グラフィクスとCAD 16－4
（１９８５．３．１２）

逆操作を持った立体集合演算

Invertible Set Operation for Solid Modeling

鳥谷　浩志　　佐藤　敏明　　千代倉　弘明
Hiroshi TORIYA　　Toshiaki SATOH　　Hiroaki CHIYOKURA

(株)リコー　　技(本)ソフトウェア研究所
(RICOH COMPANY, LTD.)

A method for implementing invertible set operations is proposed.
In our set operation algorithm, solids are generated and modified using primitive operations, each of which has a corresponding inverse. The operations applied to the solid are stored in a tree structure which represents the solid design process. An inverse set operation is executed by applying the inverses of stored primitive operations in reverse order.
Inverse set operations are more efficient because they rely less on geometrical calculation.

１．　はじめに

三次元形状設計において立体モデル生成システム(solid modeling system)を用いることの有用性は、広く認められている。このような立体モデル生成システムを対話的設計環境において用いると、設計者はしばしば過去に設計された立体形状を再生成したいと考える。なぜなら、設計という過程は多くの試行錯誤を含み、また、設計者は自分自身の誤ちによって、立体をこわしてしまうということがあるからである。そこで、我々[4]は過去の立体をすばやく再生成するための手法を提案した。我々のシステムでは、すべての立体変形は、常に基本操作(primitive operation)を用いて実現される。基本操作とは、Euler操作[1、2]を含んだ立体を変形するための原始的な操作であり、対応する逆操作が常に存在する。立体変形において用いられた基本操作は、立体生成過程の表現の中に蓄えられ、この表現を用いて過去の立体は再生成される。本論文では、基本操作によって記述される立体集合演算のアルゴリズムを示す。現在、我我はUNIX上の言語Cを用いて、形状設計システムDESIGN BASEを開発中である。DESIGN BASEは、形状設計システムMODIF[3]の特徴をすべて引き継いでいる。本論文に示されている立体集合演算は、このシステム上に実現された。

２．　過去の研究との比較

関連した研究としては、Mantyla[5]らによるGWBの開発がある。GWBでは、立体集合演算は、Euler操作を用いて実現されているが、Euler操作を用いずに立体データを変更している部分がある[6]。これに対し、我々のシステムでは、すべての立体変形を基本操作を用いて行ない、しかも、用いられた基本操作が蓄えられるので、逆操作の実現が可能になる。 さらにGWBの立体集合演算アルゴリズムでは、立体は完全に分離され、その後、接合操作によって結果の立体が生成される。立体を分離するときに、長さ0の稜線を面や稜線上に生成するため、一方の立体の面と他方の立体の頂点が干渉したときには、稜線の生成が複雑になる。これ対しに、我々の手法では、干渉線を両方の立体に対し稜線として加えた後、不要部分を除去するので、処理は統一化され、簡単になる。

多くの立体モデル生成システムでは、稜線や頂点の内部名は、大域的に決められている。たとえば、稜線名が1からn番目まで用意されているとすると、それぞれの立体はこの中の一部を使う。このような手法は、逆操作を行なうためには、適していない。なぜなら、立体Aの中で除去された稜線名を立体Bがもし使うと、立体Aの過去の立体を再生するのが困難になるからである。大域的な内部名を用いるシステムでこのことをさけるためには、一度、除去された稜線名を他の立体が使うことを禁止しなければならない。これは記憶領域のむだである。これに対し、我々のシステムでは稜線名は、いつもそれぞれの立体に対し、1から始める。この結果、同一の記憶領域を二つの立体が使うことはあっても、一つの稜線名を二つの立体間で共有することはなくなる。

３．　立体集合演算の概要

表1にこの立体集合演算で用いる基本操作を示す。このうち立体の位相情報を変更する操作は、Euler操作とほぼ同じである。異なるのは、Euler操作では、面の中の穴をringと呼んでいたのに対し、ここでは、面とringをループと呼び、統一的に扱うようにしている点である。以後、面の境界をP(parent)ループ、穴をC(children)ループと呼ぶことにする。立体集合演算には、和(union)、差(difference)、積(intersection)の3種がある。このうち、立体A、B間の差と積は、立体を反転させる基本操作NSと和の演算を以下のように組合せて実現できる。

difference(A,B) = NS(union (NS (A), B))

intersection(A,B) = NS(union (NS (A), NS (B))

