

Computational Lab I: Discrete Algebraic Surface Topology

讲师: Xianfeng (David) Gu

撰写: 洪楠方

助教: 洪楠方

最后更新: January 27, 2021

The discrete version of algebraic surface topology was built on oriented 2-dimensional simplicial complex, namely the triangular mesh, which was implemented by half-edge data structure in computer.

1 Half-edge Data Structure

Definition 1 (Half-edge Data Structure). The *half-edge* data structure of triangular mesh approximating an oriented surface has the following classes:

V *vertex* class

H *half-edge* class, oriented from one vertex (the *source* vertex) to another vertex (the *target* vertex)

E *edge* class, each edge has two opposite half-edges, with the exception that the edge on boundary only has one half-edge

F *face* class, each face has three half-edges, oriented counter-clockwisely with respect to the normal of face

with the following pointer functions which take in the realization of classes above:

$v(\cdot)$ vertex pointer function $v : \mathbf{H} \rightarrow \mathbf{V}$ parametrized by “source/target” that points a half-edge to a vertex.
 $v_{\text{sour}}(\cdot)$ points to the source vertex, and $v_{\text{targ}}(\cdot)$ points to the target vertex

$f(\cdot)$ face pointer function $f : \mathbf{H} \rightarrow \mathbf{F}$ that points a half-edge to a face that it belongs to

$e(\cdot)$ edge pointer function $e : \mathbf{H} \rightarrow \mathbf{E}$ that points a half-edge to an edge that it belongs to

$h(\cdot)$ the polymorphic half-edge pointer function can recognize different inputs and take different actions:

half-edge $h : \mathbf{H} \rightarrow \mathbf{H}$ parametrized by “next/previous” that points a half-edge to another half-edge, where
 $h_{\text{next}}(\cdot)$ points to the next half-edge, and $h_{\text{prev}}(\cdot)$ points to the previous half-edge

face $h : \mathbf{F} \rightarrow \mathbf{H}$ points a face to the *first* half-edge it contains

vertex $h : \mathbf{V} \rightarrow \mathbf{H}$ points a vertex to the *first* half-edge that the vertex was targeted

edge $h : \mathbf{E} \rightarrow \mathbf{H} \times \mathbf{H}$ points an edge to the pair of half-edges it contains (with exception of boundary edge that has only one half-edge, in that case $h : \mathbf{E} \rightarrow \mathbf{H}$)

Example 2 (2-chain). As shown in figure 1, we have realization of classes:

