Fast algorithm to split and reconstruct triangular meshes

GÁBOR FÁBIÁN, ELTE-IK, DEPARTMENT OF NUMERICAL ANALYSIS

10TH JOINT CONFERENCE ON MACS, CLUJ-NAPOCA, 2014. 05. 23.

Triangular meshes

Vertexstream

$$V: [1..m] \rightarrow \mathbb{R}^3$$

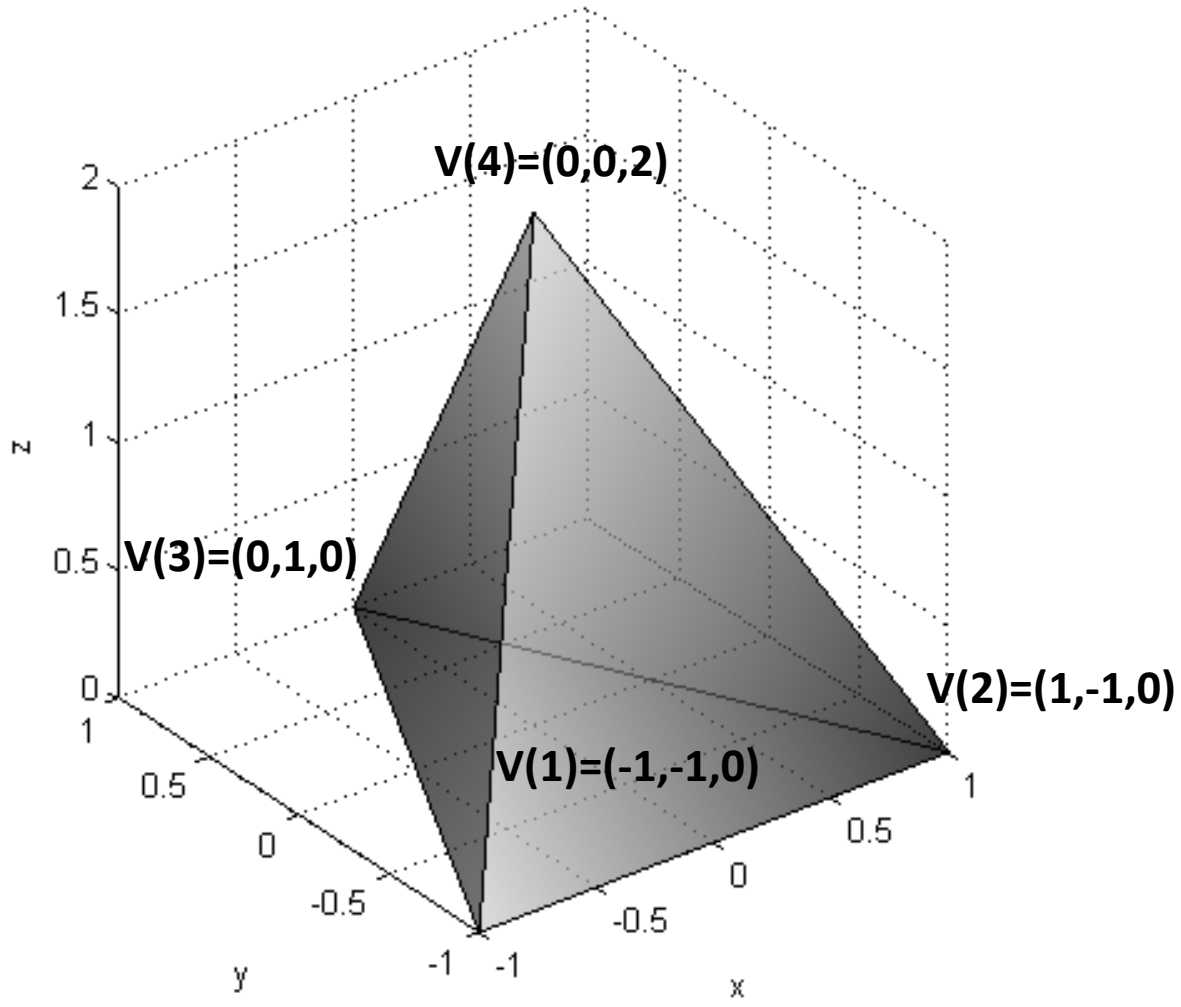
Indexstream

$$F: [1..n] \rightarrow [1..m]^3$$

Streams can be represented by matrices

$$V \in \mathbb{R}^{m \times 3}$$

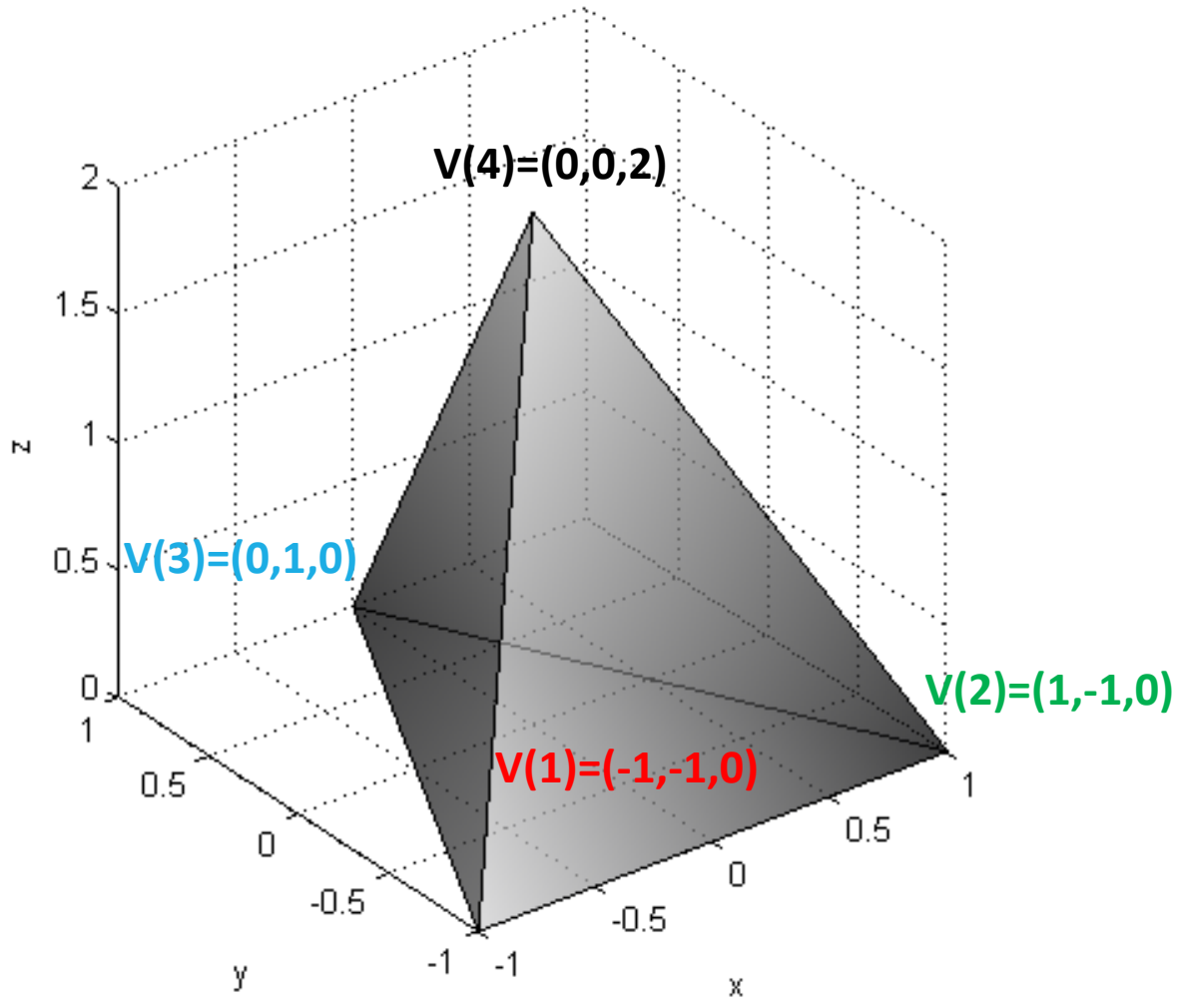
$$F \in \mathbb{N}^{n \times 3}$$



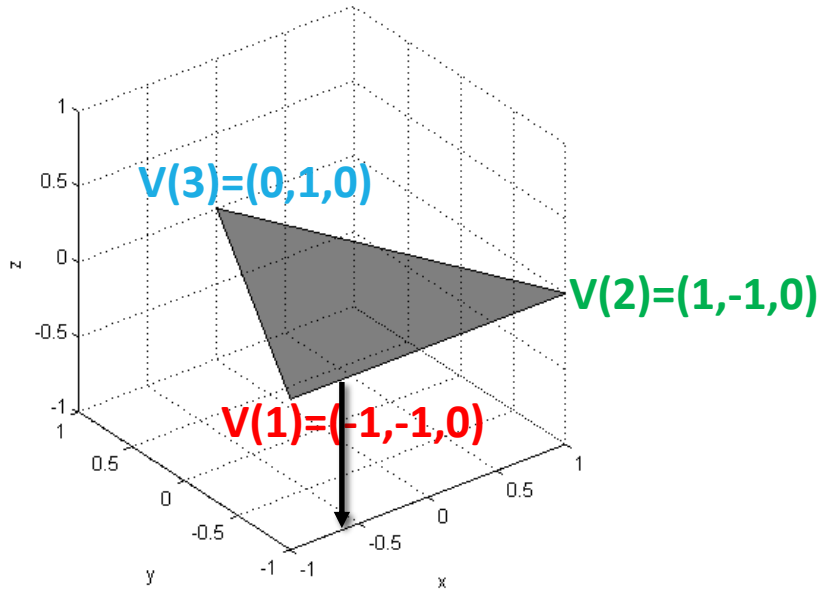
Triangular meshes

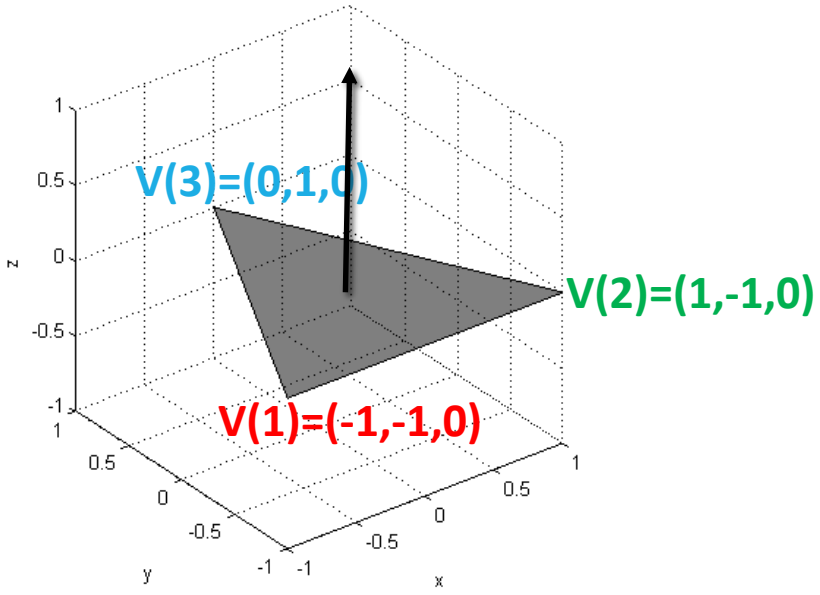
$$V = \begin{bmatrix} -1 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$F = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 3 & 1 & 4 \\ 2 & 3 & 4 \end{bmatrix}$$



Normal vectors





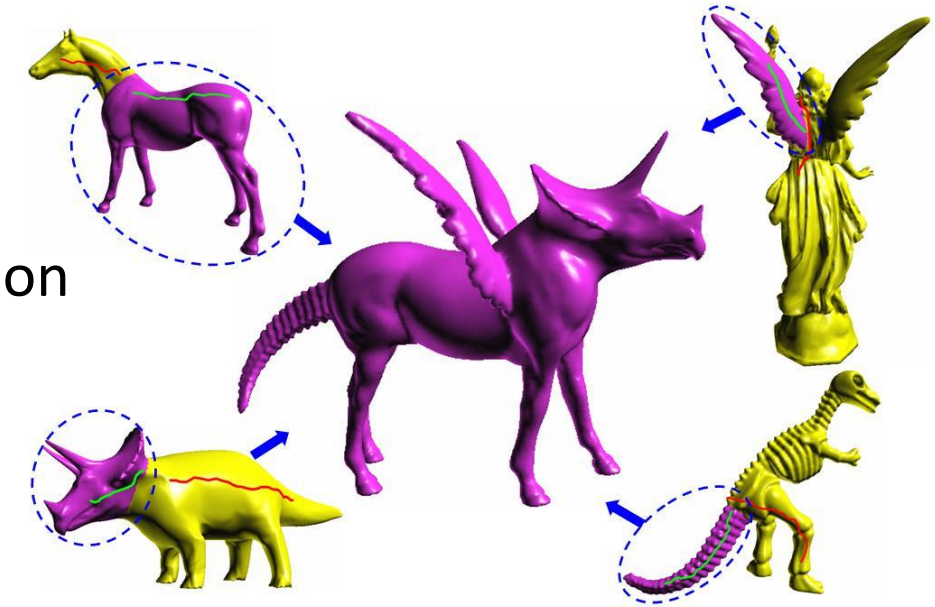
CCW order (1,2,3) (3,1,2) (2,3,1)

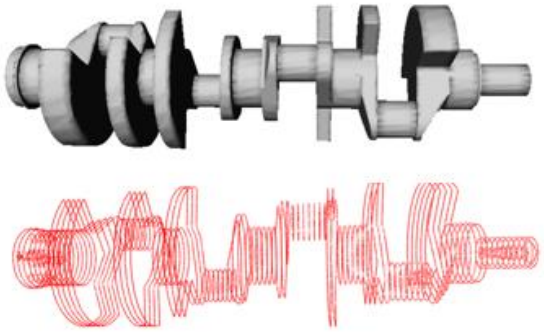
(2,1,3) (3,2,1) (1,3,2) CW order

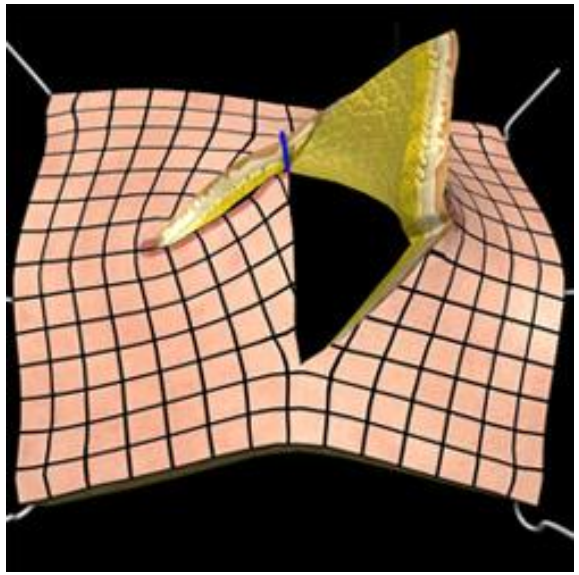
Remark: Acyclic permutation reverses the normal vectors

Mesh cutting/slicing/splitting

- CAD
- Surgery simulation
- Topologycal analysis
- Topologycal reconstruction
- 3D printing
- Games







http://pages.cs.wisc.edu/~sifakis/images/MMVR09_Local_Flaps.jpg
http://taylorwang.files.wordpress.com/2012/04/sum4.jpg
http://www.math.zju.edu.cn/ligangliu/CAGD/Projects/SketchBasedMeshCutting/EasyMeshCutting/Images/cut-and-paste.JPG

Mesh splitting

Input

- (V, F) triangular mesh
- P plane defined by (n, p)
- (V, F) defines a non-self-intersecting connected compact 2-manifold*
- There is no vertex in V lying on the cutting plane**

Output

- (V_i, F_i) triangular meshes
- (V_i, F_i) defines a non-self-intersecting connected compact 2-manifold