基本操作NSは逆操作を持つ。したがって和の集合演算を基本操作により実現すれば、すべての集合演算について逆操作が可能となる。

表1　立体集合演算で用いる基本操作

Name	Definition
MEL	Make an Edge and a Loop
KEL	Kill an Edge and a Loop
MVE	Make a Vertex and an Edge
KVE	Kill a Vertex and an Edge
KPLMCL	Kill a P-Loop* and Make a C-Loop
KCLMPL	Kill a C-Loop and Make a P-Loop
MEV	Make an Edge and a Vertex
KEV	Kill an Edge and a Vertex
MEKL	Make an Edge and Kill a Loop
KEML	Kill an Edge and Make a Loop
MEVVL	Make an Edge and a Vertex, and a Vertex and a Loop
KEVVL	Kill an Edge and and a Vertex, and a Vertex and a Loop
NS	Negate a Solid
AS	Add two Solids
DS	Decompose a Solid

次に和の集合演算の実現手段の概要を述べる。我々の手法は、大別して四つの過程から成っている。

<ステップ1>　干渉線の算出

図1(a)に示すような二立体の集合演算を考える。まず、二つの立体の干渉線を算出し、干渉線と立体の間の位置関係の情報を蓄える。

<ステップ2>　立体の稜線としての干渉線の生成

蓄えられた情報をもとに干渉線を立体の稜線として基本操作により生成する。この稜線は、二つの立体に対し、それぞれ生成されることになる。そこで、これをD(double)稜線と呼ぶ。また、D稜線の両端の頂点をD(double)頂点と呼ぶ(図1(b))。

<ステップ3>　不要部分の削除

他方の立体の内部に含まれる部分を除去する。除去すべき部分の切り口は、D稜線に囲まれている。このことを利用して、除去すべき稜線を探し出し、基本操作により除去する(図1(c))。

<ステップ4>　立体の接合

対応するD頂点に長さ0の稜線を生成し、二立体を接合させ(図1(d))、次に、不要な稜線及び頂点を除去すると、立体集合演算は完了する(図1(e))。

4　アルゴリズムの詳細

ここでは、前節で述べた四つの過程について、その詳細を述べる。

4．1　干渉線の算出

立体AとBの干渉線を求める手順を示す。

<ステップ1>　ラフチェック

物体A、Bを囲む最少の直方体をつくり、ラフな干渉チェックをする。これにより、交わる面の組(Fa, Fb)を求める。ここで、FaはAの面、FbはBの面を示す。

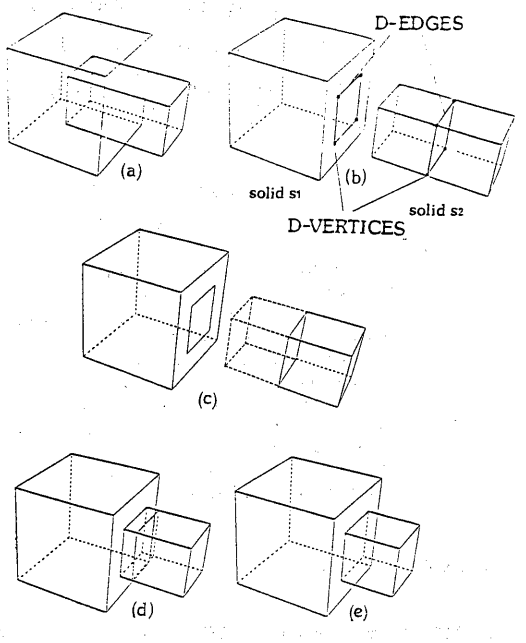


図１．立体集合演算の実行過程

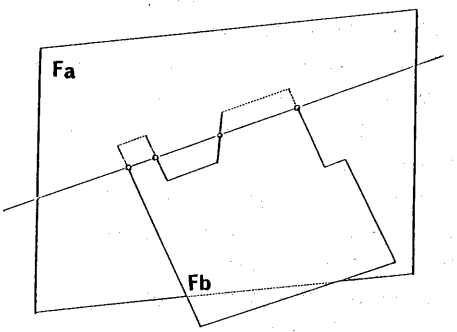


図２．面Faと面Fbの交線

<ステップ2>　交点の算出

FaとFbの干渉する面の交点を求める。このためには、Faを構成するすべての稜線と面Fbとの交点を求めて蓄える。次にFbのすべての稜線と面Faとの交点を求めて蓄えておく。このとき、得られた交点と立体A、Bとの位相的関係も求めておく。交点と立体との位相的関係には交点が、立体の頂点上にある、稜線上にある、面上にあるという三種類がある。

