Model Compression

Present by Chun-Tse Hsiao

2007/05/02

Model Data

- Geometry data.
- Connectivity data (Topology).
- Property data.

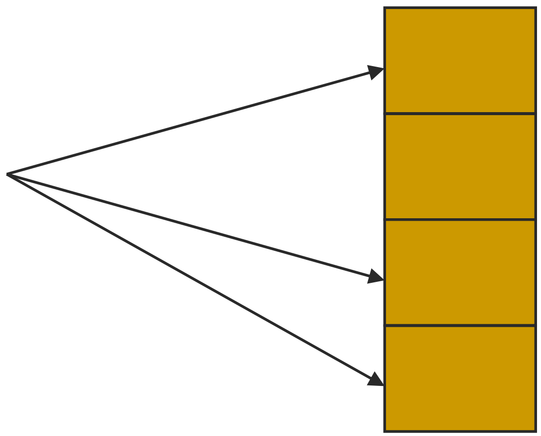
- In this presentation, we discuss compression of topology data.

Ordinary model representation

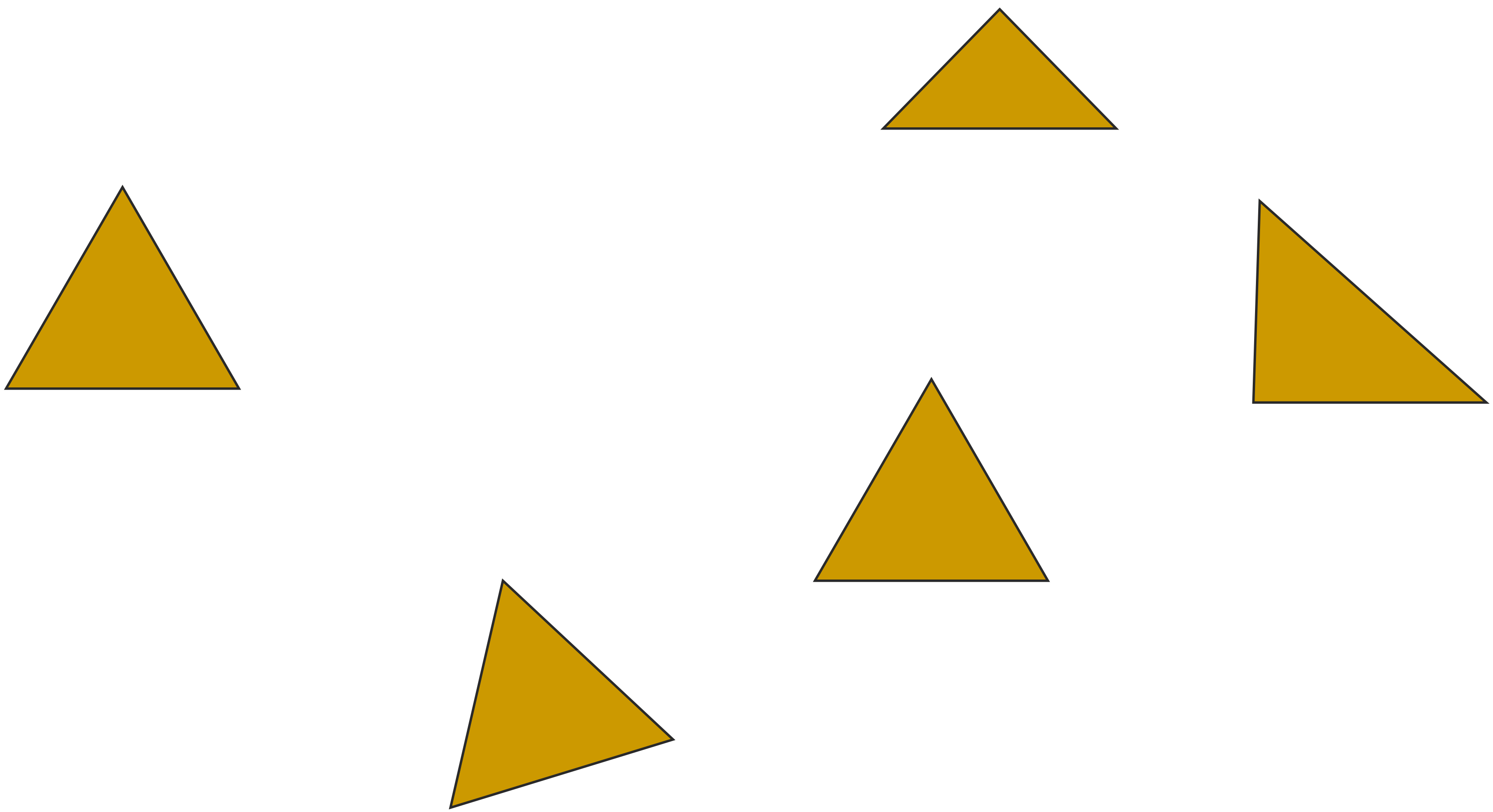
- Tri{

  int vindex[3];

  }



- The representation above can represent arbitrary triangle model.

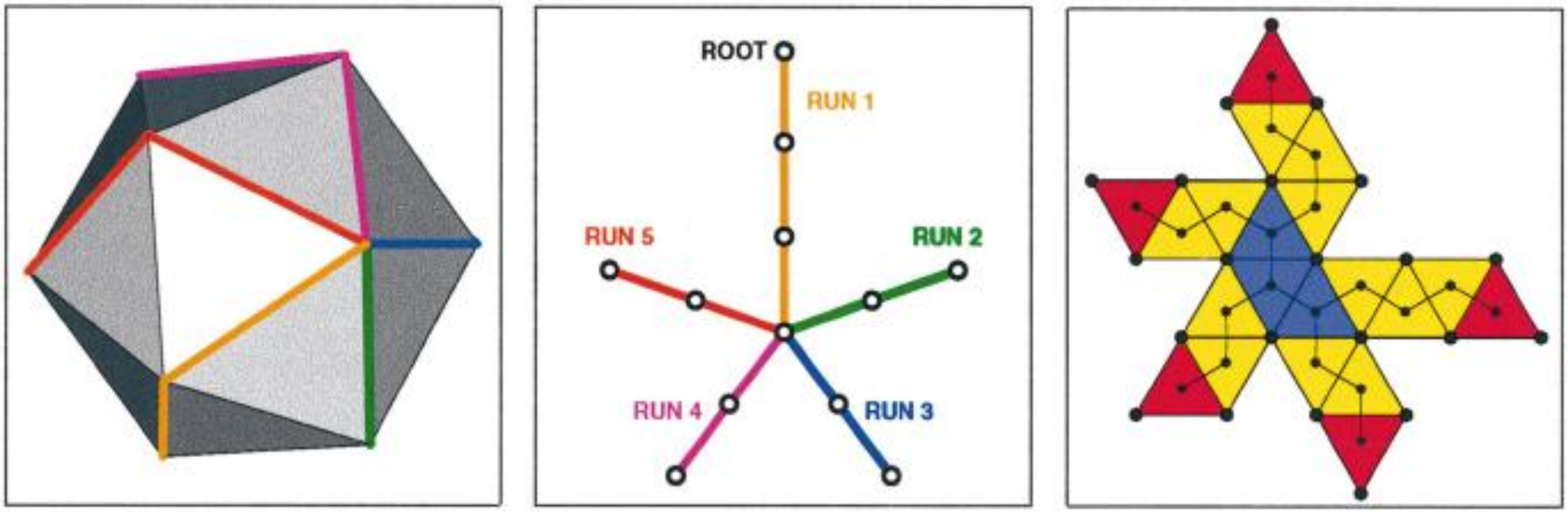


Triangle strips

Complexity of topology

- Topology complexity is proportional to total edge number.
- Edges describe Topology information.

Previous work



Topology can be represented by vertex spanning tree and face spanning tree! Total edge number of two tree is equal to total edge number of original model.

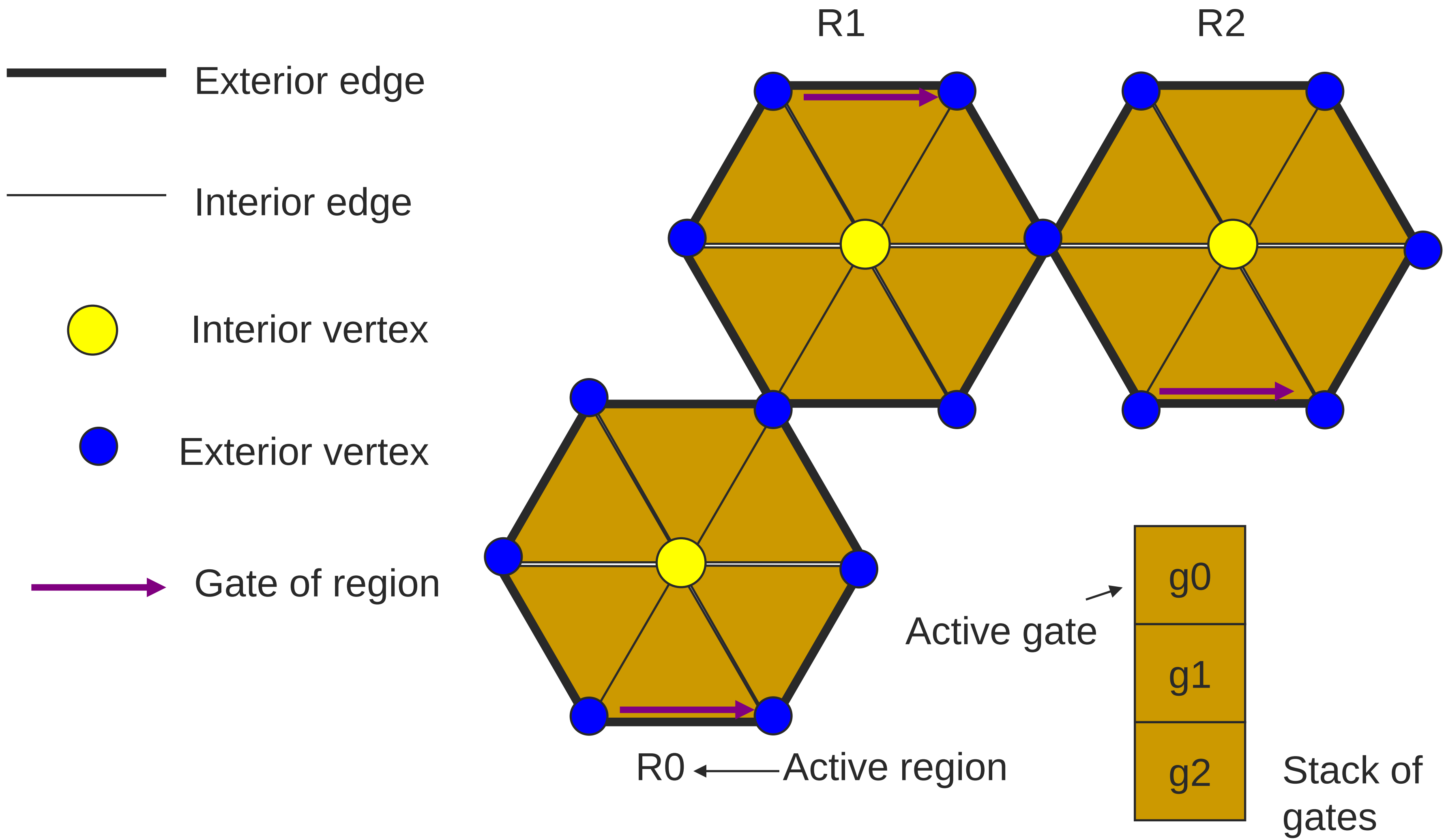
Edgebreaker

Definition

- Simple mesh
  - 2-manifold triangle mesh
  - Connected & Orientable
  - Have no handle
  - Have no boundary or have a connected, manifold, close curve boundary.

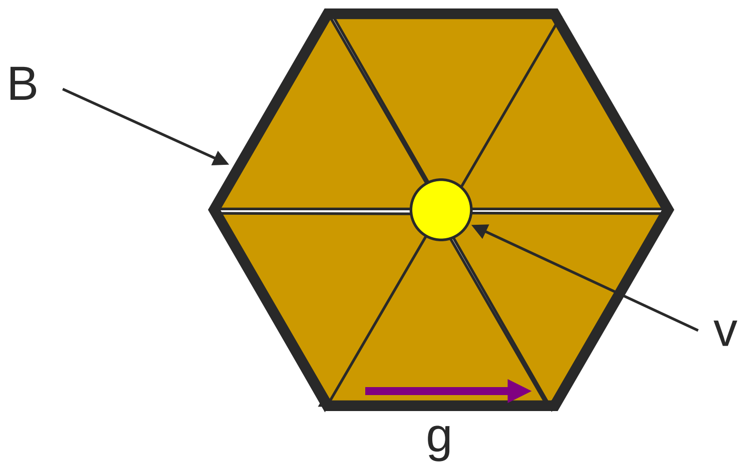
- Edgebreaker compression algorithm performs a series of steps.
- Each steps remove one triangle from current mesh.
- The remaining portion mesh is composed of one or several simple mesh region.

3 regions example

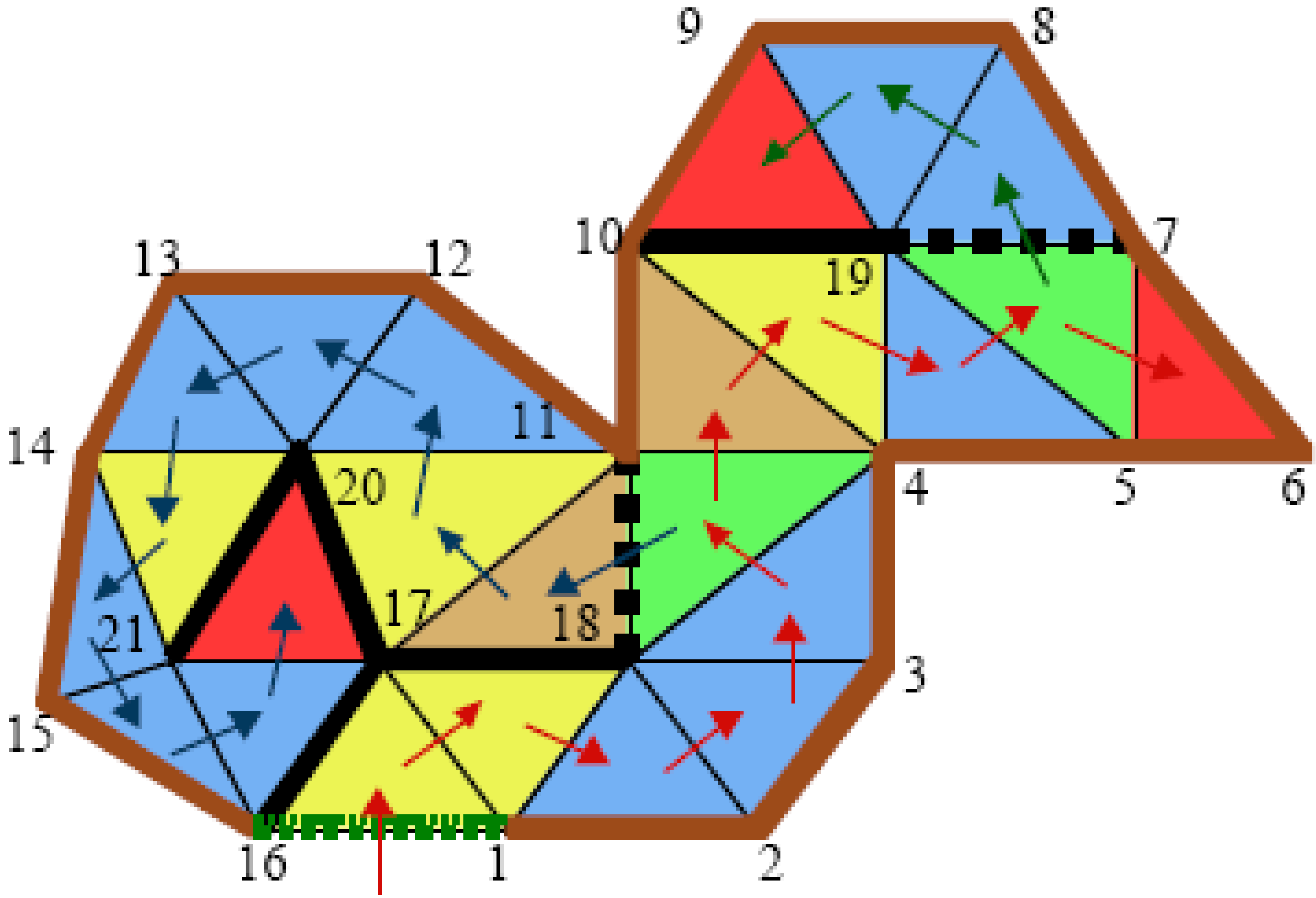


- The regions processed in the same order as gates in the gate stack.
- R0 will compress first, R1 next, and finally R2.
- Each step will remove one triangle from active region. It may introduce new region! New region will be tracked by gate stack.

