

```

1 #include "TriangleRefiner.hpp"
2
3 namespace GeDiM
4 {
5 Output::ExitCodes TriangleRefiner::RefineMesh()
6 {
7     unsigned edgeId = 0;
8     while (anyTriangleToCut()) // Il ciclo procede fino a quando ci sono lati da
tagliare
9     {
10         edgeId = (edgeId + 1) % meshPointer->NumberOfEdges(); // Il modulo è
necessario per poter scorrere il vettore in modo circolare
11         if (not
idEdgesToCut.at(edgeId)) // Se il lato non
è da tagliare lo ignoro e passo oltre
12             continue;
13
14         GenericEdge *L = meshPointer->Edge(edgeId); // Creo un puntatore al
lato da tagliare L
15
16         if (IsOnBorder(L)) // Se è un lato di bordo della mesh
17         {
18             GenericCell *C0;
19             // Creo un puntatore all'unica cella adiacente C0
20             if (L->Cell(0) == NULL)
21                 C0 = meshPointer->Cell(L->Cell(1)->Id());
22             if (L->Cell(1) == NULL)
23                 C0 = meshPointer->Cell(L->Cell(0)->Id());
24             RefineBorderTriangle(C0); // Raffino il
triangolo nella condizione di bordo
25             idEdgesToCut.at(edgeId) = false; // Indico a idEdgesToCut
che il lato non è più da tagliare
26         }
27         else if (LongestEdge(L->Cell(0)) == LongestEdge(L->Cell(1))) // Se
il lato è il più lungo di entrambi i triangoli a cui appartiene
28         {
29             GenericCell *C0 = meshPointer->Cell(L->Cell(0)->Id()); //
Creo il puntatore al primo triangolo C0
30             GenericCell *C1 = meshPointer->Cell(L->Cell(1)->Id()); //
Creo il puntatore al secondo triangolo C1
31             RefinePairedTriangles(C0,
C1); // Raffino entrambi i triangoli
contemporaneamente
32             idEdgesToCut.at(edgeId) =
false; // Indico a idEdgesToCut che il lato non
è più da tagliare
33         }
34         else
35         {
36             // Se il lato non rientra nei due casi precedenti preparo
per il raffinamento i triangoli a cui appartiene
37             if (L->HasRightCell())
38                 PrepareTriangle(L->RightCell()->Id());
39             if (L->HasLeftCell())

```

```

40         PrepareTriangle(L->LeftCell()->Id());
41     }
42 }
43     return Output::Success;
44 }
45
46 void TriangleRefiner::RotateCell(GenericCell *C)
47 {
48     GenericEdge *L = meshPointer->Edge(LongestEdge(C)->Id()); // Creo un
// puntatore al lato piú lungo del triangolo
49     // Ruoto i lati al fine di avere L in posizione 0 e mantenendo il verso
// antiorario della rappresentazione
50     if (L == C->Edge(1))
51     {
52         C->InsertEdge(C->Edge(2), 1);
53         C->InsertEdge(C->Edge(0), 2);
54         C->InsertEdge(L, 0);
55     }
56     else if (L == C->Edge(2))
57     {
58         C->InsertEdge(C->Edge(1), 2);
59         C->InsertEdge(C->Edge(0), 1);
60         C->InsertEdge(L, 0);
61     }
62     // Ruoto i punti fino a quando quello in posizione 2 non appartiene piú al
// lato piú lungo L
63     while ((C->Point(2) == L->Point(0)) or (C->Point(2) == L->Point(1)))
64     {
65         const GenericPoint *tmp = C->Point(0); // Creo una variabile
// temporanea per effettuare gli scambi di posizione
66         C->InsertPoint(C->Point(2), 0);
67         C->InsertPoint(C->Point(1), 2);
68         C->InsertPoint(tmp, 1);
69     }
70 }
71
72 void TriangleRefiner::PrepareTriangle(const unsigned int &value)
73 {
74     GenericCell *C = meshPointer->Cell(value); // Creo un puntatore al triangolo
// da preparare C
75     const GenericEdge *L = LongestEdge(C); // Creo un puntatore al lato piú
// lungo L
76     idEdgesToCut.at(L->Id()) = true; // Indico a IdEdgesToCut
// che L deve essere tagliato
77     RotateCell(C); // Ruoto
// i lati e i punti per avere una rappresentazione univoca
78 }
79 void TriangleRefiner::PensaciTuAllatoIgnoto(GenericCell *C, GenericEdge *E)
80 {
81     // Aggiorno correttamente le informazioni di vicinanza di un lato e dei
// triangoli vicini dopo aver raffinato
82     if (E->Cell(0) == C->Father())
83     {
84         E->InsertCell(C, 0);