<ステップ3>　干渉線かどうかの判定

ステップ2で得られた交点は、面FaとFbののっている無限平面の交線上にある(図2)。座標値をもとに、この交点をソートし交線上にならべる。次に、隣り合う交点を結ぶ線分が、干渉線になるかどうか調べる。もし、この線分が面Faに含まれ、かつ、面Fbに含まれていればこの線分は干渉線になる。この判定は、後述する包含判定アルゴリズムにより実現される。

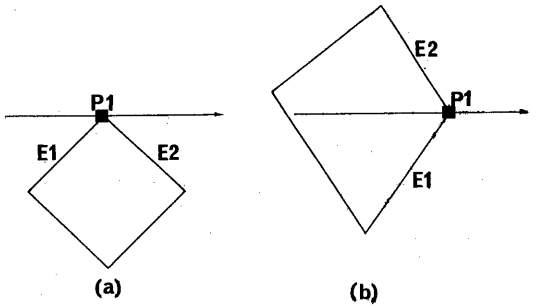
〈ステップ4〉 データの登録

こうして得られた干渉線に関する位相情報及び座標値などの幾何情報を蓄える。これらの情報は干渉線のテーブルとその端点である交点のテーブルの二つに分けて蓄えられる。

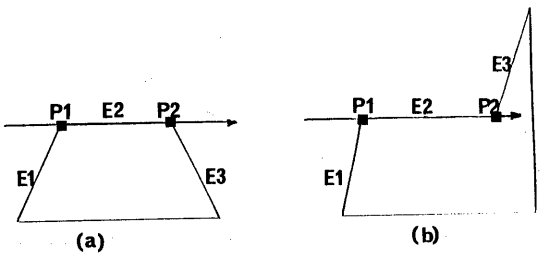
交わる可能性のある平面の組すべてについてステップ2-4を適用することにより、干渉線の算出は終了する。

次に、包含判定アルゴリズムについて述べる。これは、ある点がループ内に含まれているかどうかという問題に帰着する。これには、ある点から延ばした半直線とループの交点の数を調べるというよく知られた方法がある。この個数が偶数なら、点はループの外、奇数なら内にあるわけである。

前にも述べたように、二つのループの交点は、それぞれのループを含む無限平面の交線上にある。これを利用すれば、新たに交点を算出しなくても、すでに求められた交点の数をチェックすればすむはずである。ただし、得られた交点がループの頂点P1と一致する場合（図3(a)(b)）や、ループ内の稜線E2が無限平面の交線上にある場合(図4(a)(b)）には、特別に処理する必要がある。これらの場合、交点を数えるべきときと、数えてはならないときとがある。これを判定するためには、図3(a)(b)の場合、交点から出る二本の稜線 E1, E2が、無限平面の交線に対し、同じ側にあるかどうかをチェックすればよい。また図4(a)(b)の場合には、頂点P1、P2から出る稜線E1、E3が同じ側にあるかどうかを調べるとよい。こうして数えるべき交点がわかれば、この個数を数えることにより、簡単に包含判定を実現することができる。



⟶ : 干渉線
P1 : 交点
E1, E2 : ループ上の稜線
図３．交点とループの頂点が一致する場合



⟶ : 干渉線
P1, P2 : 交点
E1, E2, E3 : ループ上の稜線
図４．交点とループの稜線が一致する場合

４．２ 干渉線の稜線としての生成

前のステップでは、交点及び干渉線のテーブルを得た。以下の過程では、まず交点のテーブルをもとに必要な頂点の生成を行なう。次に、干渉線のテーブルを参照してその生成を基本操作を用いて行なう。このとき、生成の順序を考慮すると無駄なループの生成と除去を避けることができる。

４．２．１ 交点の生成

交点のデータを参照すれば、その交点と立体A、Bとの位相関係がわかる。前述のように、これには、交点が、i)立体の頂点上にある、ii)面の内側にある、iii)稜線上にある、の三つの場合がある。i)の場合、すでに頂点が存在するので生成する必要はない。ii)の場合、稜線の生成時に、交点も同時に生成するので、これも考えなくてよい。iii)の場合は、立体の稜線上にあって、かつ、まだ生成されていない交点を頂点として基本操作MVE(表1参照)で生成する。

