2004－DBS－134　(II)　(50)
2004／7／14

符号ベクトルに基礎を置く 空間データベースシステム Hawk's Eye のデータの圧縮

陸 応亮†　金子 邦彦‡　田中 美智子†　牧之内顕文‡

†九州大学 大学院 システム情報科学研究院　〒812-8581 福岡市東区箱崎 6-10-1

E-mail: †{ riku, tanaka } @db.is.kyushu-u.ac.jp, ‡{ kaneko, akifumi } @is.kyushu-u.ac.jp

あらまし 本論文では，我々が研究開発を行ってきている空間データベース基盤システム Hawk's Eye の空間データモデルの符号ベクトルの格納法，符号ベクトルと位置ベクトルの相互変換法，符号ベクトル圧縮法の詳細設計と実装について述べる．データベース上では，超平面，0 次元の face，independent HA-face（HA-face complex を face をノードとするような lattice 構造で表現したときの極大元の face のこと）の 3 つのデータの集まりとして表現できる．0 次元の face と independent HA-face には，+，-，0 からなる位置ベクトル(Position Vector)があるが，位置ベクトルをそのままデータベースに格納するのは無駄が多い．我々は，次元が 1 以上であるような independent HA-face の位置ベクトルのうち冗長な+と-を，符号 r で置き換え，0 次元の face の位置ベクトルの中の+と-を全て符号 d で置き換えた符号ベクトル(Sign Vector)の考え方を導入し，可能な限りコンパクトに HA-face complex を格納する方式を考案した．それについてのアルゴリズムを実装して，空間図形の VRML，DNF 表現のデータサイズと比較した。本論文では，単純に位置ベクトルを格納した場合と，圧縮される符号ベクトルを格納した場合とのデータサイズの比較も行う．

キーワード 空間 DB,超平面アレンジメント,空間モデル,データ圧縮

Reducing Data Size of Spatial Database Using Sign Vectors

Yingliang Lu†　Kunihiko KANEKO‡　Michiko Tanaka†　and　Akifumi MAKINOUCHI‡

†Graduate School of Information Science and Electrical Engineering, Kyushu University 6-10-1 Hakozaki,

Higashi-Ku Fukuoka, 812-8581 Japan

E-mail: †{ riku, tanaka } @db.is.kyushu-u.ac.jp, ‡{ kaneko, akifumi } @is.kyushu-u.ac.jp

Abstract In this paper, we describe how to effectively express spatial geometric data in spatial database systems. Hyperplane is used to expresses the position and forms of a spatial object of various dimensions. The hyperplanes are used to split a complex of convex polytopes into HA-faces. Each HA-face has its position vector, and the position vectors are used to evaluate the spatial operations: intersection, the union and the difference of complexes of convex polytopes in any dimension. First, we introduce the sign vector, which is the database representation of the position vector. Second, we present the conversion algorithms between position vectors and sign vectors of the complex model in any dimension. Third, we propose algorithms for compressing and uncompressing the sign vector to be stored in the database data. These algorithms are invoked when evaluating spatial operations. A spatial data model named HA-face-complex model is implemented on an object database system based on the sign vector. We also give the results of experimental tests to evaluate the data size of spatial database and speeds up the geometric operations.

Keyword spatial database, hyperplane arrangement, spatial data model, convex polytope, data compression

1．はじめに

空間データベースの重要性は高く，種々の研究が行われてきた[1][2][3][10][11]．我々は，この状況を踏まえ，空間データベース構築のための基盤システム Hawk's Eye の研究・開発を行ってきた．Hawk's Eye は，多角形や立体などの広がりを持った空間物を含む種々の次元の空間データを扱う空間データベースを，容易に構築できるようにするための基盤ソフトウエアシステムであり，空間データの格納・幾何計算・検索に関する種々の基本機能を提供する．Hawk's Eye には，新しい空間データモデルとして，HA-face complex モデルが実装されている．このモデルは，次元に特化しない空間データモデルであって，本質的に，任意次元の空間内の，任意次元の空間物を扱うことができる．

