

```

1  // -*- C++ -*-
2  #ifndef ELEMENT_TYPES
3  #define ELEMENT_TYPES
4
5  #include "/src/Definitions.h"
6
7  namespace ElementTypes {
8
9  // simplicial element: linear triangle (if Dimension=2) or linear tetrahedron (if
10 // Dimension=3)
11 template<unsigned int Dimension>
12 struct Simplex {
13     static const unsigned int NumberOfNodes = Dimension + 1;
14     static const unsigned int SpatialDimension = Dimension;
15     static const unsigned int NumberOfNodesPerBoundary = Dimension;
16     static const unsigned int NumberOfBoundaries = Dimension + 1;
17     typedef Eigen::Matrix<double, SpatialDimension, 1> Point;
18     static const VTKCellType VtkCellType = (Dimension==1) ? VTK_LINE :
19                                             (Dimension == 2) ? VTK_TRIANGLE : VTK_TETRA;
20
21     //TODO: compute shape functions in the reference configuration of the element
22     // (using reference coordinates)
23     array<double, NumberOfNodes>
24     computeShapeFunctions(const Point &point) const {
25
26         ignoreUnusedVariable<Point>(point);
27
28         // TODO: Evaluate all entries of shapeFunctions, initialized just below
29         array<double, NumberOfNodes> shapeFunctions;
30         if (Dimension == 1){
31             shapeFunctions[0] = point(0);
32             shapeFunctions[1] = 1 - point(0);
33         }
34         else if (Dimension == 2){
35             shapeFunctions[0] = point(0);
36             shapeFunctions[1] = point(1);
37             shapeFunctions[2] = 1 - point(0) - point(1);
38         }
39         if (Dimension == 3){
40             shapeFunctions[0] = point(0);
41             shapeFunctions[1] = point(1);
42             shapeFunctions[2] = point(2);
43             shapeFunctions[3] = 1 - point(0) - point(1) - point(2);
44         }
45         // Return
46         return shapeFunctions;
47     }
48
49     //TODO: compute shape function derivatives in the reference configuration of the
50     // element (using reference coordinates)
51     array<Point, NumberOfNodes>
52     computeShapeFunctionDerivatives(const Point &point) const {
53
54         ignoreUnusedVariable<Point>(point);
55
56         // TODO: Evaluate all entries of shapeFunctionsDerivatives, initialized just below
57         // NOTE: Of course, since we're looking at the derivatives of shape functions now,
58         // we're not handling array<double, NumberOfNodes> but instead
59         // array<double, NumberOfNodes> as of now - just wanted to stress this ;- )
60         array<Point, NumberOfNodes> shapeFunctionDerivatives;
61         if (Dimension == 1){
62             shapeFunctionDerivatives[0](0) = 1;
63             shapeFunctionDerivatives[1](0) = 0;
64         }
65         if (Dimension == 2){
66             shapeFunctionDerivatives[0](0) = 1;
67             shapeFunctionDerivatives[1](0) = 0;
68             shapeFunctionDerivatives[2](0) = -1;
69
70             shapeFunctionDerivatives[0](1) = 0;
71             shapeFunctionDerivatives[1](1) = 1;
72             shapeFunctionDerivatives[2](1) = -1;

```

```
71     }
72     if (Dimension == 3){
73         shapeFunctionDerivatives[0](0) = 1;
74         shapeFunctionDerivatives[1](0) = 0;
75         shapeFunctionDerivatives[2](0) = 0;
76         shapeFunctionDerivatives[3](0) = -1;
77
78         shapeFunctionDerivatives[0](1) = 0;
79         shapeFunctionDerivatives[1](1) = 1;
80         shapeFunctionDerivatives[2](1) = 0;
81         shapeFunctionDerivatives[3](1) = -1;
82
83         shapeFunctionDerivatives[0](2) = 0;
84         shapeFunctionDerivatives[1](2) = 0;
85         shapeFunctionDerivatives[2](2) = 1;
86         shapeFunctionDerivatives[3](2) = -1;
87     }
88
89     // Return
90     return shapeFunctionDerivatives;
91 }
92 };
93
94 }
95 #endif //ELEMENT_TYPES
96
```