Chapitre VIII : Vertex et Fragment Shaders

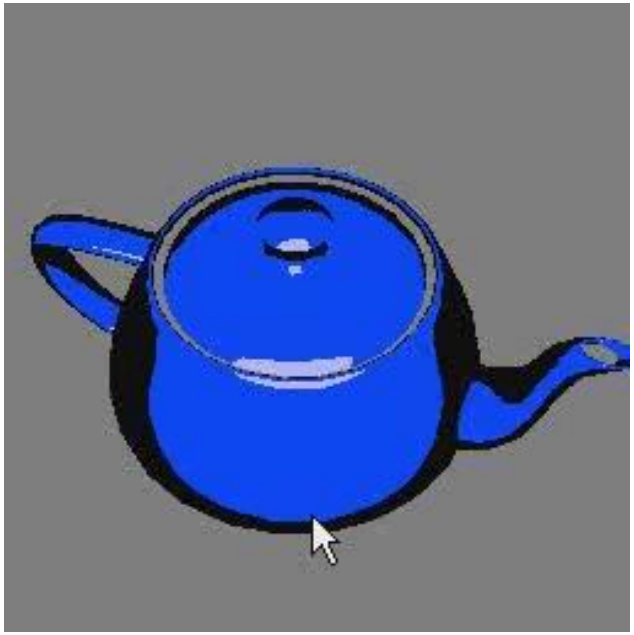
Programmation 3D

Fabrice Aubert
fabrice.aubert@lifl.fr

Licence/Master
USTL - IEEA

2008-2009

Introduction



Pipeline

- Les shaders sont des programmes qui se substituent à des phases du pipeline de rendu.
- Vertex shader : programme se substituant à la phase de transformation des sommets et au calcul d'éclairement/coordonnées de texture aux sommets.
- Pixel shader : programme qui se substitue à la phase d'attribution de la couleur et des coordonnées de texture aux fragments (lors de la rasterization).
- But : donner une grande liberté à la manipulation des sommets et des fragments... par ailleurs, les shaders sont exécutés par la carte graphique.

Langage

- Langage de haut niveau : Cg (nVidia, interfaçable avec Direct 3D et OpenGL), HLSL (direct3D), GLSL (OpenGL).
- Choix : GLSL (OpenGL 2.1/GLSL 1.2).
- Syntaxe très similaire au C.
- Manuel de référence : `http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf`
- IDE pour shaders : OpenGL shader designer (`http://www.opengl.org/sdk/tools/ShaderDesigner/`), RenderMonkey (`http://developer.amd.com/gpu/rendermonkey`).

Vertex shader

- Un vertex shader possède en entrée des **attributs** de sommets (i.e. tout ce qui concerne un sommet : position, couleur, coordonnée de texture, etc). L'utilisateur peut spécifier des attributs et/ou utiliser ceux prédéfinis par OpenGL (« built-in attributes » : `gl_Vertex`, `gl_Normal`, ...)
- Exemple : le code suivant affecte la position `gl_Vertex` et la couleur `gl_Color` du vertex shader :

```
glBegin(GL_POLYGON);
glColor3f(0.2,0.4,0.6);
glVertex3f(-1.0,1.0,2.0);
    ...
glEnd();
```

- Pour paramétrer le shader, on peut également utiliser et définir des variables qui seront constantes lors du tracé d'une primitive donnée : ce sont les variables dites **uniform**. Les états OpenGL sont par exemple accessibles grâce à des uniforms prédéfinis : `gl_ModelViewMatrix`, `gl_Light`, ...
- Les sorties d'un vertex shader (i.e. mise à jour de variables) sont qualifiées de **varying** : leurs valeurs vont être interpolées (moyenne pondérée entre les sommets) et passées au fragment shader.

Vertex Shader : exemple simple

Fichier `exemple_vertex.vert` :

```
void main() {
gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
}
```

- On se contente ici de faire ce qui est fait par défaut par le pipeline OpenGL : $P_{proj} = PROJECTION * MODELVIEW * P_{local}$.
- Les `gl_` sont des « variables »GLSL prédéfinies.
- `gl_Position` doit **obligatoirement** être affecté lors d'un vertex shader (sortie spéciale).
- Types et qualifications de ces variables prédéfinies (donc inutile de les déclarer) :

```
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
attribute vec4 gl_Vertex;
vec4 gl_Position; // variable spéciale
```

Pipeline

- Problème : toutes les opérations par défaut ne sont plus faites (substitution totale). Par exemple, si vous voulez simplement changer la génération des coordonnées de texture, il vous faut programmer les transformations, le calcul d'éclairement, ...
- Pas de connaissance des autres sommets dans un vertex shader.
- Cependant : accès à tous les états d'OpenGL (material, texture, ...) grâce aux uniform.

