
Synthèse d'Images

ESIR 2 IN
Version 2012

Rémi Cozot, Kadi Bouatouch, Fabrice
Lamarche

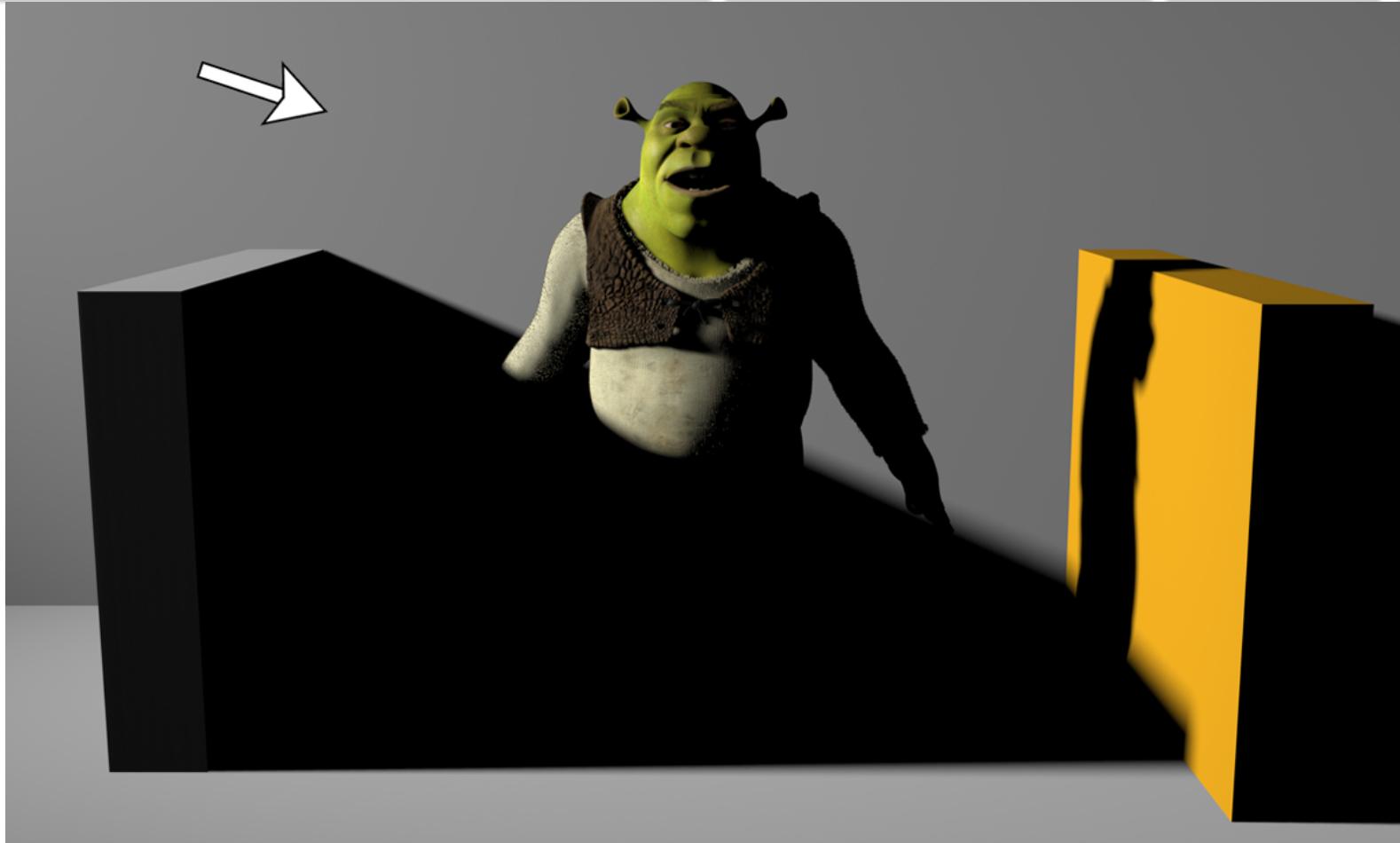
Plan général

Synthèse d'images	Synthèse d'images	Synthèse d'images	Synthèse d'images	Applications
Temps réel openGL Illumination directe R. Cozot K. Bouatouch	Photo réalistes Lancer de Rayons K. Bouatouch F. Lamarche	Photo réalistes Illumination globale K. Bouatouch R. Cozot	Animation F. Lamarche	Effets spéciaux R. Cozot E. Marchand
Applications	Jeux Vidéos			R. Cozot F. Lamarche

