II. Mission et recherches approfondies

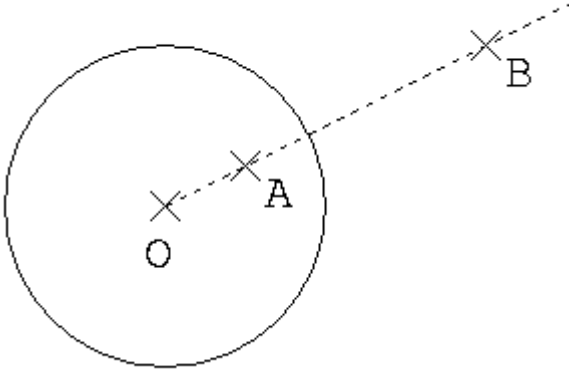
1. Recherches et définitions

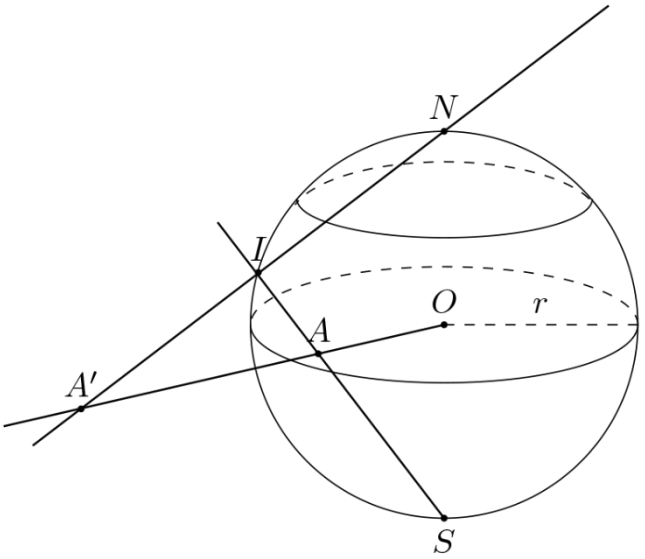
a. Inversion géométrique

L'inversion géométrique est une transformation géométrique qui associe à chaque point d'un espace, à l'exception d'un point spécial appelé « centre d'inversion », un autre point situé sur une droite de façon que le produit des distances du point original au centre d'inversion et du point image au centre d'inversion soit égal à un carré constant.

De manière plus détaillée, l'inversion géométrique se réalise en trois étapes principales :

- Choix du centre d'inversion : L'inversion d'un point est toujours réalisée par rapport au centre d'un cercle (2D) ou d'une sphère (3D) que l'on appelle centre d'inversion.
- Calcul de la distance : Pour chaque point P différent du centre O, on mesure la distance OP.
- Calcul des images inversées : Le point image P' correspondant à P est situé sur une droite passant par O et P, de telle sorte que le produit des distances OP et OP' soit égal à r^2 (le rayon au carré)





2. Mission

Après avoir bien assimilé le sujet, les concepts géométriques ainsi que le code source Python réalisé par mes collègues, j'ai pu commencer à faire quelques essais. Je me suis d'abord entraîné sur Matplotlib à réaliser une inversion d'un point par rapport à une sphère, afin de bien visualiser le fonctionnement de l'inversion géométrique.

Voici la fonction permettant de tracer une sphère :

```
def sphere(ax, rayon, centre):
    # Création des coordonnées de la sphère
    u, v = np.linspace(0, 2 * np.pi, 40), np.linspace(0, np.pi, 40)
    U, V = np.meshgrid(u, v)

    # Représentation paramétrique de la sphère
    X = rayon * np.sin(V) * np.cos(U) + centre[1]
    Y = rayon * np.sin(V) * np.sin(U) + centre[2]
    Z = rayon * np.cos(V) + centre[0]

    # Tracer la surface en 3D
    ax.plot_surface(X, Y, Z, color="pink", alpha=0.1)

    # Affichage des axes
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")

    return X, Y, Z
```

Elle prend un paramètre ax qui représente le plan sur lequel la figure va être tracée, un rayon, et un tableau de coordonnées en x,y,z pour le centre.