```

```

85         C->AddCell(E->Cell(1));
86     }
87     else
88     {
89         E->InsertCell(C, 1);
90         C->AddCell(E->Cell(0));
91     }
92 }
93 void TriangleRefiner::RefinePairedTriangles(GenericCell *C0, GenericCell *C1)
94 {
95     // Ruoto entrambi i triangoli di input
96     RotateCell(C0);
97     RotateCell(C1);
98     // Creo il puntatore al lato pi  lungo di entrambe
99     GenericEdge *longest = meshPointer->Edge(LongestEdge(C0)->Id());
100
101     // Creo gli oggetti necessari vuoti
102     GenericPoint *midpoint = NewPoint(); // Punto medio
103
104     GenericEdge *subEdge0 = NewEdge(); // Figli del lato pi  lungo
105     GenericEdge *subEdge1 = NewEdge();
106
107     GenericEdge *median0 = NewEdge(); // Mediane
108     GenericEdge *median1 = NewEdge();
109
110     GenericCell *C0_0 = NewCell(); // Figli dei due triangoli
111     GenericCell *C0_1 = NewCell();
112     GenericCell *C1_0 = NewCell();
113     GenericCell *C1_1 = NewCell();
114
115     SetFamily(longest, subEdge0, subEdge1); // Imposto le informazioni di
116     parentela fra lati
117     SetFamily(C0, C0_0, C0_1); // Imposto le
118     informazioni di parentela fra triangoli
119     SetFamily(C1, C1_0, C1_1);
120
121     // Imposto le coordinate del punto medio
122     midpoint->SetCoordinates(0.5 * (longest->Point(0)->Coordinates() +
123     longest->Point(1)->Coordinates()));
124
125     // Costruisco i lati indicando le celle con cui confinano e i punti che vi
126     appartengono
127     SetEdgeGeometry(subEdge0, C0->Point(0), midpoint, C1_0, C0_0);
128
129     SetEdgeGeometry(subEdge1, midpoint, C0->Point(1), C1_1, C0_1);
130
131     SetEdgeGeometry(median0, C0->Point(2), midpoint, C0_0, C0_1);
132
133     SetEdgeGeometry(median1, midpoint, C1->Point(2), C1_0, C1_1);
134
135     // Costruisco il primo sottotriangolo
136     SetCellPoints(C0_0, C0->Point(0), midpoint, C0->Point(2));
137     SetCellEdges(C0_0, subEdge0, median0, C0->Edge(2));

