II. Mission et recherches approfondies

1. Recherches et définitions

a. Géodésiques

En géométrie, les géodésiques sont les courbes qui représentent les chemins les plus courts entre deux points sur une surface courbe.

Par exemple sur l'image ci-dessous qui est une carte en projection Mercator représentant les routes entre Paris et New-York, c'est la route en rouge qui est le chemin le plus court, car cela est dû à la courbure de la planète Terre, pour cette raison si on était amené à parcourir cette distance il vaudrait mieux emprunter le chemin rouge que le chemin bleu ! (On appelle cela des routes loxodromiques en bleu et routes orthodromiques en rouge)

2. Mission

Cette semaine, l'objectif principal était d'obtenir : Une modélisation des géodésiques parcourant 3 cyclides de Dupin imbriqués.

Nous sommes donc passés par 4 étapes :

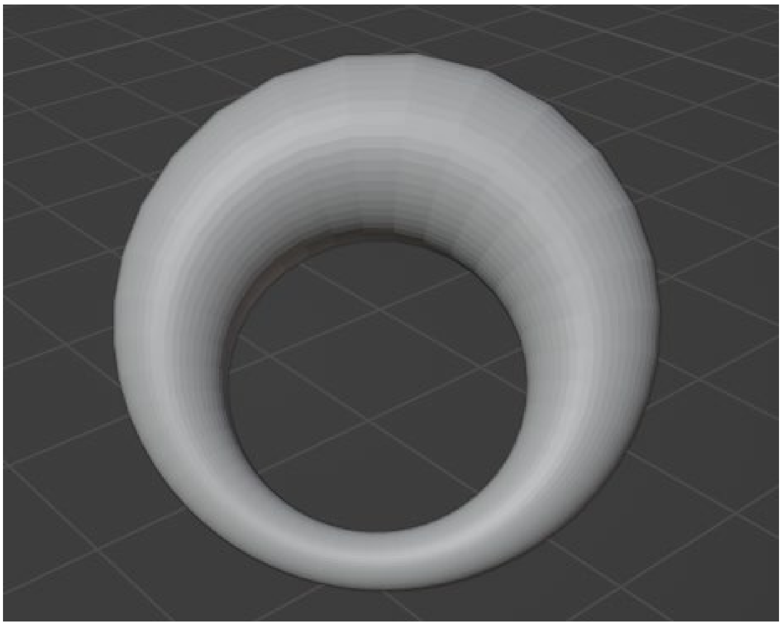
- Inversion du tore pour obtenir la cyclide de Dupin
- Imbrication des tores
- Cyclides imbriqués
- Calcul des géodésiques parcourant les cyclides de Dupin imbriqués

a. Inversion du tore

Pour obtenir la cyclide de Dupin, il faut effectuer l'inversion géométrique du tore, pour cela j'ai utilisé le même script que j'avais fait la semaine dernière permettant d'inverser un cube, j'ai simplement récupéré les coordonnées des sommets du tore cette fois-ci au lieu du cube.

```
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():

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

    # Appliquer la fonction inverse à chaque sommet du tore
    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(torus.data)  # Applique les changements au mesh

rayon = 3.0
remplacement_coord()
```

Et voici le résultat obtenu :



b. imbrication des tores

Maintenant il faut générer une série de tores imbriqués, cela permettra par la suite lors de la modélisation des géodésiques, d'obtenir plus de détails et de précision, afin de faciliter l'apparitions de patterns, ce que Mr Jouk recherche.

