

The slide features a red **WebGL** logo at the top left. In the center, there is a white rectangular box containing the word **WEBGL** in blue capital letters. Below this, a blue horizontal bar contains the text **SANDRINE LANQUETIN - BUREAU G208**. At the bottom, a purple link [sandrine.lanquetin@u-bourgogne.fr](mailto:sandrine.lanquetin@u-bourgogne.fr) is displayed.

1

The slide has a title **DÉCOUPAGE DU MODULE** at the top. It contains two blue rectangular buttons. The first button on the left shows the **Blender** logo and the text **Blender(C. Gentil)**. The second button on the right shows the **WebGL** logo and the text **Introduction au WebGL(S. Lanquetin)**.

2

## MODALITÉS

- Contrôle continu
  - TP Noté

3

## WEBGL



Consortium industriel qui a proposé ce standard ouvert



Open Graphic Library : interface de programmation (API) implémentée au niveau des pilotes graphique qui permet d'utiliser l'accélération matérielle



OPENGL for Embedded System : version allégée et destinée aux matériels mobiles et embarqués



Web Graphic Library : Interface de programmation d'OpenGL ES2.0 en JavaScript

4

## POURQUOI WEBGL



Flash Molehill intègre accélération matérielle 2D et 3D => jeux vidéo pour le web

- Plug-in présent sur la plupart des ordinateurs
- Forte communauté de développeurs
- La plupart des appareils mobiles ne sont pas compatibles
- Flash nécessite beaucoup de mémoire et de ressources



Framework de développement Microsoft

- Basé sur .net, destiné au web, plus léger et performant que Flash
- Très peu utilisé hors sites Microsoft ou de vidéo HD
- Communauté peu importante, plus de nouvelle version



Unity 3D outil de développement destiné aux jeux vidéos.

- Plugin gratuit pour porter les jeux sur le web
- Compatible avec la plupart des navigateurs majeurs
- Performances de bonne qualité
- Technologie propriétaire plus difficile à interfaçer avec du contenu web

5

## POURQUOI WEBGL



Technologie libre

Basé sur JavaScript

Implémenté par tous les navigateurs

Rendu de scène 3D : pilote graphique

Intégré dans la page web via <canvas>

6

## COMPATIBILITÉ

- Dépend du navigateur et de la version du pilote graphique => tester votre app avec un max de config
- Navigateurs compatibles :
  - Firefox v4 et +,
  - Chrome v7 et +
  - Safari v5.1 et + (pas activé par défaut pour cette version)
  - Opera v12 et +
  - IE v11 et +
- Pilotes graphiques :
  - Actuellement, n'importe quel ordinateur
  - Avant 2010 : mise à jour pilote graphique peut-être nécessaire
- Terminaux mobiles
  - Compatible avec iOS depuis iOS8 release
  - Avec Android ou RIM en utilisant Firefox ou Opera. Pour des app3D fluide, adapter la taille des textures et programmation propre (mémoire)

7

## BONNES PRATIQUES JAVASCRIPT

- Éviter d'utiliser du code en tant que chaîne de caractères
  - Éviter d'utiliser la fonction eval( )
  - Remplacer les chaînes de caractères par du code direct :

```
setTimeout("ma_fonction()", 10000);
```

```
setTimeout(ma_fonction, 10000);
```

10

## BONNES PRATIQUES JAVASCRIPT

- Tableaux et objets

- Éviter de tester un attribut s'il est absent
- Indexer les tableaux avec des entiers car sinon considéré comme un objet : manipulation moins efficace
- Déclarer la taille des tableaux lors de leur déclaration

```
var mon_tableau=Array(10);
```

- For in moins rapide qu'une boucle for

~~for(var i in mon\_tableau) {...}~~

for(var i=0; i<mon\_tableau.length; i++) {...}

11

## BONNES PRATIQUES JAVASCRIPT

- Éviter les variables globales
- Éviter les conversions implicites
  - 0, null ou « » peuvent être convertis en False
  - Égalité des types : === au lieu de ==
- Éviter la structure try-catch-finally et les console.log
- Utiliser les fonctions natives de JS

12

## HERITAGE JS

```

class Polygone {
    constructor(_nbCotes) {
        this.nom = 'Polygone';
        this.nbCotes = _nbCotes;
    }
    forme(){
        console.log('C\'est un ' + this.nom + ' à ' + this.nbCotes + ' cotés.');
    }
}
C'est un Polygone à 5 cotés.
var poly1 = new Polygone(5);
poly1.forme();

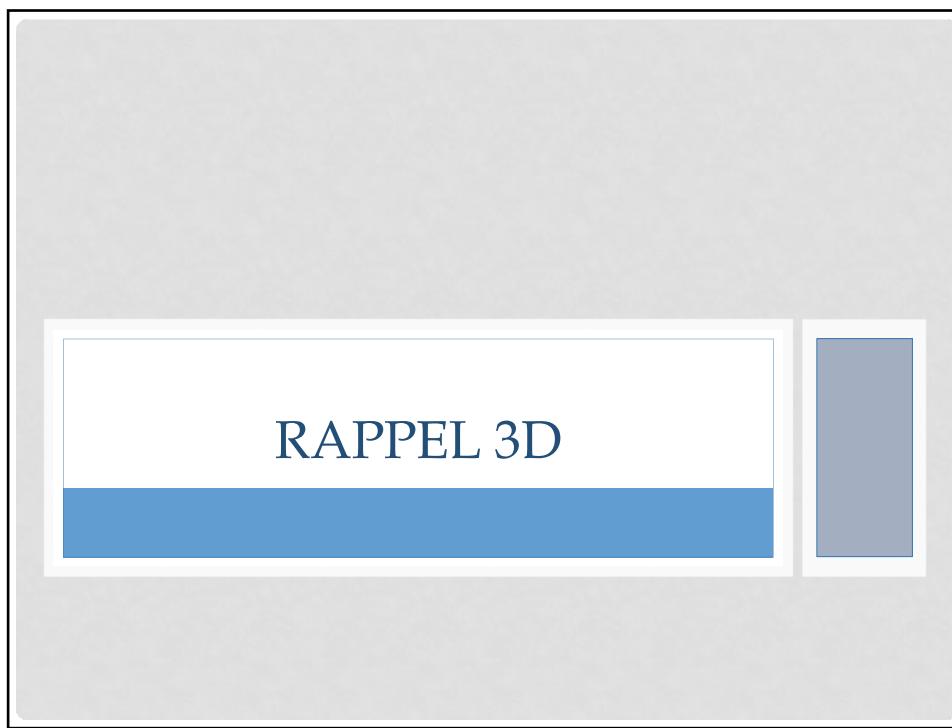
class Quad extends Polygone {
    constructor() {
        super(4);
        this.nom = "Quadrilatère"
    }
}
C'est un Quadrilatère à 4 cotés.
var poly2 = new Quad();
poly2.forme();

class Carré extends Rectangle {
    constructor(largeur) {
        super(_largeur, _largeur);
        this.nom = 'Carré';
    }
}
C'est un Carré à 4 cotés.
Aire = 4
var poly4 = new Carré(2);
poly4.forme();

class Rectangle extends Quad {
    constructor(_largeur,_longueur) {
        super(); this.nom = "Rectangle";
        this.largeur = _largeur; this.longueur = _longueur;
    }
    get aire() { return this.largeur * this.longueur; }
    set aire(_valeur) { this.aire = _valeur; }
    forme(){
        super.forme();
        console.log('Aire = ' + this.aire);
    }
}
C'est un Rectangle à 4 cotés.
Aire = 6
var poly3 = new Rectangle(2,3);
poly3.forme();