```

```

134     C0_0->AddCell(C1_0);
135     C0_0->AddCell(C0_1);
136     PensaciTuAlLatoIgnoto(C0_0, meshPointer->Edge(C0->Edge(2)->Id()));
137
138     // Costruisco il secondo sottotriangolo
139     SetCellPoints(C0_1, midpoint, C0->Point(1), C0->Point(2));
140     SetCellEdges(C0_1, median0, subEdge1, C0->Edge(1));
141     C0_1->AddCell(C0_0);
142     C0_1->AddCell(C1_1);
143     PensaciTuAlLatoIgnoto(C0_1, meshPointer->Edge(C0->Edge(1)->Id()));
144
145     // Costruisco il terzo sottotriangolo
146     SetCellPoints(C1_0, C0->Point(0), C1->Point(2), midpoint);
147     SetCellEdges(C1_0, median1, subEdge0, C1->Edge(1));
148     C1_0->AddCell(C1_1);
149     C1_0->AddCell(C0_0);
150     PensaciTuAlLatoIgnoto(C1_0, meshPointer->Edge(C1->Edge(1)->Id()));
151
152     // Costruisco il quarto sottotriangolo
153     SetCellPoints(C1_1, midpoint, C1->Point(2), C0->Point(1));
154     SetCellEdges(C1_1, subEdge1, median1, C1->Edge(2));
155     C1_1->AddCell(C0_1);
156     C1_1->AddCell(C1_0);
157     PensaciTuAlLatoIgnoto(C1_1, meshPointer->Edge(C1->Edge(2)->Id()));
158
159     // Disattivo i triangoli e il lato padri
160     C0->SetState(false);
161     C1->SetState(false);
162     longest->SetState(false);
163
164     //Eredita marker
165     midpoint->SetMarker(longest->Marker());
166
167     // Se i sottotriangoli creati hanno lati marcati li preparo per il
raffinamento
168     if (HasMarkedEdges(C0_0))
169         PrepareTriangle(C0_0->Id());
170     if (HasMarkedEdges(C0_1))
171         PrepareTriangle(C0_1->Id());
172     if (HasMarkedEdges(C1_0))
173         PrepareTriangle(C1_0->Id());
174     if (HasMarkedEdges(C1_1))
175         PrepareTriangle(C1_1->Id());
176 }
177 void TriangleRefiner::RefineBorderTriangle(GenericCell *C0)
178 {
179     RotateCell(C0); // Ruoto la cella
180
181     GenericEdge *longest = meshPointer->Edge(LongestEdge(C0)->Id()); // Creo il
puntatore al lato piú lungo
182
183     // Creo gli oggetti necessari vuoti

```

```

184     GenericPoint *midpoint = NewPoint(); // Punto medio
185
186     GenericEdge *subEdge0 = NewEdge(); // Figli del lato pi  lungo
187     GenericEdge *subEdge1 = NewEdge();
188
189     GenericEdge *median0 = NewEdge(); // Mediana
190
191     GenericCell *C0_0 = NewCell(); // Figli del triangolo
192     GenericCell *C0_1 = NewCell();
193
194     SetFamily(longest, subEdge0, subEdge1); // Imposto le informazioni di
parentela fra lati
195     SetFamily(C0, C0_0, C0_1); // Imposto le
informazioni di parentela fra triangoli
196
197     // Imposto le coordinate del punto medio
198     midpoint->SetCoordinates(0.5 * (longest->Point(0)->Coordinates() +
longest->Point(1)->Coordinates()));
199
200     // Costruisco i lati indicando i triangoli con cui confinano e i punti che
vi appartengono
201     SetEdgeGeometry(subEdge0, C0->Point(0), midpoint, NULL, C0_0);
202     SetEdgeGeometry(subEdge1, midpoint, C0->Point(1), NULL, C0_1);
203     SetEdgeGeometry(median0, C0->Point(2), midpoint, C0_0, C0_1);
204
205     // Costruisco il primo sottotriangolo
206     SetCellPoints(C0_0, C0->Point(0), midpoint, C0->Point(2));
207     SetCellEdges(C0_0, subEdge0, median0, C0->Edge(2));
208     C0_0->AddCell(C0_1);
209     PensaciTuAlLatoIgnoto(C0_0, meshPointer->Edge(C0->Edge(2)->Id()));
210     C0_0->AddCell(NULL); // Il triangolo   di bordo
211
212     // Costruisco il secondo sottotriangolo
213     SetCellPoints(C0_1, midpoint, C0->Point(1), C0->Point(2));
214     SetCellEdges(C0_1, median0, subEdge1, C0->Edge(1));
215     PensaciTuAlLatoIgnoto(C0_1, meshPointer->Edge(C0->Edge(1)->Id()));
216     C0_1->AddCell(C0_0);
217     C0_1->AddCell(NULL); // Il triangolo   di bordo
218
219     // Disattivo il triangolo padre e il lato padre
220     C0->SetState(false);
221     longest->SetState(false);
222
223     //Eredita marker
224     midpoint->SetMarker(longest->Marker());
225
226     // Se i sottotriangoli creati hanno lati marcati li preparo per il
raffinamento
227     if (HasMarkedEdges(C0_0))
228         PrepareTriangle(C0_0->Id());
229     if (HasMarkedEdges(C0_1))
230         PrepareTriangle(C0_1->Id());
231 }