$$\mathbf{V} = \{v_1, v_2, v_3, v_4, v_5, v_6\} =: v_{1:6}$$

$$\mathbf{H} = h_{1:12}$$

$$\mathbf{E} = e_{1:9}$$

$$\mathbf{F} = f_{1:4}$$

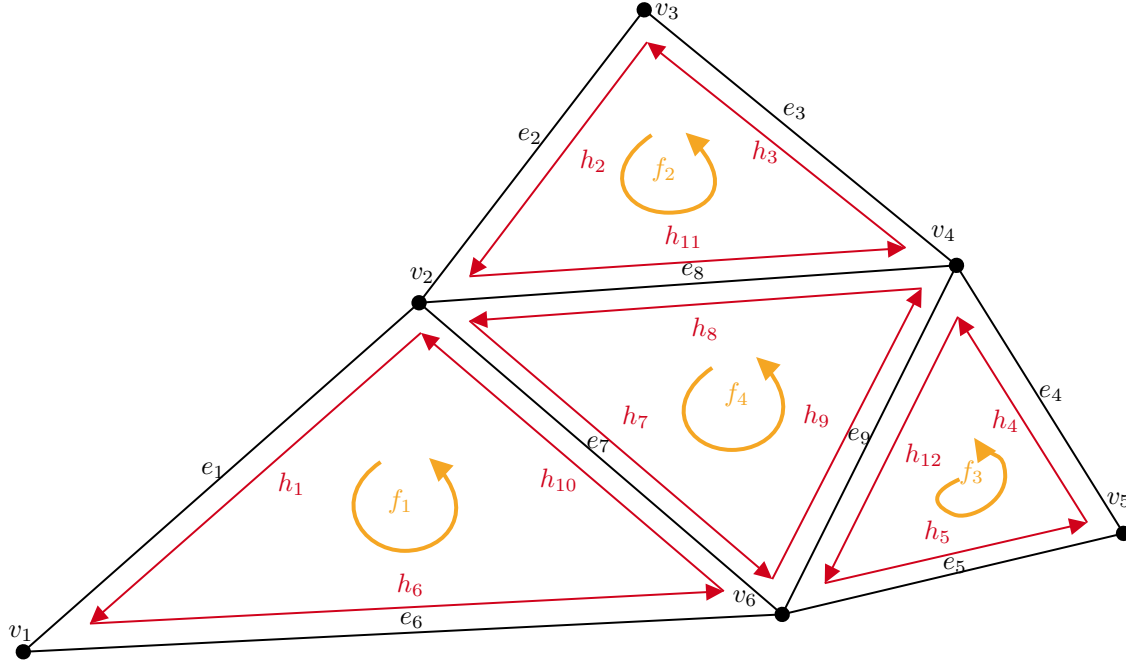


Figure 1: an example of half-edge data structure of a simplicial complex $\Sigma = f_1 + f_2 + f_3 + f_4$

and pointer functions :

1. $v(\cdot)$ is obvious to every half-edge in the picture, e.g. $v_{\text{sour}}(h_1) = v_2$ and $v_{\text{targ}}(h_1) = v_1$
2. $f(\cdot)$ is also obvious to every face in the picture, e.g. $f(h_1) = f(h_6) = f(h_{10}) = f_1$
3. $e(\cdot)$ in this picture is:

$$e(h_n) = \begin{cases} e_n & \text{if } n \leq 9 \\ e_{n-3} & \text{otherwise} \end{cases}$$

4. $h(\cdot)$ in this picture, firstly

$$h(e_n) = \begin{cases} h_n & \text{if } e_n \text{ on boundary} \\ (h_n, h_{n+3}) & \text{otherwise} \end{cases}$$

secondly we denote e.g. $h_{\text{next}}(h_1) = h_6$ and $h_{\text{prev}}(h_6) = h_1$ as $1 \rightarrow 6$, then we denote all $h : \mathbf{H} \rightarrow \mathbf{H}$ as

$$1 \rightarrow 6 \rightarrow 10 \rightarrow 1 \quad 2 \rightarrow 11 \rightarrow 3 \rightarrow 2 \quad 4 \rightarrow 12 \rightarrow 5 \rightarrow 4 \quad 8 \rightarrow 7 \rightarrow 9 \rightarrow 8$$

thirdly we set “the first half-edge” of each face and fourthly “the first half-edge” of each target vertex in this picture (note that these are merely arbitrary choices) as follows:

$$\begin{aligned} h(f_1) &= h_1 & h(f_2) &= h_2 & h(f_3) &= h_5 & h(f_4) &= h_8 \\ h(v_n) &= h_n \end{aligned}$$

Now we rephrase example 2 in discrete surface topology language.

2 Discrete Algebraic Surface Topology

Definition 3 (Simplex). Suppose $k + 1$ linear independent points embedded in \mathbb{R}^n

$$v_0, v_1, \dots, v_k$$

the standard k -simplex

$$[v_0, v_1, \dots, v_k]$$

is the minimal convex set including all of them.

e.g. in figure 2, the 2-simplex (vertices written counter-clock-wisely)

$$f_1 = [v_2, v_1, v_6]$$

the 1-simplex (vertices written from source to target)

$$h_1 = [v_2, v_1]$$

Definition 4 (Simplicial Complex). A *simplicial complex* Σ is an union of simplicies with “vertex auto-alignment”.

