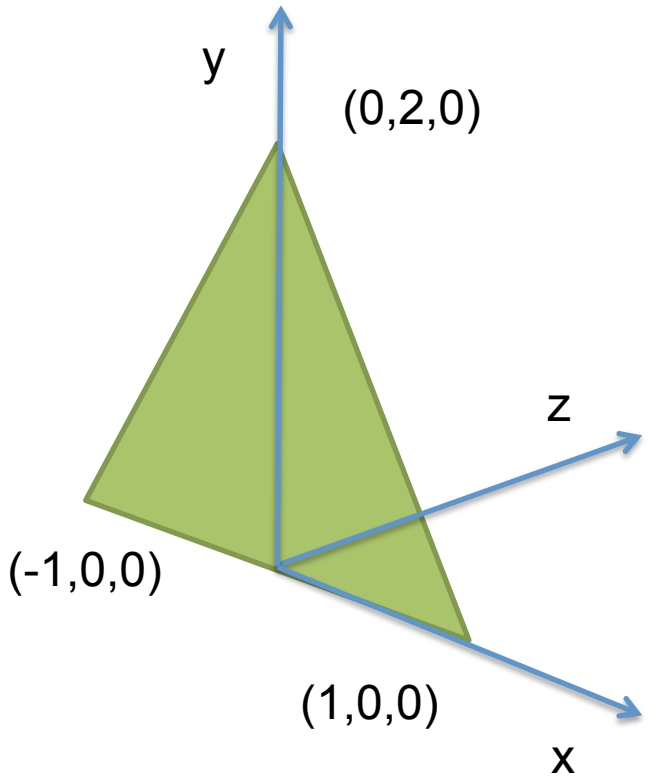
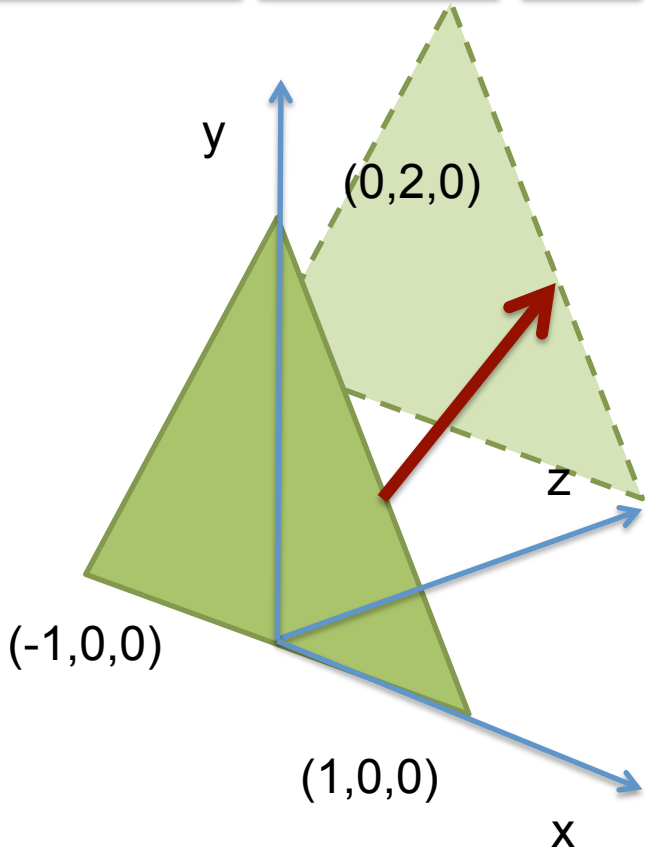
Transformations géométriques

- Un objet est défini par les coordonnées de ses sommets
  - Vertices = [1,0,0,0,2,0,-1,0,0]
  - Coordonnées dans le repère « OpenGL »
    - Caméra (implicitement) à l'origine, alignée selon l'axe Z positif



Transformations géométriques

- Transformation géométrique
  - Construire une scène complexe à partir d'objets simples
    - Positionner, orienter, mettre à l'échelle
  - Produire une animation
    - Nouvelle position, orientation, etc., à chaque image
  - Type de transformation
    - Translation
    - Rotation
    - Mise à l'échelle



Transformations géométriques

- Représentation des vecteurs en **coordonnées homogènes**
  - Toutes les transformations deviennent matricielles(4x4)
  - Composition de transformations
    - Produits de matrices

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \leftrightarrow \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ w \end{bmatrix}$$

$$T(t_x,t_y,t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(s_x,s_y,s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

