Édition 2023

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET

M DU PROJET : **CUB3**

Création d'Un rendU d'un CUBe en 3d vectorBalls

> PRÉSENTATION GÉNÉRALE :

• *Idée et objectifs :* L'idée était de réaliser un rendu d'un cube 3D en VectorBalls en langage python avec la librairie tkinter mais sans aucune librairie 3D, de contrôler les mouvements du cube de vectorballs 3D (boule en vectorisées en 3D) à l'aide du clavierdu PC, puis d'intégrer une interface homme-machine à l'aide d'une carte Arduino équipée de capteurs du type jauge de contrainte placés sur un gant permettant de prendre le contrôle des mouvements de l'objet 3D à l'écran en utilisant les 5 doigts de la main, enfin à l'aide d'un accéléromètre placé également sur le gant, prendre également le contrôle de l'objet pour des personnes ne pouvant pas mouvoir facilement les doigts (personnes âgées ou avec un handicap par exemple).

• *Origines et intérêts du projet :* L'intérêt du projet par rapport au programme de NSI est d'utiliser les algorithmes de tri (par insertion ou par sélection étudiés en cours) pour gérer l'illusion de la 3D sur le rendu 2D à l'écran en utilisant la méthode du Z-buffer voir la page suivante : https://fr.wikipedia.org/wiki/Z-buffer.
Dans un second temps nous avons modifié ce code afin de créer des objets en fils de fer et avec une gestion des faces cachées d'un objet 3D en utilisant le programme de mathématiques (calcul vectoriel).

> ORGANISATION DU TRAVAIL :

• *Présentation de l'équipe (prénom de chaque membre et rôle dans le projet) :*
Luka : Calcul des formules pour la projection tridimensionnelle sur un plan bidimensionnel
Léna : Gestion du tri des boules
Eliott : Rendu version fils de fer et des faces cachées
Johan : Gestion de l'interface homme-machine par carte Arduino avec des potentiomètres dans un premier temps puis en intégrant les jauges de contrainte.
Stefan : Création d'un effet miroir et gestion sur l'interface homme-machine des déplacements et mouvements de l'objet 3D avec un accéléromètre 3 axes.

• *Répartition des tâches* : Léna : Création du z-buffer par les algorithmes de tri,
Johan : Mise en place de l'interface homme-machine : manipulation des objets 3D à l'aide de potentiomètres puis des jauges de contraintes, Eliott : Gestion des objets en fils de fer (en remplacement des "vectorballs") et calcul de la visibilité des faces, Stefan : Création d'un miroir et ajout dans l'interface homme-machine de la manipulation à l'aide d'un accéléromètre 3 axes.

• *Organisation du travail (répartition par petits groupes, fréquence de réunions, travail en dehors de l'établissement scolaire, outils/logiciels utilisés pour la communication et le partage du code, etc.) :* Les "réunions" étaient lors des cours de NSI le plus souvent les lundis matins et vendredi après-midi.
Le logiciel pour coder en python que nous avons utilisé est Thonny, ainsi qu'une carte Arduino pour l'interface homme-machine, les entrées des différents capteurs (jauge de contrainte) ont été branchées à la carte. Le partage du code s'est effectué en grande partie à l'aide de clé USB et par mail.

ES ÉTAPES DU PROJET :

• *Présenter les différentes étapes du projet (de l'idée jusqu'à la finalisation du projet)*

Nous avons dans un premier temps appris la théorie sur la représentation 3D et les calculs mathématiques permettant d'obtenir la transformation de coordonnées 3D cartésiennes en coordonnées 3D sphérique.

Grâce à un squelette de code créé par notre professeur nous avons programmés la transformation 3D à l'aide de ces connaissances, nous avons ensuite appliqués cela à des boules graphiques aussi nommées vectorballs à l'aide du langage Python.

Nous avons intégré la gestion du calcul des profondeurs (Z-buffer) à l'aide des algorithmes de tri que nous avions déjà étudiés en cours, puis gérer les affichages des boules à l'écran en fonction des données triées par nos algorithmes.

