```
  1: //---------------------------------------------------------------------------
  2: //
  3: //  Copyright (C) 2001 Gabriel Taubin
  4: //
  5: //---------------------------------------------------------------------------
  6:
  7: import java.awt.*;
  8: import java.awt.event.*;
  9: import java.awt.image.*;
 10:
 11: public class Photo3DPanelMeshing
 12:   extends    Photo3DPanel
 13:   implements ActionListener
 14: {
 15:   private Photo3DDesktop _p3dd = null;
 16:
 17:   private float _actualEmin =  -1.0f;
 18:   private float _actualEmax =  -1.0f;
 19:   private TextField tf_ACTUAL_EMIN = null;
 20:   private TextField tf_ACTUAL_EMAX = null;
 21:
 22:   private float _targetEmin =  -1.0f;
 23:   private float _targetEmax =  -1.0f;
 24:   private TextField tf_TARGET_EMIN = null;
 25:   private TextField tf_TARGET_EMAX = null;
 26:
 27:   private Button b_PAINT = null;
 28:
 29:   private Button b_COLLAPSE = null;
 30:   private Button b_REFINE   = null;
 31:   private Button b_FLIP     = null;
 32:
 33:   private int   _smooth_N =  10;
 34:   private float _smooth_A =  0.50f;
 35:   private float _smooth_B = -0.51f;
 36:   private TextField tf_SMOOTH_N   = null;
 37:   private TextField tf_SMOOTH_A   = null;
 38:   private TextField tf_SMOOTH_B   = null;
 39:   private Button    b_SMOOTH      = null;
 40:
 41:   public Photo3DPanelMeshing(int x0, int y0, int width, int height,
 42:                              int rowHeight, int columnSpace, int rowSpace,
 43:                              Photo3DDesktop p3dd)
 44:   {
 45:     _p3dd = p3dd;
 46:
 47:     setLayout(null);
 48:     setSize(width,height);
 49:     setLocation(x0,y0);
 50:
 51:     int columnWidth = width;
 52:
 53:     int x = 0;
 54:     {
 55:       int y  = 0;
 56:       int rS = rowSpace;
 57:       int rH = rowHeight;
 58:       int cW = columnWidth;
 59:       int x1 = x;
 60:       int lW = 4*rH;
 61:       int x2 = x1+lW+rS;
 62:       int tW = cW-(x2-x1);
 63:
 64:       Label l0 =
 65:         _newLabel("ACTUAL EDGE LENGTHS",g080,g240,
 66:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
 67:
 68:       y += rowHeight+rowSpace;
 69:       {
 70:         Label l00 =
 71:           _newLabel(" MIN",g167,g000,Label.LEFT,x1,y,lW,rH,this);
 72:         tf_ACTUAL_EMIN =
 73:           _newTextField("",x2,y,tW,rH,this,this);
 74:         tf_ACTUAL_EMIN.setEditable(false);
 75:       }
 76:
 77:       y += rowHeight+rowSpace;
 78:       {
 79:         Label l01 =
 80:           _newLabel(" MAX",g167,g000,Label.LEFT,x1,y,lW,rH,this);
 81:         tf_ACTUAL_EMAX =
 82:           _newTextField("",x2,y,tW,rH,this,this);
 83:         tf_ACTUAL_EMAX.setEditable(false);
 84:       }
 85:
 86:       y += rowHeight+rowSpace;
 87:       Label l1 =
 88:         _newLabel("TARGET EDGE LENGTHS",g080,g240,
 89:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
 90:
 91:       y += rowHeight+rowSpace;
 92:       {
 93:         Label l00 =
 94:           _newLabel(" MIN",g167,g000,Label.LEFT,x1,y,lW,rH,this);
 95:         tf_TARGET_EMIN =
 96:           _newTextField("",x2,y,tW,rH,this,this);
 97:       }
 98:
 99:       y += rowHeight+rowSpace;
100:       {
101:         Label l01 =
102:           _newLabel(" MAX",g167,g000,Label.LEFT,x1,y,lW,rH,this);
103:         tf_TARGET_EMAX =
104:           _newTextField("",x2,y,tW,rH,this,this);
105:       }
106:
107:       y += rowHeight+rowSpace;
108:       b_PAINT =
109:         _newButton("PAINT TARGET EDGES",x,y,columnWidth,rowHeight,this,this);
110:
111:       y += rowHeight+rowSpace;
112:       Label l2 =
113:         _newLabel("SIMPLIFICATION",g080,g240,
114:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
115:
116:       y += rowHeight+rowSpace;
117:       b_COLLAPSE =
118:         _newButton("COLLAPSE EDGES",x,y,columnWidth,rowHeight,this,this);
119:
120:       y += rowHeight+rowSpace;
121:       Label l3 =
