Introduction à WebGL

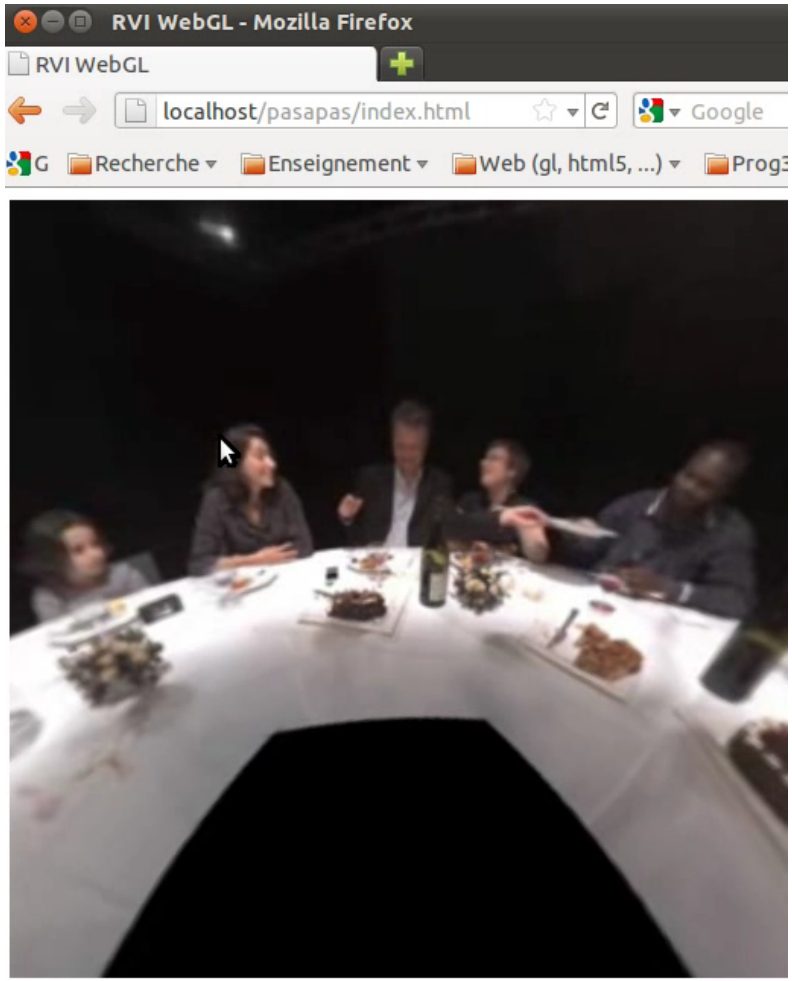
Réalité Virtuelle et Interaction

Fabrice Aubert
fabrice.aubert@lifl.fr

IEEA - Master Info - Parcours IVI

2013-2014

Objectif



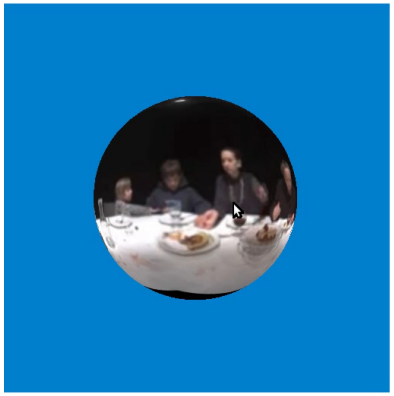
film : "un repas de famille suite au décès du grand-père" / extrait
Tournage : Caméra 360 - EDM / lieu : plateau du Fresnoy, Studio national des arts contemporains.
Atelier de recherche 2e cycle Art mis en place par Christl Lidl en collaboration avec Stéphane Dwernicki et Patrick Beaucé de l'option Design ; Etudiants : Rémi Casiez, Fabien Foulon, Laurène Marcant, Chloé Petitjean, Honorine Poisson, Saito Mitsuaki, Aleksi Fermon.

http ://ecoledesbeauxarts.valenciennes.fr

Principe

- Le film est prêt à être plaqué "naturellement" sur une sphère (projection sphérique)

- Il suffit de créer une sphère texturée



- et placer le point de vue à l'intérieur de la sphère, en son centre.

