L'objectif de ce projet est de manipuler des objets 3D dans l'espace et d'observer leur projection sur un écran simulé par une console système. L'ensemble de ce projet se fera en C++.

Il est inspiré par un projet d'A1k0n "Donut math", réalisé pour la première fois en 2006 et mis à jour en 2011, dont on pourra trouver l'explication complète et le code source ici : https://www.a1k0n.net/2011/07/20/donut-math.html

```
          k;double sin()
        ,cos();main(){float A=
      0,B=0,i,j,z[1760];char b[
    1760];printf("\x1b[2J");for(;;
  ){memset(b,32,1760);memset(z,0,7040)
  ;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
 >i;i+=0.02){float c=sin(i),d=cos(j),e=
 sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
 h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
in(B),t=c*h*g-f*        e;int x=40+30*D*
(l*h*m-t*n),y=            12+15*D*(l*h*n
+t*m),o=x+80*y,          N=8*((f*e-c*d*g
 )*m-c*d*e-f*g-l        *d*n);if(22>y&&
 y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
 ".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
  printf("\x1b[H");for(k=0;1761>k;k++)
   putchar(k%80?b[k]:10);A+=0.04;B+=
     0.02;}}/*****####******!!=;:~
       ~::==!!!**********!!!==::-
         .,~~;;;========;;;:~-.
             ..,--------,*/
```

Code source (en langage C)

```
             $$$$$$$$@@@@$
          ####******###$$$$@@$
        *##**!======!!!**###$$$$$$#
      #**!!!!===;;;;=;=!**##$$$$$$#
    *#**!!=;:~~----~:;;=!!*##$$$$$$#!
   *#**!!=;:-.,.....-~:;=!**###$$$###
  *###**!=:~,........-:;=!**#########*
 !*####*!=:-....      .,:;!!**########**
 *##$$##*!=~..         -;=!***#######**!=
;*#$$$$$#*!;.         -;=!****######***!=
;*#$$$@@$$#*;        ;=!!****#####****!!;
:!*$$$@@@@$$#*! !!!******####*****!!==
~=*#$$$@@@$$$$#####***####*******!!!=:
 ;=*##$$$$$$$$$#########*****!*!!!=;:
 ~==!**#####$$#######*******!*!!!==;:
  -;==!****####********!!!!!!===;;~
   -:;!!!!!!****!!*!!!!!!!===;;:~,
     -~;======!==!!!=!====;;:~~,
       ,~::;;;;;;;;=;;;:;::~~-.
```

Résultat sur la console système

L'intérêt de ce premier projet n'est évidemment pas de copier-coller le code source afin d'obtenir le même résultat, mais de comprendre les mathématiques derrière et d'être capable d'adapter le code pour modifier l'affichage (changer de forme, changer les distances de projections, utiliser d'autres valeurs de seuil Ascii...).

Pour obtenir le résultat sur la console système, il faut passer par **4 étapes**, qui seront les quatre étapes du projet :

- **Générer l'objet** dans un espace 3D;
- **Le faire tourner** autour d'un ou plusieurs axes;
- **Calculer la luminosité** en chaque point de l'objet en fonction d'une source de lumière donnée;
- **Projeter l'objet** sur un écran.

Afin de rapidement obtenir des résultats concrets sur la console système, ce projet se concentrera en premier lieu sur l'affichage via la console système, avant de travailler sur la génération d'objets statiques qui seront projetés sur cette console. Ensuite viendront les rotations dynamiques de ces objets, et enfin l'interaction avec la lumière.

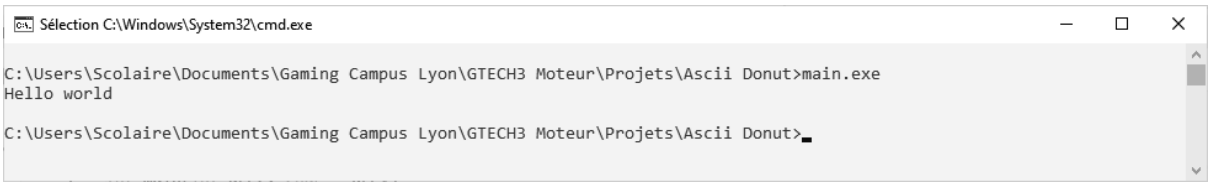
Le suivi de ce projet sera fait sur GitHub. Chaque élève créera un repository sur son GitHub et effectuera un commit/push pour chaque question. Ne pas mettre l'exe sur le repository.