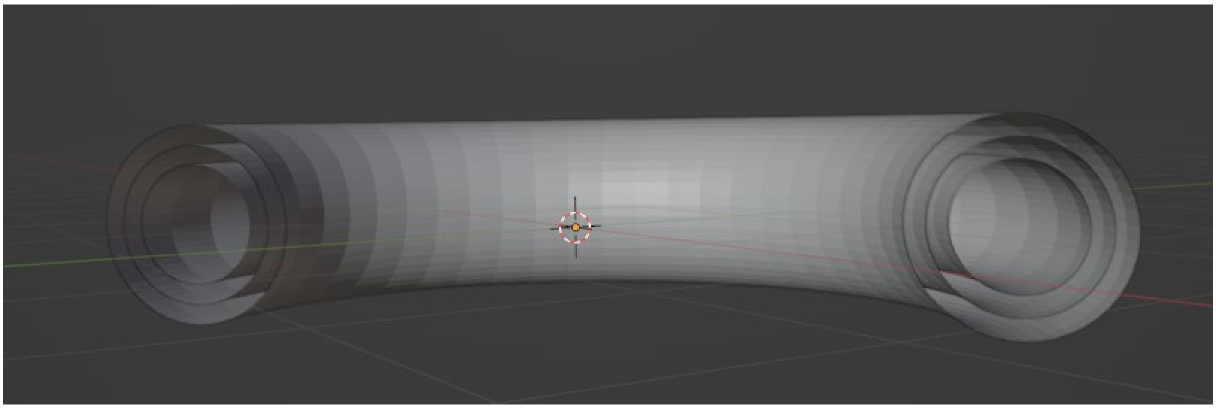
Voici comment j'ai pu réaliser cette tâche :

```
def generate_nested_toruses(radius_step, major_segments, minor_segments):

    num_toruses = 3  # Nombre de tores imbriqués
    # Créer les trois tores imbriqués
    for i in range(num_toruses):
        # Calculer le rayon des petits cercles composant le tore
        minor_radius = 0.3 + i * radius_step

        # Créer un nouvel objet tore
        bpy.ops.mesh.primitive_torus_add(
            align='WORLD',
            location=(0, 0, 0),  # Coordonnées du tore
            rotation=(math.radians(0), math.radians(0), math.radians(0)), #
Si on veut appliquer une rotation
            major_radius=1.0,  # Rayon principal du tore
            minor_radius=minor_radius,  # Rayon des petits cercles
composant le tore
            major_segments=major_segments,
```

```
            minor_segments=minor_segments
        )

    # Afficher le résultat dans la vue 3D
    bpy.context.view_layer.update()

# Utilisation de la fonction
generate_nested_toruses(0.1, 8, 8)
```

Je peux choisir le nombre de tores à générer, puis je définis dans la boucle les paramètres des tores à générer. Lors de l'appel de la fonction je choisis un radius step qui permettra d'établir l'espacement entre chaque tore, puis le nombre de segments majeurs et mineurs, qui définissent la précision du tore, plus le chiffre est élevé, plus il y aura de petits cercles et de points, mais cela augmente considérablement les ressources nécessaires afin d'effectuer les calculs des géodésiques par la suite.

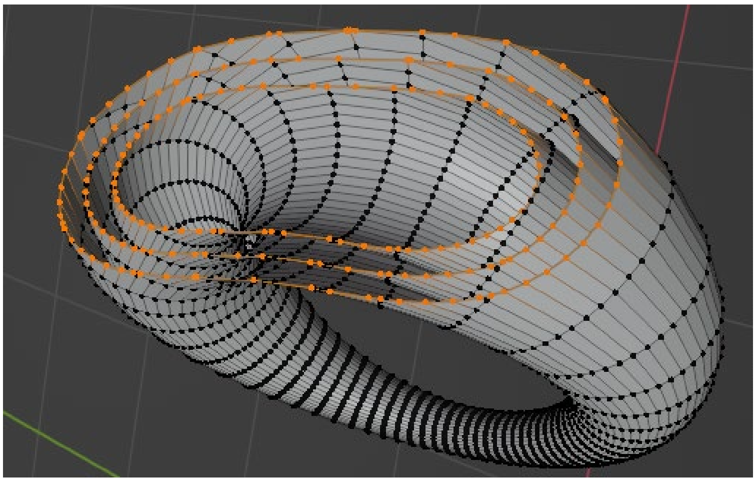
Voici une image du résultat :



J'ai sectionné les tores en deux afin de mettre en évidence l'imbrication de ces derniers.

c. Cyclides imbriqués

Maintenant que l'on peut imbriquer des tores, il faut effectuer l'inversion de nouveau, et on obtiendra nos cyclides de Dupin imbriqués comme ceci :