Example 5 (Simplicial Complex). See figure 2

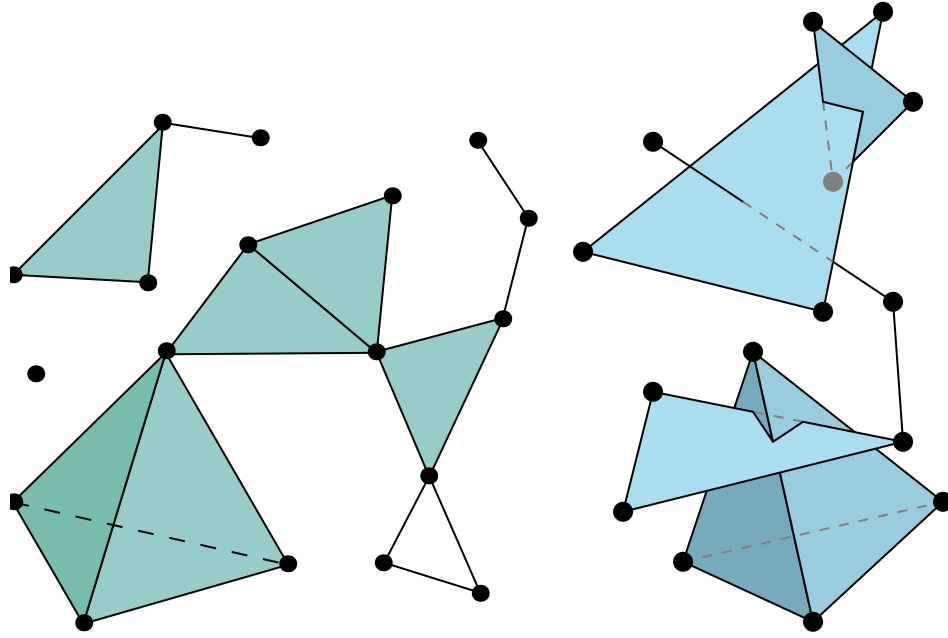


Figure 2: An example (left) and not-example (right) of simplicial complex

Definition 6 (Chain). A k -chain is a **linear combination (formal sum)** of all k -simplices in Σ .

Definition 7 (Chain Space). The k -dimensional *chain space* is the linear space formed by all k -chain by formal sum over \mathbb{Z} , denoted as

$$C_k(\Sigma, \mathbb{Z})$$

e.g. in figure 2,

- in $C_2(\Sigma, \mathbb{Z})$, a 2-chain could be f_1 , $f_1 + f_2$, or even $f_1 + 3f_2 - 4f_3$
- in $C_1(\Sigma, \mathbb{Z})$, a 1-chain could be $h_1 + h_2$, or even $2h_1 - 5h_2 + 3h_4$

Definition 8 (Boundary Operator). The discrete version of boundary operator ∂ , could be implemented as follows: we use hat notation as we delete \hat{v}_i from $[v_0, \dots, \hat{v}_i, \dots, v_k]$. Then

$$\partial[v_0, v_1, \dots, v_k] = \sum_{i=0}^k (-1)^i [v_0, \dots, \hat{v}_i, \dots, v_k]$$

e.g. in figure 2,

- $\partial f_2 = \partial[v_3, v_2, v_4] = [v_3, v_2] + [v_2, v_4] - [v_3, v_4] = h_2 + h_{11} + h_3$
- $\partial h_1 = \partial[v_2, v_1] = v_1 - v_2$

we would easily see that the boundary operator is linear, e.g.

$$\partial \Sigma = \partial(f_1 + f_2 + f_3 + f_4) = \partial f_1 + \partial f_2 + \partial f_3 + \partial f_4$$

and $\partial^2 = \mathbf{0}$, e.g.

$$\partial^2 f_2 = \partial(h_2 + h_{11} + h_3) = v_2 - v_3 + v_4 - v_2 + v_3 - v_4 = \mathbf{0}$$

Definition 9 (Cochain Space). By assigning a value to each k -simplex (opposite half-edges get same value with different symbols), one can get a k -dimensional cochain space formed by all k -cochain, which takes in a k -chain and outputs the summation of value of k -simplicies of the k -chain.

We define discrete version:

$$\begin{aligned} \text{closed 1-chain} &\longleftrightarrow \text{loop} \\ \text{exact 1-chain} &\longleftrightarrow \text{boundary} \\ \text{closed 1-cochain} &\longleftrightarrow \text{curl free field} \\ \text{exact 1-cochain} &\longleftrightarrow \text{gradient field} \end{aligned}$$