- Notations
  - Border of active region : B
  - Gate of active region : g
  - Vertex in the same triangle not bound g : v

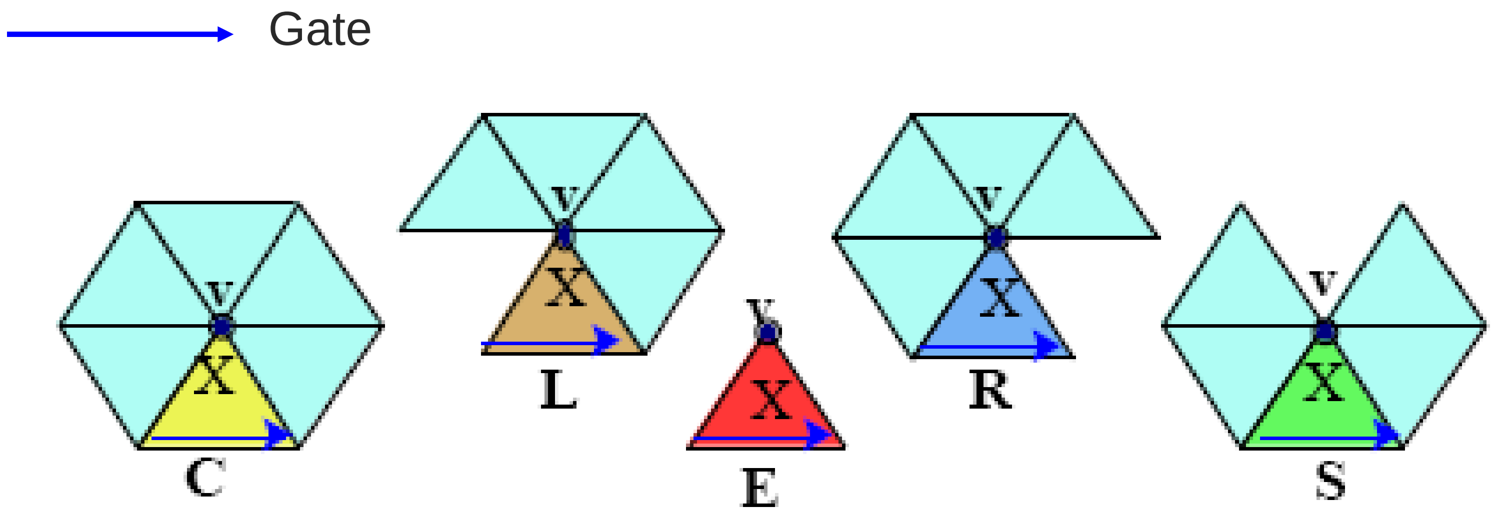


Compressing simple meshes



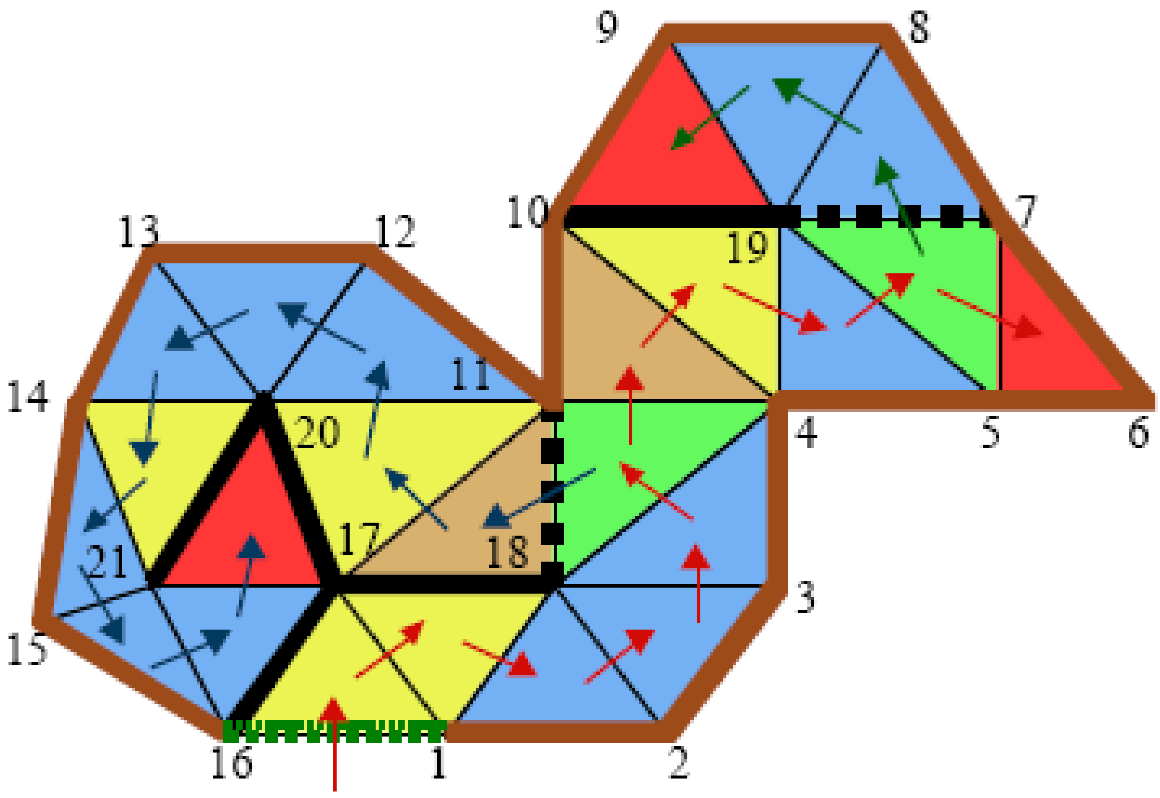
Five operation of edge breaker.

- Depend on the relation of v, g, B…there will be five possible operations.



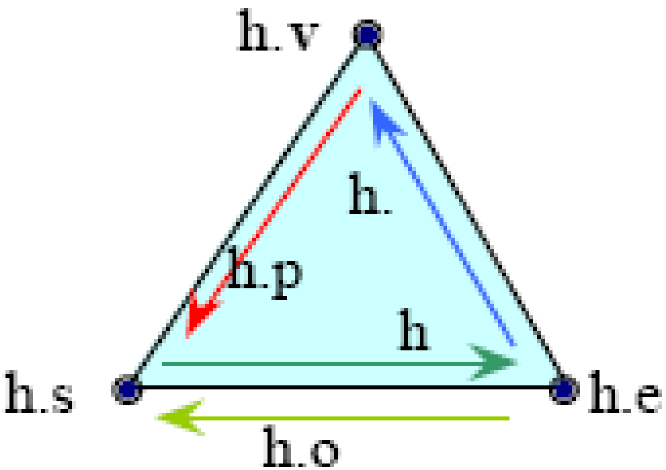
- We record series of operations, denote it by H.
- We record series of vertices, in the order in which they are reached by C operations, denote it by P.
- If mesh has boundary, P is initialized to references of vertices of initial loop B.
- H actually represent connectivity information of the mesh!

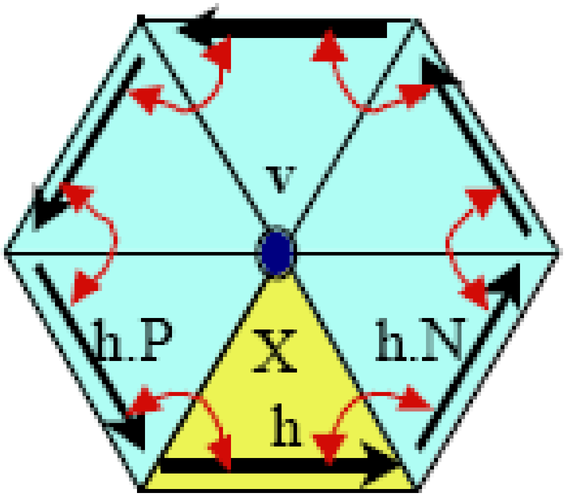
Compressing simple meshes



H = CCRRRSLCRSERRELCRRRCRRRE

Half-edge data structure

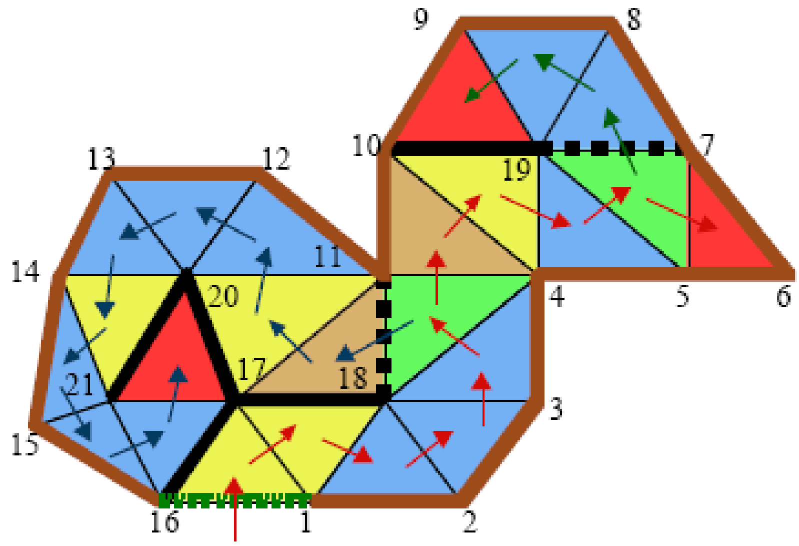




Each vertex v has flag v.m = true if it’s visited before.

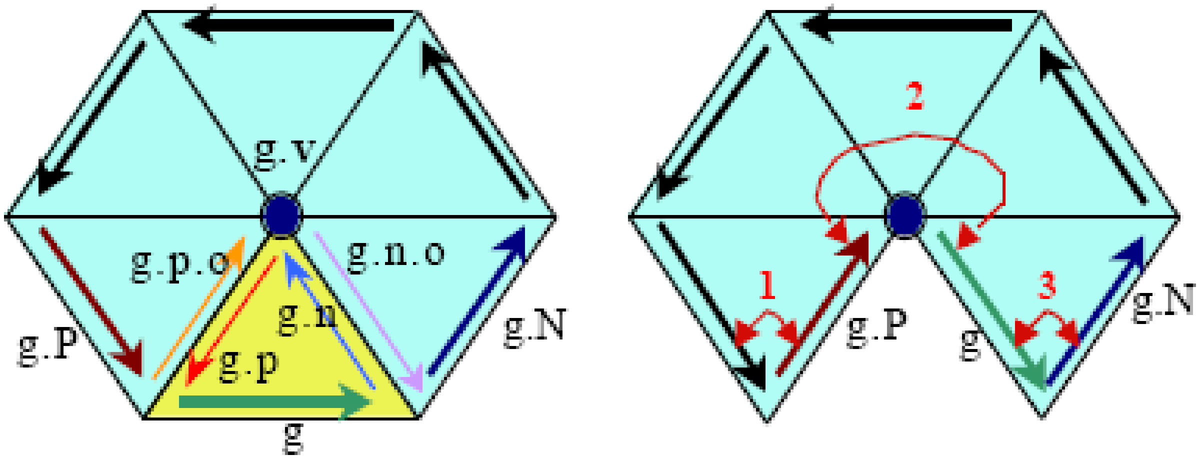
Each half-edge h has flag v.m = true if it’s in bounding loop of remaining portion of mesh

Initialize



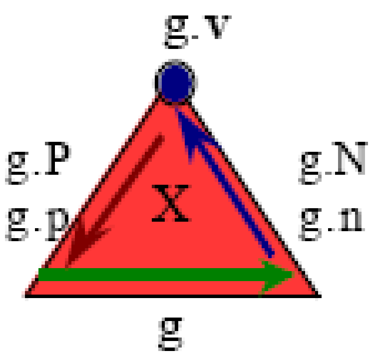
P = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
H = {}
Set V1~V16's visit flag to true.
Set boundary flag of half-edges on boundary to true.
g = (16, 1)
Stack = {g}

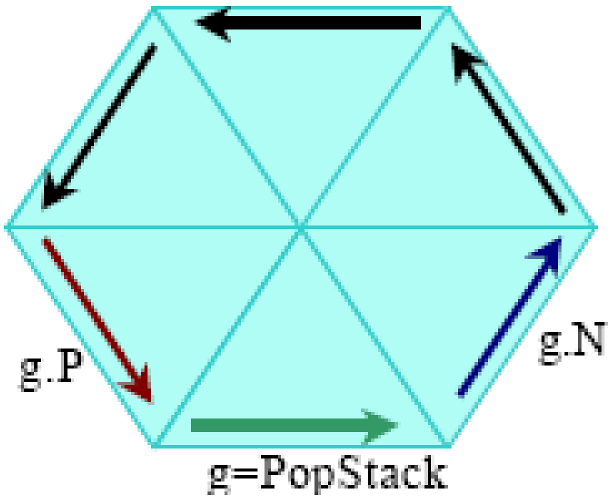
Case C



H=H|C;	# append C to history
P=**P**|g.v;	# append **v** to **P**
g.m=0; g.p.o.m=1;	# update flags
g.n.o.m=1; g.v.m=1;	
g.p.o.P=g.P; g.P.N=g.p.o;	# fix red link 1 in B
g.p.o.N=g.n.o; g.n.o.P=g.p.o;	# fix red link 2 in B
g.n.o.N;=g.N; g.N.P=g.n.o	# fix red link 3 in B
g=g.n.o; StackTop=g;	# move gate

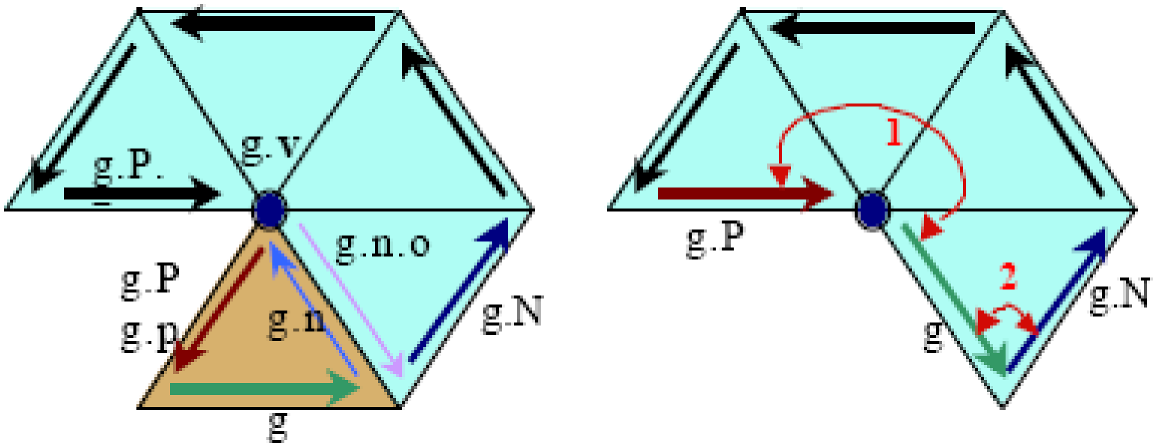
Case E





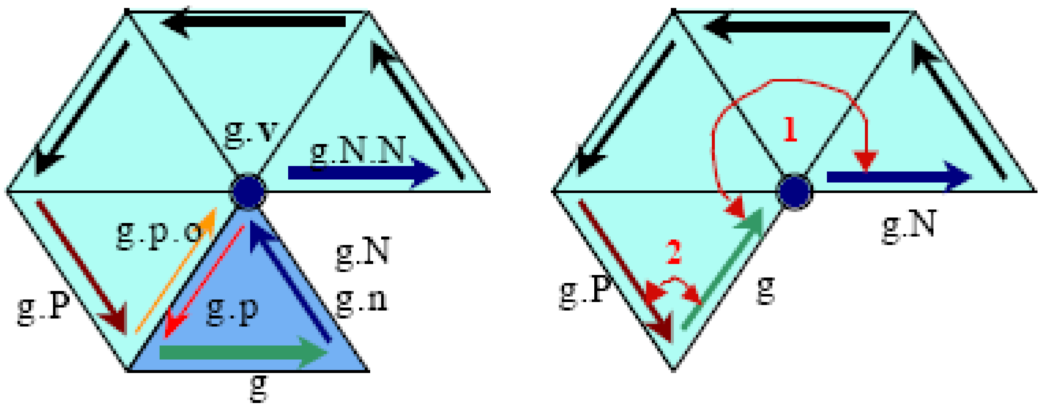
H=H|E; # append E to the history
g.m=0; g.n.m=0; g.p.m=0; # unmark edges
PopStack; g=StackTop; # pop stack: next region