La sphère

- Décomposer en méridiens (angle θ sur 360 degrés)/parallèles(angle ϕ sur 180 degrés).
- Chaque sommet P calculé par :

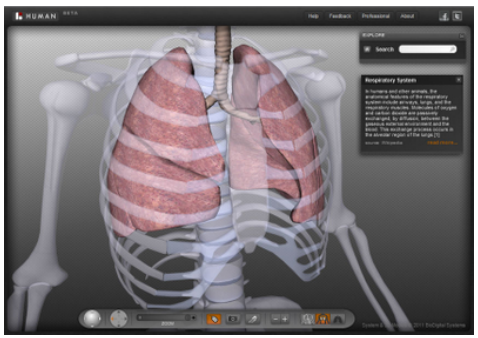
$$P = \begin{cases} x &= cos(\theta)sin(\phi) \\ y &= cos(\phi) \\ z &= sin(\theta)sin(\phi) \end{cases} \quad \text{avec} \quad \begin{array}{l} \theta \in [0, 2\pi] \\ \phi \in [0, \pi] \end{array}$$

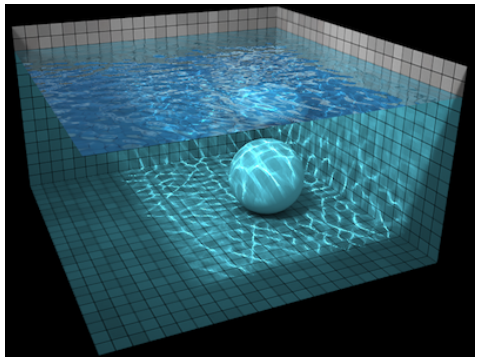
- Coordonnées de texture en P :

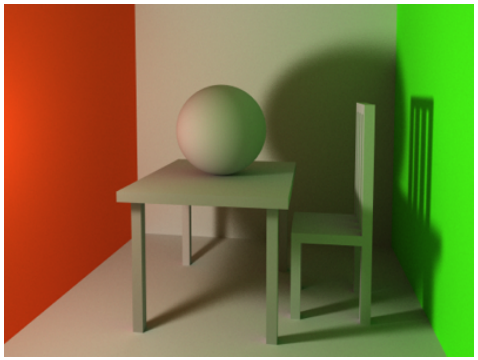
$$\begin{cases} s &= \frac{\theta}{2\pi} \\ t &= \frac{\phi}{\pi} \end{cases}$$

Introduction à WebGL

- Version courante : WebGL 1.0
- WebGL = "OpenGL pour le web" (sans plugin, directement intégré dans le navigateur)
- WebGL = Jeu d'instructions JavaScript (interprété par le navigateur client).
- Spécification basée sur openGL ES 2.0 (WebGL 2.0 $\Rightarrow$ openGL ES 3.0).
- WebGL s'adresse à la balise `<canvas>` de HTML5 (contexte d'affichage).
- Intégré dans `Firefox`, `Chrome`, `Opera` (2010) et `Safari` (sept 2011).
- Internet Explorer : à partir de la version 11 (septembre 2013 : release preview).







Quelques liens

- Site officiel :
  - http ://www.khronos.org/webgl/
- Tutorial (parmi d'autres !) :
  - http ://learningwebgl.com/blog/ ?page_id=1217
- Informations générales (très complet) :
  - https ://developer.mozilla.org/en/WebGL
- Démos (parmi d'autres...) :
  - http ://www.chromeexperiments.com/webgl

1 Mise en place

Canvas HTML5

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8" />
  <title>RVI WebGL</title>
  <script type="text/javascript" src="main.js"></script>
</head>

<body>
  <canvas id="webglCanvas" width="512" height="512">
  </canvas>
</body>

</html>
```

Initialisation du contexte WebGL (2/2)

- coté javascript (i.e. fichier `main.js`) :