122:         _newLabel("REFINEMENT",g080,g240,
123:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
124:
125:       y += rowHeight+rowSpace;
126:       b_REFINE =
127:         _newButton("SUBDIVIDE ADAPTIVELY",x,y,columnWidth,rowHeight,this,this);
128:
129:       y += rowHeight+rowSpace;
130:       Label l4 =
131:         _newLabel("OPTIMIZATION",g080,g240,
132:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
133:
134:       y += rowHeight+rowSpace;
135:       b_FLIP =
136:         _newButton("FLIP EDGES",x,y,columnWidth,rowHeight,this,this);
137:
138:       y += rowHeight+rowSpace;
139:       Label l5 =
140:         _newLabel("SMOOTHING",g080,g240,
141:                   Label.CENTER,x,y,columnWidth,rowHeight,this);
142:
143:       y += rowHeight+rowSpace;
144:       {
145:         Label l_SMOOTH_N =
146:           _newLabel("N",g167,g000,Label.LEFT,x1,y,lW,rH,this);
147:         tf_SMOOTH_N =
148:           _newTextField("",x2,y,tW,rH,this,this);
149:       }
150:
```

```
151:       y += rowHeight+rowSpace;
152:       {
153:         Label l_SMOOTH_A =
154:           _newLabel("A",g167,g000,Label.LEFT,x1,y,lW,rH,this);
155:         tf_SMOOTH_A =
156:           _newTextField("",x2,y,tW,rH,this,this);
157:       }
158:
159:       y += rowHeight+rowSpace;
160:       {
161:         Label l_SMOOTH_B =
162:           _newLabel("B",g167,g000,Label.LEFT,x1,y,lW,rH,this);
163:         tf_SMOOTH_B =
164:           _newTextField("",x2,y,tW,rH,this,this);
165:       }
166:
167:       y += rowHeight+rowSpace;
168:       b_SMOOTH =
169:         _newButton("SMOOTH",x,y,columnWidth,rowHeight,this,this);
170:
171:     }
172:     updateText();
173:   }
174:
175:   // implements ActionListener
176:   public void actionPerformed(ActionEvent e)
177:   {
178:     Object src = e.getSource();
179:     String cmd = e.getActionCommand();
180:
181:     if(src==b_PAINT) {
182:       paintTargetEdges();
183:     } if(src==b_COLLAPSE) {
184:       collapseEdges();
185:     } else if(src==b_REFINE) {
186:       refineEdges();
187:     } else if(src==b_FLIP) {
188:       flipEdges();
189:     } else if(src==b_SMOOTH) {
190:       smoothCoord();
191:     } else if(src==tf_SMOOTH_N) {
192:       String text = tf_SMOOTH_N.getText().trim();
193:       int sn = _smooth_N;
194:       try { _smooth_N = Integer.valueOf(text).intValue(); }
195:       catch(Exception ee) { _smooth_N = sn; }
196:     } else if(src==tf_SMOOTH_A) {
197:       String text = tf_SMOOTH_A.getText().trim();
198:       float sa = _smooth_A;
199:       try { _smooth_A = Float.valueOf(text).floatValue(); }
200:       catch(Exception ee) { _smooth_A = sa; }
201:     } else if(src==tf_SMOOTH_B) {
202:       String text = tf_SMOOTH_B.getText().trim();
203:       float sb = _smooth_B;
204:       try { _smooth_B = Float.valueOf(text).floatValue(); }
205:       catch(Exception ee) { _smooth_B = sb; }
206:     } else if(src==tf_TARGET_EMIN) {
207:       String text = tf_TARGET_EMIN.getText().trim();
208:       float f = _targetEmin;
209:       try { _targetEmin = Float.valueOf(text).floatValue(); }
210:       catch(Exception ee) { _targetEmin = f; }
211:     } else if(src==tf_TARGET_EMAX) {
212:       String text = tf_TARGET_EMAX.getText().trim();
213:       float f = _targetEmax;
214:       try { _targetEmax = Float.valueOf(text).floatValue(); }
215:       catch(Exception ee) { _targetEmax = f; }
216:     }
217:     updateText();
218:   }
219:
220:   private void updateText() {
221:     if(_targetEmin<0.0f || _targetEmin<_actualEmin) _targetEmin = _actualEmin;
222:     if(_targetEmax<0.0f || _targetEmax>_actualEmax) _targetEmax = _actualEmax;
223:     tf_ACTUAL_EMIN.setText("  "+_actualEmin);
224:     tf_ACTUAL_EMAX.setText("  "+_actualEmax);
225:     tf_TARGET_EMIN.setText("  "+_targetEmin);
```

```
226:     tf_TARGET_EMAX.setText("  "+_targetEmax);
227:     tf_SMOOTH_N.setText("  "+_smooth_N);