Case L



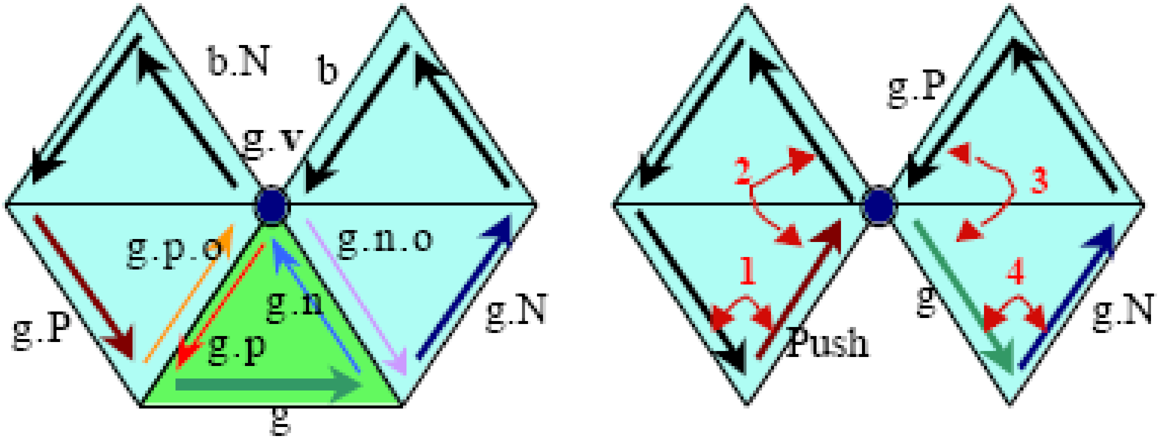
H=H|L;	# append L to history
g.m=0; g.P.m=0; g.n.o.m=1;	# update marks
g.P.P.N=g.n.o; g.n.o.P=g.P.P;	# fix red link 1 in B
g.n.o.N=g.N; g.N.P=g.n.o;	# fix red link 2 in B
g=g.n.o; StackTop=g;	# move gate

Case R



H=H|R;	# append R to history
g.m=0; g.N.m=0; g.p.o.m=1;	# update marks
g.N.N.P=g.p.o; g.p.o.N=g.N.N;	# fix red link 1 in B
g.p.o.P=g.P; g.P.N=g.p.o;	# fix red link 2 in B
g=g.p.o; StackTop=g;	# move g

Case S




```
H=H|S;                              # append S to history
g.m=0; g.p.o.m=1; g.n.o.m=1;        # update marks
b=g.n;                              # initial candidate for b
WHILE NOT b.m DO b=b.o.p;           # turn around v to marked b
g.P.N=g.p.o; g.p.o.P=g.P;           # fix red link 1 in B
g.p.o.N=b.N; b.N.P=g.p.o;           # fix red link 2 in B
b.N=g.n.o; g.n.o.P=b;               # fix red link 3 in B
g.n.o.N=g.N;  g.N.P=g.n.o;          # fix red link 4 in B
StackTop=g.p.o; PushStack;          # push g.p.o on stack
g=g.n.o; StackTop=g                 # move g
```

Decompression

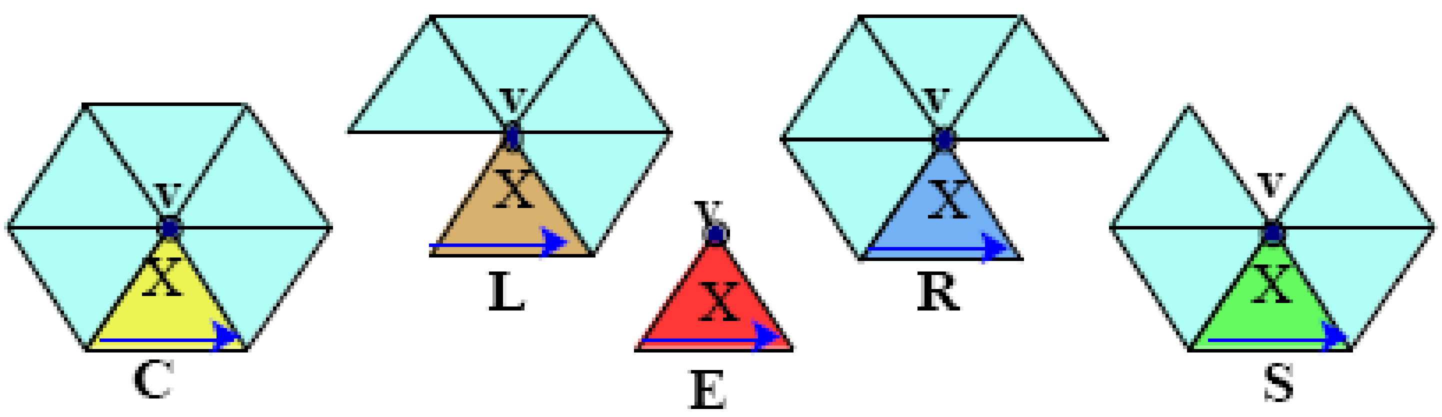
- Two pass decompression
- Preprocessing (Parse H)
  - Preprocessing |T|, |VE|, |VI|, and offset of all S operations in offset table O.
- Generation
  - Create triangles in the order in which they were deleted by compression process.

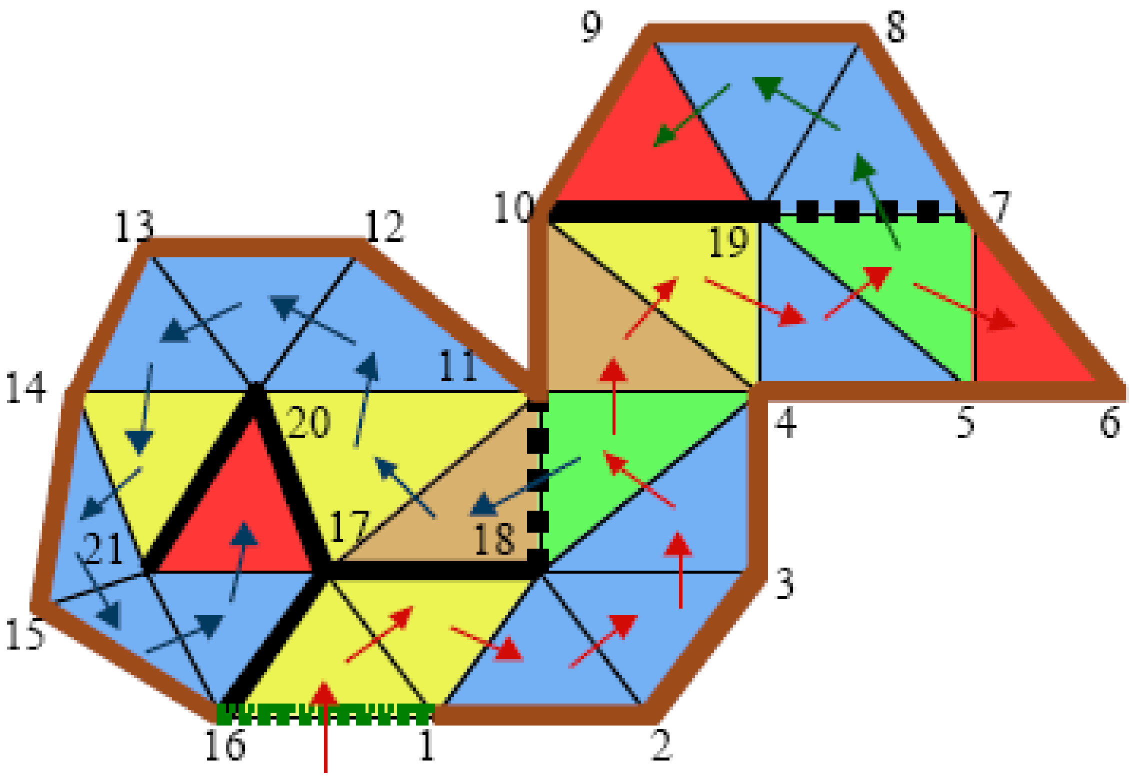
Preprocessing

- t : total number of operation. (0)
- d : |S| - |E|, after last E, it'll be negative. (0)
- c : |C|. (0)
- e : 3|E| + |L| + |R| - |C| - |S|. Final value is |VE|. (0)
- s : |S|. (0)
- Stack : save (e, s) pairs resulting from S operations. Used during E operations to compute the offset. ({})
- O : offset table. ({})

Preprocessing

- Case S : e-=1; s+=1; push(e, s); d+=1.
- Case E : e+=3; (e’, s’)=pop; O[s’]=e-e’-2; d-=1;
- Case C : e-=1; c+=1;
- Case R : e+=1;
- Case L : e+=1;





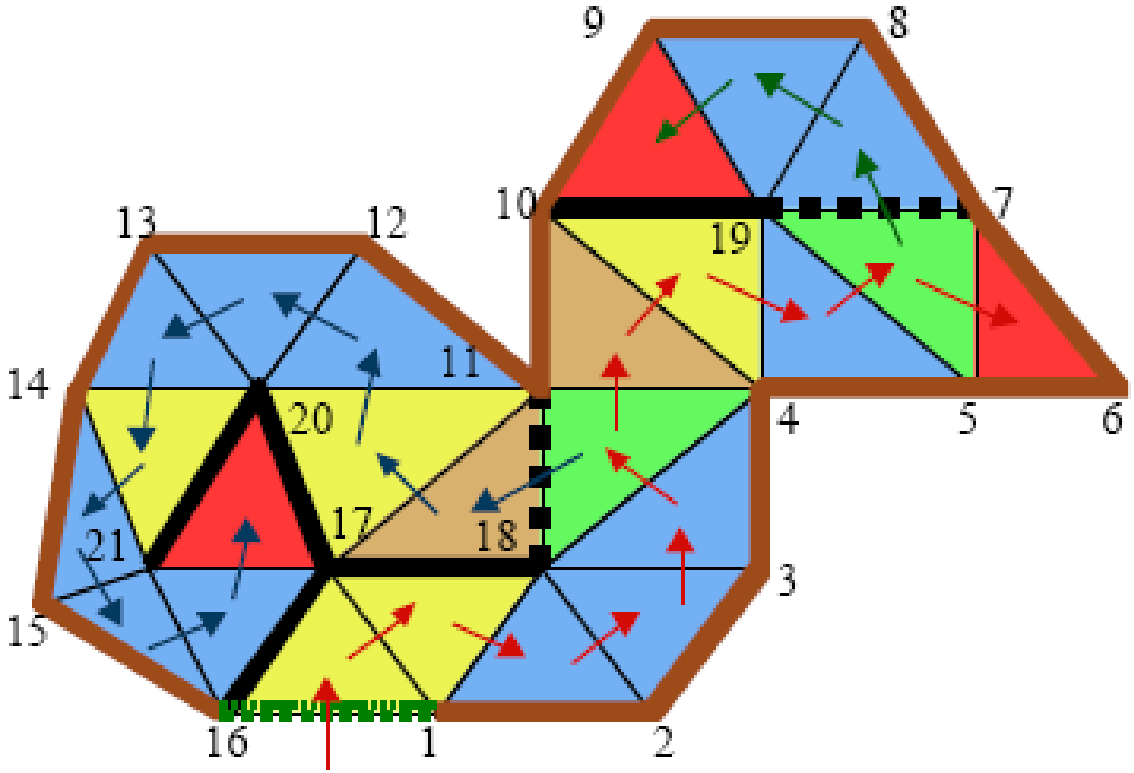
	C	C	R	R	R	S	L	C	R	S	E	R	R	E	L	C	R	R	R	C	R	R	R	E
d	0	0	0	0	0	1	1	1	1	2	1	1	1	0	0	0	0	0	0	0	0	0	0	-1
c	1	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	5	5	5	5	5
e	-1	-2	-1	0	1	0	1	0	1	0	3	4	5	8	9	8	9	10	11	10	11	12	13	16
s	0	0	0	0	0	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
e'						0	0	0	0	0	0	0	0											
s'						1	1	1	1	2	1	1	1											
O[*s*]											1		6											

Generation

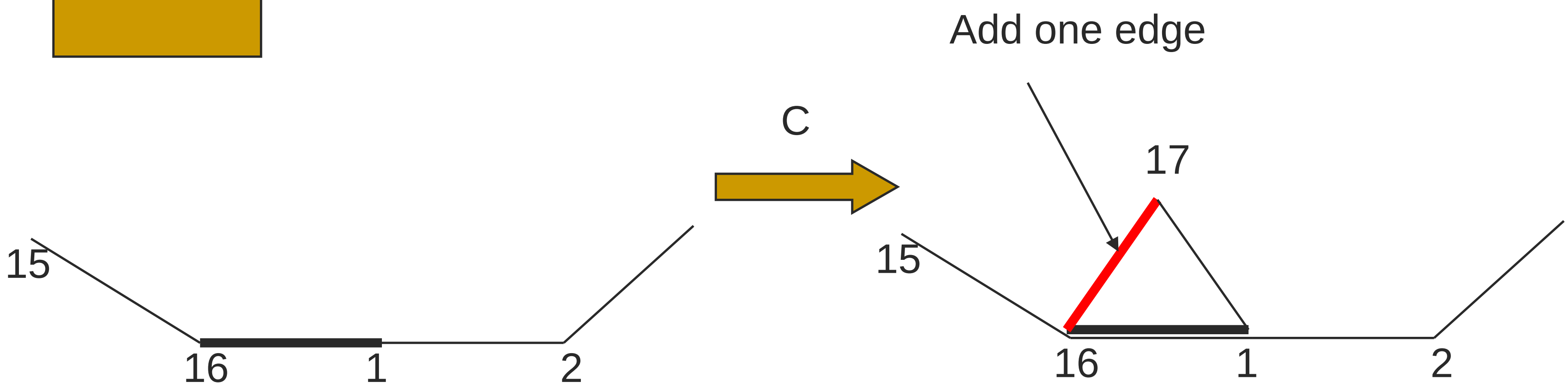
- Allocate a table of triangle-vertex incident relation TV of |T| entries.
- Initialize vertex counter c to |VE|.
- Construct the bounding loop B of |VE| vertices. (circular doubly-linked list)
- Create gate stack with single entry refer to first in B.
- Initialize t (triangle count) , s (number of S op) to zero.