```

13



17

## MODÉLISATION

- **Fil de fer**

Objet représenté par ses arêtes

- **Surfacique**

Objet représenté par la surface qui le limite

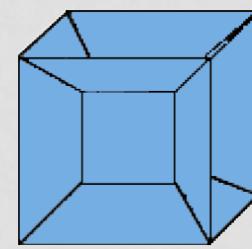
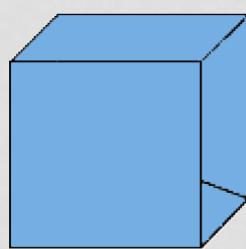
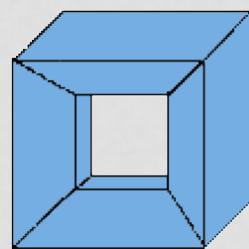
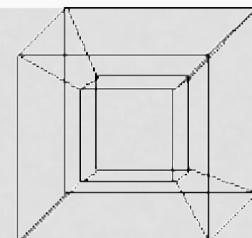
- **Volumique**

Objet représenté par le volume qu'il occupe

18

## FIL DE FER

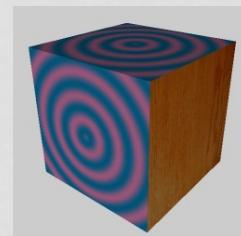
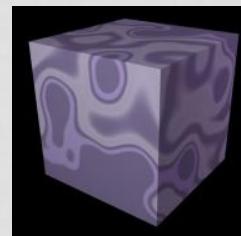
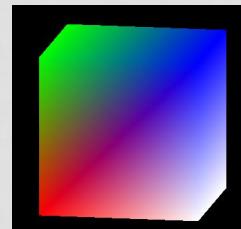
- Représentation ambiguë



19

## HABILLAGE

- Couleur
- Texture



20

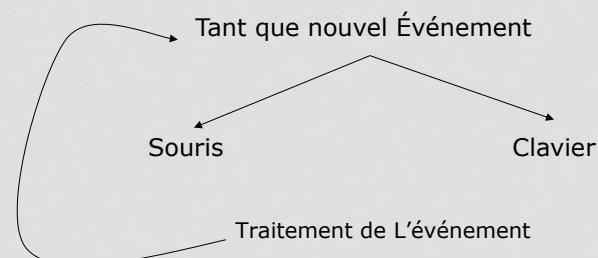
## VISUALISATION

- Définir et positionner les lumières et la caméra.
- **Lumière** : Plusieurs sources et différents types.
- **Caméra** : Projection de la scène sur l'écran : surface à deux dimensions.

21

## INTERACTION UTILISATEUR

•Événement



22

## NORME D'UN VECTEUR

$$\vec{u} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \|\vec{u}\| = \sqrt{x^2 + y^2 + z^2}$$

23

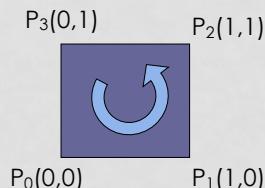
## FACETTES

- Objets simple : primitives
- Objets complexes : facettes

♦ Points : liste des coordonnées

♦ Faces : liste des indices des points

**Ex :**



<b>Points</b>	<b>Faces</b>
0 0	0 1 2 3
1 0	
1 1	
0 1	

24

## FICHIERS DE DONNÉES D'UN CUBE

```
#VRML V2.0 utf8
DEF cube Transform { translation 0.9843 0 0.4215 children [
  Shape {
    appearance Appearance {
      material Material {
        diffuseColor 0.03137 0.4314 0.5255
      }
      geometry DEF cube-FACES
    }
    IndexedFaceSet { ccw TRUE solid TRUE convex TRUE coord
      DEF cube-COORD
      Coordinate { point [
        -1.0 -1.0 -1.0,
        -1.0 -1.0 1.0,
        1.0 -1.0 1.0,
        1.0 -1.0 -1.0,
        -1.0 1.0 1.0,
        -1.0 1.0 -1.0,
        1.0 1.0 1.0,
        1.0 1.0 -1.0
      ]
      coordIndex [
        0, 3, 2, 1, -1,
        1, 2, 6, 5, -1,
        2, 3, 7, 6, -1,
        0, 4, 7, 3, -1,
        0, 1, 5, 4, -1,
        4, 5, 6, 7, -1
      ]
    }
  }
]
```



cube.obj

20/11/2020

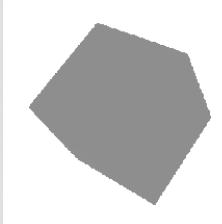


cube.wrl

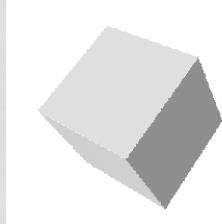
25

25

## LES NORMALES



- Sans normales -



- Avec normales -

$\vec{n} = \overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_3}$

$$\vec{n} = \vec{N} / \|\vec{N}\|$$

$P_3$        $\vec{n} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$        $P_2$   
 $P_0$        $P_1$

26

## ROTATION AUTOUR DE L'AXE X

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axe x  
non modifié

Ex : rotation d'angle  $\pi/2$   
change  $y$  en  $z$ , et  $z$  en  $-y$

$$R_x(\frac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

27

## ROTATION AUTOUR DE L'AXE Y

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axe y  
non modifié

Ex : rotation d'angle  $\pi/2$   
change  $z$  en  $x$ , et  $x$  en  $-z$

$$R_y\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

28

## ROTATION AUTOUR DE L'AXE Z

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axe z  
non modifié

Ex : rotation d'angle  $\pi/2$   
change  $x$  en  $y$ , et  $y$  en  $-x$

$$R_z\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

29

## OPENGL : MATRICES

► Sélection de la matrice active :

```
glMatrixMode(...)
```

GL\_MODELVIEW      GL\_PROJECTION      GL\_TEXTURE

```
var mvMatrix = mat4.create(); var pMatrix = mat4.create();
```