228:     tf_SMOOTH_A.setText("  "+_smooth_A);
229:     tf_SMOOTH_B.setText("  "+_smooth_B);
230:   }
231:
232:   public void updateActualEdgeLengths() {
233:     Ifs ifs = _p3dd.getIfs();
234:     if(ifs!=null) {
235:
236:       VecFloat coord = ifs.getCoord();
237:       //
238:       // the Graph class is composed of the vertices and edges of the mesh
239:       // these edges are not oriented
240:       //
241:       // GraphEdge Graph.getEdge(int i, int j)
242:       //
243:       // gets the edge connecting vertices i and j,
244:       // or null if no such edge exists
245:       //
246:       Graph     g = ifs.getEdges();
247:       //
248:       GraphEdge e = null;
249:       // public class GraphEdge {
250:       //   public int getIndex();
251:       //   public int getOtherVertex(int iV);
252:       //   public int getVertex(int i);
253:       //   public boolean isVertex(int iV);
254:       // }
255:
256:       int nV = g.getNumberOfVertices();
257:       int iV,iV0,iV1,iE;
258:       float dx,dy,dz,eL;
259:       _actualEmin = -1.0f;
260:       _actualEmax = -1.0f;
261:       // this loop allows you to visit each edge exactly onece
262:       // these edges are not oriented
263:       // the inner loop does not traverse all the edges incident to iV,
264:       // but just those which connect iV with another vertex of higher
265:       // index
266:       for(iV=0;iV<nV;iV++) {
267:         for(e=g.getFirstEdge(iV);e!=null;e=g.getNextEdge(e)) {
268:           // remember that these are not half-edges
269:           iE  = e.getIndex();
270:           // get vertex indices of edges' ends
271:           iV0 = e.getVertex(0);
272:           iV1 = e.getVertex(1);
273:           // compute displacement from one vertex to the other
274:           dx = coord.get(3*iV0  )-coord.get(3*iV1  );
275:           dy = coord.get(3*iV0+1)-coord.get(3*iV1+1);
276:           dz = coord.get(3*iV0+2)-coord.get(3*iV1+2);
277:           // compute edge length
278:           eL = (float)Math.sqrt(dx*dx+dy*dy+dz*dz);
279:           // update min-max
280:           if(_actualEmin<0.0f || eL<_actualEmin) _actualEmin = eL;
281:           if(_actualEmax<0.0f || eL>_actualEmax) _actualEmax = eL;
282:         }
283:       }
284:     }
285:   }
286:
287:   public void paintTargetEdges() {
288:     System.out.println("Photo3DPanelMeshing.paintTargetEdges() {");
289:
290:     System.out.println("  _targetEmin = "+_targetEmin);
291:     System.out.println("  _targetEmax = "+_targetEmax);
292:
293:     Ifs ifs = _p3dd.getIfs();
294:     if(ifs!=null) {
295:
296:       // make sure ifs has color per face
297:       ifs.eraseColor();
298:       VecFloat color = ifs.getColor();
299:       int nF = ifs.getNumberOfFaces();
300:       color.pushBack(3*nF,1.0f); // initialize all face colors to white
```

```
301:       ifs.setColorPerVertex(false);
302:
303:       VecFloat coord = ifs.getCoord();
304:       // class GraphFaces extends Graph
305:       // and provides a list of incident faces per edge
306:       GraphFaces g = ifs.getEdges();
307:       GraphEdge e = null;
308:       int nV = g.getNumberOfVertices();
309:       int iV,iV0,iV1,iE,iF,i,nFe;
310:       float dx,dy,dz,eL;
311:       for(iV=0;iV<nV;iV++) {
312:         for(e=g.getFirstEdge(iV);e!=null;e=g.getNextEdge(e)) {
313:           iE  = e.getIndex();
314:           iV0 = e.getVertex(0);
315:           iV1 = e.getVertex(1);
316:           dx = coord.get(3*iV0  )-coord.get(3*iV1  );
317:           dy = coord.get(3*iV0+1)-coord.get(3*iV1+1);
318:           dz = coord.get(3*iV0+2)-coord.get(3*iV1+2);
319:           eL = (float)Math.sqrt(dx*dx+dy*dy+dz*dz);
320:
321:           nFe = g.getNumberOfEdgeFaces(e);
322:           if(eL<_targetEmin) {
323:
324:             for(i=0;i<nFe;i++) {
325:               // get the index of the i-th face incident to e
326:               iF = g.getEdgeFace(e,i);
327:               // set face color to red