Plan Synthèse d'Image Temps-Réel

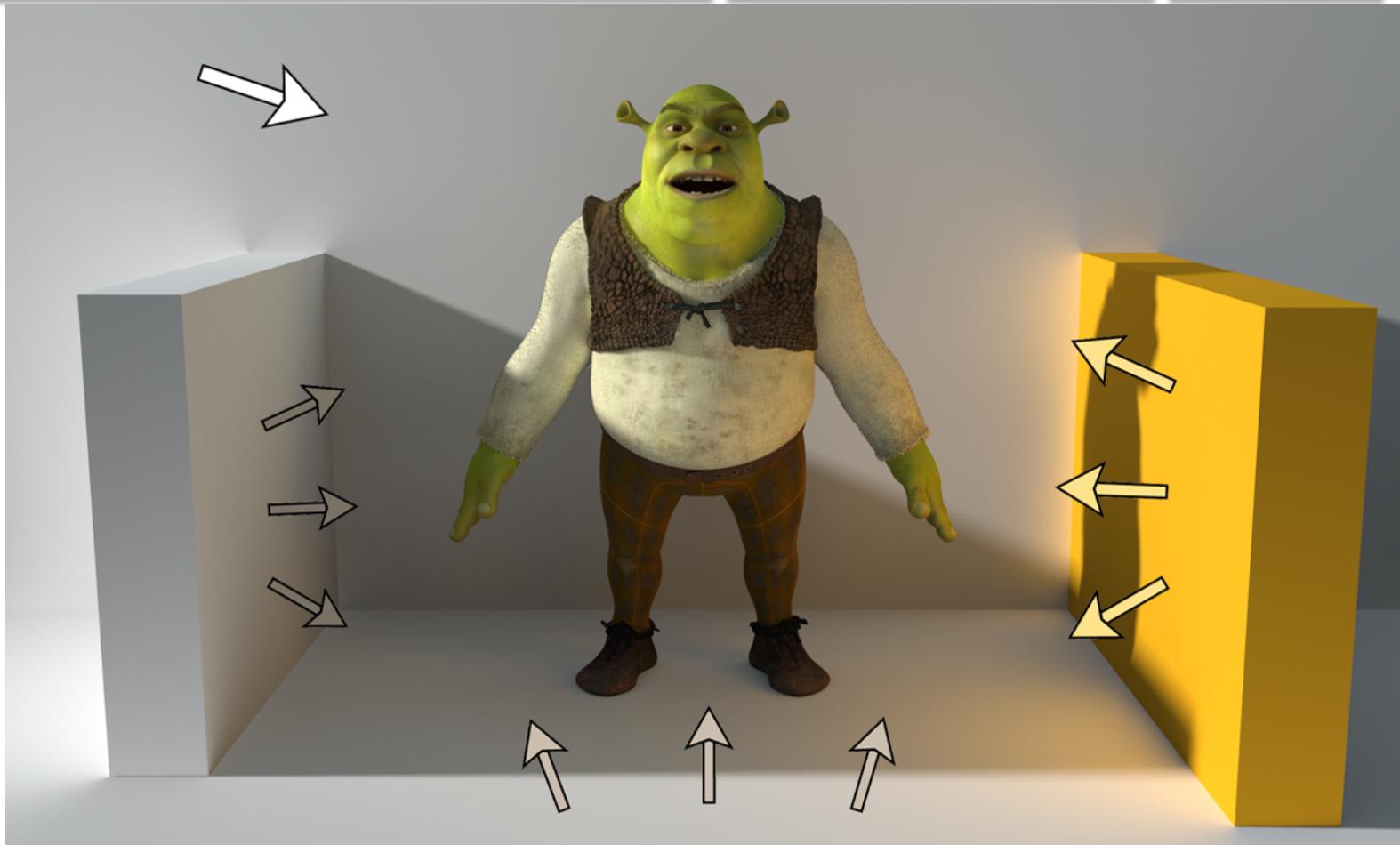
- Vers une simulation de l'éclairage
- Objectifs
- OpenGL
- Pipeline graphique
- Architecture d'un programme OpenGL
- Modélisation polyédrique

Vers une simulation de l'éclairage : illumination directe



Source : Dreamworks Animation SKG

Vers une simulation de l'éclairage : illumination globale



Source : Dreamworks Animation SKG

Vers une simulation de l'éclairage : illumination globale



Source : crytek (CryEngine 3)

Vers une simulation de l'éclairage : illumination directe

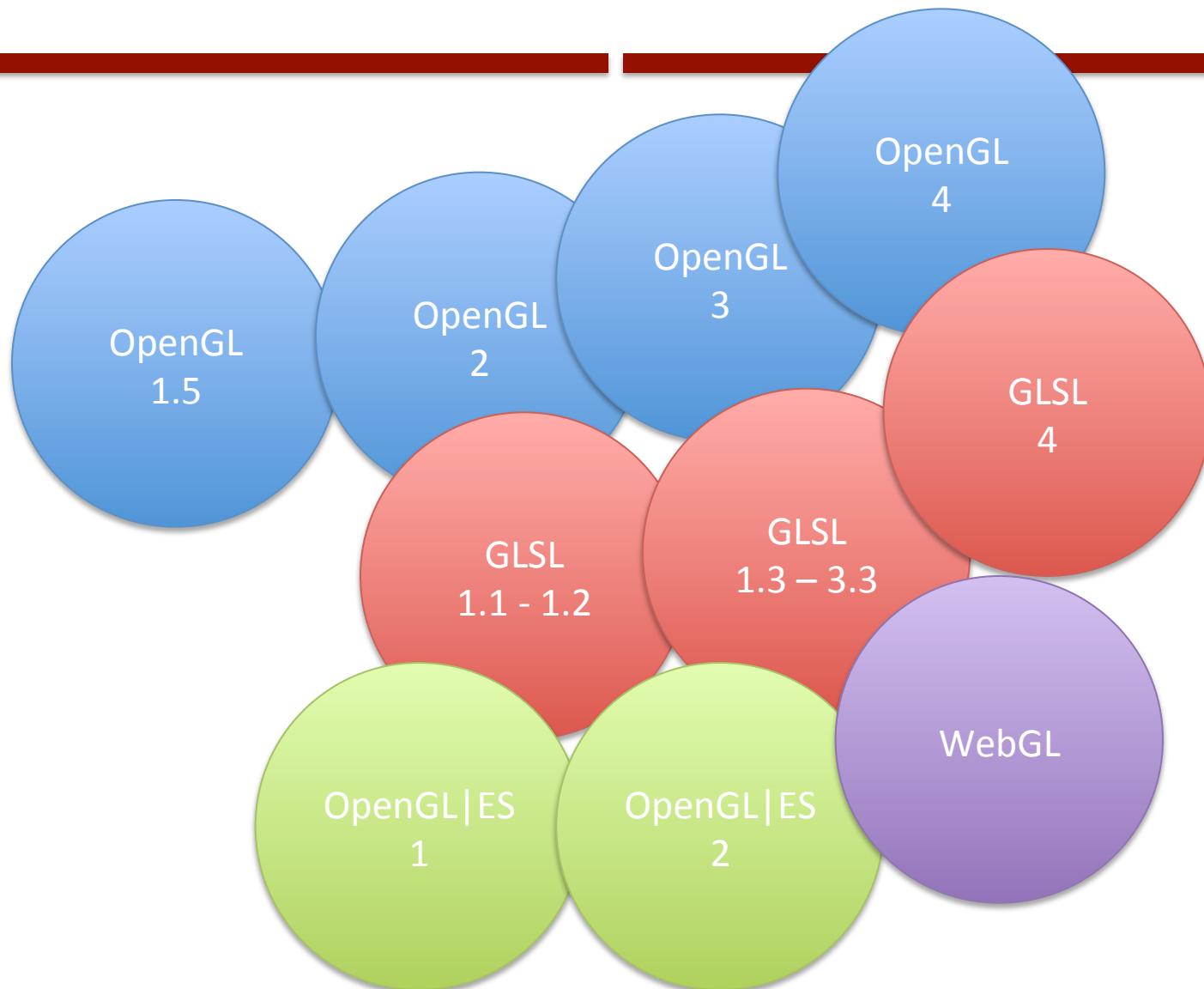


Source : crytek (CryEngine 3)