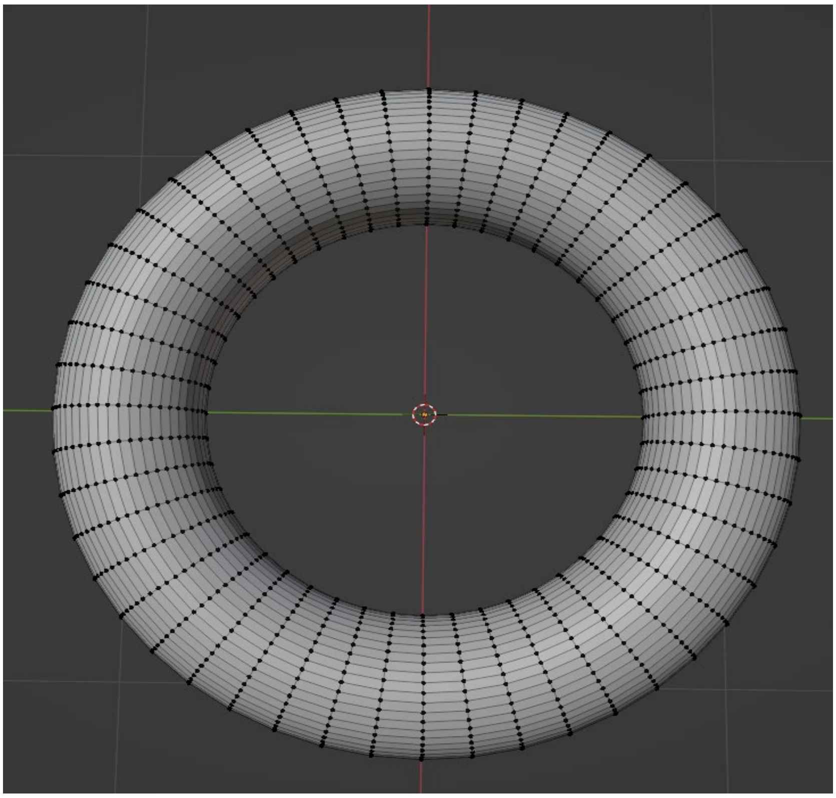
Je les ai sectionnés par le haut afin de laisser apparaître l'imbrication

Pour effectuer une inversion sur plusieurs tores, j'ai adapté mon script qui effectue l'inversion d'un seul tore, en ajoutant simplement une collection contenant tous les tores de la scène, puis une boucle qui applique l'inversion à chaque tore contenu dans la collection :

```
def inverse_coord_for_all_tores():
    # Récupére tous les objets de type MESH dont le nom commence par
"Torus"
    tores = [obj for obj in bpy.context.scene.objects if obj.type == 'MESH'
and obj.data.name.startswith("Torus")]

    # Parcourir tous les tores présents dans la scène
    for torus in tores:
        # Activer le mode édition
        bpy.context.view_layer.objects.active = torus
        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(torus.data)
        bm.verts.ensure_lookup_table()  # Mise à jour de l'indice interne

        # Appliquer la fonction inverse à chaque sommet du tore
        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
```

d. Calcul des géodésiques

Maintenant que l'on peut obtenir des cyclides de Dupin imbriqués, la dernière étape est la modélisation des géodésiques des cyclides imbriqués. L'opération d'inversion géométrique ne permet pas de conserver les géodésiques initialement tracées, donc si l'on trace les géodésiques des tores, et que l'on effectue leur inversion, on n'obtiendra pas le résultat souhaité, il faut d'abord effectuer l'inversion des tores, puis calculer les géodésiques des cyclides de Dupin. Sur Blender, les tores sont modélisés de façon discrète, On a donc un nombre de points et de paramètres bien définis que l'on peut changer, permettant ainsi de manipuler notre figure comme on le souhaite, voici une image montrant comment le tore est modélisé sur Blender :



Ici j'ai généré un tore 36 * 36, c'est dire qu'il est composé de 36 petits cercles, avec 36 points sur chacun de ces cercles, on peut bien apercevoir les cercles qui le composent, avec les points, ce qui nous permettra de calculer les géodésiques passant par ces points.

C'est mon collègue qui a principalement travaillé sur la partie concernant le calcul des géodésiques, il a d'abord réalisé ses fonctions sur python et matplotlib, puis nous avons travaillé ensemble afin de porter son travail sur Blender.

