グラフィクスとCAD 81－3
（1996. 8. 22）

リアルタイムソリッドモデラに関する研究

床井浩平
和歌山大学経済学部
kohe-t@eco.wakayama-u.ac.jp

スキャンライン法による多面体の隠れ面消去処理アルゴリズムを用いた集合演算表示は，集合演算による形状変更の結果をほぼリアルタイムに画面に表示できる高速性を持つ．しかし，この方法では描画時に何度もブール結合の記述を参照するため，これを木構造やパターンマトリクスを用いて記述した場合，一つの形状を構成するプリミティブの数が増すにしたがって，集合演算処理による可視面の判定に時間がかかるようになる．本研究ではブール結合のうち積を省き，和と差のみによる形状定義を用いることによって，処理時間が一つの形状を構成するプリミティブの数に依存しない集合演算処理の実現を目指す．

Study of Real-Time Solid Modeler

Kohe Tokoi
Faculty of Economics
Wakayama University

A scan-line hidden surface removal procedure for constructive solid geometry (CSG) has ability of real-time display of changes of the shape by Boolean operation. But in this method, it becomes to take a long time to decide visible surface with increasing number of primitives that compose one object — because this procedure refers the description of Boolean combination many times at processing. This study tries to describe of Boolean combination using only "union" and "subtraction". This aims at the realization of Boolean operation which processing time doesn't depend on the number of the primitives included one object.

1．はじめに

境界表現を内部表現に持つソリッドモデラにおいても，集合演算による形状定義は，会話的な形状モデリング作業において複雑な形状を定義するのに有効な手段として使われている．

一般にこの方法では，対象形状を空間中で位置決めした後，ブール結合の記述に基づいて集合演算処理を実行し，目的形状を得る．こうして得られた形状データは画面表示などを行って確認するが，その結果それが意図したものと異なることが判明した場合には，一度集合演算処理を取り消して以前のデータを回復しなければならない．

そこで，形状データに対して実際に集合演算処理を適用する前に，ブール結合の記述から直接集合演算表示を行うことが考えられる．これには一般に視線探索法が用いられる他，一旦空間格子表現や八分木表現に変換して表示する方法などが考えられるが，いずれの方法も形状の確認だけを目的とした場合には計算コストが高い．これに対しスキャンライン法による集合演算表示は，これらに比べて効率的に陰影画像を生成でき，形状把握という目的に適した手法だと考えられる．

しかし，この方法は画像生成時にブール結合の記述を繰り返し参照するため，一つの形状を構成するプリミティブが多くなるにつれて，集合演算処理による可視面の判定に時間がかかるようになる．それゆえ，これまで用いてきた高速化手法は，この集合演算処理の回数の削減を目的としたものが主体であった [1] [4] [5] [6] [7].

本研究では，ブール結合のうち和と差のみを対象にすることによって，処理時間が一つの形状を構成するプリミティブの数に依存しない集合演算処理アルゴリズムを提案する．これにより，プリミティブの位置決めの際のプリミティブの移動に伴う対象形状の変形を，リアルタイムに画面表示可能なソリッドモデラのユーザインタフェースの開発を目指す．

2．スキャンライン法による集合演算表示

スキャンライン法による集合演算表示は，通常のスキャンライン法による多面体の隠れ面消去処理のおいて，奥行き比較による可視面の判定のかわりに，奥行き方向の1次元の集合演算処理をサンプルスパン単位で行うことによって実現できる [1].

この処理の概略は，次のようになる [6] [7].

```
走査線を描く ( X-sort list )
{
    Z-sort list = 空;
    xl = スクリーンの左端の位置;
    while ( X-sort list != 空 ) {
        xr = X-sort list-> 交点のX座標値;
        if ( Z-sort list == 空 )
            背景の色を塗る ( xl, xr - 1 );
        else
            可視判定 ( Z-sort list, xl, xr );
        xl = xr;
        X-sort list = リストの更新 ( Z-sort list, X-sort list, xr );
    }
    xr = スクリーンの右端の位置;
    背景の色を塗る ( xl, xr );
}
```

```
可視判定 ( Z-sort list, xl, xr )
{
    順序が変化した面分 = リストの順序の検査 ( Z-sort list, xr );
    if ( 順序が変化した面分 != 空 ) {
        交点 = 交点算出 ( 順序が変化した面分 , 順序が変化した面分 -> 次の面分 );
        可視判定 ( Z-sort list, xl, 交点 );
        並べ替え ( Z-sort list, 交点 + 1 );
        可視判定 ( Z-sort list, 交点 + 1, xr );
    }
    else {
#if 集合演算処理を行う
        可視面 = 集合演算処理 ( Z-sort list );
#else
        可視面 = Z-sort list-> 先頭の要素 ;
#endif
        面分の色を塗る ( 可視面 , xl, xr - 1 );
    }
}
```