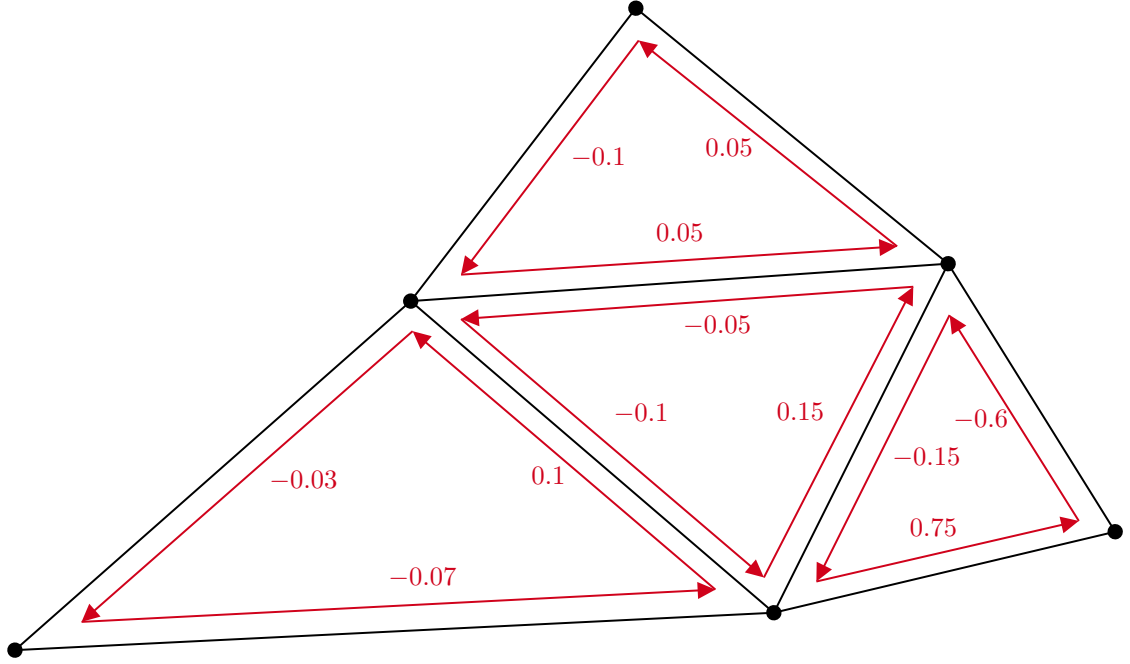
Example 10 (1-cochain). see figure 3, we have

$$w(h_2 + h_7) = w(h_2) + w(h_7) = -0.1 - 0.1 = -0.2$$

and one can see that any exact 1-chain γ in Σ , we have

$$w(\gamma) = 0$$

so w is an exact 1-cochain.

Figure 3: an example of 1-cochain w constructed from Σ

Recall that *graph* $G = (V, E)$, by its definition, is contained in a simplicial complex Σ .

Definition 11 (Dual Graph). The *dual graph* \bar{G} of a plane graph G , is a graph that

- has a vertex for each face of G
- has an edge whenever two faces of G are separated from each other by an edge, and a self-loop when the same face appears on both sides of an edge

Thus, each edge e of G has a corresponding dual edge \bar{e} in \bar{G} , whose endpoints are the dual vertices corresponding to the faces on either side of e .

Definition 12 (Spanning Tree). A *spanning tree* T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G , with a minimum possible number of edges.

3 Algorithms of $\pi_1(\Sigma)$, $\tilde{\Sigma}$, $H_1(\Sigma)$ and $H^1(\Sigma)$

We now introduce algorithms to compute

- $\pi_1(\Sigma)$, first homotopy group of Σ (algorithm 1)
- $\tilde{\Sigma}$, universal covering space Σ (algorithm 2)
- $H_1(\Sigma)$, first homology group of Σ (algorithm 3)
- $H^1(\Sigma)$, first cohomology group of Σ (algorithm 4)

represented by 1-chain or 1-cochain as group basis. For computation of the first homology group $H_1(\Sigma)$, it has the same bases with the first homotopy group $\pi_1(\Sigma)$. However, it is not the case in higher dimensions. For higher dimensional computation of homology group basis, please consult eigen-decomposition of combinatorial Laplace operator using Smith norm.

