Aile

Minh-Hoang DANG

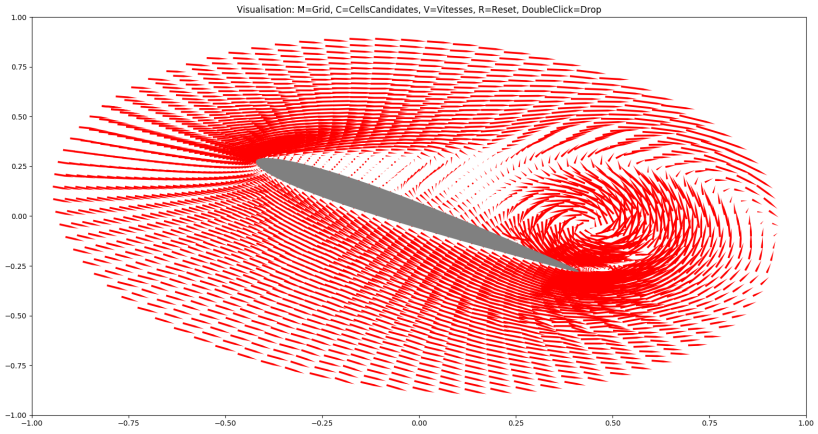
April 14, 2019

1 Matplotlib

Tout le projet sera placé dans une classe `WindowsViz` afin d'optimiser l'organisation

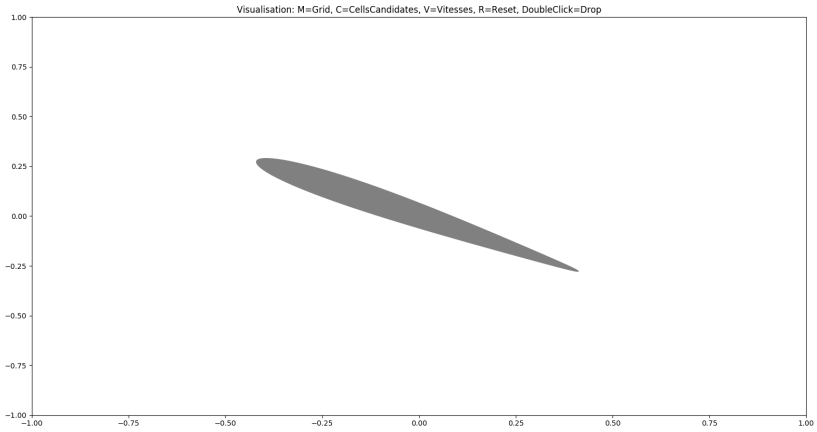
```
In [ ]: class WindowsViz:

            def __init__(self):
                self.__coords, self.__values = charger_objet("aile")
                self.__xmin, self.__xmax = min(self.__coords[:, 0]), max(self.__coords[:, 0])
                self.__ymin, self.__ymax = min(self.__coords[:, 1]), max(self.__coords[:, 1])

                self.__xs, self.__ys, self.__us, self.__vs, self.__speeds = self.recalibrate()

                self.__dispVec, self.__dispGrid, self.__dispCellule = False, False, False
                self.__dropPoints = []
                self.__cells = self.calcCells()

                self.__strm = None
                self.__beginSection = None
                self.__endSection = None

                print("WindowsViz created successfully!")
```