Consequences

If a triangle mesh is compact 2-manifold, then each edge connects *exactly two* different triangles

If there is no vertex lying on P, then every intersecting triangles need to be divided into *three parts*

Outline of algorithm

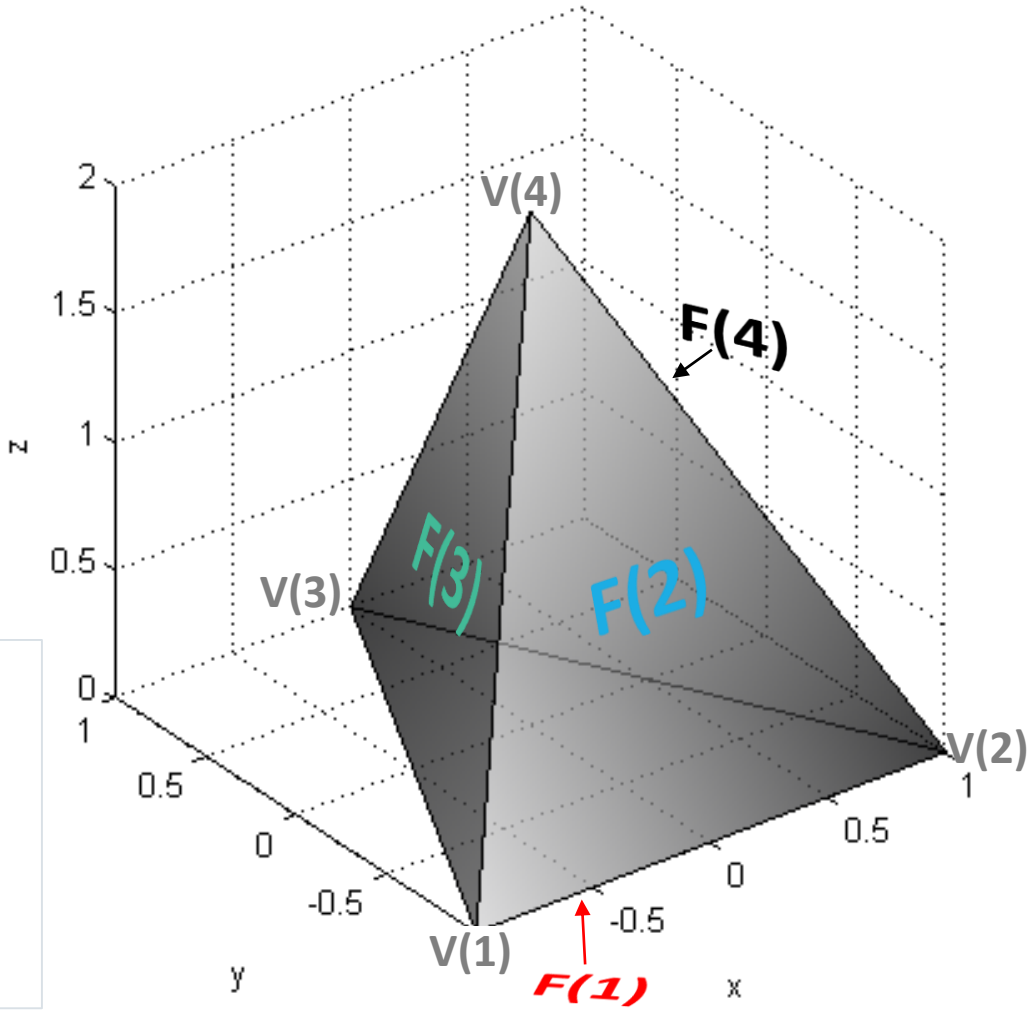
- **Generating adjacencies**
- Sorting vertices, faces and adjacencies
- Creating new vertices and faces
- Capping holes
- Decomposition

Adjacency stream

$$F = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 3 & 1 & 4 \\ 2 & 3 & 4 \end{bmatrix} \longrightarrow A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 4 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 4 \end{bmatrix}$$

$A(i,j) = k$ if and only if $F(k)$ is adjacent to $F(i)$ along the edge between the vertices $V\big(F(i,j)\big)$ and $V(F(i,j\dot{+}1))$.

Example:
$A(3,2) = 2$, because the 3rd triangle is defined by vertices (3,1,4), its 2nd edge is (1,4). The (1,4) edge connects exactly two triangles, the 3rd and the 2nd. For the same reason $A(2,3) = 3$ considering that 3rd edge of the 2nd triangle is (4,1).



Outline of algorithm

- Generating adjacencies
- **Sorting vertices, faces and adjacencies**
- Creating new vertices and faces
- Capping holes
- Decomposition

V =

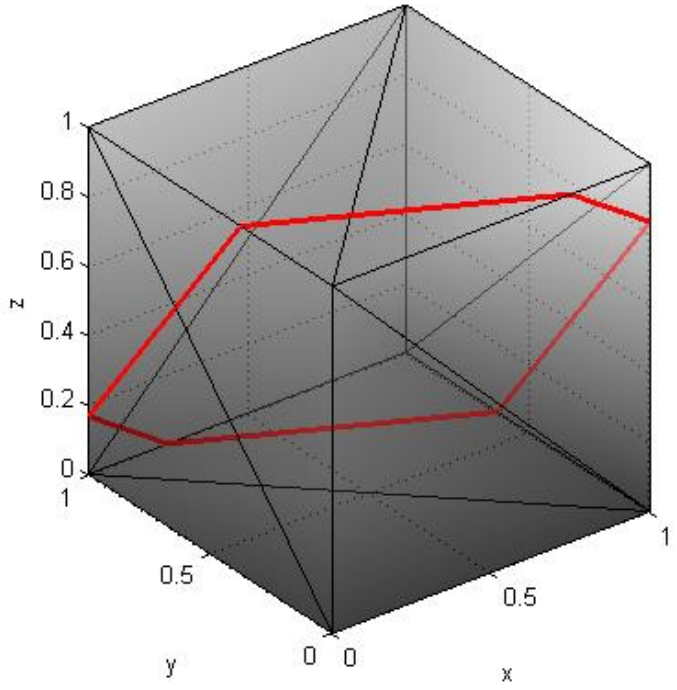
0	0	1
0	0	0
1	0	0
1	0	1
0	1	1
0	1	0
1	1	0
1	1	1

F =

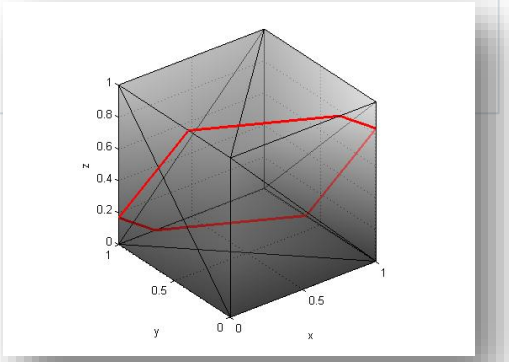
2	1	5
2	5	6
6	7	3
6	3	2
7	8	4
7	4	3
1	4	8
1	8	5
1	2	3
1	3	4
8	7	6
8	6	5

A =

9	8	2
1	12	4
11	6	4
3	9	2
11	7	6
5	10	3
10	5	8
7	12	1
1	4	10
9	6	7
5	3	12
11	2	8



Sorting vertices

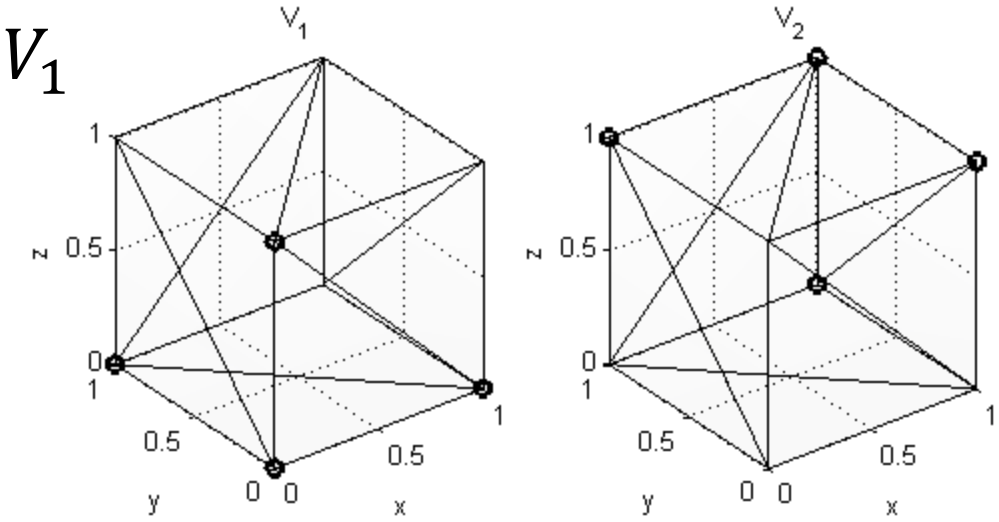


Let us define $S(k) \coloneqq \langle V(k) - p, n \rangle$
$S(k) > 0$ if $V(k)$ above the cutting plane
$S(k) < 0$ if $V(k)$ below the cutting plane
$S(k) \neq 0$, because there are no points lying on P

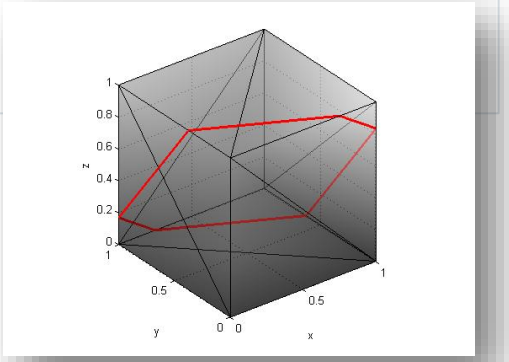
If $S(k) > 0$ then put $V(k)$ to V_2 else put it into V_1

If a vertex $V(l)$'s new index in V_1 or V_2 is k then

$$\pi_V(l) \coloneqq k$$



Sorting faces



If $S(F(i,j)) > 0$ $(j = 1,2,3)$ then put $F(i)$ to F_2
If $S(F(i,j)) < 0$ $(j = 1,2,3)$ then put $F(i)$ to F_1

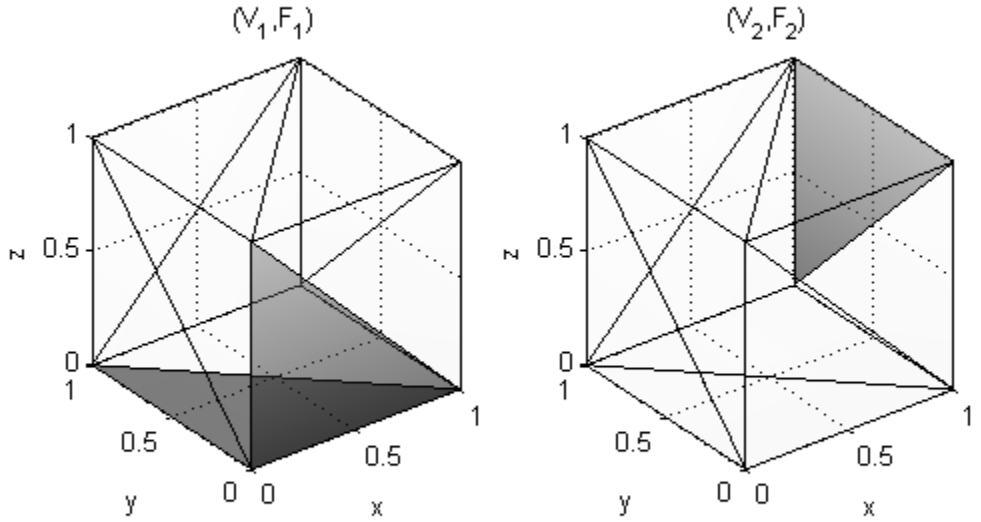
If none of the above, then $F(i)$ is intersecting face

If a face $F(l)$'s new index in F_1 or F_2 is k then

$$\pi_F(l) \coloneqq k$$

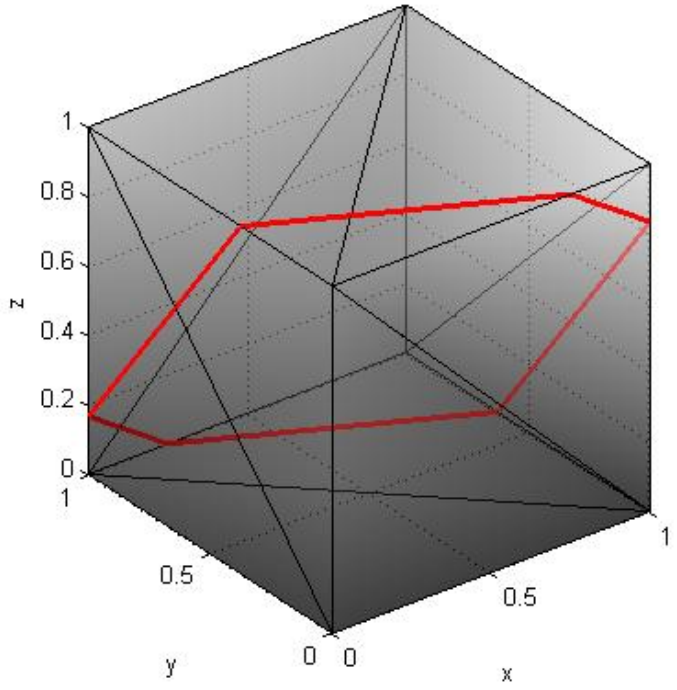
If $F(l)$ is intersecting then

$$\pi_F(l) \coloneqq 0$$



Outline of algorithm

- Generating adjacencies
- Sorting vertices, faces and adjacencies
- **Creating new vertices and faces**
- Capping holes
- Decomposition



V =

0	0	1
0	0	0
1	0	0
1	0	1
0	1	1
0	1	0
1	1	0
1	1	1

F =

2	1	5
2	5	6
6	7	3
6	3	2
7	8	4
7	4	3
1	4	8
1	8	5
1	2	3
1	3	4
8	7	6
8	6	5

A =

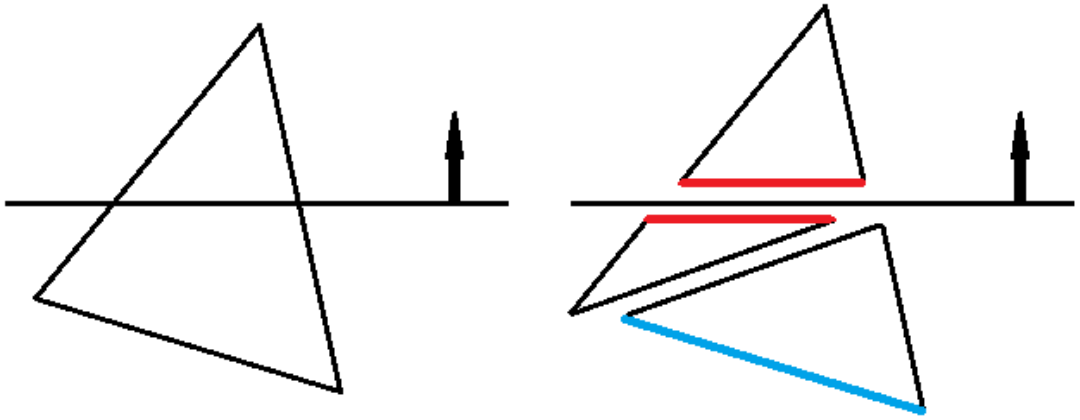
9	8	2
1	12	4
11	6	4
3	9	2
11	7	6
5	10	3
10	5	8
7	12	1
1	4	10
9	6	7
5	3	12
11	2	8

Labeling rules

Iteration

If we find an intersecting triangle we process it, after that we deal with the unique non-processed adjacent along an intersecting edge, while exists.