Fragment Shader

- Un fragment shader possède en entrée les valeurs interpolées des sommets : position, couleur, coordonnée de texture, depth. Ces variables sont qualifiées de `varying`.
- Les sorties du fragment shader consiste principalement à écrire sa couleur `gl_FragColor`.
- Pas de connaissance des autres fragments (pixels voisins par exemple).
- Accès aux états d'OpenGL (variables uniforms prédéfinies).
- Remarque : on manipule bien un fragment et non un pixel (par exemple, un fragment shader qui déplacerait un pixel sur l'écran n'a aucun sens).
- Pas de blending (opération faite après l'exécution du pixel shader).

Fragment Shader : exemple

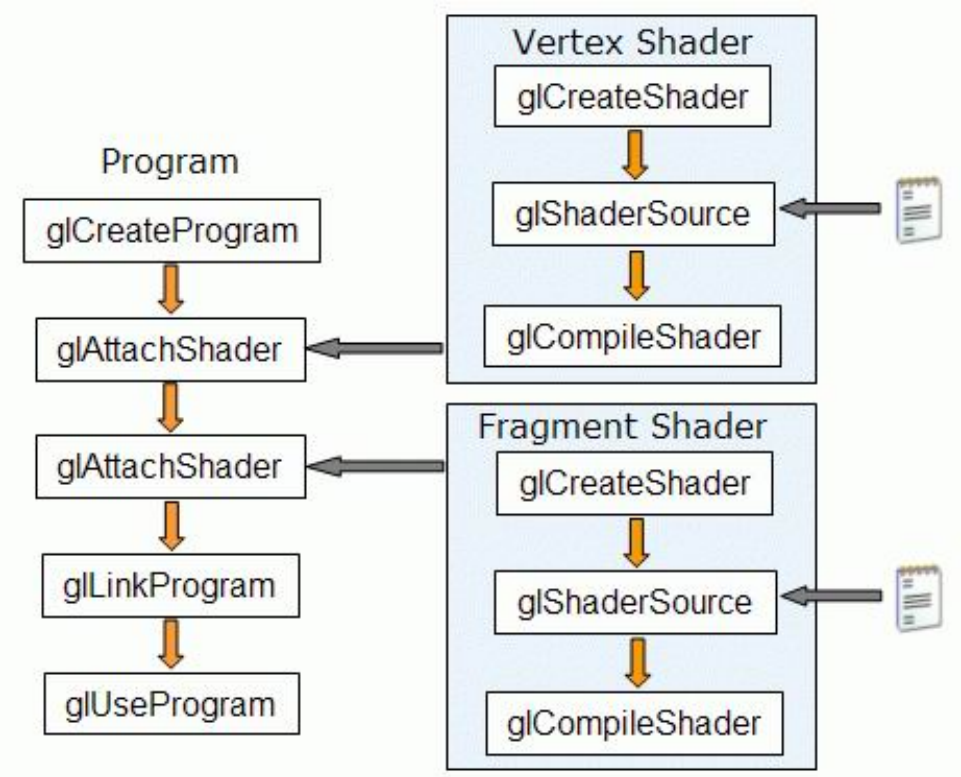
Exemple fichier : `exemple_pixel.frag`

```
void main() {
gl_FragColor = vec4(0.4,0.4,0.8,1.0);
}
```

- La variable `gl_FragColor` doit être obligatoirement affectée dans un fragment shader.