```
window.addEventListener('load',main,false);

var gl; // will contain the webgl context

/** ********************************************** */
/** initialize the gl context from the canvas + basic default gl settings
 *
 */
function initGL() {
  canvas=document.getElementById("webglCanvas");
  gl=canvas.getContext("webgl");
  if (!gl) {
    alert("cant␣initialize␣webgl␣context");
  }
  else {
  console.log(gl.getParameter(gl.VERSION) + "␣|␣" + gl.getParameter(gl.VENDOR) + "␣|␣" +
              gl.getParameter(gl.RENDERER) + "␣|␣" +
                gl.getParameter( gl.SHADING_LANGUAGE_VERSION )
  );

    gl.clearColor(0,0,0,1);
    gl.clearDepth(1.0);
    gl.enable(gl.DEPTH_TEST);

    gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT);
  }
}

/** ********************************************** */
function main() {
  initGL();
}
```

⇒ récupération du canvas html5 et création du contexte par

Boucle d'affichage

```
/** ********************************************* */
/** update data for general loop (called by loop() )
 *
 */
function updateData() {
}

/** ********************************************* */
/** draw the scene (called by loop() )
 *
 */
function drawScene() {
  gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT);
}

/** ********************************************* */
/** main loop : draw, capture event, update scene, and loop again
 *
 */
function loop() {
  drawScene();
  updateData();
  window.requestAnimationFrame(loop);
}
```

WebGL = OpenGL ?

- WebGL 1.0 est basé sur la spécification d'OpenGL ES 2.0 (OpenGL pour les mobiles).
- Conséquence : tout shader (pas de pipeline fixe)
- $\Rightarrow$ librairie très bas niveau.
- Spec : http ://www.khronos.org/opengles/sdk/docs/man/
- Pour les projets : nécessité d'avoir une librairie haut niveau.
- Exemple : `three.js` (http ://threejs.org/).
- + éventuellement d'autres librairies pour le web (i.e. Ajax)
- Pour le tp : sans librairie externe.

Objectif TP

- Initialisation des buffers pour la géométrie (une sphère).
- placage de texture (images de la vidéo)
- mise en place des shaders (positionnement + placage texture)
- + interaction souris

2 Shaders

Code source dans le html

```
<head>
  <meta charset="UTF-8" />
  <title>RVI WebGL</title>
  <script type="text/javascript" src="main.js"></script>

  <!-- Shaders -->
  <script id="hello-fs" type="x-shader/x-fragment">
    precision highp float; // obligatoire pour les float (no default)

    void main(void) {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 0.0);
    }
  </script>

  <script id="hello-vs" type="x-shader/x-vertex">
    attribute vec4 vertex;

    void main(void) {
        gl_Position = vertex;
    }
  </script>
</head>
...
```

Lecture du code source+compilation

```
/** ****************************************
 *  reads shader and compile
 * **/
function getShader(id) {
  var shaderScript = document.getElementById(id);
  var k = shaderScript.firstChild;
  var str=k.textContent;
  var shader;
  if (shaderScript.type == "x-shader/x-fragment") {
    shader = gl.createShader(gl.FRAGMENT_SHADER);
  }
  else if (shaderScript.type == "x-shader/x-vertex") {
    shader = gl.createShader(gl.VERTEX_SHADER);
  }
  gl.shaderSource(shader, str);
  gl.compileShader(shader);

  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(shader));
    return null;
  }

  return shader;
}
```

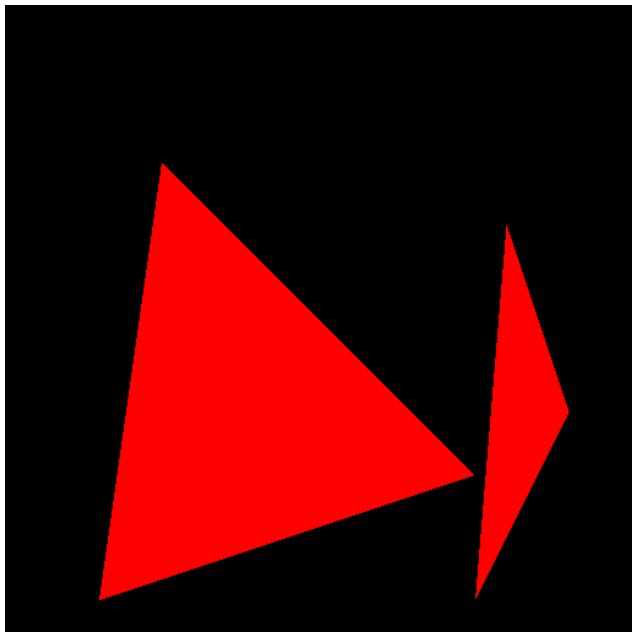
Lecture/creation du program shader