Let us suppose that we are currently processing $F(i)$ and the preceding triangles have been processed correctly, except adjacencies of **cutting edges** and **non-intersecting edges**



Rules

- The intersecting edge between $\mathrm{V}(F(i,j))$ and $\mathrm{V}(F(i,j\dot{+}1))$ processed in the previous step, and $\mathrm{V}\big(F(i,j)\big)$ is always above the cutting plane
- $F(i,j), F(i,j\dot{+}1)$ and $F(i,j\dot{+}2)$ both are the first vertex indices of some newly created triangle. New triangles' vertices are CCW ordered.
- The last added triangle contains the unique non-intersecting edge

Recovering cutting surface

Current triangle faces upwards

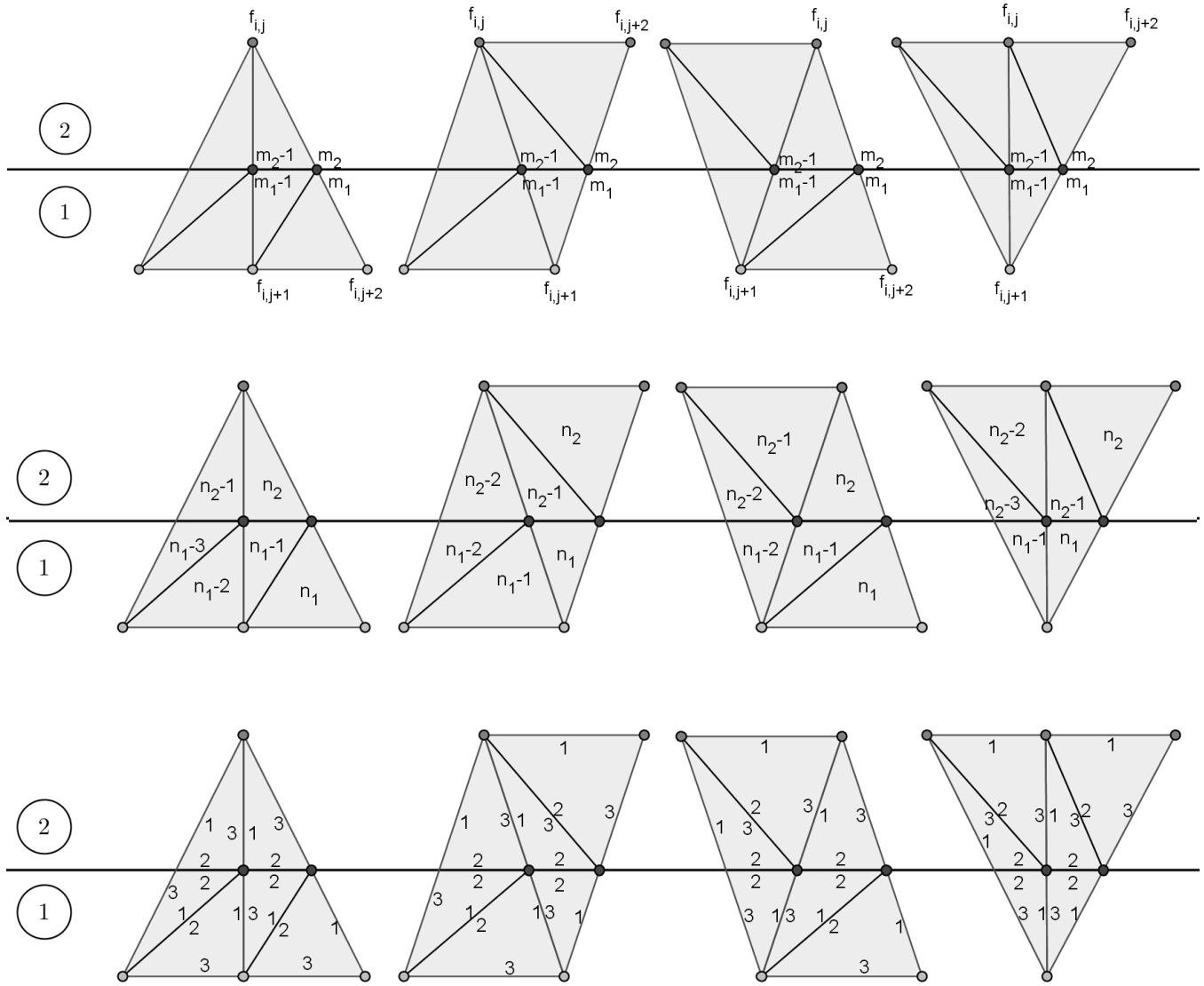
$$x := \overline{V(f_{i,j+1})V(f_{i,j+2})} \cap P$$
$$m_2 := m_2 + 1$$
$$V_2(m_2) := x$$
$$m_1 := m_1 + 1$$
$$V_1(m_1) := x$$

$$n_2 := n_2 + 2$$
$$F_2(n_2 - 1) := (\pi_V(f_{i,j}),\ m_2 - 1,\ m_2)$$
$$F_2(n_2) := (\pi_V(f_{i,j+2}),\ \pi_V(f_{i,j}),\ m_2)$$
$$n_1 := n_1 + 1$$
$$F_1(n_1) := (\pi_V(f_{i,j+1}),\ m_1,\ m_1 - 1)$$

Current triangle faces downwards

$$x := \overline{V(f_{i,j})V(f_{i,j+2})} \cap P$$
$$m_2 := m_2 + 1$$
$$V_2(m_2) := x$$
$$m_1 := m_1 + 1$$
$$V_1(m_1) := x$$

$$n_2 := n_2 + 1$$
$$F_2(n_2) := (\pi_V(f_{i,j}),\ m_2 - 1,\ m_2)$$
$$n_1 := n_1 + 2$$
$$F_1(n_1 - 1) := (\pi_V(f_{i,j+1}),\ m_1,\ m_1 - 1)$$
$$F_1(n_1) := (\pi_V(f_{i,j+2}),\ m_1,\ \pi_V(f_{i,j+1}))$$



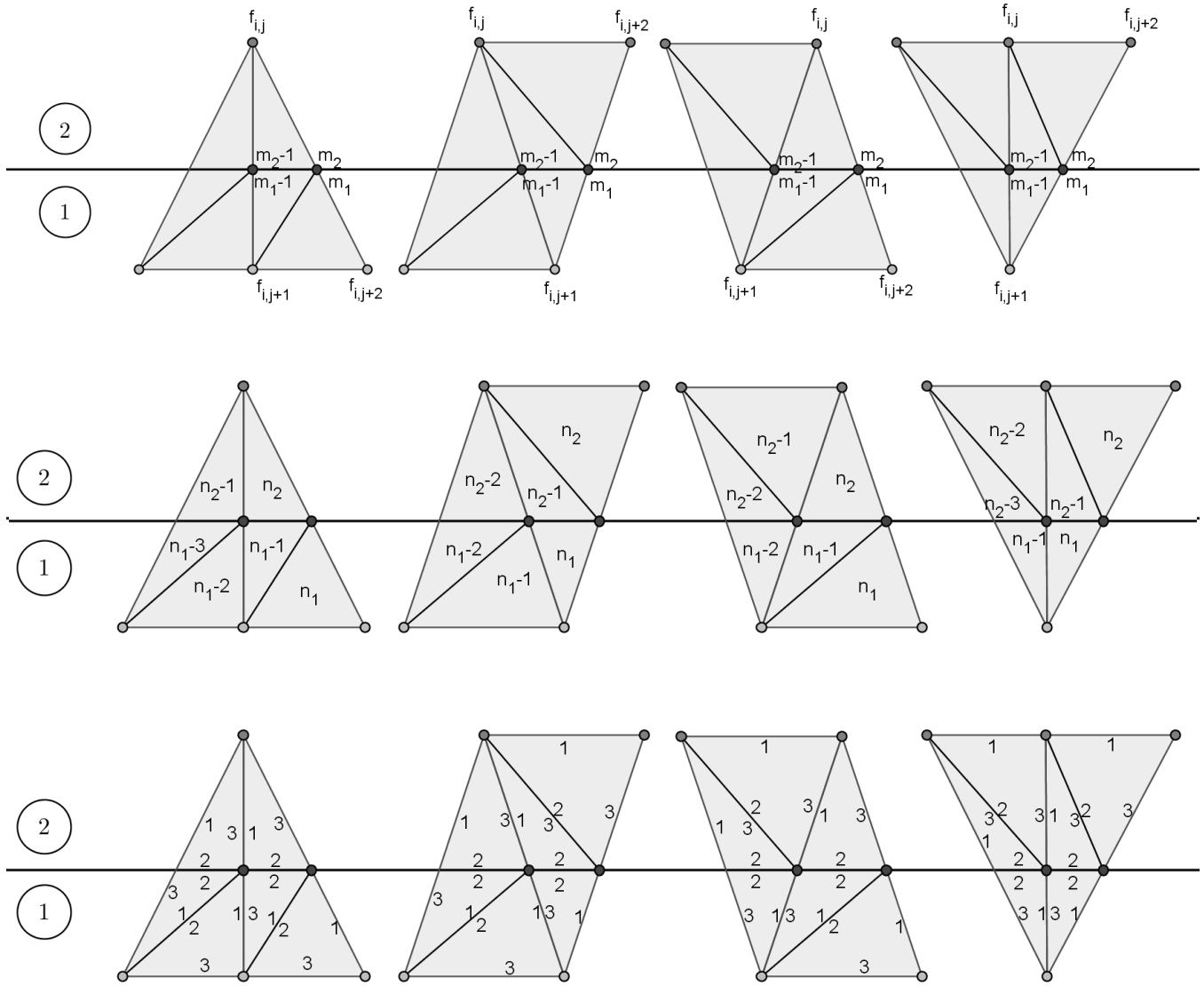
Recovering cutting surface

Current triangle faces upwards

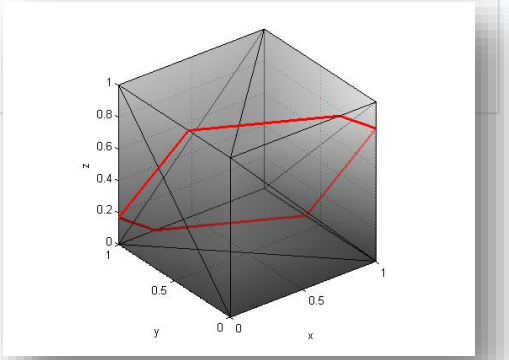
$$A_2(n_2, 1) := n_2 - 1$$
$$A_2(n_2 - 1, 3) := n_2$$
$$A_1(n_1 - 1, 1) := n_1$$
$$A_1(n_1, 2) := n_1 - 1$$
$$A_1(n_1 - 1, 3) := n_1 - 2$$
$$A_1(n_1 - 2, 1) := n_1 - 1$$
$$A_2(n_2, 2),\ A_1(n_1 - 1, 2) := 0$$
$$A_1(n_1, 3) := \pi_F(A(i, j \dot{+} 1))$$
$$A_1(\pi_F(A(i, j \dot{+} 1)), 3) := n_1$$
$$\pi_F(i) := n_1$$

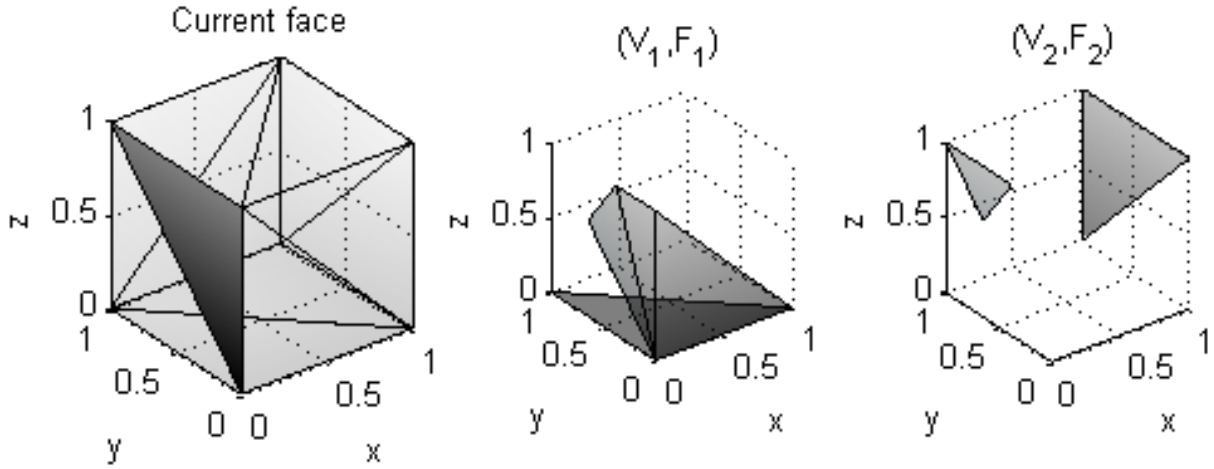
Current triangle faces downwards

$$A_1(n_1, 3) := n_1 - 1$$
$$A_1(n_1 - 1, 1) := n_1$$
$$A_2(n_2 - 1, 3) := n_2$$
$$A_2(n_2, 2) := n_2 - 1$$
$$A_2(n_2 - 1, 1) := n_2 - 2$$
$$A_2(n_2 - 2, 3) := n_2 - 1$$
$$A_1(n_1, 2),\ A_2(n_2 - 1, 2) := 0$$
$$A_2(n_2, 1) := \pi_F(A(i, j \dot{+} 2))$$
$$A_2(\pi_F(A(i, j \dot{+} 2)), 1) := n_2$$
$$\pi_F(i) := n_2$$

