`maniflow` – a (partial) documentation

Felix Widmaier

1 Introduction

Definition 1 (Mesh)**.** *Let V be a vector space over $\mathbb{R}$ of dimension n. Let $\mathcal{V}_M \subset V$ be a set of points in V. We further let $\mathcal{F}_M \subset \mathcal{V}_M^3$. The pair $M = (\mathcal{V}_M, \mathcal{F}_M)$ is then called mesh. The elements of $\mathcal{V}_M$ are called points of M and the elements of $\mathcal{F}_M$ are the faces of the mesh M.*

For a mesh $M = (\mathcal{V}_M, \mathcal{F}_M)$ we will often denote $V_M = |\mathcal{V}_M|$ and $F_M = |\mathcal{F}_M|$.

Remark. Meshes M can be considered as 2-dimensional simplicial complexes. Thus for 2-dimensional manifolds $\tilde{M} \subset V$ we may find a *triangulation* simplicial complex K of $\tilde{M}$. The corresponding mesh will be called *triangulation* mesh of the manifold $\tilde{M}$.

Example 1 (Tetrahedron)**.** *Let*

$$\mathcal{V} = \left\{ \left(\sqrt{\frac{8}{9}}, 0, -\frac{1}{3}\right), \left(-\sqrt{\frac{2}{9}}, \sqrt{\frac{2}{3}}, -\frac{1}{3}\right), \left(-\sqrt{\frac{2}{9}}, -\sqrt{\frac{2}{3}}, -\frac{1}{3}\right), (0,0,1) \right\} \subset \mathbb{R}^3$$

and $\mathcal{F} = \{f \in 2^{\mathcal{V}} : |f| = 3\}$. The mesh $T = (\mathcal{V}, \mathcal{F})$ is the tetrahedron, which is displayed in figure 1. This

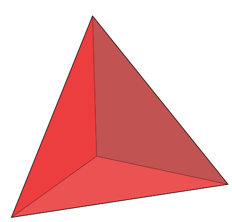


Figure 1: Tetrahedron

can be implemented using `maniflow` by using the `Mesh` class:

```
import numpy as np
import itertools
from maniflow.mesh import Mesh, Face

# computing the four vertices of the tetrahedron
v1 = np.array([np.sqrt(8/9), 0, -1/3])
v2 = np.array([-np.sqrt(2/9), np.sqrt(2/3), -1/3])
v3 = np.array([-np.sqrt(2/9), -np.sqrt(2/3), -1/3])
```

```
v4 = np.array([0, 0, 1])

tetra = Mesh()
# setting the vertices as the vertices of the new mesh object
tetra.vertices = [v1, v2, v3, v4]
# now we compute the subsets of all the vertices consiting of three vertices
subsets = set(itertools.combinations(list(range(tetra.v)), 3))
# the faces are then set as the faces of tetra
tetra.faces = [Face(tetra, *list(i)) for i in subsets]
```

This way, we obtain the `Mesh` object `tetra` which represents a tetrahedron.

Definition 2 (Undirected Graph)**.** *Let $\mathcal{V}_G$ be a set and $\mathcal{E}_G \subset \{e \in 2^{\mathcal{V}_G} : |e| = 2\}$ be a set of unordered pairs of elements from $\mathcal{V}_G$. The pair $G = (\mathcal{V}_G, \mathcal{E}_G)$ is then called undirected Graph. The elements from $\mathcal{V}_G$ are called vertices of G and the elements from $\mathcal{E}_G$ are called edges of G.*

For a Graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ we write

$$x \text{ --- } y$$

if $\{x, y\} \in \mathcal{E}_G$. If we take all edges and points together in this way, we get the picture of a graph with undirected edges.

Example 2.

$$G\colon \left(\begin{tikzcd} & 5 \arrow[dr, dash] & \\ 2 \arrow[ur, dash] \arrow[rr, dash] \arrow[d, dash] \arrow[drr, dash] & & 4 \arrow[d, dash] \\ 1 \arrow[rr, dash] \arrow[urr, dash] & & 3 \end{tikzcd}\right), \qquad H\colon \left(\begin{tikzcd} & 2 \arrow[dr, dash] & \\ 1 \arrow[ur, dash, bend left] \arrow[ur, dash] \arrow[rr, dash] \arrow[dr, dash] \arrow[dr, dash, bend right] & & 3 \\ & 4 \arrow[ur, dash] & \end{tikzcd}\right) \tag{1}$$

Definition 3 (Face Graph)**.** *Let $M = (\mathcal{V}_M, \mathcal{F}_M)$ be a mesh and*

$$\mathcal{E} = \left\{(f_1, f_2) \in \mathcal{F}_G^2 : |f_1 \cap f_2| = 2\right\}$$

The face graph of M is the graph $(\mathcal{F}_M, \mathcal{E})$.