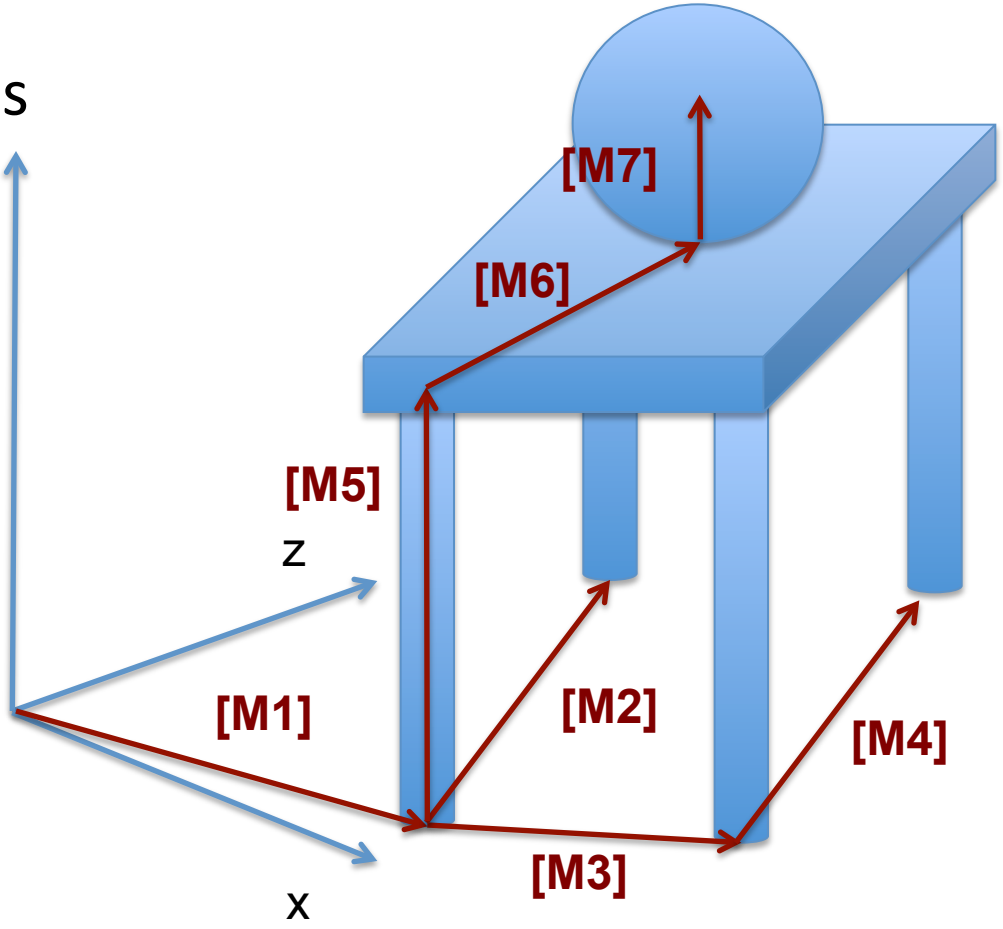
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Transformations géométriques

- Gestion des transformations
  - Pile de transformations
    - Push
    - Pop : Retour en arrière

[M1]
[M1] [M2]
[M1]
[M1] [M3]
[M1] [M3] [M4]
[M1] [M5]
[M1] [M5] [M6]
[M1] [M5] [M6] [M7]



Transformations géométriques

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- WebGL ne propose pas de bibliothèque matricielle
- Nombreuses bibliothèques
  - EWGL
  - TDLFast
  - Closure
  - mjs
  - glMatrix
  - TDLMath
  - CanvasMatrix
  - Sylvester
- Comparaison
  - http://stepheneb.github.com/webgl-matrix-benchmarks/matrix_benchmark.html

API
Multiplication
Translation
Scaling
Rotation (arbitrary axis)
Rotation (x|y|z axis)
Transpose
Inverse
Inverse (3x3)

Transformations géométriques

- Création de matrices
  - mvMatrix(Model-View)
    - Matrice permettant de passant des coordonnées de l'espace model à l'espace view
  - pMatrix (projection perspective)
    - Matrice permettant de passer de l'espace view à la projecion perpective 2D

```
// transformations géométriques
// Model->View
var mvMatrix = mat4.create();
// pile de mv matrices
var mvMatrixStack = [];
// projection matrice
var pMatrix = mat4.create();
```

```
// initialisation pile de transformation
mat4.identity(mvMatrix);
mvPushMatrix();
mat4.translate(mvMatrix, [0, 0, -5]);
mat4.rotate(mvMatrix, alpha, [0, 1, 0]);
mat4.rotate(mvMatrix, beta, [1, 0, 0]);
// envoi des buffers
gl.bindBuffer(…); gl.vertexAttribPointer(…);
gl.bindBuffer(…);
// envoi piles de transformations
setMatrixUniforms();
// dessin
gl.drawElements(…);
// retour à la transformation précédente
mvPopMatrix();
```