Generation

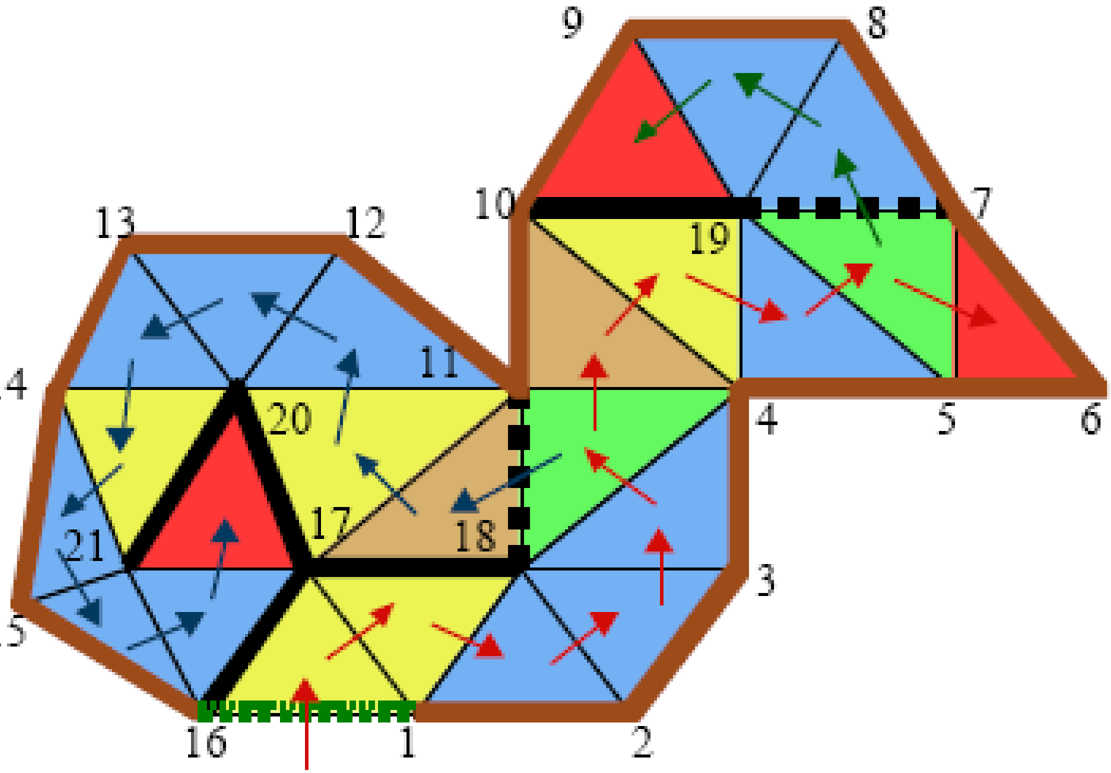
- Reading H.
- For each operation, increase t, store vertex index to TV[t].
- Update B, g (gate), and stack if necessary.

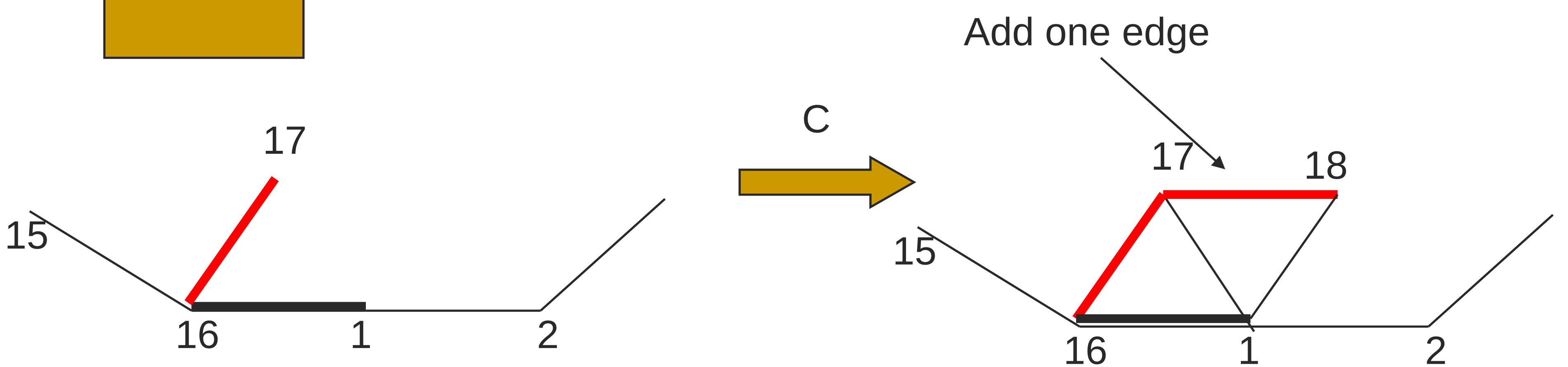


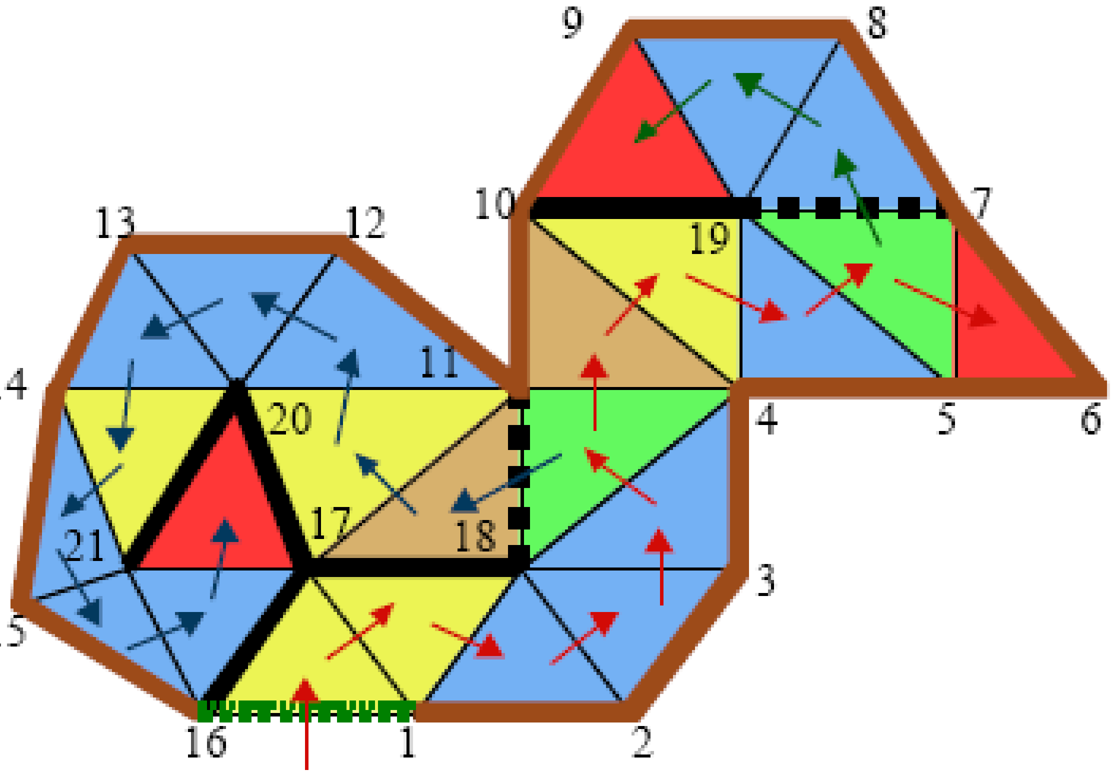
9
8
10
7
13
12
19
11
14
20
4
5
6
17
18
21
3
15
16
1
2



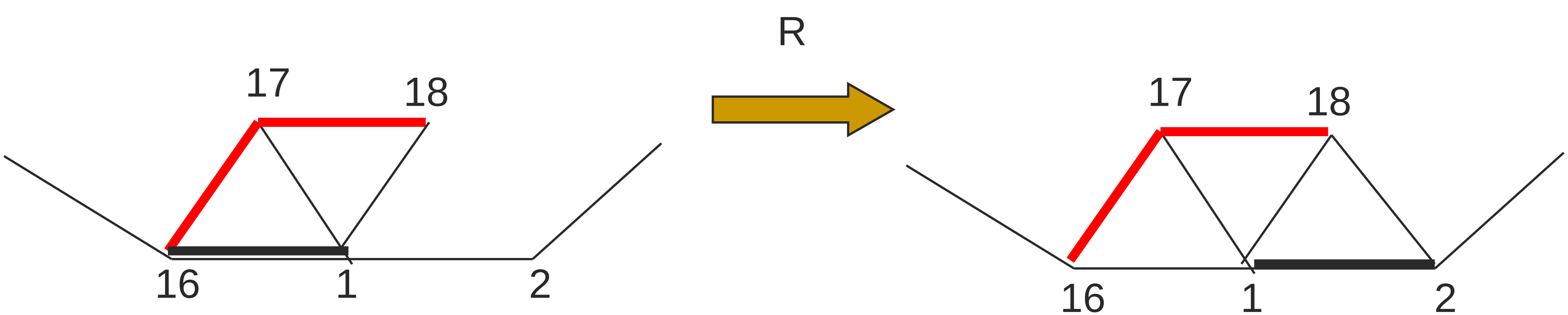
Add one edge
C
17
15
15
16
1
2
16
1
2


9
8
10
7
13
12
19
11
14
4
5
6
20
17
18
21
3
15
16
1
2

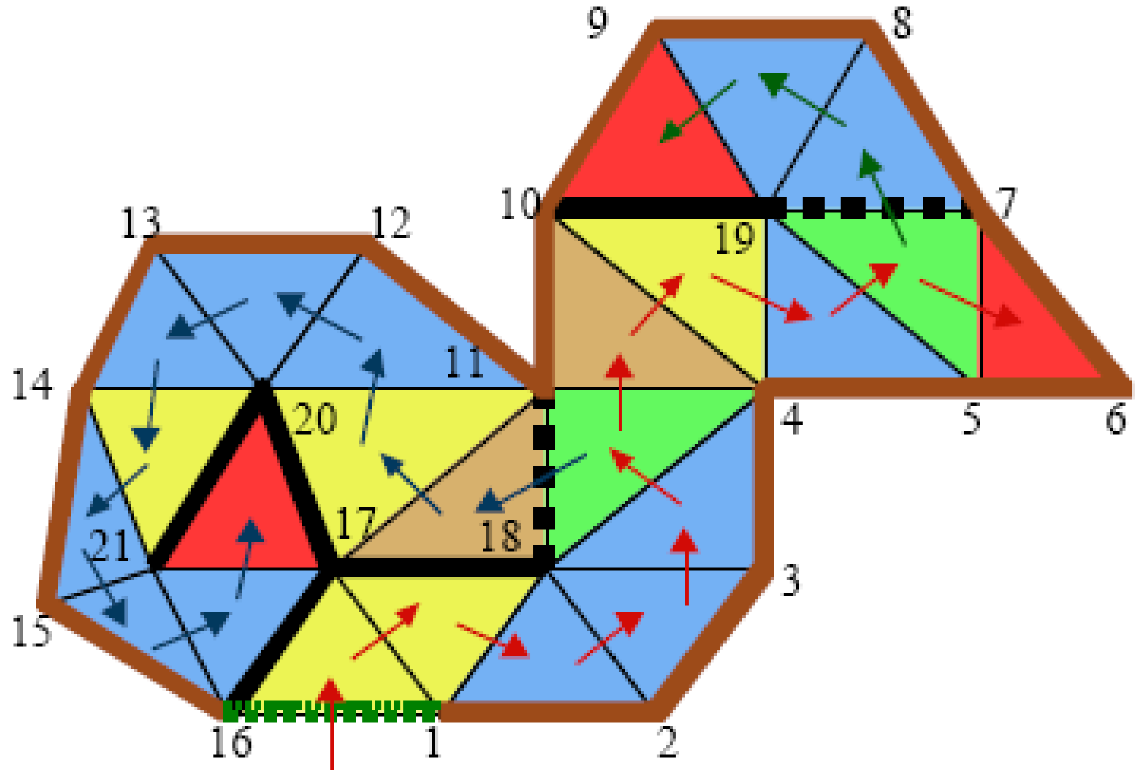

Add one edge
C
17
15
16
1
2
17
18
15
16
1
2

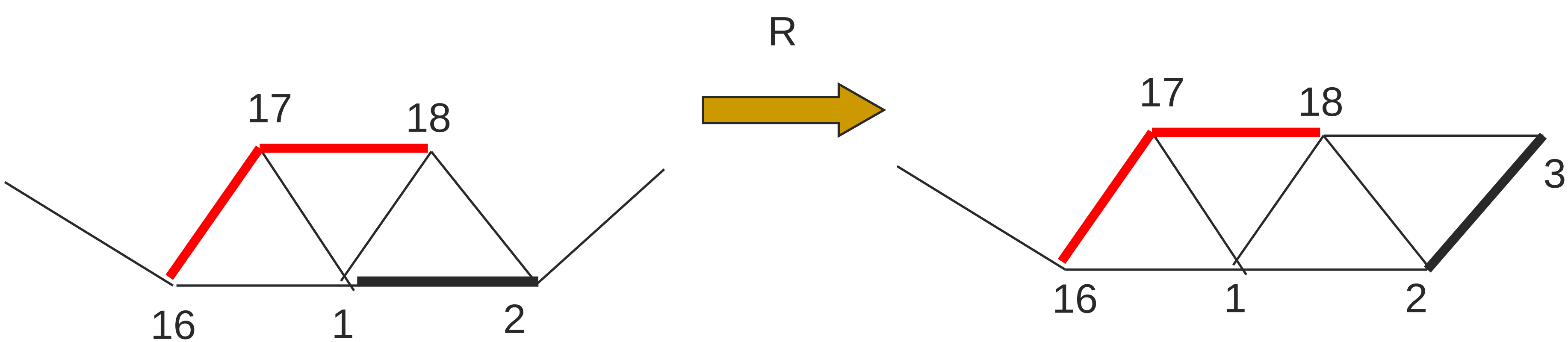


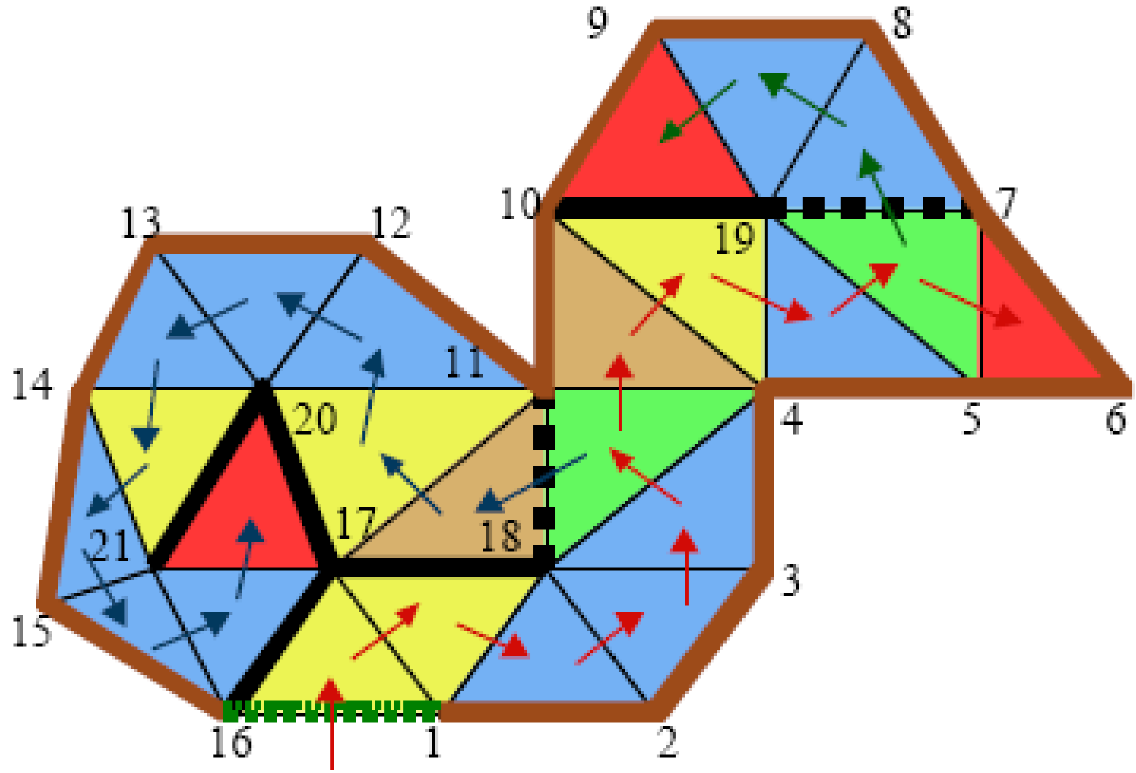
9
8
13
12
10
7
19
11
14
20
4
5
6
17
18
21
3
15
16
1
2