以上の処理により、稜線上の交点はすべて生成される。したがって交点のうちまだ生成されていないものは、すべて面の内側にあることになる。このように稜線上に頂点を先に生成することにより、稜線の生成の際、どの基本操作を用いたら良いかの判定が容易になる。

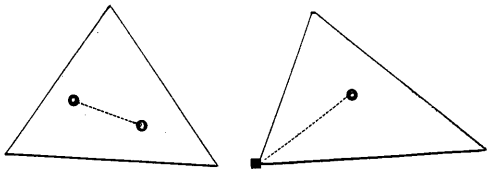
４．２．２ 立体の稜線としての干渉線の生成

干渉線のテーブルを参照しながら基本操作により稜線を生成してゆく。

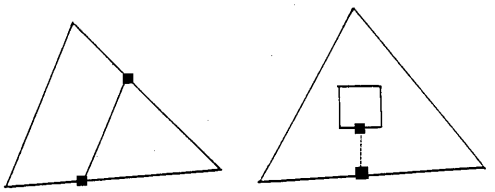
交点、すなわち干渉線の端点がすでに頂点として生成されているか否かにより、干渉線を i) 両方とも生成されていない、ii) 一方だけが生成されている、iii) 両方ともすでに生成されている、の三種類に分類できる。

ここで、生成されていない交点は、すべて面内の点であるという事実を考えると、i)、ii)、iii)に対して適用すべき基本操作は、i)はMEVVL, ii)はMEV, iii)はMELまたはMEKLとなる（図5）。iii)では、生成しようとする稜線の二端点が同じループ内にあるときにはMEL、異なるときにはMEKLを用いることになる。

稜線を生成する順序は重要である。無秩序に稜線を生成すると無駄な計算が必要となることがあるからである。たとえば、図6(a)はループL1の中に二本の稜線E1、E2を生成することを示している。もしMELにより稜線E2を先に生成してしまうと稜線E1の属するループ名がかわってしまう。このため、次に稜線E1を生成する際、これをどのループ上につくるかを求めなければならない。E1, E2の順で生成すれば、このような計算は不要である。



(a) MEVVL　　(b) MEV



(c) MEL　　(d) MEKL

■ ： 既に生成されている頂点
o ： まだ生成されていない頂点
図５．稜線の生成

また、図6(b)のようにループ内に四辺形PQRSを生成することを考える。まずMEVVLにより稜線PQとRSを生成すると、L1、L2という二本のループが存在することになる。このあと、QR、PSという稜線を生成して四辺形PQRSをつくると、L1かL2の一方にループを消す必要が生じる。いっぽうこの四辺形をPQ、QR、RS、SPの順に生成すれば、このような無駄なループをつくらないですむ。

以上を考慮すると稜線を生成する順序は以下のようになる。

<処理1> MEVにより生成できる稜線をすべて生成する。

<処理2> MEVVLにより一本稜線を生成する。

<処理3> 生成できる稜線がなくなるまで処理1、2をこの順序でくり返す。

<処理4> 残りの稜線をMEL又はMEVLで生成する。

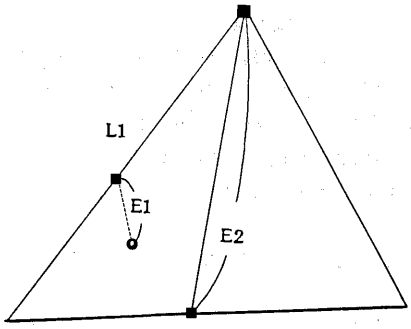
以上の稜線の生成は、与えられた二つの立体に対し、それぞれ行なわれる。こうして生成された二立体の稜線及び頂点の組は、それぞれ、D稜線、およびD頂点のデータとして蓄えられる。次の過程では、このデータをもとに不要部分の除去を行なう。