Transformations géométriques

- Gestion de la pile
  - Model-View uniquement
- Envoi des matrices
  - ModelView, Normal

```
// mvPushMatrix
function mvPushMatrix() {
  var copy = mat4.create();
  mat4.set(mvMatrix, copy);
  mvMatrixStack.push(copy);
}
// mvPopMatrix
function mvPopMatrix() {
  if (mvMatrixStack.length == 0) {
    throw "Invalid popMatrix!";}
  mvMatrix = mvMatrixStack.pop();
}
```

```
// setMatrixUniforms
function setMatrixUniforms() {
  gl.uniformMatrix4fv(currentProgram.pMatrixUniform, false, pMatrix);
  gl.uniformMatrix4fv(currentProgram.mvMatrixUniform, false, mvMatrix);

var normalMatrix = mat3.create();
  mat4.toInverseMat3(mvMatrix, normalMatrix);
  mat3.transpose(normalMatrix);
  gl.uniformMatrix3fv(currentProgram.nMatrixUniform, false, normalMatrix);
}
```

Transformations géométriques

ESIR ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- Vertex Shader

```
// vertex shader
// attributes inputs
attribute vec3 aVertexPosition;
attribute vec3 aVertexNormal;

// uniform matrices
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
uniform mat3 uNMatrix;

void main(void) {
  // model to view
  // vertex
  vec4 mvPosition = uMVMatrix *   vec4(aVertexPosition, 1.0);
  // normal
  vec3 transformedNormal = uNMatrix * aVertexNormal;
  // view to projection
  gl_Position = uPMatrix * mvPosition;
}
```

Transformations géométriques

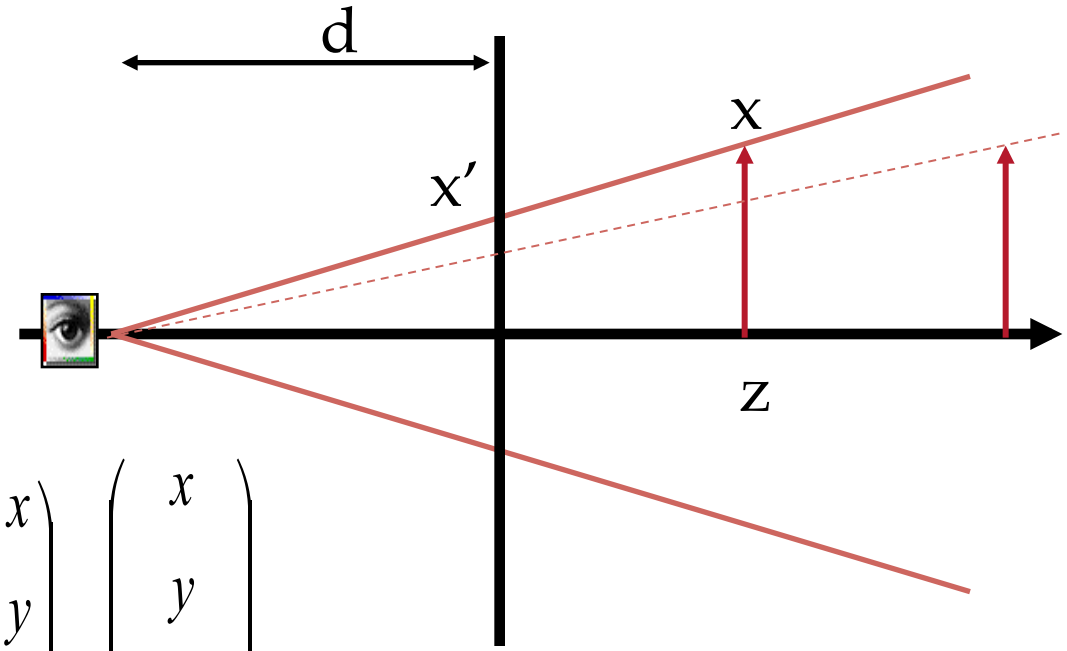
- Projection perspective

$$\frac{x'}{d} = \frac{x}{z+d}$$

$$x' = \frac{x}{\frac{z}{d}+1}$$

$$w' = \frac{z}{d}+1$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ \frac{z}{d}+1 \end{pmatrix}$$



