

```

1 #ifndef TRIANGLEREFINER_HPP
2 #define TRIANGLEREFINER_HPP
3
4 #include "GenericDomain.hpp"
5 #include "GenericMesh.hpp"
6 #include "Eigen/Eigen"
7
8 using namespace std;
9 using namespace Eigen;
10
11 namespace GeDiM
12 {
13 class TriangleRefiner
14 {
15 private:
16     /*
17     Seguono dichiarazione e definizione di funzioni semplici o di supporto
18     */
19
20     // Ritorna la distanza tra due punti
21     static double SqrDistance(const GenericPoint *P1, const GenericPoint *P2)
22     {
23         return (P1->Coordinates() - P2->Coordinates()).squaredNorm();
24     }
25
26     // Ritorna il lato più lungo di una cella
27     const GenericEdge *LongestEdge(const GenericCell *C)
28     {
29         double a = SqrDistance(C->Edge(0)->Point(0), C->Edge(0)->Point(1));
30         double b = SqrDistance(C->Edge(1)->Point(0), C->Edge(1)->Point(1));
31         double c = SqrDistance(C->Edge(2)->Point(0), C->Edge(2)->Point(1));
32         if ((a >= b) and (a >= c))
33             return C->Edge(0);
34         else if ((b >= a) and (b >= c))
35             return C->Edge(1);
36         else
37             return C->Edge(2);
38     }
39
40     // Ritorna true se il triangolo ha almeno un lato marcato
41     bool HasMarkedEdges(const GenericCell *C)
42     {
43         return (idEdgesToCut.at(C->Edge(0)->Id())) or
44 (idEdgesToCut.at(C->Edge(1)->Id())) or (idEdgesToCut.at(C->Edge(2)->Id()));
45     }
46
47     // Ritorna true se il lato è sul bordo ovvero confina con una cella NULL
48     bool IsOnBorder(const GenericEdge *E)
49     {
50         return (E->Cell(0) == NULL) or (E->Cell(1) == NULL);
51     }
52 }

```

```

52 // Creano in modo corretto e coerente vari tipi di oggetti geometrici
53 GenericPoint *NewPoint()
54 {
55     GenericPoint *P = meshPointer->CreatePoint();
56     meshPointer->AddPoint(P);
57     return P;
58 }
59 GenericEdge *NewEdge()
60 {
61     GenericEdge *E = meshPointer->CreateEdge();
62     meshPointer->AddEdge(E);
63     idEdgesToCut.push_back(false); //Creando un lato è necessario
    estendere idEdgesToCut perchè lo comprenda
64     return E;
65 }
66 GenericCell *NewCell()
67 {
68     GenericCell *C = meshPointer->CreateCell();
69     meshPointer->AddCell(C);
70     return C;
71 }
72
73 // Riempie un lato vuoto con informazioni geometriche di punti e celle
74 void SetEdgeGeometry(GenericEdge *E, const GenericPoint *P0, const
GenericPoint *P1, const GenericCell *right, const GenericCell *left)
75 {
76     E->AddPoint(P0);
77     E->AddPoint(P1);
78     E->InitializeCells(2);
79     E->AddCell(right); // RightCell = cell[0]
80     E->AddCell(left); // LeftCell = cell[1]
81 }
82
83 // Completa le informazioni di vicinanza solo nel caso in cui si stia
raffinato TUTTA la mesh
84 void SetEdgeGeometry_QuattroLati(GenericEdge *E, const GenericPoint *P0,
const GenericPoint *P1, const GenericCell *cell)
85 {
86     E->AddPoint(P0);
87     E->AddPoint(P1);
88     E->InitializeCells(2);
89     E->AddCell(cell);
90 }
91
92 // Inizializza le informazioni di parentela fra un padre e due figli
93 void SetFamily(GenericTreeNode *father, GenericTreeNode *C1, GenericTreeNode
*C2)
94 {
95     father->InitializeChilds(2);
96     father->AddChild(C1);
97     father->AddChild(C2);
98     C1->SetFather(father);
99     C2->SetFather(father);

```

```

100     }
101
102     // Inizializzano le informazioni geometriche su punti e lati di una cella
vuota
103     void SetCellPoints(GenericCell *C, const GenericPoint *P0, const
GenericPoint *P1, const GenericPoint *P2)
104     {
105         C->InitializePoints(3);
106         C->AddPoint(P0);
107         C->AddPoint(P1);
108         C->AddPoint(P2);
109     }
110     void SetCellEdges(GenericCell *C, const GenericEdge *E0, const GenericEdge
*E1, const GenericEdge *E2)
111     {
112         C->InitializeEdges(3);
113         C->AddEdge(E0);
114         C->AddEdge(E1);
115         C->AddEdge(E2);
116     }
117
118     // Ritorna true se nel vettore idEdgesToCut c'è ancora almeno un true
119     bool anyTriangleToCut()
120     {
121         for (unsigned edgeId = 0; edgeId < meshPointer->NumberOfEdges();
edgeId++)
122             if (idEdgesToCut.at(edgeId))
123                 return true;
124
125         return false;
126     }
127
128     /*
129     Seguono dichiarazioni di funzioni algoritmiche importanti o
complesse
130     */
131     void RotateCell(GenericCell
*C); // Riordina i puntatori a
lati e punti del triangolo in modo coerente
132     void PensaciTuAllLatoIgnoto(GenericCell *C, GenericEdge *E); // Sistema le
condizioni di vicinanza tra un triangolo ed un lato "esterno"
133     void RefinePairedTriangles(GenericCell *C0, GenericCell *C1); // Taglia
contemporaneamente due triangoli che condividono il lato lungo
134     void RefineBorderTriangle(GenericCell
*C0); // Taglia un triangolo che ha il lato
lungo sul bordo della mesh
135
136     /*
137     Membri della classe TriangleRefiner
138     */
139
140     GenericMesh *meshPointer; // Puntatore alla mesh
141     vector<bool> idEdgesToCut; // Vettore di variabili booleane. Se l'elemento
n-esimo del vettore è true allora il lato con id = n è da tagliare

```

```

142
143 public:
144     // Nel costruttore richiedo una mesh e preparo il vettore di bool grande
abbastanza
145     TriangleRefiner(GenericMesh &mesh)
146     {
147         meshPointer = &mesh;
148         idEdgesToCut.assign(meshPointer->NumberOfEdges(), false);
149     }
150     ~TriangleRefiner()
151     {
152         meshPointer = NULL;
153         idEdgesToCut.clear();
154     }
155
156     // Funzione che ruota la cella e marca da tagliare il lato più lungo
157     void PrepareTriangle(const unsigned int &value);
158
159     // Funzione che ricalcola tutte le informazioni di vicinanza dei punti
160     void AggiornaInformazioniPunti();
161
162     // Raffina la mesh in maniera conforme
163     Output::ExitCodes RefineMesh();
164
165     // Raffina tutta la mesh
166     Output::ExitCodes TaglioInQuattro();
167 };
168 } // namespace GeDiM
169
170 #endif

```