上記のアルゴリズム中の関数“リストの更新”は，*xr* に交点を持つ稜線を *X-sort list* から全て取り出し，その稜線を共有する２つの面分について以下の処理を行った後，*xr* のすぐ右に交点をもつ稜線の *X-sort list* における位置を返す．

その両方が既に *Z-sort list* に登録されていれば，それらを *Z-sort list* から取り除く．

その両方が *Z-sort list* に登録されていなければ，それらを *Z-sort list* に登録する．

どちらか一方だけが *Z-sort list* に登録されていれば，それを *Z-sort list* から取り除き，その位置にもう一方の面分を登録する．

この (c) の処理はセグメントチェイン [2] を利用した効率化を実現する．仮にシーン中に面分同士の交差を含まないなら，“リストの順序の検査” 以降の処理をすべて省き，*Z-sort list* の先頭の面分を可視面とするだけでよい．

集合演算表示を行おうとする場合は面分の交差の処理を省くことはできないが，(c) において集合演算処理によって判定された可視面を引き継ぐことにより，次のサンプルスパンにおける集合演算処理を省略できる [1].

面分同士の交差がない場合は，走査線の類似性を利用した高速化も可能である．ある走査線の処理が終了した後，次の走査線において進入

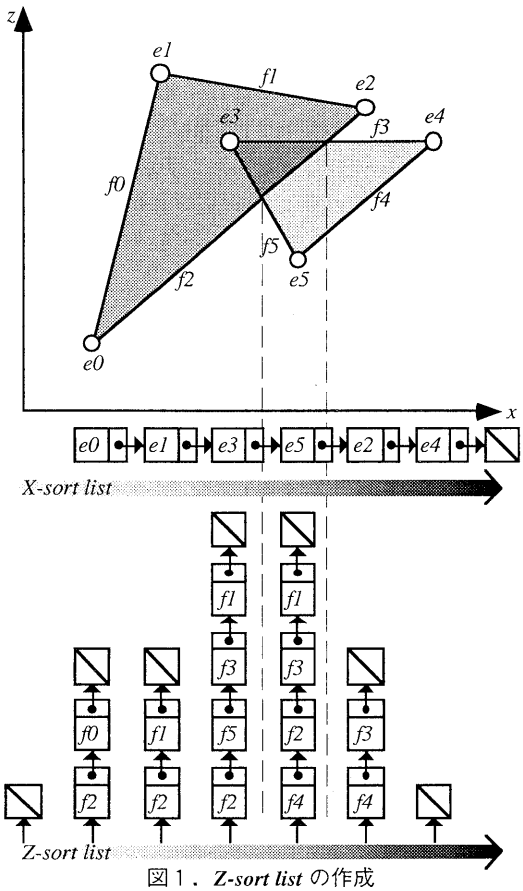

図１．*Z-sort list* の作成

あるいは退出する稜線が存在せず，かつ *X-sort list* の順序にも変化がなければ，直前の走査線上の各サンプルスパンにおける可視面が，現在の走査線の対応するサンプルスパンにおける可視面となる．したがって，この場合は上記の走査線上の処理を全て省くことができる．

集合演算表示を行おうとする場合は面分の交差の検出を行うために走査線上の処理を省くことはできないが，交差を含まないサンプルスパンにおいては，直前の走査線の対応するサンプルスパンにおける集合演算処理の結果を引き継ぐことができる．