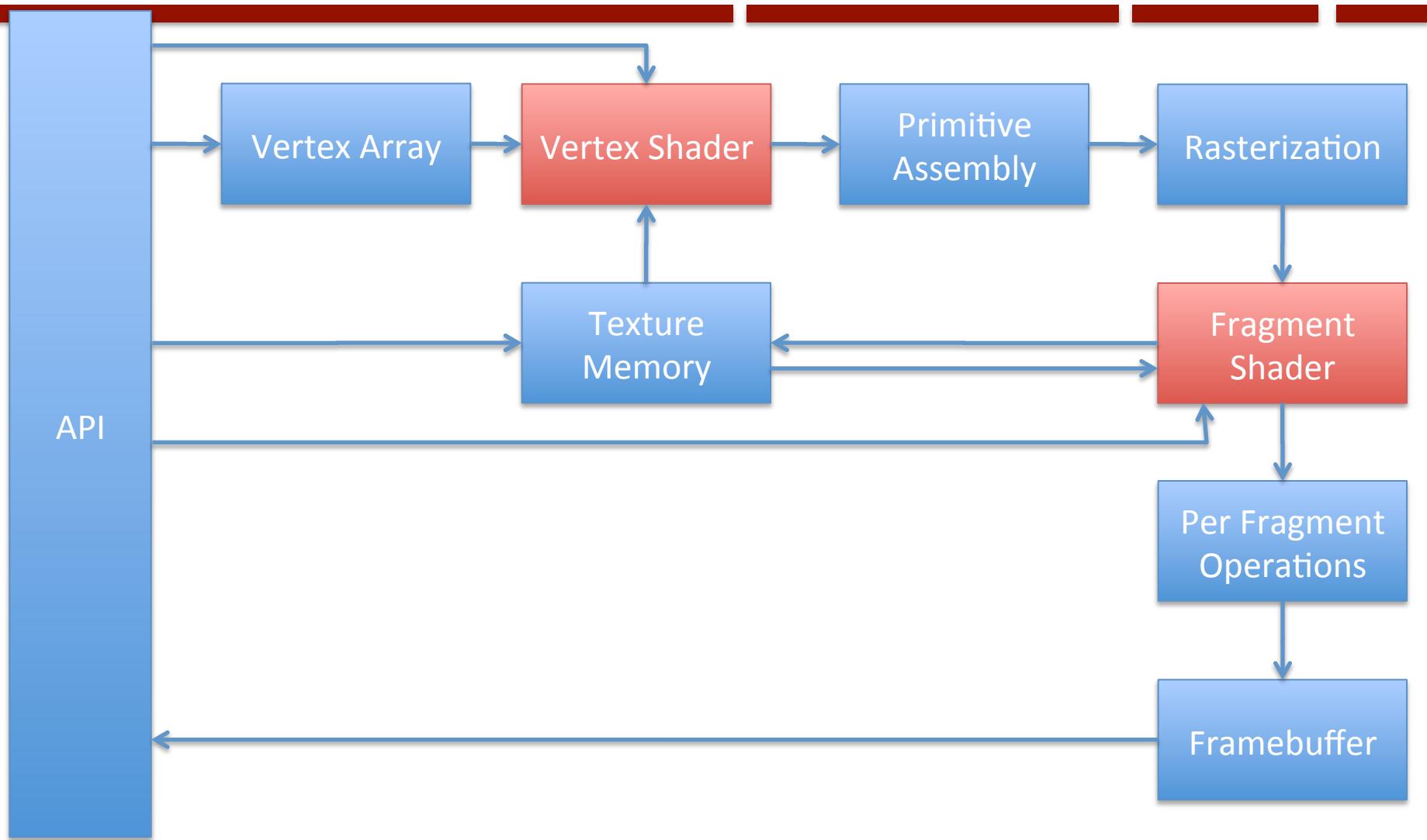
Objectifs

- Modélisation
 - Polyédrique
 - Modèle d'éclairage direct
- Technique
 - OpenGL, WebGL, GLSL
 - Programmation de shaders

