OpenGL
Mise en œuvre, vue d'ensemble

Forma3Dev pour CPE Lyon

29 septembre 2016

1 Les modes d'envoi de la géométrie

Il existe plusieurs moyen d'envoyer de la géométrie (des positions de vertex, des couleurs, des normales, des coordonnées de textures, etc.) à OpenGL.

1.1 Mode immédiat

Nommé ainsi car les données sont utilisées immédiatement par OpenGL. Il est assez simple à mettre en œuvre. Pour cela consultez la documentation des fonctions suivantes sur le site d'OpenGL :

- `glBegin(...);`
- `glEnd();`
- `glVertex*(...);`
- `glColor*(...);`

Question 1 : *Dans le programme donné (simple.tar.gz) ajoutez ce qu'il faut pour afficher un triangle coloré à l'aide du mode immédiat (voir figure 1).*

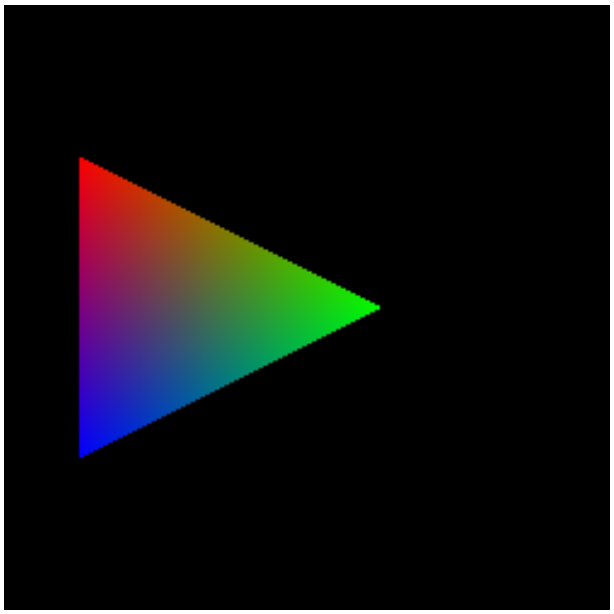

FIGURE 1 – Un triangle coloré.

Note : il suffit d'ajouter quelques lignes dans la fonction `display()`.

Note : Le mode immédiat est tombé en désuetude depuis un bon moment, il ne doit être utilisé que pour faire de rapides tests.

1.2 Tableaux

Lorsque le nombre de données devient important, il est plus pratique de les stocker dans des tableaux. OpenGL permet d'utiliser directement cela grace aux fonctions suivantes :

- `glEnableClientState(...);`
- `glVertexPointer(...);`
- `glColorPointer(...);`
- `glDrawArrays(...);`

Question 2 : *Après avoir regardé les documentations de ces fonctions, modifiez le programme afin d'ajouter un quadrilatère multicolor à l'aide de ces tableaux (voir figure 2).*

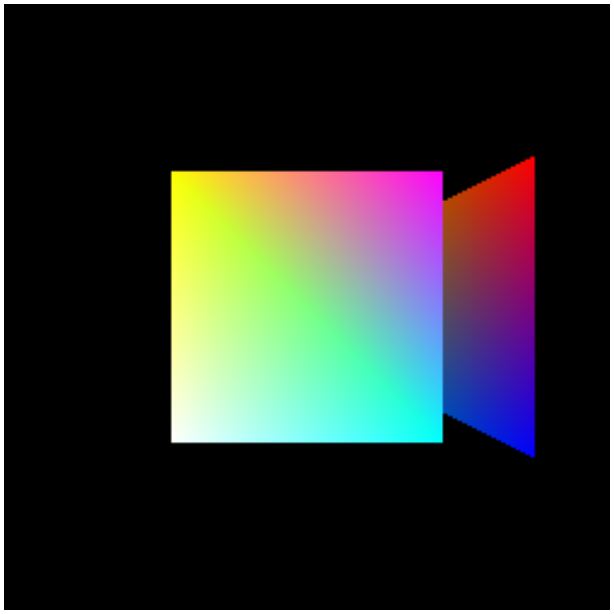

FIGURE 2 – Un quadrilatère multicolor.