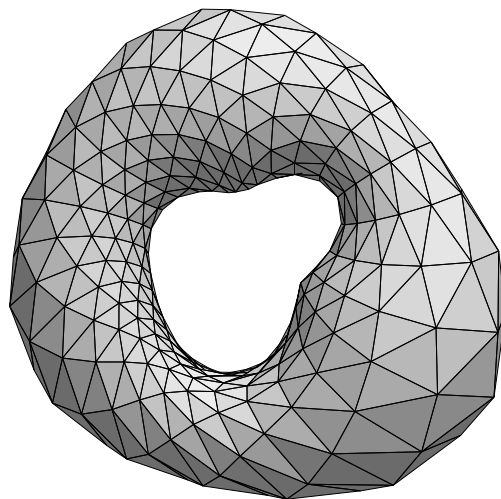


Figure 4: a triangulation mesh on a genus 1 surface

Algorithm 1: First Homotopy Group $\pi_1(\Sigma)$

Input: a closed triangular mesh Σ

Output: $\pi_1(\Sigma)$

1. compute the dual mesh $\bar{\Sigma}$ of the input mesh Σ
2. compute a spanning tree $\bar{\mathbf{T}}$ of $\bar{\Sigma}$, rooted at an arbitrary point p
3. the *cut graph* Γ of Σ is given by
$$\Gamma := \{e \in \Sigma \mid \bar{e} \notin \bar{\mathbf{T}}\}$$
4. compute a spanning tree \mathbf{T} of Γ
5. select an edge $e_i \in \Gamma \setminus \mathbf{T}$, then $e_i \cup \mathbf{T}$ gives an unique closed 1-chain γ_i ; suppose we get a set of distinct 1-chains
$$\{\gamma_1, \gamma_2, \dots, \gamma_k\}$$

which is the set of generators of $\pi_1(\Sigma)$

6. cut the mesh Σ along Γ to obtain $\tilde{\Sigma}_0$, the fundamental domain of Σ
7. set $R = \emptyset$, let $\gamma = \partial\tilde{\Sigma}_0$, traverse γ , once $e_i^{\pm 1}$ is encountered, append $\gamma_i^{\pm 1}$ to R ,

$$R \leftarrow R\gamma_i^{\pm 1}$$

8. the first homotopy group of Σ

$$\pi_1(\Sigma, p) = \langle \gamma_1, \gamma_2, \dots, \gamma_k \mid R \rangle$$

Algorithm 2: Universal Covering Space $\tilde{\Sigma}$

Input: a closed triangular mesh Σ

Output: an universal covering space $\tilde{\Sigma}$ with desired size

1. the same as step 1 \rightarrow 6 in algorithm 1
2. set $\tilde{\Sigma} = \tilde{\Sigma}_0$, glue a copy of $\tilde{\Sigma}_0$ with $\tilde{\Sigma}$ along γ_i , a homeomorphism $h : \partial\tilde{\Sigma} \supset \gamma_i \sim \gamma_i^{-1} \subset \partial\tilde{\Sigma}_0$

$$\tilde{\Sigma} \leftarrow \tilde{\Sigma} \cup_h \tilde{\Sigma}_0$$

3. trace the boundary of $\tilde{\Sigma}$, if there are two adjacent 1-chains $\gamma_j, \gamma_{j+1} \subset \partial\tilde{\Sigma}$, such that $\gamma_j^{-1} = \gamma_{j+1}$ then glue them together
 4. repeat step 2 and step 3, until $\tilde{\Sigma}$ is large enough
-

Algorithm 3: First Homology Group $H_1(\Sigma)$

Input: a closed triangular mesh Σ

Output: $H_1(\Sigma)$

1. the same as step 1 \rightarrow 5 in algorithm 1
 2. $H_1(\Sigma) = \{[\gamma_1], [\gamma_2], \dots, [\gamma_{2g}]\}$
-

Algorithm 4: First Cohomology Group $H^1(\Sigma)$

Input: a closed triangular mesh Σ

Output: $H^1(\Sigma)$

1. the same as step 1 \rightarrow 2 in algorithm 3
 2. for each γ_i , slice Σ along γ_i to obtain a mesh Σ_i with two boundaries. We have $\partial\Sigma_i = \gamma_i^+ - \gamma_i^-$
 3. set a 0-form τ_i on Σ_i such that $\tau_i(v^+) = 1$ for all vertices $v^+ \in \gamma_i^+$ and $\tau_i(v^-) = 0$ for all vertices $v^- \in \gamma_i^-$; set $w_i = d\tau_i$
 4. $H^1(\Sigma) = \{[w_1], [w_2], \dots, [w_{2g}]\}$
-