

Cours 04 : WebGL

1. Qu'est-ce que WebGL?

WebGL (Web Graphics Library) est la nouvelle norme pour les graphismes 3D sur le Web, il est conçu dans le but de rendre les graphiques 2D et des graphiques 3D interactifs. Il est dérivé de la bibliothèque ES 2.0 de OpenGL qui est une API 3D de bas niveau pour les téléphones et autres appareils mobiles. WebGL fournit des fonctionnalités similaires de ES 2.0 (systèmes embarqués) et fonctionne bien sur le matériel graphique 3D moderne. Il est une API JavaScript qui peut être utilisé avec HTML5.

Le code WebGL est écrit dans la balise **<canvas>** du **HTML5**. Il est une spécification qui permet aux navigateurs Internet l'accès aux unités de traitement graphique (GPU) sur les ordinateurs où ils ont été utilisés.

2. Qui a développé WebGL

Un ingénieur de logiciel américain-serbe nommé **Vladimir Vukicevic** a fait le travail de la fondation et a dirigé la création de WebGL.

- En 2007, Vladimir a commencé à travailler sur un prototype **OpenGL** pour l'élément **Canvas** du document **HTML**.
- En Mars 2011, Kronos Group créé WebGL.

3. Navigateurs pris en charge

Les tableaux suivants présentent une liste des navigateurs qui prennent en charge WebGL

3.1 Navigateurs Internet

navigateur Nom	Version	Soutien
Internet Explorer	11 et ci-dessus	support complet
Google Chrome	3.9 et ci-dessus	support complet

Safari	8	support complet
Firefox	3.6 et ci-dessus	Prise en charge partielle
Opéra	2.7 et ci-dessus	Prise en charge partielle

3.2 Navigateurs mobiles

navigateur Nom	Version	Soutien
Chrome pour Android	4.2	Prise en charge partielle
navigateur Android	4.0	Prise en charge partielle
IOS Safari	8.3	support complet
Opera Mini	8	Ne supporte pas
Navigateur Blackberry	Dix	support complet
IE mobiles	Dix	Prise en charge partielle

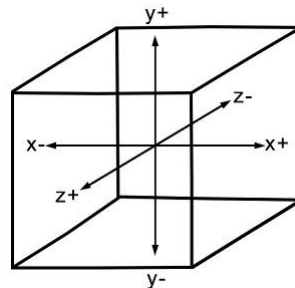
4. WebGL Notions de base :

4.1. WebGL - Système de coordonnées

Comme tout autre système 3D, vous aurez des axes x, y et z dans WebGL, où l'axe **z** signifie la **profondeur**. Les coordonnées WebGL sont limitées à (1, 1, 1) et (-1, -1, -1). Cela signifie - si l'on considère l'écran de projection graphiques WebGL comme un cube, puis un coin du cube sera (1, 1, 1) et le coin opposé sera (-1, -1, -1). WebGL n'affichera tout ce qui est établi au-delà de ces limites.

Le schéma suivant illustre les WebGL système de coordonnées. L'axe z signifie la profondeur. Une valeur positive de z indique que l'objet est près de l'écran / spectateur, tandis qu'une valeur négative de z indique que l'objet est loin de l'écran.

De même, une valeur positive de x indique que l'objet est sur le côté droit de l'écran et une valeur négative indique que l'objet se trouve sur le côté gauche. De même, les valeurs positives et négatives de y indiquent si l'objet est au sommet ou à la partie inférieure de l'écran.



4.2. Notions Fondamentales du WebGL

Après avoir obtenu le contexte WebGL de l'objet de la toile (canvas), vous pouvez commencer à dessiner des éléments graphiques en utilisant l'API WebGL en JavaScript.

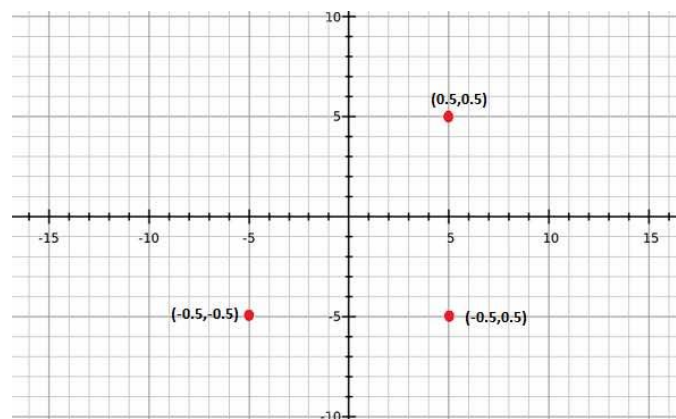
Voici quelques termes fondamentaux que vous devez savoir avant de commencer avec WebGL.