```
/** ********************************************* */
/** create the program shader (vertex+fragment) : source in html element
 *
 */
function createProgram(id) {
  var programShader=gl.createProgram();
  var vert=getShader(id+"-vs");
  var frag=getShader(id+"-fs");
  gl.attachShader(programShader,vert);
  gl.attachShader(programShader,frag);
  gl.linkProgram(programShader);
  if (!gl.getProgramParameter(programShader,gl.LINK_STATUS)) {
    alert(gl.getProgramInfoLog(programShader));
    return null;
  }
  return programShader;
}
```

⇒ Utilisation = `var helloProgramShader=createProgram("hello");`.

3 Géométrie et buffers

Obtenir :



Initialisation buffers

Utilisation de l'objet `Float32Array` (spécification WebGL) :

```
/** ********************************************* */
/** initialize triangle buffers that will be drawn
 *
 */
function initTriangle() {
  var vertex=[-0.5,0.5,0.0,0.5,-0.5,0.0,-0.7,-0.9,0.0, // first triangle
          0.6,0.3,0.0,0.8,-0.3,0.0,0.5,-0.9,0.0]; // second triangle

  var vertexBuffer=gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER,vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER,new Float32Array(vertex),gl.STATIC_DRAW);
  return vertexBuffer;

}
```

⇒ Utilisation = `var triangleVertexBuffer=initTriangle();`
Remarque : pas de VAO en WebGL 1.0

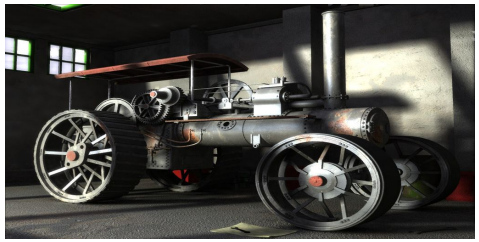
Affichage

```
** ****************************************** */
/** draw the scene (called by loop() )
 *
 */
 var triangleVertexBuffer;
 var helloProgramShader;
function drawScene() {
  gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT);

  // enable shader + get vertex location
  gl.useProgram(helloProgramShader);
  var vertexLocation=gl.getAttribLocation(helloProgramShader,'vertex');

  // draw geometry
  gl.enableVertexAttribArray(vertexLocation);

  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
  gl.vertexAttribPointer(vertexLocation,3,gl.FLOAT,gl.FALSE,0,0);

  gl.drawArrays(gl.TRIANGLES,0,6);

  // disable all
  gl.disableVertexAttribArray(vertexLocation);
  gl.useProgram(null);
}
```

4 Texture Vidéo

Obtenir :

fenêtre webgl



texture

Placage de texture simple dans shaders

```
<script id="hello-vs" type="x-shader/x-vertex">
    attribute vec4 vertex;
    attribute vec2 texCoord;

    varying vec2 vTexCoord;

    void main(void) {
        fTexCoord = texCoord;
        gl_Position = vertex;
    }
</script>

<script id="hello-fs" type="x-shader/x-fragment">
   precision highp float;

  varying vec2 fTexCoord;

  uniform sampler2D texture0;

  void main(void) {
    vec4 color=texture2D(texture0,fTexCoord);
    gl_FragColor = color;
  }
</script>
```

Initialisation texture

