

Software Engineering

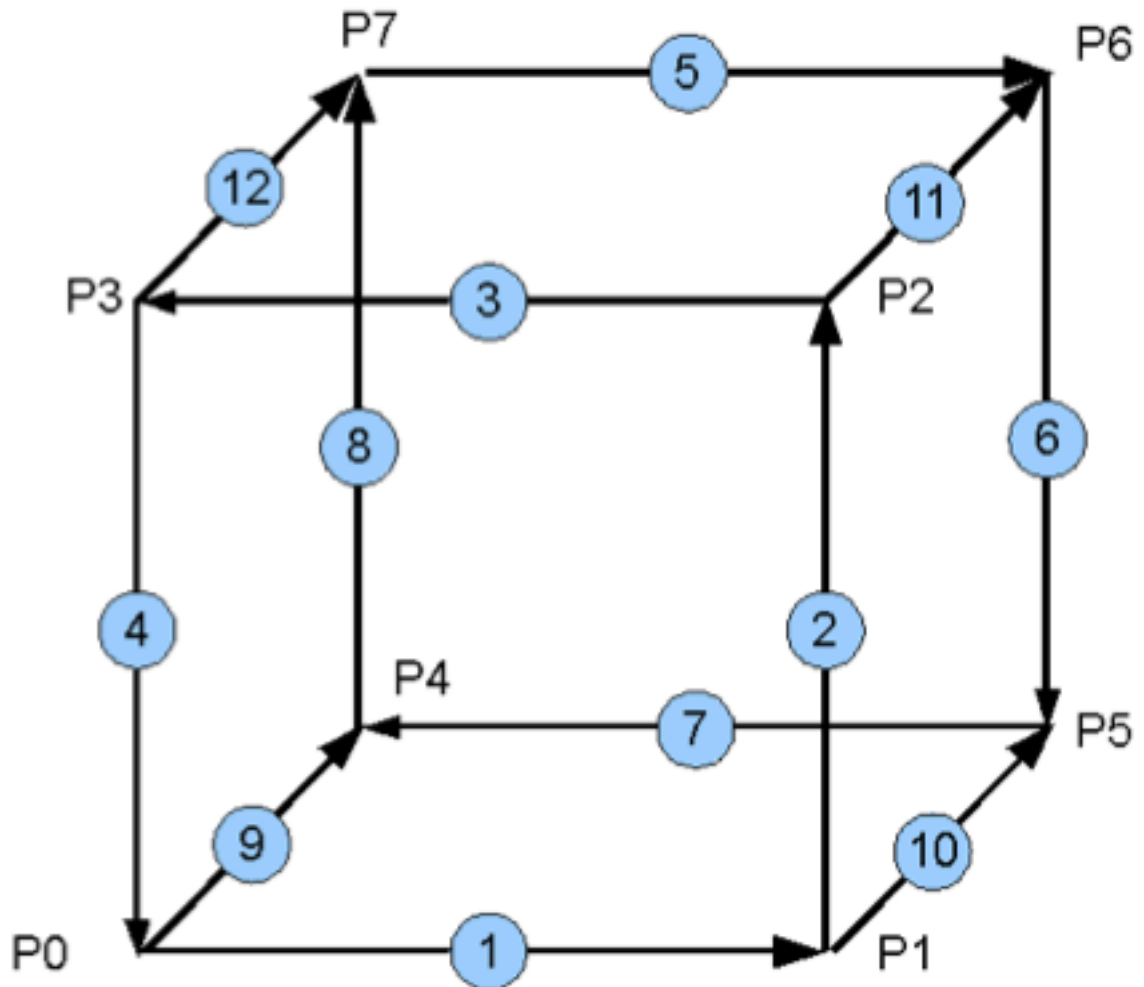
Lab 5 Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

15 November 2013

1 Introduction

Representing cubical model with the Baumgart's winged-edge data structure.