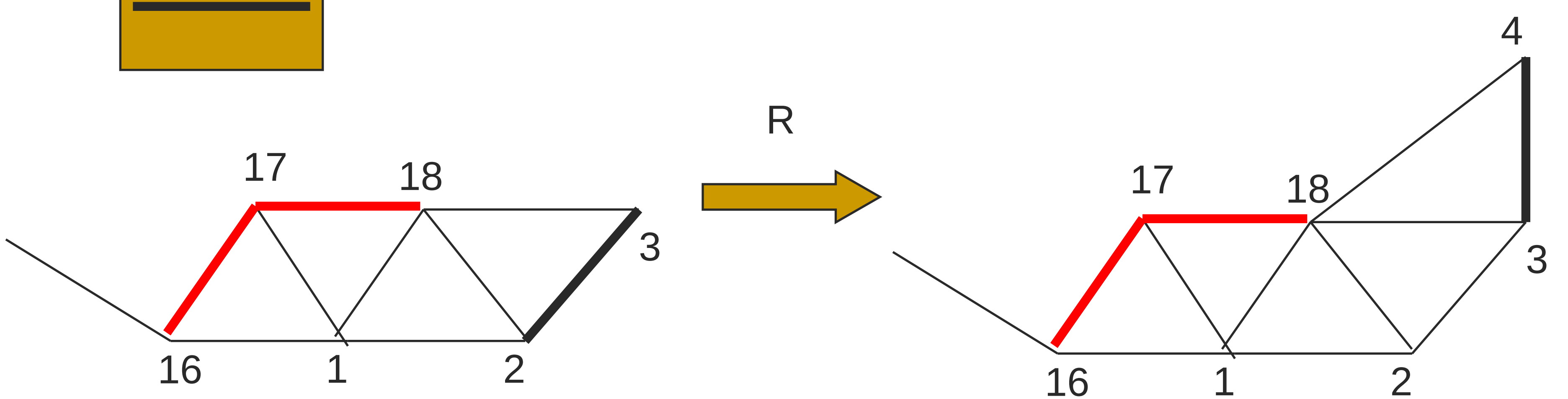


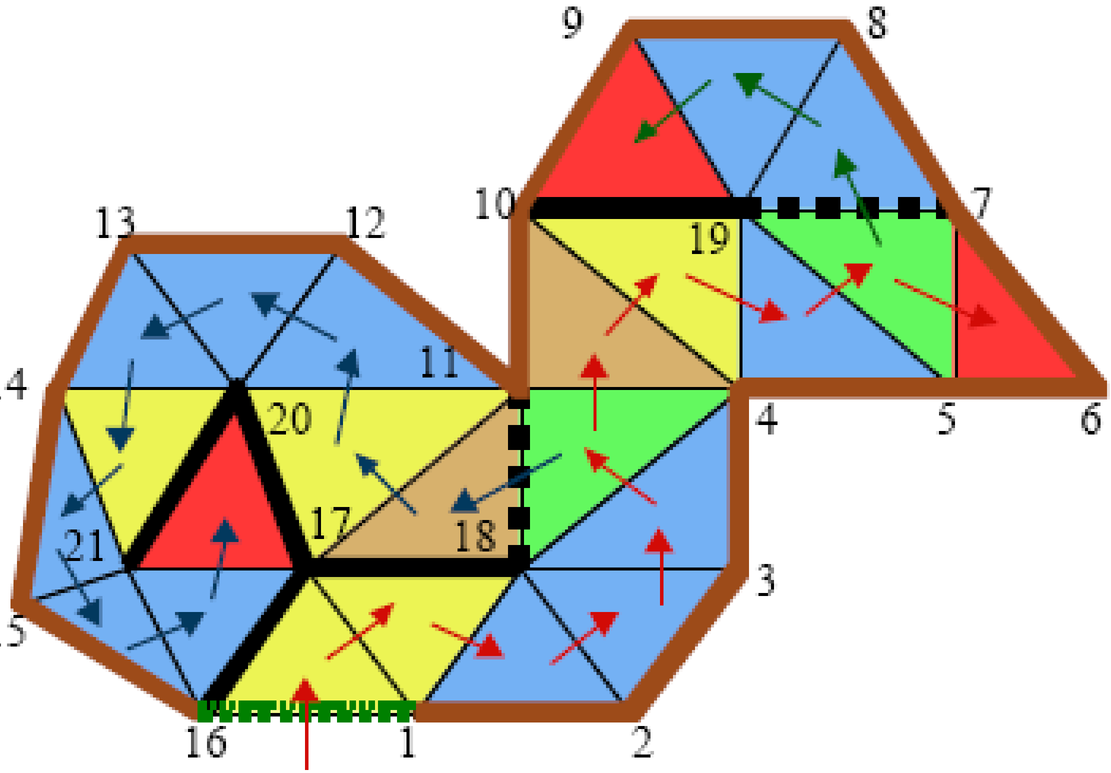
17
18
16
1
2
R
17
18
16
1
2

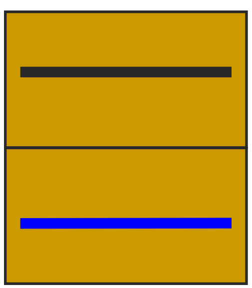

9
8
10
7
13
12
19
11
14
4
5
6
20
17
18
21
3
15
16
1
2

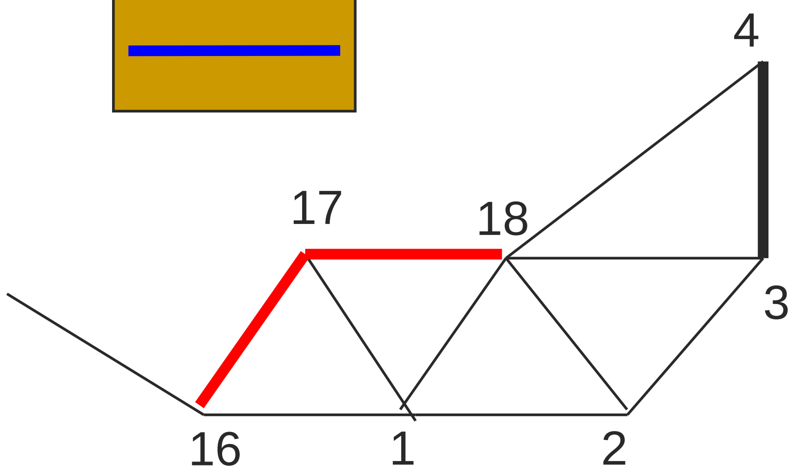

17
18
16
1
2
R
17
18
3
16
1
2

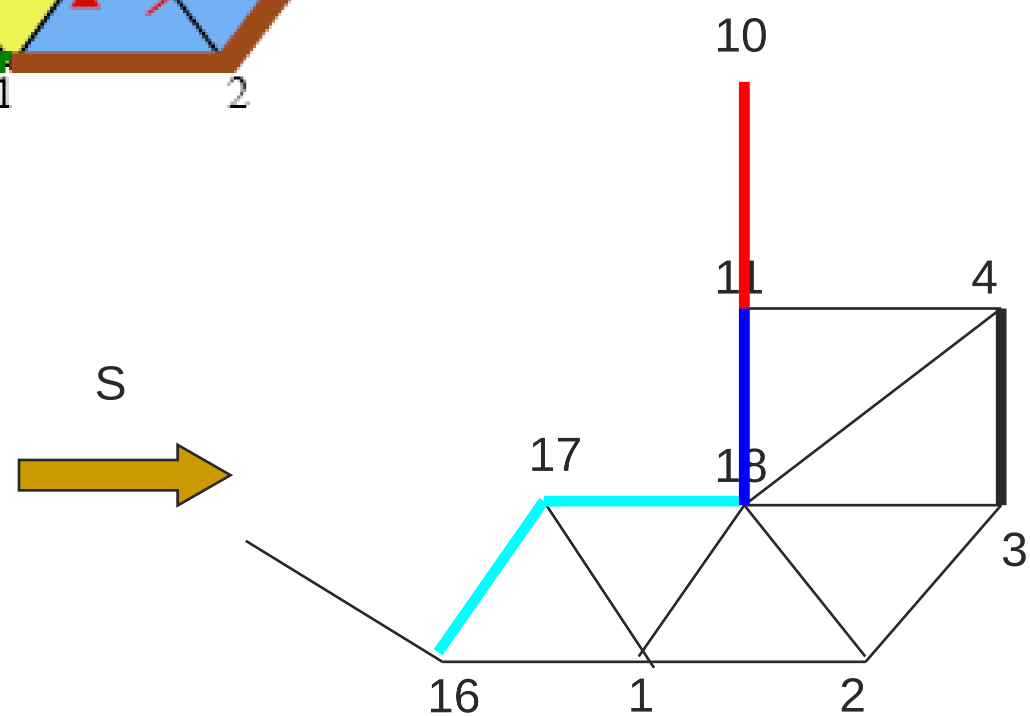

9
8
10
7
13
12
19
11
14
4
5
6
20
17
18
21
3
15
16
1
2

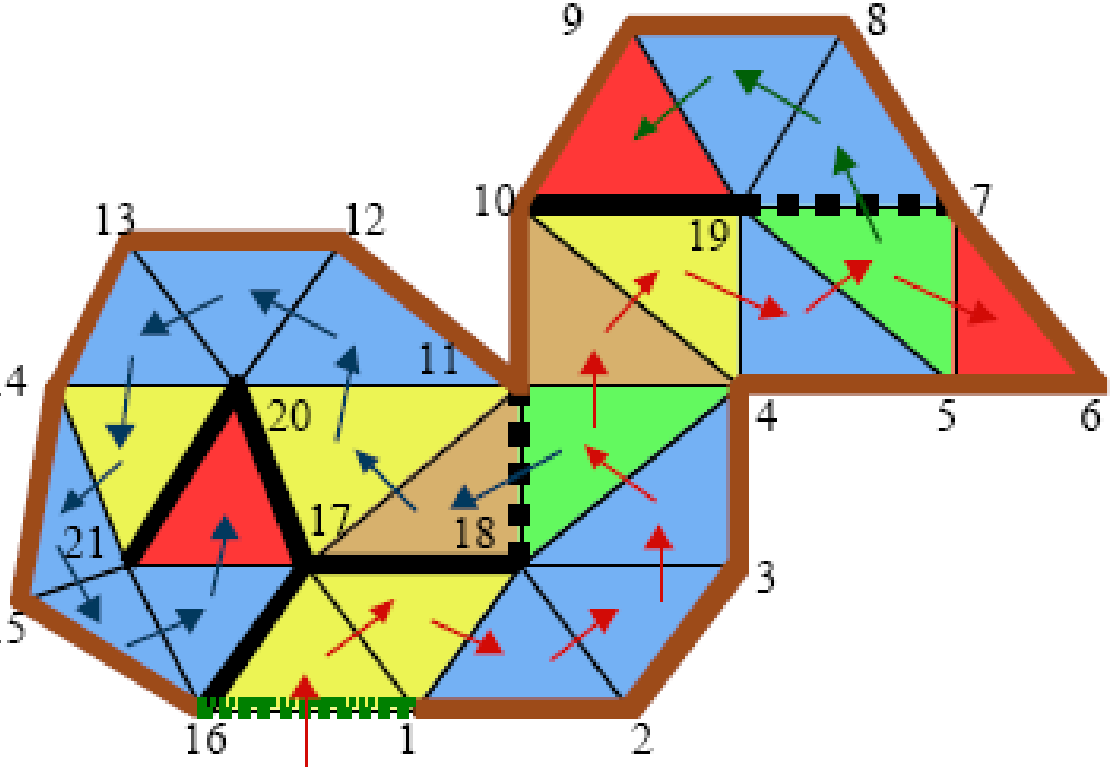

17
18
3
16
1
2
R
4
17
18
3
16
1
2

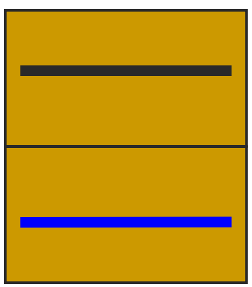

9
8
10
7
13
12
19
11
14
4
5
6
20
17
18
21
3
15
16
1
2

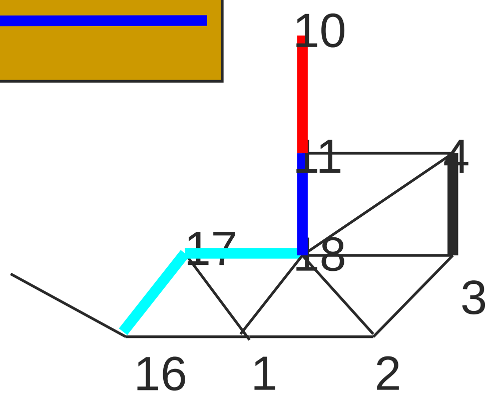


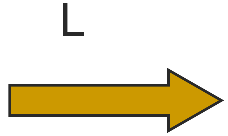

4
17
18
3
16
1
2

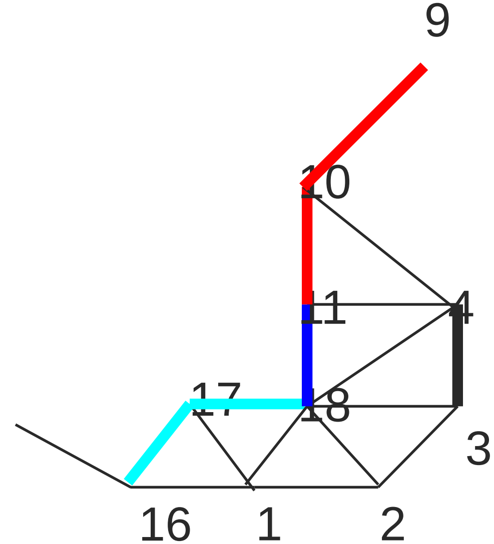

S
10
11
4
17
18
3
16
1
2