► Initialisation de la matrice active :

```
glLoadIdentity()      mat4.identity(mvMatrix);
```

► Modification de la matrice active :

```
glLoadMatrix(m), glMultMatrix(p)
```

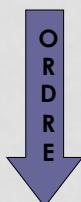
```
mat4.set (mvMatrix, m)
```

```
mat4.multiply (mvMatrix, mvMatrix, p)
```

30

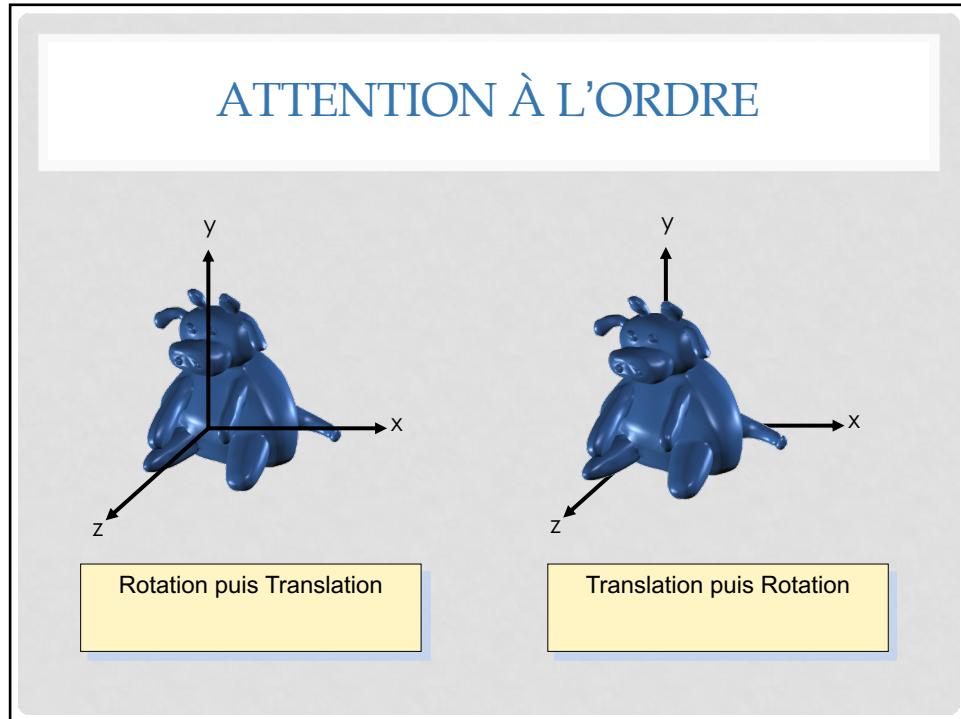
## TOUTES LES TRANSFORMATIONS 3D

- Transformation 3D  
= combinaison de translations, rotations,  
changement d'échelle

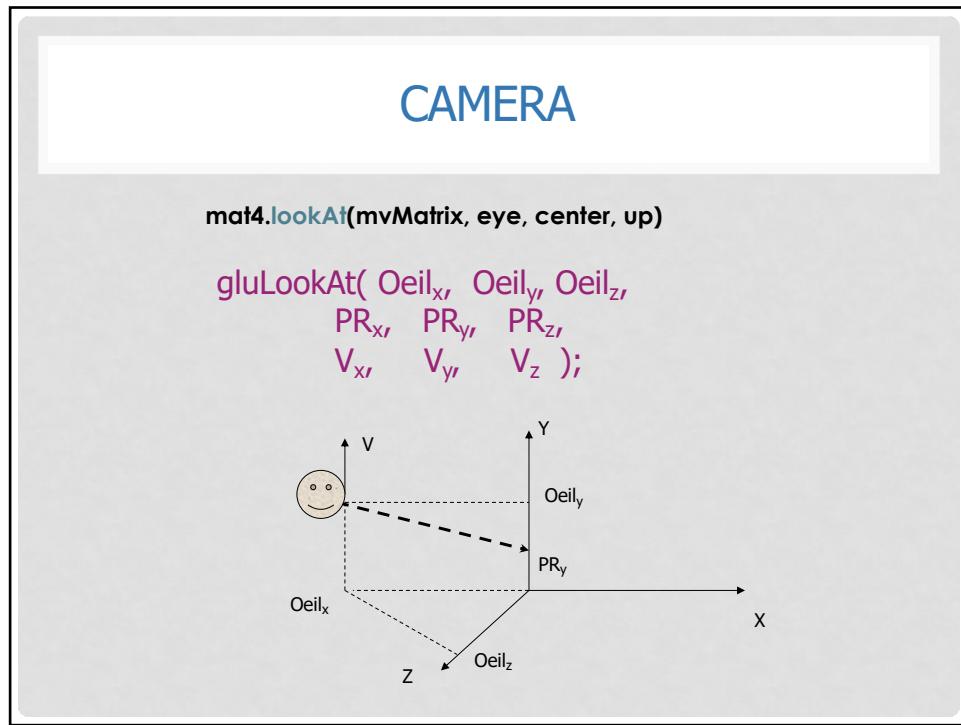


```
glTranslatef(x, y, z);
mat4.translate(mvMatrix, mvMatrix, [x, y, z]);
glRotatef(angle, x, y, z);
mat4.rotate(mvMatrix, mvMatrix, toRadian(angle), [x, y, z]);
glScalef(x, y, z);
mat4.scale(mvMatrix, mvMatrix, [x, y, z]);
```

31



32



33

## TRANSFORMATIONS DE PROJECTION : PARALLÈLE

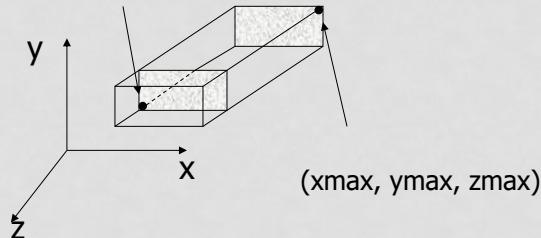
- Matrices de projection : pMatrix
- Objectif : définir un volume de vision

→ Projection orthogonale

`glOrtho(gauche, droit,bas,haut,proche,eloigne);`

`mat4.ortho(pMatrix, gauche, droite, bas, haut, proche, eloigne);`

(xmin, ymin, zmin)