以上の処理により，実際に集合演算処理を行うのは稜線が進入／退出する位置，すなわち頂点と，面分同士の交差線上のみとなる．

3．ブール結合の記述

奥行き方向の1次元の集合演算処理 (ray firing) は，サンプルスパンと重なる面分について，手前のものから順にブール結合を検査し，可視面を判定する．なお前述のアルゴリズムでは，サンプルスパンに重なる面分が *Z-sort list* に視点に近い順に登録されているため，この時点で改めて並べ替えを行う必要はない．実際の集合演算処理アルゴリズムはブール結合の記述方式によって異なるため，ここではそのいくつかについて考察する．

3．1 木構造

プリミティブを会話的に追加していく方式の形状モデリング作業によって，木構造によるブール結合の記述が得られる．

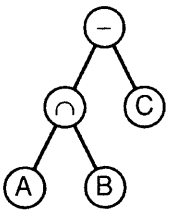

図2．木構造

図2は A∩B−C というブール結合関係を木構造で表したものである．この B のプリミティブを構成する面が可視であることを確かめるには，それが A の内部にあり，かつ C の外部にあることを知らなければならない．この内外判定は *Z-sort list* 保持されている A および C と B との前後関係をもとに容易に行うことができるが，一つの形状を構成するプリミティブの数が多くなると木が深くなり，木の探索自体に時間がかかるようになる．

3．2 パターンマトリクス

パターンマトリクス [3] は，プリミティブのブール結合を積和形のブール演算式で表し，マトリクス状の表によってブール結合を記述する方式である．差集合演算の表現には負のプリミティブを用いる．図3に A∩B−C というブール結合関係をパターンマトリクスで表した場合を示す．木構造で記述された形状は，記号処理により容易にパターンマトリクスに変換できる．

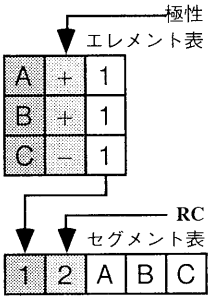

図3．パターンマトリクス

この場合の集合演算処理は，次のようなインクリメンタル計算によって実行できる [6]．あらかじめセグメントに属する正のプリミティブの数をリファレンスカウント (RC) の初期値に設定しておき，*Z-sort list* をたどり取り出した面分が表ならその面分が属するセグメントの RC を1増し，裏なら1減じる．この結果 RC が0となった面分が可視面となる．なお負のプリミティブについては，あらかじめ面分の表裏を反転しておけば，特別な処理をする必要はない．

この方法は，木を探索する手間が不要で，わずかなオーバーヘッドで集合演算処理による可視面を判定できる．しかし，一つのプリミティブが複数のセグメントに属している場合は，その全てのセグメントについてインクリメンタル計算を行う必要がある．

3．3 プリミティブの正負による方法

隠れ面消去処理アルゴリズムが面分の交差を処理できれば，それは和集合演算処理の機能を持つことになる．このとき差集合演算を負のプリミティブにより表せば，これを別段のブール結合の記述を用いずに記述することができる．また，この負のプリミティブの処理は，前述のパターンマトリクスを用いた場合の集合演算処理アルゴリズムにおいて示したように，プリミティブを構成する面分の表裏を反転するだけで実行することができる．

したがって和と差に限れば，従来の隠れ面消去処理アルゴリズムに対してわずかな拡張を行うだけで，集合演算表示が可能になる．ここではその際のプリミティブの結合規則について考察する．

算術演算方式

スキャンライン平面上で重なっているプリミティブの数が1以上の領域を可視とする（図4）．これは *Z-sort list* から取り出した面分が表ならプリミティブの重なり数を1増し，裏なら1減じることによって実現できる．この方法は簡便だが，複数の正のプリミティブが重なる部分を削り取るのに手間がかかり，利用者が意図した形状を得にくい場合がある．

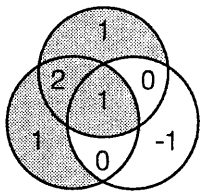

図4．算術演算方式

論理演算方式

算術演算方式の問題点を避けるために，負のプリミティブを優先する（図5）．これを論理演算方式と呼ぶことにする．この方式は直観的には理解しやすいが，負のプリミティブの領域内には他の物体を定義できなくなる．