Note : Il faudra placer certaines fonctions dans `init();` et d'autres dans `display();`

1.3 Tableaux indexés

Pour passer de l'utilisation de tableaux classiques à l'utilisation de tableaux indexés, il suffit d'ajouter un tableau d'index. Cela est détaillé dans l'aide de la fonction suivante :

- `glDrawElements(...);`

Question 3 : *Modifiez le programme afin d'utiliser un index pour l'affichage du quadrilatère.*

1.4 Tableaux en VRAM (VBO)

Pour plus de rapidité, il est possible de transmettre le contenu de ces tableaux en mémoire vidéo une fois pour toute. Pour cela, il faut créer un tampon (*buffer*) en mémoire video puis y transferer le tableau à l'aide des fonctions suivantes :

- `glGenBuffers(...);`
- `glBindBuffer(GL_ARRAY_BUFFER, ...);`

— `glBufferData(GL_ARRAY_BUFFER, ...);`
— `glVertexPointer(...);`
— `glColorPointer(...);`

Question 4 : *Modifier le programme afin d'utiliser un VBO pour les positions et couleurs du quadrilatère*

1.5 VBO indexés

Les tableaux en mémoire vidéo (VBO) peuvent aussi être indexés, soit par un index en mémoire centrale (RAM) dans quel cas cela s'utilise exactement comme pour les tableaux en RAM, soit par un index en mémoire vidéo. Pour placer les index en mémoire vidéo, on réutilise les fonctions suivantes, mais avec un paramètre différent :

— `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ...);`
— `glBufferData(GL_ELEMENT_ARRAY_BUFFER, ...);`
— `glDrawElements(...);`

Question 5 : *Modifier le programme afin d'utiliser un VBO également pour les index.*

2 Ajoutons de la texture (Optionnel)

Nos formes simplistes remplies de dégradés de couleurs sont assez trites finalement. Heureusement, on peut rendre les choses nettement plus attrayantes en utilisant la technique du plaquage de texture sur nos formes de bases.

On désire obtenir une image comparable à celle de la figure 3

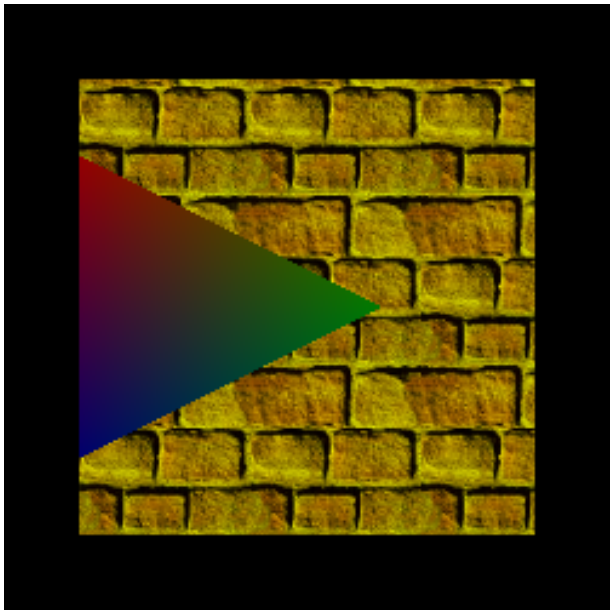

FIGURE 3 – Un quadrilatère texturé.

Pour cela, vous aurez besoin d'utiliser les fonctions suivantes :

— `glGenTextures(...);`
— `glBindTexture(GL_TEXTURE_2D, ...);`
— `glTexImage2D(GL_TEXTURE_2D, ...);`
— `glTexParameteri(GL_TEXTURE_2D, ...);`
— `glTexEnvi(GL_TEXTURE_2D, ...);`
— `glTexCoordPointer(...);`

3 Jouons avec les shaders / OpenGL 3.3

Pour cette partie, vous devrez utiliser l'archive `shader.tar.gz`. Il suffit de recompiler le tout. Vous n'aurez à éditer le fichier `main.cc` que pour modifier la fonction `update_uniforms()` et encore, uniquement quand il s'agira d'utiliser des variables uniformes. Vous pouvez cependant étudier l'ensemble du code.

