TP2 - Programmation 3D

TP GLSL

Université de Montpellier

Léa Serrano M1 IMAGINE

29 Septembre 2022

Lien de mon git pour ce tp : https://github.com/LeaSerrano/M1-IMAGINE-Prog3D-TP2.git

Dans ce tp, nous allons chercher à afficher un triangle en utilisant les pipelines OpenGL (c'est à dire en utilisant des fichiers shaders).

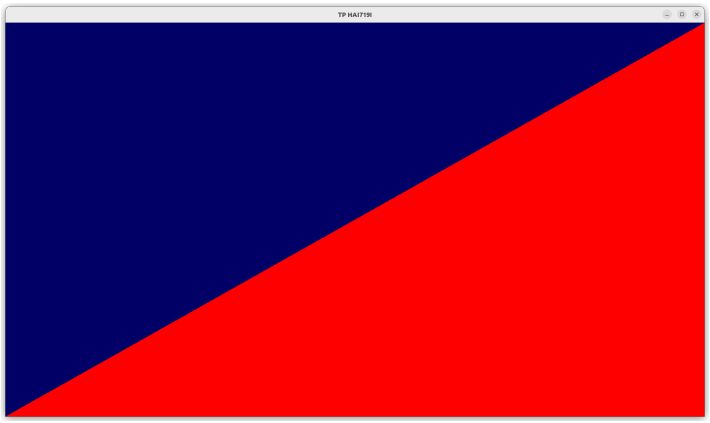
Table des matières

1 Exercice 1

Un triangle est défini dans la structure TriangleVArray, completer les fonctions de cette structure : afficher le à l'aide des shaders en complétant les parties indiquées du code.

Pour réaliser cet exercice, nous avons complété les fonctions **initBuffers()**, **clearBuffers()** et **draw()** dans la structure de TriangleVArray.
Pour ce faire, nous avons utilisé différentes fonctions d'OpenGL qui permettent de transmettre les informations de notre code (vertices, couleurs de TriangleVArray) aux shaders.

Voici le résultat obtenu :

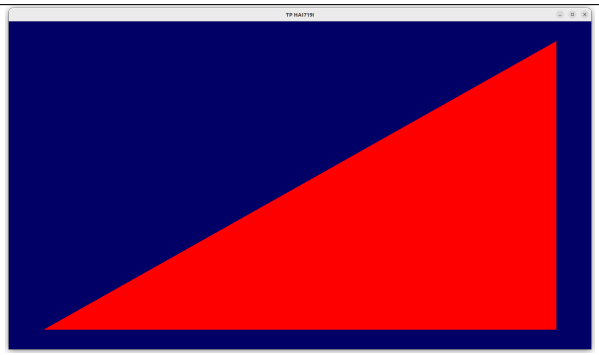
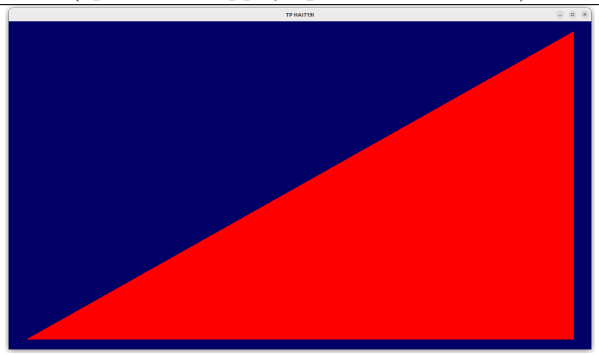


2 Exercice 2

Passer une variable uniform au shader afin d'appliquer une mise à l'échelle uniforme sur le triangle (variable globale scale definie dans tp.cpp). Permettre de contrôler ce paramètre à l'aide du clavier (+/-). Ajouter une translation contrôlée à l'aide du clavier (q/d en x et z/s en y).