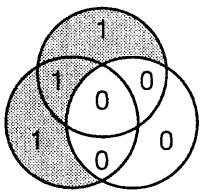

図5．論理演算方式

プリミティブの登録順序を優先度として用いる方式

論理演算方式において，プリミティブを登録した順序を負のプリミティブの優先度として利用すれば，木構造に類似した形式で形状記述を行うことができる．この場合の集合演算処理は以下のアルゴリズムで実行できる．

```
集合演算処理 ( Z-sort list )
{
    優先度 = 最低;
    面分 = Z-sort list;
    do {
        if ( 面分 -> 面の向き == 表 ) {
            if ( 面分 -> プリミティブ -> 登録順 >= 優先度 )
                return 面分;
            if ( 面分 -> プリミティブ -> 極性 == 負 )
                優先度 = 優先度リストから削除 ( 面分 );
        }
        else {
            if ( 面分 -> プリミティブ -> 極性 == 負 )
                優先度 = 優先度リストに登録 ( 面分 );
        }
        面分 = 面分 -> 次の面分;
    } while ( 面分 != 空 );
    return 空;
}
```

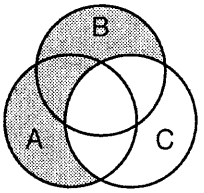

図6．A+B-C

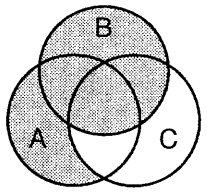

図7．A-C+B

このアルゴリズム中の関数“優先度リストに登録”および“優先度リストから削除”は，面分を優先度の順に並べた優先度リストに対して面分を登録あるいは削除し，その後の優先度の最大値を返す．

4．まとめ

スキャンライン法による集合演算表示では，画像生成の際の集合演算処理の実行回数が面分の交差のない部分においては画面の解像度に依存せず，交差の部分でも走査線数にのみ依存する．またブール結合をプリミティブの正負とその登録順序を用いて記述することにより，木構造に似た形式の形状記述を実現しながら，処理時間が形状定義の複雑さに依存しない集合演算処理が実現できる．

したがって，この場合の陰影画像生成に要する処理時間は，ほぼ走査線数と面分数に比例すると考えられる．これは集合演算処理を行わない通常のスキャンライン法による隠れ面消去処理と同等のオーダである．

またこのアルゴリズムは半透明表示も実現できるため，負のプリミティブを半透明表示することによって，切断面の確認や位置決めを行いやすくすることも可能である．

本アルゴリズムは，積集合演算処理を欠いているため一般的な形状モデリングに必要な形状の記述能力を満たしていないが，和と差および形状の登録順序に基づく形状記述は直観的に理解しやすく，形状の変更結果の画面表示も即座に行えるため，形状モデリングの際のユーザインタフェースや，簡便なソリッドモデラを実現するのには適していると考えている．

参考文献

[1] Atherton, P. R.: A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry, *Computer Graphics*, Vol. 17, No. 3, pp. 73-82 (1983)

[2] 山口，時枝：分類的手法による隠れ線・隠れ面消去アルゴリズムの創成，グラフィックスとCADシンポジウム論文集，pp. 133-140 (1983)

[3] 木下：形状モデリングを核としたCAD/CAMシステム TIPS-1, PIXEL, No. 24, pp. 70-81 (1984)

[4] 床井，藤阪，北橋：二次曲面を含む半空間式の集合演算によって定義された三次元形状のスキャンライン法による表示，情報処理学会第32回全国大会講演論文集 (III)，pp. 2083-2084 (1986)

[5] 床井，北橋：スキャンライン法を用いたCSGデータからの高速陰影画像生成アルゴリズム，昭和61年電気関係学会関西支部連合大会講演論文集，P. S50 (1986)

[6] 床井，北橋：スキャンライン法による多面体の集合演算表示の高速化，情報処理学会論文誌，Vol. 34, No. 11, pp. 2412-2420 (1993)

[7] 床井：リアルタイム集合演算表示に関する研究，情報処理学会研究会報告，Vol. 95, No. 78 (95-CG-76)，pp. 39-44 (1995)