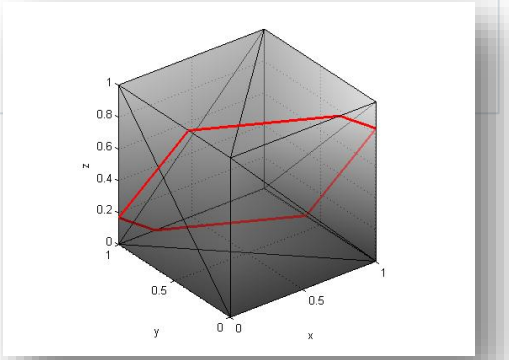


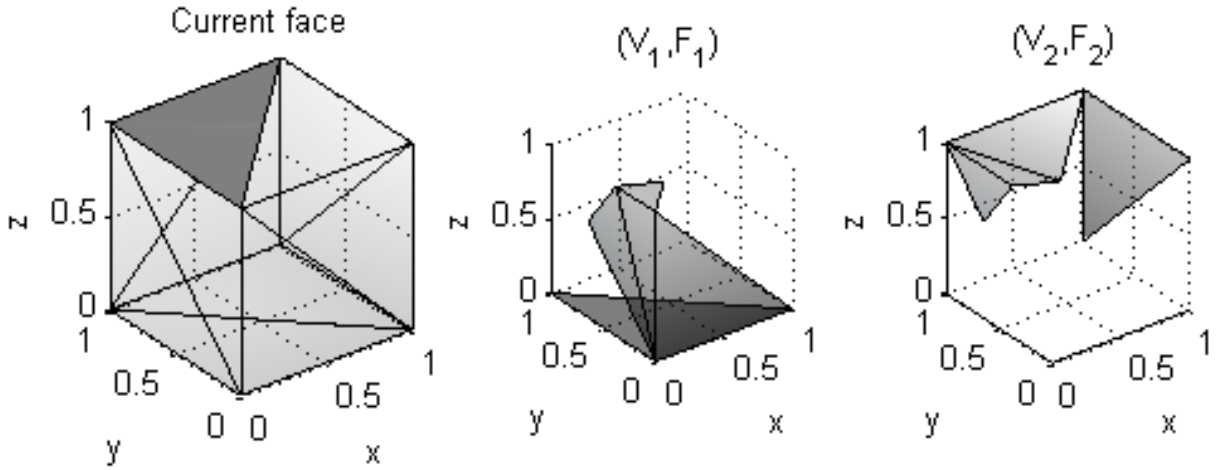
Recovering cutting surface



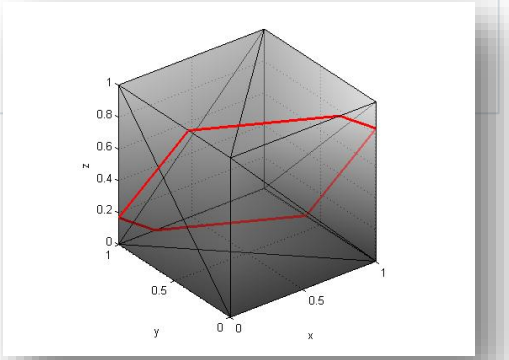


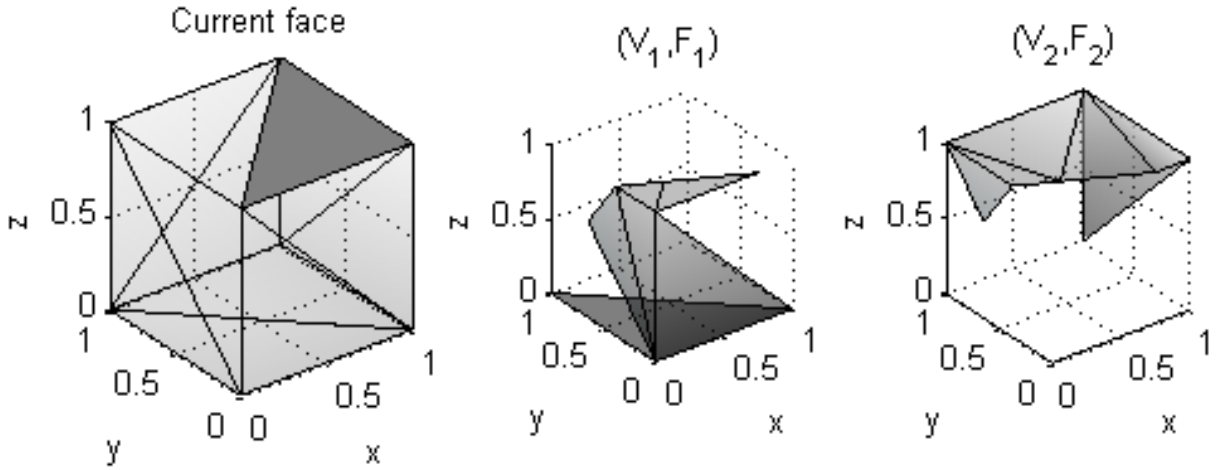
Recovering cutting surface



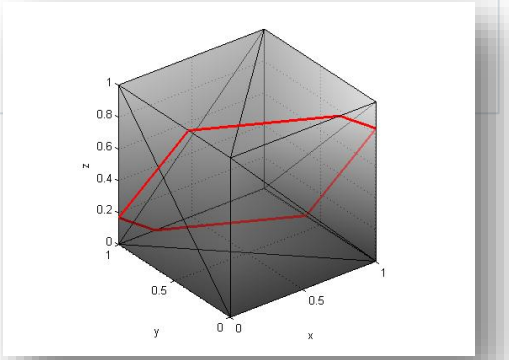


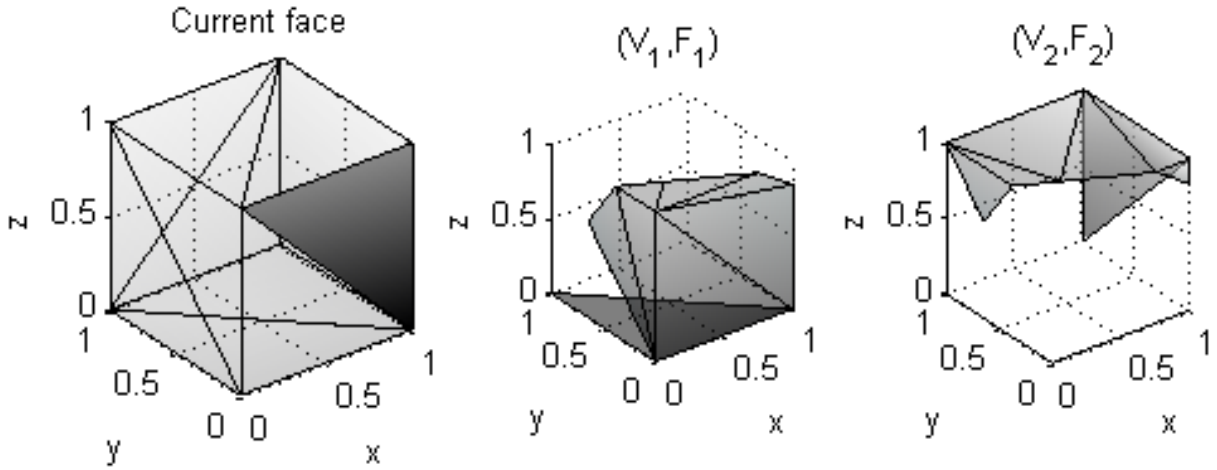
Recovering cutting surface



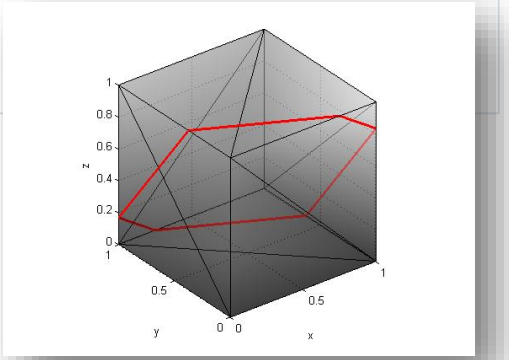


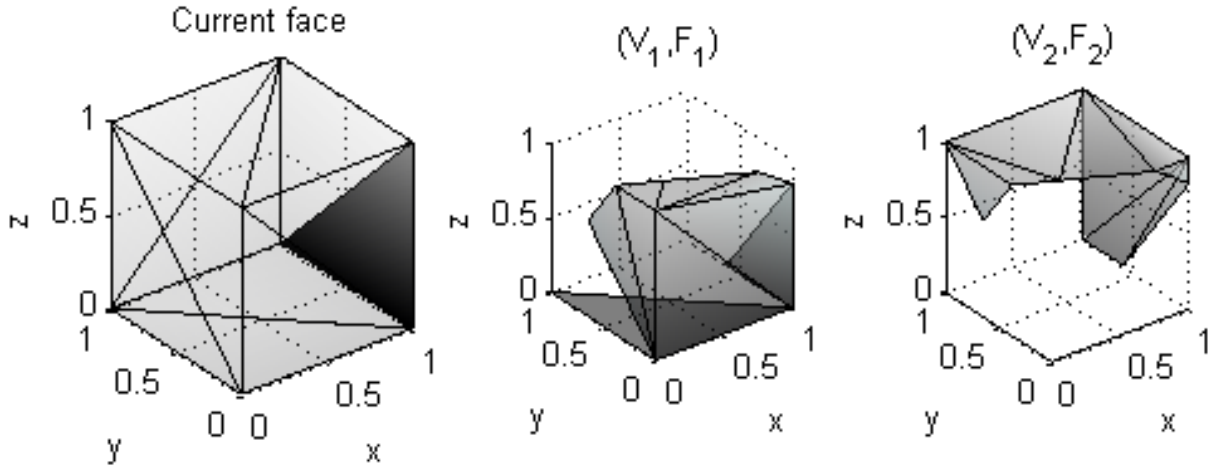
Recovering cutting surface



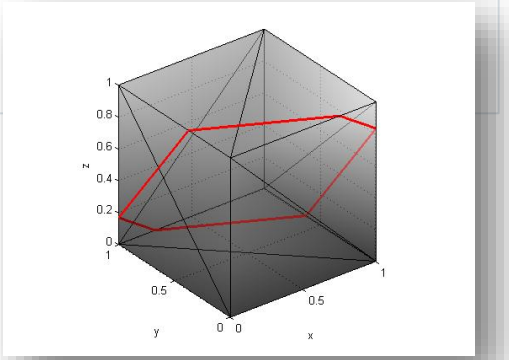


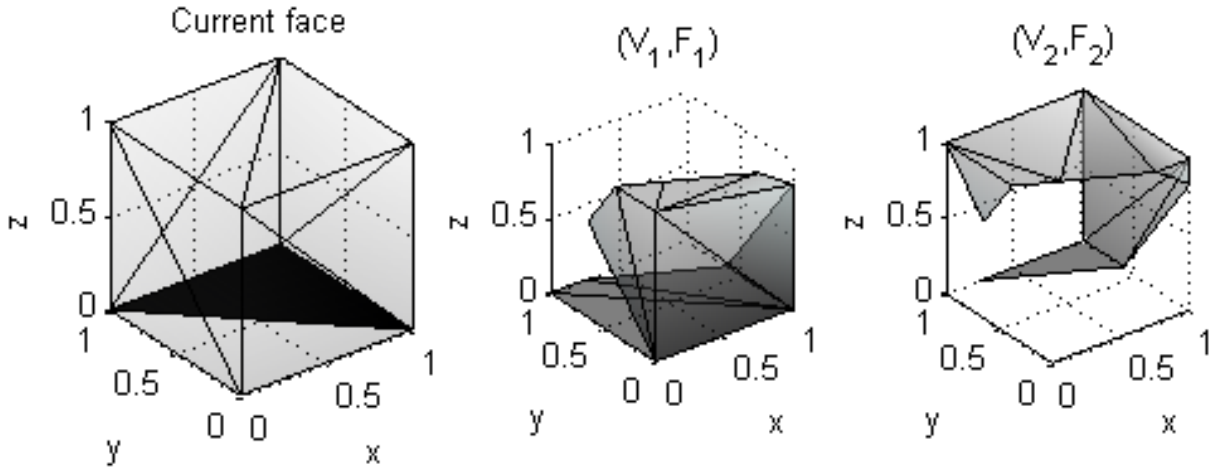
Recovering cutting surface



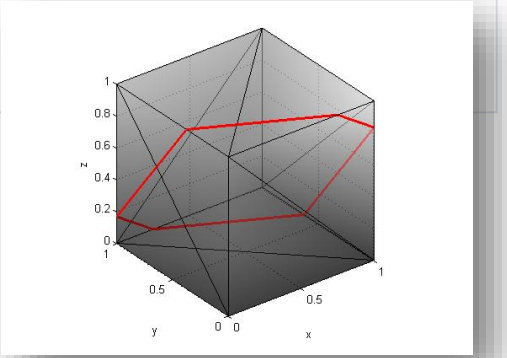


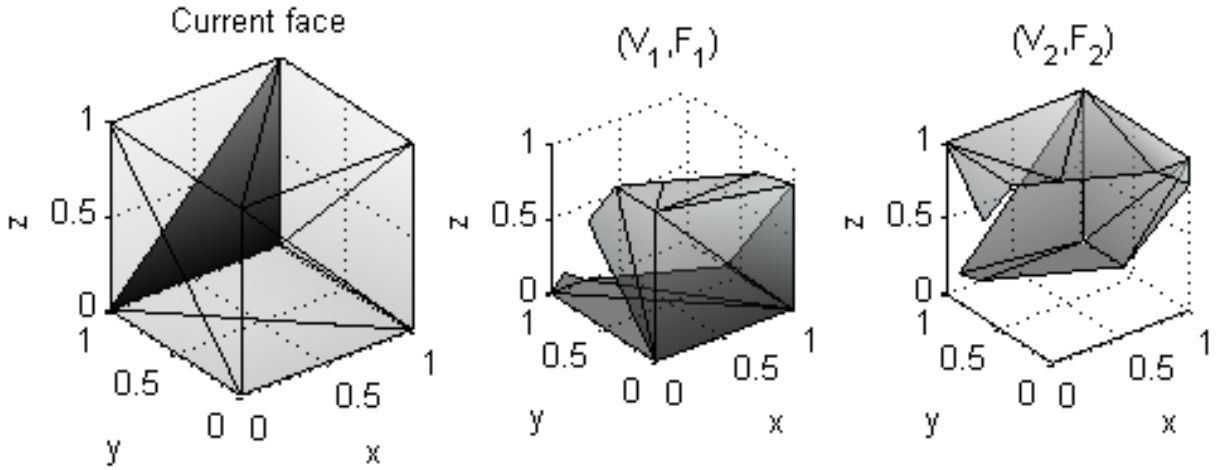
Recovering cutting surface



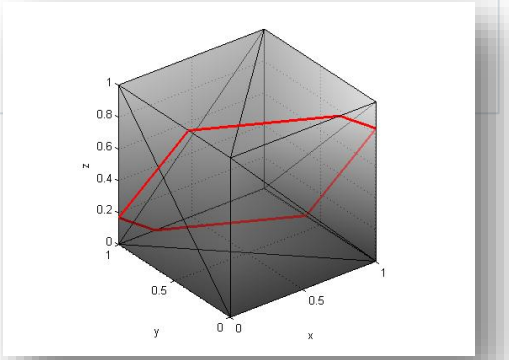


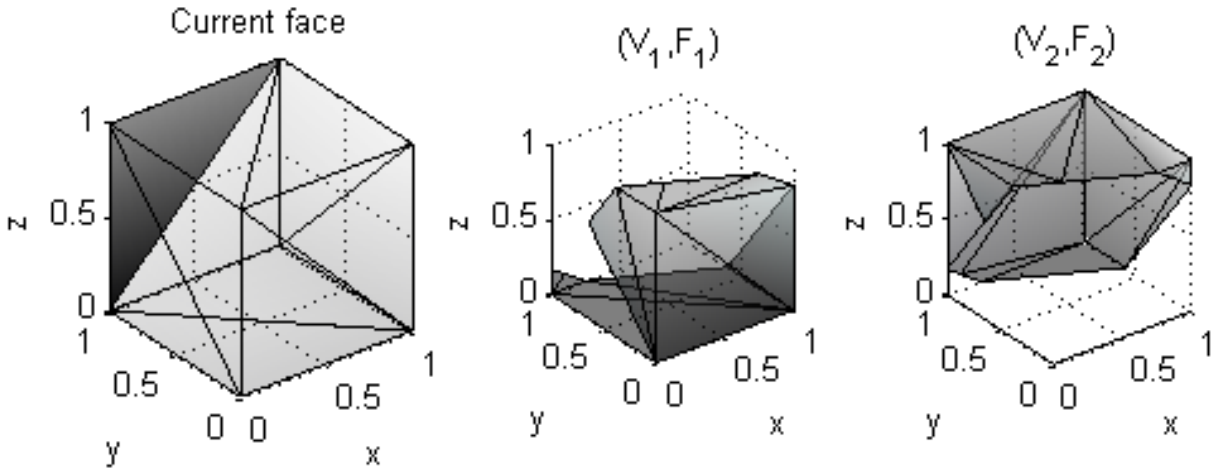
Recovering cutting surface



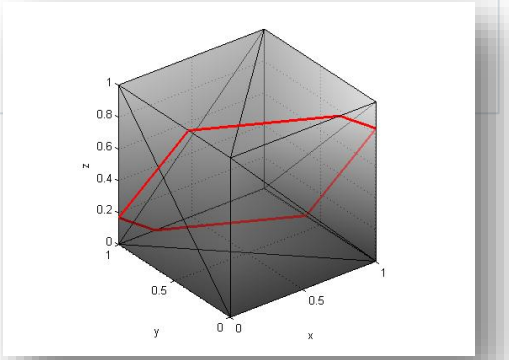


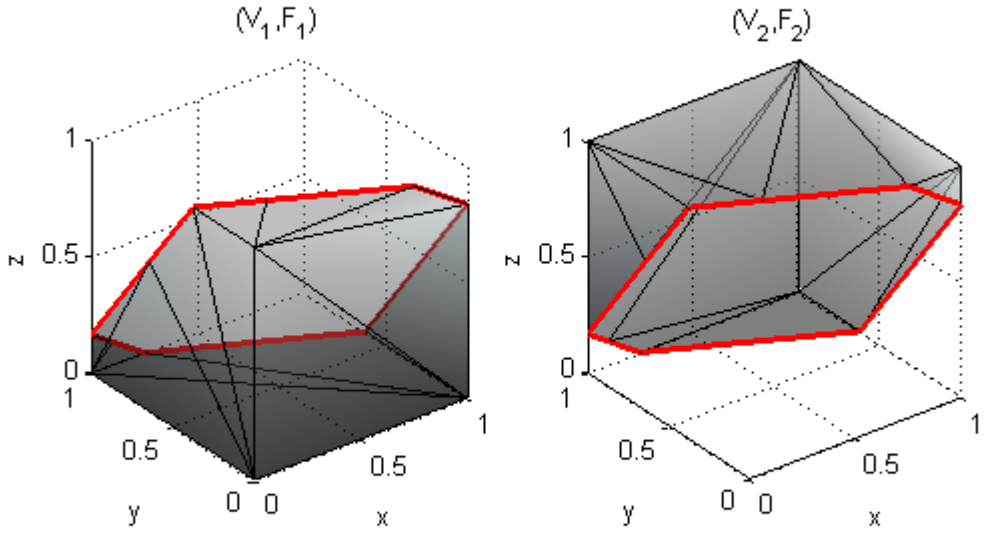
Recovering cutting surface





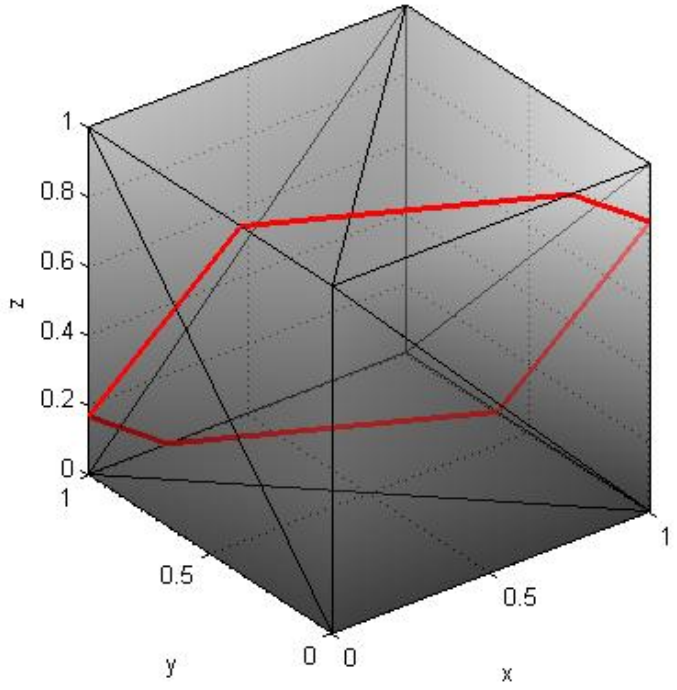
Recovering cutting surface





Outline of algorithm

- Generating adjacencies
- Sorting vertices, faces and adjacencies
- Creating new vertices and faces
- **Capping holes**