- Vertices

En général, pour dessiner des objets, comme un polygone, nous marquons les points sur le plan et les rejoinde pour former un polygone désiré. **Un sommet est un point qui définit la conjonction des bords d'un objet 3D.** Elle est représentée par trois valeurs à virgule flottante représentant chaque x , y , z Axes respectivement.

Exemple

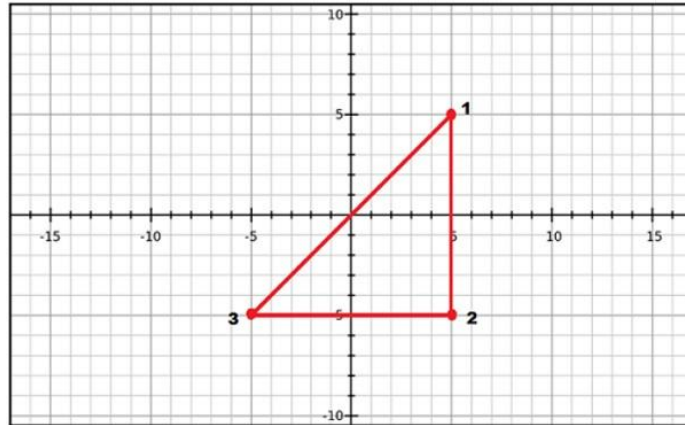
Dans l'exemple suivant, nous dessinons un triangle avec les sommets suivants - $(0,5, 0,5)$, $(-0,5, 0,5)$, $(-0,5, -0,5)$.



Remarque - Nous devons stocker ces sommets manuellement à l'aide des tableaux JavaScript et les transmettre au pipeline de rendu WebGL en utilisant un tampon de vertex.

-Indices

En WebGL, les valeurs numériques sont utilisées pour identifier les sommets. Ces valeurs numériques sont connues comme indices. Ces indices sont utilisés pour dessiner des maillages dans WebGL.



Remarque - *Tout comme les sommets, nous stockons les indices en utilisant les tableaux JavaScript et les transmettre au pipeline de rendu WebGL en utilisant un tampon d'index.*

-Arrays

Contrairement à OpenGL et JOGL, il n'y a pas de méthodes prédéfinies dans WebGL pour rendre les sommets directement. Nous devons les stocker manuellement à l'aide des tableaux JavaScript.

Exemple

```
var vertices = [ 0.5, 0.5, 0.1,-0.5, 0.5,-0.5]
```

-tampons

Les tampons sont des zones de mémoire de WebGL qui contiennent les données. Il existe différents tampons à savoir, tampon dessin, tampon de trame, tampon Vertex et tampon d'index. Le **tampon de sommet** et le **tampon d'index** sont utilisés pour **décrire et traiter la géométrie du modèle**.

- **Objets Vertex tampons** - Ce tampon stocke les données correspondant à chaque sommet (données par vertex)
- **Tampon Index des objets** : stocke les données sur les indices.
- **Le tampon de trame est une partie de mémoire graphique qui détiennent les données de scène**. Ce tampon contient des informations telles que la largeur et la hauteur de la surface (en pixels), la couleur de chaque pixel et de leur profondeur.

Après le stockage des sommets dans des tableaux, nous les passons au pipeline graphique WegGL en utilisant ces objets tampons.

-Les methodes WebGL

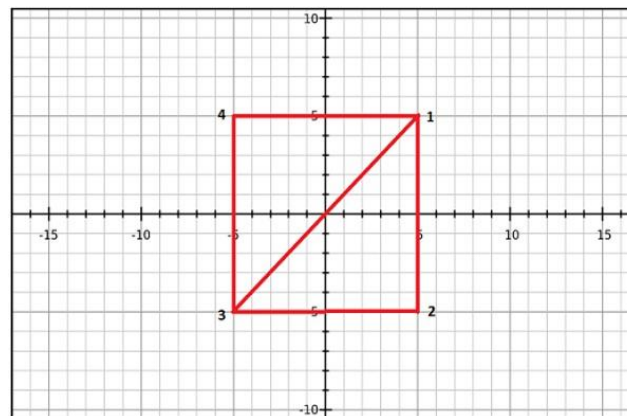
Pour dessiner des objets 2D ou 3D, l'API WebGL fournit deux méthodes, à savoir `drawArrays ()` et `drawElements ()`. Ces deux méthodes acceptent un paramètre en mode appelé à l'aide duquel vous pouvez sélectionner l'objet que vous voulez dessiner. Les possibilités offertes par ce domaine sont limitées à des points, des lignes et des triangles.