1.1 Fonction pour afficher le champs de vecteurs



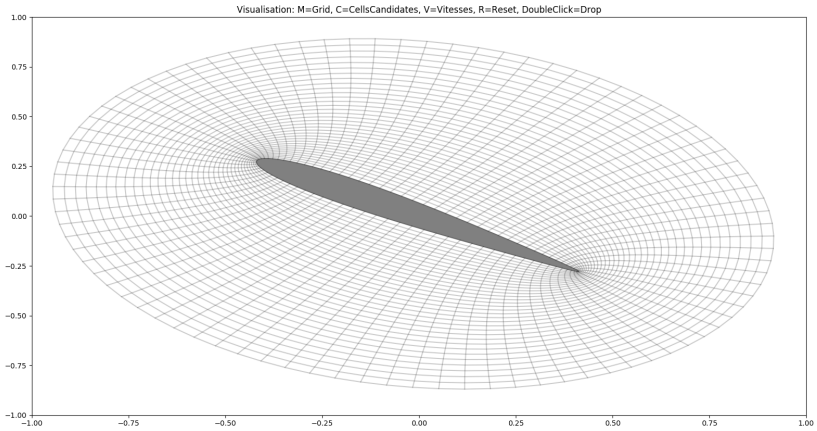

```
In [ ]: # Afficher le champs de vecteurs
        def afficher_vitesse(self):
            x, y, u, v = self.__coords[:, 0], self.__coords[:, 1], self.__values[:, 0],
        self.__values[:, 1]
            ax.quiver(x, y, u, v, color="red", headwidth=1, scale=35)
```

1.2 Fonction pour afficher le profil



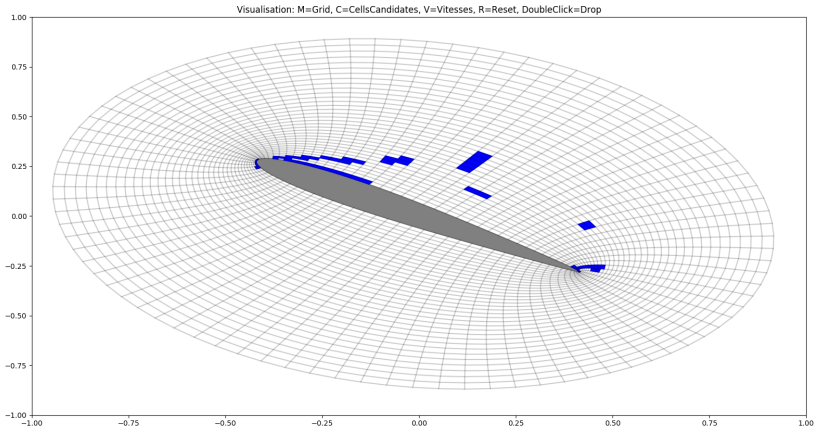

```
In [ ]: # Afficher le profil
        def afficher_profil(self, color):
            verts = self.__coords[0:len(self.__coords):35]
            poly = Polygon(verts)
            poly.set_color(color)
            ax.add_patch(poly)
```

1.3 Fonction pour afficher le maillage




```
In [ ]: # Afficher le maillage
        def afficher_maillage(self):
            tabX, tabY = np.array(self.__coords[:, 0]), np.array(self.__coords[:, 1])

            # Vertical lines
            for n in range(0, 35 * 87, 35):
                xs, ys = tabX[n:n + 35], tabY[n:n + 35]
                plt.plot(xs, ys, c=(0, 0, 0, .2))

            # Horizontal lines
            for n in range(0, 35):
                xs, ys = tabX[n:len(tabX):35], tabY[n:len(tabY):35]
                plt.plot(xs, ys, c=(0, 0, 0, .2))
```

1.4 Fonction pour affcicher les cellules candidates




```
In [ ]: # Afficher les cellules candidates:
        def afficher_cellules(self, color):

            # test sign
            def test_sign(vectors):
                for i in range(len(vectors)-1):
                    vec1, vec2 = vectors[i], vectors[i+1]

                    # Si le produit scalaire est négatif, alors les vecteurs sont opposés
                    if np.dot(vec1, vec2) < 0:
                        return False

                return True

            # Begin
            for cell in self.__cells:
                x1, x2, x3, x4 = cell[:, 0]
                y1, y2, y3, y4 = cell[:, 1]
                u1, u2, u3, u4 = cell[:, 2]
                v1, v2, v3, v4 = cell[:, 3]

                # Les 4 vecteurs
                vecs = [ [u1, v1], [u2, v2], [u3, v3], [u4, v4] ]

                if not test_sign(vecs):
                    verts = [ (x1, y1), (x2, y2), (x3, y3), (x4, y4) ]
                    poly = Polygon(verts)
                    poly.set_color(color)
                    ax.add_patch(poly)
```