Transformations géométriques

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- Projection perspective

```
// projection matrice
var pMatrix = mat4.create();

mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
```

```
// pour info
// nom des fonctions compatibles avec les versions openGL < 3
mat4.frustum=function(a,b,c,d,e,g,f){
  f||(f=mat4.create());
  var h=b-a, i=d-c, j=g-e;
  f[0]=e*2/h;      f[1]=0;       f[2]=0;            f[3]=0;
  f[4]=0;          f[5]=e*2/i;   f[6]=0;            f[7]=0;
  f[8]=(b+a)/h;    f[9]=(d+c)/i; f[10]=-(g+e)/j;    f[11]=-1;
  f[12]=0;         f[13]=0;      f[14]=-(g*e*2)/j;  f[15]=0;
  return f};

mat4.perspective=function(a,b,c,d,e){
  a=c*Math.tan(a*Math.PI/360);
  b=a*b;
  return mat4.frustum(-b,b,-a,a,c,d,e)};
```

Transformations géométriques

- Quand un maillage un maillage est dessiné avoir
  - Matrice « Model-View » [4x4] : uMVMatrix
    - Des coordonnées du maillage dans son repère (de modélisation) à celle dans le repère de la caméra
  - Matrice « Normal » [3x3] : uNMatrix
    - Des coordonnées des normales dans le repère modèle à celles dans le repère de la caméra
  - Matrice de « Projection » [4x4] : uPMatrix
    - Projection perspective

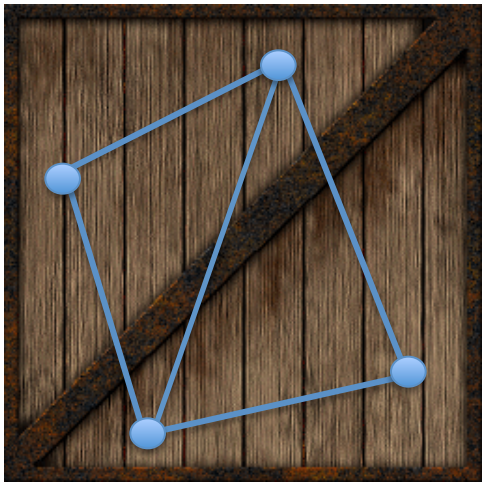
```
// vertex shader

// uniform matrices
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
uniform mat3 uNMatrix;

void main(void) {
  // model to view
  // vertex
  vec4 mvPosition =
    uMVMatrix * vec4(aVertexPosition, 1.0);
  // normal
  vec3 transformedNormal =
    uNMatrix * aVertexNormal;
  // view to projection
  gl_Position = uPMatrix * mvPosition;
}
```

Textures

- Image plaquée sur une géométrie
  - Coordonnées de texture : buffer
    - Espace texture [0,1]x[0,1]
    - Coordonnées du sommet dans la texture
  - Image texture transférée dans la « mémoire texture » de la carte
  - Unités de texture
    - 32 unité de textures disponible simultanément
    - « Sampler » dans *shader*
  - Texture
    - taille 2^n
    - Format « png » de préférence



[0.2,0.5]

Textures

- Initialisation de la texture
  - Chargement asynchrone
  - Fin de l'init quand l'image est chargée

```
// texture OpenGL
var texture;

function initTexture() {
  // creation de la texture
  texture= gl.createTexture();
  // image texture
  texture.image = new Image();
  // chargement asynchrone de la texture
  texture.image.onload = function () { handleLoadedTexture(texture)}
  texture.image.src = "texture.png";
}
```

Textures

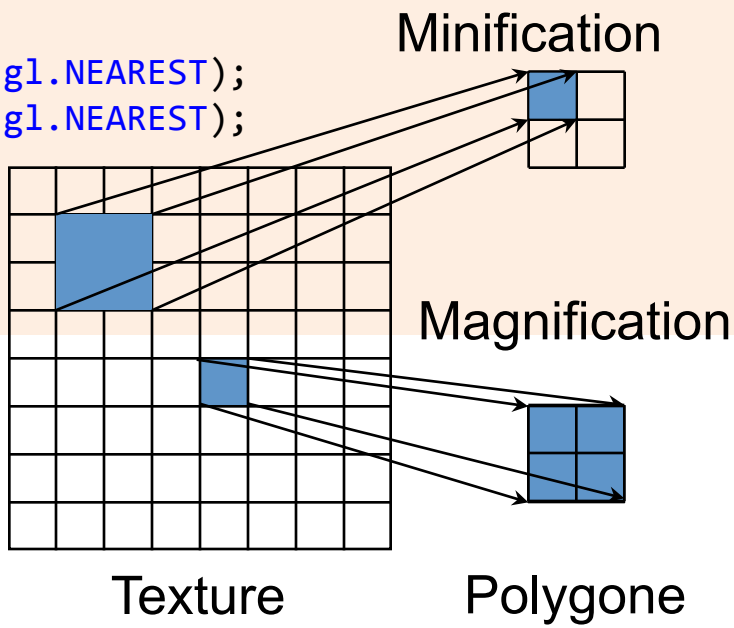
```
function handleLoadedTexture(texture) {
  // activation de la texture
  gl.bindTexture(gl.TEXTURE_2D, texture);

  // paramétrage texture
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);

  // image source et format
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, texture.image);

  // gestion des zooms
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);

  // aucune texture active
  gl.bindTexture(gl.TEXTURE_2D, null);
}
```