2 Edge Table

Edges represented in data structure hold following information;

1. vertices of this edge, P1 and P2
2. its left and right faces, F1 and F2
3. predecessor(Predccw) and successor(Nextccw) when traversing its left face(ccw)
4. predecessor(Predcw) and successor(Nextcw) when traversing its right face(cw)

Here is the Edge Table following representing closed cubic:

Edge Nb	Start Pt	End Pt	Face ccw F1	Face cw F2	Nccw	Pccw	Ncw	Pcw
1	P0	P1	F1	F3	2	4	10	9
2	P1	P2	F1	F4	3	1	11	10
3	P2	P3	F1	F5	4	2	12	11
4	P3	P0	F1	F6	1	3	9	12
5	P7	P6	F2	F5	6	8	11	12
6	P6	P5	F2	F4	7	5	10	11
7	P5	P4	F2	F3	8	6	9	10
8	P4	P7	F2	F6	5	7	12	9
9	P0	P4	F3	F6	7	1	8	4
10	P1	P5	F4	F3	6	2	7	1
11	P2	P6	F5	F4	5	3	6	2
12	P3	P7	F6	F5	8	4	5	3

Table 1: Edge Table

3 Face Table

A faces are represented only with storing start edge, here is the closed cubic representation data for faces:

Face	Point 1	Point 2	Point 3	Point 4	Start Edge
F1	P0	P1	P2	P3	1
F2	P7	P6	P5	P4	5
F3	P0	P4	P5	P1	9
F4	P1	P5	P6	P2	10
F5	P2	P6	P7	P3	11
F6	P3	P7	P4	P0	12

Table 2: Face Table

4 Exercise 1

In this example, we filled the table of edges(Table 1) in lab.

5 Exercise 2

There are 6 class available for representing winged-edge data structure: Point, Edge, Face, ArrayPoint, ArrayEdge, ArrayFace.

- class Point: Representing a 3D point.
- class Edge: Representing edge with 2 points, 2 face ans also 2 previous and 2 next edges in data structure.
- class Face: Representing a face with start edge.
- class ArrayPoint: Collection of points.
- class ArrayEdge: Collection of edges.
- class ArrayFace: Collection of faces.

In header files, there are always `#ifndef`, `#define`, and `#endif` statements for compile time check for multiple file inclusions that protecting including same header files more than once.

In main program, we can add points. edges and faces using menu selection 1, 4 and 10, respectively.

Displaying points, edges and faces can be accomplish by using menu 3, 6, 12, respectively.

Classes are built to handle or keep track of arrays of pointers, by using `type**`.

In order to achieve accomplishment in this lab, it looks like we should implement our function to array class, and call them from the menu.

6 Exercise 3

Implemented 2 functions in ArrayEdge, for menu 7 and 8; and implemented 3 function in ArrayFace to provide menu 9 and 13.

Listing 1: Exercice.h

```
1
2 // To class ArrayEdge following public function declarations added
3 // "7 : Link segment clockwise - prev and next - (student work here)"
4 void linkSegmentsCWCubic(ArrayPoint*);
5
6 // "8 : Link segment counterclockwise - prev and next - (student work here)"
7 void linkSegmentsCCWCubic(ArrayPoint*);
8
9 // To class ArrayFace-to keep coherence- following public functions declarations added
10 // "9 : Link segment to faces (student work here)"
11 void linkSegmentsToFacesCubic(ArrayEdge*);
12
13 // 13 : Check if face is closed (student work here)"
14 bool isCubicFaceClosed() const;
15
16 // and private function for face traversing
17 bool isCubicFaceClosedTraversal(Face*, int) const;
```