Partie 1 : Travail avec la console système

Pour ouvrir une console système dans un dossier sur Windows, on peut utiliser le raccourci win+R et taper “cmd” puis naviguer jusqu’au dossier voulu avec la commande “cd”, ou on peut également se rendre dans le dossier via l’explorateur windows et taper “cmd” dans la barre de l’explorateur.

Cette console servira à la fois à compiler le ou les fichiers sources, lancer l'exécutable et afficher le résultat.

1.1 Créer un fichier main.cpp qui affiche un “Hello world”. Le compiler via la console système, puis l'exécuter sur celle-ci.



Depuis 2016, Microsoft a déployé la version 1511 de Windows 10 qui reconnaît les **séquences de contrôle ANSI** dans les consoles système. Ces séquences permettent notamment de **manipuler directement le visuel de la console** (changement de couleur de texte par exemple).

1.2 Trouver les séquences d’échappement ANSI qui permettent :

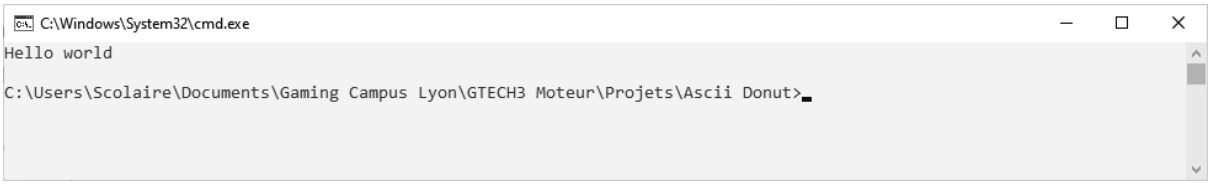
- d’effacer tous les caractères présents dans la console;
- de remettre le curseur en “home” position (0,0) (en haut à gauche);
- de rendre le curseur invisible / visible.

Pour activer la lecture et la compréhension de ces séquences par le programme, il est nécessaire d’ajouter le bloc de code suivant :

```
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
DWORD mode;
GetConsoleMode(hConsole, &mode);
SetConsoleMode(hConsole, mode | ENABLE_VIRTUAL_TERMINAL_PROCESSING);
```

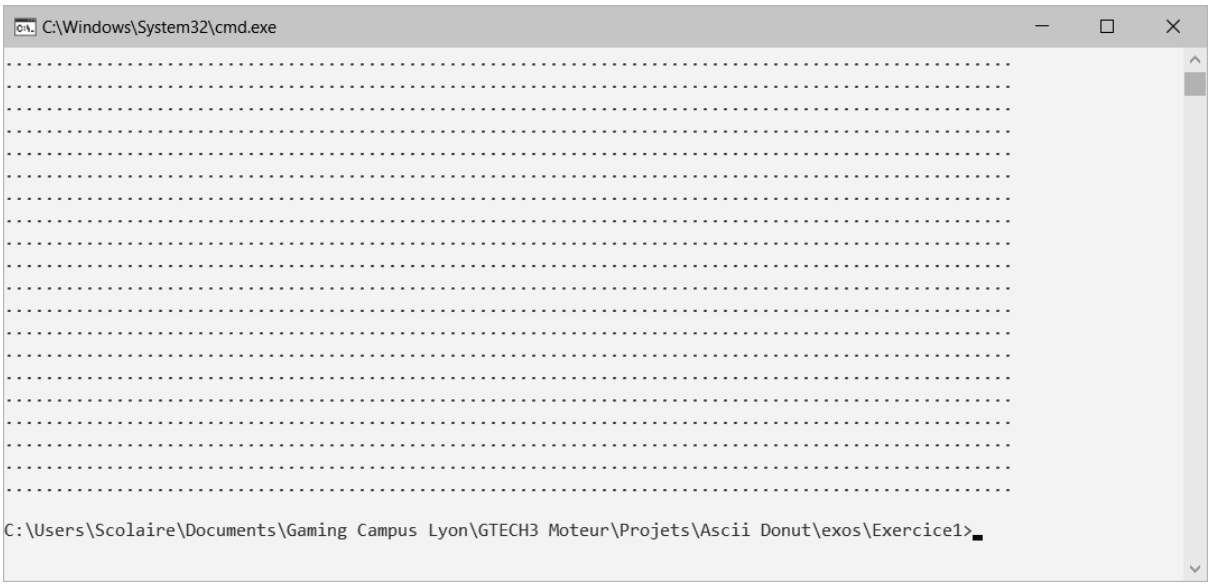
Ces fonctions sont issues de <windows.h>.