空間モデル HA-face complex[12],weak-complex[10]のディスク上でのサイズが重要な研究課題である．HA-face complex を格納するのに必要なディスク量と，ディスク上の HA-face complex を読み書きする I/O コストはサイズに比例して増大するので，ディスク上では，可能な限りコンパクトな形式で格納しておくことは重要である．空間図形の幾何計算の Localized divide-and-conquer アルゴリズム[12]では，各 face

に対して付けられた+，-，0 からなる位置ベクトル(Position Vector)を使う．位置ベクトルの長さは，空間物を構成する超平面の数である．位置ベクトルの大部分は+，-であり，0 の登場数は少ないが，+と-の登場はランダムであるため，符号語の登場頻度の偏りを利用して圧縮するという，情報理論的なアプローチを使っての圧縮はこのままでは困難である．

HA-face complex の各 face には，本来，交差する超平面と交差しない超平面とがある．交差しない超平面は，本質的には，localized divide-and-conquer アルゴリズムの処理には影響を与えない．現在までの実験と考察の結果，交差する超平面と比べて，交差しない超平面の方が大部分を占めることから，無駄なデータベース I/O 処理に時間の多くが費やされていることが分かってきた.この課題を解決するために，「データベースの格納では，交差する超平面と交差しない超平面を区別して格納する」ことを着想した．

本論文では，データベース内への符号ベクトルの格納法，符号ベクトルと位置ベクトルの相互変換法，単純に位置ベクトルを格納した場合と，符号ベクトル格納した場合とのデータサイズの実験で比較を行う．本論文の構成は以下の通りである．2 章では，HA-face complex と位置ベクトルを簡単に説明しながら，符号ベクトルの考え方を導入する．3 章では，符号ベクトルと位置ベクトルの相互の変換について,幾何計算アルゴリズムとの関係に触れながら説明する．4 章は，HA-face complex のデータサイズの比較実験を行い，単純に位置ベクトルを格納した場合と，符号ベクトルを格納した場合を比較する．

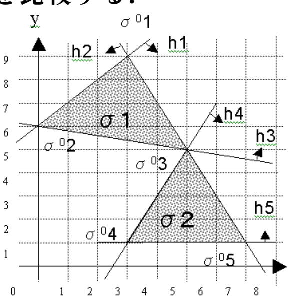

図１．HA-face complex の例

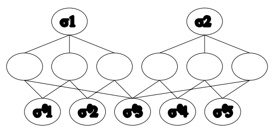

図２．図１の HA-face complex のグラフ表現

(a) 0-HA-face の位置ベクトル

0-HA-face	位置ベクトル
$\sigma^0 1$	[0 0 + - +]
$\sigma^0 2$	[0 + 0 - +]
$\sigma^0 3$	[+ 0 0 0 +]
$\sigma^0 4$	[+ + - 0 0]
$\sigma^0 5$	[+ 0 - + 0]

(b) Independent HA-face の位置ベクトル

	位置ベクトル
σ1	[+ + + - +]
σ2	[+ + - + +]

表１．図１の HA-face complex の位置ベクトル

2. HA-face complex と符号ベクトル

2.1. HA-face complex と符号ベクトル

空間データベースの空間データを表現するために、種々のデータモデルとデータファイルが提案されている現状である．例えばデータファイルとしては VRML, 3DS などがあり，データモデルとしては DEDALE [9], cell complex[11, 12], weak complex[10]などがある．我々の研究室において研究開発されてきた空間データモデル HA-face complex は，任意次元の空間図形を表現することできる, DEDALE より高速に幾何計算を行うことが可能である。

HA-face complex[12] の各 HA-face は，超平面のとの位置関係により位置ベクトル(Position Vector)を持つ．各超平面は，φ=0 という形式の線形式（例えば図１の超平面 h1 は，x-y+6=0）であり，φ>0 である領域が表側，φ<0 である領域が裏側である．位置ベクトルでは HA-face が超平面の表側にあるときに+を，裏側にあるとき-を,超平面上にあるときに 0 を割り当てる．位置ベクトルの長さは，HA-face complex を表現する超平面の数である.位置ベクトルは,定義から次のような性質を持つ．