```

```

232
233 void TriangleRefiner::AggiornaInformazioniPunti()
234 {
235     for (unsigned int i = 0; i < meshPointer->NumberOfPoints(); i++)
236     {
237         GenericPoint *punto = meshPointer->Point(i);
238
239         //cancella le celle
240         punto->AllocateCells(0);
241
242         //cancella i lati
243         punto->AllocateEdges(0);
244
245         //Ciclo su tutte le celle. Se contengono il punto aggiornano le
informazioni dei punti
246         for (unsigned int j = 0; j < meshPointer->NumberOfCells(); j++)
247         {
248             const GenericCell *cella = meshPointer->Cell(j);
249             if (punto->Id() == cella->Point(0)->Id())
250                 punto->AddCell(cella);
251             if (punto->Id() == cella->Point(1)->Id())
252                 punto->AddCell(cella);
253             if (punto->Id() == cella->Point(2)->Id())
254                 punto->AddCell(cella);
255         }
256         //Ciclo su tutti i lati. Se contengono il punto aggiornano le
informazioni del punto
257         for (unsigned int j = 0; j < meshPointer->NumberOfEdges(); j++)
258         {
259             const GenericEdge *lato = meshPointer->Edge(j);
260             if (punto->Id() == lato->Point(0)->Id())
261                 punto->AddEdge(lato);
262             if (punto->Id() == lato->Point(1)->Id())
263                 punto->AddEdge(lato);
264         }
265     }
266 }
267
268 //SOLO NEL CASO SIANO TUTTI DA TAGLIARE
269 Output::ExitCodes TriangleRefiner::TaglioInQuattro()
270 {
271     unsigned int numCelleInizio = meshPointer->NumberOfCells();
272     for (unsigned int i = 0; i < numCelleInizio; i++)
273     {
274         GenericCell *C = meshPointer->Cell(i);
275
276         RotateCell(C); //Serve?
277
278         GenericPoint *Pm0 = NewPoint();
279         GenericPoint *Pm1 = NewPoint();
280         GenericPoint *Pm2 = NewPoint();
281

```

```

282         GenericEdge *subEdge0_0;
283         GenericEdge *subEdge0_1;
284         GenericEdge *subEdge1_0;
285         GenericEdge *subEdge1_1;
286         GenericEdge *subEdge2_0;
287         GenericEdge *subEdge2_1;
288
289         GenericEdge *median0 = NewEdge();
290         GenericEdge *median1 = NewEdge();
291         GenericEdge *median2 = NewEdge();
292
293         GenericCell *C0 = NewCell();
294         GenericCell *C1 = NewCell();
295         GenericCell *C2 = NewCell();
296         GenericCell *C3 = NewCell();
297
298         const GenericCell *CellaTmp0;
299         const GenericCell *CellaTmp1;
300
301         GenericEdge *E0 = meshPointer->Edge(C->Edge(0)->Id());
302         GenericEdge *E1 = meshPointer->Edge(C->Edge(1)->Id());
303         GenericEdge *E2 = meshPointer->Edge(C->Edge(2)->Id());
304
305         Pm0->SetCoordinates(0.5 * (C->Edge(0)->Point(0)->Coordinates() +
C->Edge(0)->Point(1)->Coordinates()));
306         Pm1->SetCoordinates(0.5 * (C->Edge(1)->Point(0)->Coordinates() +
C->Edge(1)->Point(1)->Coordinates()));
307         Pm2->SetCoordinates(0.5 * (C->Edge(2)->Point(0)->Coordinates() +
C->Edge(2)->Point(1)->Coordinates()));
308
309         if (!E0->HasChilds())
310         {
311             subEdge0_0 = NewEdge();
312             subEdge0_1 = NewEdge();
313             SetFamily(E0, subEdge0_0, subEdge0_1);
314             SetEdgeGeometry_QuattroLati(subEdge0_0, C->Point(0), Pm0,
C0);
315             SetEdgeGeometry_QuattroLati(subEdge0_1, Pm0, C->Point(1),
C1);
316         }
317         else
318         {
319             subEdge0_0 = meshPointer->Edge(E0->Child(1)->Id());
320             subEdge0_1 = meshPointer->Edge(E0->Child(0)->Id());
321             subEdge0_0->AddCell(C0);
322             subEdge0_1->AddCell(C1);
323
324             if (!IsOnBorder(E0))
325             {
326                 CellaTmp0 = subEdge0_0->Cell(0);
327                 CellaTmp1 = subEdge0_1->Cell(0);
328                 C0->AddCell(CellaTmp0);
329                 C1->AddCell(CellaTmp1);

