```cpp
#include "TriangleRefiner.hpp"

namespace GeDiM
{
Output::ExitCodes TriangleRefiner::RefineMesh()
{
	unsigned edgeId = 0;
	while (anyTriangleToCut()) // Il ciclo procede fino a quando ci sono lati da tagliare
	{
		edgeId = (edgeId + 1) % meshPointer->NumberOfEdges(); // Il modulo è necessario per poter scorrere il vettore in modo circolare
		if (not idEdgesToCut.at(edgeId))                                        // Se il lato non è da tagliare lo ignoro e passo oltre
			continue;

		GenericEdge *L = meshPointer->Edge(edgeId); // Creo un puntatore al lato da tagliare L

		if (IsOnBorder(L)) // Se è un lato di bordo della mesh
		{
			GenericCell *C0;
			// Creo un puntatore all'unica cella adiacente C0
			if (L->Cell(0) == NULL)
				C0 = meshPointer->Cell(L->Cell(1)->Id());
			if (L->Cell(1) == NULL)
				C0 = meshPointer->Cell(L->Cell(0)->Id());
			RefineBorderTriangle(C0);                // Raffino il triangolo nella condizione di bordo
			idEdgesToCut.at(edgeId) = false; // Indico a idEdgesToCut che il lato non è più da tagliare
		}
		else if (LongestEdge(L->Cell(0)) == LongestEdge(L->Cell(1))) // Se il lato è il più lungo di entrambi i triangoli a cui appartiene
		{
			GenericCell *C0 = meshPointer->Cell(L->Cell(0)->Id()); // Creo il puntatore al primo triangolo C0
			GenericCell *C1 = meshPointer->Cell(L->Cell(1)->Id()); // Creo il puntatore al secondo triangolo C1
			RefinePairedTriangles(C0, C1);                                          // Raffino entrambi i triangoli contemporaneamente
			idEdgesToCut.at(edgeId) = false;                                 // Indico a idEdgesToCut che il lato non è più da tagliare
		}
		else
		{
			// Se il lato non rientra nei due casi precedenti preparo per il raffinamento i triangoli a cui appartiene
			if (L->HasRightCell())
				PrepareTriangle(L->RightCell()->Id());
			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 pi 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 pi al
    lato pi 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 pi
    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 vicni 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
    parentela fra lati
116         SetFamily(C0, C0_0, C0_1);                                  // Imposto le
    informazioni di parentela fra triangoli
117         SetFamily(C1, C1_0, C1_1);
118
119         // Imposto le coordinate del punto medio
120         midpoint->SetCoordinates(0.5 * (longest->Point(0)->Coordinates() +
    longest->Point(1)->Coordinates()));
121
122         // Costruisco i lati indicando le celle con cui confinano e i punti che vi
    appartengono
123         SetEdgeGeometry(subEdge0, C0->Point(0), midpoint, C1_0, C0_0);
124
125         SetEdgeGeometry(subEdge1, midpoint, C0->Point(1), C1_1, C0_1);
126
127         SetEdgeGeometry(median0, C0->Point(2), midpoint, C0_0, C0_1);
128
129         SetEdgeGeometry(median1, midpoint, C1->Point(2), C1_0, C1_1);
130
131         // Costruisco il primo sottotriangolo
132         SetCellPoints(C0_0, C0->Point(0), midpoint, C0->Point(2));
133         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 pi@ 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 aggiorno 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 aggiorno 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 } // namespace GeDiM
269
```