The University of Osaka
Institutional Knowledge Archive

Title	集合演算表現された多面体の直接的かつ高速な描画アルゴリズム
Author(s)	床井, 浩平
Citation	大阪大学, 2002, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/1267
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

集合演算表現された多面体の›
直接的かつ高速な描画アルゴリズム›

２００１年１２月

床井浩平

内容梗概

本論文は，筆者が豊橋技術科学大学大学院工学研究科情報工学専攻修士課程在学中，和歌山大学経済学部産業工学科，大阪大学産業科学研究所音響材料部門（北橋研究室，内地研究員），および和歌山大学システム工学部デザイン情報学科において行った，多面体の集合演算表現によって定義された形状の陰影画像を，形状データに対する集合演算処理を行わずに，直接的かつ高速に生成するアルゴリズムについて論じたものである．本論文は次の６章および補遺から構成される．

第１章ではインタラクティブコンピュータグラフィックスの成立について触れ，その延長上にあるインタラクティブな形状モデリングシステムが，デザイナーに対して果たすべき役割について論じた後，本研究の目的と位置づけについて述べる．

第２章では準備として，本研究のベースとなっている多角形の走査変換アルゴリズムについて述べ，それを応用した２次元図形の論理演算表示アルゴリズムについて論じる．

第３章では第２章で述べた２次元図形の論理演算表示アルゴリズムを３次元に拡張して，多面体をプリミティブに持つ CSG (Constructive Solid Geometry) で記述された形状の，スキャンライン法による直接集合演算表示アルゴリズムについて論じる．CSG は単純な形状を持つ基本立体（プリミティブ）を移動や回転によって空間中で位置決めした後，それらの集合演算による組み合わせを用いて形状の記述を行う手法である．この方法は接合や切削といった現実の造形手法との類似性を持つため，人間にとって馴染みやすい記述形式だと考えられるが，形状データはその形状を生成するための手続きであり，最終的に得られる表面形状に関する情報を明示的に含んでいない．そのため，作成した形状の確認のために画像を生成するような場合でも，一旦表面形状を算出する必要がある．そこで本章では，まず効率的に隠面消去処理を行うことができるスキャンライン法において，奥行き比較による可視面判定に代えて奥行き方向の１次元の集合演算処理を実行することにより，多面体をプリミティブに持つ CSG の記述から直接に陰影画像を生成する手法について論じる．次に CSG を積和形で表現し，その論理積項が持つ特性を利用して，この１次元集合演算処理を単純な計数処理によって実現する手法を提案する．さらに，この論理積項が持つ特

性や，プリミティブ形状の一様性を応用した効率化手法について論じる．

第４章では第３章で述べたアルゴリズムを，WED (Winged Edge Data) 構造で記述された多面体をプリミティブに持つ CSG により記述された形状に拡張する方法について論じる．WED 構造では 1 本の稜線を 2 枚の面で共有して表現するが，第３章で述べたアルゴリズムは一般の隠面消去処理アルゴリズムと同様，多面体を構成する面を互いに独立したものとして扱っている．このため，WED で記述された形状から陰影画像を生成するには，一般的にこのような独立した面への変換が必要になる. そこで本章では WED の記述を直接取り扱う手法について論じ, その際に WED がもつ面と稜線の接続情報を利用して１次元集合演算処理を大幅に効率化できることを示す.

第５章では前章までに述べてきたスキャンライン法による集合演算表示アルゴリズムを，曲面を含む凸形状をプリミティブにもつ CSG に適用する手法について論じる．本章の手法は画素単位に可視面の判定を行うが，プリミティブのスクリーンへの投影像の上端と下端および各スキャンライン上におけるその左端と右端を求め，プリミティブのスクリーン上への投影像に対して第３章で導入した計数にもとづく１次元集合演算処理を実行することによって，積和形で表された CSG の論理積項が表す形状のスクリーンへの投影像のシルエットを求め，その内部にある画素についてのみ奥行き方向の１次元集合演算処理を実行して可視面を決定する.

第６章では本研究で得られた成果を総括するとともに，その意義やインタラクティブな形状モデリングシステムに関する今後の筆者の展望について述べる.

なお，第１章および第６章における議論の参考として，インタラクティブな形状モデリングシステムで現在用いられている主な形状データの表現形式を補遺において概観する.

関連発表論文

A. 学会論文

1. 床井, 北橋: “凸な立体の集合演算によって定義された形状のスキャンライン法による陰影画像生成,” 情報処理学会論文誌, Vol. 30, No. 1, pp. 81-90, 1989.

2. 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.

3. 床井, 北橋: “スキャンライン法による会話的 CSG 表示アルゴリズム,” 映像情報メディア学会誌, Vol. 54, No. 7, pp. 1061-1068, 2000.

B. 国際会議

1. Tokoi, K. and Kitahashi, T.: “An Improved Scan-Line Algorithm for Display of CSG Models,” Proceedings of Information Visualization (IV2001), pp. 477-482, 2001.

C. シンポジウム，学会研究会

1. 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.

2. 床井: “リアルタイムソリッドモデラに関する研究,” 情報処理学会研究会報告, Vol. 96, No. 77 (96-CG-81), pp. 13-18, 1996.

D. 学術講演発表

1. 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,” 情報処理大会第 32 回全国大会論文集 (3), pp. 2083-2084, 1984.

2. 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論文集， P. S50, 1986.

目次

第 1 章　はじめに 1

1.1.　インタラクティブコンピュータグラフィックス 1

1.2.　インタラクティブな形状モデリング 2

1.3.　形状データの表現形式 4

1.4.　本研究の目的 6

第 2 章　走査変換時の図形の論理演算 9

2.1.　多角形の走査変換 9

2.1.1.　台形の走査変換 9

2.1.2.　任意の多角形の走査変換 10

2.2.　図形の論理演算 13

2.2.1.　交差図形を求める方法 14

2.2.2.　マスク画像を用いたクリッピング 15

2.2.3.　走査変換時の図形の論理演算 15

2.3.　まとめ 19

第 3 章　多面体の集合演算表示 21

3.1.　集合演算表示アルゴリズム 21

3.2.　スパニングスキャンライン法による多面体の隠面消去処理 22

3.3.　スパニングスキャンライン法の多面体の集合演算表示への拡張 25

3.3.1.　面の交差処理 26

3.3.2.　積和形による CSG の記述 27

3.3.3.　セグメントの表面判定 28

3.3.4.　計数による 1 次元集合演算処理 30

3.4.　効率化 32

3.4.1.　クリッピングによるセグメントの存在領域の限定 32

3.4.2.　物体形状の一様性の利用 34

3.5.　実験 36

3.5.1.　提案アルゴリズムにおける面数と処理時間の関係 36

3.5.2.　提案アルゴリズムにおけるスキャンライン数と処理時間の関係 37

3.6.　まとめ 39

第 4 章　WED で表現された形状の集合演算表示 41

4.1.　WINGED EDGE DATA 構造 41

4.2.　WED で記述された多面体の集合演算表示 42

4.2.1.　Active Polygon List (APL) の作成 42

4.2.2.　面の交差処理 43

4.2.3.　集合演算処理 45

4.3.　効率化 46

4.3.1.　サンプルスパン類似性の利用 46

4.3.2.　候補面の背後にある面の交差の無視 47

4.4.　実験 47

4.4.1.　提案アルゴリズムの処理時間 48

4.4.2.　効率化の効果 52

4.4.3.　半透明表示とインタラクティブ表示 54

4.5.　まとめ 55

第 5 章　凸形状プリミティブの集合演算表示 57

5.1.　スキャンライン法の任意の凸形状プリミティブへの応用 58

5.2.　プリミティブ選択アルゴリズム 59

5.3.　スクリーン上の集合演算処理 60

5.4.　プリミティブの実装 62

5.5.　実験 64

5.6.　まとめ 66

第 6 章　おわりに 67

6.1.　スキャンライン法 67

6.2.　ハードウェアとソフトウェア 69

6.3.　デザイナーのアプローチとエンジニアのアプローチ 71

謝辞 73

補遺　形状データの表現形式 75

参考文献 83

図表目次

図 1.1 制作のプロセス 3
図 1.2 境界による形状の記述 5
図 1.3 集合演算による形状の記述 5
図 2.1 多角形の走査変換 9
図 2.2 台形の走査変換 10
図 2.3 バケットソートによる y-Entry List の作成 11
図 2.4 y-Entry List の並べ替えアルゴリズム 11
図 2.5 Active Edge List の作成 12
図 2.6 クリッピングとマスキング 14
図 2.7 Sutherland-Hodgman のアルゴリズム 14
図 2.8 Weiler-Atherton のアルゴリズム 15
図 2.9 複数の図形の走査変換 16
図 2.10 計数によるランの論理積演算 17
図 2.11 計数によるランの論理和演算 18
図 2.12 計数によるランの差演算 18
図 3.1 スキャンライン法による隠面消去処理の概念 22
図 3.2 Active Polygon List の作成 23
図 3.3 サンプルスパン内の可視判定 24
図 3.4 サンプルスパン上の 1 次元集合演算処理 25
図 3.5 二分法によるサンプルスパンの分割 26
図 3.6 エレメントリストとセグメントリストによる形状の表現 28
図 3.7 セグメントの表面判定 29
図 3.8 計数による 1 次元集合演算 31
図 3.9 視野錐台 32
図 3.10 セグメントが表す部分物体の外接箱 33
図 3.11 サンプルスパンの類似性の利用 34
図 3.12 サンプルスパンの分割位置 35
図 3.13 多面体の差集合演算のモデル図 36
図 3.14 二つの多面体の差集合演算処理における面数と処理時間の関係 36
図 3.15 スクリーンのスキャンライン数と処理時間の関係 37

図 3.16 サンプル形状 38
図 4.1 Winged Edge Data 構造 42
図 4.2 稜線による面の APL への登録・削除パターン 42
図 4.3 Active Polygon List の作成 43
図 4.4 可視面の判定を誤る場合 44
図 4.5 サンプルスパンの分割 45
図 4.6 可視面判定の打ち切りによる生成画像の変化 46
図 4.7 9 個の多角柱からなる形状 48
図 4.8 多角形の側面数と全処理時間の関係 49
図 4.9 多角形の側面数と全集合演算処理時間の関係 50
図 4.10 スキャンライン数と全処理時間の関係 50
図 4.11 スキャンライン数と全集合演算処理時間の関係 51
図 4.12 1 次元集合演算処理の平均処理時間 52
図 4.13 図 4.7 の形状に対する効率化の効果 53
図 4.14 隠れた交差が含まれる形状 53
図 4.15 図 4.14 の形状に対する効率化の効果 54
図 4.16 マウス操作によるリアルタイム形状変形表示 55
図 5.1 プリミティブの境界情報 58
図 5.2 プリミティブ選択アルゴリズム 59
図 5.3 計数によるスクリーン上の集合演算処理 61
図 5.4 視野空間とスクリーンの関係 62
図 5.5 プリミティブの投影像の上下端 63
図 5.6 スキャンライン平面上のプリミティブの交差線の左右端 64
図 5.7 本手法による画像の生成例 66
図 補遺 1 境界情報を用いた形状の表現 75
図 補遺 2 局所変形操作と丸め変形操作 76
図 補遺 3 プリミティブインスタンス法 77
図 補遺 4 Voxel 表現 77
図 補遺 5 Oct-Tree 表現 78
図 補遺 6 集合演算と CSG の木構造による表現 80
図 補遺 7 スイープ 80
図 補遺 8 メタボール 81

表 3.1 集合演算の規則 31
表 5.1 ランダムな球の描画に要した時間 64
表 5.2 プリミティブ数固定時のセグメント数に対する処理時間 65

第1章> はじめに>

1.1.> インタラクティブコンピュータグラフィックス>

1940 年代にコンピュータがこの世に誕生したのち，1950 年代には，既にマサチューセッツ工科大学（以降 MIT と略す）の研究者らによって，コンピュータディスプレイ（当時はオシロスコープ）上に描かれた図形を，ポインティングデバイス（当時はライトペン）を用いて操作するという，インタラクティブなコンピュータ支援設計 (Computer Aided Design，以降 CAD と略す) システムが構想されていた[1]. そして 1960 年代初頭には, 当時 MIT の大学院生であった Ivan Sutherland によって，ついに実際にそれを実現するシステム “SKETCHPAD” が開発された[2]．この “SKETCHPAD” というシステムの登場こそが，コンピュータサイエンスの領域に，コンピュータグラフィックス（以降，CG と略す）という分野を誕生させたと考えられている[3]．このようにコンピュータを用いたインタラクティブな図形操作に対する欲求は，コンピュータによるリアルな映像生成と並んで，CG 分野の黎明期から，その発展の原動力となってきた.

しかし，現実の 3 次元世界を構成している「形状」は，当時のコンピュータで取り扱うにはあまりにも複雑で膨大な情報であり，陰影のついた立体感のある画面表示を見ながら対話的に形状を操作できる環境の実現には，1980 年代半ばのグラフィックワークステーション (GWS) の登場を待つ必要があった.

GWS はマイクロプロセッサ技術の向上により実現された，小型ながら非常に強力な処理能力を持つ中央演算処理装置 (CPU) に，座標変換やクリッピング，ラスタライズ，隠線・隠面消去，陰影付けなどの機能を実現する図形処理ハードウェアを密接に結合したものである．これによってデザイナーが画面に表示されている立体形状を対話的に操作し，その結果を即座に画面表示としてフィードバックできる環境が実現できるようになった．これは，特に工業製品の意匠設計（インダストリアルデザイン）の手法などに大きな変革をもたらしたほか, それまで主としてスーパーミニコンピュータが用いられてきた CG による映像制作が，映画の一部の特殊効果にとどまらず，長編映画そのものの制作などにも本格的に応用される契機にもなった.

現在では，映像表示装置に関する技術革新に付随してリアルな大画面映像のリアルタイム生成に対する需要が高まってきており，GWS のアーキテクチャも

並列処理にもとづく大規模なものへと変貌してきている．これらはバーチャルリアリティ (VR) やテーマパークのアトラクション等に応用されている．

一方，マイクロプロセッサ技術の向上は，1980 年代初頭に登場したパーソナルコンピュータ (PC) の性能も飛躍的に向上させた．また，その初期の画面表示機構は２次元図形表示が中心であったが，そこにも GWS の図形処理ハードウェアのアーキテクチャが導入され，最近では非常に精緻な３次元映像をリアルタイムに生成できるものも登場している．現時点においてその応用は，ゲームを中心としたアミューズメント目的のものが主体となっているものの，GWS の技術は民生用のゲーム機にも移転され，リアルタイム性とともに生成される画像の品質を競い合いながら，今も発展を続けている．

この状況は CG による仮想世界映像の生成を非常に一般的な技術として社会に浸透させ，インターネットの普及とともに映像メディアコンテンツに対する膨大な需要を喚起することとなった．そしてそれは，インダストリアルデザインなどへの応用と比較して，大量の「立体形状データ」を短期間に低いコストで生産するという要求を生み出した．

このため，レンジファインダ等を用いた形状計測や実画像をベースにしたモデリングなど，実物の形状データの作成を自動化する試みが盛んに取り入れられる一方で，複数画像の合成による任意視点画像の生成技術の誕生など，従来のように対象物モデルを必要としない手法も開発されている．

しかし，ゲームや仮想世界を描くような映像コンテンツの制作などでは，単に現実に存在する形状を表現するだけでは不十分であり，その上にデザイナーの自由な発想にもとづく形状制作，すなわち「造形」を加えることが必要不可欠となる．そのような造形システムには，少なくともデザイナーの発想を妨げないインタラクティビティが要求される．

加えて，PC によってエンドユーザー自身が映像制作や造形を行うことが可能になった現在では，そのような造形システムに対して，直感的な分かり易さやユーザーの発想を容易に表現できる使い易さも要求される．

このような現状は，インタラクティブコンピュータグラフィックスの要素技術の研究開発に対して，今なお根強い社会的要請があることを示している．

1.2.> インタラクティブな形状モデリング>

コンピュータを使った造形や映像制作は，デザイナーのイメージするシーン

をコンピュータ内の仮想的な空間中に再構成する作業にほかならない．しかし，その際にデザイナーが最初から完成したイメージを持っていることは少ない．多くの場合，デザイナーは最初におおまかな形状をコンピュータに入力し，画像表示などでその形状と自分のイメージを照合する．そして，それをもとに形状の修正や詳細化を行い，イメージを具体化していく（図 1.1）．すなわち，デザイナーはこのような試行錯誤を経てアイデアやコンセプトをコンピュータ上で具体化するとともに，自分の心的イメージ自体も完成させるのである[4]．

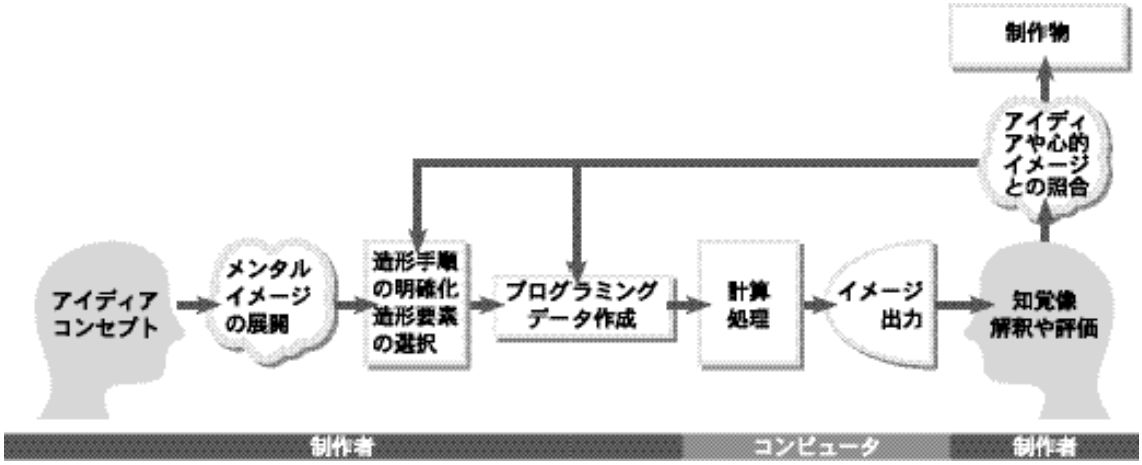


図 1.1 制作のプロセス[5]

このようにコンピュータを使った造形は，コンピュータと人間とのインタラクションの上で達成される．このときコンピュータは，デザイナーの心的イメージを表現するためだけでなく，デザイナーの発想を支援し，心的イメージを完成する役割も果たす．したがってデザイナーの心的イメージを記述するモデリングや，記述されたイメージをデザイナーにフィードバックするレンダリングは，コンピュータと人間とのインタラクションによる造形において，デザイナーの創造性を引き出すのにきわめて重要な側面を持っている．

またコンピュータを用いた造形は，現実世界に存在するさまざまな制限にとらわれない多様な表現が可能な反面，現時点ではデザイナーが現実世界で用いている造形手法をそのまま適用することが難しく，**もどかしい思い**をする場合が少なくない．形状モデリングシステムには複雑な物体の形状を正確に表現することに加えて，デザイナーの心的イメージを効率よくコンピュータに取り込むユーザーインターフェースを提供することが求められる．そのためには，デザイナーがイメージを容易に表現できる形式でコンピュータ上の形状データを操作でき，またデザイナーが直感的に理解できる形式でコンピュータ上の形状

データを再現できることが望ましい．

３次元形状を人間が直感的に理解できる形式として，形状を表す実体そのものを用いることが最も優れた方法であり，実際にそのような実体を制作する装置も開発されている．しかし，コンピュータを用いて現実の見え方に近い陰影画像を，視点や対象の位置を変化させながら生成することによっても，実体に近い実物感を得ることができる．しかしそのためには，コンピュータと人間との間に，形状を記述するための何らかの言語を介在させなければならない．ここではそれを形状データの表現形式と呼ぶことにする．

1.3.> 形状データの表現形式>

形状データの表現形式によって，形状のコンピュータ内部における表現，すなわちデータ構造が規定されると同時に，ユーザーが認識している形状の構成要素から，データ構造の個々の要素へのマッピングが行われる．たとえば，線分を 2 個の端点の座標値と，それらを直線で結ぶという情報との組み合わせにより表現すると規定したとする．このとき「線分」というキーワードは，ユーザーにおいてそのデータ構造に与えられたニーモニックコードとして機能する．

したがって形状データの表現形式は，ユーザーが対象形状の「なりたち」をどのようにモデル化するかによって異なる形式となり得る．また，そこには実装上の制限も存在する．仮に立体形状を非常に小さな粒子の集合体としてモデル化したとして，その方法が実用的かどうか，あるいは実装可能かどうかについては，その目的や応用，もしくは実装環境などの条件をもとに判断される必要がある．なお，実際に CG 等で用いられている主な形状データの表現形式については，補遺 (P.75) を参照されたい．

インタラクティブな形状モデリングにおいてユーザーにフィードバックしなければならない最も重要な情報は，対象形状の「見かけ」を形成する表面形状である．したがって，この表面形状そのものを形状データとして用いることができる．ここで表面とは，立体の内部と外部を隔てる境界であり，「面」の集まりとして記述される．それゆえこのような形状データの表現形式は，一般に**境界表現** (Boundary Representation, B-Reps) と呼ばれている．