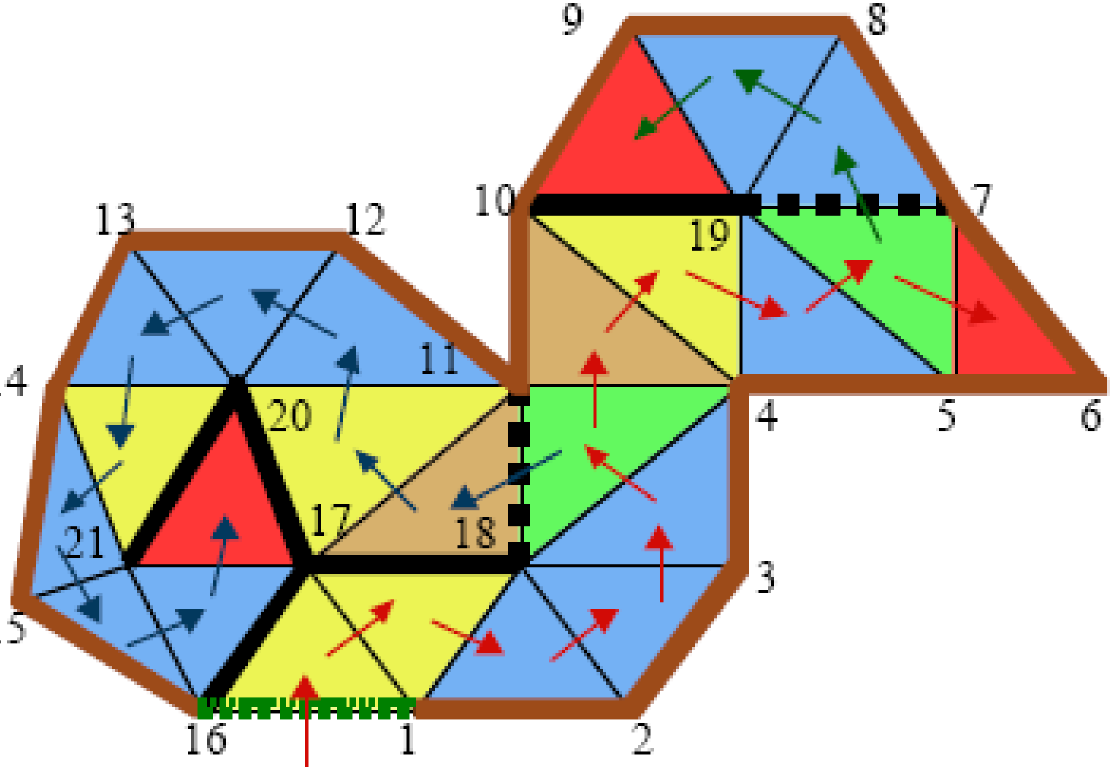

9
8
10
7
13
12
19
11
14
4
5
6
20
17
18
21
3
15
16
1
2

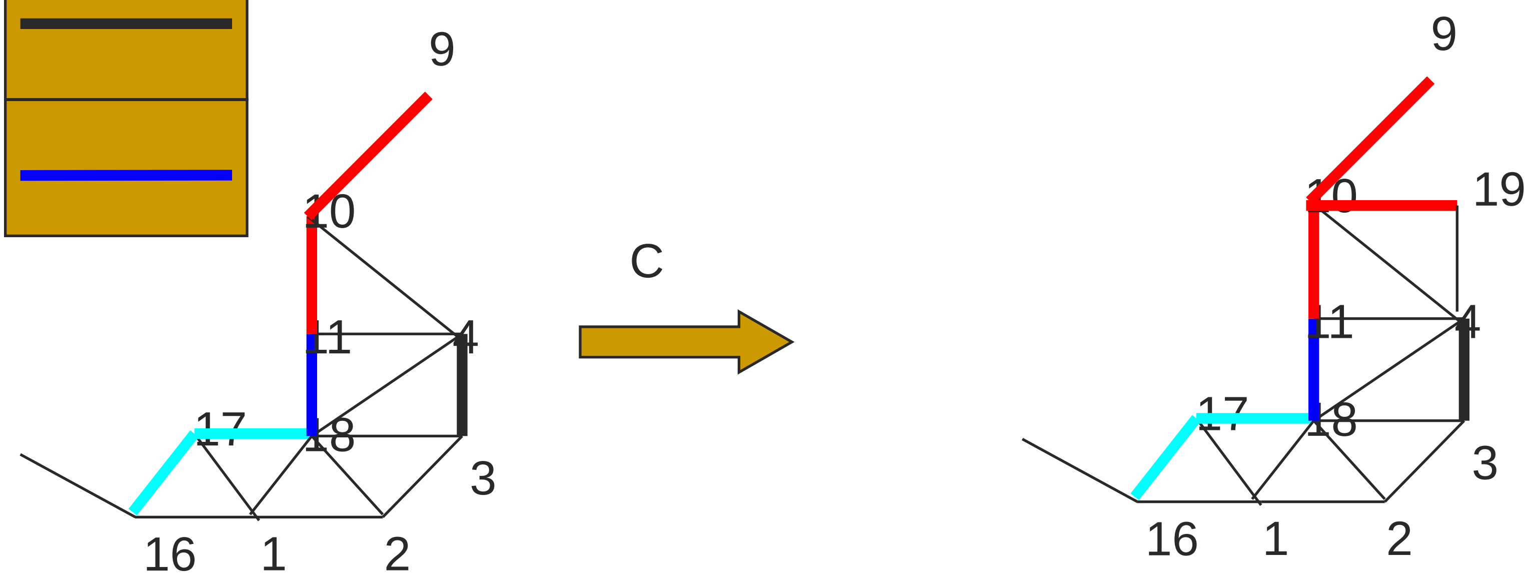


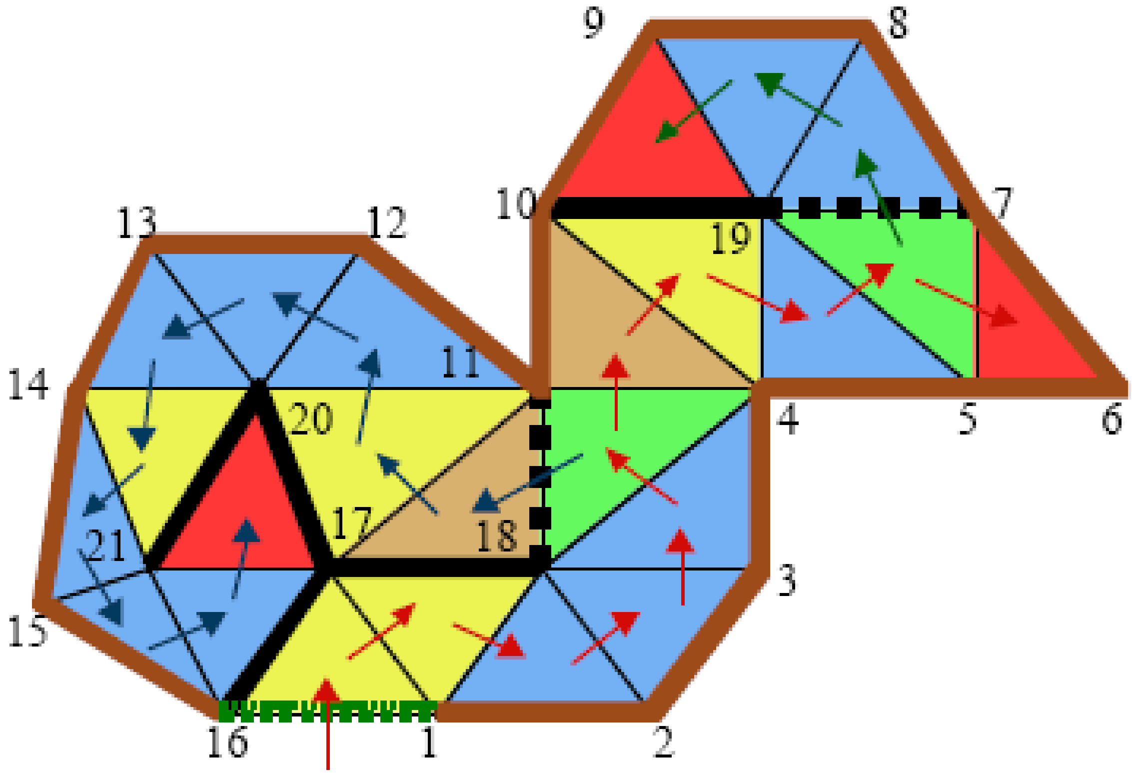

10
11
4
17
18
3
16
1
2

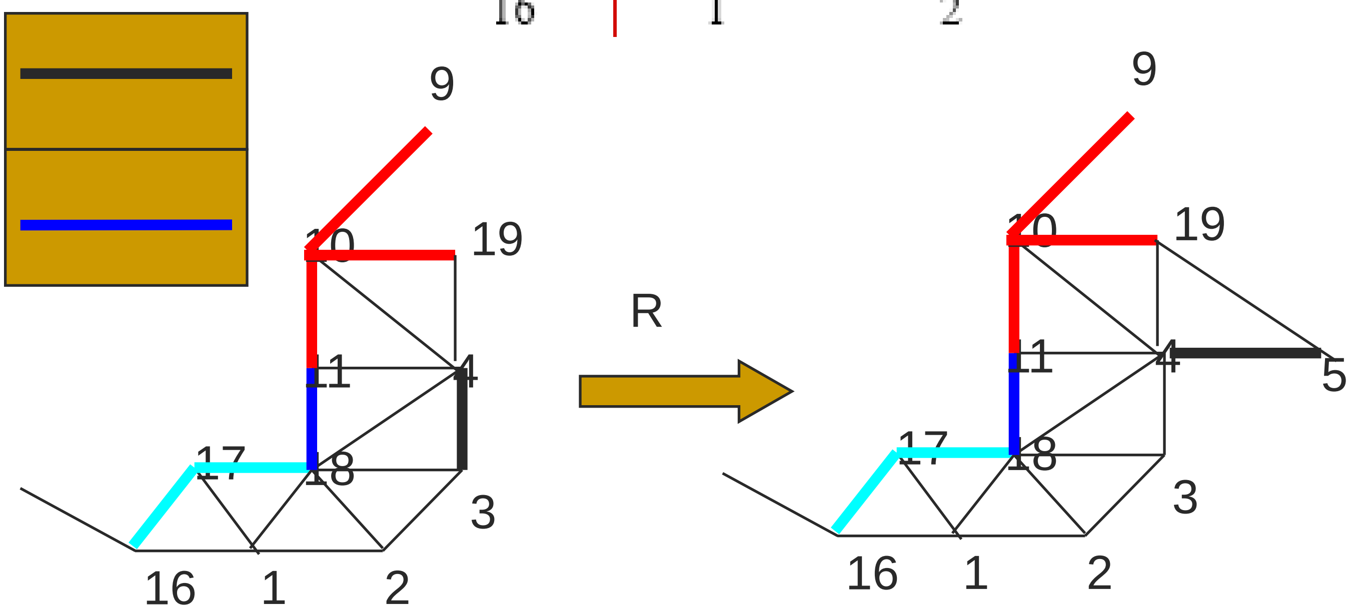

L

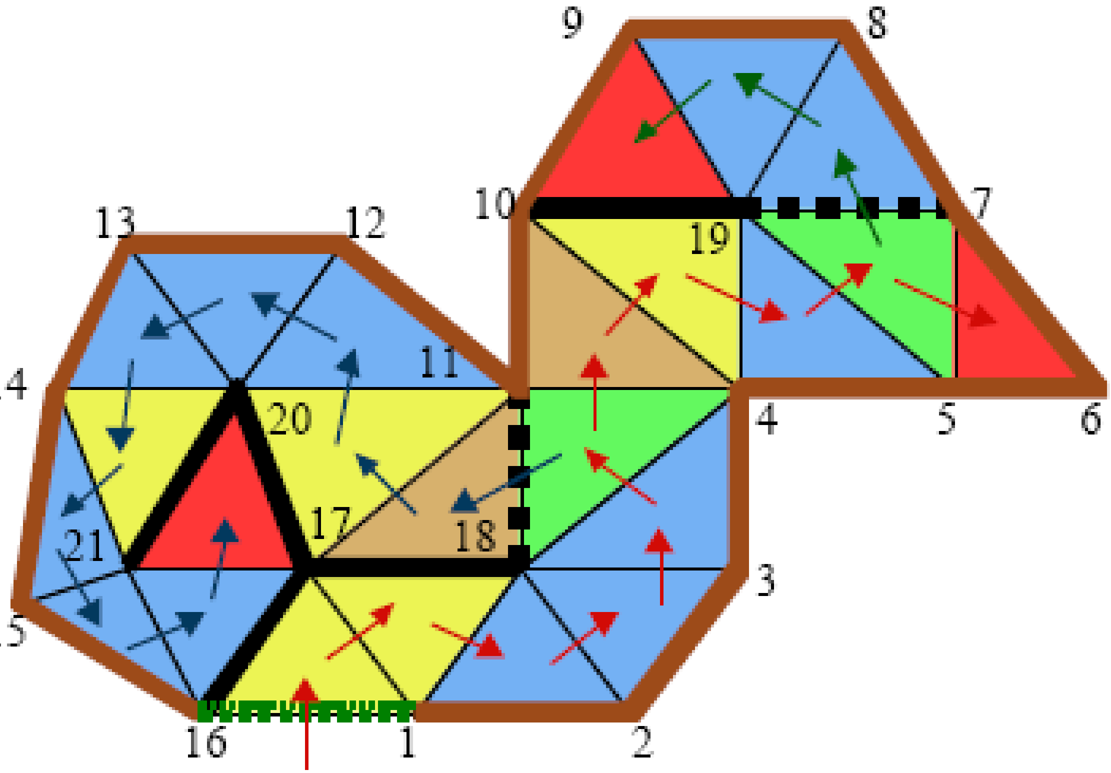

9
10
11
4
17
18
3
16
1
2

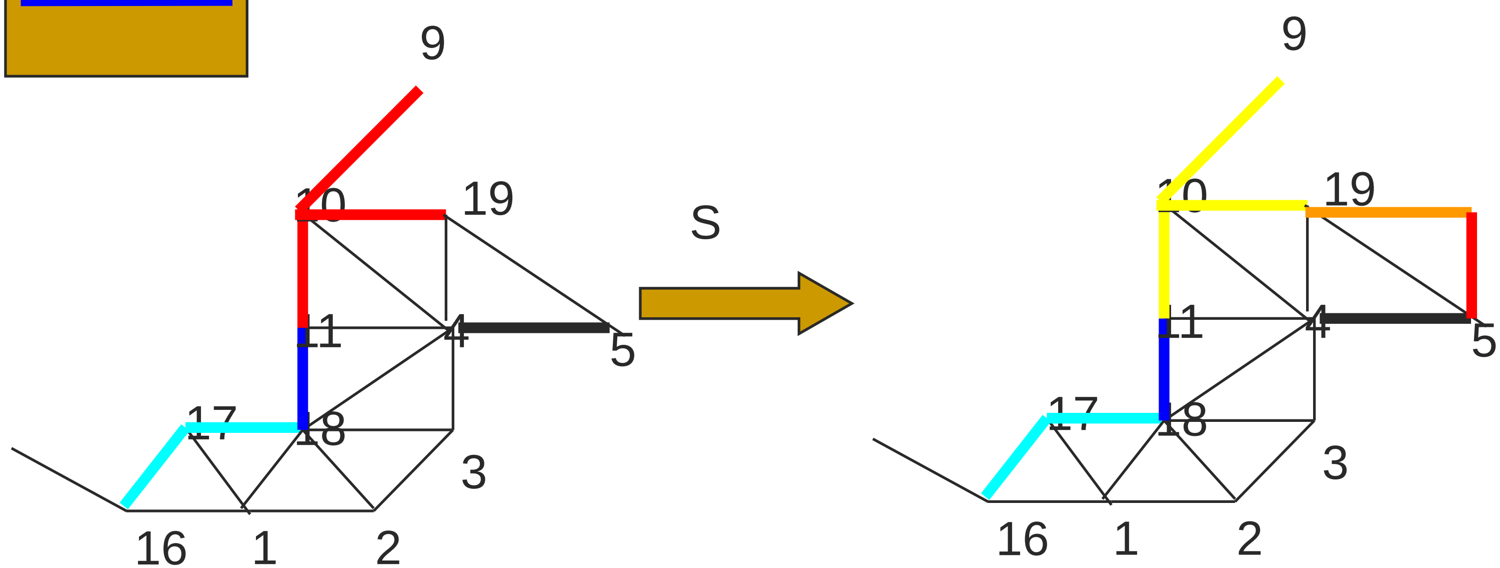

9
8
10
7
19
13
12
11
14
20
4
5
6
17
18
21
3
15
16
1
2

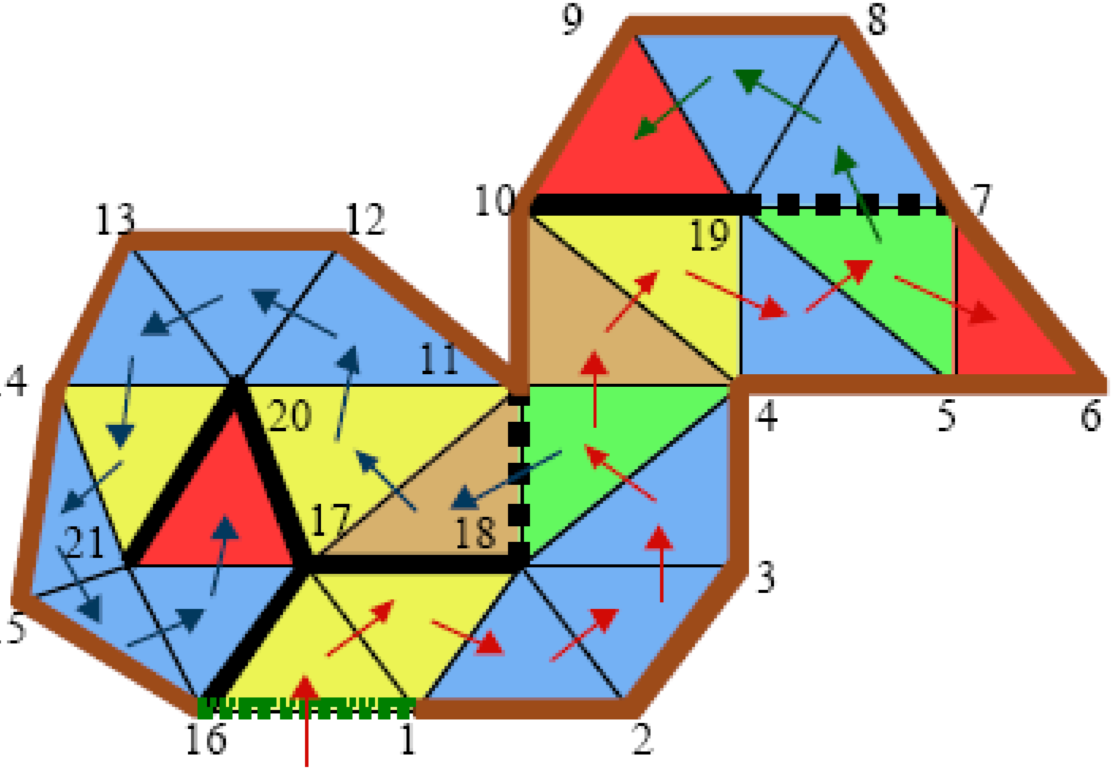

9
10
11
4
17
18
3
16
1
2
C
9
10
19
11
4
17
18
3
16
1
2

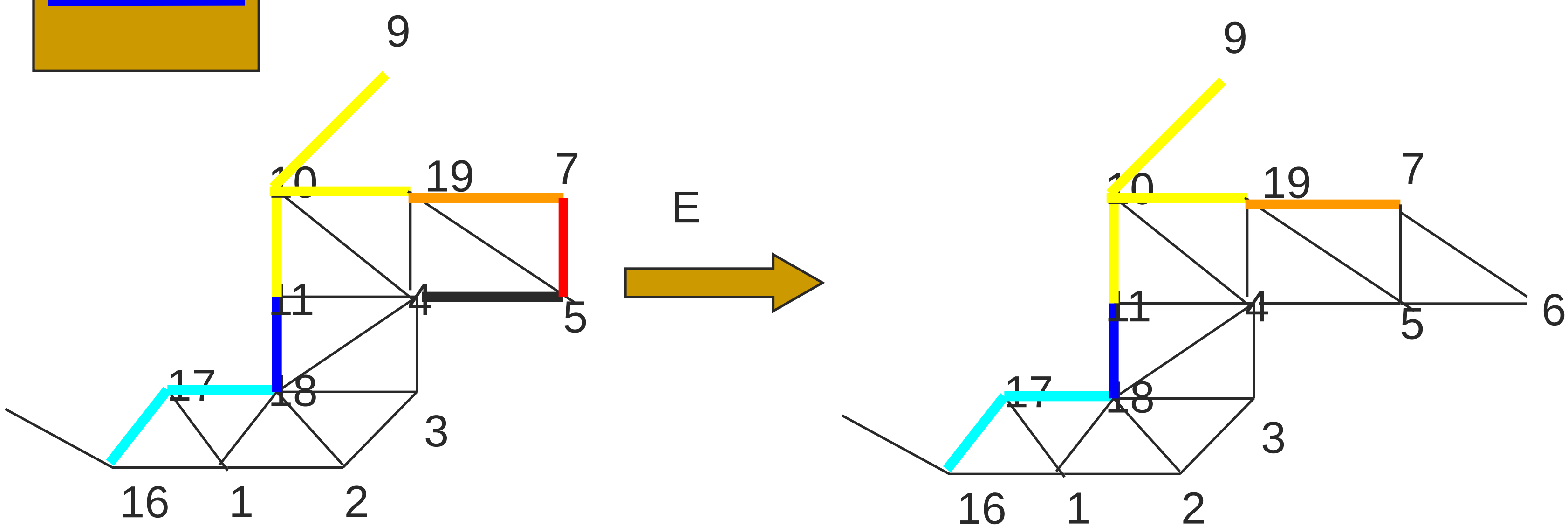

9
8
13
12
10
7
19
11
14
20
4
5
6
17
18
21
3
15
16
1
2


9
10
19
11
4
17
18
3
16
1
2
R
9
10
19
11
4
5
17
18
3
16
1
2


9
8
10
7
19
13
12
11
14
20
4
5
6
17
18
21
3
15
16
1
2


9
10
19
11
4
5
17
18
3