Ensuite on créée d'abord les coordonnées de la sphère en utilisant une grille de points dans le plan paramétrique (u, v).

Ensuite, les coordonnées en X, Y et Z sont placées dans cette grille de points afin de modéliser un ensemble de points, qui serviront ensuite à tracer la surface de la sphère à l'aide de la fonction ax.plot_surface.

u et v représentent respectivement les angles horizontaux et verticaux autour de la sphère. Pour u, les valeurs sont réparties uniformément entre 0 et 2π (angles horizontaux) et pour v, les valeurs sont réparties entre 0 et π (angles verticaux).

Ensuite la fonction meshgrid est utilisée pour créer deux tableaux bidimensionnels U et V qui contiennent toutes les combinaisons possibles de valeurs de u et de v. Ces tableaux sont ensuite utilisés pour définir la représentation paramétrique de la sphère en X, Y et Z.

Voici la fonction permettant de calculer les coordonnées inverses d'un point :

```
def inverse(X, Y, Z, rayon):
    x0, y0, z0 = 0, 0, 0  # coordonnées du centre de la sphère
    distance = (X - x0) ** 2 + (Y - y0) ** 2 + (Z - z0) ** 2

    if distance == 0:
        x_inv, y_inv, z_inv = X, Y, Z
    else:
        x_inv = (rayon**2) * (X - x0) / distance
        y_inv = (rayon**2) * (Y - y0) / distance
        z_inv = (rayon**2) * (Z - z0) / distance

    return x_inv, y_inv, z_inv
```

Elle prend en paramètre des coordonnées en X, Y et Z d'un point, ainsi qu'un rayon, afin de calculer puis retourner les coordonnées en X, Y et Z inversées d'un point.

Ici les coordonnées du centre sont établies à 0 car je décide de placer ma sphère au centre du plan.

Puis la distance entre le point et le centre est calculée à l'aide d'une formule.

Si la distance est égale à 0, c'est que le point est situé au centre, l'inverse de ce point est non défini, donc il ne se passe rien.

Sinon, on calcule les coordonnées inverses à l'aide de la formule, puis on retourne les coordonnées inversées en X, Y et Z.

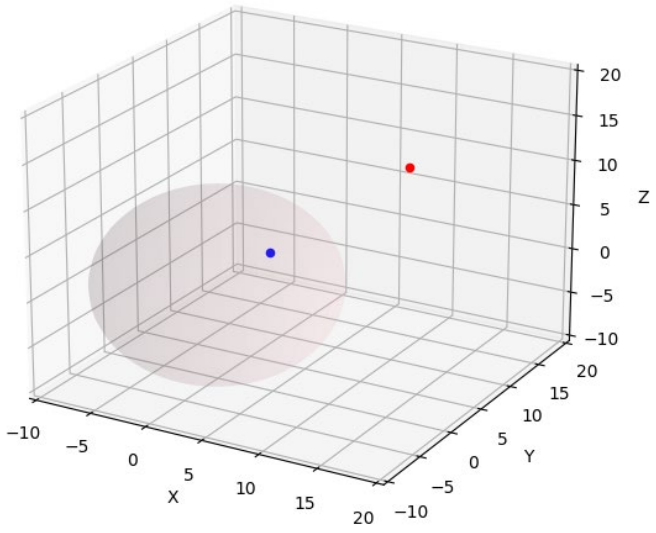
Une dernière fonction permet de tracer un point en fournissant un ensemble de coordonnées :

```
def plot_point(ax, x, y, z, color="red", marker="o"):
    ax.scatter(x, y, z, color=color, marker=marker)
```

En utilisant la fonction scatter, sur l'axe ax, et en lui fournissant les coordonnées en x, y, z du point, ainsi que quelques paramètres concernant l'esthétique du point. (couleur, forme etc)

Puis ces fonctions sont appelées en fournissant les coordonnées du point initial, ainsi que le rayon de la sphère.