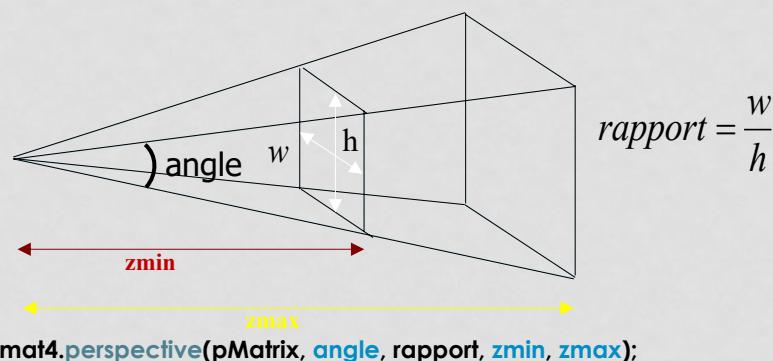


34

## VISUALISATION SOUS OPENGL

→ Projection en perspective :

`gluPerspective(angle,rapport, zmin, zmax);`



35

## GLMATRIX

- <http://glmatrix.net/>
  - crée une matrice 4\*4 identité : `mat4.create();`
  - Matrice identité : `mat4.identity(mvMatrix);`
  - set  

$$\begin{matrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 0 \end{matrix}$$

$$A = [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, x, y, z, 0]$$
`mat4.set(mvMatrix, A);`
  - Ortho `mat4.ortho(pMatrix, gauche, droite, bas, haut, proche, eloigne);`
  - Perspective `mat4.perspective(pMatrix, fovy, aspect, near, far);`  
`mat4.perspective(pMatrix, angle, rapport, zmin, zmax);`

36

## GLMATRIX

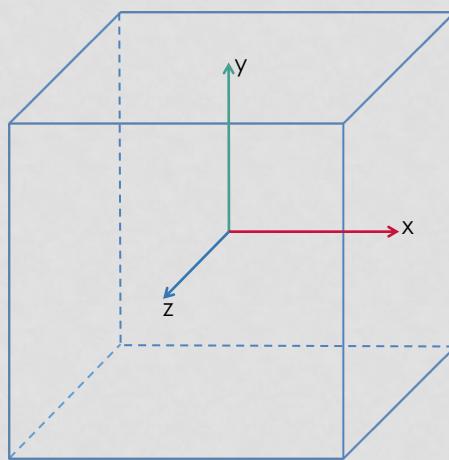
- Translation `mat4.translate(mvMatrix, mvMatrix, [x, y, z]);`
- Mise à l'échelle `mat4.scale(mvMatrix, mvMatrix, [x, y, z]);`
- Rotation :
  - rotate, `mat4.rotate(mvMatrix, mvMatrix, toRadian(angle), [x, y, z]);`
  - rotateX, `mat4.rotateX(mvMatrix, mvMatrix, toRadian(angle));`
  - rotateY, `mat4.rotateY(mvMatrix, mvMatrix, angleR);`
  - rotateZ `mat4.rotateZ(mvMatrix, mvMatrix, toRadian(angle));`
- caméra `mat4.lookAt(mvMatrix, eye, center, up)`

37

## BASES WEBGL

38

## SYSTÈME DE COORDONNÉES



$x \in [-1,1]$   
 $y \in [-1,1]$   
 $z \in [-1,1]$

39

## VOCABULAIRE

- **Sommets**(vertices / vertex): x,y,z  
On utilise des tableaux JS pour stocker les coordonnées des sommets et les transmettre au pipeline de rendu WebGL grâce aux vertex buffer
- **Indices** : valeurs pour identifier les sommets
- **Buffers** : régions de mémoire de WebGL contenant les données sur les objets à dessiner
  - Description de la géométrie : [vertex buffer](#) et [index buffer](#)
  - Représentation des pixels (nombre, couleur) : [frame buffer](#)

40

## VOCABULAIRE

- **Maillage** : objet 3D dessiné à l'aide de polygones
  - Dans WebGL, il y a deux méthodes `drawArrays()` et `drawElements()`.
  - Le paramètre mode permet de choisir ce que l'on veut dessiner : `points`, `lines` ou `triangles`
- **Matrices de transformation** :
  - Matrice de vue
  - Matrice de projection

41

## VOCABULAIRE

The diagram shows a large triangle divided into a grid of smaller triangles. The top vertex is labeled "Vertex shader". The bottom edge is labeled "Fragment shader". The interior of the triangle is filled with a grid of blue squares, labeled "Fragment".

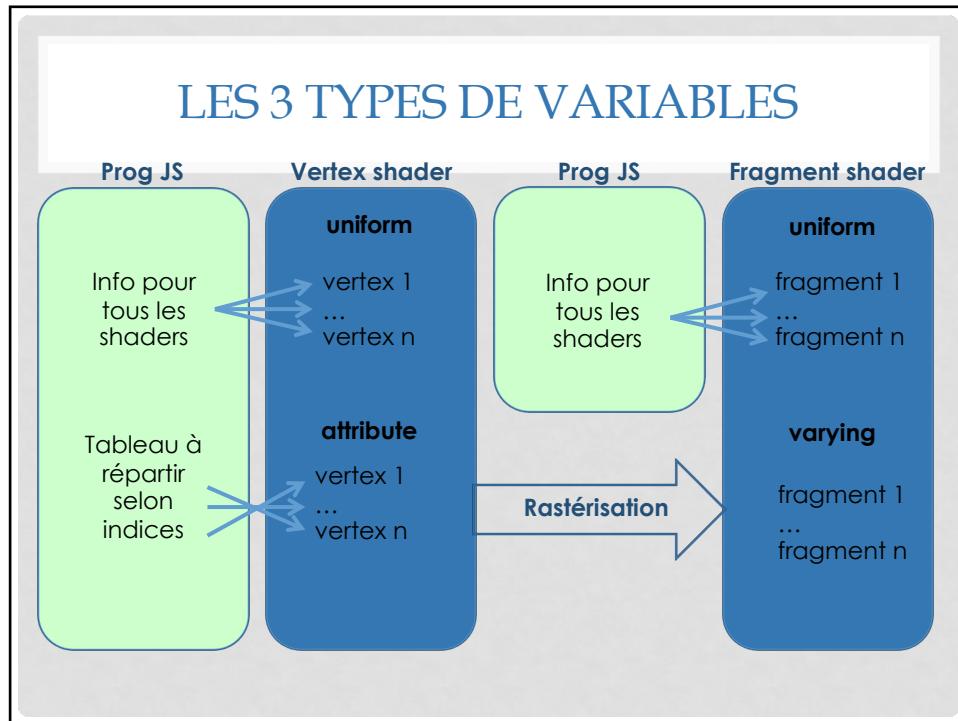
- **Shaders (GLSL ES)** : langage pour donner des instruction de rendu à la carte graphique (éclairage, couleurs, ...)
- **Fragment shader** : la surface couverte par un triangle est appelée fragment. Code appliqué à tous les pixels contenu dans un fragment (couleur de remplissage du pixel) **gère le rendu de chaque pixel**
- **Vertex shader** : code appelé pour chaque sommet, il gère les données de chaque sommet (coordonnées, normales, couleur et textures). Les attributs sont définis dans le code, ils pointent sur un Vertex Buffer Object codé en JS. **gère la géométrie 2D ou 3D**

