

```
1  #include <GL/glut.h>
2  #include <iostream>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <math.h>
6  #include <assert.h>
7  #include <string.h>
8  #include <vector>
9  #include <fstream>
10 #include "arcball.h"
11
12 using namespace std;
13
14 #define PI 3.1415926536
15 int choose = 0;
16
17 typedef struct Vertex
18 {
19     float x, y, z;
20 }Vertex;
21
22 typedef struct Face
23 {
24     int num;
25     int order[3];
26 }Face;
27
28 typedef struct HalfEdge//半边结构
29 {
30     int origin;
31     struct HalfEdge* next;
32     struct HalfEdge* opposite;
33     int IncFace;
34 }HalfEdge;
35
36 typedef struct Map
37 {
38     int vs, ve, e;
39 }Map;
40
41
42 vector<Vertex> vertex;
43 vector<Face> face;
44 vector<HalfEdge*> edge;
45 int e_num;
46 int n_node, n_face, n_edge;
47 int width = 800;
48 int height = 800;
49
50 int readoff()
51 {
52     char theoff[233] = "bunny.off";
53     // cin >> theoff;
```

```
54     ifstream fin(theoff);
55     if (!fin.is_open()) {
56         puts("error opening file.");
57         exit(1);
58     }
59     string isoff;
60     do {
61         fin >> isoff;
62     } while (isoff[0] == '#');
63     if (isoff != "OFF") {
64         puts("this is not a .off file.");
65         exit(1);
66     }
67     fin >> n_node >> n_face >> n_edge;
68
69     for (int i = 0; i < n_node; i++)
70     {
71         Vertex ver;
72         fin >> ver.x >> ver.y >> ver.z;
73         vertex.push_back(ver);
74     }
75     for (int i = 0; i < n_face; i++)
76     {
77         Face f;
78         fin >> f.num >> f.order[0] >> f.order[1] >> f.order[2];
79         face.push_back(f);
80     }
81 }
82
83 void initEdge()//生成半边存入vector
84 {
85     int** map = new int* [n_node];
86     for (int i = 0; i < n_node; i++) {
87         map[i] = new int[n_node];
88     }
89     for (int i = 0; i < n_node; i++)
90     {
91         for (int j = 0; j < n_node; j++)
92         {
93             map[i][j] = -1;
94         }
95     }
96     e_num = 0;
97     for (int i = 0; i < n_face; i++)
98     {
99         HalfEdge* edge1 = new HalfEdge();
100         HalfEdge* edge2 = new HalfEdge();
101         HalfEdge* edge3 = new HalfEdge();
102
103         edge1->origin = face[i].order[0];
104         edge2->origin = face[i].order[1];
105         edge3->origin = face[i].order[2];
106
```

```
107     edge1->next = edge2;
108     edge2->next = edge3;
109     edge3->next = edge1;
110
111     HalfEdge* tmpe = new HalfEdge();
112     if (map[face[i].order[1]][face[i].order[0]] != -1)
113     {
114         tmpe = edge[map[face[i].order[1]][face[i].order[0]]];
115         edge1->opposite = tmpe;
116         tmpe->opposite = edge1;
117     }
118     else
119     {
120         edge1->opposite = NULL;
121         map[face[i].order[0]][face[i].order[1]] = e_num;
122     }
123     e_num++;
124     if (map[face[i].order[2]][face[i].order[1]] != -1)
125     {
126         tmpe = edge[map[face[i].order[2]][face[i].order[1]]];
127         edge2->opposite = tmpe;
128         tmpe->opposite = edge2;
129     }
130     else
131     {
132         edge2->opposite = NULL;
133         map[face[i].order[1]][face[i].order[2]] = e_num;
134     }
135     e_num++;
136     if (map[face[i].order[0]][face[i].order[2]] != -1)
137     {
138         tmpe = edge[map[face[i].order[0]][face[i].order[2]]];
139         edge3->opposite = tmpe;
140         tmpe->opposite = edge3;
141     }
142     else
143     {
144         edge3->opposite = NULL;
145         map[face[i].order[2]][face[i].order[0]] = e_num;
146     }
147     e_num++;
148
149     edge1->IncFace = i;
150     edge2->IncFace = i;
151     edge3->IncFace = i;
152
153     edge.push_back(edge1);
154     edge.push_back(edge2);
155     edge.push_back(edge3);
156 }
157 n_edge = edge.size();
158 }
159
```

```
160 HalfEdge* findOriginEdge(int v)//找到从该定点出发的一条半边