境界表現は立体形状を正確に表現でき，画像の生成なども効率良く行うことができるため，一般の形状モデリングシステムにおいて広く使用されている．

しかしこの方法では，ユーザーの形状操作の対象となる形状要素も，基本的

に頂点や稜線あるいは面などの境界情報が中心となる．これらの低レベルの形状要素を一つ一つ指定して形状を操作することは，「見かけ」に対する細かな制御が可能な反面，かなり煩雑なものとなる．また，立体の局所的な形状に着目して操作を行うため，形状の全体的な見通しも悪い．そのためこの方法は，デザインの初期段階においてデザイナーがイメージを形成するために用いる形状データの表現形式として，使いやすいものであるとは言えない．

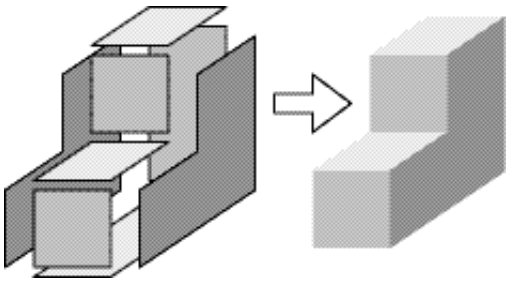


図 1.2 境界による形状の記述

そこで多くの形状モデリングシステムでは，あらかじめ組み立てられたパーツのような，高レベルの形状要素を組み合わせて形状を作成する手段や，低レベルの形状要素を一括して操作するようなツールを用意するなどして，形状モデリング作業の効率化を図っている．集合演算処理も，そのようなツールの一つとして用いられている．

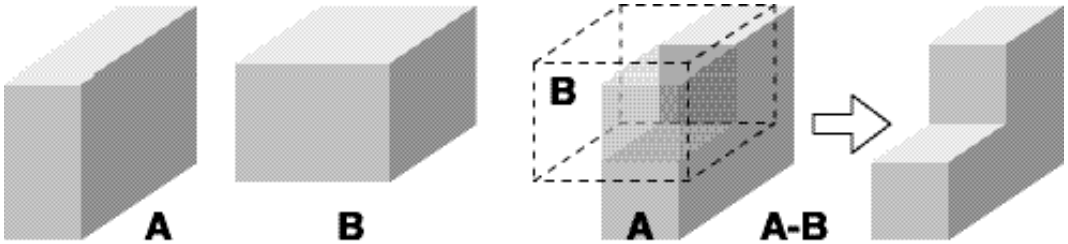


図 1.3 集合演算による形状の記述

集合演算処理は，もとになる二つの立体形状の間に，論理和，論理積，および差（補集合との論理積）のブール結合関係を指定することによって，新しい形状を作成する手法である．また，得られた形状に対して繰り返し集合演算処理を実行することにより，複雑な形状の作成を行うことができる．

この方法では，もとの立体形状が単純なものであっても，集合演算の繰り返しを経て複雑な立体形状を生成できる．そこで，このもとの形状としてあらかじめ用意しておいた基本立体（プリミティブ）を用い，それらの間にブール結

合関係を指定することによって形状を記述することができる．このような形状データの表現形式は CSG (Constructive Solid Geometry) と呼ばれている．

一般的な工業製品の形状は比較的単純な形状の組み合わせで構成されることが多いため，プリミティブとして直方体や２次曲面のような単純な形状を用いても効率よく形状の記述を行うことができる．また，この記述は目的の形状を生成するための手続きに相当するため，工作機械などによる実体モデルの作成手順が明確であるという特徴も持つ．さらに，この記述はデザイナーが目的の形状を得るまでに経た形状の生成過程とみなすこともできるため，デザイナー自身の造形手法や試行錯誤を反映しやすく，イメージの形成にも有効な手法だと考えられる．

反面，形状データが形状を生成するための手続きで記述されているということは，その記述によって表される形状が，実際にその手続きを実行しなければ得られないことを意味する．すなわち，デザイナーへのフィードバックのために作成途中の形状の画像を生成する際にも，実際にこの手続き，すなわち集合演算処理を実行して表面形状を求める必要がある．集合演算のような干渉問題は，一般に形状が複雑になる（データ量が増大する）につれて処理に長い時間を要するようになる．特に，デザイナーの試行錯誤の結果によりブール結合関係の記述が複雑化した場合には，この影響が非常に大きくなる．

形状データの表現形式は，デザイナーにとっては造形の素材であると同時に，空間を認知して形状を操作するためのスキームとしても用いられる．しかし，それをコンピュータへ実装する際に何らかの制限が加えられた場合，その実装とデザイナーのイメージとの間には，時として大きな**セマンティックギャップ**が発生する．デザイナーが感じる**もどかしさ**の原因はここにある．

このセマンティックギャップを埋めるためには，より現実の造形手法に近い，あるいはデザイナーが直感的に理解しやすい形式で形状を表現することが望ましい．しかし，そのような形式が必ずしもコンピュータにとって取り扱い易いものであるとは限らない．形状記述から画像などの出力を生成するのに長い時間がかかれば，逆に対話的に形状の作成を行うことが困難になってしまう．

1.4.> 本研究の目的>

本研究は CSG による形状モデリングを対話的に行うシステムへの応用を目指した，高速かつ堅牢なレンダリングアルゴリズムの開発を目的とする．

立体形状をコンピュータ上で表現する場合に，もっともよく用いられる形状データの表現形式は境界表現であろう．既に述べたとおり，この方式は複雑な立体の形状を正確に表現することが可能であり，また立体の表面形状の情報を明示的に保持しているため，効率良く画像の生成を行うことができる．しかし，これを直接操作して形状を作成することは，一般に手間のかかる作業となる．

このため，境界表現を内部表現に持つ形状モデリングシステムにおいても，形状操作のユーザーインターフェース言語として，CSG を採用する場合がある．

CSG による形状記述は，平行移動や回転により空間中で位置決めされた複数のプリミティブと，それらの間のブール結合関係からなる．このプリミティブのブール結合関係に対して，接合や切削などの変形操作をメタファーとして与えることにより，コンピュータによる形状作成に馴染みのないユーザーにおいても理解しやすいユーザーインターフェースが構築できる．

しかし，この手法では作成した形状を画像で確認する場合にも，CSG の記述から局所変形操作の命令列を生成し，それを対象の形状データに適用して結果の境界表現データを得る必要がある（集合演算処理）．試行錯誤が繰り返される意匠設計において形状確認の度にこの処理を実行していたのでは，形状操作のインタラクティビティを維持することが困難になる．また，得られた境界表現データに対して CSG の記述のレベルでの形状の修正や変更を行おうとすれば，集合演算処理を取り消して元の形状データを回復する必要がある．

これらの課題を克服できれば，CSG 表現はインタラクティブな形状モデリングシステムに適した形状モデルの記述形式と言える．そこで本論文では，走査変換にもとづく隠面消去アルゴリズムであるスキャンライン法が CSG モデルから直接陰影画像を生成するために利用可能な情報も保持している点に着目し，それを活用して CSG モデルから集合演算後の形状の陰影画像を直接かつ高速に生成するアルゴリズムを提案する．

このために本論文では，準備として第 2 章において走査変換による多角形の塗りつぶしアルゴリズムについて概説し，それを用いたクリッピングやマスク画像に依らない平面図形同士の論理演算アルゴリズムについて述べる．そして第 3 章において，この手法の 3 次元空間への拡張による，CSG モデルの高速陰影画像生成アルゴリズムを提案する．さらに第 4 章では，WED 構造がもつ面の接続情報などを用いた効率化手法について論じたのち，境界表現と CSG のハイブリッドな形状モデリングシステムの可能性について考察する．最後に第 5 章

では，一般の凸形状をプリミティブにもつ CSG モデルに対する本手法の応用について述べ，本手法の多面体以外への拡張について考察する．

第2章> 走査変換時の図形の論理演算>

Sutherland の “SKETCHPAD” に発するインタラクティブコンピュータグラフィックスのシステムには，当初，図形表示にオシロスコープの機構を用いたランダムスキャン型ディスプレイが用いられていた．しかし 1970 年代には，テレビジョンの表示機構を利用したラスタースキャン型ディスプレイが主流となり，それまでの線画中心の図形表示から，ハーフトーン画像（陰影画像）による図形表示が実現されるようになった．

ラスタースキャン型ディスプレイは図形を画素（輝点）の集合で表すため，図形表示を行うには，座標値や線分の情報からなる図形データから，画素の集合であるラスター画像への変換を行う必要がある．この処理を走査変換 (Scan Conversion) という．線分で囲まれた領域に含まれる画素をスキャンライン（走査線）単位に塗りつぶす処理は，多角形の走査変換と呼ばれる．

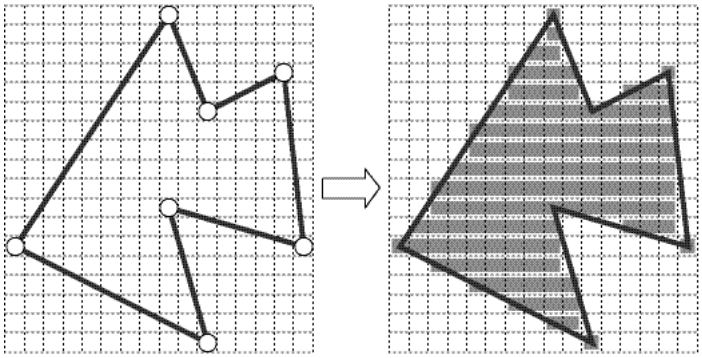


図 2.1 多角形の走査変換

本論文で提案するアルゴリズムは，スキャンライン法による隠面消去アルゴリズムを基礎にしている．本章では，スキャンライン法による隠面消去アルゴリズムを CSG モデルに適用する方法について解説するための準備として，多角形の走査変換の際の，図形の論理演算処理について述べる．

2.1.> 多角形の走査変換>

2.1.1.> 台形の走査変換>

まず簡単な例として，図 2.2 に示す，上底と下底がスキャンラインと平行な台形の走査変換を考える．

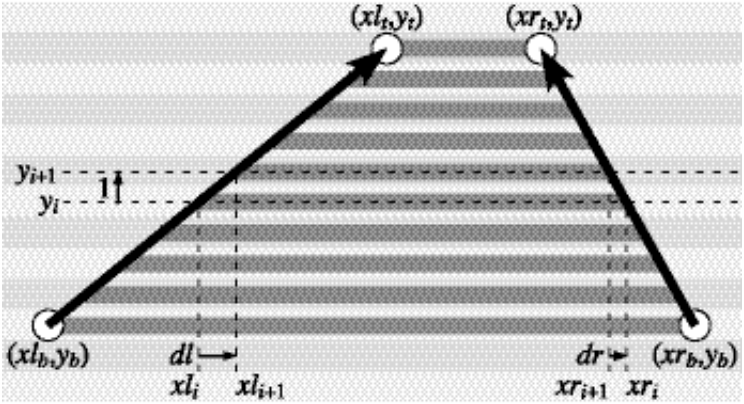


図 2.2 台形の走査変換

この台形を塗りつぶすには，台形の左右の斜辺と，上底と下底の間にある各スキャンラインとの交点を求め，左右の交点の間に水平線を描けばよい．このとき各スキャンラインにおける左右の交点の x 座標値は，下底の両端の x 座標値に斜辺の y 軸に対する傾きを増分として積算することによって得られる．なおこの増分は実数値となるが，実際には Bresenham のアルゴリズム[6,7]をもとにした方法を用いて，整数計算のみで実行される．

2.1.2.> 任意の多角形の走査変換>

前節で述べた台形の走査変換の手法を凹部や穴を含む任意の多角形に拡張するには，多角形からこの台形の斜辺のような「辺の対」を発見する方法が問題になる．あらかじめ多角形を，この「辺の対」の発見が容易な台形や三角形に分割することも可能だが，ここではバケットソート法を用いてすべての辺を並べ替えたのち，処理対象のスキャンラインと交差する辺のみを抽出して，効率良くこの「辺の対」を発見する方法[8,9]について述べる．

Step>1:>バケットソート>

(1) まず，生成するラスター画像のスキャンライン数を要素数に持つ配列（バケット）を用意し，全要素を空で初期化する．一方，この配列につなぐ線形リストのセルを用意し，それらに多角形の各辺の識別子を格納しておく．
(2) 次に，これらのセルを，格納している辺の下端の y 座標値の位置にあるバケットにつないでゆき，スキャンラインごとに y-Entry List を作成する．その際，図 2.3 の e_1 のように水平な辺は除外しておく．
(3) その後，各 y-Entry List に登録されているセルを，対応する辺の下端の x 座標値の順に並べ替える．

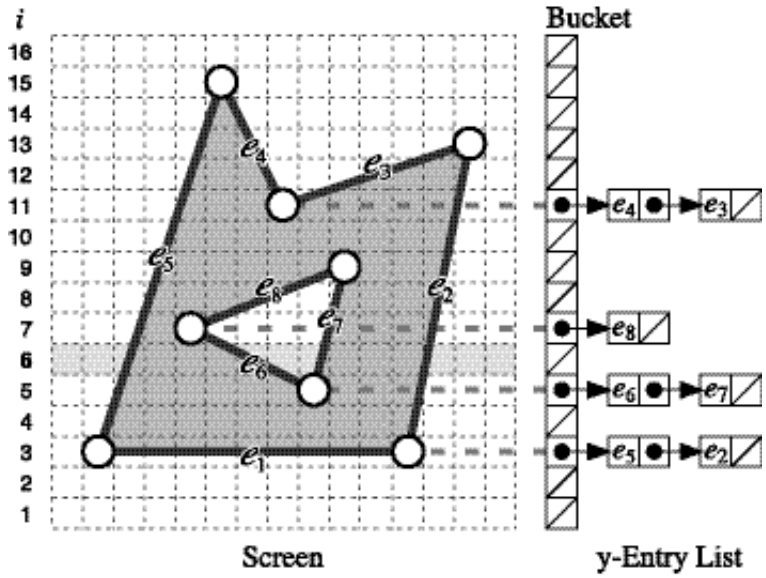


図 2.3 バケットソートによる y-Entry List の作成

この手順の (3) で実行する y-Entry List の並べ替えには，以下に示すクイックソート法に準じたアルゴリズムを用いる．まず，リストの先頭のセルを取り出し，その x 座標値をキーとする．このキーとリストの残りのセルの x 座標値を比較し，そのセルをキーより大きいもの／キーと等しいもの／キーより小さいものの３つのリストのいずれかに振り分ける．その後キーより大きいもののリストとキーより小さいもののリストについて同じ処理を再帰的に実行し，最後にこの３つのリストを結合する．

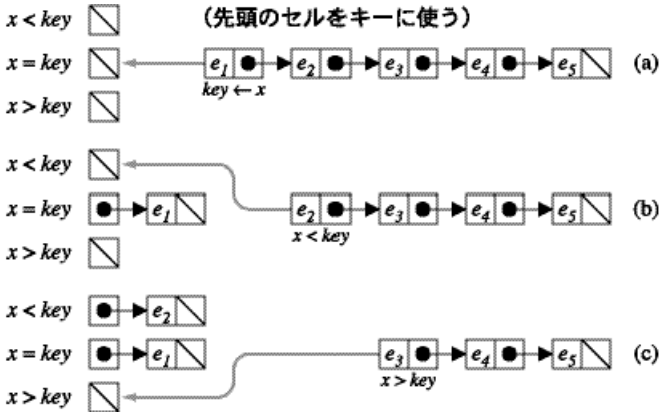


図 2.4 y-Entry List の並べ替えアルゴリズム

この並べ替えアルゴリズムの計算量はクイックソート法と同等の $O(n \log n)$ となる．なお，このアルゴリズムではリストの先頭のセルをキーに用いるため，再帰処理の際に以前に使用したキーと同じ値が使われないように，そのような

セルを独立したリストに分離している．

Step 2: Active Edge List (AEL) の作成

Step 1 で作成された y-Entry List に登録されている辺を，バケットの下端から順次 AEL に併合してゆく．この手続きにより，AEL には処理対象のスキャンラインと交差する辺が常に保持される．この手順を以下に述べる（図 2.5）．

(1) AEL の初期値を空とする．

(2) 処理対象のスキャンラインの y-Entry List に登録されている辺を一つずつ取り出す．取り出した辺を e_j とするとき，x_j にその下端の x 座標値，d_j にその辺の y 軸に対する傾斜，h_j にその辺の高さを記録して，AEL に登録する辺のリストを作成する．

(3) 作成したリストを現在の AEL に併合する．これにはマージソート法を用い，各辺が AEL 上で x_j について昇順に並ぶようにする．

(4) AEL が空でなければ，以下の処理を実行する．

(5) AEL 上で隣り合う辺を対にすれば，走査変換された図形の，そのスキャンライン上でのランが得られる．従って，その辺の対を結ぶ水平線を処理中のスキャンライン上に描けばラスター画像が得られる．

(6) AEL に登録されている各辺について $h_j \leftarrow h_j - 1$ とし，$h_j = 0$ となった辺を AEL から取り除く．残りの辺については $x_j \leftarrow x_j + d_j$ として，次のスキャンラインにおける辺の x 座標値を求める．

(7) 次のスキャンラインの処理に移る．上端のスキャンラインに達すれば，処理を終了する．

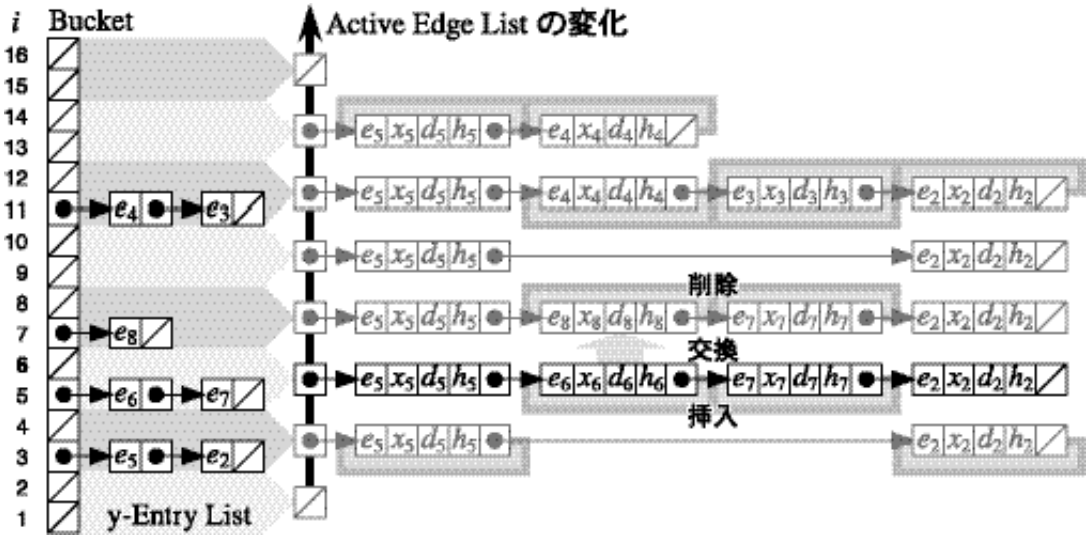


図 2.5 Active Edge List の作成

図 2.5 において，$i = 3$〜4 のスキャンラインでは e_5 と e_2 が対となり，この間に水平線が描かれる．$i = 5$ のスキャンラインにおいて e_6 と e_7 が新たに進入してくるため，ここからスキャンライン上のランは e_5 と e_6 の対の間，および e_7 と e_2 の対の間の二つに分割される．さらに $i = 7$ のスキャンラインでは $h_6 = 0$ となって e_6 が AEL から取り除かれるが，新たに e_8 が進入して e_6 と置き換わる．$i = 9$ のスキャンラインでは $h_7 = h_8 = 0$ となって e_7 と e_8 が AEL から取り除かれ，AEL には e_5 と e_2 の対が残る．

以上の処理により，AEL には常に処理中の「辺の対」が登録される．各辺は AEL に登録されている間，2.1.1. 節で述べた台形の走査変換と同様に，そのスキャンラインにおける x 座標値を保持している．このため，AEL をたどるだけで容易に処理中のスキャンライン上における図形のランが得られる．すなわち，この処理により線分で構成される図形データ（ベクトルデータ）を，ランレングス形式のラスター画像に変換することができる．

このアルゴリズムの計算量は，最初のバケットソートおよびマージソートを含む AEL 作成において，それぞれ辺の数 n に対して $O(n)$ である．一方 y-Entry List の並べ替えに要する計算量は，y-Entry List の長さ m に対して $O(m \log m)$ となるが，バケットソートによって辺のセルが各バケットに分配されるため，この並べ替えの計算量は他の処理に比べて小さい（スキャンライン数が l のとき m は平均して $m = n/l$ となる）．このため，この並べ替えの計算量の増加は全体の計算量にあまり影響せず，全体の計算量はほぼ $O(n)$ と見なすことができる．

なお，文献 8 および 9 では，現在処理中のスキャンラインと交差している辺 (Active Edge) の管理にテーブル (Active Edge Table, AET) を用いているが，本研究ではあらかじめ y-Entry List を並び替えておき，他の Active Edge との併合にマージソート法を採用したため，Active Edge の管理にも線形リスト (Active Edge List, AEL) を用いている．これにより，不要なデータの移動や並べ替えを排除できる．

2.2.> 図形の論理演算>

図形表示の際には，図形の表示領域からはみ出た部分を切り取るクリッピングや，図形の一部分を別の図形で切り取るマスキングなどの処理が行われることが多い（図 2.6）．これらはいずれも図形の共通部分を抽出する，図形同士の論理積演算に相当する．