42

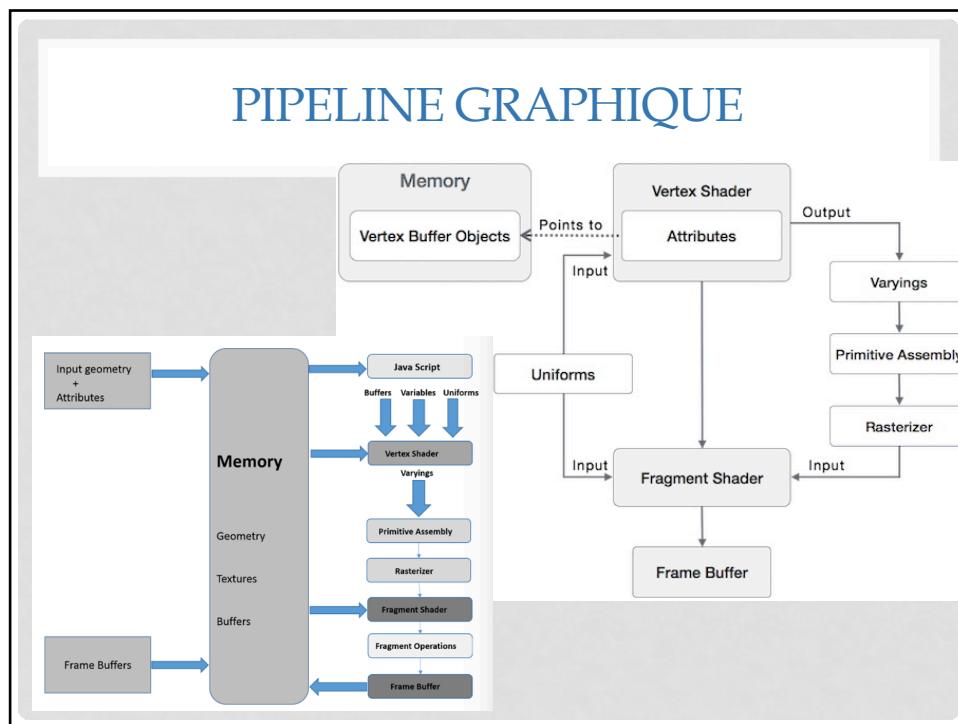
## VOCABULAIRE

- **Variables GLSL ES :**
  - **Attributes :**
    - variables contenant les entrées du vertex shader.
    - pointent sur le Vertex Buffer Object.
    - valeurs modifiées à chaque appel
  - **Uniforms :**
    - Variables contenant les données communes au fragment shader et au vertex shader (couleur, coordonnées de textures, position de la lumière...)
  - **Varyings :** variable pour transmettre les données du vertex shader vers le fragment shader