４．３　不要部分の除去

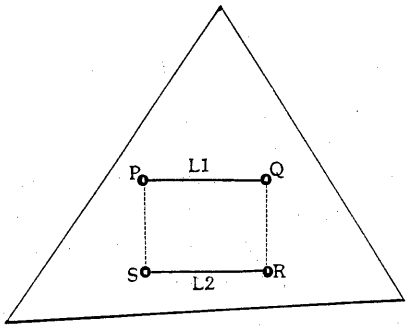
不要部分の除去は、次の三つのステップより成る。

<ステップ1>　不要な面の検出

D稜線は、二立体の干渉線上に生成されている。したがって、それぞれの立体は、対応するD稜線を持っている。このD稜線の両側の面のうち、立体の内側に含まれる面に、除去される、という情報を与え、また、立体の内側に含まれない面に、除去されない、という情報を与える。



(a) MEV and MEL



(b) MEVVL

■ ： 既に生成されている頂点
o ： まだ生成されていない頂点
図６．稜線の生成を避ける場合

<ステップ2>　不要な稜線の検出

ステップ1で与えた情報をもとに、他方の立体の中にある不要な稜線すべてに印をつける。

<ステップ3>　不要な稜線の除去

印のついた稜線を基本操作を用いて除去する。

次に、各ステップの詳細を述べる。

４．３．１　不要な面の検出

D稜線の両側の面に着目して、面と相手立体の位置関係により、面に次のような内外情報を与える。

i)面が立体の外側にあれば、"+"

ii)面が立体の内側にあれば、"−"

iii)面が相手立体の面に接していれば、"0"。

図7は二つの立体A、Bの交差をあらわしている。この図で斜線のある側が立体の内側である。"+"、"−" の情報は図に示されているようになる。この内外情報は図からわかるように、四つの面がなす角度Ai(i=1, 2, 3)で判定することができる。角度を直接求めると、計算時間のかかる逆三角関数を使う必要がでてくる。これをさけるために面の法線ベクトル同士の内積を利用して角度と一対一に対応する量を求める。これを利用すると、計算時間を減らすことができる。

二立体の交差のしかたは図8に示すように九通りに分類することができる。内外情報の組み合わせは、図8の(h)と(i)の場合が同じになって、図9のように全部で八通りの場合がある。このうち、⑤から⑧は"0"を含んでいる。この"0"は、＋あるいは－におきかえることで、①～④の場合に帰着できる。但し、⑧の場合は特別で、接している面の法線ベクトルの方向を比較して、②か④のいずれかに帰着したらよいかを判定する必要がある。

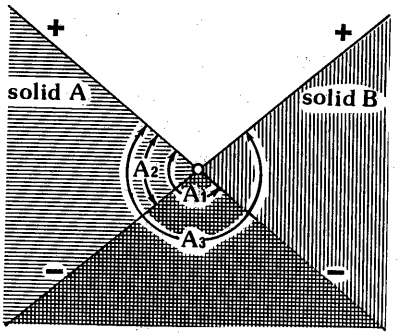


図７．立体AとBの交差

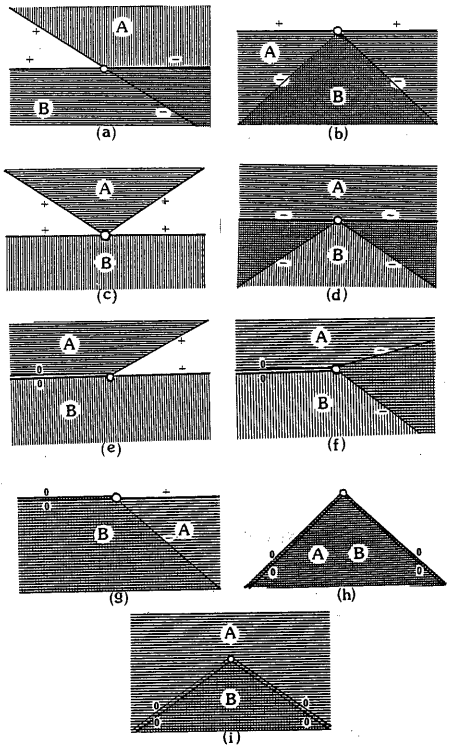


図８．干渉する2立体の関係

こうして得た内外情報の組に従って、以下の処理を行なう。

i) ①の場合

"－"の面に除去される、"＋"の面に除去されないという情報を与える。

ii) ②の場合

これは、一方の立体が他方の立体に完全に含まれる場合で、二立体の干渉線がないと考えてよい。従って"－"の面に除去される、"＋"の面に除去されないという情報を与え、D稜線という情報を取り除く。よって、システムは、以後、この稜線をD稜線とはみなさない。