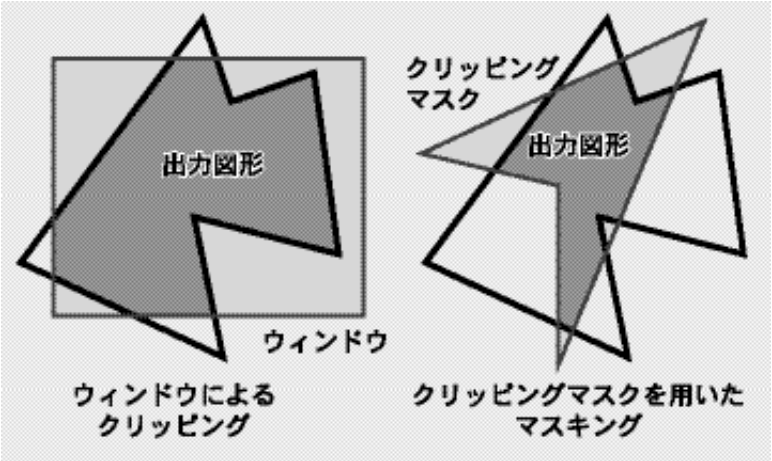


図 2.6 クリッピングとマスキング

2.2.1.> 交差図形を求める方法>

図形表示において，図形の表示領域からはみ出た部分を切り取るクリッピングには，Sutherland-Hodgman のアルゴリズム[10]がよく知られている．これは矩形のウィンドウから表示図形がはみ出る部分を，ウィンドウの辺ごとに順次切り取っていく方法である．このため，この方法では一つの辺によってクリップされた多角形（中間多角形）を一旦保持しておく必要がある．

切り取りに使うウィンドウは必ずしも矩形である必要はないが，凹部を含んではならい．またこのアルゴリズムの計算量は，ウィンドウの辺の数 n，表示図形の辺の数を m としたとき，m 個の辺に対するクリッピング処理を n 回繰り返すため，$O(mn)$ となる．

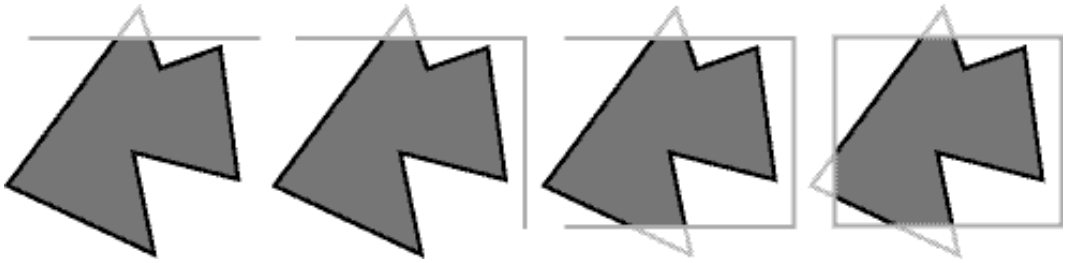


図 2.7 Sutherland-Hodgman のアルゴリズム

Sutherland-Hodgman のアルゴリズムでは, クリッピングに用いることができるウィンドウ（マスク）形状に制限があった．これに対し，任意の形状の多角形同士のクリッピングとして Weiler-Atherton のアルゴリズム[11]が知られている．

このアルゴリズムは，重なり合う一方の多角形の辺を，たとえば右回りにたどる．その際，もう一方の多角形との交差点に出会えば，そこでより右側（内

側）の辺に「乗り換える」．そして既に通過した頂点あるいは交差点に出会ったところで，処理を終了する．

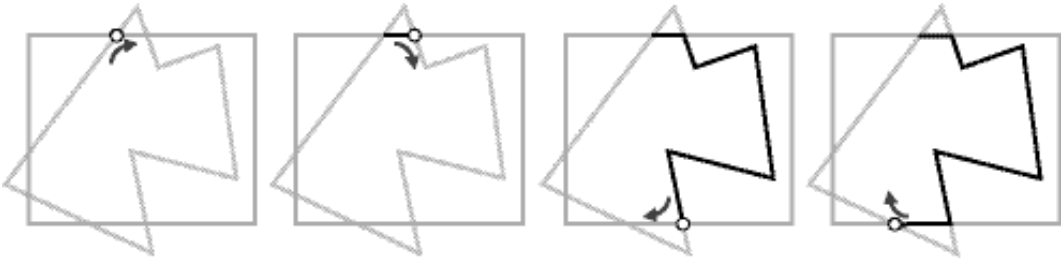


図 2.8 Weiler-Atherton のアルゴリズム

この方法は，あらかじめ二つの多角形の交差点を検出し，それを含めた二つの頂点リストの作成を行う．二つの多角形の辺の数をそれぞれ m, n とすれば，それらの交差点の検出には $O(mn)$ の計算量が必要になるが，辺（頂点）の巡回は $O(m+n)$ の手間ですむ．

なお，このアルゴリズムは元々多面体の隠面消去アルゴリズムとして開発されたもので，多角形同士の重なり合う部分を分離することによって，可視面の決定を容易にしようとしたものである．

2.2.2.> マスク画像を用いたクリッピング>

Weiler-Atherton のアルゴリズムは強力だが, 多角形の辺が重なっている部分などの例外的な部分の処理に複雑な手続きが必要になる．また，複雑な図形同士の論理演算では，安定した結果が得られない場合がある．

そこで，いかなる場合でも安定した処理を実現するために，図形を走査変換して得たラスター画像をマスク画像に用いて，図形の論理演算を画像処理として実行する場合もある．マスク画像や対象の図形をビットマップ画像とすれば，図形の論理演算を単純な画素の演算のみで実現できるため，安定した処理が期待できる．また，このような演算はハードウェア化も容易である．

一方，走査変換では直接ランレングス画像が得られるため，ランレングス画像のまま論理演算を実行することによっても，効率化を図ることができる．

2.2.3.> 走査変換時の図形の論理演算>

前節において，図形を走査変換によりラスター画像化して論理演算を行う方法について述べた．この方法は安定した結果を得られるが，演算の対象となる図形ごとに独立して走査変換を行い，それぞれの結果を保存しておく必要がある．そこで，ここではランレングス画像を用いた画像の論理演算に相当する処

理を，一度の走査変換で実現する手法について述べる．

Step›1:›バケットソート›

2.1.2. 節で示した走査変換アルゴリズムと同様に，すべての多角形の辺を一括してバケットソート法により並べ替えて，y-Entry List を作成する．

Step›2:›Active›Edge›List›(AEL)› の作成›

これも 2.1.2. 節で示した走査変換アルゴリズムと同様に，y-Entry List に登録されている辺を Bucket の下端から順次 AEL に併合してゆく．その際，個々の多角形を識別するために，辺が属している多角形の識別子を AEL に記録する．なお，図 2.9 では AEL に登録されているその他の情報（辺の x 座標値など）は省略している．

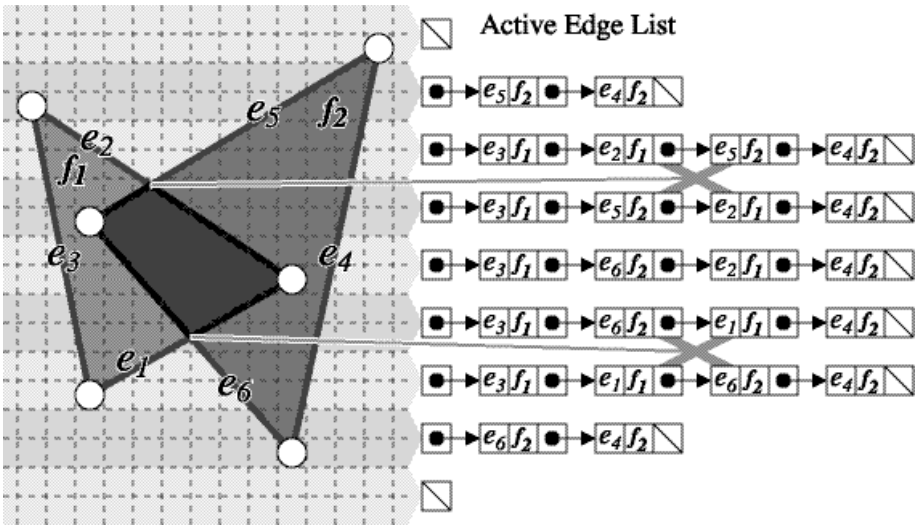


図 2.9 複数の図形の走査変換

また，図形の論理演算を行うためには，辺同士の交差を処理する必要がある．そのため，一本のスキャンラインの処理が終了するごとに，AEL に登録されている辺を，次のスキャンラインにおける x 座標値で並べ替える．その際 AEL に登録されている辺の順序は，隣接するスキャンラインの類似性により直前のスキャンラインにおける順序から変化していないか，変化していても一部の辺が入れ替わっているだけである場合が多い．図 2.9 では，e_1 と e_6 の交点および e_2 と e_5 の交点を含むスキャンラインにおいて，この入れ替えが発生する．そこで，この並べ替えにはバブルソート法を用いる．

Step>3:> ランの論理演算>

次に，ランの論理演算の実現方法について述べる．

(1) 変数 C を演算の対象になる多角形の数で初期化する．図 2.9 の例では $C=2$ である．

(2) 個々の多角形 f_k のそれぞれに対して変数 O_k を用意し，それらを -1 で初期化する．

(3) AEL を先頭からたどり，取り出した辺が属している多角形 f_k に割り当てられた変数 O_k を用いて，$C \leftarrow C + O_k$ として C に積算していく．

(4) O_k の符号を反転し，次の辺の処理に移る．

以上の処理において，$C \leftarrow C + O_k$ の結果 $C=0$ となった辺をランの起点とし，$C \neq 0$ となった辺をランの終点とすれば，多角形が重なる部分の形状，すなわちこれらの図形の論理積形状のランレングス画像が出力される．

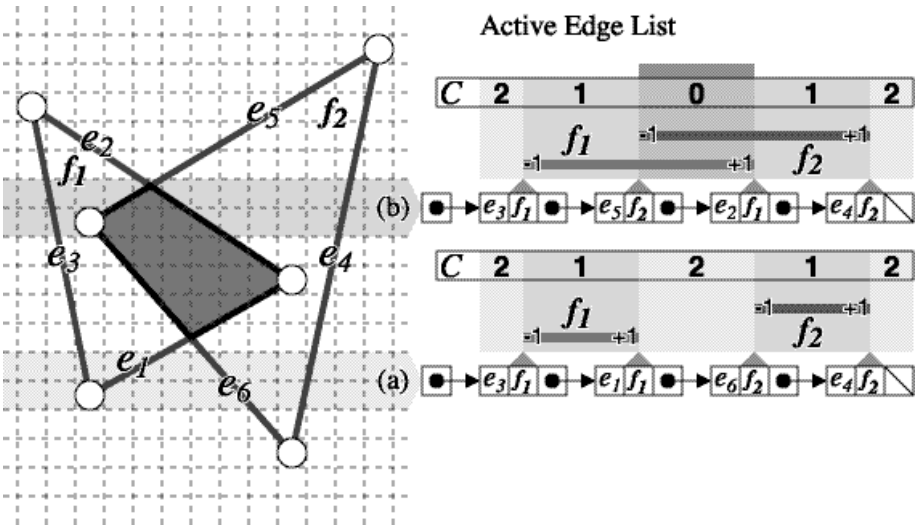


図 2.10 計数によるランの論理積演算

図 2.10 の例について説明する．(a) のスキャンラインの範囲において AEL を先頭からたどれば，最初に e_3 が現れる．これは f_1 の左端なので C が 1 減じられる．ところが次に e_1 が現れ，これは f_1 の右端なので C が 1 増され，結局 C は 0 にはならない．一方 (b) のスキャンラインの範囲では，(a) と同様 e_3 が現れて C が 1 減じられるが，次に f_2 の左端である e_5 が現れて C が再び 1 減じられて $C=0$ となり，ここからランが出力される．そして，さらに次の e_5 によって C が 1 増され $C \neq 0$ となるため，ランはこの位置で終了する．

この方法を用いて，論理和や差の形状を出力することもできる．論理和形状を出力するには，図 2.10 の例では $C \neq 2$ となった辺をランの起点とし，$C=2$

となった辺を終点とすればよい. このとき C の初期値を多角形の数に関わらず 1 にしておけば，ランの起点を $C \leq 0$，終点を $C > 0$ で判定できるので，この処理を論理積と統一できる（図 2.11）.

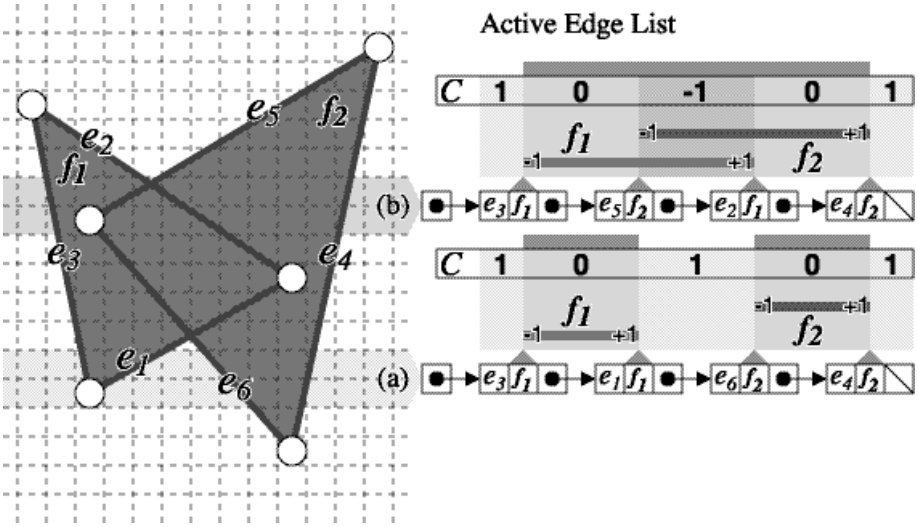


図 2.11 計数によるランの論理和演算

また差の形状を出力するには，C の初期値を引かれる側の多角形数とし，引く側の多角形の O_k の初期値を 1 とすればよい．図 2.12 の例のように f_1 から f_2 を引く場合は，C の初期値を 1 とし， O_2 の初期値を 1 とする.

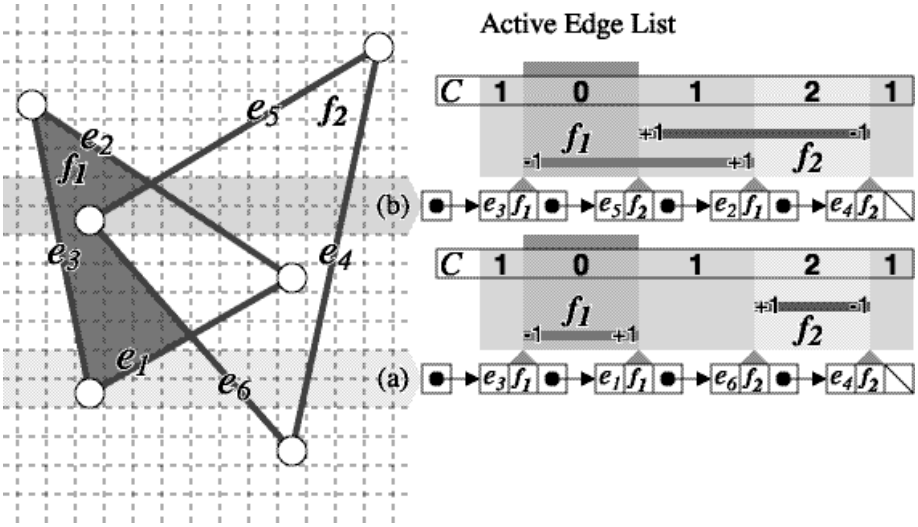


図 2.12 計数によるランの差演算

この方法は多角形の形状や数に制限がない上，中間多角形やマスク画像を保存しておく必要もない．演算の手間は AEL の更新時のバブルソート法による並べ替えの分だけ増加するが，これも隣接するスキャンラインの類似性から AEL

に登録されている辺の順序の検査のみで済む場合が大半を占める．

さらに，隣接するスキャンラインの類似性により，次のスキャンラインの処理に進む際に以下の条件のいずれにも該当しなければ，前述の計数処理によって検出されたランの始点と終点となる辺が変化することはない．

(a) 現在の AEL から辺が取り除かれたとき

(b) 次のスキャンラインの y-Entry List から辺が AEL に併合されるとき

(c) 辺が交差しているとき

このようなスキャンラインの数は, 表示しようとする図形の複雑さに依存するが，表示画面のスキャンライン数には依存しない．したがって計数処理の結果を AEL 上に記録しておけば，この計数処理の実行を大幅に省略できる（実行回数は出力画像のスキャンライン数と無関係になる）．なお，辺の交差の有無はバケットソート法による並べ替えの際に検出できる．

以上の理由により，このアルゴリズムにより図形の論理演算を行う場合の計算量は，論理演算の複雑さにほとんど影響されず，通常の多角形の走査変換と同様，すべての辺の数 n に対してほぼ $O(n)$となる．

2.3.› まとめ›

本章では，本論文で提案する多面体の集合演算表示アルゴリズムを解説するための準備として，走査変換を用いた平面図形同士の論理演算表示アルゴリズムについて述べた．この手法は，隣接するスキャンラインの類似性を利用することで，論理演算を行わない多角形の走査変換に対し，ごくわずかなオーバーヘッドで実行できる．

複雑な線図形のマスク処理は多くの作図処理ソフトウェアや PostScript などのページ記述言語をもつプリンタの制御機構などに実装されているが，本章で述べた方法は，特にライン単位での出力が必要になる大判のインクジェットプリンタ（いわゆるポスタープリンタ）への出力に適した方法であると考える．

第3章　多面体の集合演算表示

前章で述べた走査変換による図形の論理演算表示アルゴリズムを３次元に拡張することにより，多面体の集合演算表示が実現できる．本章では，まず多面体を表示する場合に必要になる隠面消去処理に関して，この走査変換をベースにしたスパニングスキャンライン法もとづく隠面消去アルゴリズムについて述べる．次に，このスパニングスキャンライン法の可視面の判定に奥行き方向の１次元集合演算処理を用いて多面体の集合演算表示を行う方法[12]と, CSG の積和表現にもとづく効率の良い集合演算アルゴリズム[13]について述べる．

3.1.　集合演算表示アルゴリズム

境界表現を内部表現に持つ形状モデリングシステムにおいても，集合演算による形状定義は複雑な形状を定義する手段としてしばしば利用される．

一般にこの方法は，あらかじめ用意された複数の立体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て目的の形状を得る．この後，得られた形状を画面表示などにより確認し，もし形状が意図したものと異なれば，集合演算処理を取り消して以前のデータを回復する．

ここでブール結合しようとする 2 個の立体がそれぞれ m, n 枚の面で構成された多面体であれば，面の干渉検出のために，集合演算処理の計算量は $O(mn)$ となることが予想される．このため，この方法では立体形状が複雑になるにつれて，操作のインタラクティビティが確保できなくなる恐れがある．

しかし，Weiler らがスクリーン上で交差図形を求めて隠面消去処理を行おうとしたこと（2.2.1. 節）からもわかるように，隠面消去処理は図形の（奥行き方向を含んだ）スクリーン空間中での干渉問題であり，集合演算処理と類似した問題だと考えられる．したがって隠面消去処理を拡張することにより，形状データに対して実際に集合演算処理を適用せずに，ブール結合の記述から直接集合演算表示を行う方法が提案されている．

多面体を対象にした場合，これには視線探索法[14], Z バッファを用いる方法[15]，拡張 Z バッファ法[16,17]，Z バッファにステンシルバッファを併用する方法，およびスパニングスキャンライン法にもとづく方法[18]などがある．

これらはいずれも，スクリーンに対して奥行き方向に１次元集合演算処理を

実行して，可視面を判定する．この１次元集合演算処理 1 回あたりの計算量はブール結合の記述の複雑さに依存するが，多面体を構成する面の数には依存しない．したがってこの場合の全体の計算量は，通常の隠面消去処理と同様に，面の数の合計 $m+n$ に対してほぼ $O(m+n)$ とすることができる．

特にスパニングスキャンライン法はサンプルスパン単位に集合演算処理を行うため，１次元集合演算処理の実行回数がスキャンライン上の画素数に依存せず効率が良い．これに対して他の方法はいずれも画素単位に１次元集合演算処理を行うため，画面表示の高解像度化やブール結合の記述の複雑化に伴う処理速度の低下が大きい．

3.2.> スパニングスキャンライン法による多面体の隠面消去処理>

スキャンライン法による隠面消去処理は，視点を通りスクリーン上のスキャンラインを含む平面（スキャンライン平面）によって物体を切断し，スキャンライン平面上で２次元の可視判定を行って，可視面を判定する手法である．

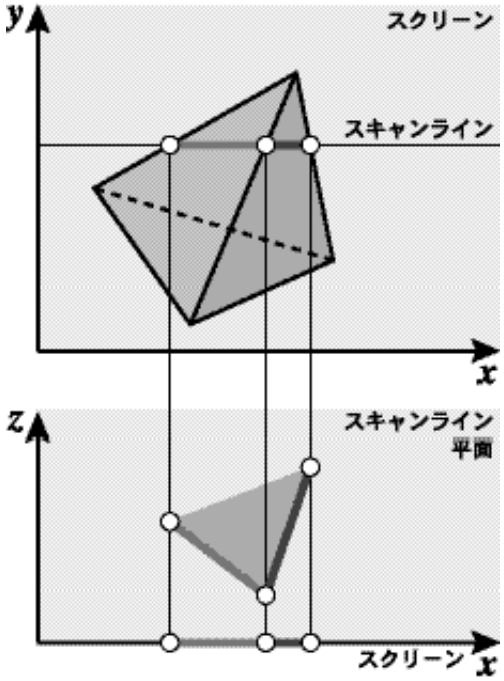


図 3.1 スキャンライン法による隠面消去処理の概念

スキャンライン上での可視面の判定には，奥行きの最小値を持つ面をスキャンライン上の画素単位に求める手法（スキャンライン Z バッファ法）や，スキ

ャンラインの再帰的な分割により効率よく可視面の判定を行う Watkins のアルゴリズム[19]などが知られているが，本研究では前章で述べた多角形の走査変換のアルゴリズムを奥行き方向に拡張し，後で述べる計数による 1 次元集合演算処理に適した隠面消去アルゴリズムを開発した．