Pour dessiner un objet 3D à l'aide de ces deux méthodes, nous devons construire une ou plusieurs polygones primitifs à l'aide de points, lignes ou triangles. Par la suite, l'utilisation de ces polygones primitifs, nous pouvons former un maillage.

Un objet 3D dessiné en utilisant des polygones primitifs est appelé un **maillage**. WebGL offre plusieurs façons de dessiner des objets graphiques 3D, toutefois, les utilisateurs préfèrent normalement dessiner un maillage.

Exemple

Dans l'exemple suivant, vous pouvez constater que nous avons tracé un carré avec deux triangles $\rightarrow \{1, 2, 3\}$ et $\{4, 1, 3\}$.



4.3. WebGL Programme de Shader

WebGL fournit une solution pour réduire les frais généraux de communication. Comme il utilise ES SL (système embarqué Shader Language) qui fonctionne sur GPU, nous écrivons tous les programmes nécessaires pour tirer des éléments graphiques sur le système client à l'aide des **programmes de shader (les programmes que nous écrivons en utilisant OpenGL ES Shading Language / GLSL)**.

Ces shaders sont les programmes pour GPU et le langage utilisé pour écrire des programmes de shader est GLSL. Dans ces shaders, nous définissons exactement comment les sommets, les transformations, les matériaux, les lumières, et la caméra interagissent les uns avec les autres pour créer une image particulière.

-Vertex Shader

Shader Vertex est le code appelé chaque sommet du programme. Il est utilisé pour transformer (déplacer) la géométrie (ex: triangle) d'un endroit à un autre. Il gère les

données de chaque sommet (par-vertex données) tels que vertex coordonnées, normals, les couleurs et les coordonnées de texture.

Dans le code **ES GL de vertex shader**, les **programmeurs doivent définir des attributs pour gérer les données**. Ces attributs indiquent un objet **Vertex Buffer écrit en JavaScript**.

Les tâches suivantes peuvent être effectuées en utilisant les vertex shaders -

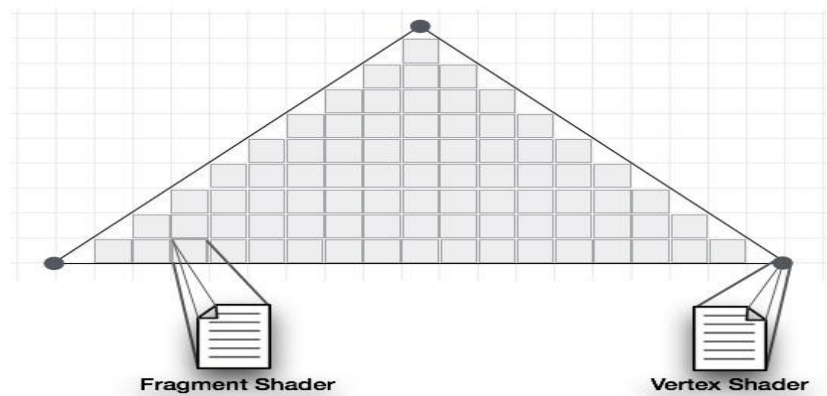
- transformation Vertex (`gl_position`)
- transformation normale et de la normalisation
- Texture coordonner génération
- Texture transformation de coordonnées
- Éclairage
- Application d'un matériau de couleur

-Fragment Shader (Pixel Shader)