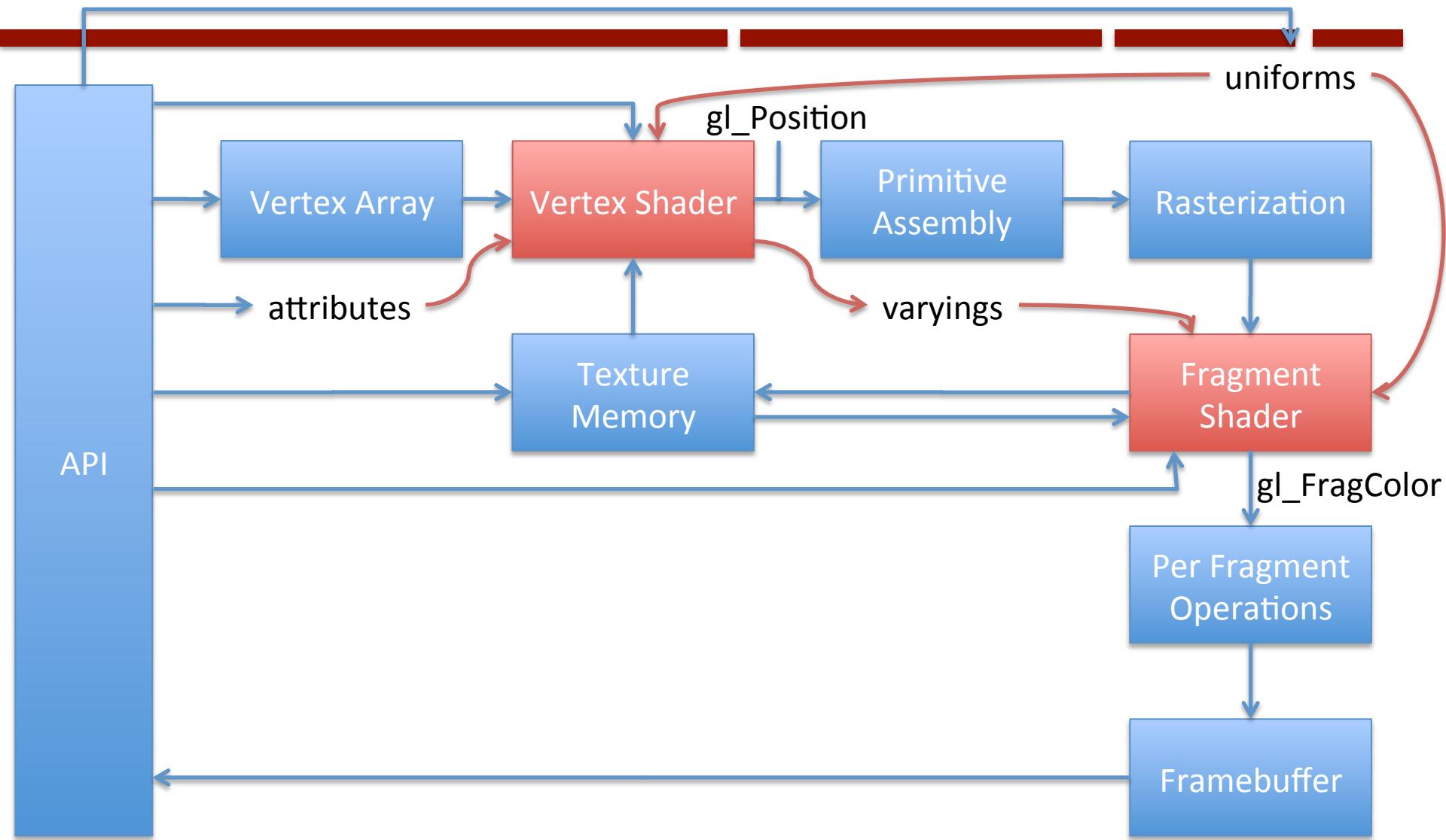
OpenGL – WebGL



Pipeline Graphique



Pipeline Graphique



Architecture d'un programme (Web)GL

- Initialisation
 - “contexte” OpenGL
 - “shaders”
 - “buffers”
- Dessiner la scène

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

- Initialisation
 - “contexte” OpenGL
 - “shaders”
 - “buffers”
- Dessiner la scène

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

```
var myPositionBuffer;

function initBuffers() {
    myPositionBuffer= gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);
    vertices = [
        0.0,  0.0,  0.0,
        1.0,  0.0,  0.0,
        0.0,  1.0,  0.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    myPositionBuffer.itemSize = 3;
    myPositionBuffer.numItems = 4;
}
```

Architecture d'un programme (Web)GL

```
var myPositionBuffer;  
  
function initBuffers() {  
    myPositionBuffer= gl.createBuffer();  
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);  
    vertices = [  
        0.0,  0.0,  0.0,  
        1.0,  0.0,  0.0,  
        0.0,  1.0,  0.0,  
    ];  
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(ve  
        gl.STATIC_DRAW);  
    myPositionBuffer.itemSize = 3;  
    myPositionBuffer.numItems = 4;  
}
```

Déclaration du buffer

Création du buffer

myPositionBuffer est activé

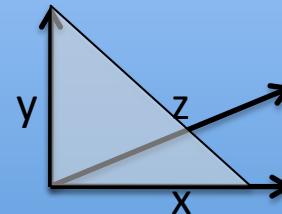
Son "type" est
ARRAY_BUFFER
(tableau de sommets)

Architecture d'un programme (Web)GL

```
var myPositionBuffer;

function initBuffers() {
    myPositionBuffer= gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);
    vertices = [
        0.0,  0.0,  0.0,
        1.0,  0.0,  0.0,
        0.0,  1.0,  0.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER,
        gl.STATIC_DRAW);
    myPositionBuffer.itemSize = 3;
    myPositionBuffer.numItems = 4
}
```

Création d'une liste (javascript)
avec les coordonnées des
sommets (ici les quatre sommets
d'un rectangle)



Architecture d'un programme (Web)GL

```
var myPositionBuffer;

function initBuffers() {
    myPositionBuffer= gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);
    vertices = [
        0.0,  0.0,  0.0,
        1.0,  0.0,  0.0,
        0.0,  1.0,  0.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
                 gl.STATIC_DRAW);
    myPositionBuffer.itemSize = 3;
    myPositionBuffer.numItems = 3;
}
```

Affectation des données (au buffer courant (dernier gl.bindBuffer))

Conversion de la liste Javascript en tableau OpenGL

Description des données

3 vecteurs

3 coordonnées par vecteurs

Architecture d'un programme (Web)GL

- Initialisation
 - “contexte” OpenGL
 - “shaders”
 - “buffers”
- Dessiner la scène

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

```
var vertexPos ;  
function initShaders() {  
    var fragmentShader = getShader(gl, "shader-fs");  
    var vertexShader = getShader(gl, "shader-vs");  
  
    shaderProgram = gl.createProgram();  
    gl.attachShader(shaderProgram, vertexShader);  
    gl.attachShader(shaderProgram, fragmentShader);  
    gl.linkProgram(shaderProgram);  
  
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {  
        alert("Could not initialise shaders");}  
  
    gl.useProgram(shaderProgram);  
  
    vertexPos = gl.getAttributeLocation( shaderProgram, "aVertexPosition");  
    gl.enableVertexAttribArray(shaderProgram.vertexPos);  
}  
}
```

Architecture d'un programme (Web)GL

```
var vertexPos ;  
function initShaders() {  
    var fragmentShader = getShader(gl, "shader-fs");  
    var vertexShader = getShader(gl, "shader-vs");  
  
    shaderProgram = gl.createProgram();  
    gl.attachShader(shaderProgram, vertexShader);  
    gl.attachShader(shaderProgram, fragmentShader);  
    gl.linkProgram(shaderProgram);  
  
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {  
        alert("Could not initialise shaders");}  
  
    gl.useProgram(shaderProgram);  
  
    vertexPos = gl.getAttributeLocation( shaderProgram, "aVertexPosition");  
    gl.enableVertexAttribArray(shaderProgram.vertexPos);  
}
```

Lecture des scripts (GLSL) des
shaders (vertex et fragment)

Compilation (à la volée)

Architecture d'un programme (Web)GL

```
var vertexPos ;  
function initShaders() {  
    var fragmentShader = getShader(gl, "shader-fs");  
    var vertexShader = getShader(gl, "shader-vs");  
  
    shaderProgram = gl.createProgram();  
    gl.attachShader(shaderProgram, vertexShader);  
    gl.attachShader(shaderProgram, fragmentShader);  
    gl.linkProgram(shaderProgram);  
  
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))  
        alert("Could not initialise shaders");  
  
    gl.useProgram(shaderProgram);  
  
    vertexPos = gl.getAttributeLocation(shaderProgram, "vertexPos");  
    gl.enableVertexAttribArray(shaderProgram, vertexPos);  
}  
}
```

Program = vertex+fragment shader

Création du Program

Attribution des vertex et fragment shaders compilés

Link du Program

Architecture d'un programme (Web)GL

```
var vertexPos ;  
function initShaders() {  
    var fragmentShader = getShader(gl.  
    var vertexShader = getShader(gl.  
  
    shaderProgram = gl.createProgram();  
    gl.attachShader(shaderProgram,  
    Program shader actif  
    gl.linkProgram(shaderProgram);  
  
    if (!gl.getProgramParameter(shaderProgram,  
        alert("Could not initialise s  
  
    gl.useProgram(shaderProgram);  
  
    vertexPos = gl.getAttributeLocation(shaderProgram, "aVertexPosition");  
    gl.enableVertexAttribArray(vertexPos);  
  
}  
}
```

Il reste à définir l'entrée du vertex shader !