前章で述べたアルゴリズムでは，多角形の走査変換において複数の多角形を取り扱うため，辺を AEL に登録する際に，その辺が属している多角形の識別子も記録している．そこで，この多角形同士の論理演算の代わりに多角形の間に優先順位を与えて，その順に手前に表示されるようにすれば，隠面消去処理が実現される．このために，AEL をたどる際に多角形の優先順位を保持する Active Polygon List (APL) を導入する．APL の作成方法を以下に述べる．

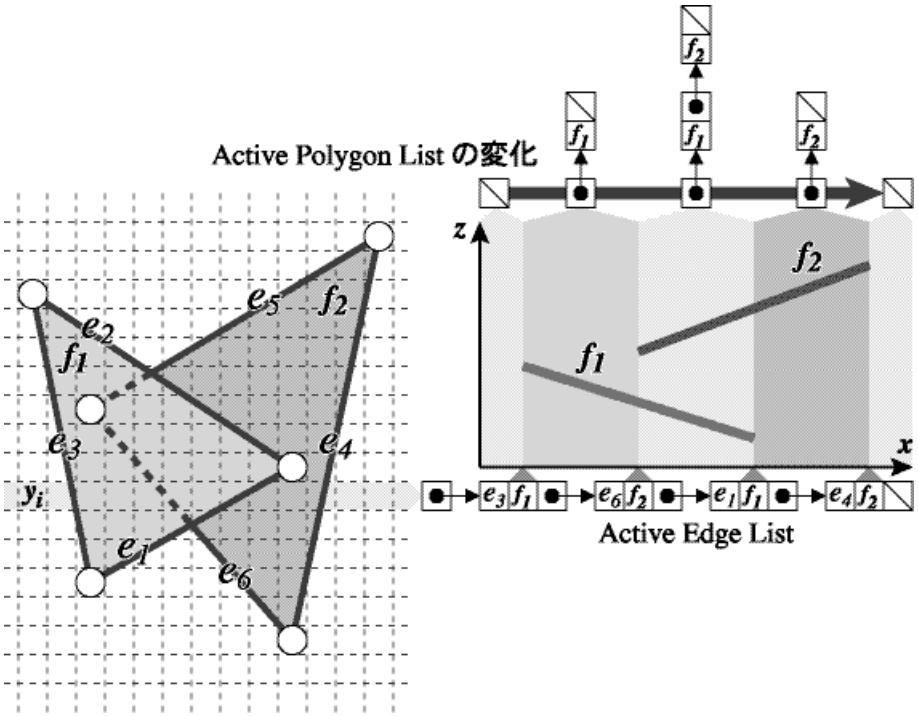


図 3.2 Active Polygon List の作成

(1) APL の初期値を空にする．

(2) 現在のスキャンラインの AEL の先頭にある辺と，その辺と同じ x 座標値を持つ辺を順に取り出し，それらについて以下の処理を行う．

  (a) 取り出した辺が属している多角形が APL に登録されていなければ，その多角形を APL に登録する．

  (b) 取り出した辺が属している多角形が既に APL に登録されていれば，APL からその多角形を削除する．

(3) AEL の残りの辺について (2) から処理を繰り返す．AEL に辺が残っていなければ，このスキャンラインにおける処理を終了する．

ここで多角形が APL に登録されているかどうかは，2.2.3. 節の Step 3 で用いた O_k のように，各多角形に APL に組み込まれたことを示すフラグを用意することによって判定できる．

図 3.2 の例について説明する．図中のスキャンライン y_i において，AEL には $e_3 \rightarrow e_6 \rightarrow e_1 \rightarrow e_4$ の順に辺が登録されている．AEL を先頭から順にたどれば，最初に e_3 が取り出され，これにより APL には f_1 が登録される．次に AEL から e_6 が取り出され，f_2 が APL に追加される．そして e_1 の取り出しにより APL から f_1 が取り除かれ，APL には f_2 のみが残る．

以上の処理により，スキャンライン上において二つの辺に挟まれた範囲内にある多角形が，常に APL に保持される．したがって，この範囲内で可視となる多角形を APL に登録されているもの中から決定すれば，隠面消去処理が実現できる．ここで単に複数の２次元図形に優先順位を付けて重ね合わせて表示するに留めるなら，APL の中でもっとも優先順位の高い多角形を可視面とすればよい．また３次元の多面体を表示する場合でも，多角形同士が交差（相貫）していなければ，この範囲内の任意の点（左端や中央点等）における多角形の奥行きを求めて，最小の奥行きを持つ多角形を可視面とすればよい．

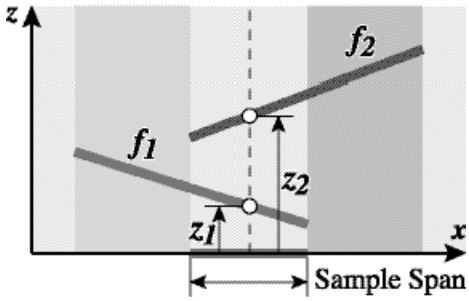


図 3.3 サンプルスパン内の可視判定

このように，スキャンライン上で可視面判定の単位となる領域を，**サンプルスパン**と呼ぶ．また，このサンプルスパン単位に可視面を決定する手法を，**スパニングスキャンライン法**と呼ぶ．Watkins のアルゴリズムもスパニングスキャンライン法の一種である．

なお，サンプルスパンごとに可視判定を行う代わりに，多角形の APL への登録に単純挿入法を用いて APL 上での多角形の順序を保つようにすれば，APL の先頭の多角形を常に可視面とすることができる．

さらに，多角形同士の交差が無い場合には，2.2.3. 節と同様に隣接するスキャンラインの類似性を用いた効率化を図ることができる．すなわち，次のスキャンラインの処理に進む際に以下の条件のいずれにも該当しなければ，直前に処理したスキャンライン上の各サンプルスパンにおける可視面を，次のスキャンラインの対応するサンプルスパンにおける可視面とすることができる．

(a) 現在の AEL から辺が取り除かれたとき

(b) 次のスキャンラインの y-Entry List から辺が AEL に併合されるとき

(c) 辺がスクリーン上で交差しているとき

したがって AEL 上の各辺に，その辺を左端とするサンプルスパンにおける可視面を記録しておけば，APL の作成や，それにもとづくサンプルスパンの可視面の決定処理を大幅に省略できる．

3.3.> スパニングスキャンライン法の多面体の集合演算表示への拡張>

前節のアルゴリズムにおけるサンプルスパン上の可視判定において，面の奥行き比較による可視判定の代わりに，奥行き方向の 1 次元集合演算処理によって可視面を決定すれば，集合演算表示が実現する．

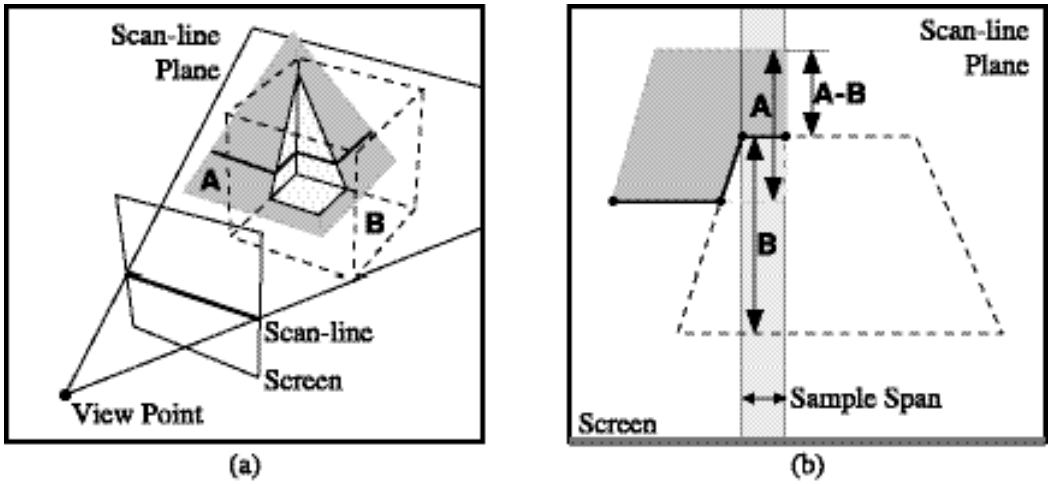


図 3.4 サンプルスパン上の 1 次元集合演算処理

図 3.4 (a) に示す 2 個の立体 A, B の差の形状を表示する場合，同図 (b) に示すスキャンライン平面上のサンプルスパン内の領域において，A-B の部分の前面は B の後面である．したがって，この面を可視面とすれば，集合演算表示が達成される．以降に，その具体的な手続きについて述べる．

3.3.1.> 面の交差処理>

3.2. 節で述べた隠面消去アルゴリズムは，面（多角形）同士の交差について考慮していない．しかし集合演算表示では，面の交差を処理して交差線を表示できることが不可欠である．そのためには，サンプルスパン内における面の交差を検出し，その交点を求める必要がある．本章で述べるアルゴリズムでは，サンプルスパンを中点で再帰的に分割して，交点の算出を行った．

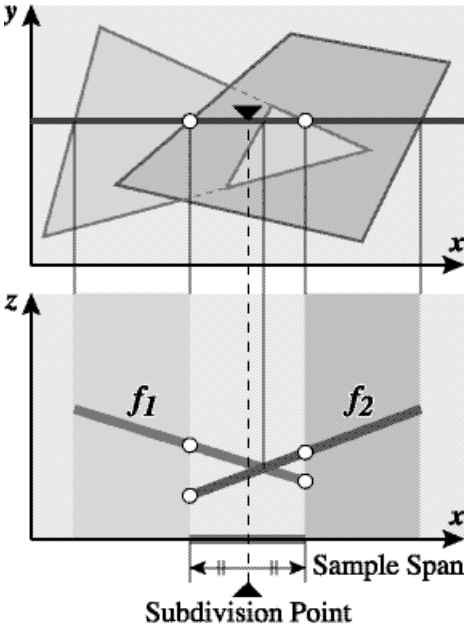


図 3.5 二分法によるサンプルスパンの分割

(1) 現在の APL に登録されている面を, サンプルスパンの左端の位置における奥行きの順に並べておく. これには面を APL に登録する際に単純挿入法を用いればよい.

(2) 現在 APL に登録されている面の, サンプルスパンの右端の位置における奥行きを求める.

  (a) 各面の奥行き値が APL 上の面の順序と一致していれば, そのサンプルスパンにおいて 1 次元集合演算処理を実行し，可視面を決定する.

  (b) 各面の奥行き値が APL 上の面の順序と一致していなければ, そのサンプルスパン内で面が交差している．したがってサンプルスパンの中点を新たなサンプルスパンの右端として，(2) の処理を繰り返す.

(3) サンプルスパンの分割により可視面が決定された場合は, 分割点で APL を並べ替える．AEL と同様の理由で，この並べ替えにもバブルソート法を用

いればよい．その後，そのサンプルスパンの右端を次のサンプルスパンの左端とし，サンプルスパンの右端を元の位置に戻して，右側のサンプルスパンの処理を行う．

3.3.2.> 積和形による CSG の記述>

基本立体（プリミティブ）のブール結合関係を用いて形状を記述する CSG は，一般には図 補遺 6 (P. 80) のような木構造で表現されることが多い．文献 18 においても，CSG の記述に木構造を用い，可視面の判定には視線探索法と同じアプローチを採るものとしている．

特に 2 分木による表現は，既に組み合わせた形状に対して新たなプリミティブを一つずつ追加していくという，実際の作業形態との親和性がよい．しかし，その記述から直接陰影画像を生成する場合には，可視面の判定を行うたびにこの木を探索する必要があり，画像生成時の大きな負担となる．このため，この木を高速に探索する手法[20]が提案されているほか，空間を等分分割して部分空間内の図形を限定する手法[21,22]によっても，この木を探索する手間を削減できる．

これらに対し，筆者らは一般的な隠面消去アルゴリズムが集合演算における和集合演算を処理できる点に着目し，積和形による CSG の記述を採用した [13]．なお Goldfeather らも，本研究の直後に同様のアイデアを発表している[23]．

いま，いくつかの形状要素 E_{ij}（これをエレメントと呼ぶ）の積集合演算によって表される，次のような組み合わせ S_j をセグメントと呼ぶ．

$$S_j = \bigcap_{i=1}^{m} E_{ij} \tag{3.1}$$

物体形状 T はこのセグメントの和集合演算によって定義される．

$$T = \bigcup_{j=1}^{n} S_j \tag{3.2}$$

この T を行列形式で表現したものを，**パターンマトリクス**と呼ぶ[24]．任意の集合演算式は，記号的処理により積和形に変形できる．

一般の形状記述には，セグメント間の差集合演算も含まれる．しかし本研究では和集合演算処理を隠面消去処理により実現するために，エレメントの補集合，すなわち負のエレメントを導入して，差集合演算をセグメントに内包されるエレメント間に限定した．これによりセグメント間のブール結合関係を和集合演算のみとすることができ，(3.1) 式で示されるセグメントの表面判定処理を隠面消去アルゴリズムに追加するだけで集合演算表示が可能になる．

また本研究では，(3.2) 式の形式によって記述された形状データの表現に，エレメントリストとセグメントリストという2本の線形リストを用いた. 図 3.6 (b) は，同図 (a) の形状をこのデータ構造を用いて表現したものである（負のエレメント C は半透明で表示している）. この形状のブール結合記述を積和形で表せば $(A \cup B) - C = (A \cap \overline{C}) \cup (B \cap \overline{C})$ となる.

この形状は 3 個のエレメントと 2 個のセグメントで構成される．セグメントリストは $A \cap \overline{C}$ を表す S_1 と $B \cap \overline{C}$ を表す S_2 の 2 個の要素を保持する．一方，エレメントリストの各要素は，それらが属しているセグメントへのリンクを持つ．このうち C は両方のセグメントに属しているため，S_1 と S_2 の両方へのリンクを持つ．また C は補集合なので (polarity) を示すフラグを負 ($\overline{C}$) としている.

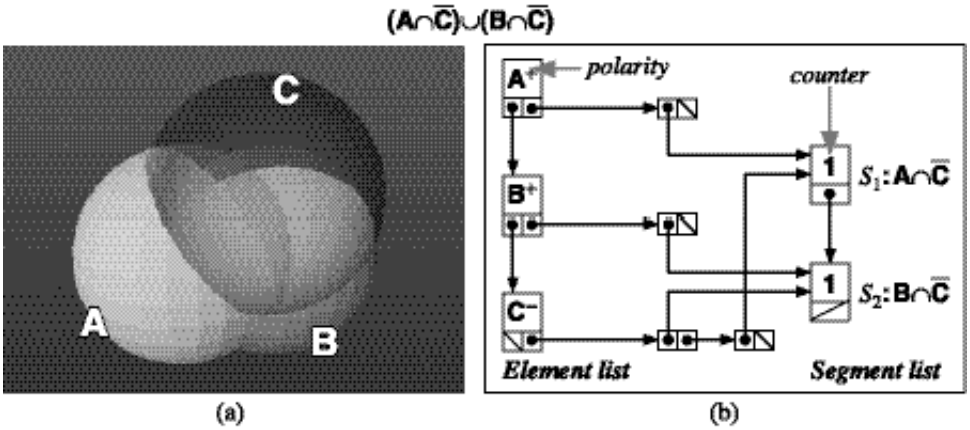


図 3.6 エレメントリストとセグメントリストによる形状の表現

なお，文献 24 ではエレメントとして不等式によって定義される半空間あるいはそれとその存在領域を限定するドメインエレメントの積を用いているが，ここでは対象形状をスキャンライン法による隠面消去処理を効率的に行うことのできる多面体に限定するため，エレメントにも多面体を用いる．よって本論文では，これを**プリミティブ**と呼ぶことにする.

3.3.3.> セグメントの表面判定>

立体形状の内外判定を行う手段の一つに，ある 1 点から任意の方向に半直線を伸ばし，それが立体の境界を通過した回数が奇数回ならその点は立体の内部，偶数回なら立体の外部にあると判断する方法がある．この点が物体の外部にある視点の場合，この半直線が奇数回目に通過した境界は物体の表面，偶数回目に通過した境界は裏面であることがわかる. そこで, この判定に 2.2.3. 節の Step

3 (P. 17) で述べたランの論理演算と同様な手法を用いることを考える．

図 3.7 (a) のように視点が立体の外部にあり，変数 C が 1 で初期化されているとする．そしてこの半直線 (probe) が立体の表面を通過したときに $C \leftarrow C-1$，裏面を通過したときに $C \leftarrow C+1$ というインクリメンタル計算を実行すれば，$C=0$ となった面が物体の表面，$C \neq 0$ となった面が物体の裏面となる．

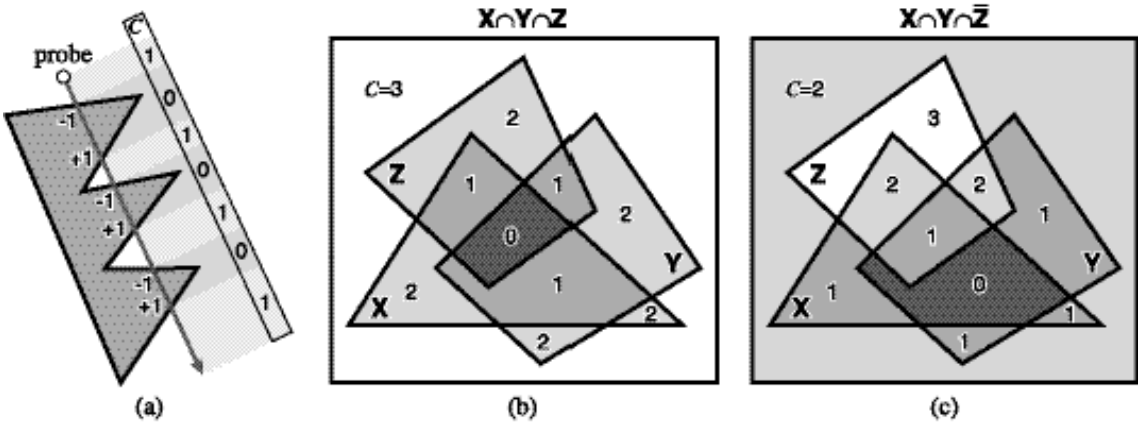


図 3.7 セグメントの表面判定

セグメントはプリミティブの積集合で表されるため，セグメントが表す部分物体は，それを構成するすべてのプリミティブに内包されるという局所性を持つ．したがって，この半直線が外部からこの部分物体の表面に到達するには，それを表すセグメントを構成するすべてのプリミティブの表面を通過しなければならない．したがって，図 3.7 (b) のように 3 個の正のプリミティブで構成されるセグメントが表す部分物体の表面を求めるには，C の初期値を 3 として前述のインクリメンタル計算を行い，$C=0$ となる面を求めればよい．

次に，セグメントが負のプリミティブを含む場合について考える．いま，プリミティブに正負を与えて，正のプリミティブを P_{ij}，負のプリミティブを $\overline{Q}_{ij}$ とすれば，(3.1) 式を次のように書き換えることができる．

$$S_j = \left\{\bigcap_{i=1}^{m'} P_{ij}\right\} \bigcap \left\{\bigcap_{i=1}^{m''} \overline{Q}_{ij}\right\} = \left\{\bigcap_{i=1}^{m'} P_{ij}\right\} \bigcap \overline{\left\{\bigcup_{i=1}^{m''} Q_{ij}\right\}} \qquad (3.3)$$

(3.3) 式はセグメントが正のプリミティブの積集合と負のプリミティブの和集合の積で表されることを示している．すなわち，セグメントが表す部分物体は，それを構成するすべての正のプリミティブの内部で，かつ，いずれの負のプリミティブにも属さない領域にある．

したがって，この部分物体の内部に外部から到達するには，それを構成する正のプリミティブの表面を，その数に等しい回数だけ通過しなければならず，

またその場合に限る．すなわち，負のプリミティブの数はセグメントが表す部分物体の表面の判定に関与しない．しかし，この部分物体が負のプリミティブの外部に存在することには留意する必要がある．半直線が負のプリミティブの表面を通過した場合は，そのプリミティブからの脱出に相当する．したがって負のプリミティブでは，その表面において $C \leftarrow C+1$，裏面において $C \leftarrow C-1$ というインクリメンタル計算を実行する．これはプリミティブを構成する面の表裏を反転して取り扱うことになる．図 3.7 (c) に示すセグメントは 2 個の正のプリミティブと 1 個の負のプリミティブで構成される．したがって C の初期値は 2 となるが，負のプリミティブ **Z** の領域では $C = 3$ となる．

3.3.4.> 計数による 1 次元集合演算処理>

CSG モデルに対して隠面消去処理を行うには, (1) 画素やサンプルスパンなどの可視判定領域内において奥行き方向に 1 次元集合演算処理を行い, (2) 得られた可視面の中から最も視点に近いものを選べばよい．この (2) の処理は通常の隠面消去処理であり, (1) はセグメントが表す部分物体の表面を求める処理である．