1.5 Fonction pour charger les objets

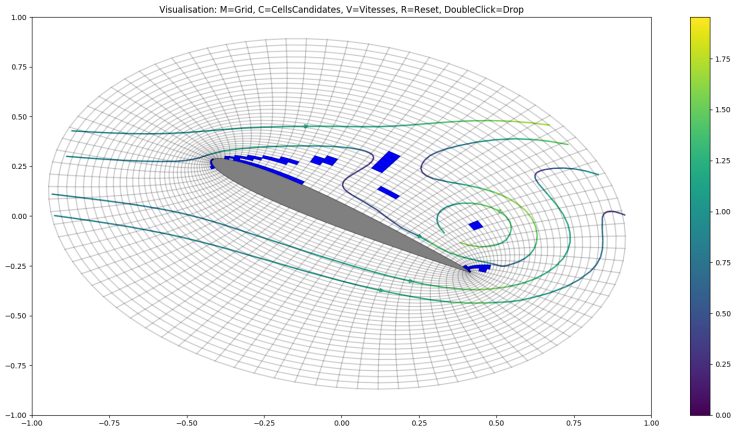
```
In [ ]: def charger_objet(obj, limit=0):
            coords = []
            values = []
            count = 0

            file = open(obj, "r")
            angle = 0.
            x, y = 0., 0.
            u, v = 0., 0.

            R = np.zeros((2, 2))
```

```
            for line in file.readlines():
                data = line.split()

                if count == 0:
                    angle = float(data[2])

                    theta = np.radians(angle)
                    c, s = np.cos(theta), np.sin(theta)
                    R = np.array(((c, -s), (s, c)))

                    count += 1
                    continue

                x, y, u, v = float(data[0]), float(data[1]), float(data[2]), float(data[3])

                # Rotation matrix
                coords.append(np.matmul(R, np.array((x, y))))
                values.append(np.matmul(R, np.array((u, v))))

                if limit != 0 and count == limit: break
                count += 1

            return np.array(coords), np.array(values)
```

1.6 Fonction pour afficher les objets

```
In [ ]: # Afficher l'objet
        def afficher_objet(self):

            plt.cla()

            self.afficher_profil("grey")
            if self.__dispVec:
                self.afficher_vitesse()
            if self.__dispGrid:
                self.afficher_maillage()
            if self.__dispCellule:
                self.afficher_cellules("blue")

            #print([min(self.__coords[:, 0]), max(self.__coords[:, 0]), min(self.__coords[:,
        1]), max(self.__coords[:, 1])])
            ax.set_xlim(-1, 1)
            ax.set_ylim(-1, 1)

            # draw lines
            self.calcul_lignes()

            plt.title('Visualisation: M=Grid, C=CellsCandidates, V=Vitesses, R=Reset,
        DoubleClick=Drop')
            plt.show()
```