Une fois que cela était fonctionnel, on a développé l'interface homme-machine en essayant de transférer une valeur donnée par un potentiomètre branché sur carte Arduino dans le code Python grâce à la librairie Serial.

Nous avons ensuite ajouté plusieurs potentiomètres pouvant modifier chacun un aspect du vectorballs 3D (orientation, position sur l'écran, zoom…). Ensuite Stefan a implémenté un accéléromètre pouvant remplacer les potentiomètres pour effectuer des rotations sur trois axes à l'aide de la transformation d'Euler et ainsi améliorer l'expérience de l'interface homme-machine 3D.

Cette nouvelle version pouvant être utiliser pour aider des personnes handicapées, ou en rééducation ou encore des personnes âgées à apprendre ou réapprendre les mouvements de la main droite ou de la main gauche.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

• *Avancement du projet (ce qui est terminé, en cours de réalisation, reste à faire) :* (Le 10 avril), le rendu en boule 3D est terminé, la version avec l'interface homme-machine (potentiomètres) est fonctionnelle, celle avec un accéléromètre et un gyroscope est bientôt finie. Il reste la fabrication du gant, pour améliorer l'expérience et permettre à des personnes âgées ou handicapées d'utiliser notre travail.

• *Approches mises en œuvre pour vérifier l'absence de bugs et s'assurer de la facilité d'utilisation du projet :* Nous avons réalisés des tests à chaque fois qu'on a implémente une nouvelle fonctionnalité pour s'assurer que cette dernière marchait comme prévu à l'aide d'assertion ou par vérification des valeurs en affichant les données calculées.

• *Difficultés rencontrées et solutions apportées :* Nous avons eu des problèmes de latence entre la carte Arduino et notre code python sous Thonny quand nous élaborons la version de l'interface homme-machine avec les potentiomètres, (moment entre une commande sur le potentiomètre et action sur le cube). La solution (trouvée par Stefan) à été d'ajouter un délai à la fin du code sur la carte Arduino.
Autre problème lors du passage au gyroscope pour la rotation, la question de la méthode de calcul d'angle s'est posée (solution de transformation d'Euler). Nous avons également rencontré un problème lors du rendu des faces visibles / cachées (calcul des vecteurs normaux aux surfaces).
Nous avons pu palier à ce problème en faisant des ajustements au niveau du point de fuite.

> OUVERTURE :

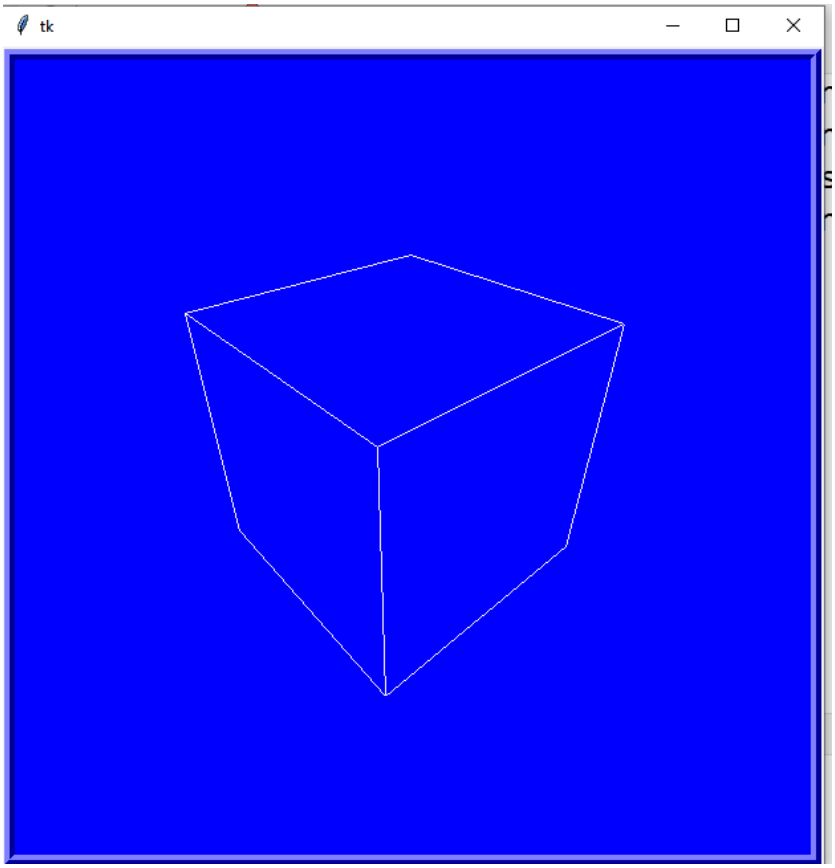
• *Idées d'améliorations (nouvelles fonctionnalités) :* Prise en charge du rendu des faces pour les formes concaves, l'algorithme de rendu ne fonctionne qu'avec les formes convexes. Ajouter également un remplissage des faces à l'aide d'algorithmes de polygones. Fonctionnement sans fil, utiliser le bluetooth, importation de fichier de forme et enregistrement de ce même fichier.

