Exercise 6: Mesh smoothing

In this exercise, we will smooth meshes using the Laplacian. You can use the implementation of the combinatorial (1 point) and cotan (2 points) Laplacian vectors from the last time. If you did not completed implementation of Laplacian vectors, you can download it from this website: https://home.zcu.cz/~hachaf/ZPOS/

We will visualize a series of smoothing steps in an animation. In order to do that, we have to create an array of TriangleMesh structures, which we will then pass to the renderer. For each timestep of the animation, you should do the following:

- create a deep copy of the mesh from the previous step by calling the `DeepCopy()` function
- compute the coordinate Laplacian, i.e. a vector for each vertex, by using the implementation from the last exercise
- change the vertex positions in the copy by simply updating the `points` field. The vertices should move in the direction of the coordinate Laplacian. Use a scaling parameter lambda to influence the smoothing strength

Keep in mind that if you have a particular data structure that is used by the implementation of the Laplacian, then you should also copy it in the `DeepCopy()` function.

You can navigate through the animation in the renderer by pressing the "," and "." keys. Finally, you can play the animation in the renderer by pressing the M key.

If you have both versions of the Laplacian implemented, you can check the correctness by checking that the uniform Laplacian smooths the surface and pushes the vertices towards uniform sampling, while the cotan Laplacian preserves the sampling, i.e. it preserves the tangential position of vertices.

Implicit smoothing (2 points)

While explicit smoothing is trivial to implement, it often leads to problems when choosing the lambda parameter. Therefore, implement implicit smoothing using provided `Meta.Numerics` library. It allows constructing a sparse matrix using the `SparseSquareMatrix` class. You may set the components of the matrix using the default two-integer indexer. Finally, you should construct appropriate righthandsides using the `ColumnVector` class. The system can then be solved by calling the `Solve` method of the matrix instance, passing the righthandside as parameter.

Test both discretizations of the Laplace operator, and compare the results with explicit smoothing.