

PROJET DE FIN D'ANNÉE

CAHIER DES CHARGES

GLShaderDev

Environnement de développement pour shaders OpenGL



Auteurs :

Thibault SCHUELLER

Maxime GRANDJEAN

19 février 2014



Historique des versions

Date	Version	Auteur	Section	Commentaire
2013/10/06	1	Thibault Schueller	-	Version initiale
2013/10/17	2	Thibault Schueller	-	Correction orthographique
2014/02/03	3	Thibault Schueller	1.2.2	Suppression des flags de texturing
2014/02/19	4	Thibault Schueller	-	Modification des auteurs

Table des matières

1	Fonctionnalités	2
1.1	Édition du code	2
1.1.1	Colorisation	2
1.1.2	Completion	2
1.1.3	Gestion de projet	2
1.2	Visualisation	3
1.2.1	Contenu de la scène	3
1.2.2	Contrôle sur la scène	3
1.2.3	Historique	3
1.2.4	Compilation	3
2	Technologies utilisées	4

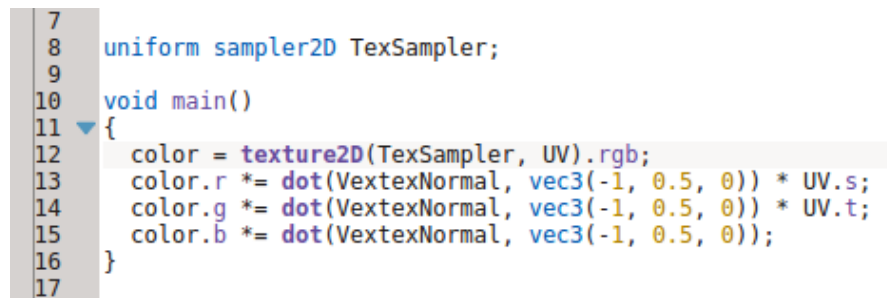
1 Fonctionnalités

Cet IDE sera axé sur la facilité d'édition du code GLSL et la facilité à passer du code à la visualisation du résultat.

1.1 Édition du code

1.1.1 Colorisation

Comme tout IDE qui se respecte, ce programme proposera une colorisation du langage GLSL, qui comporte une série de mots-clés et d'opérateurs intégrés.

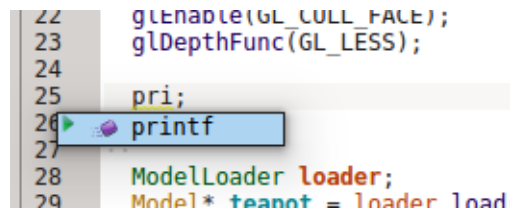


```
7
8 uniform sampler2D TexSampler;
9
10 void main()
11 {
12     color = texture2D(TexSampler, UV).rgb;
13     color.r *= dot(VextexNormal, vec3(-1, 0.5, 0)) * UV.s;
14     color.g *= dot(VextexNormal, vec3(-1, 0.5, 0)) * UV.t;
15     color.b *= dot(VextexNormal, vec3(-1, 0.5, 0));
16 }
17
```

FIGURE 1 – Colorisation de base de KDevelop

1.1.2 Completion

Pour être efficace dans l'écriture du code, une completion sera aussi intégrée à l'IDE, avec au minimum le support des fonctions de base GLSL.



```
22 glEnable(GL_CULL_FACE);
23 glDepthFunc(GL_LESS);
24
25 pri;
26 printf
27
28 ModelLoader loader;
29 Model* teapot = loader load
```

FIGURE 2 – Exemple de completion

Il suffira alors d'invoquer le parser de completion avec Ctrl+Space pour afficher une liste de fonctions correspondant au mot tapé.

1.1.3 Gestion de projet

Comme un shader est composé de plusieurs fichiers sources, il sera possible de créer des projets shader.

Il contiendront le nom des fichiers ouverts au sein du projet et ses options.

1.2 Visualisation

Le principal intérêt dans la création d'un IDE pour shader réside dans la visualisation directe du résultat. Une fenêtre OpenGL sera donc ouverte en permanence à l'écran, qui affichera la scène avec son traitement via le shader.

Celle-ci sera constamment à jour avec le code pour plus de flexibilité.

Il sera aussi possible de facilement prendre des screenshots.

1.2.1 Contenu de la scène

Comme le but de l'IDE n'est pas de permettre de créer de la géométrie 3D mais d'en modifier l'affichage, il devra prévoir des scènes de bases sur lesquelles appliquer le shader. En voici une liste de base :

- Un cube
- Un modèle chargé à partir d'un fichier
- Une simple texture (pour les effets de post-processing notamment)

1.2.2 Contrôle sur la scène

En sus de pouvoir afficher une variété de scènes à l'écran, maximiser le contrôle du développeur sur le rendu est important.

Il sera donc possible de tourner le rendu autour de son origine avec la souris, de zoomer avec la molette, voir même se déplacer librement dans la scène à la manière d'une caméra flottante.

1.2.3 Historique

Pour facilement comparer le travail en cours, l'utilisateur pourra pousser dans un historique sa version courante.

Il sera possible de parcourir l'historique de toutes les versions poussées, les visualiser et revenir à leur code. Cet historique sera sauvegardé et associé au projet courant.

Une fenêtre OpenGL flottante sera ouverte si l'on désire visualiser une version précédente.

1.2.4 Compilation

La procédure de compilation des shaders sera automatique et le log de compilation sera visible dans un panel de l'application. En cas d'erreur, il sera possible de cliquer dans le log, ce qui nous renverra sur la ligne de code en question, à la manière d'un IDE classique.

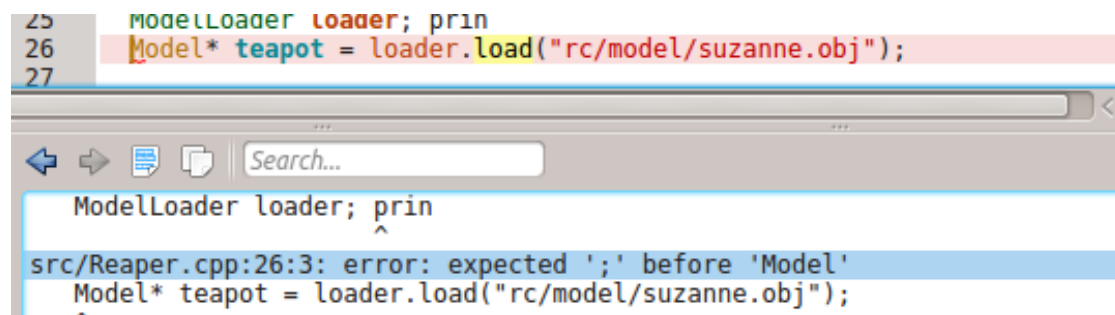


FIGURE 3 – Exemple de log interactif

2 Technologies utilisées

Pour permettre de viser un public large de développeurs, le programme sera réalisé en C++ couplé avec Qt (supérieur à 4.6). Quant à la version d'OpenGL, seront supportées les versions dites "modernes" (supérieur à 3.3).