328:               color.set(3*iF  ,1.0f);
329:               color.set(3*iF+1,0.0f);
330:               color.set(3*iF+2,0.0f);
331:             }
332:
333:           } else if(eL>_targetEmax) {
334:
335:             for(i=0;i<nFe;i++) {
336:               // get the index of the i-th face incident to e
337:               iF = g.getEdgeFace(e,i);
338:               // set face color to blue
339:               color.set(3*iF  ,0.0f);
340:               color.set(3*iF+1,0.0f);
341:               color.set(3*iF+2,1.0f);
342:             }
343:           }
344:         }
345:       }
346:       ifs.setHasChanged(true);
347:       _p3dd.refresh();
348:     }
349:     System.out.println("}");
350:   }
351:
352:   public void updateState() {
353:     updateActualEdgeLengths();
354:     updateText();
355:   }
356:
357:   private void collapseEdges() {
358:     Ifs ifs = _p3dd.getIfs();
359:     if(ifs!=null && ifs.isTriMesh()) {
360:       /*
361:
362:       // remove properties
363:       ifs.eraseColor();
364:       ifs.eraseNormal();
365:       ifs.eraseTexCoord();
366:
367:       // number of vertices
368:       int nV = ifs.getNumberOfCoord();
369:
370:       VecFloat coord      = ifs.getCoord();
371:       VecInt   coordIndex = ifs.getCoordIndex();
372:
373:       // use an array of ints to indicate which vertices collapse
374:       int i;
375:       int[] father = new int[nV];
376:       for(i=0;i<nV;i++)
377:         father[i] = i;
378:
379:       // select an independent set of edges shorter than _targetEmin
380:       //   use an array of booleans to indicate which vertices are
381:       //   available while traversing the edges
382:
383:       // collapse the chosen vertices
384:       //   for each edge(i,j) in the set
385:       //     if valence(i)>valence(j) set father[j]=i
386:       //     else set father[i]=j
387:
388:       // the used vertices (father[i]==i) are not consecutive 0...nV'-1
389:       // to fix this create a new array newVertex
390:       // so that { newVertex[i] : father[i]==i } = { 0...nV'-1 }
391:       // and if father[i] = j then newVertex[i]=newVertex[j]
392:
393:       // create a newCoord array copying vaues from coord
394:       VecFloat newCoord = new VecFloat();
395:
396:       // create a newCoordIndex array
397:       // replacing each non-negative index i in coordIndex by newVertex[i]
398:       // and deleating triangles with repeated indices
399:       VecInt newCoordIndex = new VecInt();
400:
401:       // swap coord and newCoord
402:       // swap coordIndex and newCoordIndex
403:       coord.swap(newCoord);
404:       coordIndex.swap(newCoordIndex);
405:
406:       // remake connectivity data structures
407:       ifs.makeConnectivity();
408:
409:       // append normals per face for shading
410:       ifs.addNormalPerFace();
411:
412:       // refresh display
413:       ifs.setHasChanged(true);
414:       _p3dd.refresh();
415:
416:       */
417:     }
418:   }
419:
420:   private void refineEdges() {
421:     Ifs ifs = _p3dd.getIfs();
422:     if(ifs!=null && ifs.isTriMesh()) {
423:       /*
424:
425:       // remove properties
426:       ifs.eraseColor();
427:       ifs.eraseNormal();
428:       ifs.eraseTexCoord();
429:
430:       VecFloat coord      = ifs.getCoord();
431:       VecInt   coordIndex = ifs.getCoordIndex();
432:
433:       // mark vertices of edges longer than _targetEmax
434:
435:       int nE = ifs.getNumberOfEdges();
436:
437:       // for each edge {
438:       //   if both ends are marked {
439:       //     create a new vertex at the midpoint of the edge
440:       //     new vertex index is coord.size()/3
441:       //     append coordinates to coord array
442:       //     save the new vertex index in an array indexed by edge index
443:       //   }
444:       // }
445:
446:       VecInt newCoordIndex = new VecInt();
447:
448:       // for each triangle {