Rappel : entrées d'un vertex shader

- attributes
- uniforms

Récupération de l'adresse (location) de l'attribut "aVertexPosition" du program shader

Autorise l'utilisation d'un tableau de type VertexArray

Architecture d'un programme (Web)GL

- Initialisation
 - “contexte” OpenGL
 - “shaders”
 - “buffers”
- Dessiner la scène

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

```
var gl;  
function initGL(canvas) {  
    try {  
        gl = canvas.getContext("experimental-webgl");  
        // récupération taille du canvas  
        gl.viewportWidth = canvas.width;  
        gl.viewportHeight = canvas.height;  
    } catch (e) {}  
    if (!gl) {  
        alert("Echec initialisation WebGL");  
    }  
}
```

Architecture d'un programme (Web)GL

- Initialisation
 - “contexte” OpenGL
 - “shaders”
 - “buffers”
- Dessiner la scène

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

```
function drawScene() {  
    // taille du viewport OpenGL  
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);  
    // mise à zero du frame buffer (image) et du z-buffer  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    // rendre le buffer myPositionBuffer actif  
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);  
  
    // lier l'attribut du vertex shader avec le buffer  
    gl.vertexAttribPointer(vertexPos,myPositionBuffer.itemSize,  
        gl.FLOAT, false, 0, 0);  
  
    // dessiner le « buffer »  
    gl.drawArrays(gl.TRIANGLES, 0, myPositionBuffer.numItems);  
}
```

Architecture d'un programme (Web)GL

```
function drawScene() {  
    // taille du viewport OpenGL  
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);  
    // mise à zero du frame buffer (image) et du z-buffer  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    // rendre le buffer myPositionBuffer actif  
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);  
  
    // lier l'attribut du vertex shader à ce buffer  
    gl.vertexAttribPointer(vertexPos, myPositionBuffer.numComponents,  
        gl.FLOAT, false, 0, 0);  
  
    // dessiner le « buffer »  
    gl.drawArrays(gl.TRIANGLES, 0, myPositionBuffer.numItems);  
}
```

Le buffer image et le buffer de profondeur doivent être remis à zéro pour chaque nouvelle image

Architecture d'un programme (Web)GL

```
function drawScene() {  
    // taille du viewport OpenGL  
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);  
    // mise à zero du frame buffer (image)  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    // rendre le buffer myPositionBuffer actif  
    gl.bindBuffer(gl.ARRAY_BUFFER, myPositionBuffer);  
  
    // lier l'attribut du vertex shader avec le buffer  
    gl.vertexAttribPointer(vertexPos, myPositionBuffer.itemSize,  
        gl.FLOAT, false, 0, 0);  
  
    // dessiner le « buffer »  
    gl.drawArrays(gl.TRIANGLES, 0, myPositionBuffer.numItems);  
}
```

3 étapes :

1. Définir le buffer (vertex array) actif
2. Lier le buffer avec l'entrée du shader acrif
3. Dessiner le buffer

Architecture d'un programme (Web)GL

- Code des shaders

```
// initialisation canvas OpenGL
initGL(canvas);

// initialisation shaders
initShaders();

// initialisation buffers
initBuffers();

// couleurs du fond
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// activation z-buffer
gl.enable(gl.DEPTH_TEST);

// dessin
drawScene();
```

Architecture d'un programme (Web)GL

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;

    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

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

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

Architecture d'un programme (Web)GL

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;

    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

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

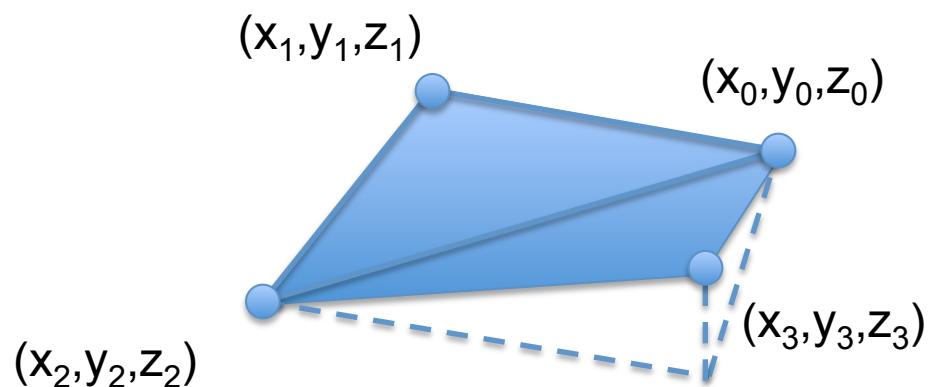
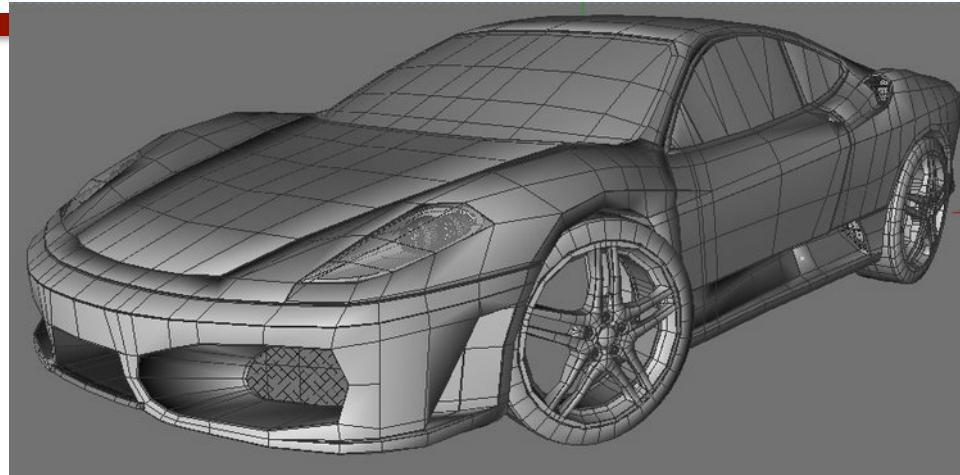
    void main(void) {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```

Au-delà de la technique : Modélisation 3D

- Modélisation 3D
 - Représentation d'un monde (scène)
- Deux questions fondamentales
 - Quoi
 - Que doit on modéliser/représenter afin de pouvoir calculer une image de synthèse ?
 - Comment
 - Comment représenter (en machine) les différents éléments ?