前節で述べた特性を利用すれば, 2.2.3. 節で述べた計数によるランの論理演算と同様に，簡単な計数処理によって奥行き方向の 1 次元集合演算が実現する．サンプルスパン内には端点や交点が存在しないので，この処理は安定して行うことができる．以下にその手順を述べる．

(1) 一つのセグメントに対して一つの変数 *counter* を用意し，そのセグメントに属している正のプリミティブの数で初期化する．

(2) 多面体プリミティブを構成する各面 f_k のそれぞれに対して変数 O_k を用意し，面 f_k が視点に対して正面を向いていれば $O_k \leftarrow -1$, 背面を向いれば $O_k \leftarrow 1$ とする．面の表裏はスクリーン座標系におけるその面の法線ベクトルの z 座標値の符号によって決定する．負のプリミティブではこの符号を反転して，前方面と後方面の関係を逆転しておく．

(3) APL の先頭から面を順に取り出し，取り出した面 f_k に割り当てられた変数 O_k を用いて，$counter \leftarrow counter + O_k$ として *counter* に積算していく．

(4) この結果，$counter = 0$ となれば，その時の面が可視面となる．また APL の最後に達すれば，サンプルスパン内に可視面は存在しない．

(5) APL 上に同一の z 座標値を持つ面が複数存在する場合は，そのすべてについて (3) の計数処理を行った後に可視面を決定する．

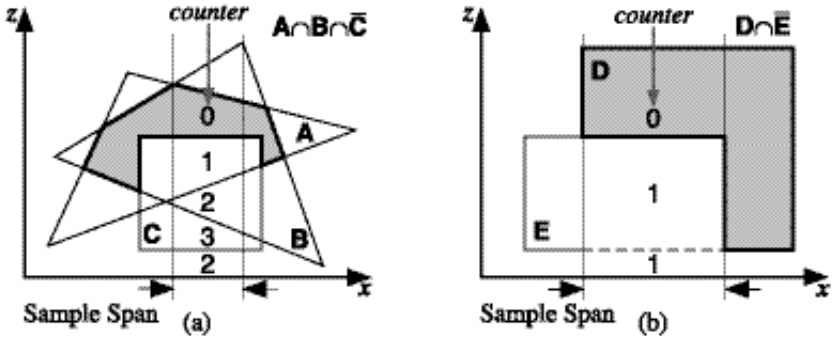


図 3.8 計数による 1 次元集合演算

図 3.8 にこの処理による可視面決定の過程を示す．同図 (a) に示すセグメントでは *counter* の初期値は 2 であり，サンプルスパン内では APL の先頭に *C* の前面が登録されている．この面は表を向いているが *C* の極性は負のため，APL からのこの面を取り出した際に *counter* は 1 増され 3 となる．以降 *B* および *A* の前面が APL から順次取り出されるが，これらの極性はいずれも正なので *counter* は 1 ずつ減じられる．そして負の極性をもつ *C* の後面を取り出したときに *counter* = 0 となり，*C* の後面がこのサンプルスパンにおける可視面となる．

表 3.1 に上記の計数処理による集合演算処理によって実現される集合演算の規則を示す．この表の C の部分は，境界において前方面同士，あるいは後方面同士が接していれば内部，前方面と後方面が接していれば外部であることを示している（図 3.8 (b) の破線部）．この処理は (5) の手順によって実現される．

表 3.1 集合演算の規則

∩		+		
		I	B	O
+	I	I	I	O
	B	I	C	O
	O	O	O	O

∩		+		
		I	B	O
−	I	O	O	O
	B	I	C	O
	O	I	I	O

∩		−		
		I	B	O
−	I	O	O	O
	B	O	C	I
	O	O	I	I

+:Positive Primitive
−:Negative Primitive
I:In
B:Boundary
O:Out
C:If both bondaries
  are same side;
 then
  C⇒In
 else
  C⇒Out

この方法は，あるセグメントリストを参照している（そのセグメントリストに属している）プリミティブのうち，アクティブなもの（サンプルスパン内にあるもの）の数を数えることによって，可視面を決定している．このため，本論文ではこれを**リファレンスカウント**と呼ぶ[13].

この方法は隠面消去処理のために作成された情報 (APL) をもとに，単純なインクリメンタル計算のみによって集合演算処理の結果の可視面の判定を行うため，隠面消去処理に対する集合演算処理のオーバーヘッドが少ない．

3.4.> 効率化>

3.4.1.> クリッピングによるセグメントの存在領域の限定>

本章の手法では視点の上方や背後にある図形を処理できないため，これを表示領域のクリッピングに代えることはできない．そのため本手法でも，表示領域からはみ出る図形を，通常の隠面消去処理と同様にあらかじめクリッピングしている．これには Sutherland-Hodgman のアルゴリズムを用いている．

通常このクリッピングは，図 3.9 に示す視点を頂点とする四角錐を前方面と後方面で切り取った視野錐台 (View Frustum) に対して行われる．これはクリッピングボリュームとも呼ばれる．

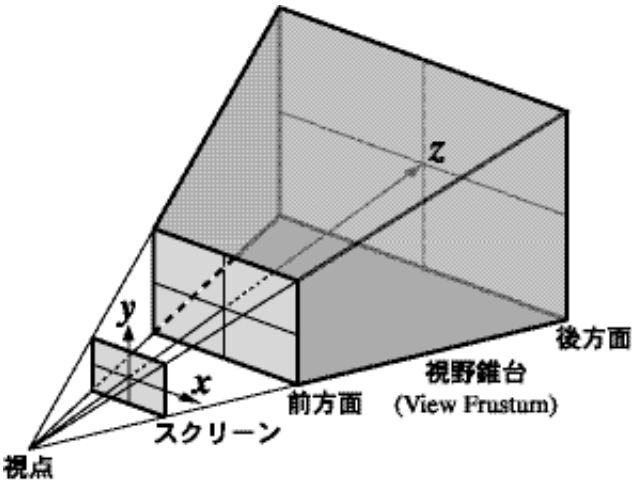


図 3.9 視野錐台

一方, 計数による 1 次元集合演算処理では, APL を最後までたどらなければ, その領域が集合演算によってすべての面が取り除かれた「空」の領域であることを検出できない．このため，そのような領域では図形が存在する領域に比べて 1 次元集合演算処理の完了までに時間を要する．

そこで，セグメントが表す部分物体の外接箱を求めて（図 3.10），表示領域によるクリッピングに用いる視野錐台の代わりに用いれば，集合演算によって空となる面の大半を事前に処理対象から除外できる．これは表示領域によるクリッピングを兼ねており，視点が物体の外部にあることを保証する．

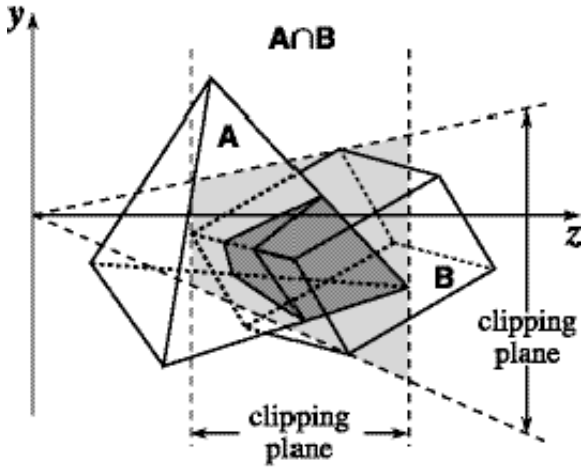


図 3.10 セグメントが表す部分物体の外接箱

3.3.3. 節で述べたように，セグメントが表す部分物体は，そのセグメントに属する全てのプリミティブに内包される．共通集合はその構成要素であるいずれの集合，あるいはその任意の部分集合の共通集合によっても被覆されるから，セグメントが表す部分物体の外接箱は，それに属する正のプリミティブを包含する外接箱の共通部分として求めることができる．以下にその手順を示す．

(1) 個々の正のプリミティブのすべての頂点について，それらの点を含み y 軸を通る平面群の xz 平面に対する傾きの範囲, それらの点を含み x 軸を通る平面群の yz 平面に対する傾きの範囲, およびそれらの点の z 値の範囲を求める．

(2) セグメントが表す部分物体の外接箱は，そのセグメントに属している正のプリミティブの，これらの範囲の論理積として求める．

(3) 個々のプリミティブの外接箱に，そのプリミティブが属しているセグメントが表す部分物体の外接箱を設定する．プリミティブが複数のセグメントに属しているときは，それらの範囲の和を外接箱に設定する．プリミティブがどのセグメントにも属していない（他のプリミティブと集合演算関係を持たない）ときは, もとのクリッピングボリュームを外接箱に設定する．

この後プリミティブごとに，この外接箱を用いて面のクリッピングを行う．

なお，リファレンスカウントを用いた１次元集合演算処理では，可視面の判定を視点から可視面に至るまでの間に存在する面に対する計数処理によって行うため，奥行き方向のクリッピングによって途中の面が削り取られてしまうと，正常な集合演算処理を行うことができない．これを防ぐには，奥行き方向のクリッピングの際に，切断面に「蓋」となる面を生成する必要がある．これは多面体の面と前方面との交差線を，多面体ごとにまとめて一つの面として登録するだけでよい．直後に隠面消去処理が行われるため，この面に対して稜線の連結等を考慮する必要はなく，穴も問題なく処理される．奥行方向のクリッピングによって蓋が付けられた様子を図 3.16 (d) に示す．

3.4.2.> 物体形状の一様性の利用>

3.3.1. 節で述べたアルゴリズムでは，可視面の背後にある面の交差によってもサンプルスパンが分割されてしまう．これを避けるために，物体形状の一様性のうち，サンプルスパン類似性を利用する．

直前（左）のサンプルスパンにおける可視面を候補面としたとき，それに続くサンプルスパンにおいて候補面が不可視となる可能性があるのは，以下の条件のうちいずれかが成立した場合である．

(a1) APL において候補面より前方に別の面が挿入されたとき．

(a2) 候補面が別の面と交差しているとき．

(a3) 候補面が APL から取り除かれたとき．

可視面が集合演算処理によって決定される場合は，これらに加えて次の条件も考慮する必要がある．

(a4) 候補面より前方に面同士の交差が存在するとき．

したがって，これらの条件がいずれも成立していなければ，候補面をそのサンプルスパンにおける可視面とし，集合演算処理を省略できる．

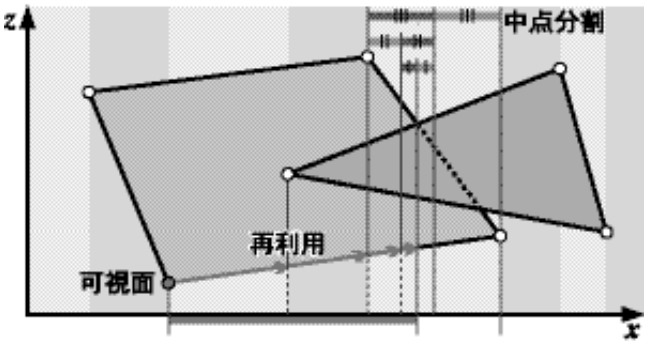
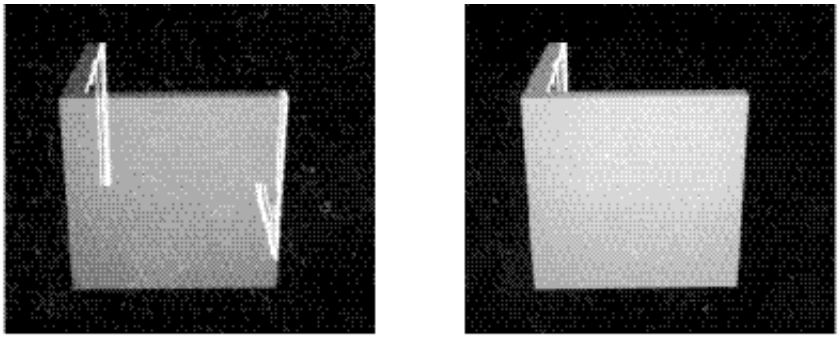


図 3.11 サンプルスパンの類似性の利用

これによって，可視面の背後にある面の交差によるサンプルスパンの分割が防止できるとともに，面の交差の近傍のサンプルスパンの分割が繰り返される部分の集合演算処理も削減される．図 3.12 (a) および (b) において，輝点はサンプルスパンの分割点を示す．同図 (b) では可視面の背後にある交差によるサンプルスパンの無用な分割が回避されている．



(a) サンプルスパンの類似性を利用しない場合　　(b) サンプルスパンの類似性を利用した場合

図 3.12 サンプルスパンの分割位置

物体形状の一様性のうち，スキャンライン類似性も集合演算処理の削減に利用できる．面の交差を考慮しない，すなわち集合演算処理を行わない場合，あるスキャンラインから次のスキャンラインに処理が移る際に，AEL に対して以下の条件がすべて成り立てば，この 2 本のスキャンラインの対応するサンプルスパンにおける可視面は同一である．

(b1)　削除される稜線がない．

(b2)　稜線の順序が変化しない．

(b3)　次のスキャンラインに AEL に結合する稜線が存在しない．

しかし面の交差を処理する場合は，さらに次の条件も考慮する必要がある．

(b4)　処理しようとするサンプルスパンの両端で面の順序が変化しない．

したがって，スキャンライン上の個々のサンプルスパンにおける可視面を AEL に記憶しておけば，これらの条件のもとで，隣り合うスキャンラインにおいて対応するサンプルスパンにおける集合演算処理を省略できる．

この二つの処理を組み合わせることによって，1 回の集合演算処理の結果をスクリーン上の 2 次元領域で共有することができる．すなわち，これらの効率化手法を用いることによって，集合演算処理の実行回数の，スクリーンの表示画素数に対する依存度を下げることができる．

3.5.> 実験>

本章で述べたアルゴリズムに対して，評価実験を行った．実験には Sun 社製 SPARCstation 330 を使用し，画像表示には NEXUS 600（表示画素数 512×480）を用いた．

3.5.1.> 提案アルゴリズムにおける面数と処理時間の関係>

図 3.14 は図 3.13 に示すような，二つの多角柱プリミティブによって定義された形状に対して，多角柱の側面の数を 3〜100（面の数 5〜102）と変化させた場合の陰影画像生成に要する時間の変化を示している．

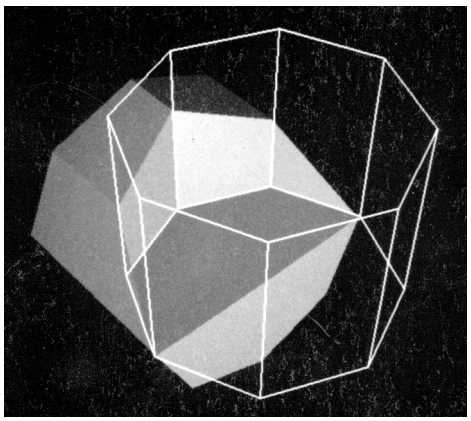


図 3.13 多面体の差集合演算のモデル図

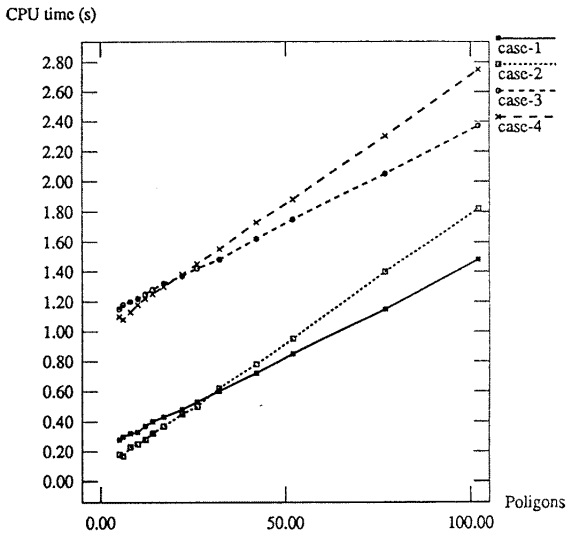


図 3.14 二つの多面体の差集合演算処理における面数と処理時間の関係

図 3.14 において，case-1 は二つのプリミティブの間に差集合演算を定義した場合を示し，case-2 はこれを定義しなかった場合（和集合演算のみ）を示す．case-3

およびcase-4はcase-1およびcase-2に画像のディスプレイ表示に要する時間(グーローシェーディングのためのサンプルスパン内の色の補間と背景処理および表示装置への転送時間）を加えた全体的な処理時間の変化を示す．

このように n 個の面からなる二つの多面体の集合演算を処理する場合，面の干渉処理の計算量は一般的に $O(n^2)$ となるが，本論文のアルゴリズムでは，これがほぼ $O(n)$ で処理されていることが示されている．またこの図で case-1 と case-2 の処理時間が途中で逆転しているのは，case-1 では可視面が差集合演算処理によって削られているために，隠面消去処理に要する時間が case-2 に比べて少なくなったためだと考えられる．このことから，本論文のアルゴリズムにおいて集合演算処理に費やされる時間が，全体の処理時間に対して十分小さいことがわかる．

3.5.2.> 提案アルゴリズムにおけるスキャンライン数と処理時間の関係>

図 3.15 は図 3.16 (a), (b), (c) のような形状（いずれも面の数 772）に対して，表示解像度を 128×128～4096×4096 と変化させた場合の，可視面決定に要する時間の変化を示したものである（画像のディスプレイ表示に要する時間，およびモデリング変換，視野変換，透視変換，陰影計算に要する時間を除く．なお，データ量が固定なので，後者の時間は一定である）．

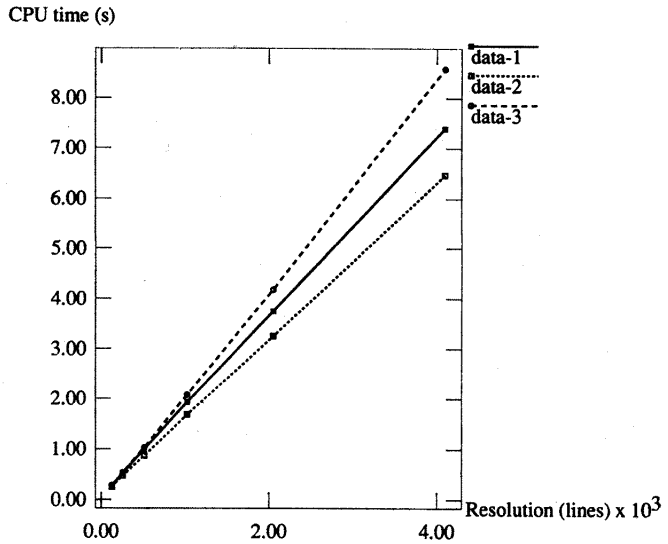


図 3.15 スクリーンのスキャンライン数と処理時間の関係

ここで四面体を A，球を B，立方体を C としたとき，data-1 には $A \cup B \cup C$（セグメント数 = 0），data-2 には $A \cup B \cap C$（セグメント数 = 1），data-3 には

$A \cap C \cup \overline{B} \cap C$（セグメント数 $=2$）という集合演算関係を与えている．

視線探索法のように画素単位に集合演算処理を行った場合，処理時間は表示画素数，すなわち（スキャンライン数）×（スキャンライン上の画素数）に比例するが，本論文のアルゴリズムではほぼスキャンライン数に比例している．一方，セグメント数に対する処理時間の変化をみると，data-1 に対して data-2 の処理時間が減少している．これは図 3.14 の場合と同じく，集合演算によって可視面が減少したためである．

これに対し，data-2 の処理時間の増加に対する data-3 の処理時間の増加が大きいことは，集合演算処理の発生回数がスキャンライン数に依存していることを示している．これより，このデータではスキャンライン類似性を利用した集合演算処理の省略の効果が十分あらわれていないことがわかる．この理由は，このデータが可視面の前方に不可視面の交差を多く含んでいるために，サンプルスパンの分割による集合演算処理の発生が避けられなかったためである．

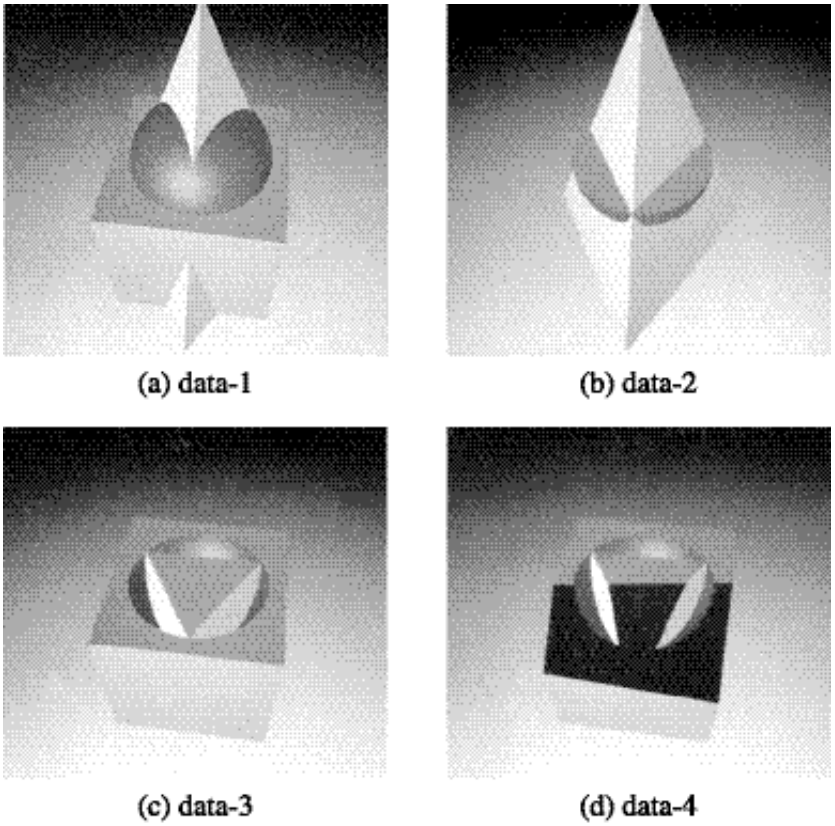


図 3.16 サンプル形状

3.6. まとめ

スパニングスキャンライン法では，各種の並べ替えアルゴリズムをうまく組み合わせることによって，隠面消去問題をほぼ $O(n)$ の計算量で処理することができる．本論文で示したアルゴリズムは，スキャンライン法のこのような性質を，集合演算処理においても積極的に活用することで，集合演算処理を盛り込んだ隠面消去処理を，隠面消去処理のみを行う場合と同じオーダーの計算量で実行できる．