• *Stratégie de diffusion pour toucher un large public (faites preuve d'originalité !)* : Proposer un kit avec une carte Arduino et une version incomplète du code pour faire un "refaite le vous-même" dans l'optique d'une initiation à la programmation de niveau intermédiaire pour des élève de niveau première ou terminale NSI.

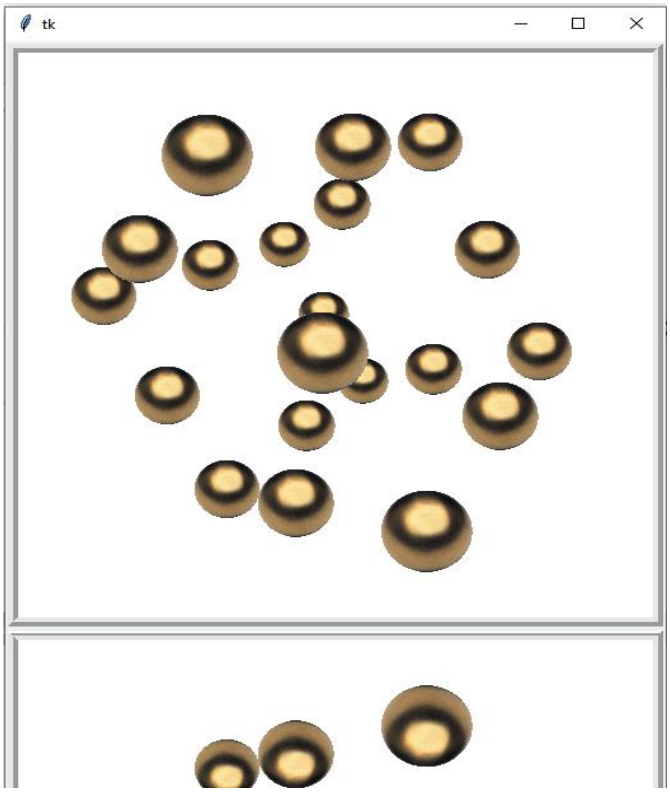
• *Analyse critique du résultat (si c'était à refaire, que changeriez-vous dans votre organisation, les fonctionnalités du projet et les choix techniques ?) :* Le gant est un peu épais, le travail aurait pu être un peu mieux répartis et le montage prends trop de place, il aurait pu être accroché sur le dos du gant si la carte avait été plus petite.