- Decomposition



V =

0	0	1
0	0	0
1	0	0
1	0	1
0	1	1
0	1	0
1	1	0
1	1	1

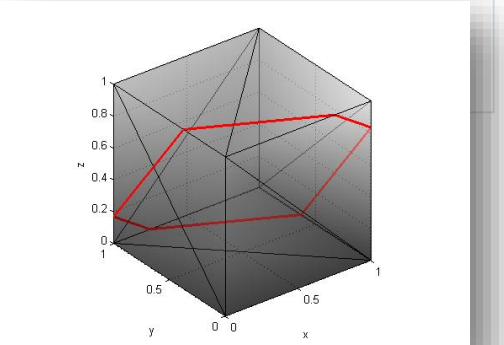
F =

2	1	5
2	5	6
6	7	3
6	3	2
7	8	4
7	4	3
1	4	8
1	8	5
1	2	3
1	3	4
8	7	6
8	6	5

A =

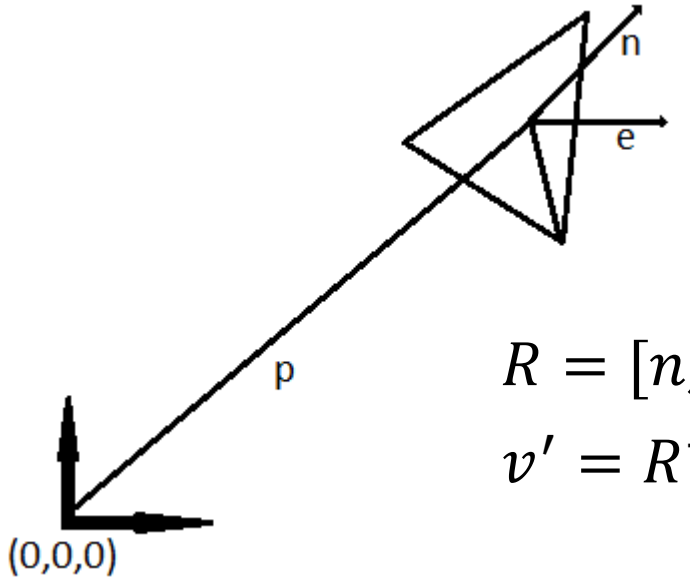
9	8	2
1	12	4
11	6	4
3	9	2
11	7	6
5	10	3
10	5	8
7	12	1
1	4	10
9	6	7
5	3	12
11	2	8

Capping holes



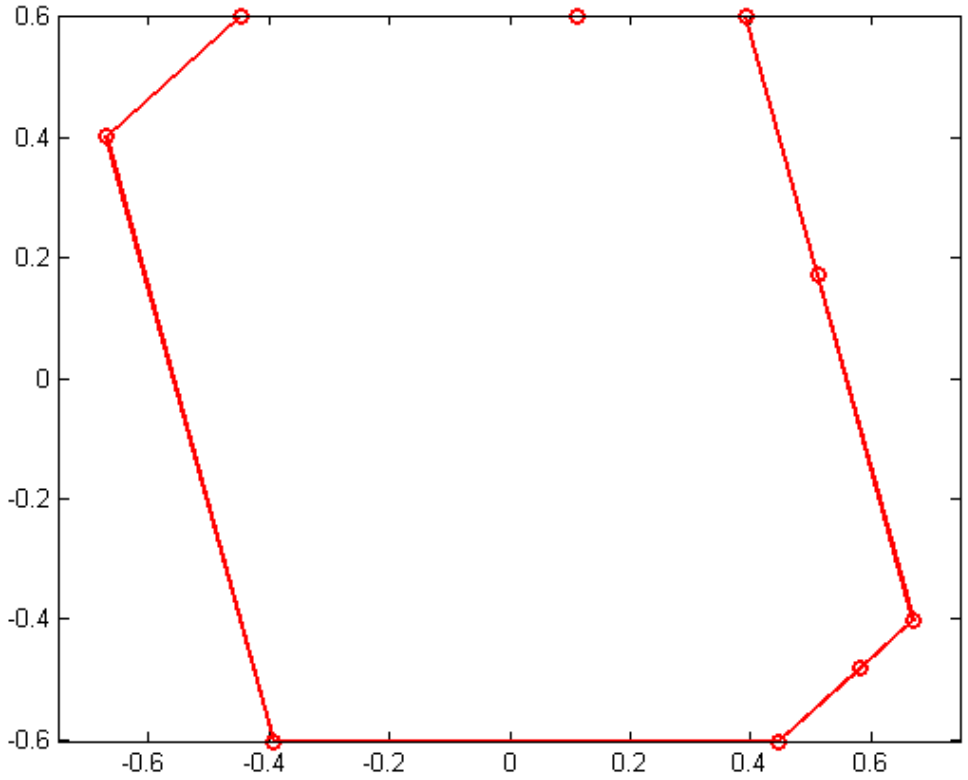
We need to rotate points to the cutting plane
to obtain a common 2D polygon triangulation problem

It can be done by translation and base change

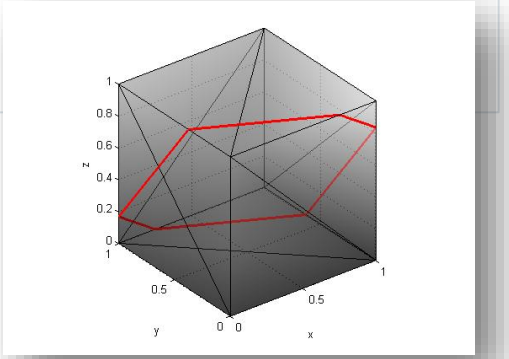


$$R = [n, -n \times e, e]$$

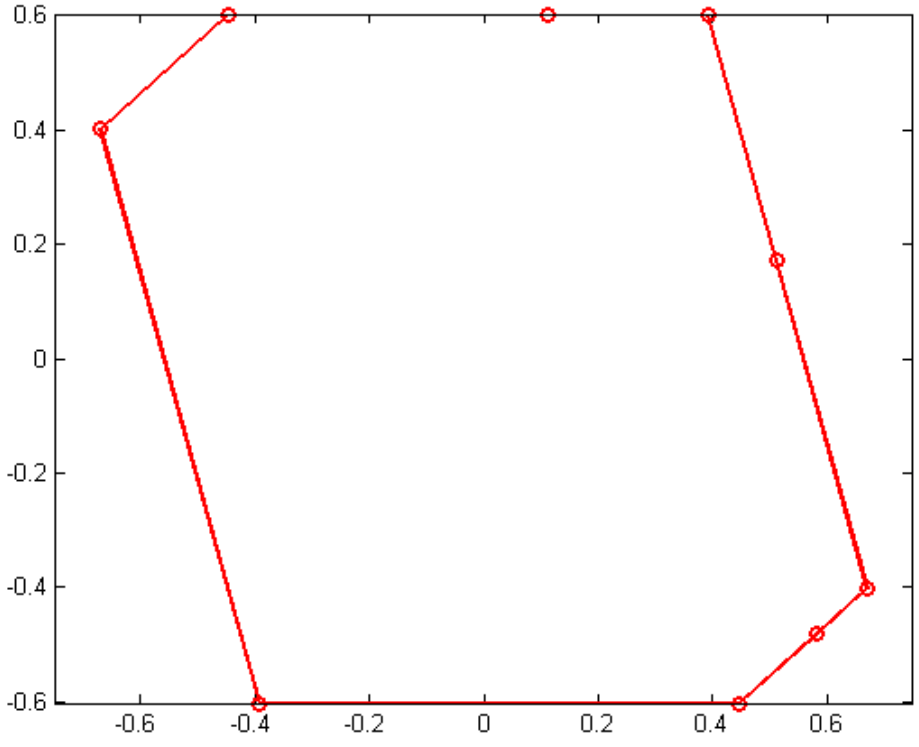
$$v' = R^{-1}(v - p)$$

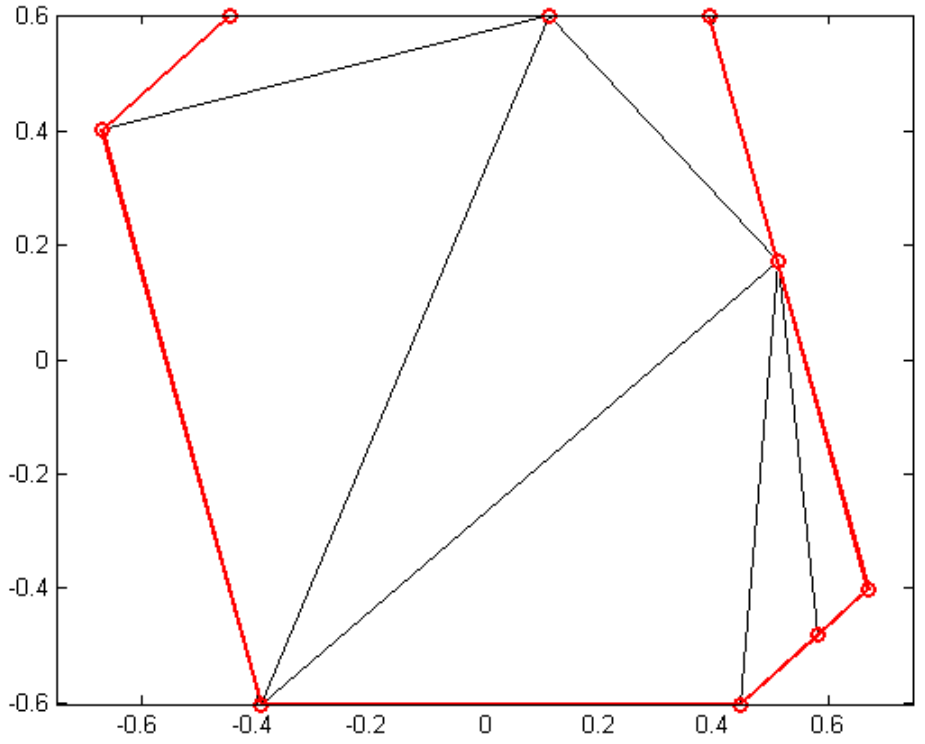


Capping holes

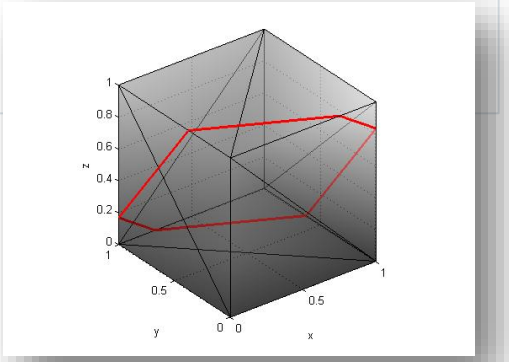


Common triangulation problem of a simple polygon
We used Constrained Delaunay Triangulation



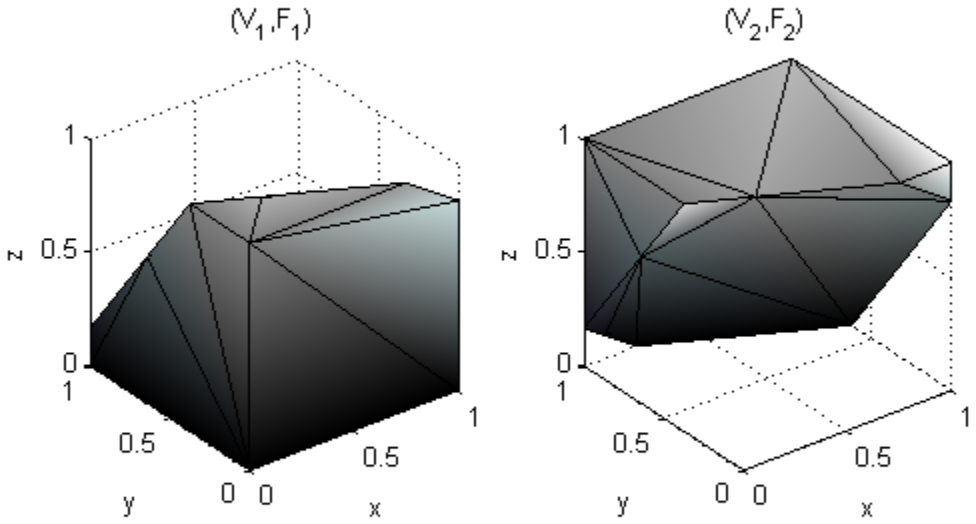


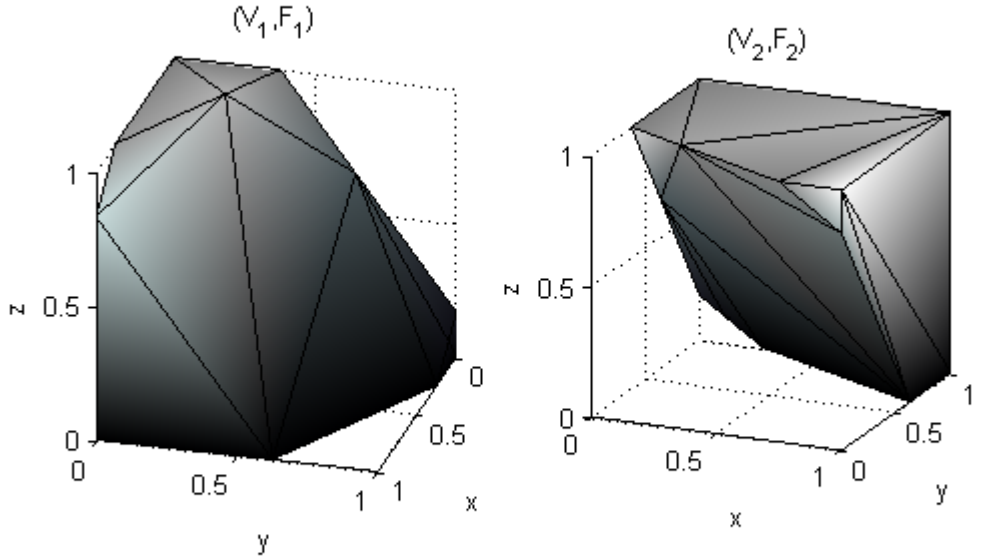
Reindexing faces and adjacencies



We need to set normals approriate direction above and below the cutting plane

It can be done by use an acyclic permutation if we find the normal's direction is wrong