Le programme `shader` affiche notre chat mascotte que l'on peut manipuler à la souris (!). Les plus curieux peuvent aller voir comment cela est réalisé) en remplaçant les fonctions de calcul de *vertex* et de *fragment* par deux programmes : `data/shader.vert` et `data/shader.frag` respectivement.

Ce sont ces deux fichiers qu'il vous faudra modifier avant de relancer le programme `shader` (pas besoin de recompiler quoi que ce soit).

Par défaut `shader.vert` contient ceci (en fait un peu plus, mais inutilisé pour l'instant) :

```
1  #version 330
2
3  in vec3 position;
4
5  uniform mat4 projection;
6
7  void main(void)
8  {
9    gl_Position = projection * vec4(position, 1.0);
10 }
```

C'est le minimum que peut contenir un *vertex shader*. Il se contente de calculer la position (`gl_Position`) de chaque *vertex* en appliquant une matrice nommée projection sur leur position de départ. La ligne `# version 330` signifie que l'on désire utiliser les fonctions définies dans la spécification de GLSL pour la version 3.3.

Et par défaut, le *shader* de *fragment* nommé `shader.frag` contient ceci :

```
1  #version 330
2
3  out vec4 color;
4
5  void main(void)
6  {
7    color = vec4 (1.0, 0.0, 0.0, 1.0);
8  }
```

Le fragment shader doit avoir une seule variable de type `out vec4`. Et cette variable doit obligatoirement recevoir une valeur qui sera la couleur du pixel. Ici, on colore tous les pixels en rouge.

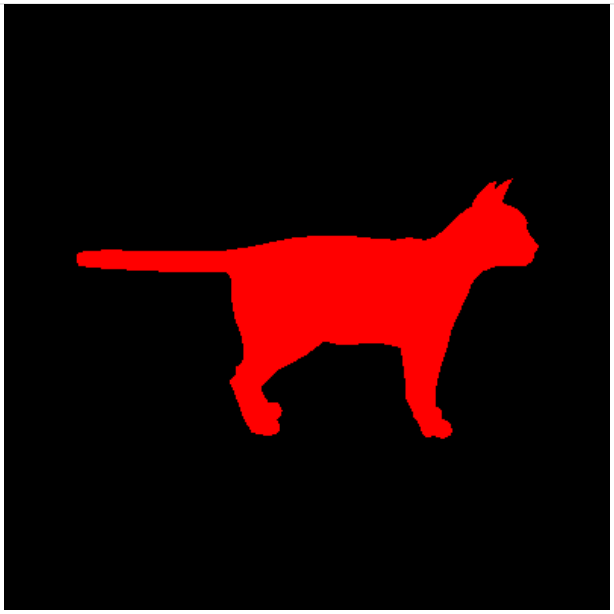

FIGURE 4 – Un pauvre chat tout rouge

3.1 Un peu de fantaisie

On peut heureusement faire mieux que des couleurs unies grâce à l'utilisation de fonctions et de certaines variables. Par exemple, la variable `gl_FragCoord` contient les coordonnées du pixel dans la fenêtre. Ainsi en utilisant la ligne suivante :

```
1  gl_FragColor = vec4 (mod (gl_FragCoord.xy / 30.0, vec2(1.0,1.0)), 0.0, 1.0);
```

On obtient un chat habillé en Tartan écossais du plus bel effet (voir figure 5).