16
1
2
S
9
10
19
11
4
5
17
18
3
16
1
2


9
8
10
7
19
13
12
11
14
20
4
5
6
17
18
21
3
15
16
1
2


9
10
19
7
11
4
5
17
18
3
16
1
2
E
9
10
19
7
11
4
5
6
17
18
3
16
1
2

Face Fixer

Face Fixer

- Can handle arbitrary polygon meshes.
  - Quadrangular meshes can be compressed more efficient than their triangulated counterparts.

- Take properties and structures into account.

Face Fixer

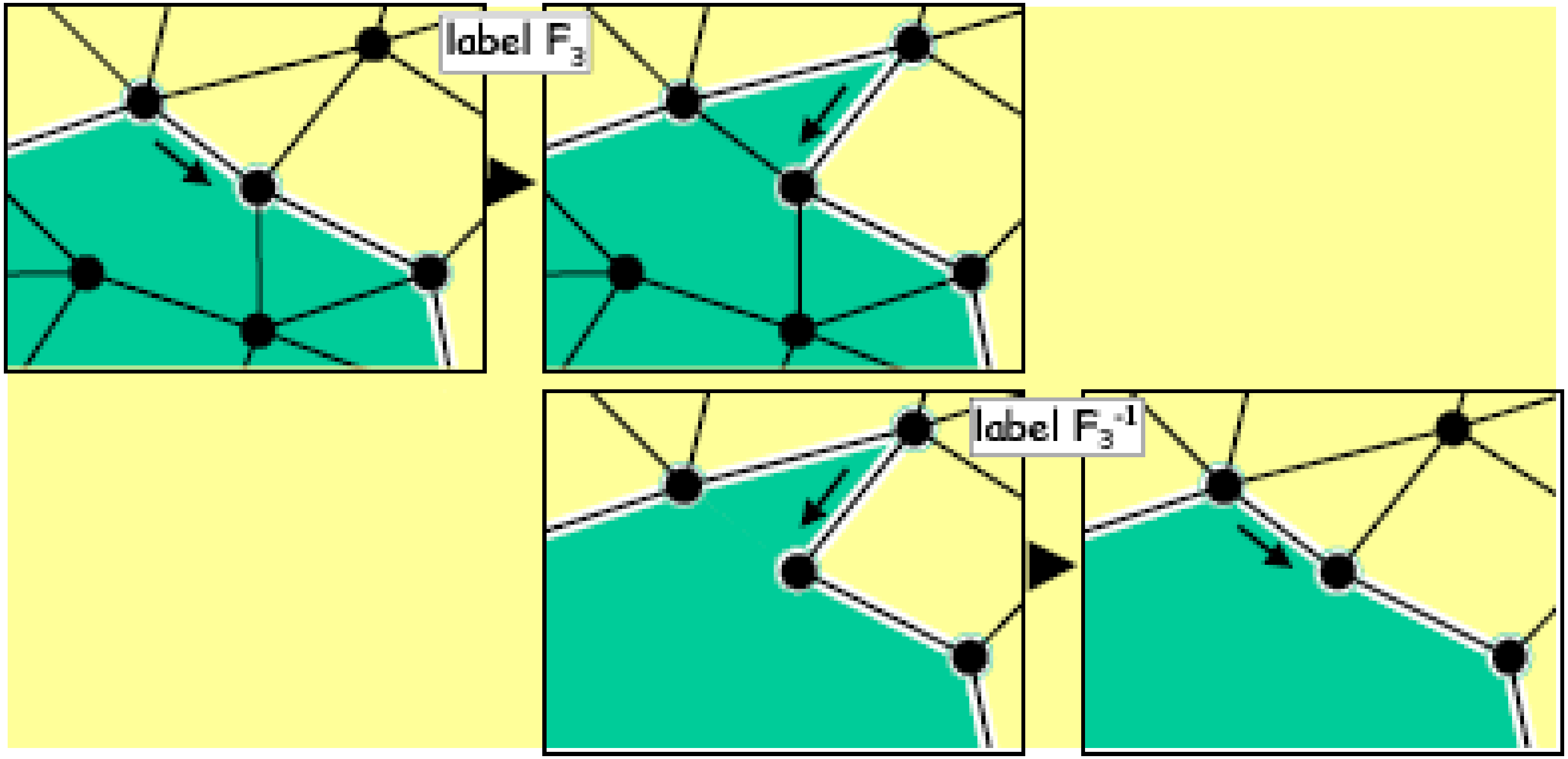
- Instead of take faces off, Face Fixer method take edge off.

- The total number of operations equal to total number of edges.

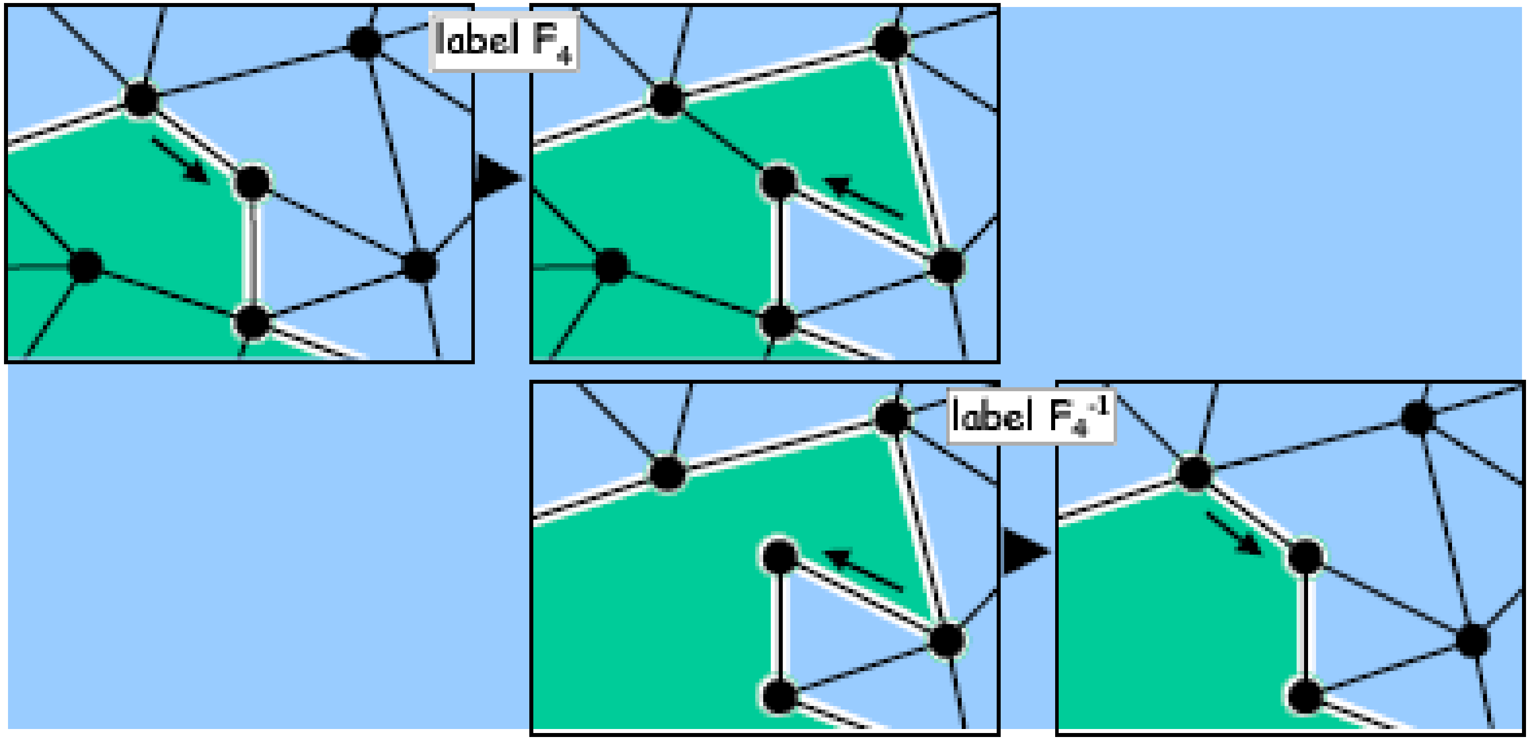
Compression

- Variable
  - f : face count (0)
  - h : hole count (0)
  - v : vertex count (0)
  - e : edge count (0)
  - h : handle count (0)