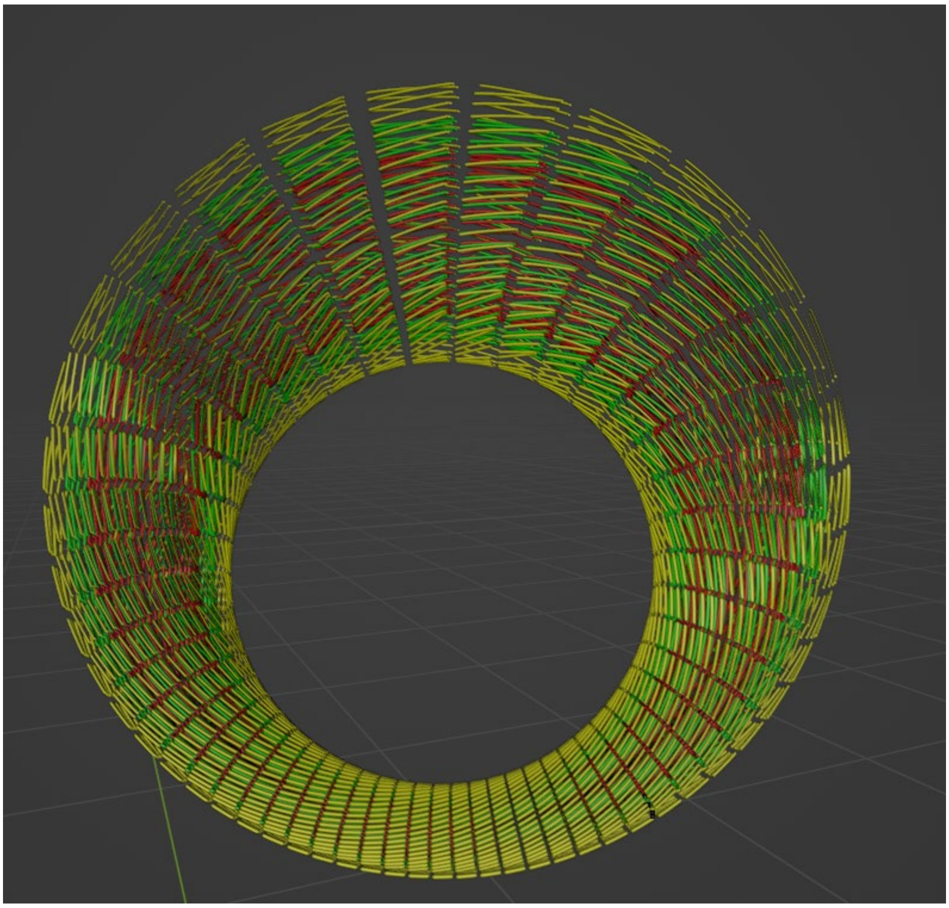
Voici la première fonction qui prend en paramètre 3 points initiaux qui se trouvent chacun sur un cercle distinct et adjacent, dont un point qu'on appelle point courant, et retourne le rayon du cercle circonscrit au triangle formé par ces 3 point :

```
def calcul_parametres_cercle(Point1, Point2, Point3):
    # Extraction des coordonnées x, y et z de chaque point
    x1, x2, x3 = Point1[0], Point2[0], Point3[0]
    y1, y2, y3 = Point1[1], Point2[1], Point3[1]
    z1, z2, z3 = Point1[2], Point2[2], Point3[2]

    # Calcul des coordonnées du centre du cercle
    Mx = (x1 + x2 + x3) / 3
    My = (y1 + y2 + y3) / 3
    Mz = (z1 + z2 + z3) / 3

    # Calcul des vecteurs AB et AC
    AB = [x2 - x1, y2 - y1, z2 - z1]
    AC = [x3 - x1, y3 - y1, z3 - z1]

    # Calcul du produit vectoriel des vecteurs AB et AC pour obtenir le
vecteur normal
    N = [AB[1] * AC[2] - AB[2] * AC[1], AB[2] * AC[0] - AB[0] * AC[2],
AB[0] * AC[1] - AB[1] * AC[0]]

    # Calcul de la norme du vecteur normal
    NN = np.sqrt(N[0] * N[0] + N[1] * N[1] + N[2] * N[2])

    # Normalisation du vecteur normal
    N_normalisé = [N[0] / NN, N[1] / NN, N[2] / NN]

    # Calcul de la projection du centre M sur le vecteur normal N
    MN = Mx * N[0] + My * N[1] + Mz * N[2]

    # Calcul des coordonnées du centre du cercle
    Centrex = Mx - (MN * N[0])
    Centrey = My - (MN * N[1])
    Centrez = Mz - (MN * N[2])

    # Calcul du rayon du cercle
    r = np.sqrt((Centrex - x2) * (Centrex - x2) + (Centrey - y2) * (Centrey
- y2) + (Centrez - z2) * (Centrez - z2))

    return r
```