DOCUMENTATION

• Spécifications fonctionnelles (guide d'utilisation, déroulé des étapes d'exécution, description des fonctionnalités et des paramètres)
L'utilisateur peut appliquer rotations, déplacements et grossissements sur une forme en pliant ses doigts. Il peut également grâce à un accéléromètre déterminer l'angle de la forme sur trois axes (angles d'Euler). Un calcul des faces à été ajouté pour déterminer la visibilité de chaque face et les "vectorballs" initiales ont été remplacées par des lignes pour plus de fluidité.
• Spécifications techniques (architecture, langages et bibliothèques utilisés, matériel, choix techniques, format de stockage des données, etc) :
Bibliothèques : Tkinter, math, Serial (les outils de rendu 3d ont été recréés à la main)
Langage : Python, C++ (Arduino)
Matériel : HARDWARE : Carte arduino UNO, gant, potentiomètres, accéléromètre
SOFTWARE : Thonny, Arduino
Note d'utilisation : Le port spécifié lors de la connexion au serial doit être le même que celui qui connecte la carte ARDUINO UNO (par défaut COM3).
• Illustrations, captures d'écran, etc



Version Fil de fer et faces cachées



Version Vectorballs avec Z-Buffer

```python
#Module graphique de base (outils 3d non-intégrés)
from ...

##---Rendu des faces---##

#Transforme un tuple composé de deux éléments 2d considérés comme points en un vecteur partant du premier point pointant vers le second
def vectorialize(vect):
    return (pos_points_init[vect[1]][0] - pos_points_init[vect[0]][0],
            pos_points_init[vect[1]][1] - pos_points_init[vect[0]][1],
            pos_points_init[vect[1]][2] - pos_points_init[vect[0]][2])

#Transforme le tableau faces pour renvoyer un tableau normals contenant les vecteurs de faces.
def normalize(faces):
    normals = []
    for i in faces:
        a = vectorialize(i[0])
        b = vectorialize(i[1])
        normals.append((a[1] * b[2] - a[2] * b[1], a[2] * b[0] - a[0] * b[2], a[0] * b[1] - a[1] * b[0])) #Produit vectoriel de chaque couple de vecteurs pour obtenir les vecteurs de faces
    return normals

#Applique les même transformations que calcul_pos mais sur les vecteurs de faces en fonction des deux angles thêta et phi ainsi que le rayon de grossissement r.
def update_normals(normals, t, p, R):
    nnormals = []
    for i in normals:
        nnormals.append((-i[0] * sin(t) + i[1] * cos(t),
                         -i[0] * cos(t) * sin(p) - i[1] * sin(t) * sin(p) + i[2] * cos(p),
                         -i[0] * cos(t) * cos(p) - i[1] * sin(t) * cos(p) - i[2] * sin(p) + R))
    return nnormals

#Calcul la visibilité d'une face
def isVisible(normal, pos):
    global theta, phi, r
    vVector = pos[1] #Vecteur de vision, part de 0, 0, 0 pour aller à un point aléatoire du plan actualisé à chaque rotation
    scalarP = (normal[0] * vVector[0] + normal[1] * vVector[1] + normal[2] * vVector[2]) #Produit scalaire entre le vecteur de face et de vision
    if scalarP >= 81000000: #Valeur calculée grâce au point de fuite R et aux valeurs choisies pour la fonction calcul_proj
        return True
    else:
        return False

##---Traitement des lignes---##
#Actualisation des lignes : si la variable all se trouve seule dans le tableau de ligne, chaque point est relié au reste des points. Dans le cas contraire, le rendu des faces entre en jeu (voir tableau ligne)
def update_lines(tab, proj, norms,new_pos_points):
    canvas.delete('all') #Suppression des éléments du canvas pour redessiner chaque ligne
    if tab == ["all"]:
        for i in proj:
            for j in proj:
                if i != j:
                    canvas.create_line(i[0], i[1], j[0], j[1], fill="OldLace", width=0.1)
    else:
        if doesFaceRender: #Exécutée si le rendu des faces est activé
            for face in range(len(norms)):
                if isVisible(norms[face],new_pos_points): #Calcul de la visibilité de chaque face
                    for i in tab:
                        if i[2][0] == face or i[2][1] == face: #Si la ligne fait partie de la face visible, alors elle est affichée
                            canvas.create_line(proj[i[0]][0],proj[i[0]][1],proj[i[1]][0],proj[i[1]][1],fill="OldLace",width=0.1)
```