1.3 Modifier le fichier main.cpp pour qu'à l'exécution du .exe tous les caractères soient retirés de la console et que "Hello word" s'affiche au début de celle-ci. Le curseur doit également disparaître durant l'exécution du programme, et réapparaître à la fin de celui-ci (ce qui sera invisible pour le moment).



La console système va servir à simuler un écran d'une certaine résolution. Les pixels de cet écran seront stockés dans un **vecteur de dimension 1 contenant des char**.

1.4 Modifier le fichier main.cpp pour simuler un écran de 20x100 pixels au lieu d'afficher "Hello world". Pour l'instant, afficher le caractère '.' pour chaque pixel.



1.5 Modifier le fichier main.cpp pour ajouter deux arguments de ligne de commande : "-h" et "-w", qui permettent respectivement d'indiquer la hauteur (height) et largeur (width) de l'écran en pixels.

Afin de structurer le projet et avoir une meilleure architecture de code, créer une classe Settings qui aura son propre .cpp / .h et dont le rôle sera de donner à l'algorithme les différents paramètres voulu pour l'exécution. Pour l'instant la classe Settings devra être capable de retourner une hauteur et une largeur via des getters, mais elle servira à récupérer de nouveaux paramètres au fur et à mesure de l'avancée du projet.

1.6 Créer la classe Settings et implémenter un constructeur de cette classe prenant en entrée argc et argv de la fonction main afin d'ensuite pouvoir récupérer hauteur et largeur via getters.

1.7 Créer une classe Screen qui permettra de manipuler l'écran simulé sur la console (application de Settings, fonction *Display()* et toute fonctionnalité jugée pertinente pour la gestion de l'écran). Si ce n'est pas déjà fait, créer un fichier .cpp et .h pour chacune de ces classes.

Partie 2 : Génération d'objet 3D

L'objectif de cette partie est d'être capable de générer différents objets de différentes résolutions et de les afficher à l'écran. Ces objets 3D peuvent être vus comme des Mesh qui possèdent simplement une liste de vertices. Plus la résolution du mesh est élevée, plus ce mesh contiendra de vertices.

2.1 Créer un fichier Mesh.cpp/.h dans lequel il y aura la définition d'une structure Vertex permettant de définir un point de l'espace 3D et une classe Mesh contenant un vecteur de Vertex. Implémenter une fonction *Debug()* dans la classe Mesh ainsi que dans la structure Vertex. Ces fonctions permettront de lister les points qui composent un mesh et leur coordonnées.
Pour tester ces fonctions, on pourra créer un mesh factice ayant des vertices en coordonnées {-1, -1, 0}, {-1, 1, 0}, {1, -1, 0} et {1, 1, 0}.

Ce mesh factice a une résolution de 4 car 4 vertices le composent. Si on voit ces vertices comme les coins d'un carré et qu'on souhaite que le mesh ressemble à un carré plein, on peut ajouter des vertices au sein de ce mesh pour le densifier. Par exemple, en notant n^2 la résolution, on peut décrire le mesh comme l'ensemble :

$$\left\{ \left(\frac{2i}{n-1} - 1, \frac{2j}{n-1} - 1, 0 \right), 0 \leq i < n, 0 \leq j < n \right\}$$

La manière de décrire un mesh n'est pas unique, c'est une solution parmi d'autres.

2.2 Ajouter un setting "MeshResolution", dont l'argument en ligne de commande sera -r et dont la valeur par défaut sera 32. Ce settings sera passé au constructeur de la classe Mesh.

2.3 Ajouter quatre fonctions à la classe Mesh : *GenerateCircle(float radius)*, *GenerateHalfCircle(float radius)*, *GenerateRectangle(float width, float height)* et *GenerateSquare(float size)*. La fonction *Debug()* implémentée précédemment permettra de vérifier les résultats obtenus.

Afin d'observer ces objets générés sur l'écran, il est nécessaire de projeter chaque vertex du mesh sur l'écran. Le projet d'A1k0n l'explique très bien, s'y référer pour plus d'information (le lien est en première page).

2.4 Dans la classe Scren, ajouter une fonction *Display(Mesh const& mesh)* afin d'afficher une projection du mesh sur l'écran. Ajouter également des settings "ScreenBackground" (-b, par

défaut ‘ ‘), “ScreenMeshProjection” (-p, par défaut ‘X’), “ScreenPosition” (-s, par défaut 9) et “ViewerPosition” (-v, par défaut 10) qui permettent d’obtenir les résultats suivants :