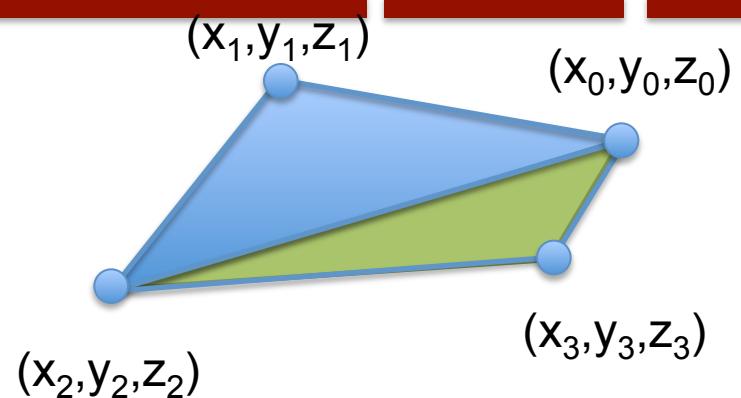
Modélisation polyédrique

- Modélisation polyédrique
- Approximation de la surface d'un objet par un ensemble de faces polygonales
- Faces triangulaires
 - Non ambiguëes
 - 3 points définissent une triangle plan



Modélisation polyédrique

- Modélisation
 - Géométrie
 - Ensemble de triangles
 - Représentation « compacte »
 - Liste de sommets (vertex)
 - Liste de triangles
 - » Par indirection
 - Apparence



# Liste de sommets	
0	(x ₀ ,y ₀ ,z ₀)
1	(x ₁ ,y ₁ ,z ₁)
2	(x ₂ ,y ₂ ,z ₂)
3	(x ₃ ,y ₃ ,z ₃)

# Liste des triangles	
0,1,2	
0,2,3	

Modélisation polyédrique

```
// dans la partie initialisation des buffers

var cubeVertexPositionBuffer;
var cubeVertexIndexBuffer;

cubeVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
vertices = [
  -1.0, -1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0,  1.0,  1.0, // Front face
  -1.0, -1.0, -1.0,  1.0, -1.0,  1.0,  1.0, -1.0,  1.0, -1.0, // Back face
  -1.0,  1.0, -1.0, -1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0, // Top face
  -1.0, -1.0, -1.0,  1.0, -1.0, -1.0,  1.0, -1.0, -1.0,  1.0, // Bottom face
  1.0, -1.0, -1.0,  1.0, -1.0,  1.0,  1.0,  1.0, -1.0,  1.0, // Right face
  -1.0, -1.0, -1.0, -1.0, -1.0,  1.0, -1.0,  1.0,  1.0, -1.0, // Left face
];

gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 24;
```

Modélisation polyédrique

```
// dans la partie initialisation des buffers -suite
```

```
cubeVertexIndexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
```

```
var cubeVertexIndices = [
  0, 1, 2,          0, 2, 3,      // Front face
  4, 5, 6,          4, 6, 7,      // Back face
  8, 9, 10,         8, 10, 11,    // Top face
  12, 13, 14,        12, 14, 15,   // Bottom face
  16, 17, 18,        16, 18, 19,   // Right face
  20, 21, 22,        20, 22, 23   // Left face
];
```

```
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
  gl.STATIC_DRAW);
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 36;
```

Création du buffer
(pour mes indices)
Type : **gl.ELEMENT_ARRAY_BUFFER**

Affectation des données
(au buffer courant)

Modélisation polyédrique

```
// dans la partie initialisation des shaders
```

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

```
// dans la partie dessin de la scène
```

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

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

Modélisation polyédrique

```
// dans la partie initialisation des shaders
```

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

- 1 – lier l'entrée du vertex shader avec le tableau des positions
 - 1.1 – rendre le buffer « actif »
 - 1.2 – lien buffer / vertex shader
- 2 – dessiner à partir du tableau d'indices
 - 2.1 – rendre le buffer d'indice « actif »
 - 2.2 – dessiner

```
// dans la partie dessin de la scène
```

```
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);  
gl.vertexAttribPointer(  
    vertexPositionAttribute, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);  
  
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);  
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
```

Éclairage direct

- Modèle d'apparence
 - La géométrie : nécessaire mais pas suffisant
 - Apparence
 - Couleur, matière et lumière
 - BRDF (Bidirectional Reflectance Distribution Function)
 - Source de lumière
 - Ombre et ombrage



