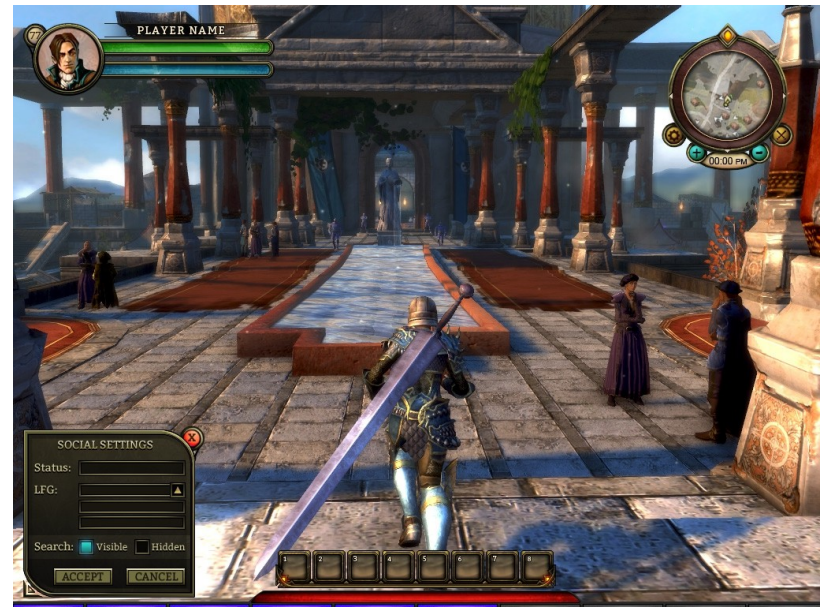
```
</body>
```

```
</html>
```

HUD

« Head Up Display » ou « Affichage tête haute ». Ce terme provient de l'aviation, où il désigne des informations qui sont affichées à un pilote de chasse, en superposition à sa vision.

Dans des applications 3D, ce terme désigne des informations (texte, éléments graphiques) qui sont affichés en 2D par dessus le visuel 3D. Cela peut servir à afficher un score, une barre d'énergie, un radar, le nombre de munitions, etc.



Avec ThreeJS, on peut faire un HUD en positionnant un élément HTML (par exemple un `<div>`) par dessus la balise `<canvas>` où s'affiche la 3D, et en y insérant les informations qu'on souhaite visualiser.



`<div>`

`<canvas>`

Par exemple :

```
<head>
  <style>
    body {
      margin : 0px;
      padding: 0px;
      overflow: hidden;
    }

    #info {
      font-family: arial;
      background-color : rgba(0,0,0,0.5);
      color: white;
      position: absolute;
      left : 10px;
      bottom: 10px;
      width : 200px;
      padding : 1em;
      z-index: 100;      // permet de faire en sorte que le <div> avec l'id #info
                        // apparaisse par dessus la balise <canvas> de WebGL
    }
  </style>
```

...

```
<body>
  <div id="info"></div>
```

```
...
```

```
<script>
```

```
...
```

```
var animate = function () {
  requestAnimationFrame( animate );
  controls.update();
```

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

```
  document.getElementById("info").innerHTML = "Triangles : "
                                              + renderer.info.render.triangles;
};
```

```
...
```

Animation physique

Utilisation de la librairie **ammo.js** (portage JavaScript de **Bullet Physics**) pour réaliser des animations physiques avec Three.js

<https://github.com/kripken/ammo.js/>

Animation de solides :

https://threejs.org/examples/#webgl_physics_terrain

Animation de tissus :

https://threejs.org/examples/#webgl_physics_cloth

Animation de cordes :

https://threejs.org/examples/#webgl_physics_rope

Animation d'objets déformables :

https://threejs.org/examples/#webgl_physics_volume

Fracture d'objets 3D :

https://threejs.org/examples/#webgl_physics_convex_break

Divers

Skybox

Animations d'objets 3D

Personnages

Postprocessing

Wireframe