1.7 Fonction pour simuler le lâchage des particules dans le champ des vitesses



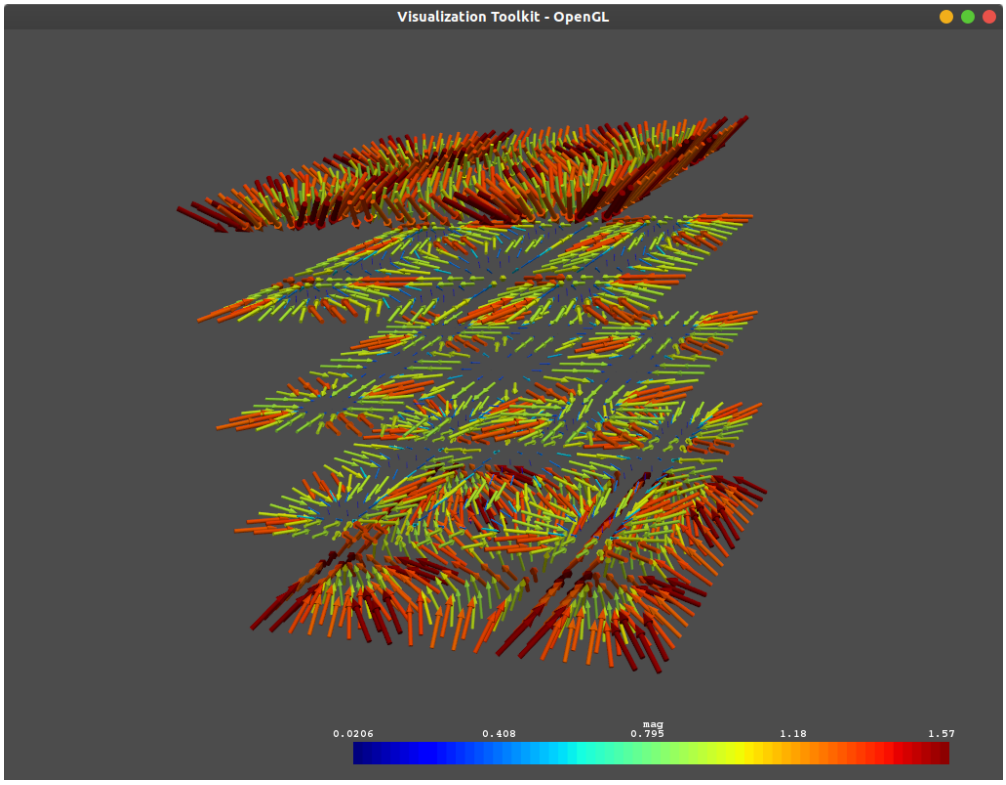

```
In [ ]: # calcul_lignes
        def calcul_lignes(self):
            if(len(self.__dropPoints) > 0):
                strm = ax.streamplot(self.__xs, self.__ys, self.__us, self.__vs,
        start_points=self.__dropPoints, color=self.__speeds, linewidth=2, cmap="viridis")
                if self.__strm is None:
                    self.__strm = strm
                    self.__colorbar = fig.colorbar(strm.lines)
```

2 PyVTK

Pour cette partie, nous allons utiliser VTKI, une interface de VTK pour Python. Elle offre les fonctionalités ressemblables à celles de Matplotlib, avec une documentation assez élaborée.

Malgré tout, je n'ai pas eu le temps nécessaire pour adapter le code fait avec Matplotlib à VTKi, on voit encore dans le fichier `vtki_viz.py`, les traces de ceux qui ont été réalisés.

Cependant, on peut voir un exemple de code qui permet de afficher un champs de vecteurs:




```
In [ ]: import vtki
        import numpy as np

        # Make a grid:
        x, y, z = np.meshgrid(np.linspace(-5, 5, 20),
                              np.linspace(-5, 5, 20),
                              np.linspace(-5, 5, 5))

        grid = vtki.StructuredGrid(x, y, z)

        vectors = np.sin(grid.points)**3


        # Compute a direction for the vector field
        grid.point_arrays['mag'] = np.linalg.norm(vectors, axis=1)
        grid.point_arrays['vec'] = vectors

        # Make a geometric obhect to use as the glyph
        geom = vtki.Arrow() # This could be any dataset

        # Perform the glyph
        glyphs = grid.glyph(orient='vec', scale='mag', factor=0.8, geom=geom)

        # plot using the plotting class
        p = vtki.Plotter()
        p.add_mesh(glyphs)
        p.show()
```

3 Difficulés rencontré

3.1 Temps

Avec autres projets, le temps dédié à ce projet est moins par rapport que j'ai plannifié et j'arrête de travailler sur ce projet à partir du 15 Avril (stage)

3.2 Python

Beaucoup de temps ont été consacrés à la recherche des fonctions Numpy, Scipy, Matplotlib pour ne pas ré-inventer la roue. Le code semble simple mais c'est le résulat des multitudes de correction et tests.

3.3 VTK

VTK est beaucoup moins facil à apprendre et à déployer que Matplotlib, malgré ses fonctionalités intéressante.