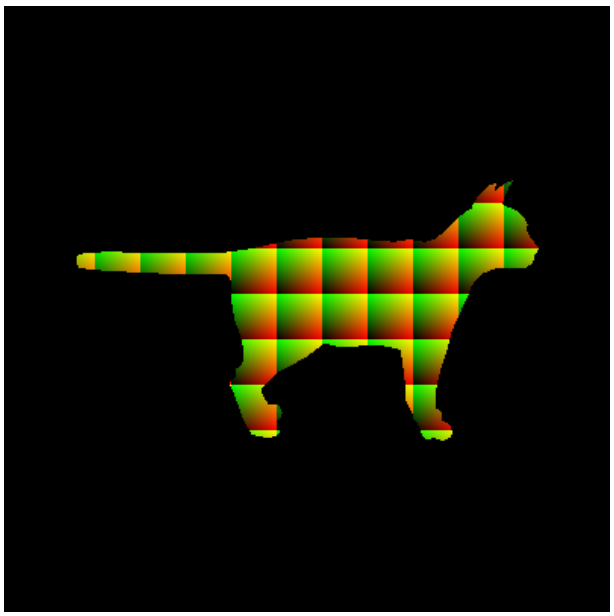

FIGURE 5 – Un chat en Tartan

Question 6 : *Modifiez le fragment shader afin d'afficher un damier ou une fractale de mandelbrot (ou n'importe quoi d'autre qui vous amuse...) sur la silouhette du chat.*

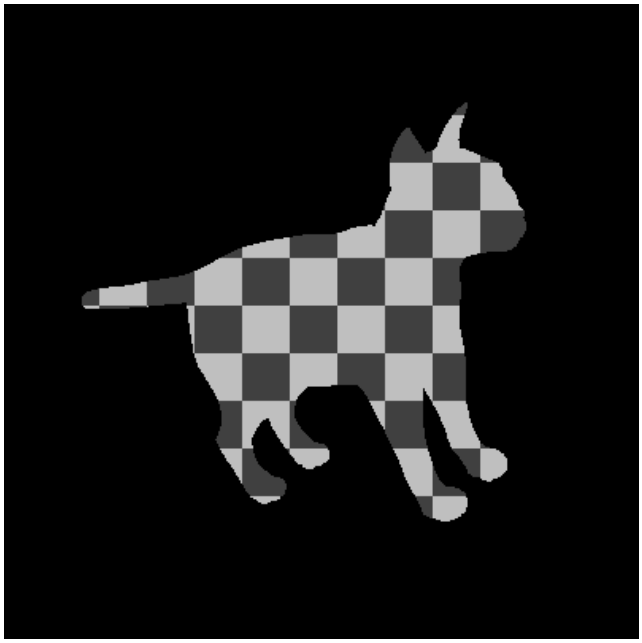

FIGURE 6 – Un chat-damier

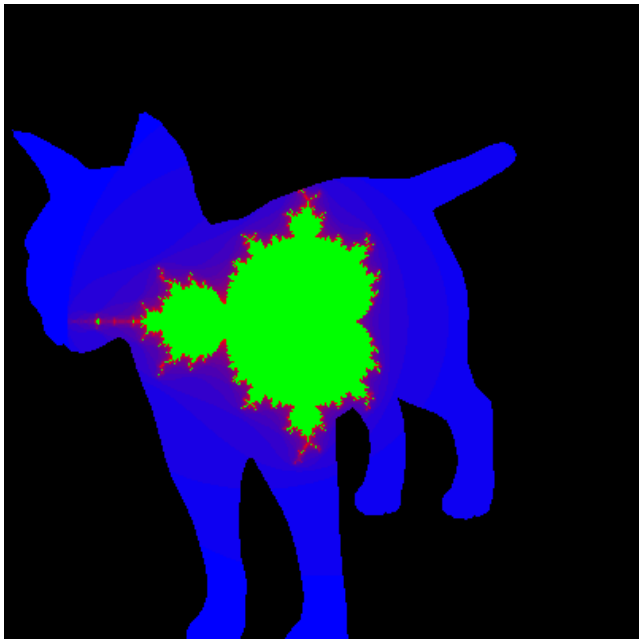

FIGURE 7 – Un fracto-chat (ça devient n'importe quoi...)