Le résultat obtenu est le suivant :



Plus le point est proche du centre, plus son inverse sera éloigné, et plus le point est proche de la surface de la sphère, plus son inverse sera proche. Lorsque le point est situé au même endroit que le centre d'inversion, la distance entre le point et le centre est nulle. Par conséquent, la division par zéro se produit lorsqu'on essaie de calculer l'inversion du point, ce qui n'est pas défini mathématiquement.

En d'autres termes, on ne peut pas inverser un point qui est situé exactement au centre du cercle (ou sphère) d'inversion. Cependant, dans certains contextes, on peut considérer que le point au centre du cercle est transformé en un point à l'infini dans la direction de la droite d'inversion. Cette notion est liée à la géométrie projective, où des points à l'infini sont introduits pour compléter l'espace géométrique.

Après cela, Monsieur Jouk m'a proposé de travailler sur Blender, car cela peut s'avérer être un outil formidable pour la suite du projet. Blender nous permettrait de créer des modélisations beaucoup plus professionnelles et didactiques, permettant ainsi une meilleure compréhension.

J'ai alors commencé à me familiariser avec l'outil, et j'ai également dû me documenter sur l'API (**A**pplication **P**rogramming **I**nterface) appelée Blender Python API (bpy).

Je me suis donc concentré lors de cette deuxième semaine, sur la réalisation d'une inversion d'un cube par rapport à un centre et un rayon donné. Je fournis directement une valeur pour le rayon et je mets les coordonnées du centre à 0, 0, 0. Je ne récupère pas les coordonnées d'une sphère pour le moment, je fournis ces valeurs « en brut » car ce n'est pas nécessaire d'avoir une sphère pour faire l'inversion, les seuls paramètres nécessaires sont un rayon et un centre.

Voici comment j'ai procédé :

```
import bpy
import bmesh
import math

def inverse(point, rayon):
    # Coordonnées du centre
    x0, y0, z0 = 0, 0, 0

    # Calcul de la distance entre les points
    distance = math.sqrt((point[0] - x0) ** 2 + (point[1] - y0) ** 2 +
(point[2] - z0) ** 2)

    # Calcul des coordonnées de l'inverse géométrique
    x_inv = (rayon ** 2) * (point[0] - x0) / distance ** 2
    y_inv = (rayon ** 2) * (point[1] - y0) / distance ** 2
    z_inv = (rayon ** 2) * (point[2] - z0) / distance ** 2

    return [x_inv, y_inv, z_inv]

def remplacement_coord():
    # Activation du mode edit
    if bpy.context.scene.objects.find('Cube') >= 0:
        cube = bpy.context.scene.objects['Cube']
        bpy.context.view_layer.objects.active = cube
        bpy.ops.object.mode_set(mode='OBJECT')
        bpy.ops.object.select_all(action='DESELECT')
        bpy.ops.object.mode_set(mode='EDIT')
        bpy.context.view_layer.update()

    bm = bmesh.from_edit_mesh(cube.data)
    bm.verts.ensure_lookup_table()  # Mise à jour de l'indice interne

    # Appliquer la fonction inverse à chaque sommet du cube
    for vertex in bm.verts:
        old_x, old_y, old_z = vertex.co
        new_coords = inverse((old_x, old_y, old_z), rayon)
        vertex.co = new_coords

    bmesh.update_edit_mesh(cube.data)  # Applique les changements au mesh

rayon = 3.0
remplacement_coord()
```

j'ai donc récupéré la fonction inverse. Puis j'ai dû créer la fonction remplacement_coord() afin de pourvoir effectuer des modifications sur l'objet sphère.