- N 次元空間では，ある 0 以上から N-2 次元以下の HA-face を包含する超平面は，最低でも N-k 個あり，N-k 個以上のこともありえる．従って，その HA-face の位置ベクトルは N-k 個以上の 0 を持つ．
- N 次元空間での N-1 次元の HA-face は，1 つの HA-face にしか包含されない．従って，位置ベクトルは 1 個の 0 を持つ．
- N 次元空間での N 次元の HA-face は，いかなる超平面とも交差しない．従って，位置ベクトルは 0 を含まない．

位置ベクトルは，Localized divide-and-conquer アルゴリズムで使用されるが，位置ベクトルをそのままの形で，データベースに格納するのは得策ではない．位置ベクトルの中の+，-，0 のうち，+，-の中には，HA-face の表現にとっては冗長なものがある．例えば，図 1 の HA-face σ1 の表現には，σ1 を取り囲んでいる h1, h2, h3 の 3 つの超平面に対する+，-値 [+ + +] だけで十分であり，その意味で，σ1 と接しているだけであり，σ1 とは離れている超平面 h4 と h5 に対する分は冗長である．一方，0-HA-face の位置ベクトルでは 0 だけが必要である.例えば,図 1 の $\sigma^0 1$ では, h1 と h2 だけが必要であり,他の超平面は不要である．以上のアイデアから，次に定めるような符号ベクトル(Sign Vector)を導入する．

1 次元以上の HA-face に対して

位置ベクトルの+，-，0 のうち冗長な+と-を r で置き換えたものを符号ベクトルとする．符号ベクトルは，+，-，0，r の 4 つの符号(sign)から構成される．k 次

元 HA-face (k≧1) の符号の意味は次のように定める

+ の意味

HA-face に接続していて次元が 1 つ低い HA-face が，対応する超平面の上にあり，かつ HA-face 自身が超平面の表側にある．

- の意味

HA-face に接続していて次元が 1 つ低い HA-face が，対応する超平面の上にあり，かつ HA-face 自身が超平面の裏側にある．

0 の意味

HA-face は，対応する超平面に包含されている．

r (redundant) の意味

上記のいずれにも当てはまらない．つまり，HA-face の表現にとって冗長である

0 次元の HA-face に対して

位置ベクトルの+,-,0 のうち+と-を全て d で置き換えたものを符号ベクトルとする．0 次元 HA-face の符号ベクトルは，0, d の 2 つの符号から構成される．それぞれの符号の意味は，0 次元 HA-face と，対応する超平面の位置関係により，次のように定める．

0 次元 HA-face の符号(sign)

0 の意味

0-HA-face が，対応する超平面の上にある．

d (disjoint) の意味

0-HA-face が，対応する超平面の上にない．

表 2. 図 1 の HA-face complex の表現

(a) 0-HA-face

0-HA-face	座標	符号ベクトル
$\sigma^0 1$	(3, 9)	[0 0 d d d]
$\sigma^0 2$	(0, 6)	[0 d 0 d d]
$\sigma^0 3$	(5, 5)	[d 0 0 0 d]
$\sigma^0 4$	(3, 1)	[d d d 0 0]
$\sigma^0 5$	(7, 1)	[d 0 d d 0]

(b) Independent HA-face

	符号ベクトル	接続関係
σ1	[+ + + r r]	[1 2 3]
σ2	[r + r + +]	[3 4 5]

(c) 超平面の方程式

h1	x - y + 6 = 0
h2	2x + y - 15 = 0
h3	x/5 + y - 6 = 0
h4	2x - y - 5 = 0
h5	y - 1 = 0

2.2. 0-HA-face の符号ベクトルのデータベース上での表現

本研究での中心となる課題は,表 2 のような HA-face complex について,データベース上にいかなる形式で格納することが最も有利であるかを検討することにある．符号ベクトルでは，大半が r あるいは d になり，+，-，0 の頻度は少ないという性質を持つので，簡単に圧縮することができる．3.1 節で議論するように 1 次元以上の HA-face の符号ベクトルへ変換して 3.3 節の圧縮アルゴリズムで圧縮してデータベースに格納する．N 次元空間中の 0 次元 HA-face の符号ベクトルは，N 個とは決まっておらず,N 個以上の 0 を持つことから，0 次元 HA-face の格納法としては N 個を超える 0 があった場合も，すべて 0 である超平面の番号をデータベースに格納する．