3.2 Passage de valeur entre les shaders

Heureusement, il est possible de faire plus intéressant. Notamment en passant des variables entre le *vertex shader* et le *fragment shader* Cela se fait en déclarant la même variable dans les deux *shader* en la préfixant par `out` dans le *vertex shader* et par `in` dans le *fragment shader*. Par exemple, on peut avoir ceci comme *vertex shader* :

```
1  #version 330
2  in vec3 position;
3  in vec4 normal;
4  out vec3 v_position;
5  out vec3 v_normal;
6  void main (void)
7  {
8    v_position = position;
9    v_normal   = normal;
10   gl_Position = projection * vec4(position, 1.0);
11 }
```

et ceci dans le *fragment shader* :

```
1  #version 330
2  in vec3 v_position;
3  in vec3 v_normal;
4  void main(void)
5  {
6    gl_FragColor = ...  ;
7  }
```

Cela permet de récuperer dans le *fragment shader* la position sur le chat et la normale. Ceci permet d'avoir des effets qui suivent la surface du chat. Par exemple, en utilisant ceci :

```
1    color = vec4 (abs(position) / 17.0, 1.0);
```

On obtient une couleur qui ne dépend que de la position sur le chat (voir figure 8).

En utilisant la normale, il est possible d'obtenir une impression d'éclairage et de relief comme on peut le voir sur la figure 9.

Question 7 : *Comment ?*

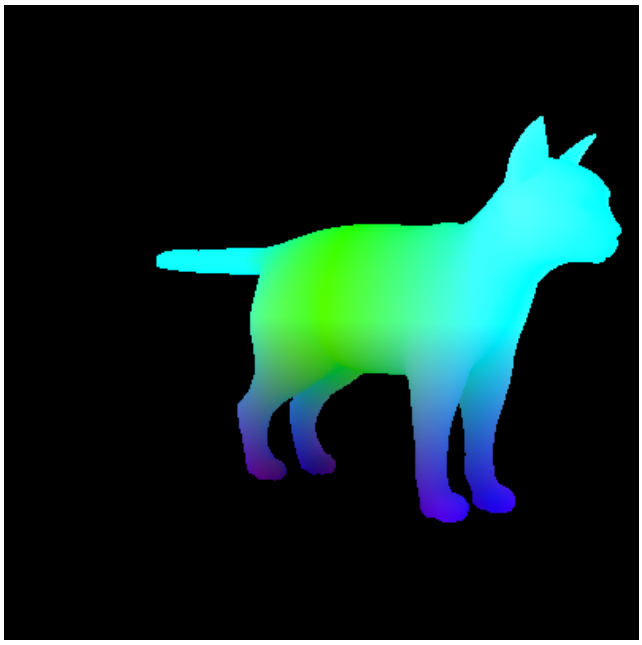

FIGURE 8 – Un chat en plastique moche

FIGURE 9 – Un chat en porcelaine

3.3 Variables globales (uniformes)

Il est aussi possible d'avoir des variables globales, qui ont une valeur qui ne change pas (qui reste uniforme donc...). Ces variables sont préfixées par le mot-clef `uniform` et peuvent être modifiées via le programme principal en utilisant les fonctions `glUniform*`.

Question 8 : *Modifiez la fonction `update_uniforms()` et vos shaders afin d'utiliser une telle variable.*

3.4 Utilisation des textures

Une utilisation très fréquente des variables uniformes concerne les textures.

Pour cela, il suffit d'ajouter les lignes suivantes dans le *fragment shader* :

```
1 uniform sampler2D  myTexture;
2 in vec2 v_tex_coords;
3 ...
4
5   vec4 color = texture(myTexture, v_tex_coords);
```

Question 9 : *Essayez d'obtenir un affichage ressemblant à la figure 10.*

FIGURE 10 – Un chat en brique pour toujours plus de réalisme

FIGURE 11 – Un chat nettement plus normal

4 Quelques liens utiles

- Le site officiel concernant OpengGL : http://opengl.org/.
- La documentation en ligne des fonctions OpenGL : http://www.opengl.org/sdk/docs/man2/.
- Un résumé des fonctions OpenGL et GLSL (malheureusement pour OpenGL 3.2 et GLSL 1.5) : http://www.khronos.org/files/opengl-quick-reference-card.pdf.
- Un site qui abuse des fragment shaders : https://www.shadertoy.com/.