```
/** ********************************************************
 * Init texture from html id
 * **/

function initTexture(id) {
  var imageData=document.getElementById(id);

  textureId=gl.createTexture();
  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D,textureId);

  gl.texParameteri(gl.TEXTURE_2D,gl.TEXTURE_MIN_FILTER,gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D,gl.TEXTURE_MAG_FILTER,gl.LINEAR);
  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.texImage2D(gl.TEXTURE_2D,0,gl.RGB,gl.RGB,gl.UNSIGNED_BYTE,imageData);

  return textureId;
}
```

$\Rightarrow$ Attention aux dimensions en puissance de 2 si non clamp to edge !

Affichage

```
function drawScene() {
  gl.clear(gl.DEPTH_BUFFER_BIT | gl.COLOR_BUFFER_BIT);

  gl.useProgram(helloProgramShader);
  // get location (should be done once for all)
  var vertexLocation=gl.getAttribLocation(helloProgramShader,'vertex');
  var texCoordLocation=gl.getAttribLocation(helloProgramShader,'texCoord');
  var textureLocation=gl.getUniformLocation(helloProgramShader,'texture0');

  // set up uniform
  gl.uniform1i(textureLocation,0);

  // draw geometry
  gl.enableVertexAttribArray(vertexLocation);
  gl.enableVertexAttribArray(texCoordLocation);


  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
  gl.vertexAttribPointer(vertexLocation,3,gl.FLOAT,gl.FALSE,0,0);

  gl.bindBuffer(gl.ARRAY_BUFFER, triangleTexCoordBuffer);
  gl.vertexAttribPointer(texCoordLocation,2,gl.FLOAT,gl.FALSE,0,0);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, theTexture);


  gl.drawArrays(gl.TRIANGLES,0,6);

  // disable all
  gl.disableVertexAttribArray(vertexLocation);
  gl.disableVertexAttribArray(texCoordLocation);
  gl.useProgram(null);
}
```

Vidéo

- balise vidéo HTML5 :

```
<video id="myVideo" src="repas.webm" autoplay="true" controls="true">
</video>
```

- Récupération de la texture à chaque image :

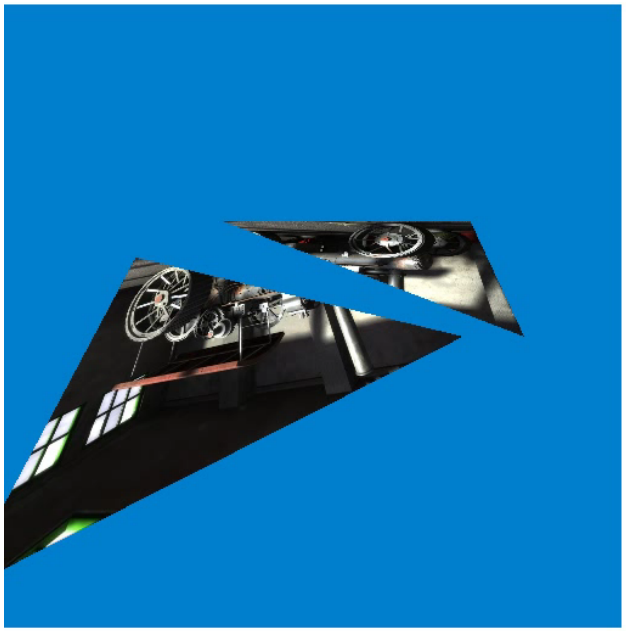
```
/** ********************************************* */
/** update data (called by loop() )
 *
 */
function updateData() {
  var imageData=document.getElementById("myVideo");
  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D,theTexture);
  gl.texImage2D(gl.TEXTURE_2D,0,gl.RGB,gl.RGB,gl.UNSIGNED_BYTE,imageData);
}
```

5 ModelView et Projection

Obtenir

```
function initData() {
...
  projection.setFrustum(-0.1,0.1,-0.1,0.1,0.1,1000);
...
}
```

```
function updateData() {
  ...
  angle+=0.01;
  modelview.setIdentity();
  modelview.translate(0,0,-4);
  modelview.rotateX(angle);
}
```



Définir un constructeur Mat4