3. 計算アルゴリズムの振る舞い

3.1 位置ベクトルの処理

位置ベクトルを，符号ベクトルに変換する処理を行う．処理の順序としては，まず，0 次元 HA-face を先に行う．0 次元 HA-face では，符号は 0, d の 2 通りであり，位置ベクトルの+, - 値を全て d に変換するという操作を行う．1 次元以上の HA-face については，図 2 のような HA-face complex のグラフよりの接続関係と前にとった接続している 0 次元 HA-face 符号ベクトルより図 4 に示したアルゴリズムで行う．

図 3 0 次元 HA-face を位置ベクトルから符号ベクトルへの変換アルゴリズム

```
Algorithm make0SignVector(P)
Input:  0_HA-face の位置ベクトル P=[p1 … pk]
output: 0_HA-face の符号ベクトル S=[s1 … sk]

   1. for i ← 1 to k
   2.  do if pi=0
   3.     then si ← 0
   4.     else si ← ”d”
   5. return S
```

図 4 1 次元以上の HA-face を位置ベクトルから符号ベクトルへの変換アルゴリズム

```
make_sign_vector_R(P,dim,NIDSet)
   入力：independent HA-face の位置ベクトル
        P=[p1・・・pNh]
        independent HA-face の次元 dim
        接続する 0-HA-face の番号の集合 NIDSet
   出力：independent HA-face の位置ベクトルで，冗長
        な部分を r で置き換えたベクトル
   1.for i ←1 to Nh
   2. do if  is_num0_more_than_face_dim(
                 dim,i,NIDSet) = false
   3.       then pi ← “r”
```

指定超平面と該当 k-HA-face との接続関係の判定アルゴリズム

```
is_num0_more_than_face_dim(dim,hid,NIDSet)
  入力：independent HA-face の次元 dim
        超平面番号 hid
        0-HA-face の番号の集合 NIDSet
  出力：超平面 hid に対する符号が”0”である
        0-HA-face が dim 個以上あれば true，なければ
        false を返す
 1.NIDSet の要素をキューQ に入れる
```

```
  2.count ← 0
  3.while Q≠φ
  4.  do Qの先頭の要素を取り出しkとする
  5.番号kの0-HA-faceの位置ベクトルをP=[p1・・・
    pNh]とする
  6.      if  Phid = “0”
  7.        then  count ← count + 1
  8.            if count ≧ dim
  9.              then  return true
  10.return false
```

更に，データベース上には，同じ符号ベクトルを複数保存することはしないので，同じ符号ベクトルがあった場合は，その中の１つだけを保存する．また，この時に，接続関係を書き換える処理も行う．例えば，下記の$\sigma^0 8$, $\sigma^0 11$は同じなので，[d 0 0 0 d d d d]を一つだけ保存する．

$\sigma^0 8$の符号ベクトル[d 0 0 0 d d d d]

$\sigma^0 11$の符号ベクトル[d 0 0 0 d d d d]

以上で，independent HA-faceと0-HA-faceの符号ベクトルが作成される．出力は，表 4の様になる．

3.2符号ベクトルから位置ベクトルへ

図形のintersection，unionなどの幾何計算のために， faceを分割[5,10]しなければいけないので，データベースに格納されているk-HA-faceと0-HA-faceの符号ベクトルから位置ベクトルへの変換が必要になる。例えば，図5のように、既にデータベースに格納された図形１と別の図形との intersection を求めるときに，データベースに格納された図１の符号ベクトルは位置ベクトルに戻さないとはいけない。我々はそれについてのアルゴリズムも実装した。

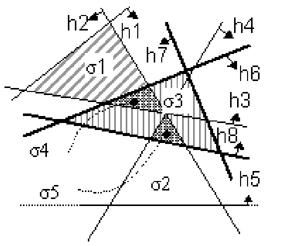

図 5　図形の intersection

図形を構成するHA-faceの符号ベクトルを位置ベクトルへ変換する処理の順序としては，まず，k 次元HA-faceを先に行う．符号 r ではない部分だけを抽出する．それから，抽出されたk次元HA-faceに 対応する超平面集合に対してk個以上0がある0次元HA-faceを抽出する。0 次元 HA-face の符号が d であれば、k次元HA-faceの位置ベクトルの対応する+, - 値に変換するという操作を行う.具体的には,図7に示したアルゴリズムで行う．図形のintersection，unionなどの幾何計算に関するフェースに接続している超平面を$[h_1 \cdots h_n]$とすると，k-HA-face の場合，符号ベクトルから位置ベクトルのアルゴリズムは図６のように行う．