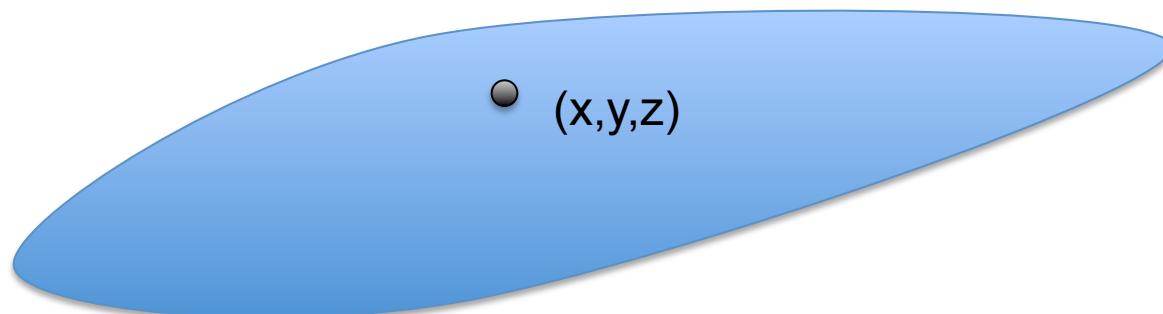
Éclairage direct



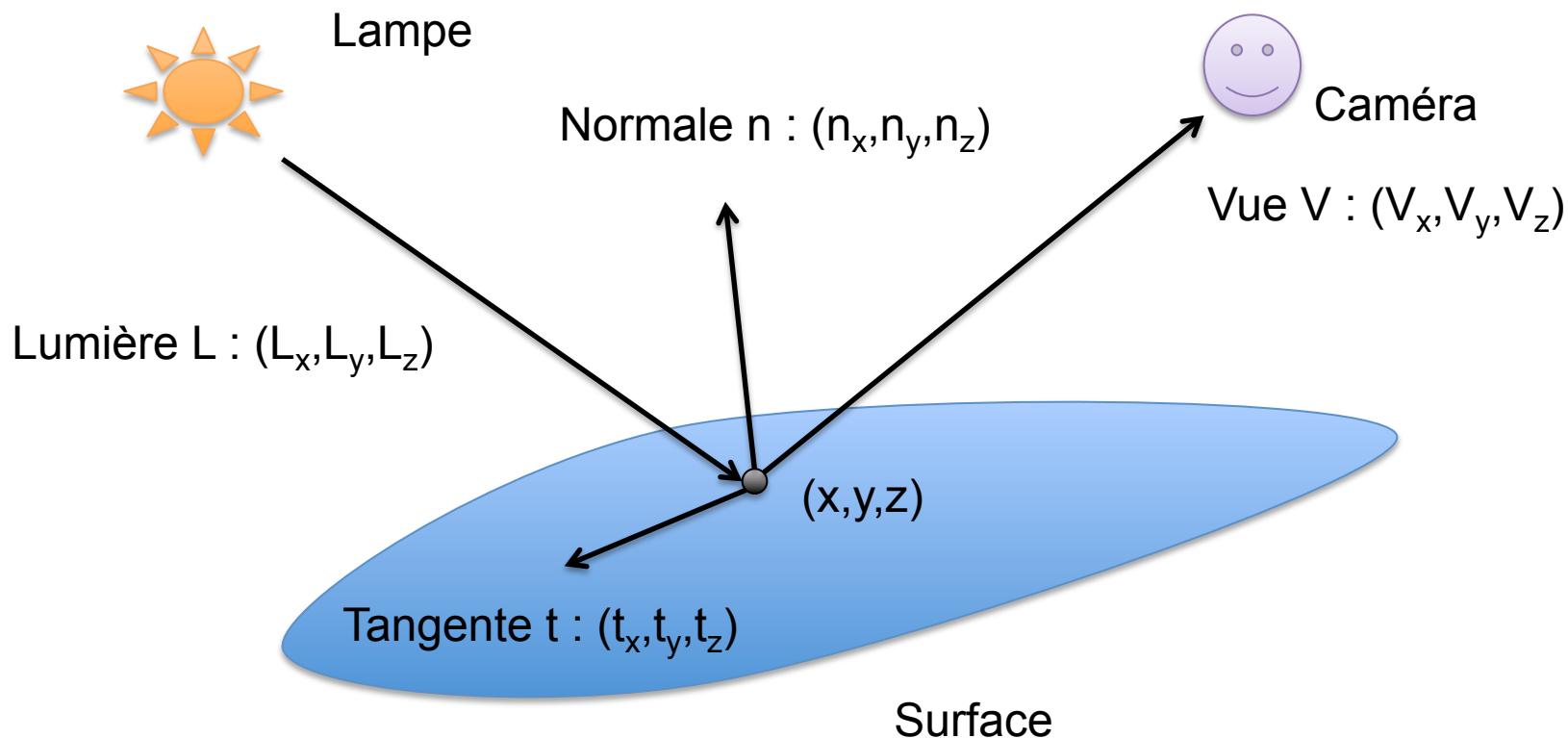
Lampe



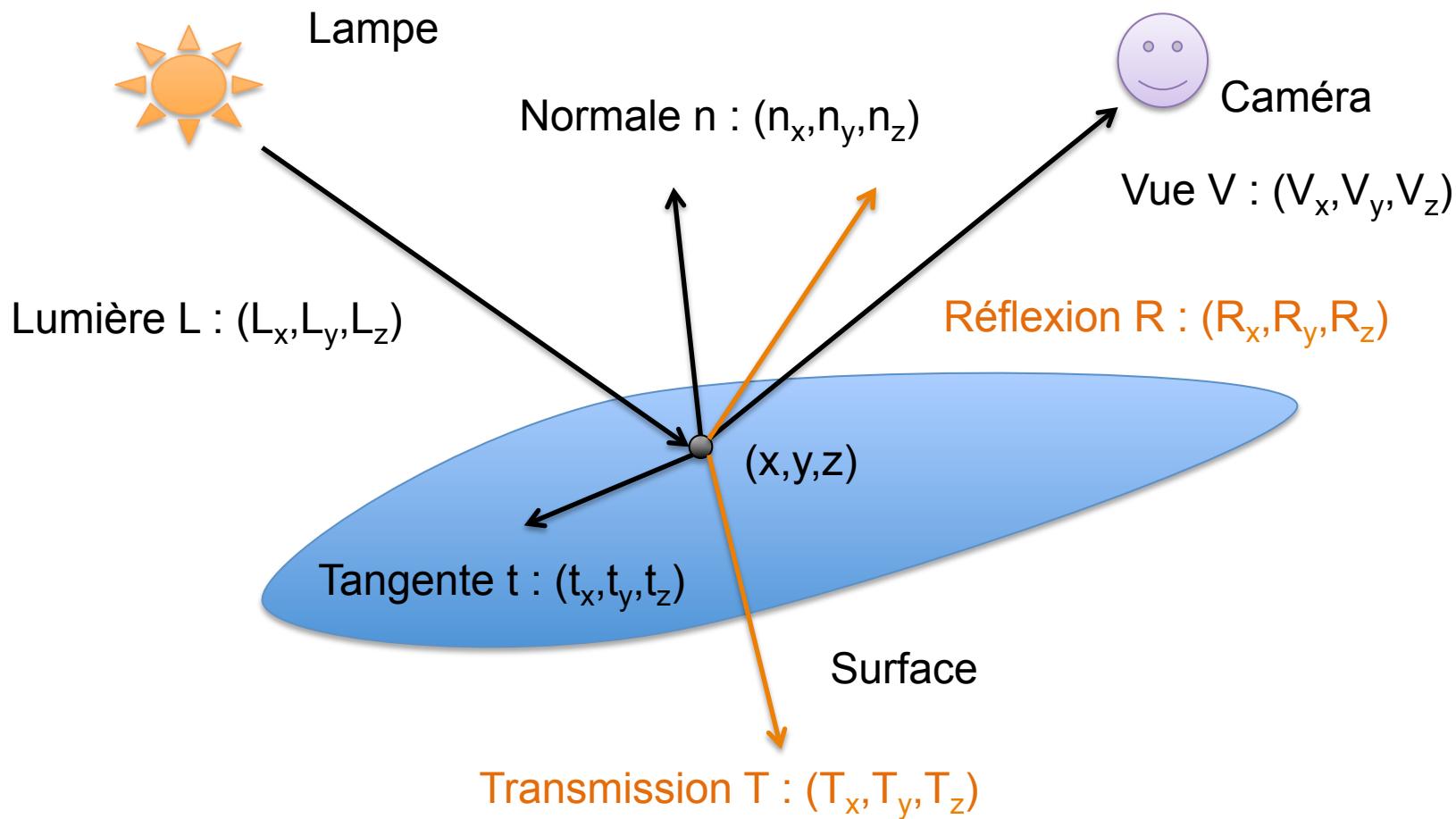
Caméra



Éclairage direct



Éclairage direct



Éclairage direct

- Quelles phénomènes modéliser ?
 - Matériaux :
 - Diffus, « brillant » (spéculaire)
 - Modèle de Phong