Outline of algorithm

- Generating adjacencies
- Sorting vertices, faces and adjacencies
- Creating new vertices and faces
- Capping holes
- **Decomposition**

V =

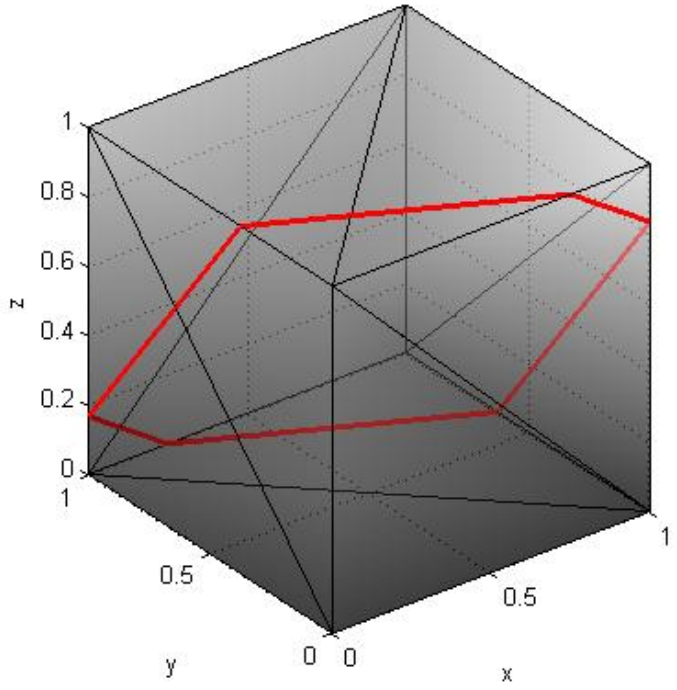
0	0	1
0	0	0
1	0	0
1	0	1
0	1	1
0	1	0
1	1	0
1	1	1

F =

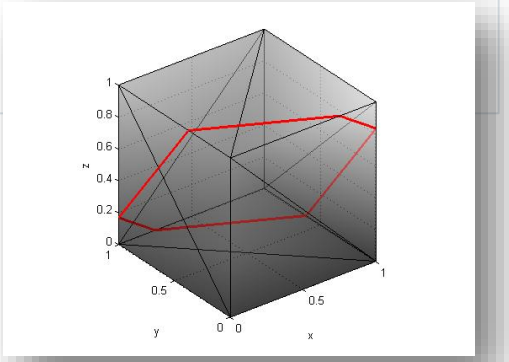
2	1	5
2	5	6
6	7	3
6	3	2
7	8	4
7	4	3
1	4	8
1	8	5
1	2	3
1	3	4
8	7	6
8	6	5

A =

9	8	2
1	12	4
11	6	4
3	9	2
11	7	6
5	10	3
10	5	8
7	12	1
1	4	10
9	6	7
5	3	12
11	2	8

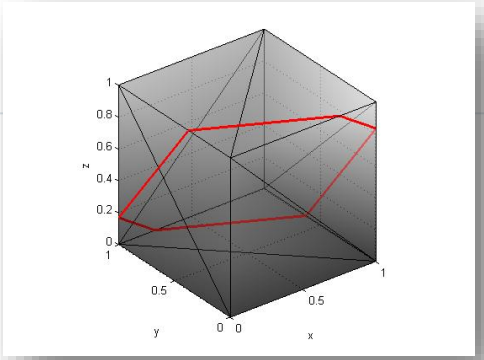


Decomposition



- We mark a triangle and add its adjacencies to set of possible triangles
- If there is an unmarked triangle between the possible triangles, we mark it, and add its adjacencies to the possible triangles
- If there is no more unmarked triangle between the possibles, we found a connected component
- If there is no more unmarked triangle, we found all connected components

Result



V1 =

0	0	1.0000
0	0	0
1.0000	0	0
0	1.0000	0
0	0.6429	0.6429
0	0.3750	1.0000
0.2500	0.2500	1.0000
0.7500	0	1.0000
1.0000	0	0.8333
1.0000	0.6250	0
0.2500	1.0000	0
0.1000	1.0000	0.1000
0	1.0000	0.1667

F1 =

4	3	2
1	2	3
2	6	5
1	6	2
1	7	6
1	8	7
1	9	8
3	9	1
3	10	9
3	11	10
4	11	3
4	12	11
4	13	12
4	5	13
2	5	4
9	10	7
9	7	8
10	5	7
7	5	6
10	11	5
11	12	5
12	13	5

A1 =

11	2	15
4	1	8
4	19	15
5	3	2
6	19	4
7	17	5
8	17	6
9	7	2
10	16	8
11	20	9
12	10	1
13	21	11
14	22	12
15	22	13
3	14	1
9	18	17
16	6	7
20	19	16
18	3	5
10	21	18
12	22	20
13	14	21

V2 =

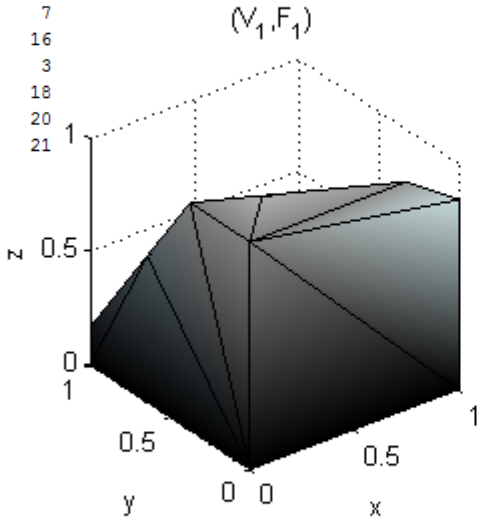
1.0000	0	1.0000
0	1.0000	1.0000
1.0000	1.0000	0
1.0000	1.0000	1.0000
0	0.6429	0.6429
0	0.3750	1.0000
0.2500	0.2500	1.0000
0.7500	0	1.0000
1.0000	0	0.8333
1.0000	0.6250	0
0.2500	1.0000	0
0.1000	1.0000	0.1000
0	1.0000	0.1667

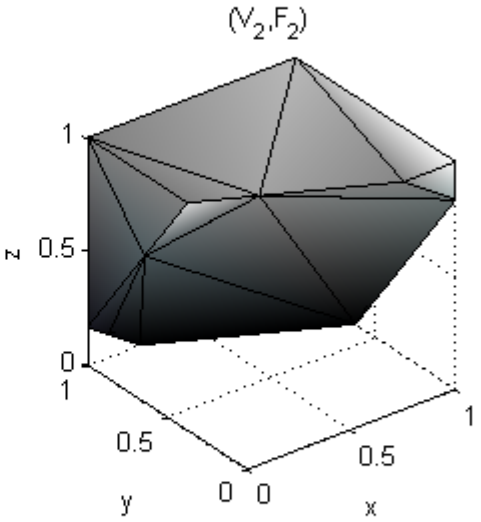
F2 =

3	4	1
2	5	6
2	6	7
4	2	7
4	7	8
1	4	8
1	8	9
1	9	10
3	1	10
3	10	11
3	11	12
4	3	12
4	12	13
2	4	13
2	13	5
9	10	7
9	7	8
10	5	7
7	5	6
10	11	5
11	12	5
12	13	5

A2 =

12	6	9
15	19	3
2	19	4
14	3	5
4	17	6
1	5	7
6	17	8
7	16	9
1	8	10
9	20	11
10	21	12
1	11	13
12	22	14
4	13	15
14	22	2
8	18	17
16	5	7
20	19	16
18	2	3
10	21	18
11	22	20
13	15	21





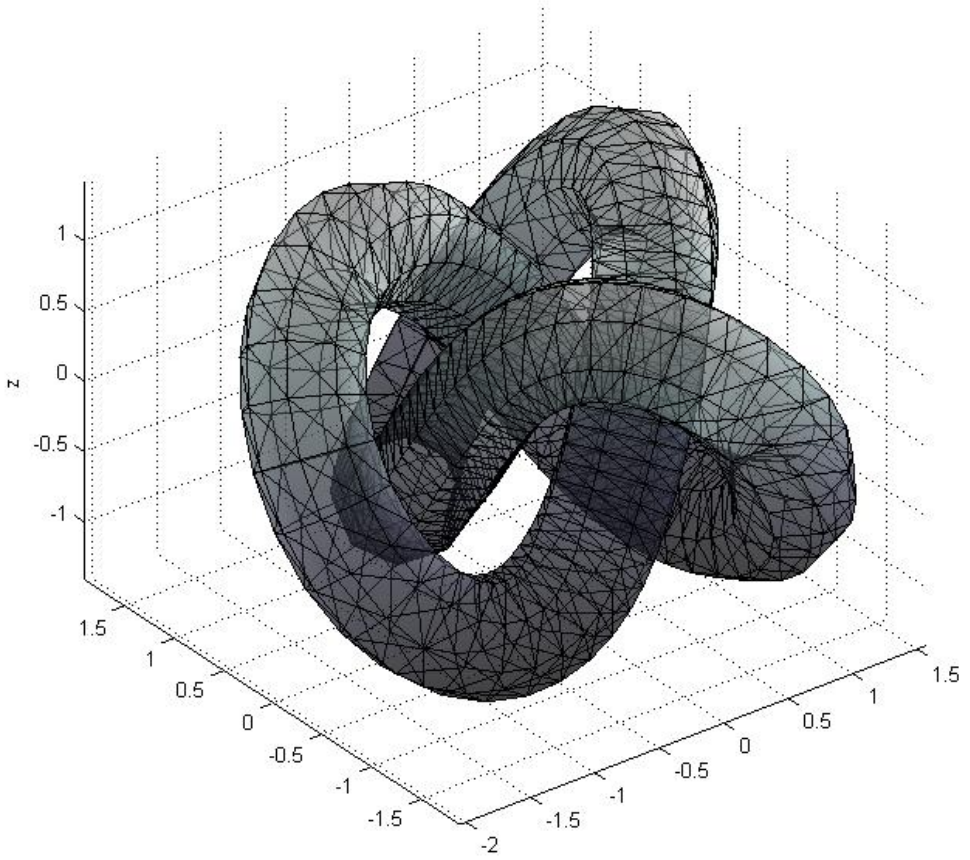
Time complexity

- Generating adjacencies: $|F|$
- Sorting vertices, faces and adjacencies: $|V| + |F|$
- Creating new vertices and faces: $\sum_{i=1}^{L} l_i$
- Capping holes: $\sum_{i=1}^{L} l_i \log(l_i)$
- Decomposition $|F|$

$|V|<|F|$

Generally true: $\sum_{i=1}^{L} l_i \log(l_i) < |F|$

Linear time complexity respect to number of faces



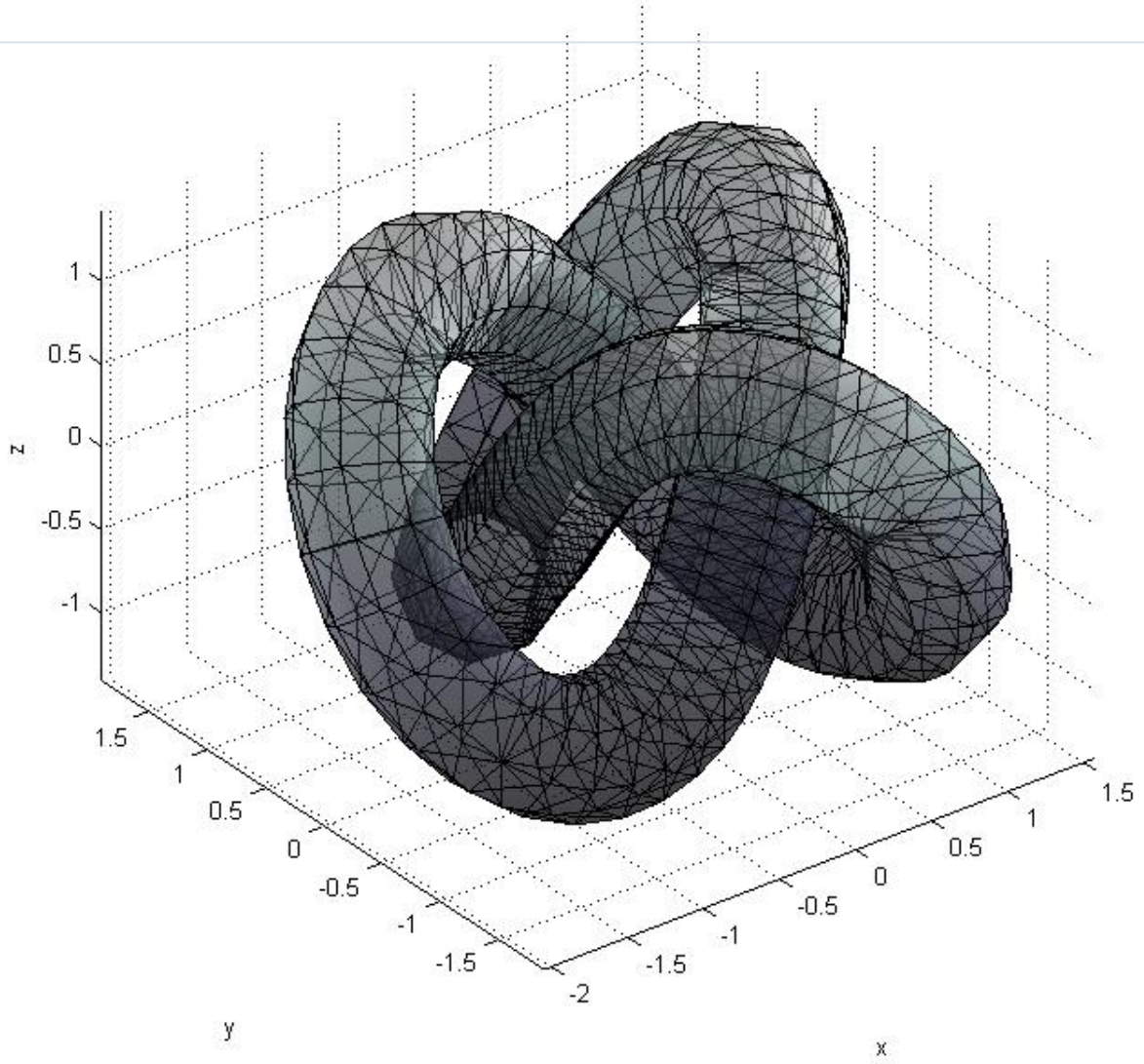
Tested for complex models, as torusknot:

