三次元形状のデータ構造と形状変形について

渡辺大地

1　主な三次元形状データ構造

名称	特徴
ワイヤーフレームモデル	頂点と稜線のみで表現される形状モデル。
サーフェスモデル	多角形平面あるいは単純な曲面の集合として表現される形状モデル。
ソリッドモデル	サーフェスモデルの特徴に加え、立体の内側・外側という概念を導入した形状モデル。実際には、次に挙げる CSG モデルと境界表現モデルに大別される。
CSG モデル	基本的な立体の集合演算によって形状を定義される形状モデル。
境界表現モデル	サーフェスモデルの延長上にあるモデルで、頂点・稜線・面の接続関係 (これを **位相情報** または単に **位相** という) を明確化した形状モデル。B-Rep モデルとも呼ばれる。
非多様体モデル	境界表現モデルを拡張し、二次元多様体でない形状も包含した形状モデル。
メタボールモデル	減衰影響関数の和によって定義される形状モデル。
ボクセルモデル	三次元格子状に並んだ直方体の集合によって定義される形状モデル。

2　典型的な形状データ表現

2.1　ポリゴンセット

構造解説:

平面によるサーフェスモデル。形状は複数の多角形の集合であり、多角形は複数の頂点の集合で表現される。

長所:

- 実装が容易。

短所:

- 頂点座標データに重複が起こるため、メモリ利用量が多い。
- 形状変形、形状分析は不可能。
- 多角形の頂点数が可変な場合、パフォーマンスが落ちる。

2.2　インデックスフェースセット

構造解説:

配列による頂点座標データの集合と、頂点番号による多角形データの集合に二つからなるソリッドモデル用データ構造。

長所:

- メモリ効率が良い。
- 頂点移動だけならば変形に適している。

短所:

- ポリゴンセットからの変換はやや面倒。
- 形状分析や、位相変化を伴う変形には不適。
- 多角形の頂点数が可変な場合、パフォーマンスが落ちる。

2.3 Half-Edge 構造

構造解説:

頂点、稜線、半稜線、ループの四種類の位相要素の集合によって表現するソリッドモデル用データ構造。

長所:

- あらゆる分析、変形に適している。
- 拡張性が高い。

短所:

- 他のデータ構造と比べると概念が高度。
- 多くの位相用データを保持するため、インデックスフェースセットと比較するとメモリ効率が悪い。(ポリゴンセットよりは良い。)

3 Half-Edge 構造

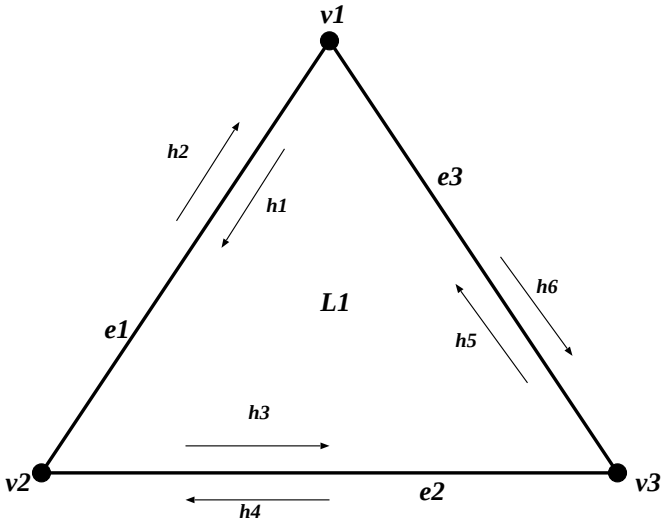

図 1: Half-Edge 構造の概念図

図 1 は、Half-Edge 構造によるデータ表現の概念図である。以下に各要素の簡単な解説を述べる。

名称	図中表記	解説	保持データ
頂点	v1 〜 v3	形状中の頂点を表す。単独で存在する場合と、稜線の端点となる場合がある。	• 頂点座標値。(必須) • 自身を始点とする半稜線一個。(適宜)
稜線	e1 〜 e3	形状中の面同士の境界線を表す。必ず両端点に頂点があり、それぞれを始点とする半稜線を一個ずつ持つ。	• 両側の半稜線。(必須)
半稜線	h1 〜 h6	稜線に二個ずつ属している要素で、稜線の端点のうちの片方を始点とする。	• 親稜線。(必須) • 始点頂点。(必須) • 前後の半稜線。(必須) • 属する面。(適宜)
ループ	L1	形状中の面を表す。複数の半稜線の連結によって構成される。	• 構成する半稜線一個。(必須)