図 6符号ベクトルから位置ベクトルのアルゴリズム

```
Algorithm getIndeFaceVector(S, H, P)
Input:
  k-HA-faceの符号ベクトル S=[s1 … sk]
Output:
  k-HA-face の位置ベクトル P=[p1 …pn]
  k-HA-face に接続している超平面集合P
H=[h1 …hn]
    1. for i ← 1 to k
    2.  do if si  is not ‘r’
    3.    then pj ← si
    4.    hj ← i add the I hyperplane into H
    5. return H
```

k-HA-face の位置ベクトルを取得した後，0 次元の HA-face の符号ベクトルを位置ベクトルへ変換するアルゴリズムは図７のようになる．

図 7　1次元以上のHA-faceを符号ベクトルから位置ベクトルへの変換アルゴリズム

```
Algorithm get0IndeFaceVector(S, H, P)
Input:
  0-HA-faceの符号ベクトル S=[s1 …sn]
  k-HA-face に接続している超平面集合P  H=[h1 …hk]
Output:
  0-HA-face の位置ベクトル P=[p1 … pk]
    1.        for i ← 1 to k
    2.          do if the all values of the s[hi] is
   "d"
    3.          then return that the 0_HA-face’s
   sign vector is not in needed
    4.          else if  s[hi]  is not ‘0’
    5.          then pi ←s[hi]
    6.        return P
```

このように，independent HA-face の符号ベクトルを位置ベクトルに戻して，independent HA-face の位置ベクトルにLocalized divide-and-conquerアルゴリズム[12]を用いる幾何計算をすれば，表３に示したように、図１のσ1とσ3のintersection部分である、新しいindependent HA-face σ4，図１のσ2とσ3のintersection部分であるσ5がそれぞれ得られる．

表 3. Intersection でのHA-faceの位置ベクトル例

(1) independent HA-face

Independent HA-face	位置ベクトル
σ4	[+ + + + + +] (h1,h2,h3,h6,h7,h8)
σ5	[+ + + + + +] (h2,h4,h5,h6,h7,h8)

(2) 0-HA-face

0-HA-face	位置ベクトル
$\sigma^0 6$	[+ 0 + 0 + +] (h1,h2,h3,h6,h7,h8)
$\sigma^0 7$	[+ + 0 0 + +] (h1,h2,h3,h6,h7,h8)
$\sigma^0 8$	[+ 0 0 + + +] (h1,h2,h3,h6,h7,h8)
$\sigma^0 9$	[0 + + + + 0] (h2,h4,h5,h6,h7,h8)
$\sigma^0 10$	[+ 0 + + + 0] (h2,h4,h5,h6,h7,h8)
$\sigma^0 11$	[0 0 + + + +] (h2,h4,h5,h6,h7,h8)

3.3Independent HA-face の符号ベクトルのデータベース上での表現

表３の位置ベクトルに対応する符号ベクトルは表４に示したようである.

表 4. Intersection の計算結果例

(1) independent HA-face

Independent HA-face	符号ベクトル
σ4	[r + + r r + r r]
σ5	[r + r + r r r +]

(2) 0-HA-face

0-HA-face	符号ベクトル
$σ^0$6	[d 0 d d d 0 d d]
$σ^0$7	[d d 0 d d 0 d d]
$σ^0$8	[d 0 0 0 d d d d]
$σ^0$9	[d 0 d d d d d 0]
$σ^0$10	[d d d 0 d d d 0]
$σ^0$11	[d 0 0 0 d d d d]

このような符号ベクトルをデータベースに格納すれば、無駄なｒ，ｄが多いので，それを圧縮してデータベースに格納する．０次元の HA-face では,0 に対応する超平面番号だけを格納する．そのアルゴリズムは以下のようになる.