iii) ③の場合

これは図8(c)に示すように二立体が稜線のみで接している場合である。この場合は、干渉があるとは考えない。従って、D稜線という情報を除去し、あとは何も行なわない。

iv) ④の場合

これは、二立体の各面がお互いに相手立体の内側に含まれてしまう場合である。四つの面全てに除去されるという情報を与え、D稜線という情報を除去する。

４．３．２　不要な稜線の検出

4.3.1でD稜線という情報を除去されなかったD稜線に囲まれた部分のうちで、除去されるという情報がある側に存在する全ての稜線をたどり、印をつけていく。

４．３．３　不要な稜線の除去

除去すべき領域にある稜線を基本操作を用いて除去していけば、頂点やループも同時に除去できる。我々のシステムでは、逆操作によって立体を再構成することも考慮している。稜線を無作為に除去していくだけでは、幾何学的な判定計算の必要な基本操作が現れる可能性がある。このような基本操作が現れると、逆操作の効率が低下してしまう。そこで、我々は、そのような基本操作の出現を避けることを考えた。

基本操作は、多くの場合、データ構造の論理的な書き換えにすぎないが、次の二つの場合に幾何学的な判定を行なう必要が生じる。

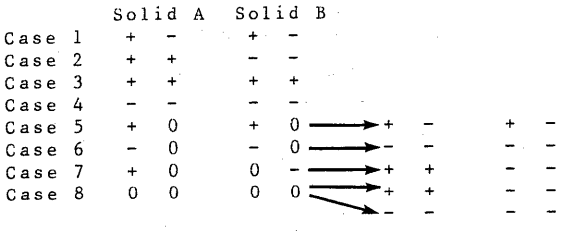


図９．2立体A、Bの内外情報の分類

[場合1] その第一は、Cループを含んでいるループに稜線を生成して、ループを分割する場合である。たとえば図10の場合は、最初ループL1に含まれていたCループLcは、稜線Eが生成された後、ループL2に含まれるのか、L1に含まれるのかを判定しなければならない。

[場合2] 図11(a)は、頂点Vから単独でのびている稜線E1を示している。今後、このような稜線をS(stick)稜線と呼ぶ。第二の場合は、図11(b)のように、S稜線がつながっている頂点Vに対して新たに稜線をつけ加える場合である。このような場合、稜線の順序は、E2-E0-E1-E3となるのか、E2-E1-E0-E3となるのかを判定しなければならない。

これらの判定には幾何学的計算が必要となる。また、立体の再構成時には、平面でないような面が現れるため、このような判定は容易ではない。そこで、上に述べた二つの場合が生じないようにするため、次の方法で稜線の除去を行なう。

まず、除去されるべき領域内にある全てのCループをPループにかえる。こうすることで、場合1の出現を避ける。次に、処理1～3を印のついた稜線Eがなくなるまでくり返して、場合2の出現を避ける。

<処理1> EがS稜線であるとき、KEVまたはKEVVLを用いてEを消す。

<処理2> Eの両側のループが異なり、かつ両側のループのどこにもS稜線がつながっていないとき、KELを用いてEを消す。

<処理3> 印のついた稜線は残っているが、どの稜線も処理1、または、2で除去できない状況が生じたら、印のついた稜線を適当に一本選んでKEMLで消す。

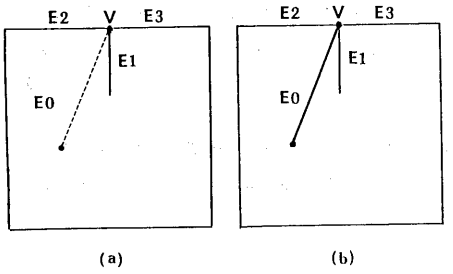


図１１．S稜線につながっている頂点に新しい稜線をつけ加える場合

４．４ 立体の接合

不要部分の除去を行なった結果、各立体は互いに対となっている接合されるべき面を持っている。二立体の接合は、この対となっている面の境界をなすループ同士を接合することよりなされる。すなわち、対となるループの頂点（これらは全てD頂点である）間に長さ0の稜線を生成することで立体を接合する。長さ0の稜線は、まず適当に選んだD頂点間にMEKLを用いて生成し、それから残ったD頂点間にMELを用いて生成する。