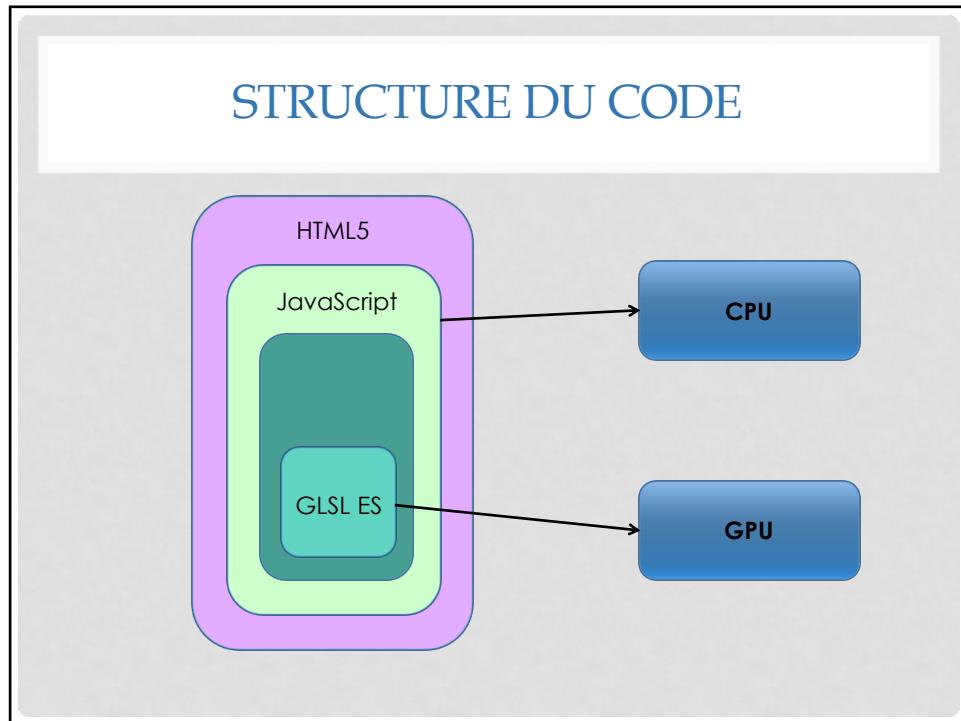
43



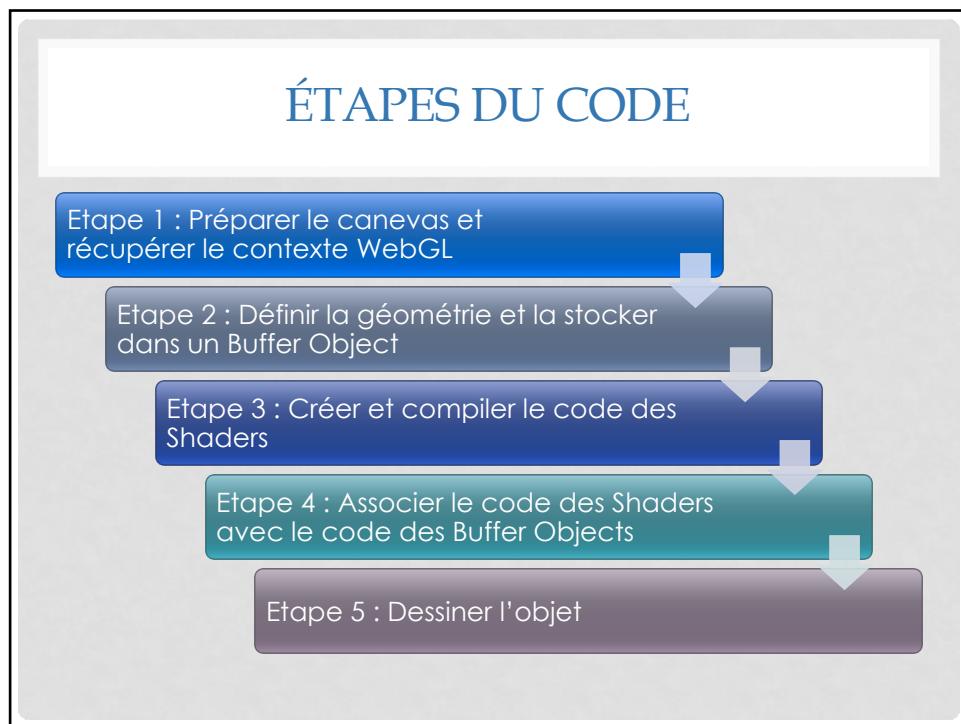
45



46



47



48

## PRÉPARER LE CANVAS

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>WebGL Demo</title>
<link rel="stylesheet" href="../webgl.css" type="text/css">
</head>
<body>
<canvas id="glcanvas" width="640" height="480"></canvas>
</body>
<script src="../gl-matrix.js"></script>
<script src="ex2.js"></script>
</html>
main();
function main() {
  const canvas = document.querySelector("#glcanvas");
  // Initialisation du contexte GL
  const gl = canvas.getContext("webgl");
  // Continue seulement si WebGL est disponible et fonctionne
  if (!gl) {
    alert("Impossible d'initialiser WebGL.");
    return;
}
```



49

## LES SHADERS

- Programme écrit en GLSL ([OpenGL ES Shading Language](#)) : Utilise les informations des sommets pour générer les données nécessaires au rendu
  - Vertex shader
  - Fragment shader

} Shader program
- Transmis à WebGL pour compilation et exécution GPU

50

## PROGRAMME SHADER DE SOMMET

```
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec4 aVertexPosition;
uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
void main() {
    gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
</script>

function main() {
...
// Récupérer le texte du shader de sommet
const vertexShaderSource = document.getElementById('shader-vs').textContent;
...
}
```

51

## PROGRAMME SHADER DE FRAGMENT

```
<script id="shader-fs" type="x-shader/x-fragment">
void main(void) {
    gl_FragColor = vec4(0.8, 0.8, 1.0, 1.0);
}
</script>

function main() {
...
// Récupérer le texte du shader de fragment
const fragmentShaderSource = document.getElementById('shader-fs').textContent;
...
}
```

52

## CRÉER LES SHADERS

```
// Crée un shader du type fourni, charge le source et le compile.
function loadShader(gl, type, source) {
    const shader = gl.createShader(type);

    // Envoyer le source à l'objet shader
    gl.shaderSource(shader, source);

    // Compiler le programme shader
    gl.compileShader(shader);

    // Vérifier s'il a été compilé avec succès
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert('An error occurred compiling the shaders: ' + gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}
```

53

## CRÉER LES SHADERS

```
// Initialiser un programme shader, de façon à ce que WebGL sache comment dessiner nos données
function initShaderProgram(gl, vsSource, fsSource) {
    const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource);
    const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource);

    // Créer le programme shader
    const shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    // Si la création du programme shader a échoué, alerte
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert('Unable to initialize the shader program: ' + gl.getProgramInfoLog(shaderProgram));
        return null;
    }
    return shaderProgram;
}
```

54

## CRÉER LES SHADERS

```
function main() {
...
//Initialiser un programme shader pour que WebGL sache comment dessiner les données
const shaderProgram = initShaderProgram(gl, vertexShaderSource, fragmentShaderSource);

// Toutes les informations nécessaire pour le programme de shader .
// Recherche quel emplacement est assigné à nos entrées (1 attribut et deux uniformes)
const programInfo ={
    program: shaderProgram,
    attribLocations: {
        vertexPosition: gl.getAttribLocation(shaderProgram, 'aVertexPosition'),//création d'un pointeur
        pour les données de sommets
    },
    uniformLocations: {
        projectionMatrix: gl.getUniformLocation(shaderProgram, 'uProjectionMatrix'),
        modelViewMatrix: gl.getUniformLocation(shaderProgram, 'uModelViewMatrix'),
    },
};
...
}
```

55

## CRÉER LES SHADERS

```
function main() {
...
// Initialiser les buffers dont on a besoin
const buffers = initBuffers(gl);

// Rendu de la scène
drawScene(gl, programInfo, buffers);
...
}
```

56

## CRÉER LE(S) BUFFER(S)

```
function initBuffers(gl) { // Initialiser les buffers dont on a besoin
    // Crée un tampon des positions pour le triangle.
    const positionBuffer = gl.createBuffer();
    // On applique des opérations sur le tampon position buffer
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    // tableau des positions des sommets pour le triangle.
    const positions = [
        0.0, 1.0, 1.0,
        -Math.sqrt(3)/2.0, -1.0/2.0, 1.0,
        Math.sqrt(3)/2.0, -1.0/2.0, 1.0
    ];
    // liste des positions passée à WebGL pour construire la forme.
    // En créant un Float32Array à partir du tableau JS, puis en remplaçant le tampon en cours.
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions), gl.STATIC_DRAW);
    return {
        position: positionBuffer,
    };
}
```

57

## RENDU DE LA SCENE

- Effacer le canevas
- Perspective de la caméra
- Position de l'objet
- Lier le tampon des sommets à l'attribut aVertexPosition utilisé par le shader
- Dessin de l'objet

58

## INITIALISATION DE LA SCÈNE

```
function drawScene(gl, programInfo, buffers) {
    gl.clearColor(0.0, 0.0, 0.0, 1.0); // couleur d'effacement
    gl.clearDepth(1.0);           // tout effacer
    gl.enable(gl.DEPTH_TEST); // test de profondeur activé
    gl.depthFunc(gl.LEQUAL); // élimination des parties cachées