$L = 3$

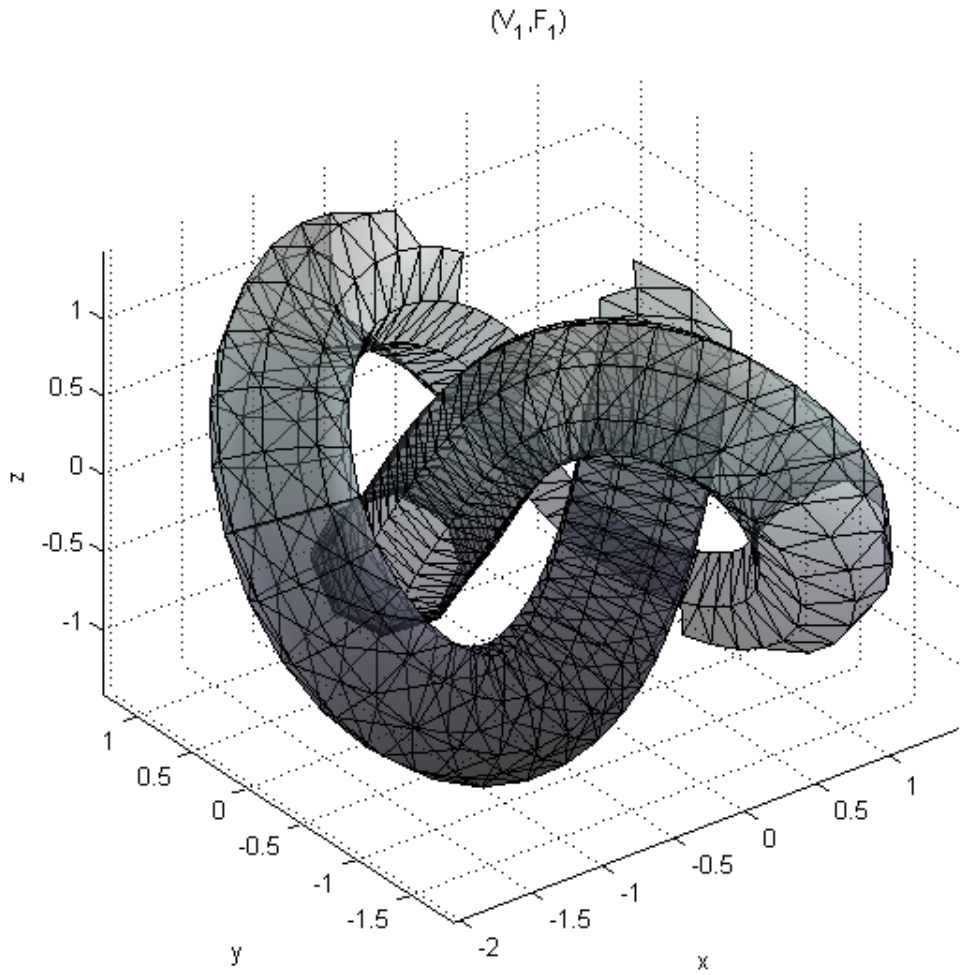
$\sum_{i=1}^{L} l_i \log(l_i) \approx 776$

