Geometric Data Processing (IN4255)
TU Delft, EEMCS, Department of Intelligent Systems
K. Hildebrandt

2022-2023 Q. 4
Due: 14-06-2023

Practical Assignment 2

(Differential Coordinates and Shape Editing)

Task 1 (Differential Coordinates).

Implement methods that compute the sparse matrices G, which maps a function that is continuous and a linear polynomial over all triangles of a mesh to its gradient vectors, the combinatorial Laplace matrix L, which can be used to compute the combinatorial Laplace coordinates, and the cotangent matrix S and the mass matrix M, which can be used to compute the geometric Laplace coordinates.
Hints:

- As a first step to compute G, compute the 3×3 matrix that maps a linear polynomial over a triangle to its gradient vector. Then use this method to compute the matrix G for a triangle mesh.
- You can use G to compute S.
- Design tests to check that the matrices are correctly computed. Include a summary of the tests to the report.

Task 2 (Shape editing using differential coordinates).

Implement tools for shape editing using differential coordinates. The minimum requirement is to implement (a simplified version of) the brushes tool for editing triangle meshes we discussed in the lecture. It should allow to specify a 3×3 matrix A, which is applied to the gradient vectors of all selected triangles of the mesh. Then, the vertex positions of the mesh are modified such that the gradient vectors of the new mesh are as-close-as-possible (in the least-squares sense) to the modified gradients.
Remark: When constructing a mesh that best matches given gradients, the vertex positions are only determined up to translations of the whole mesh in $\mathbb{R}^3$. You can deal with this by keeping the barycenter of the mesh constant.

Ideas for further shape editing tools:

- A brushes tool using Laplace Coordinates
- Shape editing with constraints

Task 3 (User Manual and Summary of Experiences).

Write a report that describes and illustrates

- the algorithms and functionality implemented
- how the implementation can be used

- your tests for correctness of the implemented algorithms
- your evaluation (parameter settings, comparing alternative, discussion of benefits and limitations) of the implemented methods

In addition, you can report on the division of labor amongst the group members.

Required deliverables on Brightspace.

- For Tasks 1 and 2, provide the source files and example meshes. Pack all the files in one ZIP archive.
- The report should be one PDF file

Deadline: June 14, 20:00.

Hints.
Concerning the construction of the sparse matrix G:

- Use the `class jvx.numeric.PnSparseMatrix`
- Constructor: `PnSparseMatrix(`$n, m, 3$`)`, where n equals 3∗the number of faces of the mesh and n equals the number of vertices of the mesh. The last argument reserves space for storing 3 entries per row. Alternatively, you could use `PnSparseMatrix(`n, m`)`. Then the space for the entries will be allocated when you set them.
- You can use the method `addEntry(int` k`, int` l`, double` $value$`)` for constructing the matrix. The method adds $value$ at position k, l in the matrix. If the matrix entry with k, l does not exist, the space for storing it is created.

For multiplication of sparse matrices you can use:
$AB =$`PnSparsematrix.multMatrices`$(A, B, null)$;

For multiplication of a sparse matrix and a vector you can use:
$Av =$`PnSparsematrix.rightMultVector`$(A, v, null)$;. This method generates an new PdVector that is the product of the matrix and the vector. If a PdVector w for storing the result is already allocated, use
`PnSparsematrix.rightMultVector`(A, v, w);. The method then additionally returns a reference to w.

For solving the sparse linear systems (Task 2), you can use
`dev6.numeric.PnMumpsSolver`. This class offers an interface to the direct solvers of the MUMPS library. To solve the system $A\,x = b$, you can use the method `solve(A, x, b, PnMumpsSolver.Type.GENERAL_SYMMETRIC)`.
For solving a number of systems with the same matrix, compute the factorization once using:

- `public static long factor(PnSparseMatrix matrix, Type sym)`

and use

- `public static native void solve(long factorization, PdVector x, PdVector b)`

for solving the systems.

The MUMPS library should work on WINDOWS 64-bit systems. If the MUMPS library does not work on your system, you can use `jvx.numeric.PnConjugateGradient` and `jvx.numeric.PnBiconjugateGradient` instead. However, this is less efficient (do not use too large meshes in this case).