    // Effacer le canevas
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

59

## MATRICES

```
...
// Matrice de perspective
//(45 degrés, rapport largeur/hauteur = taille d'affichage du canvas ;
// plan proche et plan éloigné entre 0,1 et 100
const fieldOfView = 45 * Math.PI / 180; // en radians
const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
const zNear = 0.1;
const zFar = 100.0;
const projectionMatrix = mat4.create();
// matrice de projection
mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
// Matrice de modélisation
const modelViewMatrix = mat4.create();
// Déplacer le dessin du triangle
mat4.translate(modelViewMatrix, // matrice destination
               modelViewMatrix, // matrice à translater
               [-0.0, 0.0, -6.0]); // paramètres de translation
```

60

## REmplir l'attribut

```
// Indiquer à WebGL comment extraire les positions à partir du tampon des
// positions pour les mettre dans l'attribut vertexPosition.
{
    const numComponents = 3; // 3 valeurs par itération
    const type = gl.FLOAT; // les données du tampon sont des flottants 32bit
    const normalize = false; // ne pas normaliser
    const stride = 0; // nbre d'octets à extraire entre 2 jeux de valeurs
        // 0 = utiliser le type et numComponents ci-dessus
    const offset = 0; // nombre d'octets avant début dans le tampon
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);
    gl.vertexAttribPointer(
        programInfo.attribLocations.vertexPosition,
        numComponents, type, normalize, stride, offset);
    gl.enableVertexAttribArray(
        programInfo.attribLocations.vertexPosition);
}
```

61

## AFFICHAGE

```
// Indiquer à WebGL d'utiliser notre programme pour dessiner
gl.useProgram(programInfo.program);

// Définir les uniformes du shader
gl.uniformMatrix4fv(
    programInfo.uniformLocations.projectionMatrix,
    false, projectionMatrix);
gl.uniformMatrix4fv(
    programInfo.uniformLocations.modelViewMatrix,
    false, modelViewMatrix);

{
    const offset = 0;
    const vertexCount = 3;
    gl.drawArrays(gl.TRIANGLES, offset, vertexCount);
}
```

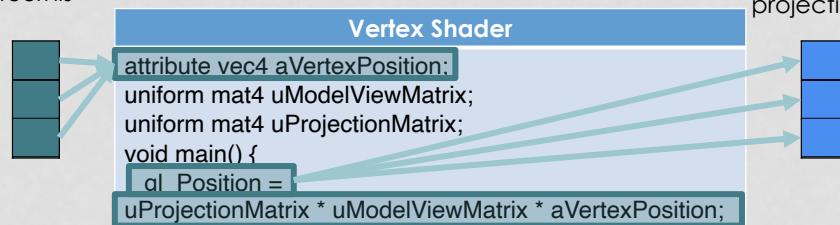
62

## VERTEX SHADER

```
{
    const offset = 0;
    const vertexCount = 3;
    gl.drawArrays(gl.TRIANGLES, offset, vertexCount);
}
```

Sommets fournis

Sommet ds l'espace de projection

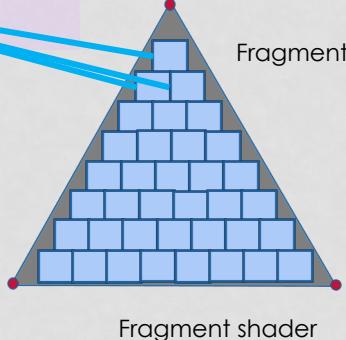


63

## FRAGMENT SHADER

```
<script id="shader-fs" type="x-shader/x-fragment">
void main(void) {
    gl_FragColor = vec4(0.8, 0.8, 1.0, 1.0);
}
</script>
```

Vertex shader



Fragment shader

64

## MÉTHODE DE DESSIN

- Dessiner à partir des coordonnées des sommets  
`void drawArrays(enum mode, int first, long count)`
  - Mode : choix des primitives de dessin (gl.POINTS, gl.LINE\_STRIP, gl.LINE\_LOOP, gl.LINES, gl.TRIANGLE\_STRIP, gl.TRIANGLE\_FAN, gl.TRIANGLES)
  - First : l'élément du tableau d'où le dessin part
  - Count : nombre d'éléments à dessiner
- Dessiner à partir des coordonnées des sommets et des indices de points des faces  
`void drawElements(enum mode, long count, enum type, long offset)`
  - Type : types de données des indices gl.UNSIGNED\_BYTE ou gl.UNSIGNED\_SHORT
  - Offset : Point de départ de l'affichage (0 en général)

65

## BUFFER

- Type de buffer
  - ARRAY\_BUFFER représente les données des sommets pour dessin avec drawArrays
  - ELEMENT\_ARRAY\_BUFFER représente les données des indices pour dessin avec drawElements
- Tableau typé
  - Float32Array pour les données des sommets
  - Uint16Array pour les indices
- Utilisation des données et stockage
  - gl.STATIC\_DRAW
  - gl.STREAM\_DRAW
  - gl.DYNAMIC\_DRAW

66

## VERTEX BUFFER OBJECTS

- Création d'un buffer vide : `const positionBuffer = gl.createBuffer();`
- Associer un tableau : `gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);`
  - ARRAY\_BUFFER ou ELEMENT\_ARRAY\_BUFFER
- Transmettre les données au buffer :
  - `gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions), gl.STATIC_DRAW);`
  - `Float32Array, Uint16Array`
- Libérer le buffer: `gl.bindBuffer(gl.ARRAY_BUFFER, null);`

67

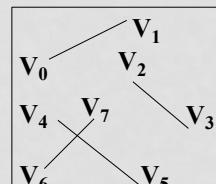
## POINTS

$V_0$	$V_1$	$V_2$
•	•	•
•	•	•
$V_3$	$V_4$	

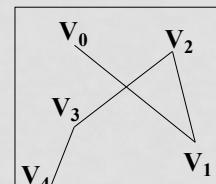
`gl.POINTS`

68

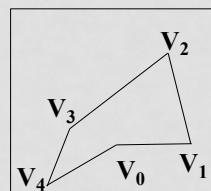
## LIGNES



gl.LINES



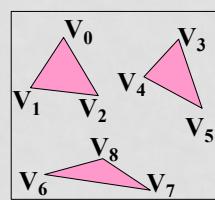
gl.LINE\_STRIP



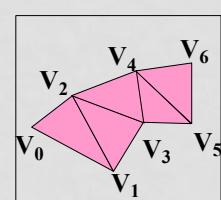
gl.LINE\_LOOP

69

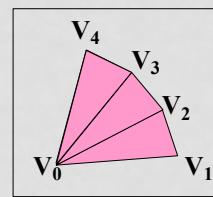
## TRIANGLES



gl.TRIANGLES



gl.TRIANGLE\_STRIP



gl.TRIANGLE\_FAN

70

## TAILLE / ÉPAISSEUR

- Modifier taille des points

Dans vertex shader

```
gl_PointSize = 10.0;
```

- Modifier épaisseur des lignes

Dans JS

```
gl.lineWidth(5);
```

71

## COULEURS

- Couleur appliquée à l'objet dans le fragment shader

```
gl_FragColor = vec4(0.8,0.8, 1.0,1.0);
```