161 {
162     for (int k = 0; k < n_edge; k++)
163     {
164         if (edge[k]->origin == v)
165             return edge[k];
166     }
167     return NULL;
168 }
169
170 void subdivide()
171 {
172     vector<Vertex> vertex2;
173     vector<Face> face2;
174     vector<HalfEdge*> edge2;
175     HalfEdge* he = new HalfEdge();
176     int n;
177     float p_sumx, p_sumy, p_sumz;
178     float px, py, pz;
179     float beta;
180     cout << "细分开始" << endl;
181     for (int i = 0; i < n_node; i++)//旧点更新
182     {
183         he = findOriginEdge(i);
184
185         if (he != NULL)
186         {
187             n = 0;
188             p_sumx = 0;
189             p_sumy = 0;
190             p_sumz = 0;
191             HalfEdge* e = new HalfEdge();
192             e = he->next;
193             int p0 = e->origin;
194
195             while (e->next->origin != p0)
196             {
197                 n++;
198                 p_sumx += vertex[e->next->origin].x;
199                 p_sumy += vertex[e->next->origin].y;
200                 p_sumz += vertex[e->next->origin].z;
201                 HalfEdge* te = new HalfEdge();
202                 te = e->next->opposite;
203                 e = te->next;
204             }
205             n++;
206             p_sumx += vertex[p0].x;
207             p_sumy += vertex[p0].y;
208             p_sumz += vertex[p0].z;
209             beta = 1 / (double)n * (0.625 - pow(0.375 + 0.25 * cos(2 * PI / n),
210                 2));
211
212             px = (1 - n * beta) * vertex[i].x + beta * p_sumx;
```

```

212         py = (1 - n * beta) * vertex[i].y + beta * p_sumy;
213         pz = (1 - n * beta) * vertex[i].z + beta * p_sumz;
214
215         Vertex v;
216         v.x = px;
217         v.y = py;
218         v.z = pz;
219         vertex2.push_back(v);
220     }
221 }
222 int** map1 = new int* [n_node];
223 for (int i = 0; i < n_node; i++) {
224     map1[i] = new int[n_node];
225 }
226
227 cout << "map1=" << sizeof(map1[0]) / sizeof(int) << endl;
228 float qx, qy, qz;
229
230 for (int i = 0; i < n_edge; i++)//新点生成
231 {
232     if (!map1[edge[i]->origin][edge[i]->next->origin])
233     {
234         int p = edge[i]->origin;
235         int pi = edge[i]->next->origin;
236         int pil = edge[i]->next->next->origin;
237         int pi0 = edge[i]->opposite->next->next->origin;
238         qx = 0.375 * (vertex[p].x + vertex[pi].x) + 0.125 * (vertex[pil].x +
239             vertex[pi0].x);
240         qy = 0.375 * (vertex[p].y + vertex[pi].y) + 0.125 * (vertex[pil].y +
241             vertex[pi0].y);
242         qz = 0.375 * (vertex[p].z + vertex[pi].z) + 0.125 * (vertex[pil].z +
243             vertex[pi0].z);
244
245         Vertex v;
246         v.x = qx;
247         v.y = qy;
248         v.z = qz;
249         vertex2.push_back(v);
250
251         map1[edge[i]->origin][edge[i]->next->origin] = vertex2.size() - 1;
252         map1[edge[i]->next->origin][edge[i]->origin] = vertex2.size() - 1;
253     }
254 }
255 /*
256 cout<<"新点"<<endl;
257 for(int i=0;i<vertex2.size();i++)
258 {
259     cout<<vertex2[i].x<<" "<<vertex2[i].y<<" "<<vertex2[i].z<<endl;
260 }
261 */
262 for (int i = 0; i < n_face; i++)//新面
263 {
264     int a, b, c, d, e, f;

```

```
262     a = face[i].order[0];
263     b = face[i].order[1];
264     c = face[i].order[2];
265     d = map1[a][b];
266     e = map1[b][c];
267     f = map1[a][c];
268
269     Face f2;
270     f2.num = 3;
271
272     f2.order[0] = a;
273     f2.order[1] = d;
274     f2.order[2] = f;
275     face2.push_back(f2);
276
277     f2.order[0] = d;
278     f2.order[1] = b;
279     f2.order[2] = e;
280     face2.push_back(f2);
281
282     f2.order[0] = d;
283     f2.order[1] = e;
284     f2.order[2] = f;
285     face2.push_back(f2);
286
287     f2.order[0] = f;
288     f2.order[1] = e;
289     f2.order[2] = c;