Listing 2: Exercice.cpp

```

1 // "7 : Link segment clockwise - prev and next - (student work here)"
2 void ArrayEdge::linkSegmentsCWCubic(ArrayPoint* pointArray)
3 {
4     if(pointArray->n < 8)
5     {
6         cerr<<"In Point Array, there is not enough element to represent cube."<<endl;
7         return;
8     }
9
10    Edge* edge01;
11    Edge* edge02;
12    Edge* edge03;
13    Edge* edge04;
14    Edge* edge05;
15    Edge* edge06;
16    Edge* edge07;
17    Edge* edge08;
18    Edge* edge09;
19    Edge* edge10;
20    Edge* edge11;
21    Edge* edge12;
22
23    if(this->n < 12)
24    {
25        edge01 = new Edge(pointArray->GetAt(0), pointArray->GetAt(1));
26        edge02 = new Edge(pointArray->GetAt(1), pointArray->GetAt(2));
27        edge03 = new Edge(pointArray->GetAt(2), pointArray->GetAt(3));
28        edge04 = new Edge(pointArray->GetAt(3), pointArray->GetAt(0));
29        edge05 = new Edge(pointArray->GetAt(7), pointArray->GetAt(6));
30        edge06 = new Edge(pointArray->GetAt(6), pointArray->GetAt(5));
31        edge07 = new Edge(pointArray->GetAt(5), pointArray->GetAt(4));
32        edge08 = new Edge(pointArray->GetAt(4), pointArray->GetAt(7));
33        edge09 = new Edge(pointArray->GetAt(0), pointArray->GetAt(4));
34        edge10 = new Edge(pointArray->GetAt(1), pointArray->GetAt(5));
35        edge11 = new Edge(pointArray->GetAt(2), pointArray->GetAt(6));
36        edge12 = new Edge(pointArray->GetAt(3), pointArray->GetAt(7));
37
38        this->AddEdge(edge01);
39        this->AddEdge(edge02);
40        this->AddEdge(edge03);
41        this->AddEdge(edge04);
42        this->AddEdge(edge05);
43        this->AddEdge(edge06);
44        this->AddEdge(edge07);
45        this->AddEdge(edge08);
46        this->AddEdge(edge09);
47        this->AddEdge(edge10);
48        this->AddEdge(edge11);
49        this->AddEdge(edge12);
50    }
51    else
52    {
53        edge01 = this->GetAt(0);
54        edge02 = this->GetAt(1);
55        edge03 = this->GetAt(2);
56        edge04 = this->GetAt(3);
57        edge05 = this->GetAt(4);
58        edge06 = this->GetAt(5);
59        edge07 = this->GetAt(6);
60        edge08 = this->GetAt(7);
61        edge09 = this->GetAt(8);
62        edge10 = this->GetAt(9);
63        edge11 = this->GetAt(10);
64        edge12 = this->GetAt(11);
65    }
66
67    edge01->Nextcw = edge10;
68

```

```

69     edge01->Prevcw = edge09;
70
71     edge02->Nextcw = edge11;
72     edge02->Prevcw = edge10;
73
74     edge03->Nextcw = edge12;
75     edge03->Prevcw = edge11;
76
77     edge04->Nextcw = edge09;
78     edge04->Prevcw = edge12;
79
80     edge05->Nextcw = edge11;
81     edge05->Prevcw = edge12;
82
83     edge06->Nextcw = edge10;
84     edge06->Prevcw = edge11;
85
86     edge07->Nextcw = edge09;
87     edge07->Prevcw = edge10;
88
89     edge08->Nextcw = edge12;
90     edge08->Prevcw = edge09;
91
92     edge09->Nextcw = edge08;
93     edge09->Prevcw = edge04;
94
95     edge10->Nextcw = edge07;
96     edge10->Prevcw = edge01;
97
98     edge11->Nextcw = edge06;
99     edge11->Prevcw = edge02;
100
101     edge12->Nextcw = edge05;
102     edge12->Prevcw = edge03;
103
104     return;
105 }
106
107 // "8 : Link segment counterclockwise - prev and next - (student work here)"
108 void ArrayEdge::linkSegmentsCCWCubic(ArrayPoint* pointArray)
109 {
110     if(pointArray->n < 8)
111     {
112         cerr<<"In Point Array, there is not enough element to represent cube."<<endl;
113         return;
114     }
115
116     Edge* edge01;
117     Edge* edge02;
118     Edge* edge03;
119     Edge* edge04;
120     Edge* edge05;
121     Edge* edge06;
122     Edge* edge07;
123     Edge* edge08;
124     Edge* edge09;
125     Edge* edge10;
126     Edge* edge11;
127     Edge* edge12;
128
129     if(this->n < 12)
130     {
131         edge01 = new Edge(pointArray->GetAt(0), pointArray->GetAt(1));
132         edge02 = new Edge(pointArray->GetAt(1), pointArray->GetAt(2));
133         edge03 = new Edge(pointArray->GetAt(2), pointArray->GetAt(3));
134         edge04 = new Edge(pointArray->GetAt(3), pointArray->GetAt(0));
135         edge05 = new Edge(pointArray->GetAt(7), pointArray->GetAt(6));
136         edge06 = new Edge(pointArray->GetAt(6), pointArray->GetAt(5));
137         edge07 = new Edge(pointArray->GetAt(5), pointArray->GetAt(4));

