OpenGL 2.0 – GLSL

David Odin

Septembre 2007

Définition

- A l'origine, script utilisant le multitexturage pour réaliser des effets (voir Quake 3 Aréna)

Définition

- A l'origine, script utilisant le multitexturage pour réaliser des effets (voir Quake 3 Aréna)
- Par extension, programme permettant de calculer la couleur d'un pixel

Définition

- A l'origine, script utilisant le multitexturage pour réaliser des effets (voir Quake 3 Aréna)
- Par extension, programme permettant de calculer la couleur d'un pixel
- Maintenant : programme exécuté directement par le GPU

Définition

- A l'origine, script utilisant le multitexturage pour réaliser des effets (voir Quake 3 Aréna)
- Par extension, programme permettant de calculer la couleur d'un pixel
- Maintenant : programme exécuté directement par le GPU
- Différents langages : Cg, Sh, HLSL, GLSL (celui d'OpenGL, ressemble à du C)

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée
- la normale interpolée

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée
- la normale interpolée
- les coordonnées de textures interpolées

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée
- la normale interpolée
- les coordonnées de textures interpolées
- les textures courantes

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée
- la normale interpolée
- les coordonnées de textures interpolées
- les textures courantes
- des variables “utilisateurs” (uniformes ou interpolées)

Fragment Shader

Ou “Pixel shader”.
Détermine la couleur d’un pixel (ou d’un fragment) en fonction de :

- la couleur interpolée
- la normale interpolée
- les coordonnées de textures interpolées
- les textures courantes
- des variables “utilisateurs” (uniformes ou interpolées)
- etc.

VERTEX SHADER

- Détermine la position (dans l'espace de l'écran) du vertex courant

VERTEX SHADER

- Détermine la position (dans l'espace de l'écran) du vertex courant
- Positionne les variables qui seront interpolées par le fragment shader

Geometry Shader

- Introduit avec les processeurs G80 chez NVidia

Geometry Shader

- Introduit avec les processeurs G80 chez NVidia
- Permet de générer des vertex à partir d'autres vertex

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : `GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);`
- glShaderSource() permet d’indiquer le code source d’un shader
  exemple : `glShaderSource (VS, 1, &vertexTab, NULL);`

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : `GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);`
- glShaderSource() permet d’indiquer le code source d’un shader
  exemple : `glShaderSource (VS, 1, &vertexTab, NULL);`
- glCompileShader () permet de compiler le shader (le code reste sur le GPU)
  exemple : `glCompileShader (VS);`

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : `GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);`
- glShaderSource() permet d’indiquer le code source d’un shader
  exemple : `glShaderSource (VS, 1, &vertexTab, NULL);`
- glCompileShader () permet de compiler le shader (le code reste sur le GPU)
  exemple : `glCompileShader (VS);`
- glCreateProgram() permet de créer un objet programme (un remplacement d’une partie du pipeline)
  exemple : `Prog = glCreateProgram ();`

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : `GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);`
- glShaderSource() permet d’indiquer le code source d’un shader
  exemple : `glShaderSource (VS, 1, &vertexTab, NULL);`
- glCompileShader () permet de compiler le shader (le code reste sur le GPU)
  exemple : `glCompileShader (VS);`
- glCreateProgram() permet de créer un objet programme (un remplacement d’une partie du pipeline)
  exemple : `Prog = glCreateProgram ();`
- glAttachShader() permet d’ajouter un code objet à un programme
  exemple : `glAttachShader (Prog, FS);`

FONCTIONS OPENGL 2.0

Pour supporter les shaders, OpenGL introduit quelques fonctions :

- glCreateShader() permet de créer un “objet shader”.
  exemple : `GLuint FS = glCreateShader (GL_FRAGMENT_SHADER);`
- glShaderSource() permet d’indiquer le code source d’un shader
  exemple : `glShaderSource (VS, 1, &vertexTab, NULL);`
- glCompileShader () permet de compiler le shader (le code reste sur le GPU)
  exemple : `glCompileShader (VS);`
- glCreateProgram() permet de créer un objet programme (un remplacement d’une partie du pipeline)
  exemple : `Prog = glCreateProgram ();`
- glAttachShader() permet d’ajouter un code objet à un programme
  exemple : `glAttachShader (Prog, FS);`
- glLinkProgram() réalise l’édition de lien
  exemple : `glLinkProgram (Prog);`

Fonction OpenGL 2.0 suite

Pour manipuler un programme shader, on peut utiliser les fonctions suivantes :

- glUseProgram() indique d'utiliser ce programme à la place du pipeline (0 permet de retrouver le pipeline traditionnel)
  exemple : glUseProgram(Prog)

FONCTION OPENGL 2.0 SUITE

Pour manipuler un programme shader, on peut utiliser les fonctions suivantes :

- glUseProgram() indique d'utiliser ce programme à la place du pipeline (0 permet de retrouver le pipeline traditionnel)
  exemple : `glUseProgram(Prog)`
- glGetUniformLocation() permet de récuperer un handle sur une variable globale d'un programme
  exemple : `glGetUniformLocation(Prog, "BrickColor");`

FONCTION OPENGL 2.0 SUITE

Pour manipuler un programme shader, on peut utiliser les fonctions suivantes :

- glUseProgram() indique d'utiliser ce programme à la place du pipeline (0 permet de retrouver le pipeline traditionnel)
  exemple : `glUseProgram(Prog)`
- glGetUniformLocation() permet de récuperer un handle sur une variable globale d'un programme
  exemple : `glGetUniformLocation(Prog, "BrickColor");`
- glUniform*() permet de changer la valeur d'une variable globale d'un programme
  exemple : `glUniform3f (glGetUniformLocation (Prog, "BrickColor"), 1.0, 0.3, 0.2);`

Exemple de fragment shader

Le plus simple possible.

```
void main (void)
{
  gl_FragColor = vec4 (1.0, 0.0, 1.0, 1.0);
}
```

FRAGMENT SHADER “MUR DE BRIQUE”

```
uniform vec3  BrickColor, MortarColor;
uniform vec2  BrickSize;
uniform vec2  BrickPct;

varying vec2  MCposition;
varying float LightIntensity;

void main(void)
{
    vec3  color;
    vec2  position, useBrick;

    position = MCposition / BrickSize;

    if (fract(position.y * 0.5) > 0.5)
        position.x += 0.5;

    position = fract(position);

    useBrick = step(position, BrickPct);

    color  = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
    color *= LightIntensity;
    gl_FragColor = vec4 (color, 1.0);
}
```

VERTEX SHADER “MUR DE BRIQUE”

```
uniform vec3 LightPosition;

const float SpecularContribution = 0.5;
const float DiffuseContribution  = 1.0 - SpecularContribution;

varying float LightIntensity;
varying vec2  MCposition;

void main(void)
{
    vec3 ecPosition = vec3 (gl_ModelViewMatrix * gl_Vertex);
    vec3 tnorm      = normalize(gl_NormalMatrix * gl_Normal);
    vec3 lightVec   = normalize(LightPosition - ecPosition);
    vec3 reflectVec = reflect(-lightVec, tnorm);
    vec3 viewVec    = normalize(-ecPosition);
    float diffuse   = max(dot(lightVec, tnorm), 0.0);
    float spec      = 0.0;

    if (diffuse > 0.0)
    {
        spec = max(dot(reflectVec, viewVec), 0.0);
        spec = pow(spec, 56.0);
    }

    LightIntensity  = DiffuseContribution * diffuse +
                      SpecularContribution * spec;

    MCposition      = gl_Vertex.xy;
    gl_Position     = ftransform();
}
```