```

```

330         }
331         else
332         {
333             C0->AddCell(NULL);
334             C1->AddCell(NULL);
335         }
336     }
337
338     if (!E1->HasChilds())
339     {
340         subEdge1_0 = NewEdge();
341         subEdge1_1 = NewEdge();
342         SetFamily(E1, subEdge1_0, subEdge1_1);
343         SetEdgeGeometry_QuattroLati(subEdge1_0, C->Point(1), Pm1,
C1);
344         SetEdgeGeometry_QuattroLati(subEdge1_1, Pm1, C->Point(2),
C2);
345     }
346     else
347     {
348         subEdge1_0 = meshPointer->Edge(E1->Child(0)->Id());
349         subEdge1_1 = meshPointer->Edge(E1->Child(1)->Id());
350         subEdge1_0->AddCell(C1);
351         subEdge1_1->AddCell(C2);
352
353         if (!IsOnBorder(E1))
354         {
355             CellaTmp0 = subEdge1_0->Cell(0);
356             CellaTmp1 = subEdge1_1->Cell(0);
357             C1->AddCell(CellaTmp0);
358             C2->AddCell(CellaTmp1);
359         }
360         else
361         {
362             C1->AddCell(NULL);
363             C2->AddCell(NULL);
364         }
365     }
366
367     if (!E2->HasChilds())
368     {
369         subEdge2_0 = NewEdge();
370         subEdge2_1 = NewEdge();
371         SetFamily(E2, subEdge2_0, subEdge2_1);
372         SetEdgeGeometry_QuattroLati(subEdge2_0, C->Point(2), Pm2,
C2);
373         SetEdgeGeometry_QuattroLati(subEdge2_1, Pm2, C->Point(0),
C0);
374     }
375     else
376     {
377         subEdge2_0 = meshPointer->Edge(E2->Child(1)->Id());
378         subEdge2_1 = meshPointer->Edge(E2->Child(0)->Id());

```



```

379         subEdge2_0->AddCell(C2);
380         subEdge2_1->AddCell(C0);
381
382         if (!IsOnBorder(E2))
383         {
384             CellaTmp0 = subEdge2_0->Cell(0);
385             CellaTmp1 = subEdge2_1->Cell(0);
386             C2->AddCell(CellaTmp0);
387             C0->AddCell(CellaTmp1);
388         }
389         else
390         {
391             C2->AddCell(NULL);
392             C0->AddCell(NULL);
393         }
394     }
395
396     SetFamily(C, C0, C1);
397     SetFamily(C, C2, C3);
398
399     SetEdgeGeometry(median0, Pm0, Pm1, C1, C3);
400     SetEdgeGeometry(median1, Pm1, Pm2, C2, C3);
401     SetEdgeGeometry(median2, Pm2, Pm0, C0, C3);
402
403     SetCellPoints(C3, Pm0, Pm1, Pm2);
404     SetCellEdges(C3, median0, median1, median2);
405     C3->AddCell(C0);
406     C3->AddCell(C1);
407     C3->AddCell(C2);
408
409     SetCellPoints(C0, C->Point(0), Pm0, Pm2);
410     SetCellEdges(C0, subEdge0_0, median2, subEdge2_1);
411     C0->AddCell(C3);
412
413     SetCellPoints(C1, Pm0, C->Point(1), Pm1);
414     SetCellEdges(C1, subEdge0_1, subEdge1_0, median0);
415     C1->AddCell(C3);
416
417     SetCellPoints(C2, Pm1, C->Point(2), Pm2);
418     SetCellEdges(C2, subEdge1_1, subEdge2_0, median1);
419     C2->AddCell(C3);
420 }
421
422 return Output::Success;
423 }
424
425 } // namespace GeDiM

```