```

```

138     edge08 = new Edge(pointArray->GetAt(4), pointArray->GetAt(7));
139     edge09 = new Edge(pointArray->GetAt(0), pointArray->GetAt(4));
140     edge10 = new Edge(pointArray->GetAt(1), pointArray->GetAt(5));
141     edge11 = new Edge(pointArray->GetAt(2), pointArray->GetAt(6));
142     edge12 = new Edge(pointArray->GetAt(3), pointArray->GetAt(7));
143
144     this->AddEdge(edge01);
145     this->AddEdge(edge02);
146     this->AddEdge(edge03);
147     this->AddEdge(edge04);
148     this->AddEdge(edge05);
149     this->AddEdge(edge06);
150     this->AddEdge(edge07);
151     this->AddEdge(edge08);
152     this->AddEdge(edge09);
153     this->AddEdge(edge10);
154     this->AddEdge(edge11);
155     this->AddEdge(edge12);
156 }
157 else
158 {
159     edge01 = this->GetAt(0);
160     edge02 = this->GetAt(1);
161     edge03 = this->GetAt(2);
162     edge04 = this->GetAt(3);
163     edge05 = this->GetAt(4);
164     edge06 = this->GetAt(5);
165     edge07 = this->GetAt(6);
166     edge08 = this->GetAt(7);
167     edge09 = this->GetAt(8);
168     edge10 = this->GetAt(9);
169     edge11 = this->GetAt(10);
170     edge12 = this->GetAt(11);
171 }
172
173 edge01->Nextccw = edge02;
174 edge01->Prevccw = edge04;
175
176 edge02->Nextccw = edge03;
177 edge02->Prevccw = edge01;
178
179 edge03->Nextccw = edge04;
180 edge03->Prevccw = edge02;
181
182 edge04->Nextccw = edge01;
183 edge04->Prevccw = edge03;
184
185 edge05->Nextccw = edge06;
186 edge05->Prevccw = edge08;
187
188 edge06->Nextccw = edge07;
189 edge06->Prevccw = edge05;
190
191 edge07->Nextccw = edge08;
192 edge07->Prevccw = edge06;
193
194 edge08->Nextccw = edge05;
195 edge08->Prevccw = edge07;
196
197 edge09->Nextccw = edge07;
198 edge09->Prevccw = edge01;
199
200 edge10->Nextccw = edge06;
201 edge10->Prevccw = edge02;
202
203 edge11->Nextccw = edge05;
204 edge11->Prevccw = edge03;
205
206 edge12->Nextccw = edge08;

```

```

207     edge12->Prevccw = edge04;
208
209     return;
210 }
211
212 // "9 : Link segment to faces (student work here)"
213 void ArrayFace::linkSegmentsToFacesCubic(ArrayEdge* edgeArray)
214 {
215     if(edgeArray->n < 12)
216     {
217         cerr<<"In Edge Array, there is not enough edge to represent cube."<<endl;
218         return;
219     }
220
221     Face *F1;
222     Face *F2;
223     Face *F3;
224     Face *F4;
225     Face *F5;
226     Face *F6;
227
228     if(this->n < 6)
229     {
230         F1 = new Face;
231         F2 = new Face;
232         F3 = new Face;
233         F4 = new Face;
234         F5 = new Face;
235         F6 = new Face;
236
237         this->AddFace(F1);
238         this->AddFace(F2);
239         this->AddFace(F3);
240         this->AddFace(F4);
241         this->AddFace(F5);
242         this->AddFace(F6);
243     }
244     else
245     {
246         F1 = this->GetAt(0);
247         F2 = this->GetAt(1);
248         F3 = this->GetAt(2);
249         F4 = this->GetAt(3);
250         F5 = this->GetAt(4);
251         F6 = this->GetAt(5);
252     }
253
254     Edge* edge01 = edgeArray->GetAt(0);
255     Edge* edge02 = edgeArray->GetAt(1);
256     Edge* edge03 = edgeArray->GetAt(2);
257     Edge* edge04 = edgeArray->GetAt(3);
258     Edge* edge05 = edgeArray->GetAt(4);
259     Edge* edge06 = edgeArray->GetAt(5);
260     Edge* edge07 = edgeArray->GetAt(6);
261     Edge* edge08 = edgeArray->GetAt(7);
262     Edge* edge09 = edgeArray->GetAt(8);
263     Edge* edge10 = edgeArray->GetAt(9);
264     Edge* edge11 = edgeArray->GetAt(10);
265     Edge* edge12 = edgeArray->GetAt(11);
266
267     edge01->Fccw = F1;
268     edge01->Fcw = F3;
269
270     edge02->Fccw = F1;
271     edge02->Fcw = F4;
272
273     edge03->Fccw = F1;
274     edge03->Fcw = F5;
275