290     face2.push_back(f2);
291 }
292
293 n_face = face2.size();
294 n_node = vertex2.size();
295 cout << n_node << " " << n_face << endl;
296
297
298 int** map2 = new int* [n_node];
299 for (int i = 0; i < n_node; i++) {
300     map2[i] = new int[n_node];
301 }
302 for (int i = 0; i < n_node; i++)
303 {
304
305     for (int j = 0; j < n_node; j++)
306     {
307         map2[i][j] = -1;
308     }
309 }
310 e_num = 0;
311 for (int i = 0; i < n_face; i++)//新边
312 {
313     HalfEdge* edge4 = new HalfEdge();
314     HalfEdge* edge5 = new HalfEdge();
```

```
315     HalfEdge* edge6 = new HalfEdge();
316
317     edge4->origin = face2[i].order[0];
318     edge5->origin = face2[i].order[1];
319     edge6->origin = face2[i].order[2];
320
321     edge4->next = edge5;
322     edge5->next = edge6;
323     edge6->next = edge4;
324
325     HalfEdge* tmpe = new HalfEdge();
326     if (map2[face2[i].order[1]][face2[i].order[0]] != -1)
327     {
328         tmpe = edge2[map2[face2[i].order[1]][face2[i].order[0]]];
329         edge4->opposite = tmpe;
330         tmpe->opposite = edge4;
331     }
332     else
333     {
334         edge4->opposite = NULL;
335         map2[face2[i].order[0]][face2[i].order[1]] = e_num;
336     }
337     e_num++;
338     if (map2[face2[i].order[2]][face2[i].order[1]] != -1)
339     {
340         tmpe = edge2[map2[face2[i].order[2]][face2[i].order[1]]];
341         edge5->opposite = tmpe;
342         tmpe->opposite = edge5;
343     }
344     else
345     {
346         edge5->opposite = NULL;
347         map2[face2[i].order[1]][face2[i].order[2]] = e_num;
348     }
349     e_num++;
350     if (map2[face2[i].order[0]][face2[i].order[2]] != -1)
351     {
352         tmpe = edge2[map2[face2[i].order[0]][face2[i].order[2]]];
353         edge6->opposite = tmpe;
354         tmpe->opposite = edge6;
355     }
356     else
357     {
358         edge6->opposite = NULL;
359         map2[face2[i].order[2]][face2[i].order[0]] = e_num;
360     }
361     e_num++;
362
363     edge4->IncFace = i;
364     edge5->IncFace = i;
365     edge6->IncFace = i;
366
367     edge2.push_back(edge4);
```

```
368     edge2.push_back(edge5);
369     edge2.push_back(edge6);
370 }
371 n_edge = edge2.size();
372 /*
373 cout<<"新边"<<endl;
374 for(int i=0;i<edge2.size();i++)
375 {
376     cout<<edge2[i]->origin<<" "<<edge2[i]->next->origin<<" "<<edge2[i]->IncFace<<endl;
377 }
378 */
379 vertex.assign(vertex2.begin(), vertex2.end());
380
381 face.assign(face2.begin(), face2.end());
382
383 edge.assign(edge2.begin(), edge2.end());
384
385 cout << "完成一次细分" << endl;
386 cout << n_node << " " << n_edge << " " << n_face << endl;
387
388
389
390
391
392
393
394
395
396
397
398 }
399
400 ArcBallT arcBall(600.0f, 400.0f);
401 ArcBallT* ArcBall = &arcBall; // new ArcBallT(600.0f,400.0f); //&arcBall;
402
403 void display()
404 {
405     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
406     glColor3f(1.0, 1.0, 1.0);
407     glLoadIdentity();
408     gluLookAt(0.0, 0.0, 5.0,
409              0.0, 0.0, 0.0,
410              0.0, 1.0, 0.0);
411     glScalef(1.0, 2.0, 1.0);
412
413     //glPushMatrix();
414     glTranslatef(ArcBall->rightPan, ArcBall->upPan, 0); //1. 移动
415     glScalef(ArcBall->zoomRate, ArcBall->zoomRate, ArcBall->zoomRate); //2. 缩放
416     glMultMatrixf(ArcBall->Transform.M); //3. 旋转
417
418     glBegin(GL_TRIANGLES);
419     for (int i = 0; i < n_face; i++)
```



```
420     {
421         glVertex3f(vertex[face[i].order[0]].x, vertex[face[i].order[0]].y, vertex
422             [face[i].order[0]].z);
423         glVertex3f(vertex[face[i].order[1]].x, vertex[face[i].order[1]].y, vertex
424             [face[i].order[1]].z);
425         glVertex3f(vertex[face[i].order[2]].x, vertex[face[i].order[2]].y, vertex
426             [face[i].order[2]].z);
427     }