図 8 0-HA-face 符号ベクトルの圧縮アルゴリズム

```
Algorithm encode_0_HA_face_sign_vector(S, ES)
Input:
    S=[s1 … sk], 0-HA-faceの符号ベクトル
Output:
    ES=[es1 … esk] 圧縮された 0_HA-faceの符号ベクトル
    圧縮されたベクトルの長さES

    1. for i ← 1 to k
    2.   do if si is the sing-0
    3.     then es[j++] ← i
    4. return j
```

１次元以上 の HA-Face のデータ格納法は,以下で説明する単純な方式を考えている．Independent HA-face の符号ベクトルは，0，+，-，ｒの４通りであるが，大部分が ｒであるので，連続している同じ符号数と符号を覚えるという方針である.

図 9 k-HA-face 符号ベクトルの圧縮アルゴリズム

```
Algorithm encode_Ind_HA_face_sign_vector(S, ES)
Input:
    S=[s1 … sk], k-HA-face の符号ベクトル
Output:
    ES=[es1 … esk] 圧縮された k_HA-face の符号ベクトル
    圧縮されたベクトルの長さ ES1.
1.  for i ← 1 to k2.
2.  do if s[i] != s[i+1] or i=k3.
3.    then the same number of signs ← count C4.
4.  es[j++] ← the same number of signs * 4 +si5.
5.  count ← 06.
6.      else count ← count+17.
7.  return j
```

4．実験

データ圧縮の効果を検証するために，我々は，既存の方法である VRML，DNF，位置ベクトルの HA-face complex と,提案の圧縮された符号ベクトルの HA-face complex データサイズの比較を行った.

4.1実験データ

HA-face complex の特質から言って，立体や領域図形など，広がりを持った図形の場合でも，次元に関係なく同様の傾向を示すと考えるので，我々は３次元の球面ポリゴンを実験データとして実験を行った．まずランダムに球面上に点を取り，取得した点より超平面（三次元の場合はフェース）の方程式を構成する．得られた超平面の方程式より球体の HA-face complex を構成することができる．今回はこの HA-face complex の超平面の方程式，face 数と 0-face 数を実験データとして，圧縮効果を測った。

4.2実験方法

4.1 章で得られた超平面の方程式，face 数と 0-face 数を実験データとしてそれぞれの空間モデルのデータサイズ計算式（仮）に代入し，３次元球体の HA-face complex データサイズの計算を行う。4.3 章に同じ図形を違うモデルで表現した場合のデータサイズを比較する.

まず HA-face Complex のサイズ計算式は次のように考えた.

数式 1 符号ベクトルの HA-face Complex のサイズ

データサイズ=
超平面方程式サイズ ＋ 「face」サイズ ＋「0-face」サイズ
=「超平面数」×N×(N+1) ×sizeof(double)
＋「face 数」×a×sizeof(int)
＋「0-face 数」×N×sizeof(int)

N: 空間物の次元数
a: 超平面数と冗長によって変る，1<a<1face 接している超平面数

一方，位置ベクトルをそのままの形でデータベースに格納した場合には，HA-face Complex のサイズは次のように計算することができる.

数式 2 位置ベクトルの HA-face Complex のサイズ

データサイズ=
超平面方程式サイズ ＋ 「face」サイズ ＋「0-face」サイズ
=「超平面数」×N×(N+1) ×sizeof(double)
＋「face 数」×「超平面数」×sizeof(int)
＋「0-face 数」×「超平面数」×sizeof(int)

N: 空間物の次元数

VRML 表示法と DEDALE の DNF 表示法で表現する空間図形のデータサイズは以下のような計算式から得られ

る．

数式 3　VRML モデルの仮計算式

データサイズ＝
点座標サイズ+
フェイス数×1 つフェースの頂点数×sizeof(int)

数式 4　DEDALE の DNF モデルの仮計算式

データサイズ＝
超平面方程式サイズ ＋ 図形の DNF サイズ

超平面方程式サイズ＝「超平面数」×N×(N+1) sizeof(double)　+「超平面数」×sizeof(int)}
図形の DNF サイズ＝(超平面番号+符号)×「超平面数」
=(sizeof(int)+ sizeof(int)) ×「超平面数」

4.3実験結果

4.1 章の実験データより得られる超平面数,頂点数とフェース数を 4.2 章の式に代入して計算した．計算結果は図 10 に示しているように,フェースの増加とともに前者の効果が良くなる．以上のように，圧縮効果が確認できた．