問題 1

図 1 中の全位相要素に対し、保持データを挙げよ。

4 Half-Edge 構造の形状状態参照

Half-Edge 構造の特徴の一つとして、様々な位相要素の情報を形状全体の大きさとは関係ない処理数で得られることがある。たとえば、

半稜線 H の対になっている半稜線 H' を得たい ···

→ H の親稜線 E を得る。

→ E が保持する半稜線のうち、H でない方を得る。

ループ L の角数を得たい ···

→ L を構成する半稜線 H を得る。

→ H の次の半稜線 H' を得る。

→ H' が H と一致しない場合、さらに次の半稜線 H'' を得る。

→ 上記を繰り返し、H と一致した時点で出てきた半稜線の本数を数える。

問題 2

頂点に接続している稜線の本数を得る方法を述べよ。

5 Half-Edge 構造の形状変形

Half-Edge 構造の最大の特色はその変形の汎用性にある。概念的には、以下のような変形操作がある。

MakeVertex:
　何もないところに頂点を生成する。

RemoveVertex:
　稜線に接続していない頂点を削除する。

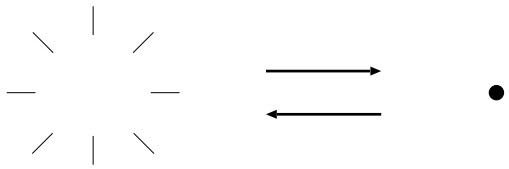

図 2: MakeVertex ・ RemoveVertex

MoveVertex:
　頂点を移動する。

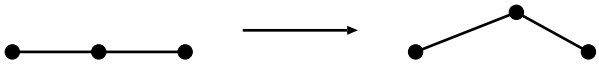

図 3: MoveVertex

MakeEdge:
　二つの頂点間に稜線を生成する。

DeleteEdge:
　稜線を削除する。

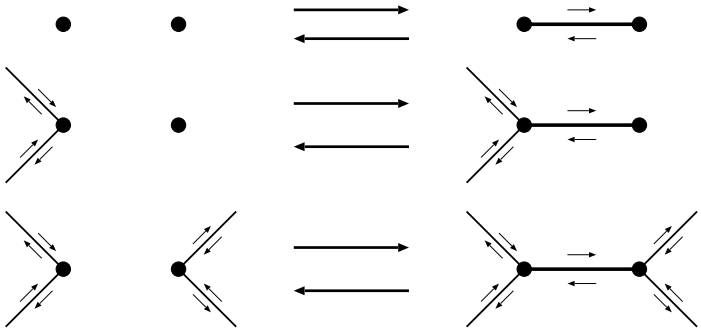

図 4: MakeEdge ・ DeleteEdge

MakeLoop:
ループを生成する。

DeleteLoop:
ループを削除する。

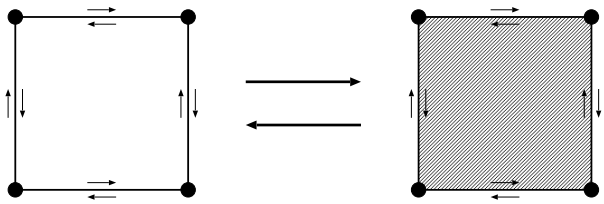

図 5: MakeLoop ・ DeleteLoop

SeparateLoop:
ループを稜線で分割する。

UniteLoop:
二つのループ間にある稜線を削除し、一つのループにする。

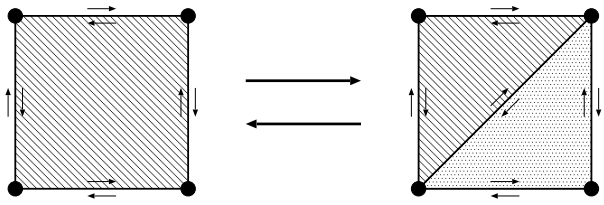

図 6: SeparateLoop ・ UniteLoop

SeparateEdge:
稜線の中点に新たな頂点を生成する。

UniteLoop:
二本の稜線に接続している頂点を削除し、一本の稜線にする。

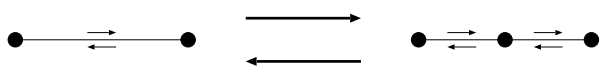

図 7: SeparateEdge ・ UniteEdge

6 サンプルプログラム

```
 1: #include <FK/FK.h>
 2:
 3: int main()
 4: {
 5:     fk_ShapeViewer  viewer(300, 360);
 6:     fk_Solid        solid;
 7:     fk_Vector       pos1, pos2, pos3;
 8:     fk_Vertex       *v1, *v2, *v3;
 9:     fk_Edge         *e1, *e2, *e3;
10:     fk_Half         *h1, *h2, *h3, *h4;
11:
12:     // 各頂点座標設定
13:     pos1.set(0.0, 10.0, 0.0);
14:     pos2.set(-10.0, -10.0, 0.0);
15:     pos3.set(10.0, -10.0, 0.0);
16:
17:     // 各頂点生成
18:     v1 = solid.makeVertex(pos1);
19:     v2 = solid.makeVertex(pos2);
20:     v3 = solid.makeVertex(pos3);
21:
22:     // e1 の生成と h1, h2 の取得
23:     e1 = solid.makeEdge(v1, v2);
24:     h1 = v1->getOneHalf();
25:     h2 = v2->getOneHalf();
26:
27:     // e2 の生成と h3, h4 の取得
28:     e2 = solid.makeEdge(v2, v3, h1, h2);
29:     h3 = h1->getNextHalf();
30:     h4 = h2->getPrevHalf();
31:
32:     // e3 の生成
33:     e3 = solid.makeEdge(v3, v1, h3, h4, h2, h1);
34:
35:     // ループの生成
36:     solid.makeLoop(h1);
37:
38:     // fk_ShapeViewer へ形状を設定
39:     viewer.setShape(0, &solid);
40:
41:     // 各種設定
42:     viewer.setDrawMode(FK_POLYMODE | FK_LINEMODE | FK_POINTMODE);
43:     viewer.setScale(10.0);
44:
45:     while(viewer.draw() == true) {}
46:
47:     return 0;
48: }
```

問題 3

サンプルコードと FK システムのマニュアルを参照し、変形機能を用いて (1) 正方形、(2) 三角錐、(3) 立方体、(4) 四角い貫通穴を持つ立方体を作成せよ。