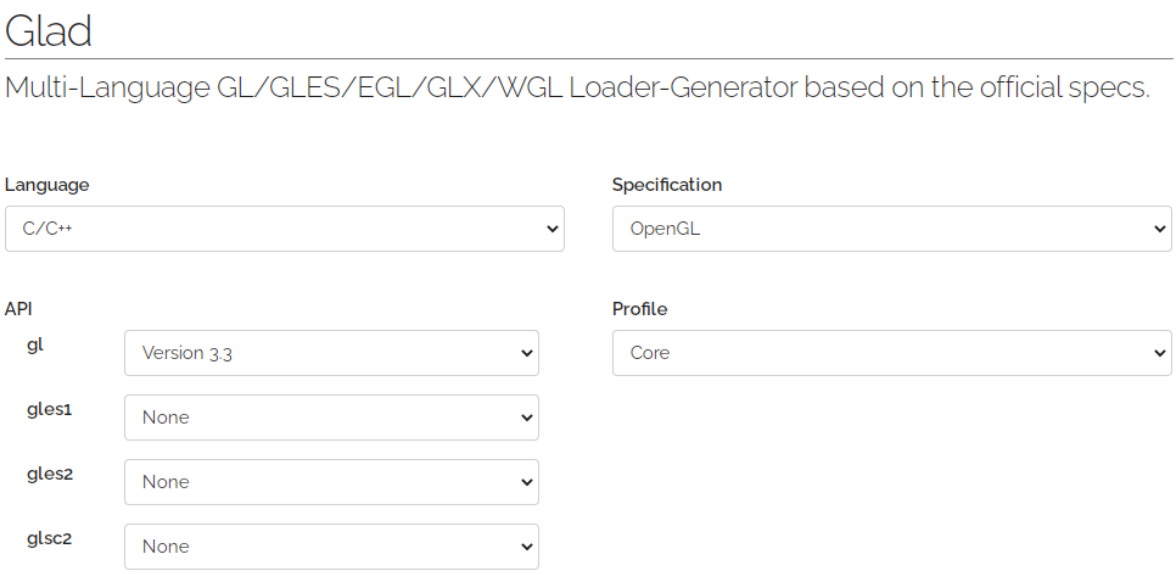
# TP N°3
# Premier dessin OpenGL 3+

## Différence entre opengl 2 et opengl 3+ :

Dans le rendu direct (OpenGL 2), à chaque exécution de la boucle d'affichage, l'ensemble des données est calculé et envoyé à la carte graphique. Afin de minimiser le temps passé dans cette boucle, le rendu par buffer (OpenGL 3+) permet de stocker les données directement sur la carte graphique.
Les méthodes de rendu direct (glBegin, glEnd, glTranslate, glRotate, ...) ont été dépréciées et elles pourraient ne plus être disponibles un jour.

## GLAD :

La bibliothèque « Glad » nous permet d'avoir accès aux fonctions OpenGL 3. Sans elle on peut considérer que l'on a accès qu'aux fonctions OpenGL 2.
- Télécharger la bibliothèque précompilée à partir du web service : https://glad.dav1d.de/




- Le dossier compressé téléchargé contient un dossier include (à ajouter à votre projet) et un fichier source « glad.c » (à copier dans votre projet et ajouter avec les sources).

## Premier dessin :

- Ajouter la directive include :

```
#include <glad/glad.h>
```

- Ajouter le code suivant après l'initialisation de GLFW :

```
glfwWindowHint(GLFW_FSAA_SAMPLES, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR , 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE,GLFW_OPENGL_CORE_PROFILE);
```

- Après la création de la fenêtre et du contexte OpenGL, il faut initialiser Glad :

```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
   cout << "Could not initialize GLAD" << endl;
   return -1;
}
```

- Déclarer les sommets :

```
float vertices[] = {
                     1.0f, 0.0f, 0.0f,
                      0.0f, 1.0f, 0.0f,
                      -1.0f,  0.0f, 0.0f,
                     };
```

- Déclarer les buffers :

```
GLuint VAO;
GLuint VBO;
```

- Générer et lier le VAO :

```
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);
```

- Il faut qu'OpenGL puisse stocker dans le VAO une référence vers le VBO contenant les données. Le VBO contient les données et le VAO décrit les données.
Pour cela, il faut générer le VBO et le lier sur la cible GL_ARRAY_BUFFER :

```
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

- Fournir les sommets à OpenGL pour qu'ils soient placés dans le VBO :

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

- Lier le premier buffer d'attributs (les sommets) et configurer le pointeur :

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat) , (void*)0 );
```

- On indique à OpenGL qu'on utilise un attribut donné :

```
glEnableVertexAttribArray(0);
```

- Débinder le VAO et le VBO :

```
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

Dans la boucle d'affichage :

- Lier le VAO :

```
glBindVertexArray(VAO);
```

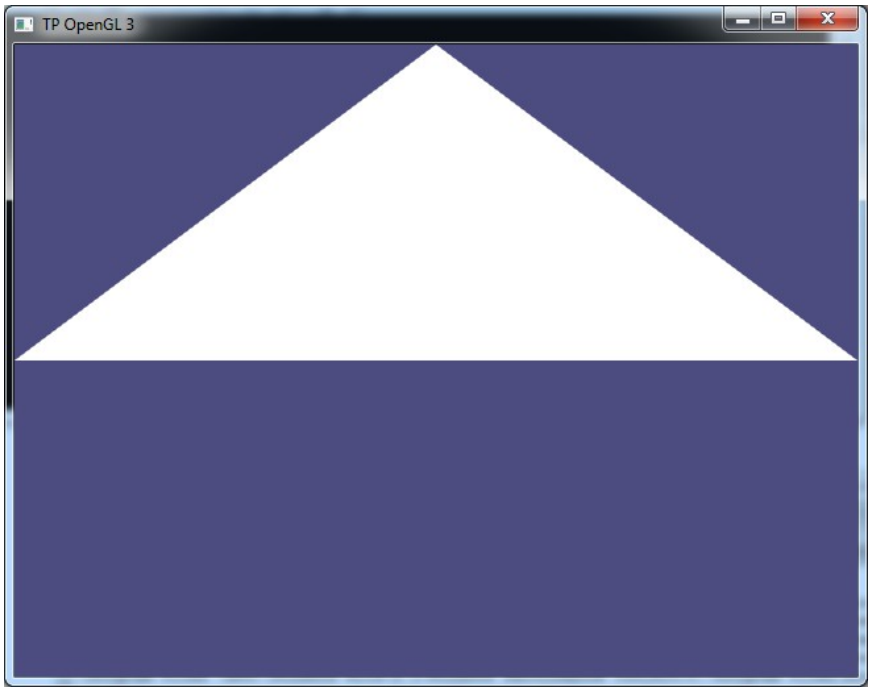
- Dessiner le triangle :

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

- En sortant de la boucle d'affichage, on procède au nettoyage :

```
glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
```

- On obtient :




## Les prototypes des fonctions :

glGenBuffers(GLsizei n, GLuint* buffers)

glBindBuffer(GLenum target, GLuint buffer)

glBufferData(GLenum target, GLsizeiptr size, const GLvoid* data, GLenum usage)

glGenVertexArrays(GLsizei n, GLuint *arrays)

glBindVertexArray(GLuint array)

glEnableVertexAttribArray(GLuint index)

glVertexAttribPointer(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid* pointer)

glDrawArrays(GLenum mode, GLint first, GLsizei count)