Intégration des shaders en OpenGL

- Dans ce cours : spécification OpenGL 2.1 (version actuelle 3.1 : pas de modification pour l'intégration).
- Etapes pour la définition des shaders :



Exemple

```
void myShader() {
    GLchar *v_source,*f_source;
    GLuint v,f,p;
  // creation handle :
    v = glCreateShader(GL_VERTEX_SHADER);
    f = glCreateShader(GL_FRAGMENT_SHADER);

// un programme = (au moins) un fragment et un vertex shader
    p = glCreateProgram();
    glAttachShader(p,v);
    glAttachShader(p,f);

    // lecture du GLSL source :
    v_source = textFileRead("exemple_vertex.vert");
    f_source = textFileRead("exemple_pixel.frag");
  // affectation des sources aux handles
    glShaderSource(v, 1, (const char **)&v_source,NULL);
    glShaderSource(f, 1, (const char **)&f_source,NULL);
    // compilation des sources
    glCompileShader(v);
    glCompileShader(f);

  // mise en place dans le pipeline :
    glLinkProgram(p);

    glUseProgram(p); // si p=0, pipeline par défaut
}
```

1 GLSL (GL Shading Language)

Types et qualifications

- `mat4` type matrice (16 floats) ; `vec4` type vecteur (4 floats) ; etc
- Exemples :

```
void main() {
  vec4 a=vec4(0.1,0.2,0.3,0.4);
  float b=a.x;
  vec2 c=a.xy;
  float d=a.w;
  mat2 e = mat2(1.0,0.0,1.0,0.0);
  float f=e[1][1];
  vec2 g=e[1]; // 2ieme colonne.
  ...
}
```

- `uniform` : constant lors d'une primitive (glBegin(...) ... glEnd()) ; uniquement en lecture seule par le shader.
- `attribute` : peut changer entre chaque sommet. En lecture seule (sauf spécial).
- `varying` : valeur qui sera interpolée (généralement en écriture dans un vertex, et en lecture dans un pixel).

Communiquer avec un shader

- Les shaders peuvent lire (et non écrire) des variables globales passées par OpenGL : utilisation des uniform.
  vertex shader GLSL :

```
uniform float time; // variable globale au shader

void main(void) {
vec4 v = vec4(gl_Vertex);
  v.z = v.z+sin(5.0*sqrt(v.x*v.x+v.y*v.y) + time*0.5)*0.25;
  gl_Position = gl_ModelViewProjectionMatrix * v;
}
```

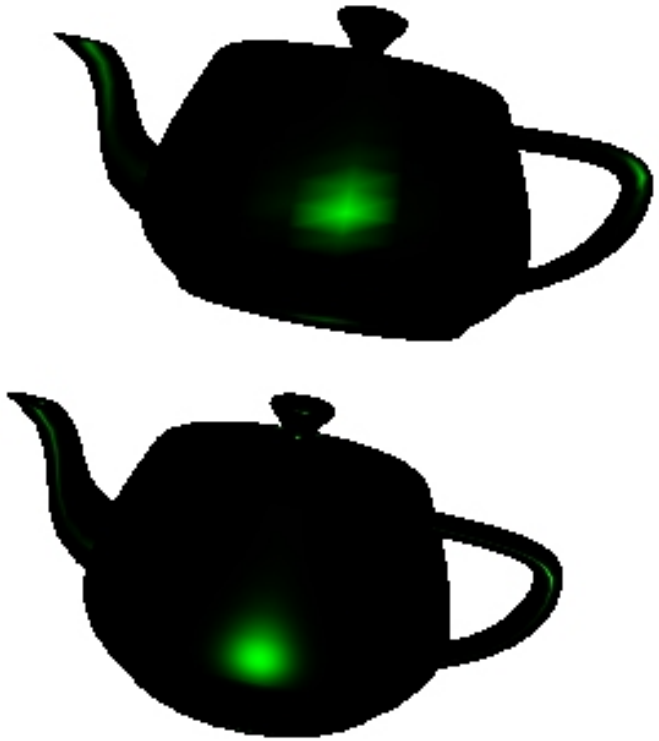
Modification de time

programme OpenGL :

```
int location_time; // handle pour la variable time
                   // du shader
float mon_temps; // varie entre chaque image
...
void myShader () {
  ...
  glLinkProgram(p);

  location_time = glGetUniformLocation(p,"time"); // détermination du handle
  // le programme p doit avoir été linké
}
...
void myDisplay(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glUseProgram(p);
  // affectation de l'uniform "time" du shader à la valeur mon_temps :
  glUniform1f(location_time,mon_temps);
  // glUniform fonctionne uniquement sur le programme courant (donné par glUseProgram)
  glutSolidTeapot(1);

  glutSwapBuffers();

  mon_temps+=0.01;
}
```