Afin de récupérer les données d'un objet (arêtes, sommets, coordonnées etc) il faut passer cet objet en mode « edit » . Puis je déclare bm qui est une instance de l'objet bmesh qui lui-même est une structure de données permettant de représenter et manipuler les données géométriques d'un

maillage (mesh). Celles qui nous intéressent sont les sommets, car ce sont les points que l'on va inverser afin d'obtenir l'inversion du cube.

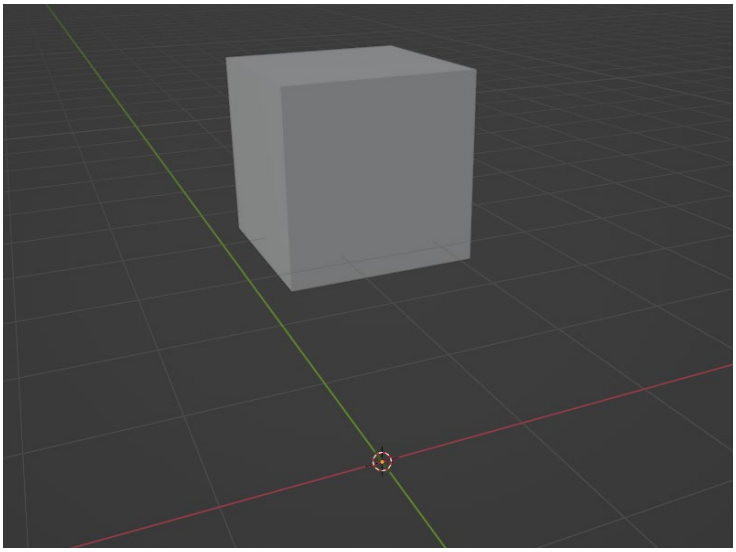
J'utilise ensuite une boucle qui itère sur chaque sommet du maillage « bm ». bm.verts est une collection contenant tous les sommets du maillage (du cube).

J'assigne ensuite les coordonnées des sommets mon cube à l'aide vertex.co qui contient les coordonnées en x, y, z de chaque sommet du cube.

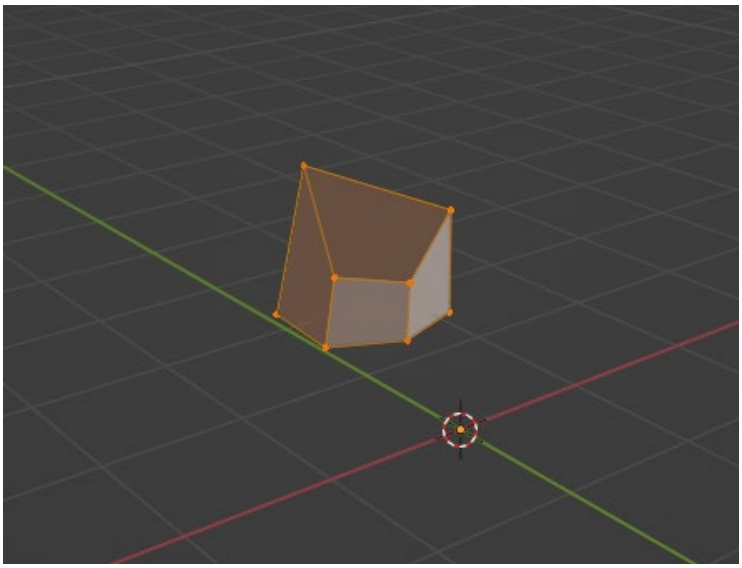
Puis je calcule leur inverse, et je remplace les coordonnées actuelles de mon cube par les coordonnées inversées.