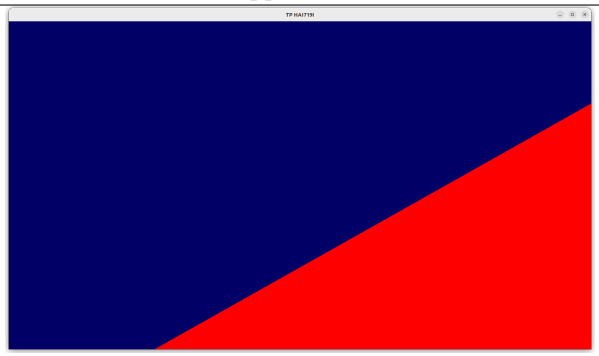
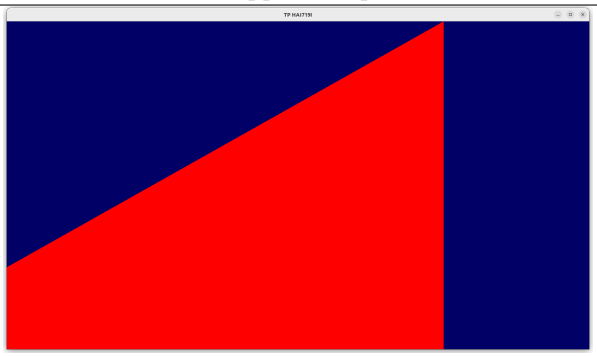
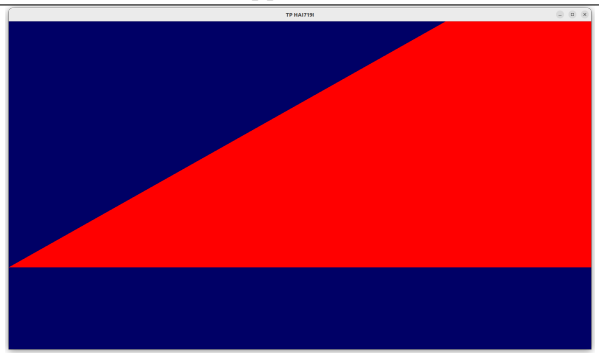
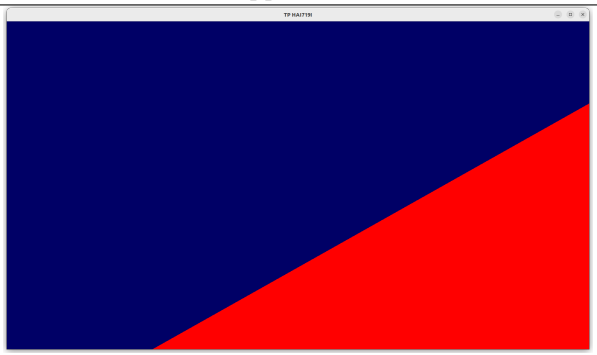
Nous allons donc commencer par envoyer la valeur de scale aux shaders, cela se fait grâce à cette ligne de code **glUniform1f(glGetUniformLocation(programID, "scale"), scale) ;**. Cela va nous permettre de récupérer cette valeur dans le vertex_shader, nous allons ensuite la multiplier par la position des vertices afin d'avoir le résultat souhaité.
Ensuite nous allons faire en sorte que ce paramètre soit augmenté lorsque que l'on appuiera sur + et diminué lorsqu'on appuiera sur - et que les nouvelles valeurs soient bien envoyées à notre vertex_shader.

Voici ce qu'on obtient :

Triangle lorsqu'on a appuyé plusieurs fois sur -	Triangle lorsqu'on a appuyé plusieurs fois sur + (après avoir appuyé plusieurs fois sur -)
	

Ensuite, nous avons géré la translation. Nous avons donc, dans le fichier vertex_shader, additionné la translation à la position multiplée par scale.
Ensuite dans notre fichier principal, nous avons inialisé un vecteur de translation à (0, 0, 0) et lorsque l'on va appuyer sur une touche du clavier (d pour augmenter la valeur de x et q pour la diminuer, z pour augmenter la valeur de y et s pour la diminuer).

Voici ce qu'on obtient :

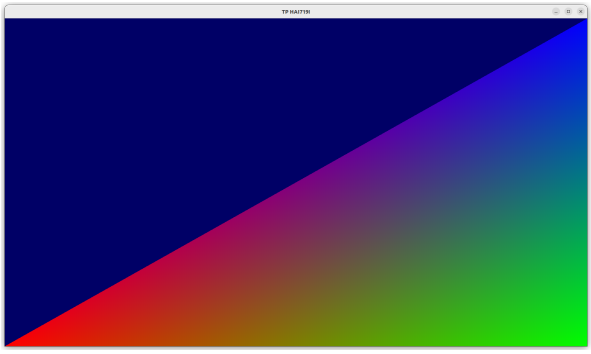
Appui sur d	Appui sur q
	
Appui sur z	**Appui sur s**
	

3 Exercice 3

Ajouter un attribut couleur, mettre à jour le code CPU et GPU. Définir comme couleur = normal du sommet et interpoler la valeur pour les fragments (piste, variable in et out).

Nous allons maintenant ajouter un attribut couleur afin de pouvoir définir la couleur de notre triangle. Pour ce faire, nous allons récupérer les couleurs de chaque sommet que nous avons définies dans l'exercice 1 dans le vertex_shader. Nous allons faire ça en récupérant dans un attribut vec3 la couleur correspondante dans le buffer. Ensuite, nous allons transmettre cette valeur à fragment_shader à travers des variables.
Enfin, dans fragment_shader on va définir la couleur actuelle avec la couleur que l'on a récupérée.

Voici ce que nous obtenons :



4 Exercice 4

Mettre à jour la structure de mesh pour afficher le triangle en tant que liste indexée de sommets.

Pour ce dernier exercice, nous allons compléter les mêmes fonctions qu'à l'exercice 1 mais cette fois-ci nous travaillerons dans la structure Mesh. Cela va nous permettre d'afficher les maillages de notre triangle.
Ensuite nous allons devoir compléter le vertex_shader afin de définir la position avec notre nouvelle structure

Malheureusement, cela n'a pas fonctionné dans mon code car rien ne s'affichait.

5 Conclusion

Pour conclure, ce tp a été assez bien réussi (même si il manque un exercice). L'objectif de départ était d'afficher un triangle coloré qu'on pourra translater et dont on pourra réduire l'échelle, aussi le but était d'afficher le maillage.
Nous avons découvert, avec ce tp, le principe des buffers ainsi que la programmation avec les shaders.