この方法で二立体の接合を行なうためには、対となるループが必ず存在することが必要である。ところが、図12(a)のような立体集合演算を行なうと、稜線の除去を行なった結果、図12(b)のように、くびれを持ったループが現れる。くびれを持ったループとは、ループを一周するとき二度以上同じ頂点を通過してしまうようなループのことである。また、このような二度以上通過する頂点をくびれの頂点と呼ぶ。くびれを持ったループと対になっている相手立体のループがくびれのループでない場合は、接合を行なうためのループが一

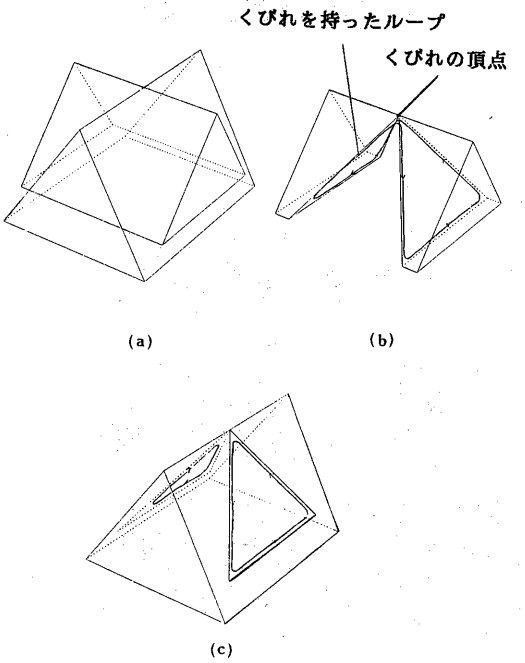


図１２．くびれを持ったループ

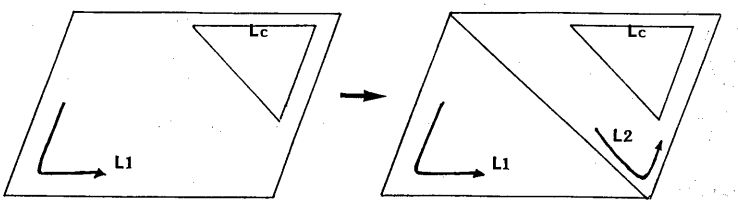


図１０．Cループを含んだループを新しい稜線の生成で分割する場合

対一に対応しない。たとえば図12の場合、(b)のくびれを持ったループは、(c)のくびれを持たない二つのループと対応しなければならなくなる。このように接合するループが対とならない場合は、前述の方法が適用できない。したがってこのときは、くびれの頂点でくびれを持ったループを分割することが必要となる。そのためには、くびれの頂点がある場所と同じところに新たに頂点を生成し、この新しい頂点で立体を二分割してくびれを持ったループをなくせばよい。

このようにして二立体の接合を行なうと、結合部分にある頂点と稜線(D頂点とD稜線)は重複することになる。そこで、これらを次の順序で消去する。

<処理1> 一方の立体のD稜線をKELを用いて全て除去する。

<処理2> 一方の立体のD頂点をKEVまたはKVEを用いて全て除去する。

<処理3> 処理1の結果、D稜線の対のうちの一方は除去されずに残っている。この残った稜線の両側の面の法線ベクトルの向きが同じ時は、この稜線は立体を表現する上で不要であるのでKELまたはKVEを用いて除去する。

以上の処理が全て終了すると、立体集合演算は終了する。

5. 例題

図13(a)、(b)は、集合演算をする前の二立体の線画である。この二立体の和を(c)、差を(d)、積を(e)に示す。

6 結論

本論文では、逆操作を持った立体集合演算の実現法を提案した。我々の手法は、二つの立体の干渉する面に曲面がない場合に適用され、このプログラムは、三次元形状処理システムDESIGN BASEの中で実現された。

本手法においては、立体集合演算に必要なすべての形状変形を、基本操作を用いて行なう。この基本操作列を保持しておくことにより、逆操作が可能な集合演算が実現できる。立体集合演算は、干渉チェックのために多くの計算が必要であるが、その逆操作は、基本操作を用いるだけなので、もとの形状を高速に復元できる。

我々のアルゴリズムは、逆操作の計算効率を考慮している。たとえば、干渉線の生成の過程では、無駄なループを生成しないように基本操作を適用する。また、不必要な稜線の消去の過程では、逆操作時に無駄な幾何学的計算が生じないようにしている。

形状の設計をする際には、自由曲面を持った立体間の集合演算が必要となる場合も多い。 今後、曲面同士の干渉をも含む立体集合演算の実現を予定している。

[謝辞]

多くの有益な助言をいただいた東京大学 木村文彦助教授、ならびに、國井秀子所長に感謝いたします。

参考文献

[1] B. G. Baumgart,
A Polyhedron Representation for Computer Vision,
AFIPS Conf. Proc., Vol.44, pp.589-596, May 1975.