Example 3. *The face graph of the tetrahedron is given by*

$$G\colon \left(\begin{tikzcd} 3 \arrow[rr, dash, bend left] \arrow[dr, dash] \arrow[ddr, dash, bend right] & & 2 \arrow[dl, dash] \arrow[ddl, dash, bend left] \\ & 1 \arrow[d, dash] & \\ & 4 & \end{tikzcd}\right) \tag{2}$$

The face graph of a given mesh can be constructed by algorithm **??**. Since this algorithm loops over the faces of the mesh in a nested way, the complexity of it lies in $O(F_M^2)$. As this runtime complexity has the consequence of the algorithm being very slow at execution for somewhat large meshes, the face graph is computed dynamically by `maniflow.mesh.Mesh.faceGraph`.

Algorithm 1: Construction of the face graph of a given mesh

Input : A mesh $M = (\mathcal{V}_M, \mathcal{F}_M = \{f_1, f_2, f_3 \ldots\})$
Output: The adjacency matrix of the face graph of the mesh M

$G := 0 \in \mathbb{R}^{F_M \times F_M}$;
for $i = 1$ **to** F_M **do**
    $neighbors := 0$;
    for $j = 1$ **to** F_M **do**
        if $neighbors = 3$ **then**
            break;
        end
        if $|f_i \cap f_j| = 2$ **and** $i \neq j$ **then**
            $G_{ij} \leftarrow 1$;
            $neighbors \leftarrow neighbors + 1$;
        end
    end
end
return G

1.1 A first application: `maniflow.mesh.utils.connectedComponents`

The method `maniflow.mesh.utils.connectedComponents` decomposes the given mesh into its connected components. Now that we have an algorithm with which to compute the face graph, the connected components of a mesh can now be identified as the connected components of the face graph. These can be determined via the breadth-first traversal of the face graph.

Algorithm 2: Construction of the face graph of a given mesh

Input : A mesh $M = (\mathcal{V}_M, \mathcal{F}_M = \{f_1, f_2, f_3 \ldots\})$
Output: The connected components of the mesh M

Compute the adjacency matrix G using **??**;
$start := 1$;
$n := 1$;
while $\mathcal{F}_M \neq \emptyset$ **do**
    Compute a breadth first traversal sequence $T_n \leftarrow \{f_{start}, f_b, f_c, \ldots\} \subseteq \mathcal{F}_M$;
    $n \leftarrow n + 1$;
    $\mathcal{F}_M \leftarrow \mathcal{F}_M \setminus T_n$;
    Set $1 < start \leq F_M$ such that $f_{start} \in \mathcal{F}_M$;
end
return T_1, T_2, $\ldots$

Runtime analysis. The algorithm **??** has a runtime complexity which lies in $O(F_M^2)$. The breadth-first traversal on the face graph has a runtime[1] complexity of $O(F_G + 3 \cdot F_G) = O(F_G)$. The computation of

[1] Since on a graph with the number of vertices being V and the number of edges being E the breadth first search has a complexity of $O(E + V)$. As every face has at most three neighbors we obtain the given runtime complexity.

$\mathcal{F}_M \setminus T_n$ has also quadratic complexity $O(|\mathcal{F}_M|^2)$. Thus the overall complexity of algorithm **??** lies in $O(F_G^2)$.

Example 4. *In this example we analyse the connected components of the teapot from* `examples/teapot.obj`*. The teapot is displayed in figure 2.*

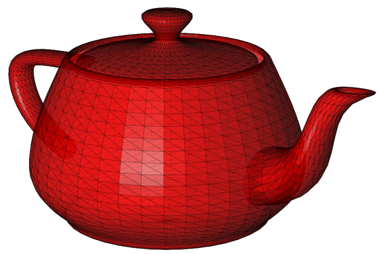
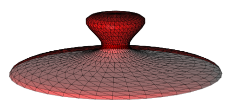


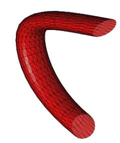
Figure 2: The teapot from `examples/teapot.obj`

The connected components can be computed using the following code

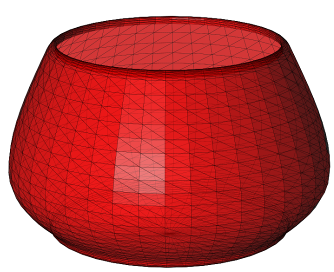
```
from maniflow.mesh import Mesh
from maniflow.mesh.obj import OBJFile
from maniflow.mesh.utils import connectedComponents, coincidingVertices

teapot = OBJFile.read("examples/teapot.obj")
coincidingVertices(teapot)
components = connectedComponents(teapot)

for i, component_list in enumerate(components):
    component = Mesh.fromFaceList(teapot, *component_list)
    OBJFile.write(component, "teapot" + str(i + 1) + ".obj")
```



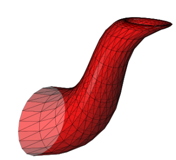
(a) The lid of the teapot



(b) The handle of the teapot



(c) The body of the teapot



(d) The spout of the teapot

Figure 3: The connected components of the teapot