



IMN401 - Infographie et jeux vidéo - TP 1

Guillaume GILET - guillaume.gilet@usherbrooke.ca

1. Consignes

- Travail à faire par groupe de 3.
- Le travail sera à rendre avant le dimanche 2 février.
- Livrables : Une archive **nettoyée**¹ contenant le code du TP1. A rendre sur turnin.dinf.usherbrooke.ca

2. Objectif

L'objectif de ce TP est de se familiariser avec les commandes OpenGL, la gestion des buffers et des *shaders* pour afficher de la géométrie simple en 2D. Vous partirez de la base de code fournie, qui ouvre une fenêtre munie d'un contexte OpenGL. La section 3 propose des étapes à suivre pour se familiariser avec les mécanismes OpenGL. La partie du code à rendre pour ce TP est décrite dans la section 4.

3. Premières étapes et prise en main

Cette première étape va vous permettre de commencer pas à pas à réaliser un programme en OpenGL. Le projet fourni contient un fichier source, comprenant une fonction main et deux fonctions pour afficher les erreurs de compilation des *shaders*, ainsi que deux *shaders* (VS et FS). Commencez par lire le code et essayez de repérer la partie d'initialisation du programme ainsi que la boucle de rendu. Notre première étape consiste à afficher un unique triangle en 2D dans la fenêtre. Nous travaillerons ici directement dans le plan image (donc sans utiliser les matrices de transformations vues en cours).

- Commencez par déclarer et remplir un tableau de 3 sommets. Chaque sommet sera défini par un point 3D (situé sur le plan $Z = 0$) indiquant sa position dans l'espace (Rappel : les coordonnées du plan image en X et Y sont centrées sur 0 et vont de -1 à 1)
- Créez ensuite un **Vertex Buffer Object** et un **Vertex Array Object** pour charger le tableau des sommets en mémoire graphique. Suivez pour cela les indications données dans le cours.
- A l'aide de la fonction de lecture de fichier donnée dans le squelette de code, les variables ***vsCode*** et ***fsCode*** contiennent le code des fichiers (resp.) ***triangle-vs*** et ***triangle-fs***. A l'aide du cours, définissez les objets OpenGL correspondants aux *vertex shader* et *fragment shader* ainsi qu'un pipeline de rendu. Pensez dès maintenant à appeler les fonctions d'affichage des éventuelles erreurs de compilation des *shaders* et du *pipeline*.
- Votre *vertex shader* devra écrire la position du sommet dans la variable ***gl_Position*** (cf. cours). Rappel : nos sommets sont ici directement définis dans le plan. Il n'est donc pas nécessaire de réaliser une projection.
- Votre *fragment shader* devra recevoir en variable uniform une couleur et assigner cette couleur à chaque fragment.

¹ N'hésitez pas à demander à votre chargé de TP pour le nettoyage des archives

4. Travail demandé

Objectif : Nous souhaitons afficher un cercle coloré, centré sur le plan 2D, et animé par un *vertex shader* lors du rendu. Ce cercle verra son rayon s'agrandir et se réduire en fonction du temps, ainsi que sa position évoluer dans la fenêtre. Vous **devez** utiliser pour cela, le mode de **géométrie indexée** (i.e. en définissant explicitement un tableau de faces stocké **sur le GPU**). Attention, votre code devra respecter les spécifications OpenGL 4.6 étudiées en cours. L'utilisation de fonctions OpenGL plus anciennes ne sera pas pris en compte lors de l'évaluation.

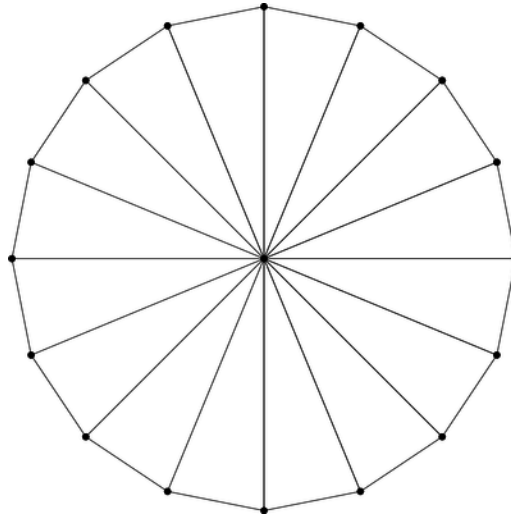


Figure 1 : Géométrie du cercle

En partant de la fin de la section précédente, voici quelques étapes pour arriver à ce résultat :

- Générez, lors de l'initialisation, **n (n>30)** sommets situés sur un cercle centré de rayon 0.5, à l'aide d'une boucle. Créez dans le même temps un tableau de faces (chaque triangle doit avoir pour origine le sommet central, comme montré sur la figure 1)
- Votre *vertex shader* devra transmettre au *fragment shader* une nouvelle variable contenant la position dans l'espace 2D de chaque sommet. A l'aide du code suivant, assignez dans le *fragment shader* une nouvelle couleur pour chaque fragment (ce code convertit une position 2D entre -1 et 1 vers une couleur RGB)
- Transmettez au *vertex shader* une variable **uniform** contenant le temps écoulé (en ms) depuis le début du programme (utilisez pour cela *time.h* ou *std::chrono*). A l'aide de cette variable et de la fonction **glsl** cosinus, déplacez dans le *vertex shader* chaque sommet le long du rayon du cercle, de manière à faire grandir et rétrécir le cercle en fonction du temps.
- Ajoutez une translation en fonction du temps à chaque sommet, vous pouvez utiliser *sin()* pour faire osciller le cercle horizontalement dans la fenêtre.



```
vec3 posToRGB( vec2 pos )
{
float a = 0.5*atan(pos.y,pos.x)/3.14159;
  if (a < 0.0)
    a += 1.0 ;
float l = length(pos);
vec3 rgb = clamp( abs(mod(a*6.0+vec3(0.0,4.0,2.0),6.0)-3.0)-1.0, 0.0, 1.0 );
  return mix( vec3(1.0), rgb, l);
}
```