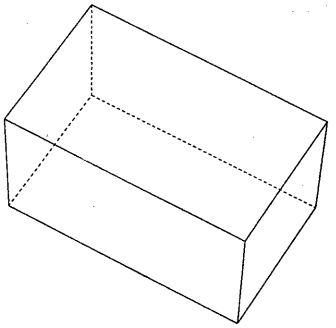
[2] I. C. Braid, R.C.Hillyard and I.A.Stroud,
Stepwise Construction of Polyhedra in Geometric Modelling,
Mathematical Methods in Computer Graphics and Design (Ed. by K.W.Brodlie) Academic Press, pp.123-141, 1980.

[3] H. Chiyokura and F. Kimura,
Design of Solids with Free-form Surfaces,
Computer Graphics (SIGGRAPH'83 Proc.), Vol.17, No.3, pp.289-298, 1983.

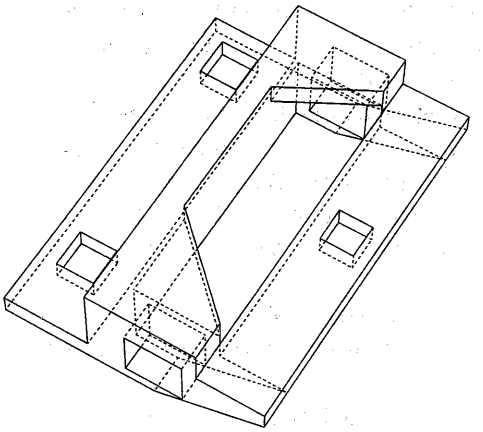
[4] H. Chiyokura and F. Kimura,
A Representation of Solid Design Process using Basic Operations,
CGTokyo'84 Proc., T4-6, April 1984.

[5] M. Mantyla and R. Sulonen,
GWB : A Solid Modeler with Euler Operators,
IEEE Computer Graphics and Applications, Vol.2, No.7, pp.17-31, September 1982.

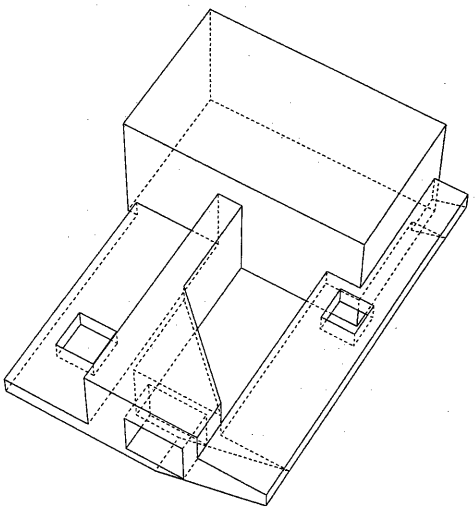
[6] M. Mantyla,
Computational Topology: A Study of Topological Manipulations and Interrogations in Computer Graphics and Geometric Modeling,
Acta Polytechnica Scandinavica, Mathematics and Computer Science Series No.37, Helsinki 1983.



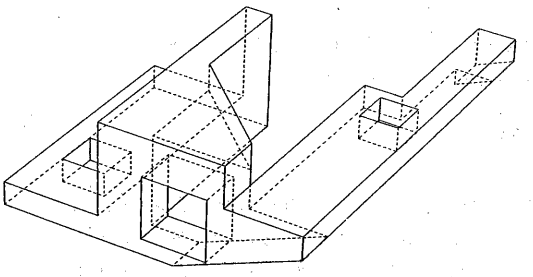
(a)



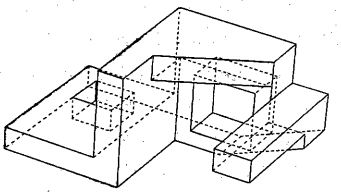
(b)



(c)



(d)



(e)

図１３．集合演算の例