Voici le résultat,

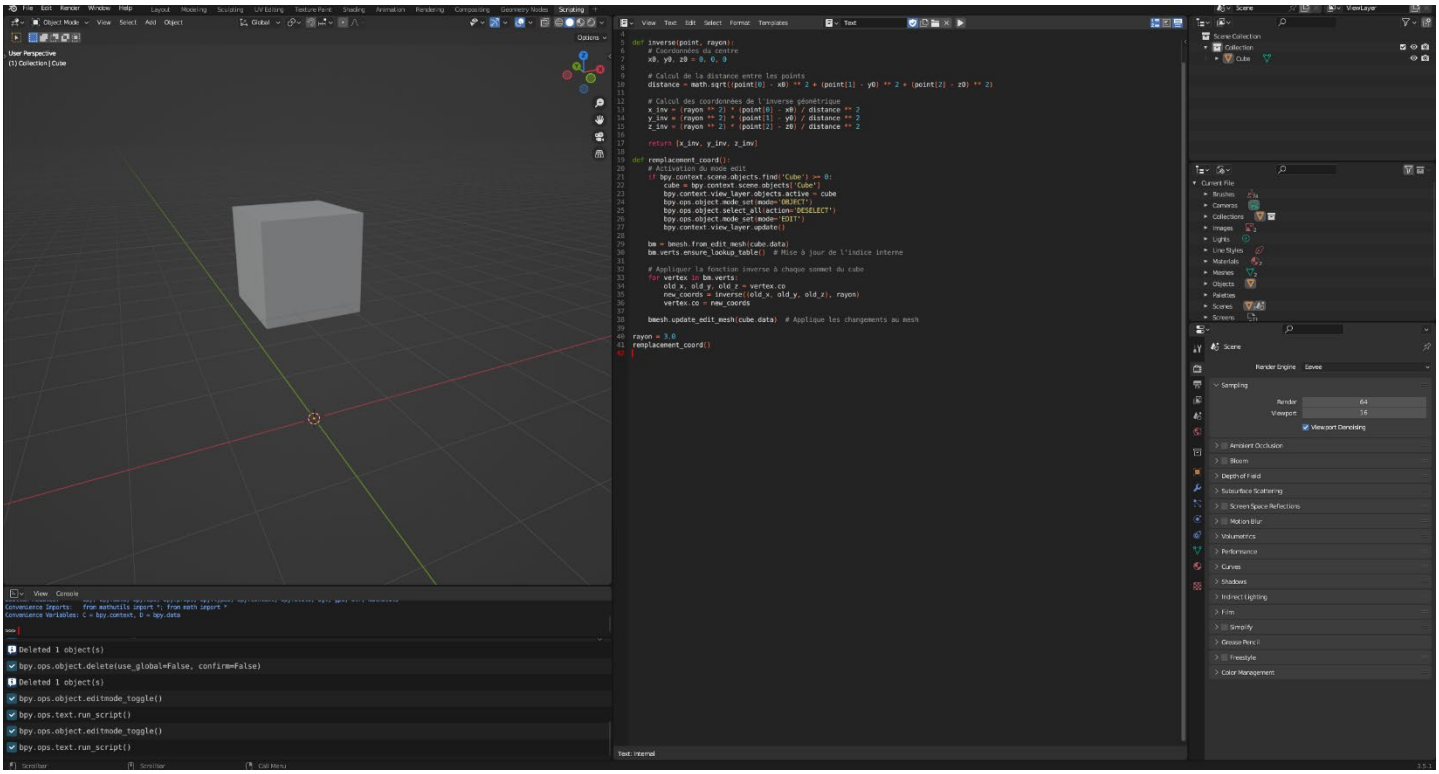
Avant inversion :



Après inversion :



Et voici à quoi ressemble l'interface, lorsque je me place dans l'onglet « Scripting » :



Conclusion

Cette deuxième semaine a été pour moi l'occasion de réaliser mes premiers scripts, de tester et modéliser différentes formes, et expérimenter l'opération d'inversion géométrique afin de bien intégrer ce concept. J'ai pu découvrir Blender, qui est un outil remarquable, et très puissant qui je pense, sera d'une grande utilité pour la suite si je parviens à le maîtriser.