428     glEnd();
429
430     float a, b, c;
431     a = vertex[choose].x;
432     b = vertex[choose].y;
433     c = vertex[choose].z;
434
435     //设置点的大小
436     glPointSize(7);
437     //进行平滑处理
438     glEnable(GL_POINT_SMOOTH);
439     glHint(GL_POINT_SMOOTH, GL_NICEST);
440
441     glBegin(GL_POINTS);
442     //指定的点, 换成绿色
443     glColor3f(0, 255, 0);
444     glVertex3f(a, b, c);
445     glEnd();
446
447     HalfEdge* edge = findOriginEdge(choose);
448
449     do {
450         glBegin(GL_POINTS);
451         //相邻的点, 换成红色
452         glColor3f(255, 0, 0);
453         glVertex3f(vertex[edge->origin].x, vertex[edge->origin].y, vertex[edge-
454             >origin].z);
455         glEnd();
456
457         glBegin(GL_LINE_LOOP);
458         //glLineWidth(3.0f);
459         //相邻的边, 换成蓝色
460         glColor4ub(0, 0, 255, 255);
461         glColor3f(0, 0, 255);
462         glVertex3f(a, b, c);
463         glVertex3f(vertex[edge->opposite->origin].x, vertex[edge->opposite-
464             >origin].y, vertex[edge->opposite->origin].z);
465         glEnd();
466
467         edge = edge->opposite->next;
468     } while (edge != findOriginEdge(choose));
469
470     glPopMatrix();
```

```
468     glutSwapBuffers();
469 }
470
471 void keyboard(unsigned char key, int x, int y)
472 {
473     switch (key)
474     {
475     case '1':
476         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
477         break;
478     case '2':
479         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
480         break;
481     case '3':
482         glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
483         break;
484     case 'w':
485         subdivide();
486         break;
487     }
488     glutPostRedisplay();
489 }
490
491 void reshape(int w, int h) {
492     //定义视口大小
493     glViewport(0, 0, (GLsizei)w, (GLsizei)h);
494     //投影显示
495     glMatrixMode(GL_PROJECTION);
496     //坐标原点在屏幕中心
497     glLoadIdentity();
498     //操作模型视景
499     gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);
500     glMatrixMode(GL_MODELVIEW);
501 }
502
503 //移动
504 void move(int x, int y)
505 {
506     ArcBall->MousePt.s.X = x;
507     ArcBall->MousePt.s.Y = y;
508     ArcBall->upstate();
509     glutPostRedisplay();
510 }
511 //点击
512 void mouse(int button, int state, int x, int y)
513 {
514     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
515         ArcBall->isClicked = true;
516         move(x, y);
517     }
518     else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
519         ArcBall->isClicked = false;
520     else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
```

```
521     ArcBall->isRClicked = true;
522     move(x, y);
523 }
524 else if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
525     ArcBall->isRClicked = false;
526 else if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
527     ArcBall->isRClicked = false;
528 else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) {
529     ArcBall->isMClicked = true;
530     move(x, y);
531 }
532 else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_UP)
533     ArcBall->isMClicked = false;
534 ArcBall->upstate();
535 glutPostRedisplay();
536 }
537
538 int main(int argc, char** argv)
539 {
540     cout << "给出顶点的索引（第几个点），将该点变为绿色，该点连接的边变为蓝色，与 ？
541         该点相连的顶点变为红色：" << endl;
542     cin >> choose;
543     readoff();
544     initEdge();
545
546     glutInit(&argc, argv);
547     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
548     glutInitWindowSize(width, height);
549     glutInitWindowPosition(100, 100);
550     glutCreateWindow("loop");
551     glutReshapeFunc(reshape);
552     glutDisplayFunc(display);
553     glutIdleFunc(display);
554     glutMouseFunc(mouse);
555     glutMotionFunc(move);
556     glutKeyboardFunc(keyboard);
557     glutMainLoop();
558     return 0;
559 }
```