2 Un exemple : éclairement par pixel

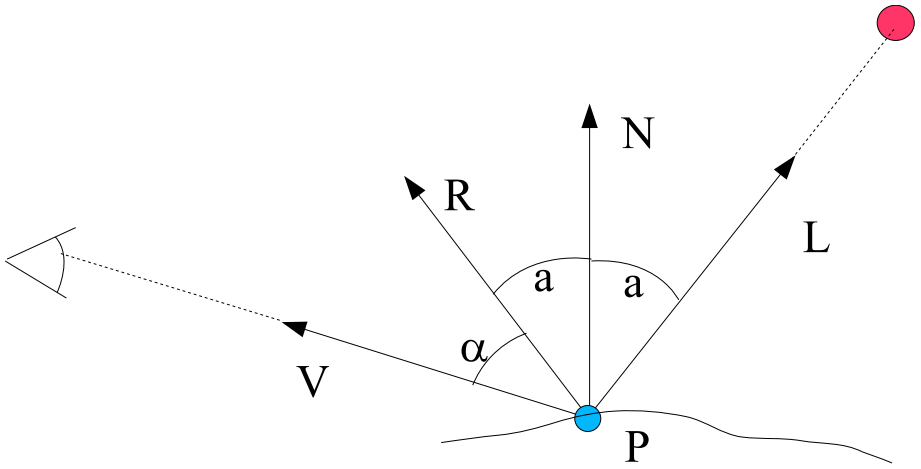
Rendre le spéculaire



- Eclairement par défaut d'OpenGL : calcul d'éclairement en chacun des sommets puis interpolation des couleurs pour chacun des pixels.
- Intérêt avec le fragment shader : provoquer un éclairement pour chaque pixel tracé pour nuancer le mauvais rendu du spéculaire d'OpenGL.

Rappel

- Calcul du spéculaire :



$$C_s = I_s * K_s(V \cdot R)^s$$

avec $R = 2(N \cdot L)N - L$

Principe

- Dans le vertex shader : pour chaque sommet on détermine les valeurs N, L et V (dans le repère de l'observateur).
- En chacun des pixels on pourra alors récupérer ces valeurs interpolées (moyennées).
- (L, N et V seront donc qualifiés de `varying`).
- Dans le fragment shader : on calcul le vecteur R puis le spéculaire résultant.

Vertex shader

```
varying vec3 N,L,V;

void main() {
  L=gl_LightSource[0].position.xyz;
  N=gl_NormalMatrix*gl_Normal;

  vec4 vertex=gl_Vertex;
  vertex=gl_ModelViewMatrix*gl_Vertex;

  V=normalize(-vertex.xyz);
  N=normalize(N);
  L=normalize(L);

  gl_Position = gl_ProjectionMatrix*vertex;
}
```

- `normalize` est fournie par GLSL pour normer un vecteur.
- `gl_LightSource[0]` correspond à la `LIGHT0`. On a accès à tous les paramêtres de `LIGHT0`.

Fragment Shader

```
varying vec3 N,L,V;

void main() {
  vec3 N2,L2,V2;
  vec3 R;

  N2=normalize(N);
  V2=normalize(V);
  L2=normalize(L);

  R=2.0*dot(N2,L2)*N2-L2;
  normalize(R);

  // intensité spéculaire
  float i_spec;
  i_spec=pow(max(dot(V2,R),0.0),gl_FrontMaterial.shininess);
  vec4 speculaire;
  speculaire=gl_LightSource[0].specular*gl_FrontMaterial.specular*i_spec;

  gl_FragColor = speculaire;
}
```

- Il faut normer N, L et V.
- dot est la fonction produit scalaire.
- On accède au K_s par `gl_FrontMaterial.specular`, au I_s par `gl_LightSource[0].specular`, et à la brillance s par `gl_FrontMaterial.shininess` (uniforms pré définis).

Remarques

- Attention : il n'y a que le spéculaire ici. Il faut ajouter le diffus et l'ambiant dans le vertex et le pixel shader.
- Remarque : pour ces 2 derniers, il suffit de calculer la couleur aux sommets, puis d'interpoler cette couleur (on ajoute directement à `couleur` les 2 varying provenant du diffus et de l'ambiant).