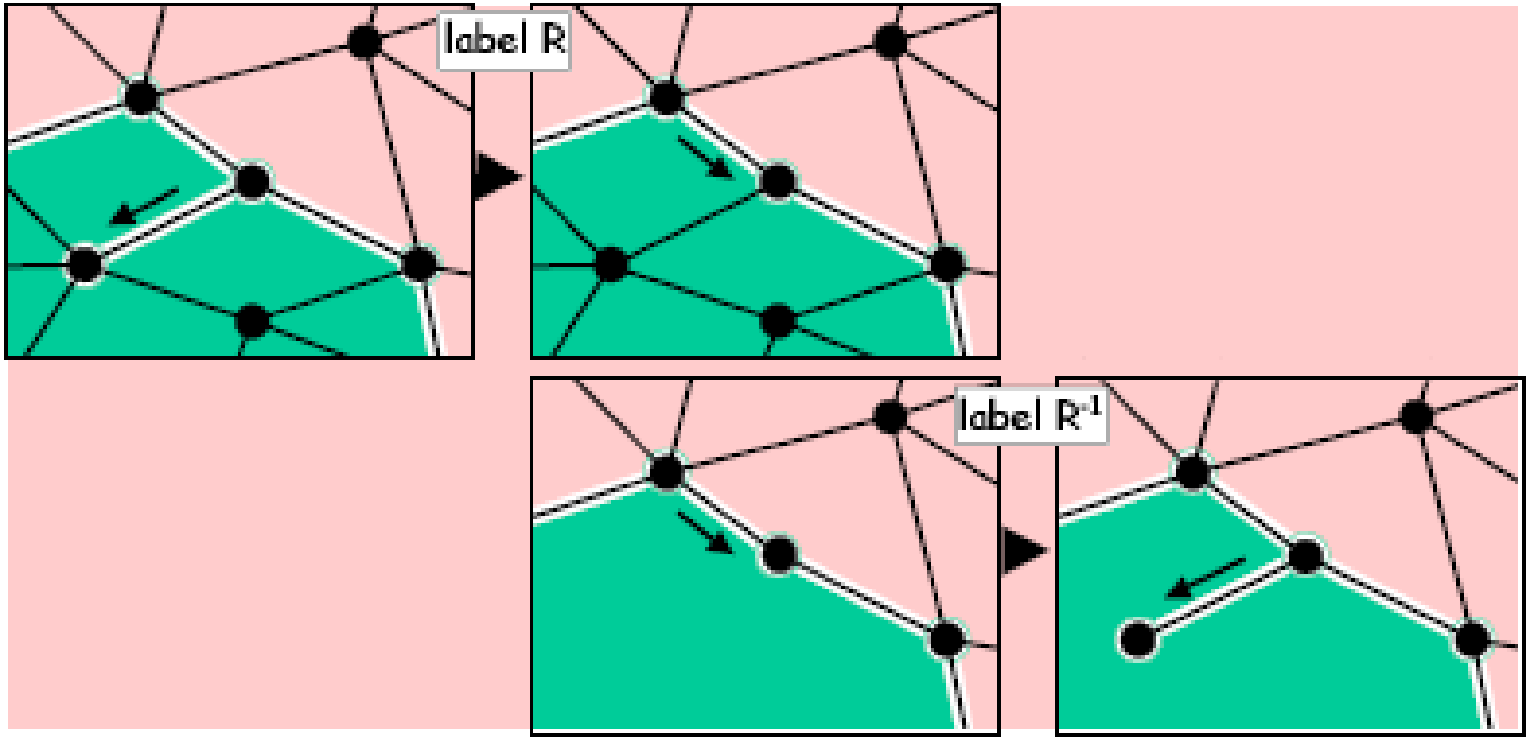
Case F3



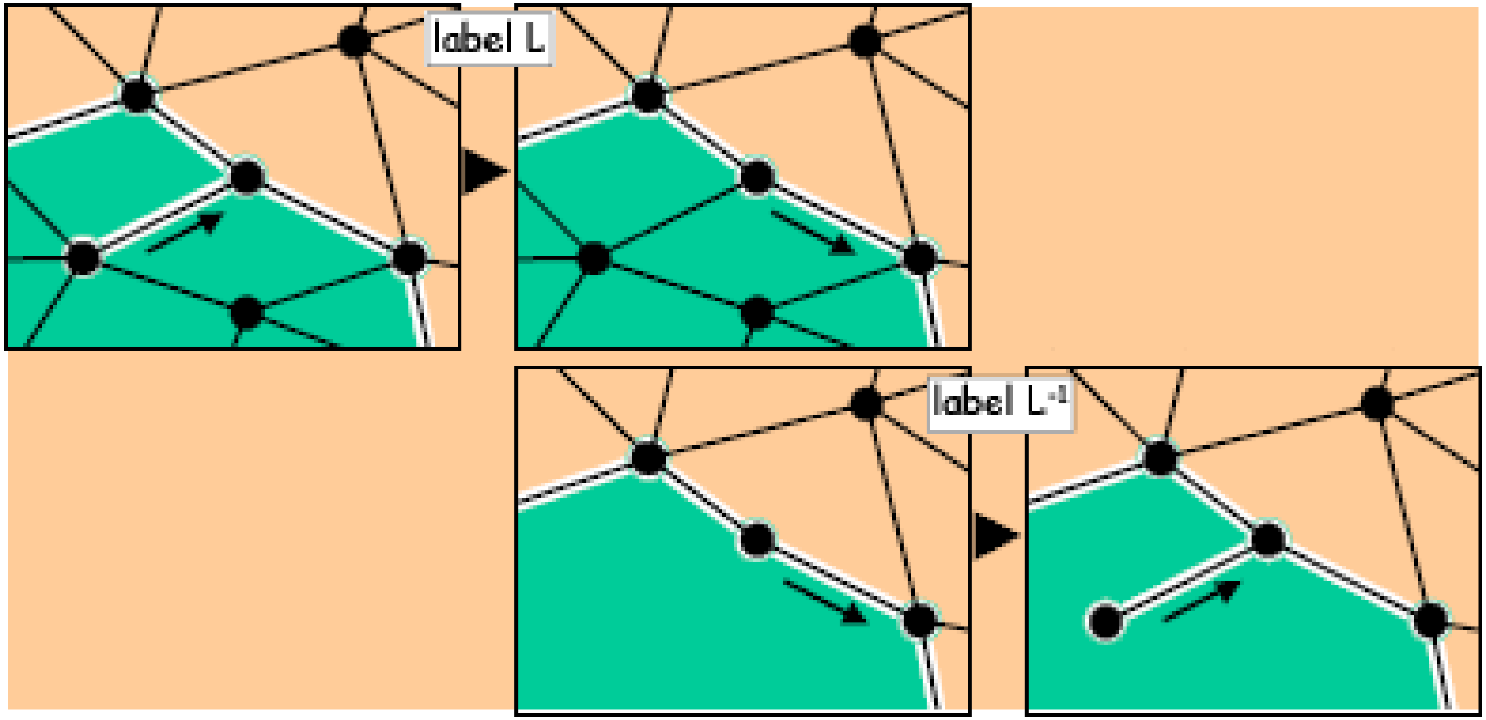
Case F4



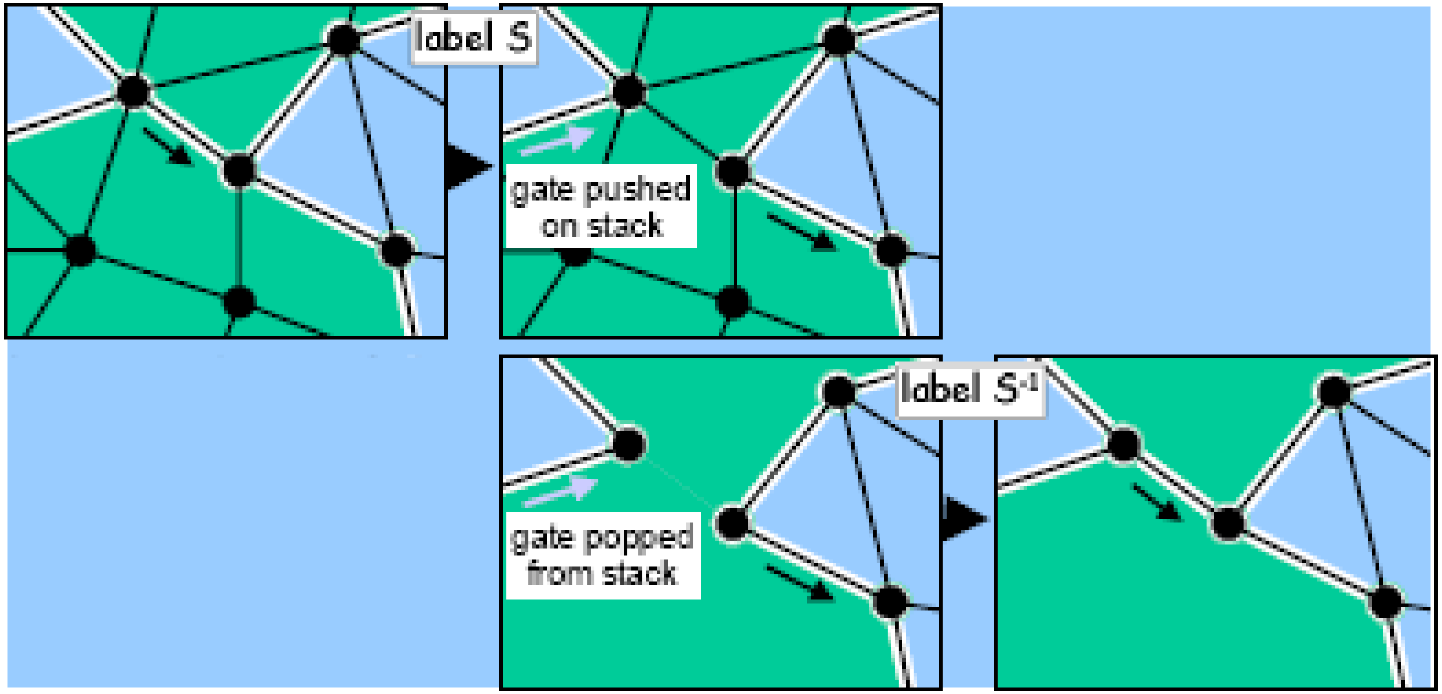
Case R



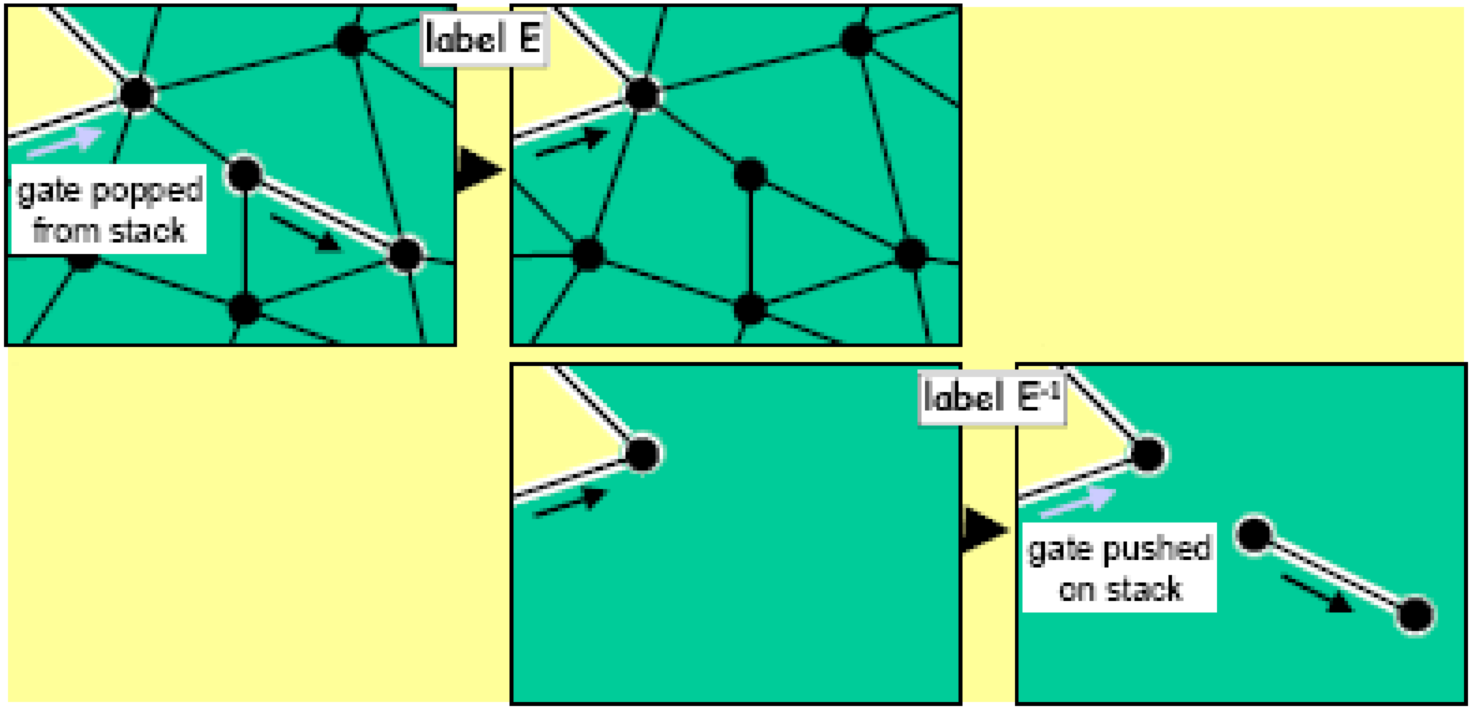
Case L



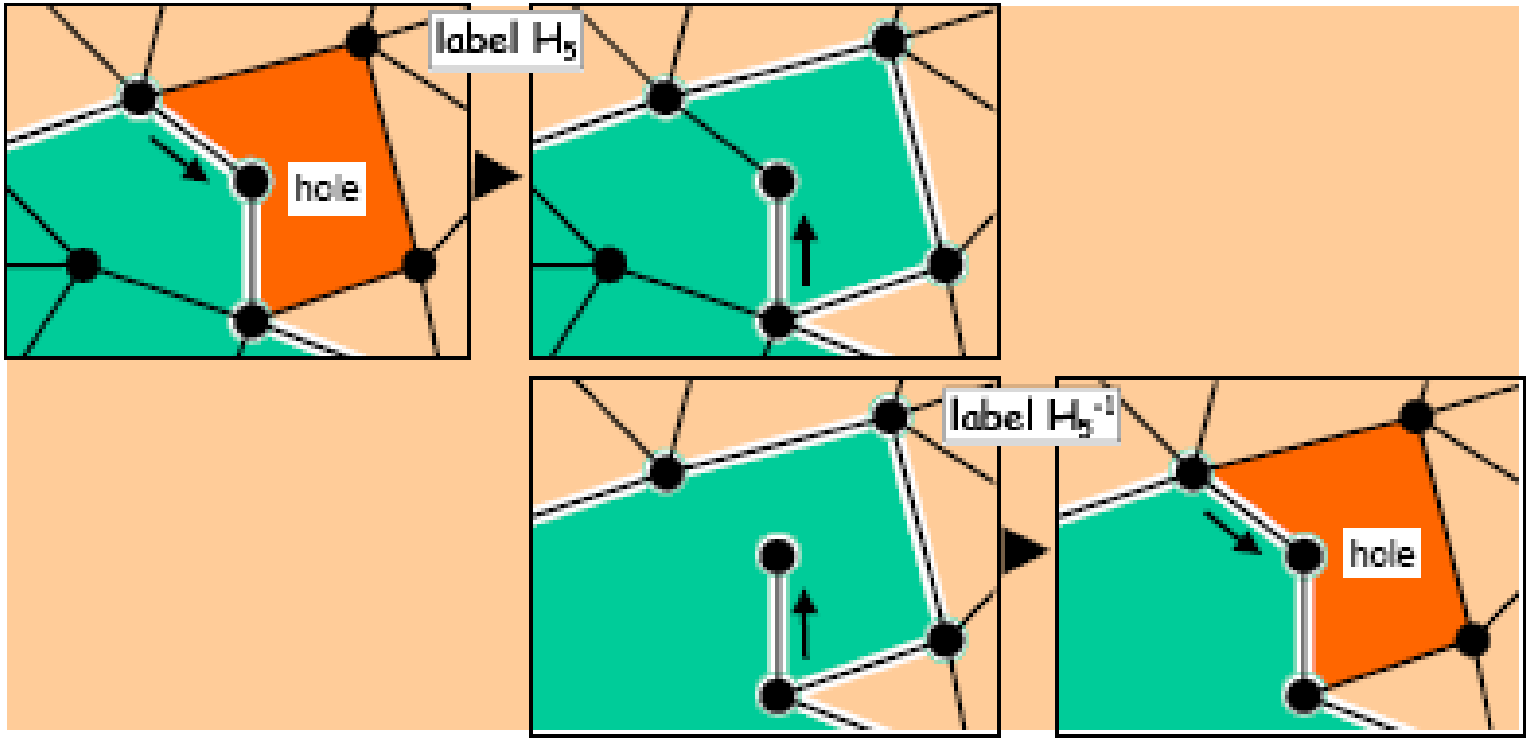
Case S



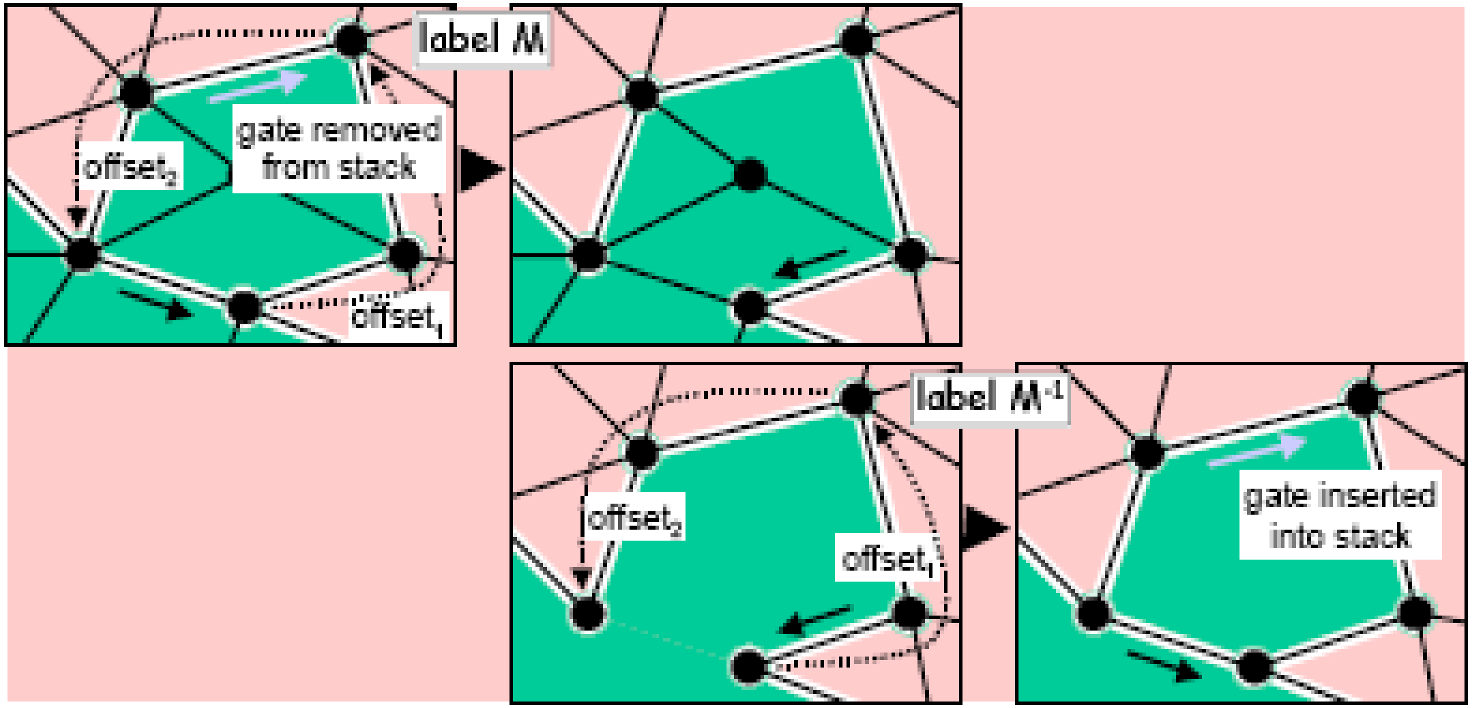
Case E



Case H5



Case M


label M
gate removed
from stack
offset$_2$
offset$_1$
label M^{-1}
offset$_2$
offset$_1$
gate inserted
into stack