$$\left\{ \begin{array}{l} I = I_a + I_d + I_s \\ I_d = K_d I_{obj} I_{source} \cos(\text{ang}(n, L)) / d^2 \\ I_s = K_s I_{obj} I_{source} \cos^m(\text{ang}(R, V)) / d^2 \end{array} \right.$$

Éclairage direct

- Shader « Diffus »
 - Entrées
 - Objet
 - Coordonnées
 - Normales
 - Couleur
 - Lampes
 - Position
 - Couleur

Éclairage direct

```
// vertex shader: diffuse per vertex

// inputs - attributes
attribute vec3 aVertexPosition;
attribute vec3 aVertexNormal;

// inputs - uniform
uniform vec3 uPointLightingLocation;
uniform vec3 uPointLightingColor;
uniform vec3 uDiffuseColor;

// outputs - varying
varying vec3 vLightWeighting;

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

    vec3 lightDirection = normalize(uPointLightingLocation - gl_Position.xyz);
    float directionalLightWeighting = max(dot(aVertexNormal, lightDirection), 0.0);
    vLightWeighting = uDiffuseColor*uPointLightingColor*directionalLightWeighting;
}
```

Éclairage direct

```
// vertex shader: diffuse per vertex

// inputs - varying come from vertex shader output
varying vec3 vLightWeighting;

void main(void) {
    gl_FragColor = vec4(vLightWeighting, 1.0);
}
```

- Focus sur les **varying**
 - Sortie du vertex shader
 - Valeur calculée à chaque sommet
 - Entrée du fragment shader
 - Valeur interpolée en chaque fragment
- Attention
 - L'interpolation ne conserve pas la normalisation

Éclairage direct

```
function createProgram(fragmentShader, vertexShader) {  
    // program from compiled shaders  
    var program = gl.createProgram();  
    gl.attachShader(program, vertexShader);  
    gl.attachShader(program, fragmentShader);  
    gl.linkProgram(program);  
  
    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {alert("Error");}  
  
    // attributes  
    program.vertexPositionAttribute = gl.getAttribLocation(program, "aVertexPosition");  
    gl.enableVertexAttribArray(program.vertexPositionAttribute);  
  
    program.vertexNormalAttribute = gl.getAttribLocation(program, "aVertexNormal");  
    gl.enableVertexAttribArray(program.vertexNormalAttribute);  
  
    // uniforms  
    program.pointLightingLocationUniform =  
        gl.getUniformLocation(program, "uPointLightingLocation");  
    program.pointLightingColorUniform =  
        gl.getUniformLocation(program, "uPointLightingColor");  
  
    return program;  
}
```

Éclairage direct

```
var cubeVertexPositionBuffer; var cubeVertexNormalBuffer; var cubeVertexIndexBuffer;

function initBuffers() {
    // creation des buffers
    // 2 array_buffers pour positions et normales + 1 element_array_buffer
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [-1.0, -1.0, 1.0, ..., -1.0, 1.0, -1.0];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3; cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [ 0.0, 0.0, 1.0, ..., -1.0, 0.0, 0.0 ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3; cubeVertexNormalBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices = [0, 1, 2..., 20, 22, 23];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
        gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1; cubeVertexIndexBuffer.numItems = 36;
}
```

Éclairage direct

```
function drawScene() {  
    // init viewport and clear buffers  
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    // active shader  
    gl.useProgram(program);  
  
    // set uniforms gl.uniform+SIZE+FORMAT SIZE=1,2,3,4 FORMAT = i, f  
    gl.uniform3f(program.pointLightingLocationUniform, 25.0, 25.0, 0.0) ;  
    gl.uniform3f(program.pointLightingColorUniform, 128.0, 64.0, 64.0) ;  
  
    // vertex and elements arrays  
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);  
    gl.vertexAttribPointer(program.vertexPositionAttribute,  
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);  
  
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);  
    gl.vertexAttribPointer(program.vertexNormalAttribute,  
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);  
  
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);  
    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);  
}
```

Point d'étape OpenGL (1/3)

- « ArrayBuffer »
 - Sommets
 - Normales
 - Couleurs
 - Coordonnées de textures
 - Etc.
- « Element ArrayBuffer »
 - Indices des sommets pour décrire les sommets

```
// création
// array buffer

var Buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER,Buffer);
var data= [...];
gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(data), gl.STATIC_DRAW);
Buffer.itemSize = 3;
Buffer.numItems = 24;

// element array buffer
Ind= gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
Ind);
var idata= [...];
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new
Uint16Array(idata),gl.STATIC_DRAW);
Ind.itemSize = 1;
Ind.numItems = 36;
```

Point d'étape OpenGL (2/3)

- Création (code slide précédent)
- Récupération du *pointeur* sur l'*attribute* dans le *program shader*
- Dessin
 - Activation buffer
 - Array, Element
 - Connexion buffer / attribute
 - Array
 - Dessin

```
// Connexion avec le program shader
// array buffer -> attributes
progshader.myAttribute =
gl.getAttributeLocation(
  program,
  "aMyAttrib");
gl.enableVertexAttribArray(
  program. progshader);
```

```
// connexion avec le program shader
// array buffer -> attributes
gl.bindBuffer(gl.ARRAY_BUFFER, Buffer );
gl.vertexAttribPointer(
  progshader.myAttribute, Buffer.itemSize,
  gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, Ind);
gl.drawElements(
  gl.TRIANGLES, Ind.numItems,
  gl.UNSIGNED_SHORT, 0);
```

Point d'étape OpenGL (3/3)

- Uniform
 - Donnée transmise du programme « principal » au *program shader*

```
// Connexion avec un Uniform
// du shader

var ptrUnif = gl.getUniformLocation(
    program,
    "uMyUniform");

// Définition d'une valeur pour un Uniform

gl.uniform3f(ptrUnif, 25.0, 25.0, 0.0) ;
```