```
function Mat4() {
  this.fv = new Float32Array(16);
}
Mat4.prototype.setIdentity = function() {
    this.fv[0]=1.0;  this.fv[4]=0.0;  this.fv[8] =0;   this.fv[12]=0.0;
    this.fv[1]=0.0;  this.fv[5]=1.0;  this.fv[9] =0.0; this.fv[13]=0.0;
    this.fv[2]=0.0;  this.fv[6]=0.0;  this.fv[10]=1.0; this.fv[14]=0.0;
    this.fv[3]=0.0;  this.fv[7]=0.0;  this.fv[11]=0.0; this.fv[15]=1.0;
  };
Mat4.prototype.copy = function() {
    var res=new Mat4();
    for(i=0;i<16;i++) {res.fv[i]=this.fv[i];}
    return res;
  };
Mat4.prototype.setFrustum = function(left,right,bottom,top,near,far) {
    this.fv[0]=2.0*near/(right-left);   this.fv[4]=0.0;                  this.fv[8] =(right+left)/(right-left); this.fv[12]=0.0;
    this.fv[1]=0.0;                     this.fv[5]=2.0*near/(top-bottom); this.fv[9] =(top+bottom)/(top-bottom); this.fv[13]=0.0;
    this.fv[2]=0.0;                     this.fv[6]=0.0;                  this.fv[10]=-(far+near)/(far-near);    this.fv[14]=-2.0*f
    this.fv[3]=0.0;                     this.fv[7]=0.0;                  this.fv[11]=-1.0;                      this.fv[15]=0.0;
  };
```

Modelview et projection dans le shader

```
<script id="hello-vs" type="x-shader/x-vertex">
    attribute vec4 vertex;
    attribute vec2 texCoord;

    uniform mat4 modelview,projection;

    varying vec2 fTexCoord;

    void main(void) {
  fTexCoord = texCoord;
  gl_Position = projection*modelview*vertex;
    }
</script>
```

Passer les matrices au shader

```
function drawScene() {
...
  var modelviewLocation=gl.getUniformLocation(helloProgramShader,'modelview');
  var projectionLocation=gl.getUniformLocation(helloProgramShader,'projection');
  // set up uniform
  gl.uniform1i(textureLocation,0);
  gl.uniformMatrix4fv(modelviewLocation,gl.FALSE,modelview.fv);
  gl.uniformMatrix4fv(projectionLocation,gl.FALSE,projection.fv);
...
}
```

6 Evénements

Définir des callbacks sur élément HTML5

```
function main(event) { // called by event load from window
    canvas=document.getElementById("webglCanvas");
...
  // callback for mouse events
  canvas.addEventListener('mousedown',handleMouseDown,false);
  canvas.addEventListener('mousemove',handleMouseMove,false);
  canvas.addEventListener('mouseup',handleMouseUp,false);

  loop();
...
}
```

```
function handleMouseDown(event) {
  // get the mouse coordinates relative to canvas
  oldMouseX = event.layerX-canvas.offsetLeft;
  oldMouseY = (canvas.height-1.0)-(event.layerY-canvas.offsetTop);
  mouseDown=true;
}
```

7 Quelques remarques sur Javascript

Prototypes

- Toute fonction peut être un constructeur d'objets.

```
function f() {
  this.coucou="toto";
}

var a=new f(); // a est une instance de f
console.log(a.coucou);
```

- Toute fonction est un objet.
- Toute fonction possède une propriété `prototype`.
- Toutes les instances d'une fonction partagent la propriété `prototype` (implicitement).

Exemple

```
function f() {
  this.coucou="toto";
}

f.prototype.bipbip="titi";

var a=new f();
var b=new f();

console.log(a.coucou); // non partagé
console.log(b.coucou);
console.log(a.bipbip); // partagé
console.log(b.bipbip);

a.bipbip="blop"; // ! attention : création de la propriété bipbip pour a !
console.log(b.bipbip); // le bipbip partagé (i.e. du prototype).
console.log(a.bipbip); // le bipbip propre à a
```

Méthode

```
function f() {
  this.prop1="valeur"; // la propriété sera créée pour chaque instance
}

f.prototype.methode1=function() {console.log(this.prop1);} // methode1 sera partagée

var a=new f();
// a.methode1() => appel à la méthode (partagée)
// a.prop1 => valeur de prop1 de a (non partagée).
```