449:       //   if three vertices are marked {
450:       //     get vertex indices of three vertices and three edge vertices
```

```
451:       //      and append four triangles to newCoordIndex:
452:       //      be careful with triangle orientation
453:       //
454:       //      newCoordIndex.pushBack(i);
455:       //      newCoordIndex.pushBack(j);
456:       //      newCoordIndex.pushBack(k);
457:       //      newCoordIndex.pushBack(-1);
458:       //
459:       //   } else if two vertices are marked {
460:       //      get vertex indices of three vertices and edge vertex
461:       //      and append two triangles to newCoordIndex:
462:       //   } else {
463:       //      copy triangle to newCoordIndex
464:       //   }
465:       // }
466:
467:       // swap coordIndex and newCoordIndex
468:       coordIndex.swap(newCoordIndex);
469:
470:       // remake connectivity data structures
471:       ifs.makeConnectivity();
472:
473:       // append normals per face for shading
474:       ifs.addNormalPerFace();
475:
476:       // refresh display
477:       ifs.setHasChanged(true);
478:       _p3dd.refresh();
479:
480:       */
481:     }
482:   }
483:
484:   private void flipEdges() {
485:     Ifs ifs = _p3dd.getIfs();
486:     if(ifs!=null && ifs.isTriMesh()) {
487:       /*
488:
489:       // similar to collapsEdges()
490:       //
491:       // remove properties
492:       //
493:       // choose independent set of collapsible edges
494:       // read paintTargetEdges() to learn how to access
495:       // the triangles incident to each edge
496:       //
497:       // use the method
498:       // boolean GraphFaces.isRegularEdge(GraphEdge e)
499:       // as a first "collapsibility" test
500:       // you can also test the angle of triangle normals
501:       // (should be small) and the shape of the projected
502:       // quadrilateral on a tangent plane (should be convex)
503:       // as better choices
504:
505:       // the coord array needs no changes here
506:
507:       // the coordIndex array can be edited in place
508:       // each edge to be flipped is regular, and so, it has two
509:       // incident triangles
510:       // flipping the two triangles only require switching a few
511:       // values of coordIndex
512:
513:       // remake connectivity
514:
515:       // append face normals
516:
517:       // refresh display
518:
519:       */
520:     }
521:   }
522:
523:   private void computeLaplacian(Graph g, float[] v, float[] Kv) {
524:     /*
525:
526:     int nV = g.getNumberOfVertices();
527:     // we assume that v.length>=3*nV && Kv.length>=3*nV
528:
529:     // use a new array to accumulate the vertex weights
530:     float[] w = new float[nV];
531:
532:     // initialize w and Kv to zero
533:
534:     // for each edge e=(i,j) {
535:     //   compute the displacement vector dv = v_i -v_j
536:     //   add  dv to Kv_i
537:     //   add   1 to w_i
538:     //   add -dv to Kv_j
539:     //   add   1 to w_j
540:     // }
541:
542:     // normalize Kv dividing the three components of Kv_i by w_i
543:
544:     // return
545:
546:     */
547:   }
548:
549:   private void smoothCoord() {
550:     Ifs ifs = _p3dd.getIfs();
551:     if(ifs!=null) {
552:       /*
553:
554:       // erase properties
555:
556:       int     nV = ifs.getNumberOfCoord();
557:       float[] v  = ifs.getCoord().getArray();
558:       // use an auxiliary array to compute the laplacian of v
559:       float[] Kv = new float[3*nV];
560:
561:       // for(i=0;i<_smooth_n;i++) {
562:       //   lambda = _smooth_a if i is even and _smooth_b if i is odd
563:       //   compute the laplacian Kv of v
564:       //   update v
565:       //   v -= lambda*Kv
566:       // }
567:
568:       // append face normals
569:
570:       // no need to remake connectivity !!!
571:
572:       // refresh display
573:
574:       */
575:     }
576:   }
577:
578:
579: }
```