計算機の内部表現として境界表現を用いている場合は，形状変形操作の結果が直接形状データ上に反映されるため，試行錯誤により直前の操作を取り消して，もとの形状を復元することが難しい．そこでこの取り消し操作を実現するために，基本変形操作に対して逆操作を用意し，基本変形操作の履歴を保存することによって，もとの形状を復元する方法[25]などが提案されている．

形状定義におけるユーザーインターフェースの部分に CSG を採用している場合は，一つの集合演算処理が複数の基本変形操作によって実現される．この場合も対応する一連の逆操作を実行することによって，形状を復元することが可能である．しかし本研究で示したアルゴリズムは，形状データに変形操作を加える前に形状を確認するための，より簡便な方法を提供する．これは変形操作を加えた後に陰影画像を生成するよりも高速に行われなければ意味がないが，本論文のアルゴリズムはこれに十分な速度を達成している．

第4章 WEDで表現された形状の集合演算表示

一般的な多面体の隠面消去アルゴリズムは，互いに独立した面を貼りあわせて作った多面体を対象にしている．すなわち，一つの辺は一つの面にのみ属しており，他の面と共有されることはない．したがって，この方法では閉じた多面体であっても，1本の稜線を2本の辺として取り扱う必要がある．

一般の形状モデリングシステムの中には1本の稜線を2本の辺（ハーフエッジ）で表現するものも存在する．しかし，境界表現による形状データの表現方法としてもっとも一般的なWinged Edge Data (WED) 構造は，1本の稜線を2枚の面で共有するデータ構造になっているため，WEDで表現された形状の画面表示を行う場合には，一旦互いに独立した面データを生成し，それに対して隠面消去処理を実行する必要がある．

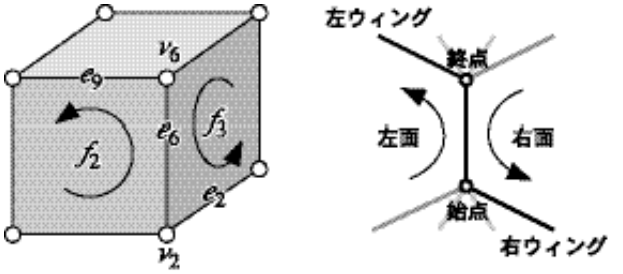
前章で述べた手法をWEDで表されたプリミティブに適用する場合も，これと同様に互いに独立した面データを生成する必要がある．さらに，その際にはWEDから面データへの変換の手間に加えて，もとの稜線数の2倍の辺が生成されることになる．その結果，同じ始点と終点を持つ2本の辺に対して，同じ増分計算やスキャンライン上の処理が行われてしまう．

しかし，WEDには面や稜線の接続情報も保持しているため，それらを利用すれば前述のような無駄が省けるほか，隠面消去処理の効率化を図ることも可能である[26]．

本章では，1次元集合演算処理による可視面判定をWED構造が持つ稜線と面の接続情報を利用して効率化することによって，WEDで表現された多面体をプリミティブに用いたCSGモデルから高速に陰影画像を生成することが可能な直接集合演算表示アルゴリズム[27,28,29]について述べる．

4.1 Winged Edge Data 構造

Winged Edge Data (WED) 構造は，境界表現において，特に稜線に着目して他の稜線や面との接続関係を記述するデータ構造である．たとえば図4.1の稜線e_6は，それに接続する6個のデータ$v_2, v_6, e_9, e_2, f_2, f_3$への指標で表される．したがってこのデータから，稜線e_6が2枚の面f_2, f_3で共有されているという情報が得られる．



	始点	終点	左ウィング	右ウィング	左面	右面
e_6→	v_2	v_6	e_9	e_2	f_2	f_3

図 4.1 Winged Edge Data 構造

4.2.> WED で記述された多面体の集合演算表示>

この WED を直接処理するためには，1 本の稜線によって 2 枚の面を処理する必要がある．そこで本研究では，稜線による面の APL への登録および削除のパターンを，図 4.2 の (a) 2 枚の面がともに登録される，(b) 一方の面をもう一方の面で置き換える，(c) 2 枚の面がともに削除される，の三つに分類した．

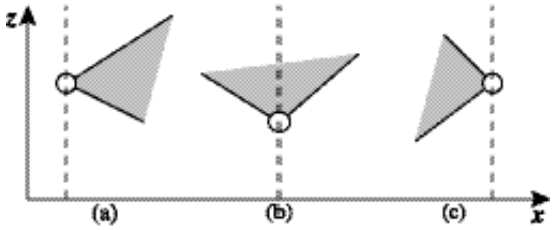


図 4.2 稜線による面の APL への登録・削除パターン

これにしたがって，3.2. 節で示した Active Polygon List (APL) の作成方法と面の交差の処理方法を，以下のように変更する．

4.2.1.> Active>Polygon>List>(APL)> の作成>

(1) AEL の先頭の稜線の x 座標値を，サンプルスパンの左端の位置の初期値とする．

(2) 現在のサンプルスパンの左端の位置にある稜線を， AEL の先頭から順次取り出す．

(3) 取り出した稜線を共有する 2 枚の面が APL に登録されているか否かを調べ，以下のいずれかの処理を行う（図 4.3）．

(a) e_0 のように，取り出した稜線を共有する 2 枚の面 (f_0, f_2) がいずれも APL に登録されていない場合は，両方を APL に登録する．APL への登録

には z 座標値をキーにした単純挿入法を用い，視点から見た面の順序を維持する．

(b) e_1 のように，2 枚の面のうち一方だけが APL に登録されている場合は，APL に登録されている面（f_0）をもう一方の面（f_1）で置き換える．

(c) e_2 のように，取り出した稜線を共有する 2 枚の面（f_1, f_2）がともに APL に登録されている場合は，両方を APL から削除する．

(4) AEL に残された稜線の先頭の x 座標値をサンプルスパンの右端とする．

(5) APL が空でなければ，4.2.2. 節の面の交差処理処理を実行する．

(6) サンプルスパンの右端を次のサンプルスパンの左端として，AEL が空になるまで (2) から処理を繰り返す．

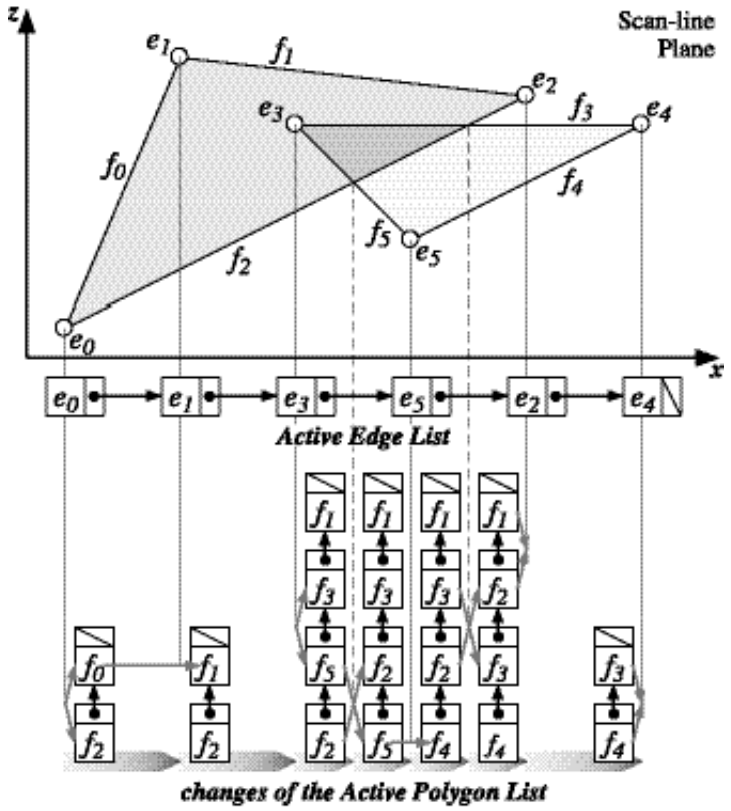


図 4.3 Active Polygon List の作成

4.2.2.> 面の交差処理>

前章のアルゴリズムでは，サンプルスパン内の面の交差がなくなるまで再帰的にサンプルスパンを分割し，その後に 1 次元集合演算処理を実行して可視面を判定していた．しかし，これには以下のような問題があった．

(a) 一つの交点を求めるためにサンプルスパンを何度も分割する必要がある．

(b) サンプルスパンを分割するたびに, 奥行き値の計算をやり直す必要がある.

(c) サンプルスパンを分割して可視面を決定した後，APL に登録されているすべての面に対して並べ替えを行う必要がある.

サンプルスパン内での面の交差位置を求めるのに中点分割法を用いているのは，面の交差角が小さいときに，誤差により交点位置が大きくずれることを避けるためだが，この方法にも次のような問題があった.

(d) サンプルスパンの長さが, その内部で交差している 2 枚の面のなす角度に対して相対的に短いときに，可視面の判定を誤ることがある.

特に (d) の問題は，サンプルスパンの類似性を利用して効率化を行った場合（4.3.1. 節）に，生成画像を崩してしまう原因になる．以下にその理由を示す.

図 4.4 においてサンプルスパンの幅は 1 画素とする．このサンプルスパンの左端は面 f_1 および f_2 の交点として決定されたものである. ここで f_1 と f_2 のなす角が小さい場合，これらのサンプルスパンの中央での奥行きが丸めによって一致してしまうことがある．するとこのサンプルスパン内では 2 枚の面が接していることになり，可視面を一意に決定できなくなる.

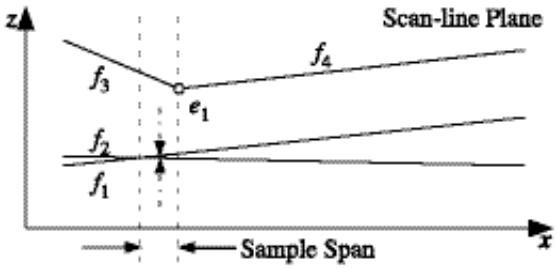


図 4.4 可視面の判定を誤る場合

サンプルスパンの類似性を利用しないなら，ここで可視面として f_1 を選んでも交点の位置が 1 画素ずれるだけで, 次のサンプルスパンにおいて f_1 と f_2 の間隔が充分に開いていれば正しい可視面が表示される.

一方，サンプルスパンの類似性を利用した場合は，この f_1 を次のサンプルスパンにおける可視面の候補として利用する．ところが次のサンプルスパンでは節に挙げる候補面を無効にする条件が成立しないため，この f_1 をそのまま可視面としてしまう.

そこで本研究では，交点における奥行き計算の繰り返しを避け，かつ確実に可視面を判定できるように，バブルソート法に似た次のような手段を用いた.

(1) APL の各面のサンプルスパンの右端における奥行きを求める（図 4.5 (1)）.

(2) APL 上で隣接する 2 枚の面のサンプルスパンの右端における奥行きが，それらの順序と一致しているかどうかを調べる. 現在の APL には面がサンプルスパンの左端における奥行きの順序で並んでいるため，もし順序が一致していなければ，その面の対の少なくとも一方がサンプルスパン内で他の面と交差している.

(3) 交差している面の対があれば，それらの交点を求め，サンプルスパンの右端をサンプルスパンの左端に最も近い交点の位置に変更する．同時にその交点をもつ面の対を記録する（図 4.5 (2)）.

(4) このサンプルスパン内には面の交差がないので，変更後のサンプルスパンの長さが 0 でなければ現在の APL をもとに 4.2.3. 節の集合演算処理を実行して可視面を決定する（図 4.5 (3)）．サンプルスパンの長さが 0 なら何もしない.

(5) 記録した面の対がなければ，この処理を終わる.

(6) 記録した面の対の APL 上での順序を入れ替える. この結果 APL の各面は，現在のサンプルスパンの右端の位置における奥行きの順序に並べ替えられる.

(7) サンプルスパンの左端を，現在の右端の位置に変更する．右端は最初の位置に戻す.

(8) (2) から繰り返す.

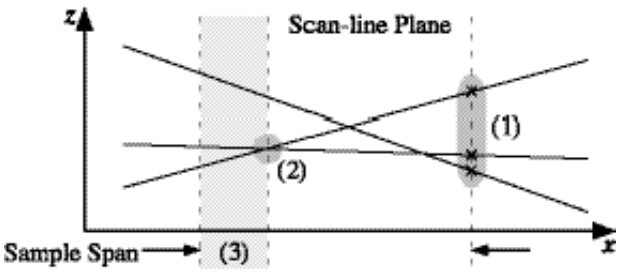


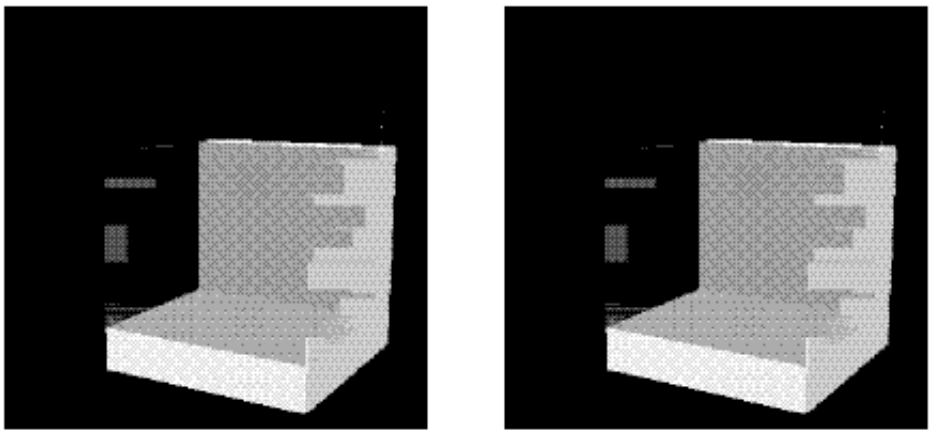
図 4.5 サンプルスパンの分割

4.2.3.> 集合演算処理>

これまでに述べた手続きによって，サンプルスパンと重なる面が APL に奥行きの順に保持され，そのサンプルスパン内には面の交差がないことが保証される．集合演算表示を行うには，この APL に保持されている面をもとに 1 次元集合演算処理を実行して，このサンプルスパンにおける可視面を決定すればよい. これには 3.3.3. 節で述べた計数による 1 次元集合演算処理を用いる.

なお，3.3.3. 節の手順 (5) において，すべてのセグメントについて検査を行わずに $counter = 0$ となったときに直ちに処理を打ち切れば，このループを非常に単純化できる．その場合，図 3.8 (b) の破線部のように，面が接している部分での可視面の判定が不正確になるが，このような部分は処理を打ち切らなくても，サンプルスパン両端における奥行き値の丸め誤差の影響により“くし状”に表示されてしまうことが多い．

図 4.6 に同じ大きさの立方体を一部の面が重なるように配置し，一方からもう一方を引いた形状について，判定の打ち切りの有無による生成画像の変化を示す．この例では面が重なる部分において二つの画像に差異は認められない．



(a) 判定を打ち切った場合　　(b) 判定を打ち切らなかった場合

図 4.6 可視面判定の打ち切りによる生成画像の変化

本論文のアルゴリズムを会話的な形状モデリングに応用した場合は，生成画像を見ながらこのような状況を避けて物体を配置することが可能であり，この現象は実用上問題とならないと考える．

4.3.> 効率化>

計数による１次元集合演算処理の効率は高いが，それでも隠面消去のための可視面判定と比較すれば重い処理となる．これまでに述べた集合演算表示アルゴリズムは，サンプルスパン単位に１次元集合演算処理を実行して可視面を決定するため，その実行回数を削減すればさらに処理時間を短縮できる．

4.3.1.> サンプルスパン類似性の利用>

あるサンプルスパンにおける可視面は，次のサンプルスパンにおいても可視

となる可能性が高い．そこで，サンプルスパンの可視面を決定する際，直前のサンプルスパンにおける可視面を候補面とし，それが無効になった場合のみ集合演算処理を実行して可視面を決定する．

候補面が無効になるのは次の場合である．

(a) 候補面より前方に面が挿入された場合．

(b) 候補面が APL より削除された場合．

(c) 交差によりサンプルスパンが分割された場合．

ただし，新たに APL に登録された面が，4.2.1. 節の Active Polygon List の作成の処理の手順 (3) の (b) により既に登録されている面と入れ替えられたものの場合は，それが候補面より前方にあっても候補面は無効とはしない．また候補面自体が入れ替えられた場合は，新たに登録された面を候補面とする．

この処理の結果，実際に集合演算処理が実行されるのはプリミティブのシルエットになる稜線の位置と面の交差の位置のみとなり，集合演算処理の実行回数がプリミティブを構成する面の数に依存しなくなる．

4.3.2.> 候補面の背後にある面の交差の無視>

4.2.2. 節の面の交差処理において, サンプルスパン内に面の交差が存在すれば, 4.2.2. 節の手順 (3) によってサンプルスパンが分割され, 交点の右側のサンプルスパンに対して集合演算処理が実行される．しかし，この交差が 4.3.1. 節のサンプルスパン類似性の利用において決定した候補面の背後にあれば，これは候補面に影響を与えない．

面の交差が空間中に一様に分布していれば，任意の面の前後にある交差の数の平均値は等しくなる．さらに透視投影の場合は，面の後方の空間の方が前方の空間より大きい．したがって，候補面の背後にある面の交差に対して，サンプルスパンの分割を行わずに APL の並べ替えのみを実行すれば，サンプルスパンの分割に伴う集合演算処理の実行回数を半分以下に削減できる．

4.4.> 実験>

実験に使用したコンピュータは SGI 社製 Indy R5000（R5000PC 150MHz，メモリー 96MB，XZ グラフィックス，IRIX 6.2）である．処理時間の計測にはプロファイラを用いたので，プログラムのコンパイル時に最適化は指定していない．また計測時間には画像表示（サンプルスパンの色のディスプレイ上への転

送）に要した時間は含んでいない．

4.4.1.> 提案アルゴリズムの処理時間>

図 4.7 に示す形状を用いて，提案アルゴリズム（すべての効率化技法を用いたもの）の処理時間を計測した．

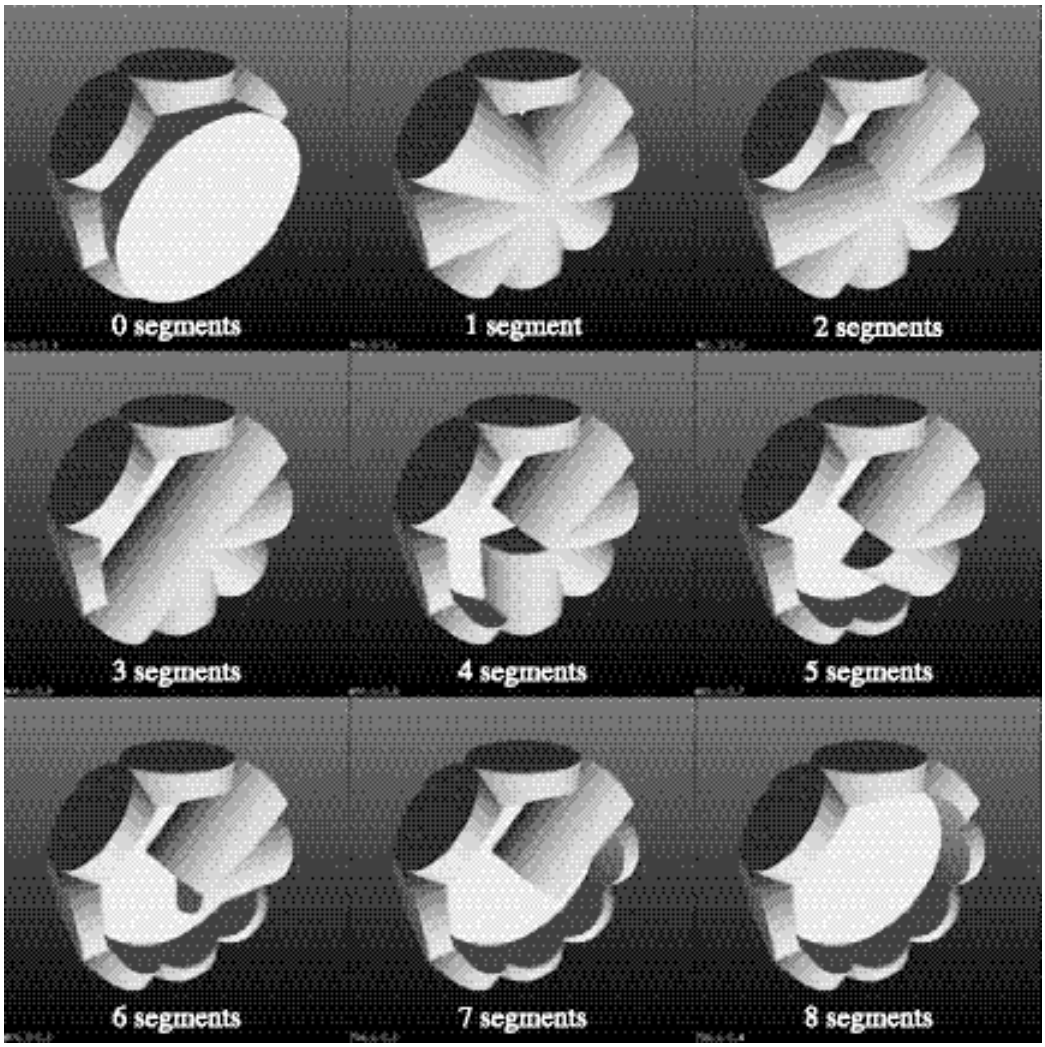


図 4.7 9 個の多角柱からなる形状

この形状はいずれも同じ側面数をもつ 9 個の多角柱からなり，それぞれ全多角柱の和，および 1 個ないし 8 個の多角柱の和から中央部の多角柱を引いたものと残りの多角柱との和である．この形状では，一つの差集合演算が 1 個のセグメントで表現される．