```

```

276     edge04->Fccw = F1;
277     edge04->Fcw = F6;
278
279     edge05->Fccw = F2;
280     edge05->Fcw = F5;
281
282     edge06->Fccw = F2;
283     edge06->Fcw = F4;
284
285     edge07->Fccw = F2;
286     edge07->Fcw = F3;
287
288     edge08->Fccw = F2;
289     edge08->Fcw = F6;
290
291     edge09->Fccw = F3;
292     edge09->Fcw = F6;
293
294     edge10->Fccw = F4;
295     edge10->Fcw = F3;
296
297     edge11->Fccw = F5;
298     edge11->Fcw = F4;
299
300     edge12->Fccw = F6;
301     edge12->Fcw = F5;
302
303     F1->Start = edge01;
304     F2->Start = edge05;
305     F3->Start = edge09;
306     F4->Start = edge10;
307     F5->Start = edge11;
308     F6->Start = edge12;
309
310     return;
311 }

```

7 Exercise 4

In order to check our cubical is represented closed in our data structure, here are the functions to check it.

Listing 3: Exercice.h

```

1 // 13 : Check if face is closed (student work here)"
2 bool isCubicFaceClosed() const;
3 private:
4 bool isCubicFaceClosedTraversal(Face*, int) const;

```

Listing 4: Exercice.cpp

```

1 // 13 : Check if face is closed (student work here)"
2 bool ArrayFace::isCubicFaceClosed() const
3 {
4     bool isCubicFaceClosed = true;
5
6     for(int i = 0; i < this->n; i++)
7     {
8         Face* face = this->GetAt(i);
9
10        bool traversal = this->isCubicFaceClosedTraversal(face, 4);
11
12        if(!traversal)
13        {

```



```

14         isCubicFaceClosed = false;
15         break;
16     }
17 }
18
19 return isCubicFaceClosed;
20 }
21
22 bool ArrayFace::isCubicFaceClosedTraversal(Face* face, int numberOfEdges) const
23 {
24     Edge* temp = face->Start;
25
26     for(int i = 0; i < numberOfEdges; i++)
27     {
28         if(temp->Fccw == face)
29             temp = temp->Nextccw;
30         else
31             temp = temp->Prevcw;
32     }
33
34     if(temp == face->Start)
35         return true;
36     return false;
37 }

```

Here are the some function call from main.cpp

Listing 5: main.cpp

```

1
2 //_____
3 //student work here
4 //_____
5
6     case 7 : {
7         //add your code here
8
9         // "7 : Link segment clockwise - prev and next - (student work here)"
10        ArrayE.linkSegmentsCWCubic(&ArrayP);
11
12
13        }break;
14
15 //_____
16 //student work here
17 //_____
18
19     case 8 : {
20         //add your code here
21
22         // "8 : Link segment counterclockwise - prev and next - (student work here)"
23        ArrayE.linkSegmentsCCWCubic(&ArrayP);
24
25
26        }break;
27
28 //_____
29 //student work here
30 //_____
31
32     case 9 : {
33         //add your code here
34
35         // "9 : Link segment to faces (student work here)"
36        ArrayF.linkSegmentsToFacesCubic(&ArrayE);
37
38        }break;
39
40     case 13 : {

```

```

41         //student work here
42
43     // "13 : Check if face is closed (student work here)"
44     bool isClosed = ArrayF.isCubicFaceClosed();
45     if(isClosed)
46         cout<<"Cubic Faces looks closed ^_^!"<<endl;
47     else
48         cout<<"Cubic Faces looks like NOT closed :'("<<endl;
49
50     }break;

```

8 Results/Lessons Learned

This lab gave us the message not to use winged-edge again, otherwise it's data structure is a pain.