- Autre solution : passer une couleur par sommet

- Attribution d'une couleur par sommet avec un tableau JS

```
var couleurs = [ ... ];
```

- Le stocker dans un Buffer

```
const positionBuffer= gl.createBuffer(); //création du buffer object
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer); //association
Var couleurs=[...]
//passer le tableau JS au buffer
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(couleurs), gl.STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER, null);
return {
  position: positionBuffer,
  couleur:couleurBuffer
};
```

72

## COULEURS

- Dans le vertex shader :

```
attribute vec4 aVertexColor;
varying lowp vec4 vColor;
vColor = aVertexColor;
```

- Ensuite varying est associée à gl\_FragColor :

```
varying lowp vec4 vColor;
gl_FragColor = vColor;
```

- Associer le code des shaders aux buffer objects :

```
//récupérer l'emplacement de l'attribut
shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
"aVertexColor");
//activer l'attribut
gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
// pointe l'attribut sur le color buffer object
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
vertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

73

## VARYING VCOLOR

```
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec4 aVertexPosition;
attribute vec4 aVertexColor;
uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
varying lowp vec4 vColor;
void main() {
    gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
    vColor = aVertexColor
}
</script>
```



INTERPOLATION

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



gl.TRIANGLES

74

## ANIMATION CLAVIER

- Variable globale

```
var rotAxeX =0;
```

- Appel de l'événement

```
window.onkeydown = handleKeyPressed;
```

- Fonction associée

```
function handleKeyPressed(evenement){
    switch(evenement.keyCode){
        case 88: if(evenement.shiftKey) rotAxeX =0.1;
        else rotAxeX +=0.1; break;
        default: break;}}
```

- Affichage

```
function render() {
    drawScene(gl, programInfo, buffers);
    window.requestAnimationFrame(render);
}
window.requestAnimationFrame(render);
```

- Transformation

```
mat4.rotateX(modelViewMatrix,modelViewMatrix, rotAxeX);
```

76

## ANIMATION AUTOMATIQUE

- Variable globale

```
var rotAxeZAuto =0;
```

- Affichage

```
var then = 0;
function render(now) {
    now *= 0.001; // convert to seconds
    const deltaTime = now - then;
    then = now;
    drawScene(gl, programInfo, buffers, deltaTime);
    window.requestAnimationFrame(render);
}
window.requestAnimationFrame(render);
```

- Fonction animation

```
rotAxeZAuto+=deltaTime;
```

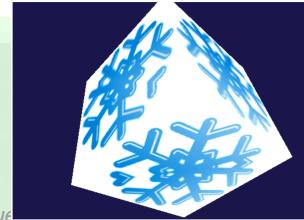
- Transformation

```
mat4.rotate(modelViewMatrix,modelViewMatrix, rotAxeZAuto,[0, 0, 1]);
```

77

## CHARGER LES TEXTURES

```
function loadTexture(gl, url) {
  const texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  const level = 0; const internalFormat = gl.RGBA;
  const width = 1; const height = 1; const border = 0;
  const srcFormat = gl.RGBA;
  const srcType = gl.UNSIGNED_BYTE;
  const pixel = new Uint8Array([0, 0, 255, 255]); // opaque blue
  gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, width, height, border, srcFormat,
  srcType, pixel);
  const image = new Image();
  image.onload = function() {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, srcFormat, srcType, image);
    // wrapping to clamp to edge
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  };
  image.src = url;
  return texture;
}
```



78

## CHARGER LES TEXTURES

```
const image = new Image();
image.onload = function() {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, level, internalFormat,
  srcFormat, srcType, image);
  // wrapping to clamp to edge
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
};
image.src = url;
return texture;
}
```

```
const texture = loadTexture(gl, 'flocon.png');
```



79

## METTRE À JOUR LES SHADERS

- initShaders
 

```
const programInfo = {
    program: shaderProgram,
    attribLocations:
      textureCoord: gl.getAttribLocation(shaderProgram, 'aTextureCoord'),
    },
    uniformLocations: {
      uSampler: gl.getUniformLocation(shaderProgram, 'uSampler'),
    },
};
```
- Fragment shader
 

```
varying highp vec2 vTextureCoord;
uniform sampler2D uSampler;

gl_FragColor = texture2D(uSampler, vec2(vTextureCoord));
```
- Vertex shader
 

```
attribute vec2 aTextureCoord;
varying highp vec2 vTextureCoord;

vTextureCoord = aTextureCoord;
```

80

## DESSINER AVEC TEXTURE

- Lier les coordonnées de texture aux sommets

```
const textureCoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
const textureCoords = [ ... ];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
  gl.STATIC_DRAW);
```

- Attention :

Adresses web pour l'image du même domaine

81

## DESSINER AVEC TEXTURE

- Attribut texture

```
{
  const numComponents = 2;
  const type = gl.FLOAT;
  const normalize = false;
  const stride = 0;
  const offset = 0;
  gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord);
  gl.vertexAttribPointer(
    programInfo.attribLocations.textureCoord,
    numComponents,
    type,
    normalize,
    stride,
    offset);
  gl.enableVertexAttribArray(
    programInfo.attribLocations.textureCoord); }
```

- Dessiner le cube texture

```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(programInfo.uniformLocations.uSampler, 0);
```

82

## LIBRAIRIES

- glMatrix : librairie JavaScript pour les calculs matriciels

glmatrix v 2 <http://glmatrix.net/>

- webgl-utils

<https://webglfundamentals.org/docs/module-webgl-utils.html>

83

## THREE.JS

- [https://dinosaurscode.xyz/tutorials/2016/07/15/three  
\\_js-tutorial-for-beginners/](https://dinosaurscode.xyz/tutorials/2016/07/15/three-js-tutorial-for-beginners/)
- [http://davidscottlyons.com/threejs/presentations/fr  
ontporch14/#slide-0](http://davidscottlyons.com/threejs/presentations/fr<br/>ontporch14/#slide-0)
- ...

84

## PLUS D'INFORMATIONS

### Outils

- <http://www.babylonjs.com/cyos/>

### WebGL

- [http://learningwebgl.com/blog/?page\\_id=1217](http://learningwebgl.com/blog/?page_id=1217)
- [https://developer.mozilla.org/fr/docs/Web/API/WebGL\\_API](https://developer.mozilla.org/fr/docs/Web/API/WebGL_API)
- <https://www.khronos.org/>
- <https://www.tutorialspoint.com/webgl/index.htm>

85