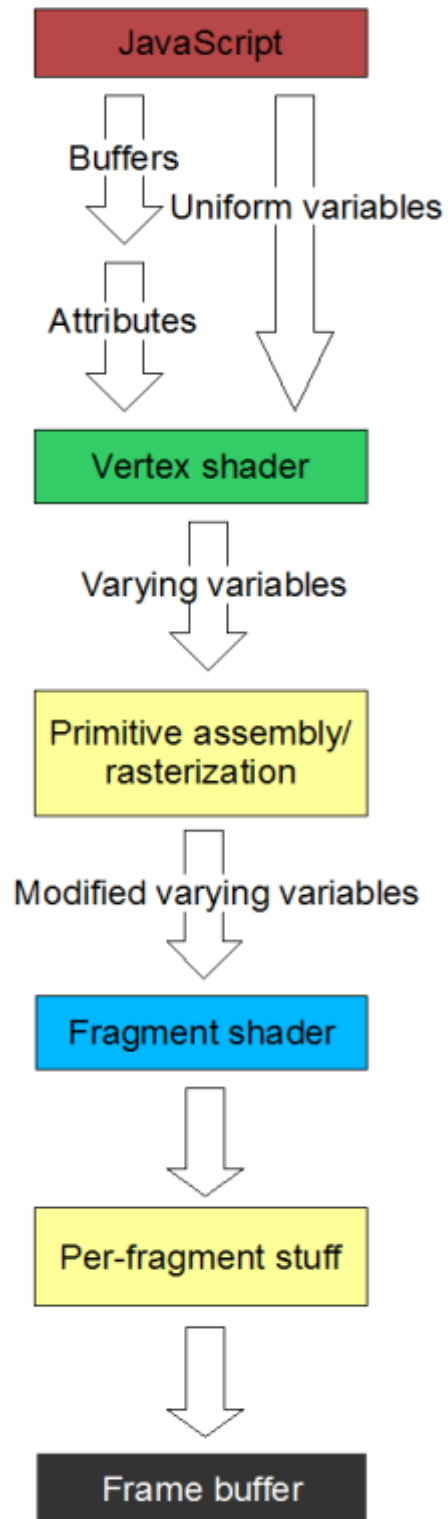
Une maille est formée par plusieurs triangles, et la surface de chacun des triangles est connu en tant que **fragment**. Shader Fragment est le code qui fonctionne sur tous les pixels de chaque fragment. Il est écrit pour calculer et remplir la couleur sur les *pixels individuels*.

Les tâches suivantes peuvent être effectuées en utilisant des shaders Fragment -

- Les opérations sur les valeurs interpolées
- accès Texture
- application Texture
- Brouillard
- la somme des couleurs(`gl_fragColor`)



5. Le pipeline de rendu :



Le diagramme montre, de façon très simplifiée, comment les données transmises aux fonctions JavaScript dans `drawScene` sont transformées en pixels affichés dans le canvas WebGL sur l'écran.

Au plus haut niveau, le processus fonctionne comme ceci : chaque fois que vous appelez une fonction comme `drawArrays`, WebGL traite les données que vous avez

précédemment spécifié sous la forme d'attributs (comme les tampons utilisés pour les vertex dans le TP) et les variables uniformes (utilisées pour les matrices de projection et modèle-vue), et les transmet au vertex shader.

Il fait ceci en appelant le vertex shader une fois pour chaque vertex, chaque fois avec les attributs correctement définis pour le vertex ; les variables uniformes sont également passées, mais comme l'indique leur nom, elles ne changent pas d'un appel à l'autre. Le vertex shader traite ces données, il a appliqué les matrices de projection et modèle-vue de sorte que les vertex soient tous en perspective et déplacés en fonction de l'état actuel de la matrice modèle-vue — et met ses résultats dans ce que l'on appelle des variables *varying*. Il peut fournir en sortie un certain nombre de variables *varying* ; en particulier une donnée est obligatoire : `gl_Position`, qui contient les coordonnées du vertex une fois que le shader a terminé de travailler dessus.

Une fois le vertex shader terminé, WebGL effectue le nécessaire pour convertir l'image 3D à partir de ces variables *varying* vers une image 2D, puis appelle le fragment shader une fois pour chaque pixel de l'image. Bien sûr, cela signifie qu'il appelle le fragment shader pour les pixels qui n'ont pas de vertex — autrement dit ceux situés entre les pixels sur lesquels les vertex se terminent. Pour ces derniers, il remplit des points dans les positions entre les vertex par un processus appelé interpolation linéaire — pour les positions des vertex qui composent notre triangle, ce processus « comble » l'espace délimité par les vertex avec des points pour former un triangle visible. Le but du fragment shader est de retourner la couleur pour chacun de ces points interpolés, et il le fait dans une variable *varying* nommée `gl_FragColor`.

Une fois le fragment shader terminé, ses résultats sont encore modifiés avec un peu de WebGL et ils sont mis dans le frame buffer, qui est finalement affiché sur l'écran.