それぞれの形状について多角柱の側面数を 10～100（全面数 108～918）に変化させて 512×512 画素の画像を生成した．図 4.8 には，このうち全多角柱の和

（セグメント数 = 0），8 個中 4 個の多角柱の和から 1 個の多角柱を引いたもの（セグメント数 = 4），8 個の多角柱の和から 1 個の多角柱を引いたもの（セグメント数 = 8）の三つの場合について処理時間を示す．

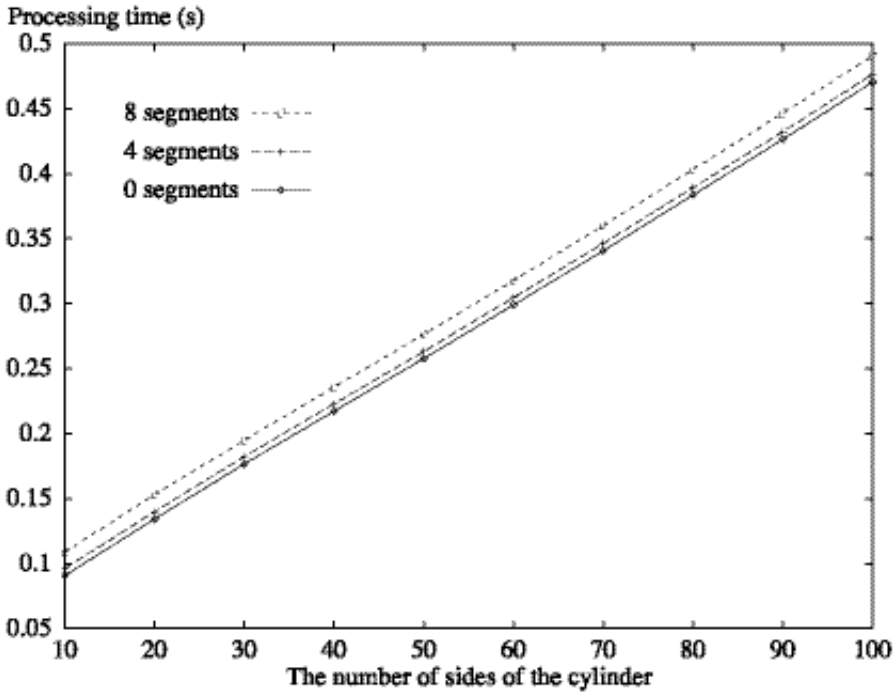


図 4.8 多角形の側面数と全処理時間の関係

一般にそれぞれ n 個の面で構成される 2 個の多面体に対して集合演算処理を実行する場合，面の干渉処理の計算量は $O(n^2)$ となる．しかし提案アルゴリズムでは，これが $O(n)$ で処理されている．

一方，セグメント数の増加，すなわちブール結合の記述の複雑化に伴う処理時間の増分は，多角柱の側面数によらずほぼ一定である．図 4.9 に，この実験において 1 次元集合演算処理に要した時間の合計のみを示す．この図は 1 次元集合演算処理に要する時間が，セグメント数－すなわちブール結合記述の複雑さには依存するが，多面体の側面数－すなわちデータ量には依存しないことを示している．これは，提案アルゴリズムにおける 1 次元集合演算処理の実行回数が，対象形状を構成する面数に対して，ほぼ一定であることを示している．

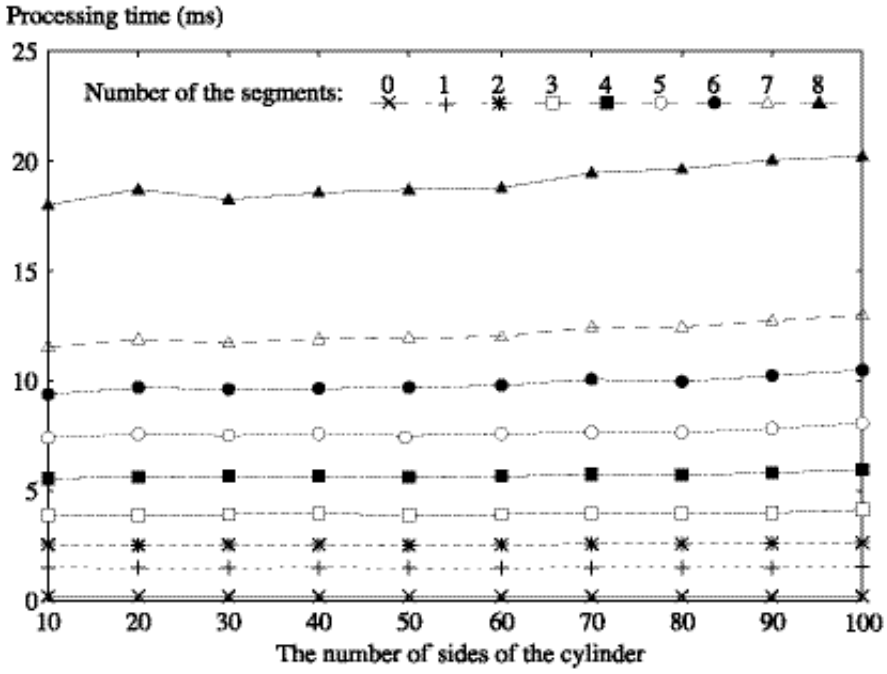


図 4.9 多角形の側面数と全集合演算処理時間の関係

図 4.10 は，同じ形状について多角柱の側面数 32（全面数 306）とし，解像度を 128×128～4096×4096 と変化させた時の処理時間を計測した結果である．これもセグメント数が 0, 4 および 8 の場合について示す．

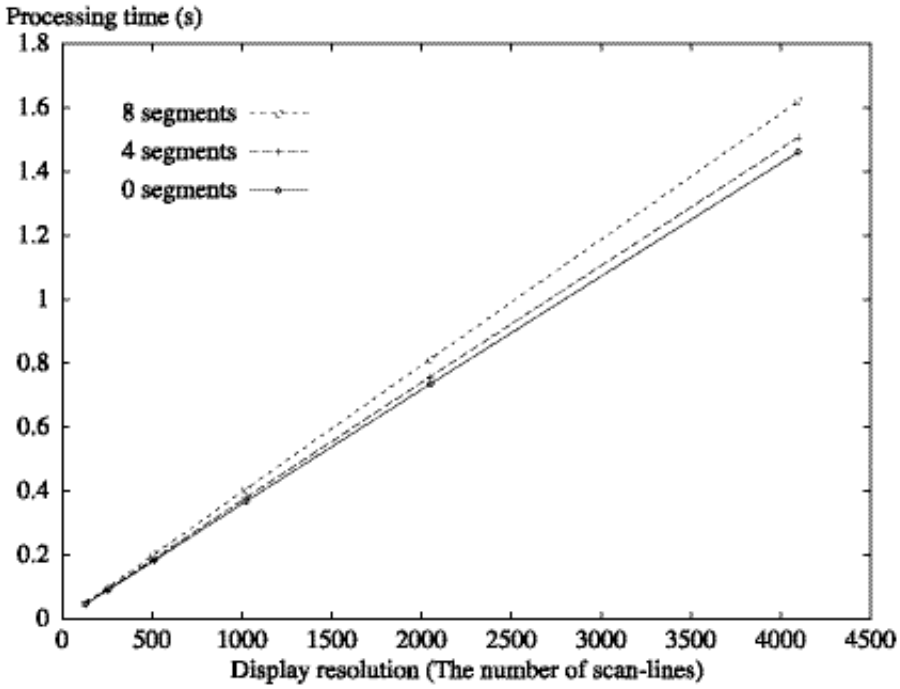


図 4.10 スキャンライン数と全処理時間の関係

視線探索法などのように画素単位に集合演算処理を実行する方式では処理時間は表示画素数に比例するが，提案アルゴリズムではほぼスキャンライン数に比例している．この実験において１次元集合演算処理に要した時間の合計のみを調べれば，図 4.11 に示す通り，これもスキャンライン数に比例している．これは提案アルゴリズムにおいて集合演算処理の実行回数が，生成画像のスキャンライン数に比例することを示している．

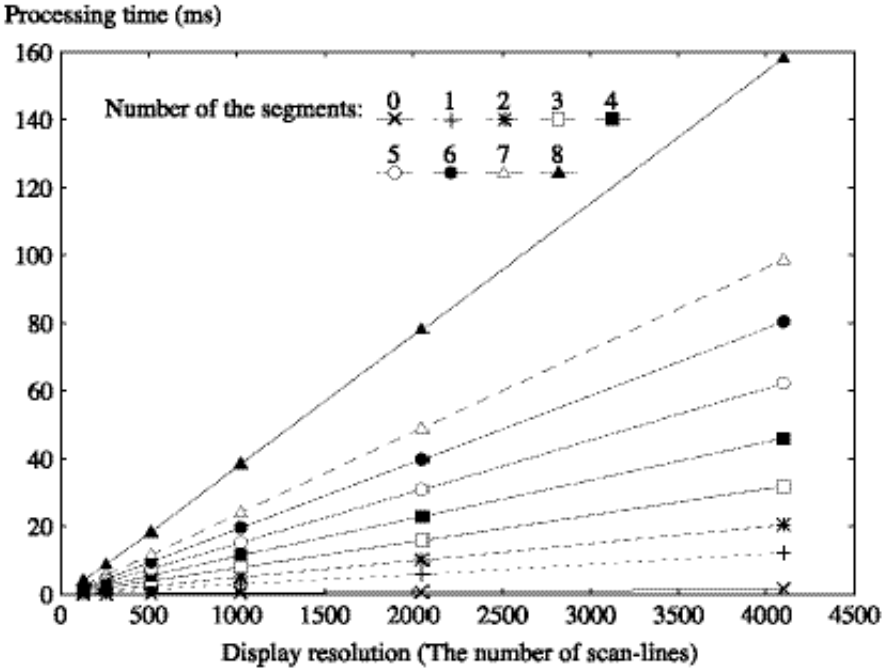


図 4.11 スキャンライン数と全集合演算処理時間の関係

図 4.12 はこの実験においてセグメント数に対する集合演算処理 1 回あたりの平均処理時間を，１次元集合演算処理におけるリファレンスカウントの検査を打ち切った場合と打ち切らなかった場合について示したものである．検査を打ち切った場合は打ち切らなかった場合に対して，処理時間はセグメント数が 8 の場合で約 48% 短縮された．

一つの面の可視判定において実行されるインクリメンタル計算は，その面が属しているセグメントの数だけ繰り返される．したがって 1 回の集合演算処理の処理時間は，セグメント数に比例すると考えられる．

図 4.12 においてセグメント数が 8 の場合に１次元集合演算処理の平均処理時間が急激に増加しているのは，図 4.7 の形状においてセグメント数が 8 の場合，比較的処理時間が短くて済む，どのセグメントにも登録されていないプリミティブが存在しないためである．

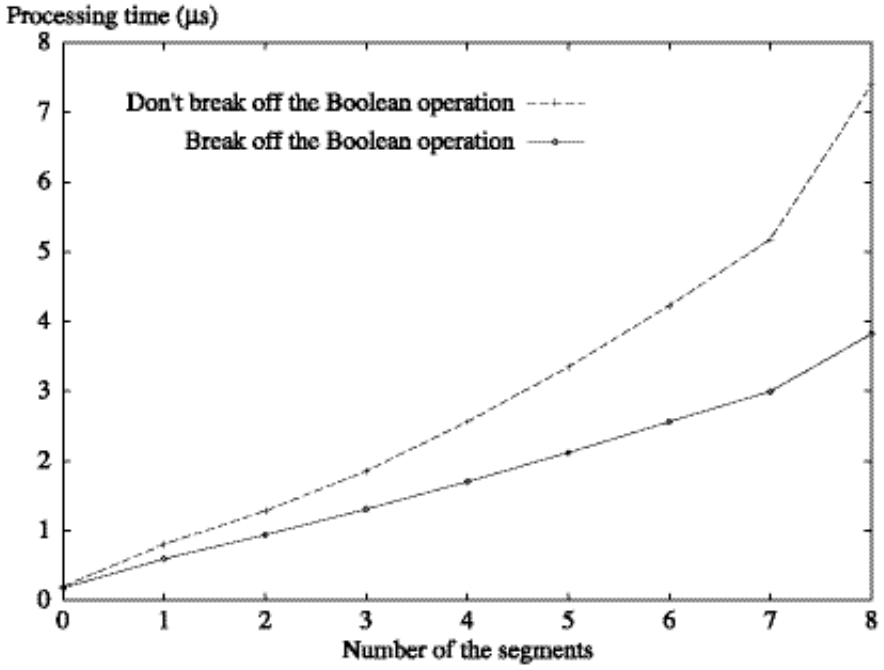


図 4.12 1 次元集合演算処理の平均処理時間

4.4.2.> 効率化の効果>

次に，図 4.7 の形状を用いて 4.3. 節で示した効率化の効果を調べた．図 4.13 は多角柱の側面数を 32（全面数 306），解像度を 512×512 として，以下の条件で画像を生成したときの処理時間を，プログラムの各部分ごとに示したものである．バーは上から集合演算処理，サンプルスパンの処理，APL の更新処理，その他の処理を示す．

(a) 何の効率化手法も用いなかった場合

(b) サンプルスパンの類似性を用いた場合

(c) 候補面の背後の面の交差を無視した場合

この図から (b) のサンプルスパンの類似性を利用した場合の効果が非常に高いことがわかる．一方，(c) の候補面の背後の面の交差を無視した場合は，集合演算処理自体の処理時間は半減しているものの，サンプルスパンの処理に追加した交差と候補面の前後判定の処理量が集合演算処理の削減量を上回り，かえって処理時間が悪化している．ただ，この増分はブール結合の記述の複雑さ（セグメント数）には依存していないので，ブール結合の記述がより複雑な場合や面の交差数が多い場合には，この手法が有効になると予想される．

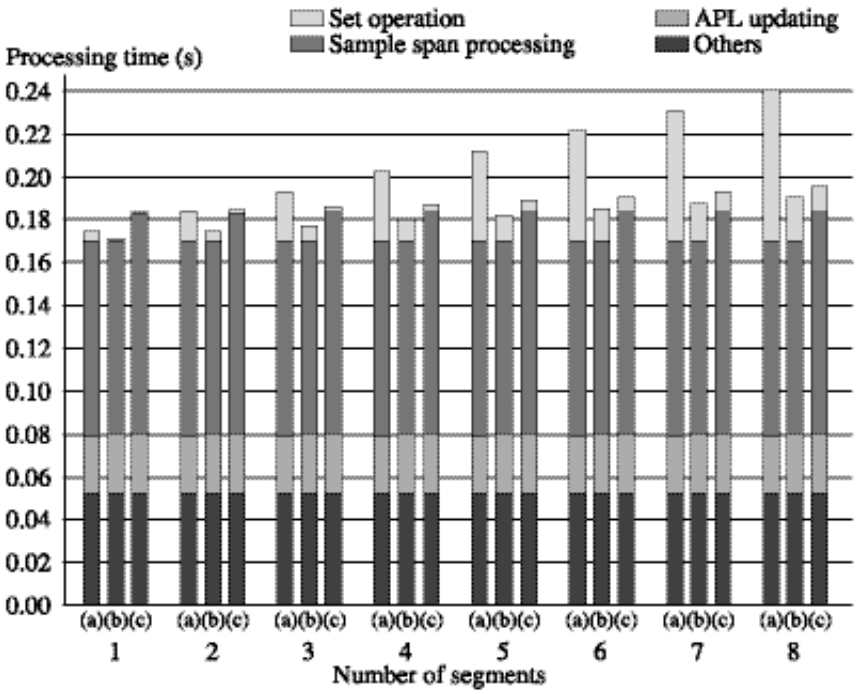


図 4.13 図 4.7 の形状に対する効率化の効果

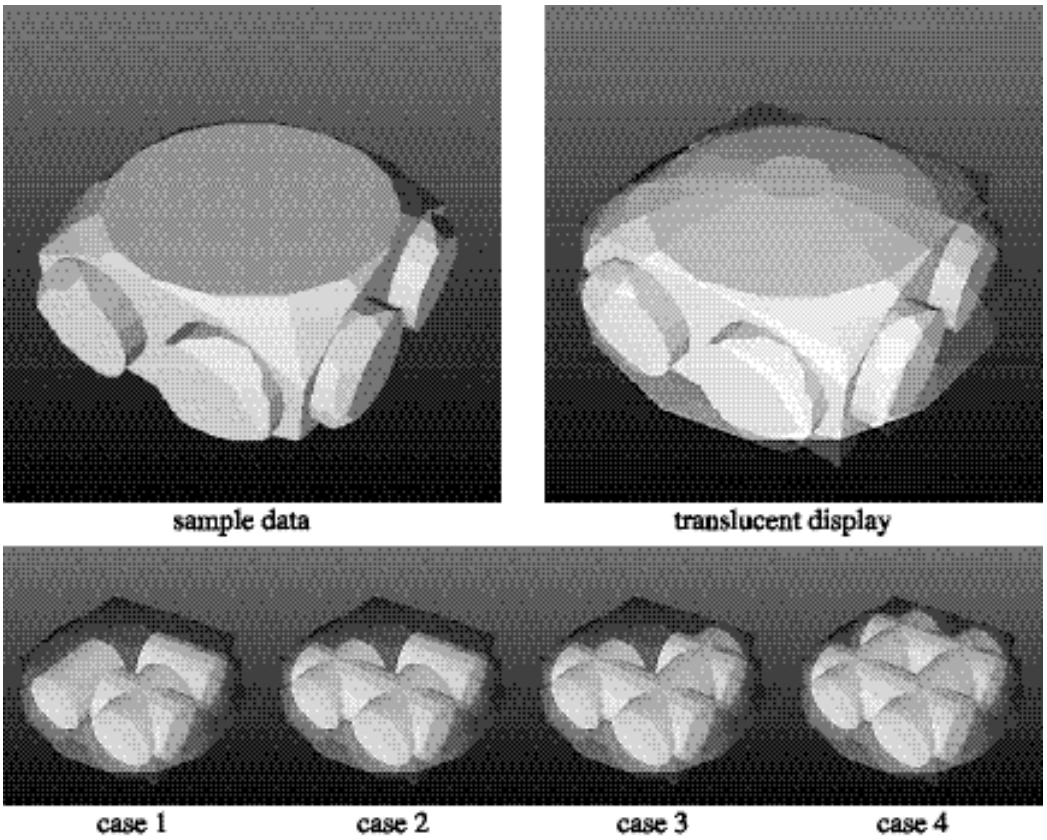


図 4.14 隠れた交差が含まれる形状

そこで図 4.14 に示す形状に対して同様の実験を行った．その結果を図 4.15 に示す．この形状は直方体と 4 個の多角柱の和と，楕円体の積である．translucent display は，説明のために積集合演算によって削られる部分を半透明表示したものである．この形状において図の case 1 ～ case 4 に示すように円柱の長さを変更し，形状に含まれる面の交差の数を変化させてプログラムの各部分の処理時間を計測した．

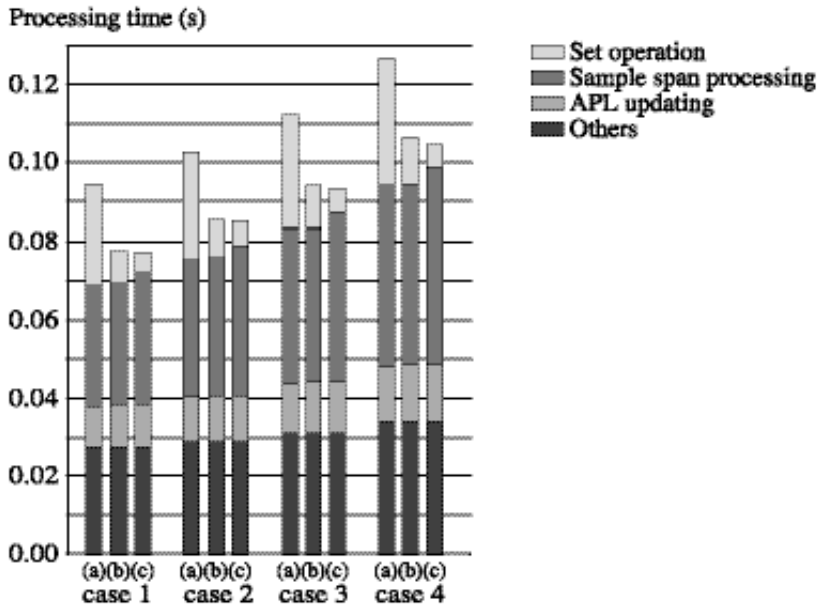


図 4.15 図 4.14 の形状に対する効率化の効果

この形状では図 4.7 の形状に比べてブール結合の記述は単純だが（セグメント数 = 5），候補面の背後の面の交差を無視した場合に処理時間が改善されている．交差の増加に伴ってその他の処理の処理時間が増加しているのは，スクリーン平面上での稜線の交差の増加により AEL の並べ替え等にかかる時間が増加したためである．