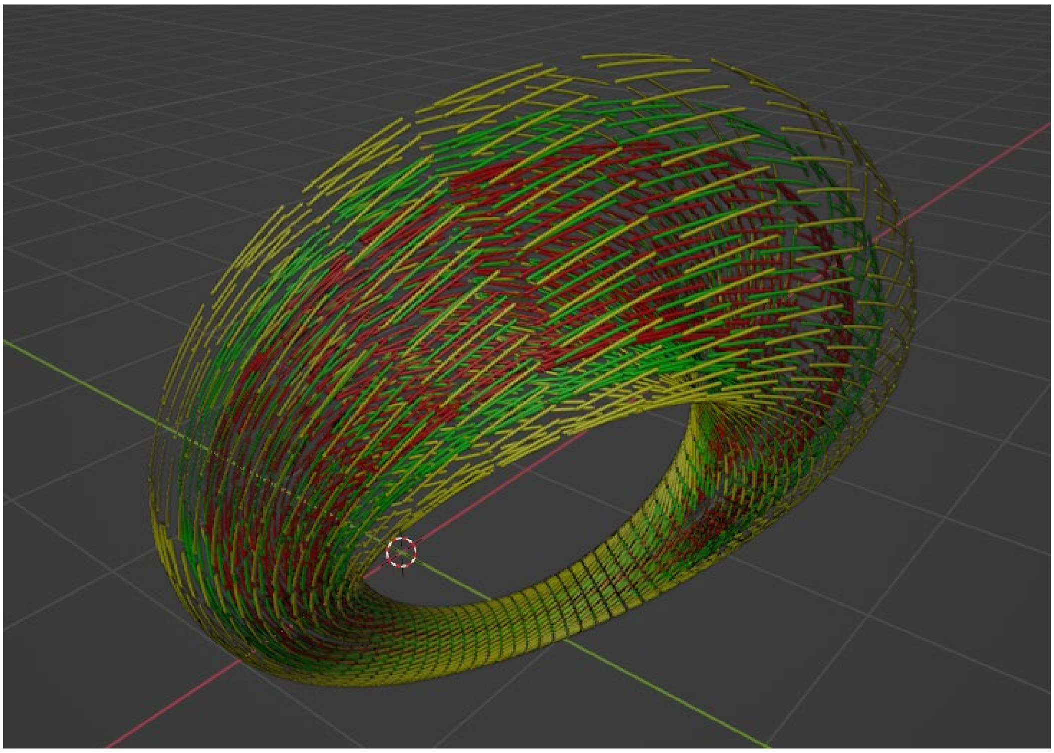
Puis la deuxième fonction, à l'aide de boucles, détermine tous les rayons possibles entre 3 points, en faisant varier seulement les deux points, pas le point courant, et seul le rayon le plus petit est retenu, ce rayon-là est en fin de compte la géodésique entre le point courant et les deux autres points, puis cette géodésique est modélisée. Le script permet donc d'itérer sur tous les points du tore, et de déterminer les géodésiques à l'aide de boucles imbriquées, puis de les tracer.

Vous pouvez la trouver en *annexe*

Voici quelques images du rendu final :



J'ai choisi 3 couleurs différentes afin de bien distinguer les géodésiques appartenant à chaque cyclide.



Ici je les ai sectionnés par le haut afin de bien apercevoir les trajectoires prises par les géodésiques à l'intérieur :