POGL
TP1 : OpengGL 4 - Premier programme

Jonathan Fabrizio
LRDE - EPITA

Objectif

L'objectif de ce T.P. est d'écrire un premier programme en utilisant OpenGL 4.

1 Travail préliminaire

Pour pouvoir réaliser notre premier programme nous devons disposer d'un certain nombre de fonctions que nous allons écrire dans cette première partie du T.P..

1.1 Matrices et transformations

Écrire une classe `matrix4` contenant (au moins) les méthodes et constructeur publics suivants :

```
matrix4();
void operator*=(const matrix4& rhs);
static matrix4 identity();
```

Ajoutez une fonction pour écrire une matrice sur un flux :

```
std::ostream& operator<<(std::ostream &out, const mygl::matrix4 &m);
```

Écrire les fonctions suivantes qui construisent une matrice et la multiplie à la matrice donnée en argument afin de gérer la position de la caméra :

```
void frustum(mygl::matrix4 &mat,
    const float &left, const float &right,
    const float &bottom, const float &top,
    const float &z_near, const float &z_far
);

void look_at(mygl::matrix4 &mat,
     const float &eyeX, const float &eyeY, const float &eyeZ,
     const float &centerX, const float &centerY, const float &centerZ,
     float upX, float upY, float upZ
);
```

On pourra s'aider des anciennes spécifications d'OpenGL pour les écrire (`https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glFrustum.xml` et `https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml`)

1.2 Manipulation des *Shaders*

Écrire une classe *program* qui permet de gérer les *shaders* et les programmes. Elle devra comporter au moins les méthodes et constructeur/destructeur suivants :

```
program::program()
program::~program()
static program *make_program(std::string &vertex_shader_src, std::string &fragment_shader
char *get_log();
bool is_ready();
void use();
```

A la construction du programme (`make_program`), il faut compiler les deux *shaders* séparément puis invoquer l'éditeur de liens. `get_log()` renvoie les logs pour la compilation ou l'édition de liens. `is_ready()` indique si un programme est prêt à être utilisé (i.e. si la compilation et l'édition de liens ont réussi). `use()` active l'utilisation de ce programme pour le rendu.

1.3 Pour aller plus loin...

Lorsque vous aurez terminé votre T.P., vous pourrez ajouter des fonctionnalités tel que :
— une classe vecteur ainsi que la multiplication matrice vecteur,
— d'autres transformations tels que la rotation, la translation, la mise à l'échelle,
— la possibilité de charger le code de *shaders* écrit dans des fichiers séparés.

2 Programme OpenGL

2.1 Initialisations

Ajoutez les fonctions d'initialisation de GLUT, de GLEW et complétez les initialisations d'OpenGL avec notamment l'activation du *z-buffer* et du *backface culling*. Adapter la version d'OpenGL à la version supportée par votre carte graphique. Sous linux, aidez vous de l'outil `glxinfo` pour connaître la version supportée par votre *driver*.

```
bool initGlut(int &argc, char *argv[]) {
  glutInit(&argc, argv);
  glutInitContextVersion(4,5);
  glutInitContextProfile(GLUT_CORE_PROFILE);
  glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
  glutInitWindowSize(1024, 1024);
  glutInitWindowPosition ( 10, 10 );
  glutCreateWindow("Test OpenGL - POGL");
  glutDisplayFunc(display);
  return true;
}

bool initGlew() {
  return (glewInit()==GLEW_OK)
}

bool init_gl();
```

3 *Shaders*

3.1 Écriture des *shaders*

Vous devez maintenant écrire le *vertex shader* et le *fragement shader*. En entrée du *vertex shader* vous aurez :
— un *uniform* pour la couleur de l'objet,
— un *uniform* pour la matrice qui fait passer du repère monde à la projection,
— un VBO pour les sommets de l'objet à dessiner,
Avec ces *shaders* l'objet sera d'une couleur unie. Il n'aura donc pas d'effet de volume.
A l'aide de votre classe *program*, compilez vos *shaders* et corrigez les éventuelles erreurs.

3.2 Paramétrage des *shaders*

Maintenant que vous avez écrit vos *shaders* et une fois ces derniers compilés et liés, vous devez fixer :
— les *uniforms* de votre *vertex shader* (couleur de l'objet et matrice pour la projection)
— le *VBO* avec un objet de votre choix (un triangle, un cube, une sphère...).
Vous devez donc faire :
— une fonction pour fixer les *uniforms*,
— une fonction pour charger l'objet et initialiser le *VBO* (inclus dans un *VAO*).

4 Pour finir

4.1 Ajout de la fonction *display*

Vous avez mis un *call back* sur la fonction `display()`. Vous devez implémenter cette fonction.

4.2 *Go ! go ! go !*

Invoquer `glutMainLoop()` et Tadam !

4.3 Amélioration du rendu

Maintenant que votre programme fonctionne (Youpi !), faites un *TAG* sur votre dernier *commit* pour pouvoir le retrouver au cas ou vous auriez tout cassé.

Modifiez votre programme pour réaliser le lissage de Gouraud. Pour cela vous devez :

— ajoutez un *VBO* pour les normales

— ajoutez un *uniform* pour la position de la lumière et un autre pour la couleur/l'intensité de la lumière.

Modifiez le vertex shader pour prendre en compte ces modifications.

Maintenant que vous avez un second programme qui fonctionne (Youpi !), faites à nouveau un *TAG* sur votre dernier *commit* pour pouvoir le retrouver au cas ou vous auriez tout cassé. Modifiez vos *shaders* pour réaliser le lissage de Phong.

Pour finir, et comme il vous reste encore un peu de temps, amusez vous avec vos *shader* pour faire des effets sympas...

5 Après avoir fini et pour aller plus loin

5.1 *Shaders*

Vous pouvez modifier votre programme et imaginer plein d'améliorations comme :

— donner la possibilité de changer l'angle de vue de votre objet,

— modifier la couleur du rendu en fonction de la distance à l'observateur,

— ...