Textures

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

```
var cubeVertexPositionBuffer;
var cubeVertexIndexBuffer;

var cubeVertexTextureCoordBuffer;

function initBuffers() {
  …
  // création buffer coordonnées de textures

  cubeVertexTextureCoordBuffer = gl.createBuffer();
  // activation buffer
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
  var textureCoords = [
    0.0, 0.0,  1.0, 0.0,    1.0, 1.0,     0.0, 1.0, // Front face
    …
    0.0, 0.0, 1.0, 0.0,     1.0, 1.0,     0.0, 1.0]; // Left face

  // association des données au buffer
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
  cubeVertexTextureCoordBuffer.itemSize = 2;
  cubeVertexTextureCoordBuffer.numItems = 24;


  …
}
```

Textures

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

```
function initShaders() {

  …

  shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
    "aVertexPosition");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

  // coordonnees de textures : “attribute”
  shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram,
    "aTextureCoord");
  gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

  // matrice de transformation
  shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
  shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");

  // unité de texture (“sampler” dans les shaders)
  shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");

}
```

Textures

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

```
function drawScene() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  // transformations géométriques

   gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
   gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
      cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

   // coordonnées de textures
   gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
   gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
     cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

   // gestion des unités de textures
   gl.activeTexture(gl.TEXTURE0);                  // unite de texture active : 0
   gl.bindTexture(gl.TEXTURE_2D, texture);         // activation de la texture
   gl.uniform1i(shaderProgram.samplerUniform, 0);  // envoi de l'unité active : 0

   gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
   setMatrixUniforms();
   gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
```

Textures

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

```
<script id="shader-vs" type="x-shader/x-vertex">

  // vertex shader
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

  // sortie vers le fragment shader
    varying vec2 vTextureCoord;


    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

        vTextureCoord = aTextureCoord;
    }

</script>
```

Textures

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

```
<script id="shader-fs" type="x-shader/x-fragment">
 // fragment shader

    precision mediump float;

  //coordonnées de texture
  // calculée dans le vertex shader

    varying vec2 vTextureCoord;

  // sampler (unité de texture)
  // 32 unité de textures disponible

    uniform sampler2D uSampler;

    void main(void) {

        // texture2D : texel aux coordonnées u,v de l'unité de texture uSampler

        gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }

</script>
```

Moteurs 3D

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- Deux stratégies
  - Encapsuler les fonctions classiques
    - Chargement, dessin objets
    - Shaders classiques
    - Animations

    sans imposer un cadre de programmation
  - Même fonctionnalités mais en proposant en cadre de travail « fermé » et intégré

Moteurs 3D WebGL

ESIR ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- C3DL
- Cesium
- CopperLicht
- CubicVR.js
- EnergizeGL
- GammaJS
- **GLGE**
- GlowScript
- GTW
- Inka3D
- J3D
- Jax
- Nutty
- KickJS
- KriWeb
- Lightgl.js
- **O3D**
- Oak3D
- OpenWebGlobe SDK
- OSG.JS
- Parallax
- PhiloGL
- SceneJS
- **SpiderGL**
- TDL
- **Three.js**

Conclusion

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- OpenGL
  - Ensemble de fonctionnalités
    - Formater et envoyer des données à la carte graphique
      - Maillage, texture, etc.
    - Programmer des shaders
      - Compilation, éditions de liens
      - Langage de programmation : GLSL
  - Nécessite de la pratique
    - Conception de shaders
    - Programmation efficace

Conclusion

ECOLE SUPERIEURE D'INGENIEUR DE RENNES

- Ce qui manque dans le cours
  - Shader multi-passes
  - Rendu dans un FrameBufferObject (FBO)
    - Cible de rendu -> FBO
  - Utilisation du FBO comme une entre (ou une texture) dans un second shader.