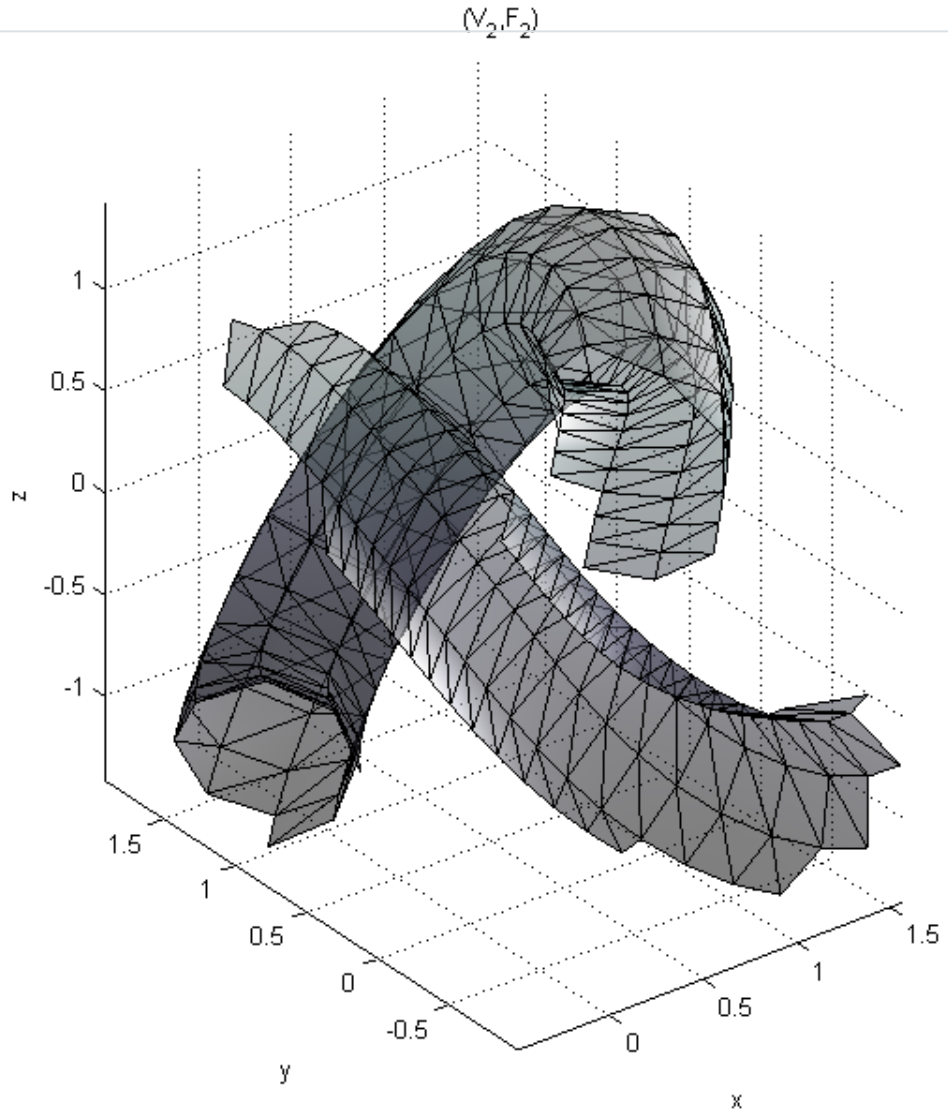
$|F| = 1760$

More loops

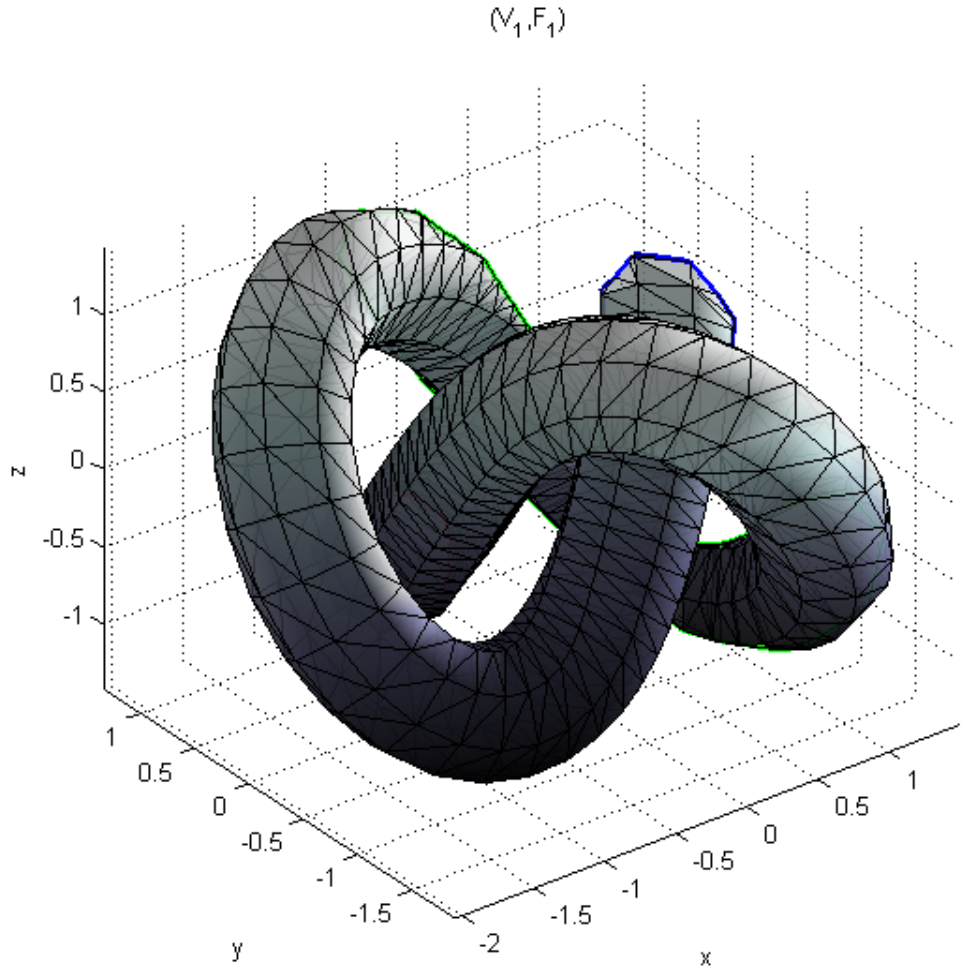


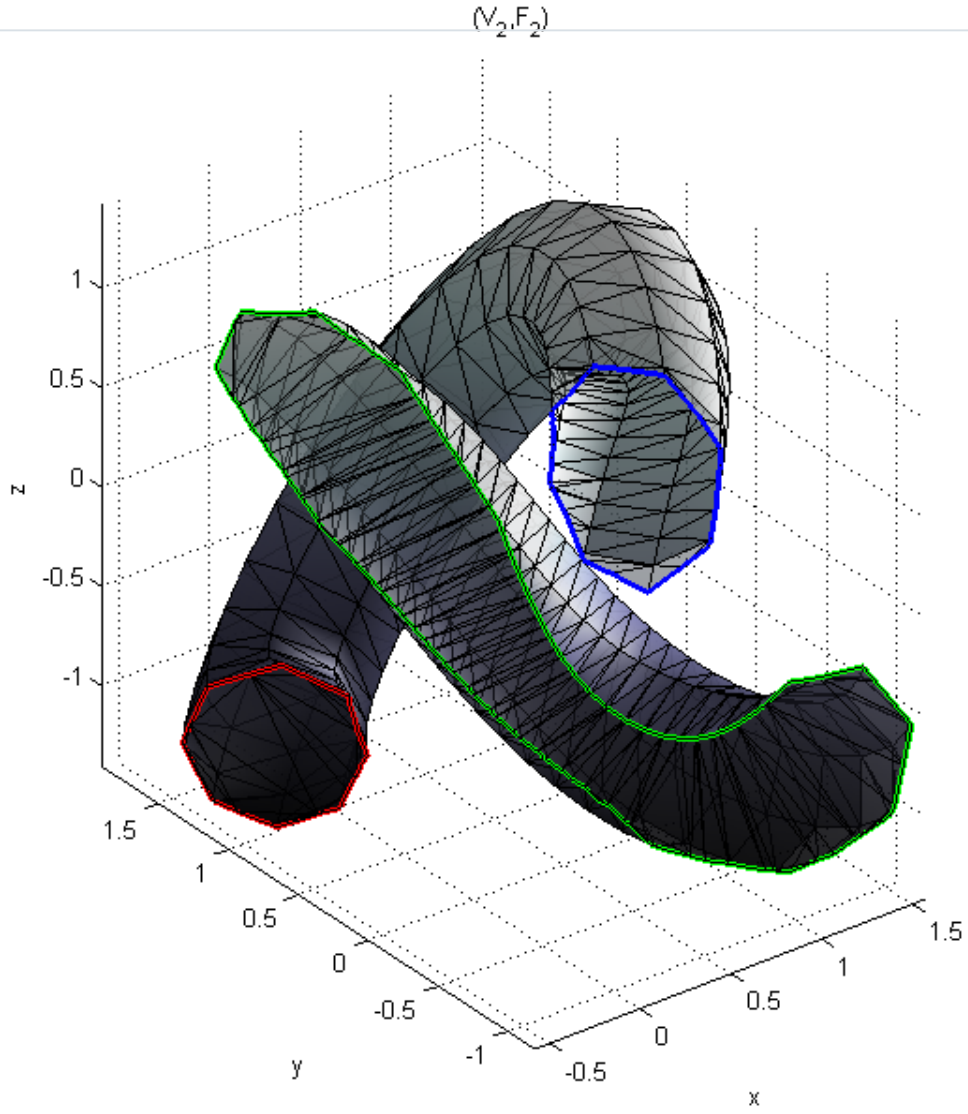
More loops



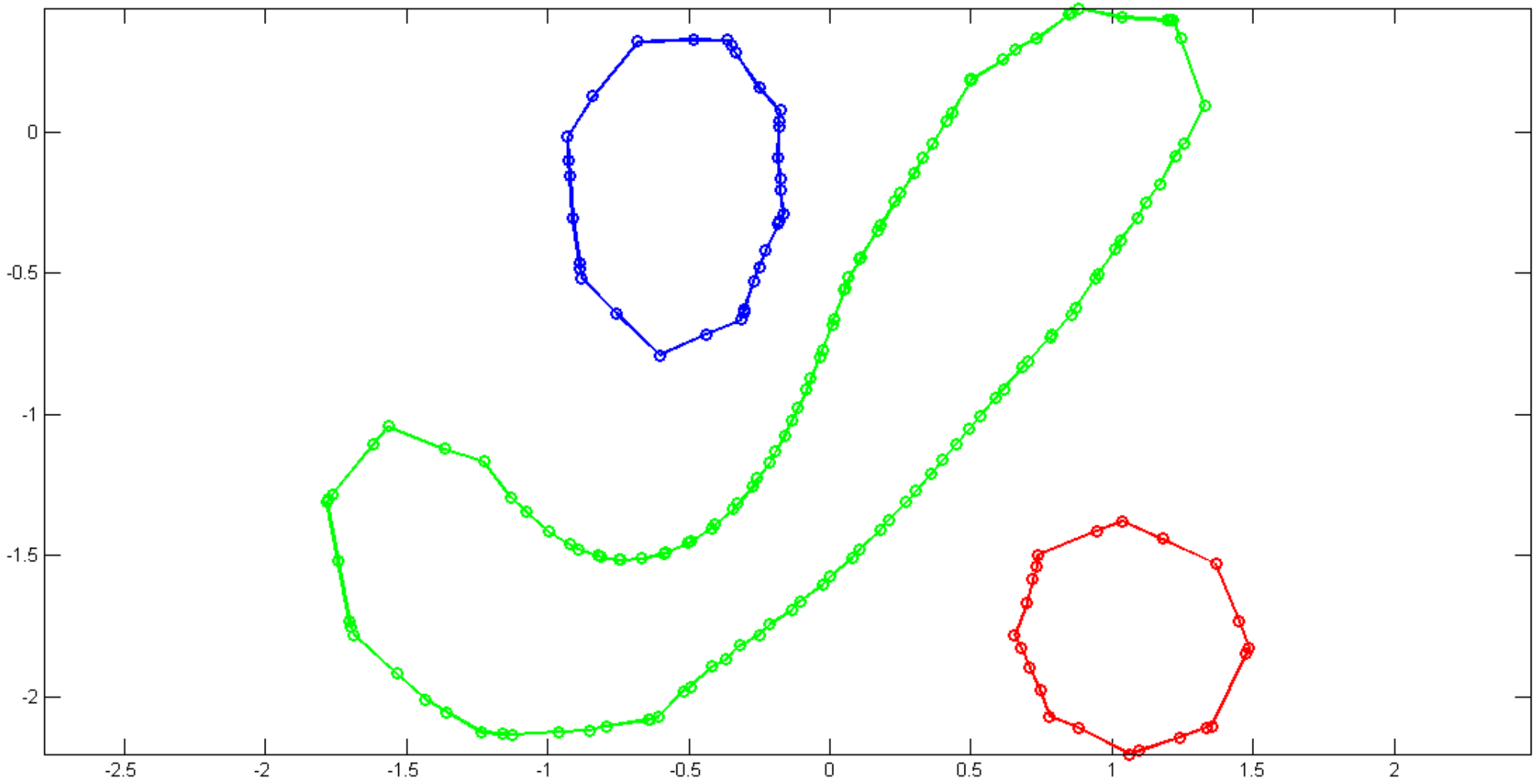


More loops

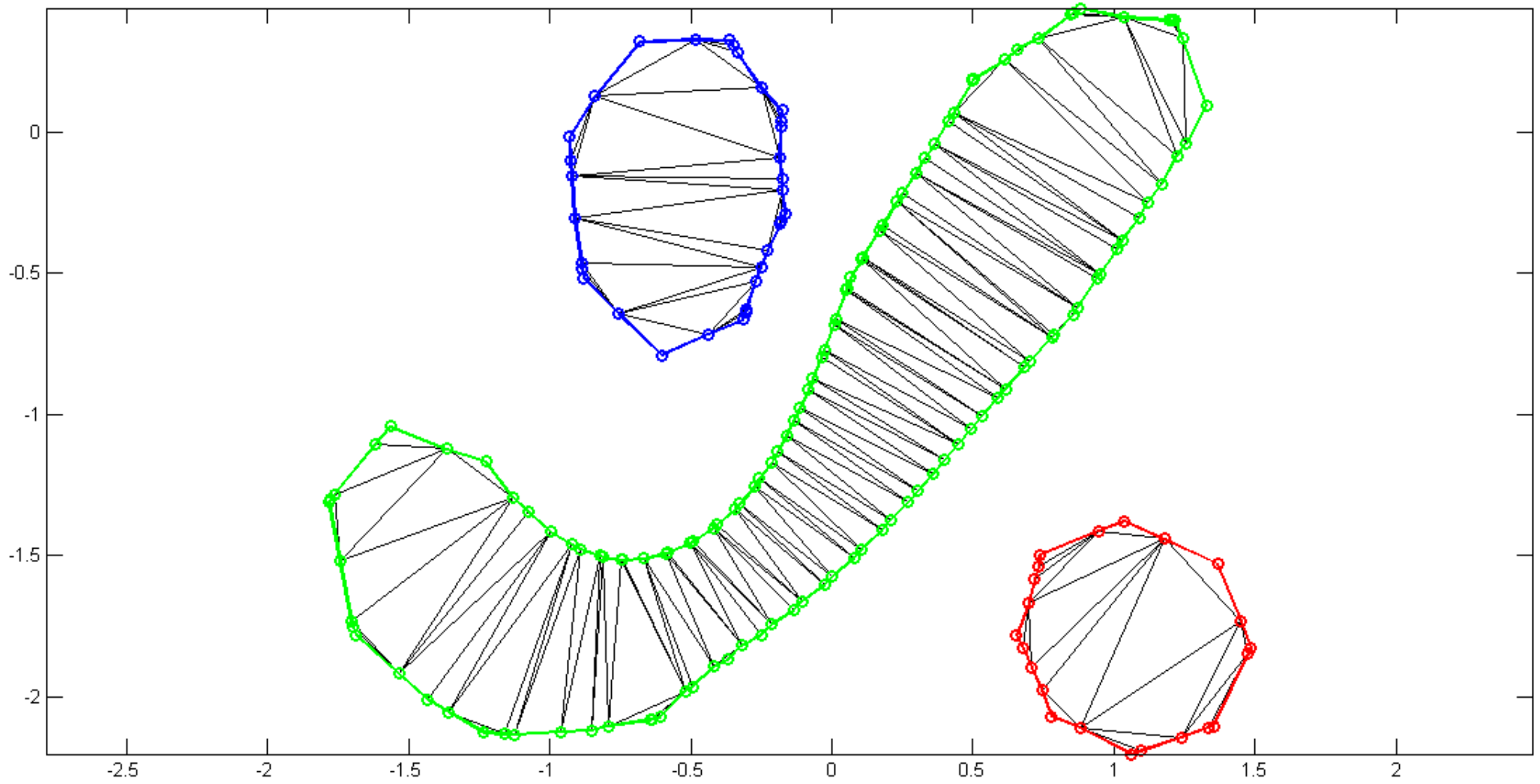




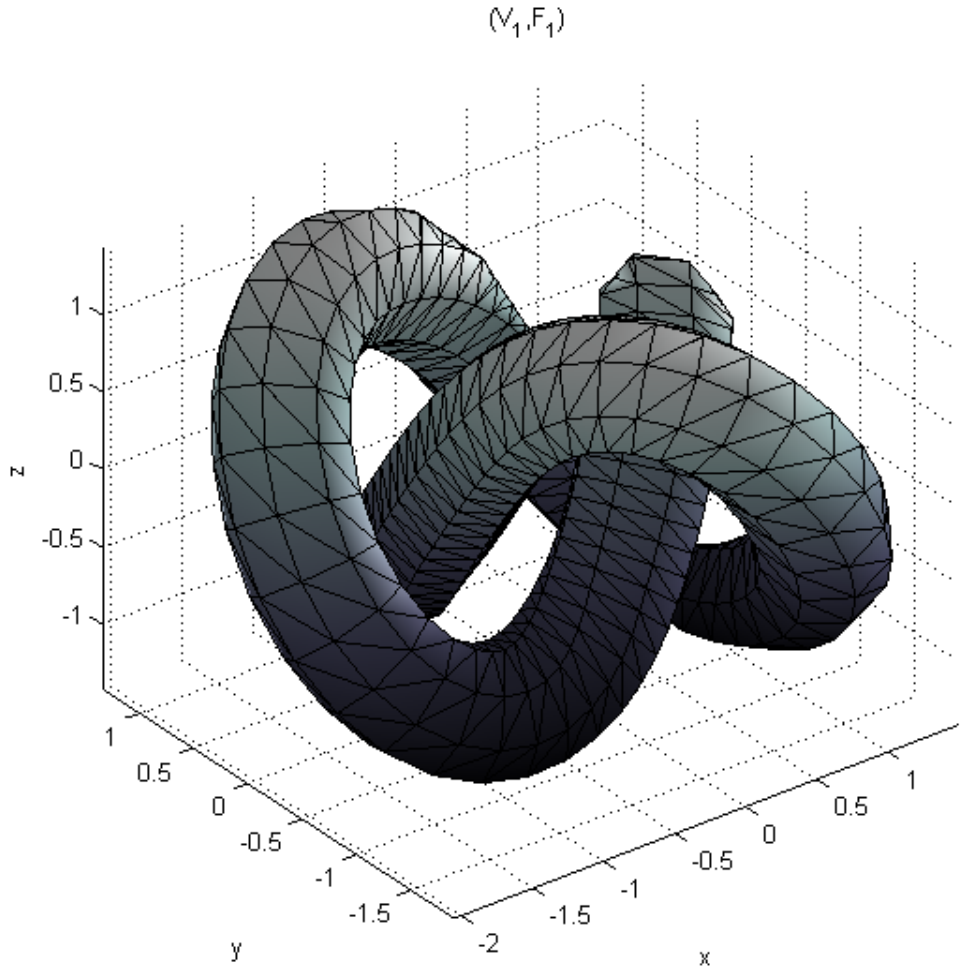
More loops

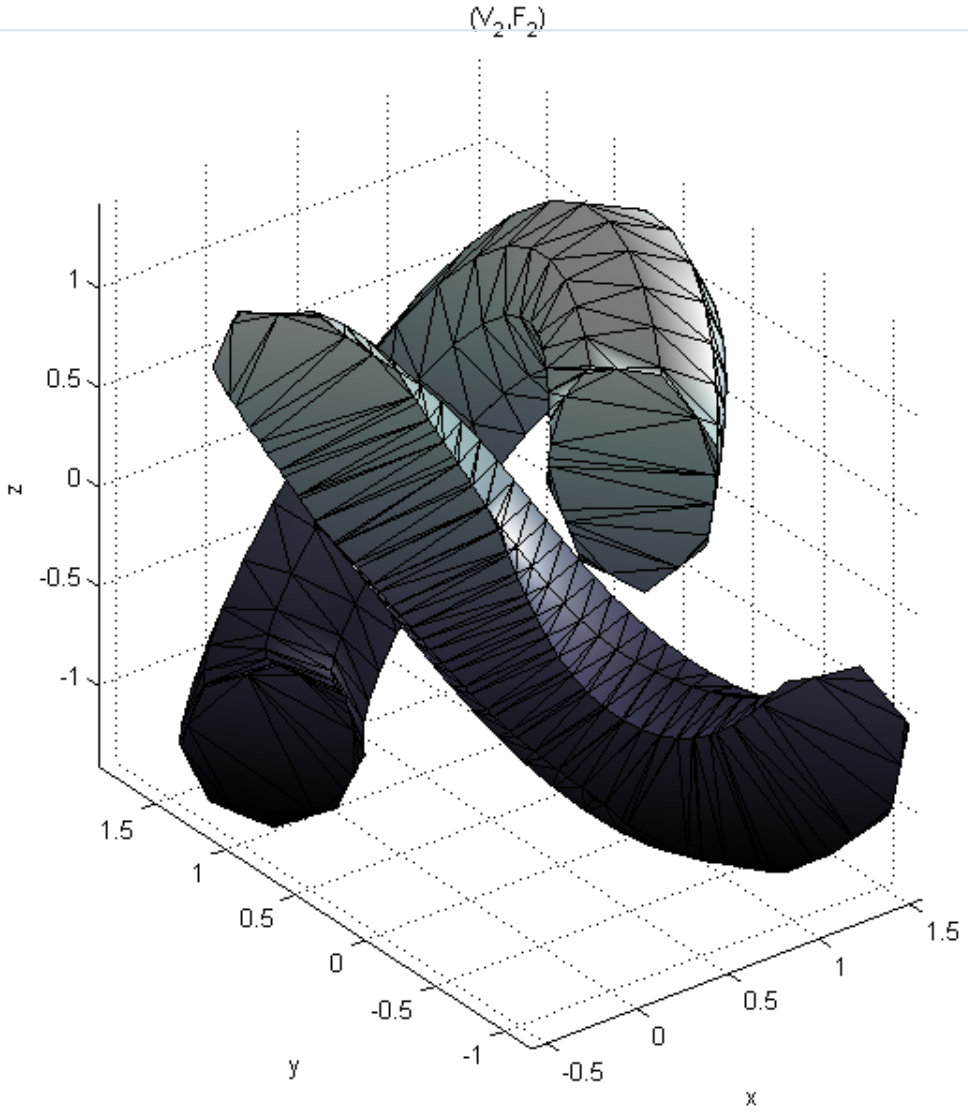
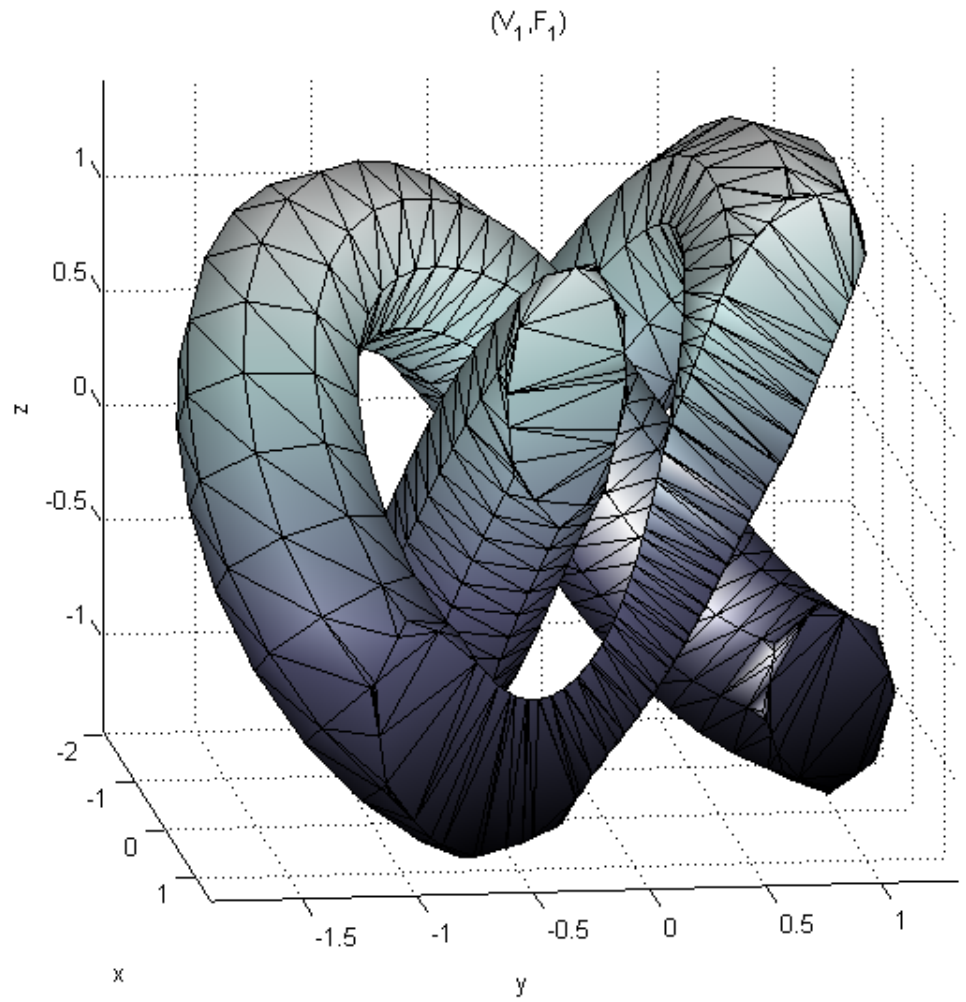
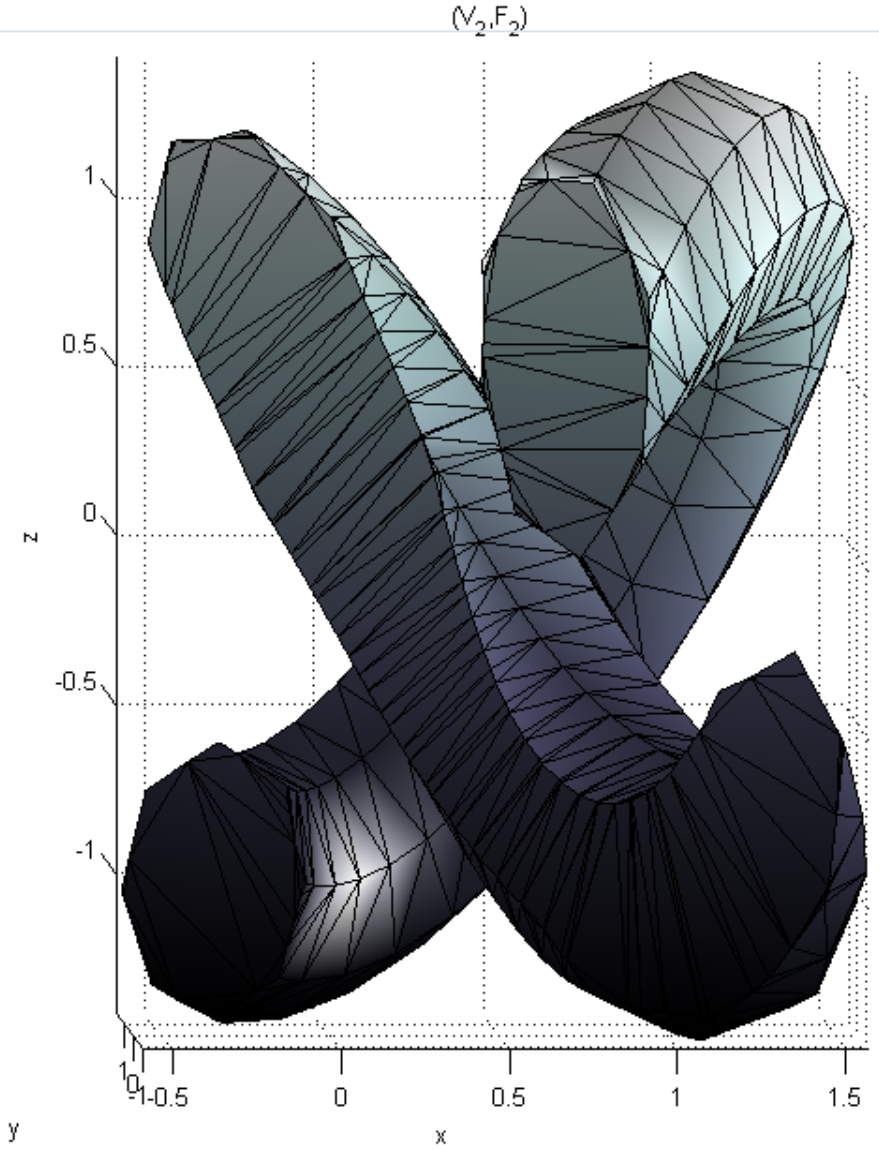


More loops



More loops





Thank you for your attention!