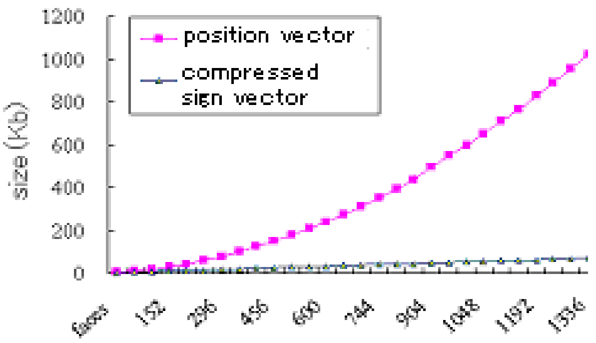

図 10 データサイズの比較

同じ実験データで，我々は VRML，DNF でデータベースに格納する場合と、圧縮された HA-face complex でデータベースに格納する場合のデータサイズを比較した． 圧縮された HA-face complex と DNF のデータサイズの差はない．VRML よりはデータサイズは大 きいが，それほど大差が無いことが分かった．

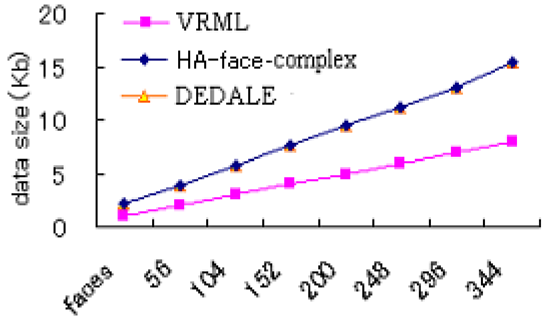

図 11 各データモデルデータサイズの比較結果

5．おわりに

本論文では，空間データベース Hawk’s Eye の空間データモデル HA-face complex のデータベース上でのデータ圧縮について報告を行った．交差しない超平面群と交差する超平面群を区別するビット列（符号ベクトル）を導入した．このことは，冗長な超平面を区別してデータベースに格納するために必要であった．以上の結果,データベースサイズの削減が可能になったのが本研究の成果である．

今回提案の方式は，データ量の削減に効果があることを実験で示した．

参　考　文　献

[1] Agnes Voisard,Benoit David:”A Database Perspective on Geospatial Data Modeling”,IEEE TKDE,Vol.14,No.2,pp.226-243,2002.

[2] Grunback,S.,Rigaux,P.,and Segoufin,L:”The DedaleSystem for Complex Spatial Queries”,Proc.1998 SIGMOD,pp.213-224,1998.

[3] R.H.Guting:”An Introduction to Spatial Database Systems” VLDB Journal,vol.3,no.4,pp357-400,1994.

[4] David P.Dobkin,and Ayellet Tal,Efficient and Small Representation of Line Arrangements with Applications,ACM Symposium on Computational Geometry,pp.293-301,2001

[5] H. Edelsbrunner and J.O'Rourke and R. Seidel, Constructing arrangements of lines and hyperplanes with applications, SIAM J. Comput. vol.15, pp.341-363. 1986.

[6] Herbert Edelsbrunner, Algorithms in Combinatorial Geometry", Springer-Verlag, 1987.

[7] J.E. Goodman and J.O’Rourke,editors.Handbook of Discrete and Computational Geometry,CRC Press LLC,BocaRaton,FL,1997.

[8] Handbook of Computational Geometry, Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000

[9] Philippe Rigaux, Michel Scholl, Luc Segoufin & Stéphane Grumbach, Building a constraint-based spatial database system: model, languages, and implementation, Information Systems archive Volume 28, 563 - 595, 2003

[10] Chandrajit L. Bajaj, Valerio Pascucci, Splitting a Complex of Convex Polytopes In Any Dimension, Computational Geometry'96, USA, 1996

[11] CHAZELLE, B. An optimal algorithm for intersecting three-dimensional convex polyhedra. SIAMJ. Comput. 21, 4 (1992), 671-696.

[12]金子邦彦，牧之内顕文，”超平面アレンジメントに基づく多次元空間幾何アルゴリズムの実装と評価”，情報処理学会研究報告 2003-DBS-131，pp.219-226，2003．