3 Accès aux textures

Exemple : multi-texturing simple

Vertex :

```
varying vec2 coord0,coord1;

void main() {

    coord0=gl_MultiTexCoord0.st;
    coord1=gl_MultiTexCoord1.st;
  gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
}
```

Fragment :

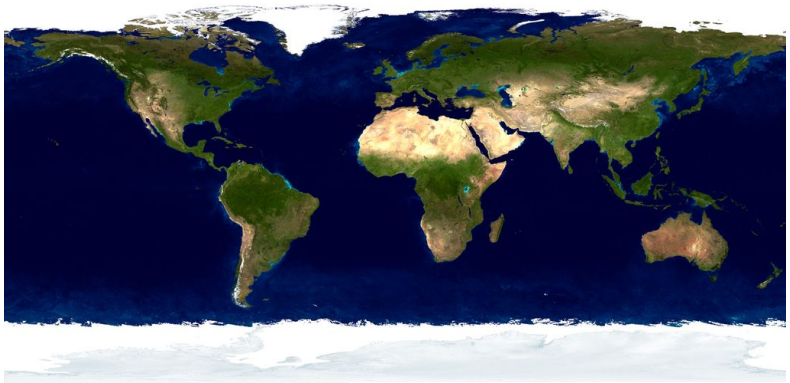
```
varying vec2 coord0,coord1;
uniform sampler2D tex1,tex0;

void main() {
  vec4 color0,color1;
  color0=texture2D(tex0,coord0);
  color1=texture2D(tex1,coord1);
  gl_FragColor = color0*color1;
}
```

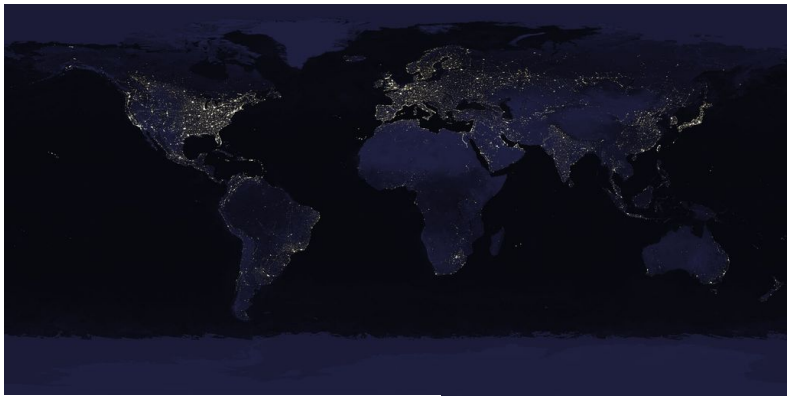
Le type sampler2D

- Doit être déclaré en uniform pour associer la variable du shader à une unité de texture.
- Exemple :

```
...
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D,tex_id);
...
location=glGetUniformLocation(p,"tex0");
glUseProgram(p);
glUniform1i(location,2);
// => tex0 fera référence à la texture courante de l'unité 2 (ici tex_id).
```

Exemple





Exemple

Vertex :

```
uniform vec3 posSource;

varying vec2 coord0,coord1;
varying vec3 normal;
varying vec3 L;

void main() {

  vec3 vertexEye;

  coord0=gl_MultiTexCoord0.st;
  coord1=gl_MultiTexCoord1.st;

  vertexEye=(gl_ModelViewMatrix*gl_Vertex).xyz;
  L=posSource-vertexEye;
  normal=(gl_ModelViewMatrix*vec4(gl_Normal,0)).xyz;

  gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Vertex;
}
```

Exemple

Fragment :

```
varying vec2 coord0,coord1;
uniform sampler2D tex0,tex1;
varying vec3 normal;
varying vec3 L;

void main() {
  vec4 color0,color1;
  vec3 normal2,L2;
  normal2=normalize(normal);
  L2=normalize(L);
  float diffus=dot(L2,normal2);

  if (diffus<0.0) {
    color0=texture2D(tex0,coord0);
    diffus=-diffus;
  }
  else {
    color0=texture2D(tex1,coord0);
  }

  gl_FragColor = color0*diffus*2.0;
}
```