4.4.3.> 半透明表示とインタラクティブ表示>

APL にはサンプルスパンと重なる面が奥行きの順に登録されているから，可視面あるいは背景から視点側に向かって APL を逆順にたどり，その間にある面の色を合成すれば半透明表示が可能になる．図 3.6 (a) や図 4.14 の translucent display は，この方法により積集合演算によって削り取られる面を半透明表示したものである．

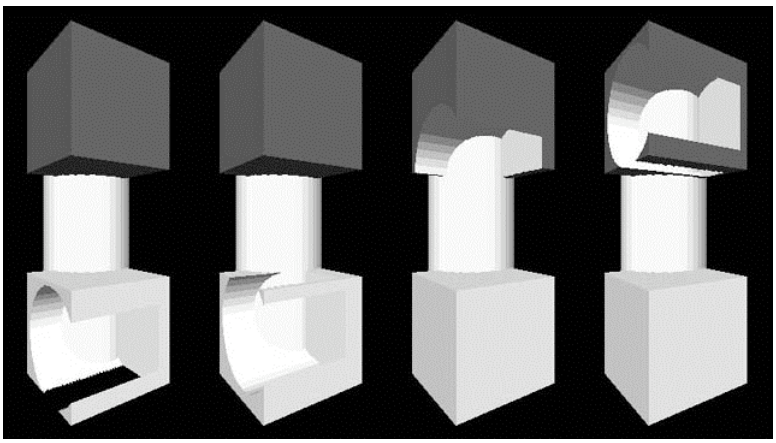


図 4.16 マウス操作によるリアルタイム形状変形表示

一方，図 4.16 はマウスによって水平の円柱（多角柱）を下から上に移動し，それに追従してリアルタイムに形状が変化する様子を示す．円柱が立方体に接するような状況でも，安定して画像生成を行うことができた．

4.5.> まとめ>

本研究では物体形状の一様性に関する情報を APL の順序や可視面の候補面などの形で保存し，可視面の決定や集合演算処理に活用して集合演算表示処理の効率化を図った．その結果，提案手法では集合演算処理に要する時間の，面数への依存を解消することに成功した．

サンプルスパンの類似性を利用した効率化は，隠面消去処理におけるセグメントチェイン [26] を利用した効率化手法に類似した手法だと考えられるが，１次元集合演算処理による可視面判定は，隠面消去処理に比べて複雑な処理となるため，集合演算表示ではこの手法により大きな効果が得られた．

一方，集合演算処理に要する時間を短縮した結果，面の奥行き計算などのサンプルスパンの処理の，全処理量に占める割合が増した．今後はこの部分の効率化を行う必要がある．また，多数のプリミティブの和に対して積をとった形状において，セグメント数の増加により集合演算処理の効率が低下する問題を避けるために，複数のセグメントに属しているプリミティブを優先的に処理できるようデータ構造を改良する必要がある．

提案手法は WED で記述された一般的な多面体形状から直接画像生成を行う手法であり，WED で記述されたプリミティブを集合演算表示するだけでなく，

形状データに対して実際に集合演算処理を実行して得られた WED 構造の形状データの表示も可能である．したがって，本手法によって集合演算後の形状を見ながらインタラクティブにプリミティブの位置決めを行い，形状データに対して集合演算を実行して得られた結果をそのまま新たなプリミティブとして利用できるような，CSG と境界表現のハイブリッドな形状モデリングシステムを効率よく実現することが可能になる．

第5章　凸形状プリミティブの集合演算表示

前章までで多面体をプリミティブに用いた場合の，CSG モデルの高速陰影画像生成アルゴリズムについて述べた．本章ではこれを，曲面を含む凸形状プリミティブに拡張する方法[13]について述べる．

このような形状をプリミティブに採用した場合，本研究のベースになっているスキャンライン法をそのまま適用することは難しい．プリミティブを多面体近似すれば前章までに述べた方法がそのまま使用できるが，曲面を含む場合は相貫線や切断線がなめらかに表示される必要があり，精度よく近似しようとすれば面の数が非常に多くなる．

また，スキャンライン法を用いてパラメトリック曲面を直接表示する方法も Blinn[30] らによっていくつか提案されているが，プリミティブとスキャンライン平面との交差線が曲線となるため，サンプルスパンのような可視面判定のための領域を設定することが難しい．

したがって，曲面を含む任意の形状をプリミティブに採用した CSG の画像生成を行う場合には，画素単位に可視判定を行う視線探索法を用いる方法[14]や拡張デプスバッファを用いる方法[16,17]が簡便である．これらは可視面の判定を 1 画素あるいは視線上で行うため，集合演算処理を 1 本の直線上の論理演算に帰着できる．反面，これらの方法は処理に時間を要するため，その高速な描画には補助的な記憶装置やハードウェアの支援が必要になる．

視線探索法を用いる方法[14]では，各プリミティブに外接箱を設定し，それを用いて視点から直接届く（すなわち反射や屈折を経ていない）「第 1 世代の視線」を制限して，不要な可視判定を省いている．また空間の等分分割によって部分空間にある図形を限定する手法[21,22]も同様に有効である．

しかし，前章までで用いてきたセグメントの局所性やプリミティブ形状の一様性を利用した効率化手法によっても，このような第 1 世代の視線や，1 本の直線上の論理演算，すなわち奥行き方向の 1 次元集合演算処理を効率化できる．たとえば積和形で記述された CSG の場合，セグメントのスクリーンへの投影像もそのセグメントを構成する正のプリミティブの投影像の内部にあるため，この投影像に対して前章に述べた計数処理による論理演算を行って第 1 世代の視線を制限できる．本章ではこれらの効率化手法の適用方法について述べる．

5.1.> スキャンライン法の任意の凸形状プリミティブへの応用>

スキャンライン法によって任意の凸形状プリミティブのスクリーンへの投影増を求めるためには，そのプリミティブから以下の情報を算出する必要がある．

(a) 物体のスクリーンへの投影像の上端および下端の位置（図 5.1 の y_1, y_2）．

(b) この投影像と交差するスキャンライン上の範囲（図 5.1 の x_1, x_2）．

また，前章までの手法ではサンプルスパンを単位として可視面を決定していたが，本章で述べる手法は基本的に画素単位に可視面を決定するため，次の情報も必要となる．

(c) スキャンライン上の 1 点におけるプリミティブの奥行き方向の範囲（図 5.1 の z_1, z_2）．

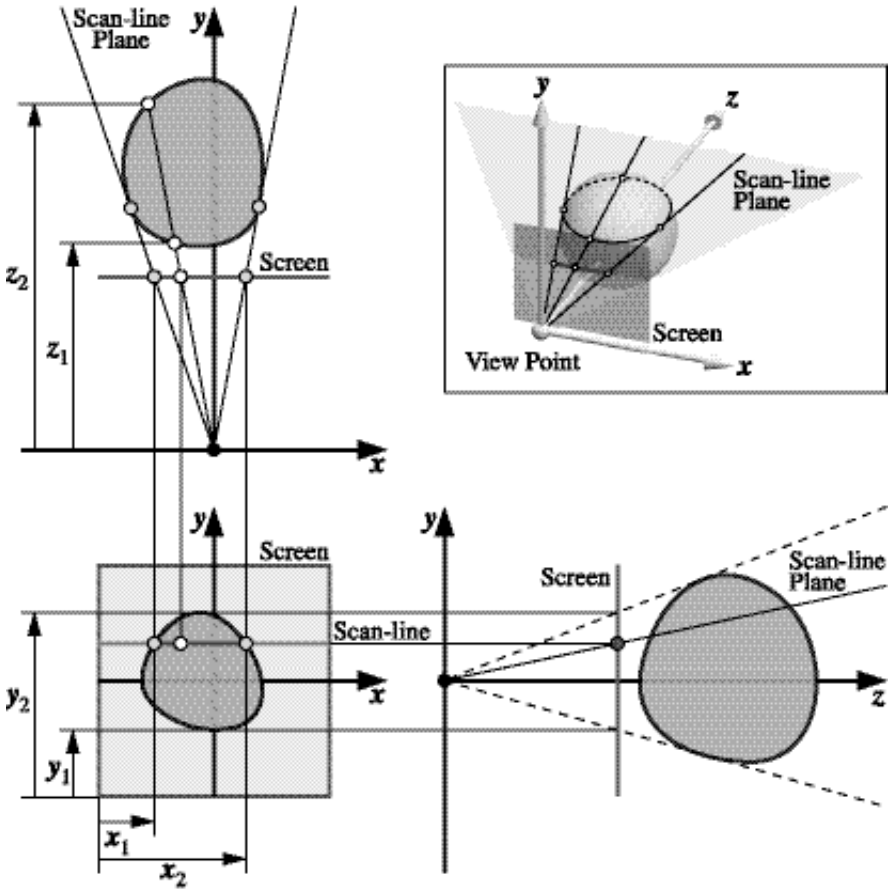


図 5.1 プリミティブの境界情報

本研究では，これらの情報を求めることができる凸形状をプリミティブに用い，一旦スクリーンエリア上で計数による論理演算を実行して，可視判定の対象となるプリミティブの絞り込みを行った．

5.2.> プリミティブ選択アルゴリズム>

各プリミティブのスクリーンへの投影像の上端と下端の位置を求め，2.1.2. 節で述べた任意の多角形の走査変換と同じ手法で y-Entry List を作成する．このとき，この y-Entry List を並べ替えておく必要はない．次に，y-Entry List をもとに，AEL の作成と同じ方法で処理中のスキャンラインと重なるプリミティブを保持する y-Occupation List を作成する．

各スキャンラインの処理は，多面体の場合と異なり，y 軸と同様にバケットソート法を用いて x-Entry List を作る．その手順を図 5.2 を例に説明する．

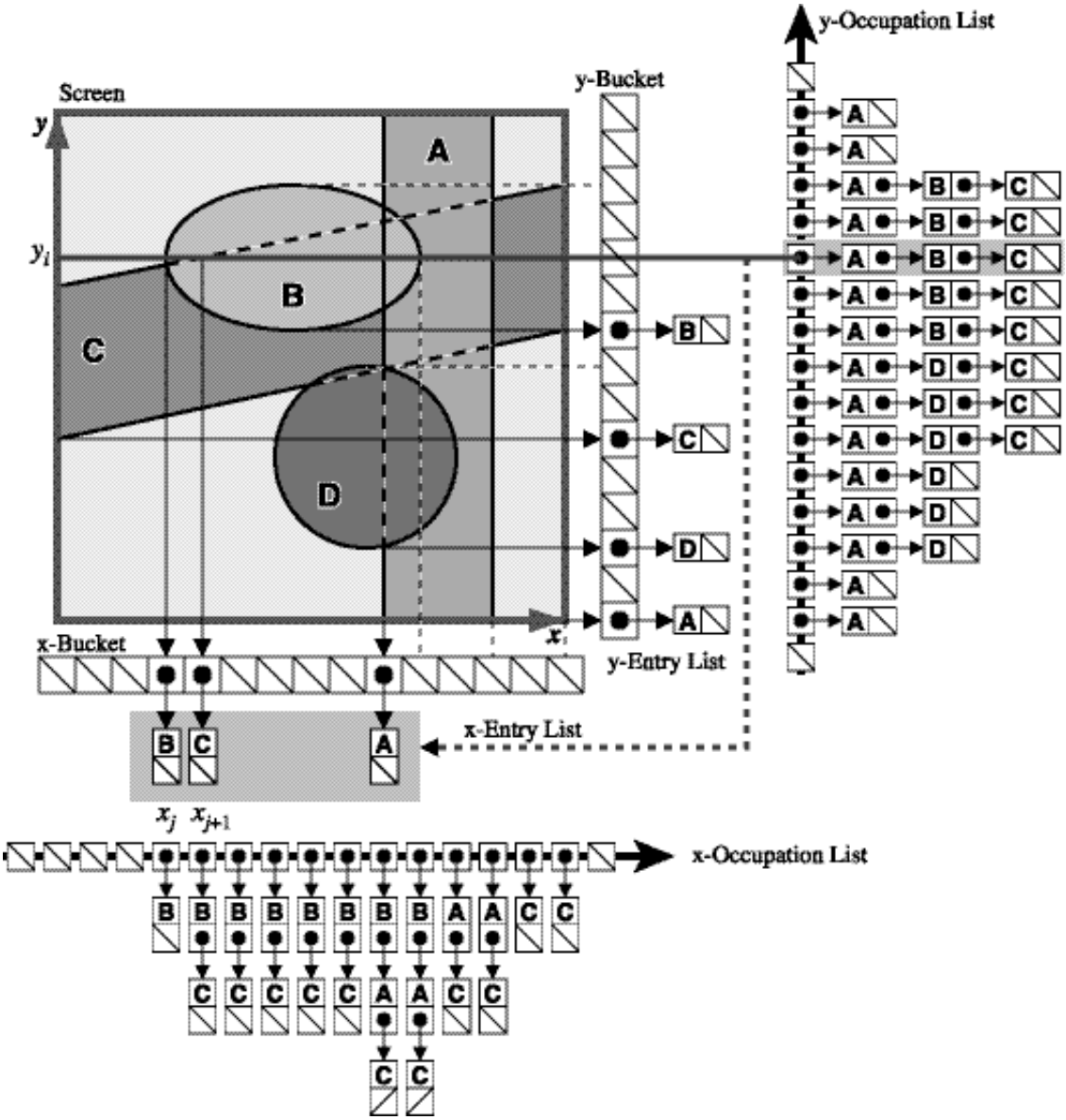


図 5.2 プリミティブ選択アルゴリズム

(1) 多角形の走査変換と同様の手法により，y_i の高さのスキャンラインと交差するプリミティブ A, B, C が y-Occupation List に登録されている．これらのプリミティブについて，スキャンライン上での左右端の位置を求める．これはスキャンライン平面上において，プリミティブとスキャンライン平面との交差線の，視点を通る接線などを求めることによって得られる．

(2) 得られたプリミティブの左端の位置をもとにバケットソート法を用いて並べ替え，x-Entry List を作成する．その際，右端の位置も記録しておく．

(3) x-Bucket の先頭の要素から順に x-Entry List を取り出し，それに登録されている各プリミティブについて処理対象の画素における奥行きの範囲を求め，単純挿入法を用いて手前側の奥行き値の順に x-Occupation List に登録する．図 5.2 では x_j の画素から B が x-Occupation List に登録され，x_{j+1} の画素から C がその背後に登録される．

(4) 現在の x-Occupation List を先頭から順にたどり，１次元集合演算処理を実行して可視面を決定する．

(5) x-Occupation List に登録されている各プリミティブのうち，処理対象の画素がスキャンライン上の範囲の右端の位置と一致するものを取り除き，残りのものについて次の画素における奥行きの範囲を求める．

(6) x-Occupation List を，求めた奥行きの範囲の手前側の位置で並べ替える．隣接する画素の類似性により，x-Occupation List の順序はそれまでと変化していないか，一部のプリミティブのみが入れ替わっているだけなので，この並べ替えにはバブルソート法を用いる．

この処理により，視点を出て処理対象の画素のスクリーン上の点を通る半直線，すなわち視線と交差するプリミティブのみを，x-Occupation List に登録することができる．この方法は視線探索法における交差判定処理と比較して，能率よく可視判定の対象となるプリミティブを選択できる．

5.3.> スクリーン上の集合演算処理>

対象の CSG モデルを 3.3.2. 節で述べた積和形で記述すれば，プリミティブのスクリーン上での投影像に対する計数処理によって，y-Occupation List や x-Occupation List に登録するプリミティブを，さらに絞り込むことができる（文献 13 では形状を “AND-table” と “Primitive-table” で表現しているが，ここでは形状が図 3.6 で示したデータ構造を用いるものとする）．

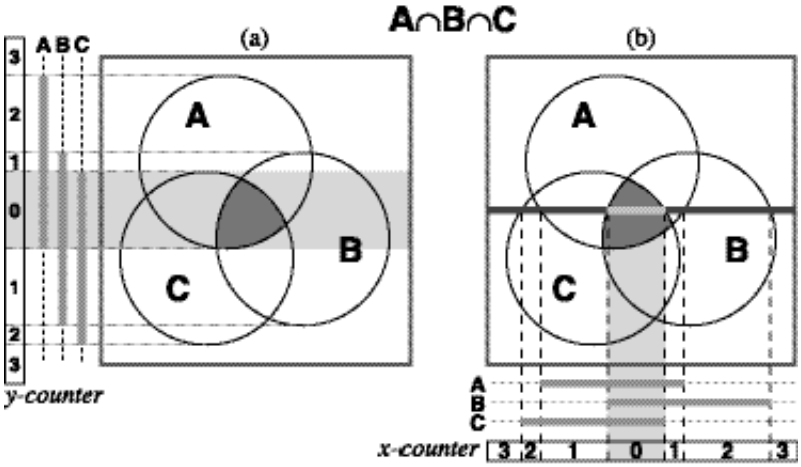


図 5.3 計数によるスクリーン上の集合演算処理

図 5.3 (a) のように，同じセグメントに属しているプリミティブのスクリーンへの投影像の上端と下端の位置から，それぞれのプリミティブが占有するスキャンラインの範囲が重なる部分を求めれば，その範囲外のスキャンラインの処理において，これらのプリミティブに関する処理を省くことができる．さらに同図 (b) のように，同じセグメントに属しているプリミティブが占有するスキャンライン上の範囲が重なる部分を求めれば，スキャンライン上のその範囲外の画素での可視判定において，これらのプリミティブを除外できる．

これらの処理は，3.3.3. 節で用いた計数による 1 次元集合演算処理と同じ手法により，効率よく実行することができる．その手順を以下に示す．

(1) 図 3.6 に示すデータ構造では，奥行き方向の 1 次元集合演算処理のために変数 *counter* を使用していた．ここではスクリーン上へのプリミティブの投影像に対する集合演算処理を同様な計数処理で行うために，二つの変数 *x-counter* および *y-counter* を追加する．これらはいずれも，そのセグメントに属している正のプリミティブの数で初期化しておく．

(2) 正プリミティブを y-Entry List から y-Occupation List に登録する際に，そのプリミティブが属しているセグメントの *y-counter* を 1 減じ，y-Occupation List から削除する際に 1 増す．なお，負のプリミティブはこの計数処理には関与しない．

(3) この処理の結果，現在の y-Occupation List に登録されているプリミティブのうち，それが属しているセグメントの *y-counter* が 0 になっているものだけが，処理中のスキャンラインにおいて可視となる可能性をもつ．したが

って，それらについてスキャンライン平面との交差線を求め，そのスクリーンへの投影像のスキャンライン上の範囲を求めて，x 軸方向のバケットソート法をもちいて並べ替える．

(4) 同様に，正のプリミティブを x-Entry List から x-Occupation List に登録する際に，そのプリミティブが属しているセグメントの *x-counter* を 1 減じ，x-Occupation List から削除する際に 1 増す. この場合も負のプリミティブはこの計数処理に関与しない．

(5) この処理の結果，現在の x-Occupation List に登録されているプリミティブのうち, それが属しているセグメントの *x-counter* が 0 となっているものが可視判定の対象となる．したがって，それらのプリミティブの処理対象の画素における前方と後方の奥行き値を求める．

(6) 奥行き方向の 1 次元集合演算処理を実行して，可視となったプリミティブのうち，もっとも視点に近いものを選ぶ.

5.4.> プリミティブの実装>

次に，以上の処理に対応したプリミティブの実装方法について，一般的に良く用いられる球（楕円体），円柱（楕円柱），円錐（楕円錐），および直方体・多角柱・多角錐などの凸多面体の各々の場合について概説する．

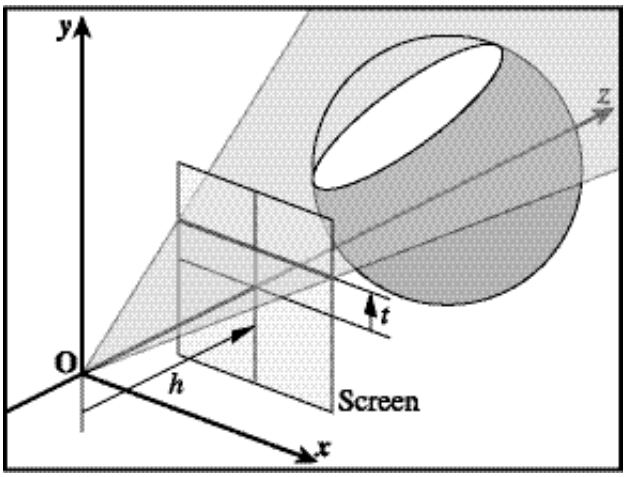


図 5.4 視野空間とスクリーンの関係

視点は図 5.4 のような左手系空間において原点の位置にあり，z 軸上の正の方向を向いているとする．またスクリーンは z 軸上の h の位置に xy 平面と平行に置かれ，スキャンラインは x 軸と平行で，$y = t$ であるスクリーン上の直線と

して表される．このとき，視点とこのスキャンラインを含むスキャンライン平面は，t をパラメータとして $hy - tz = 0$ という平面群となる．

球（楕円体）

この平面群のうち，球（楕円体）のスクリーンへの上端と下端は，この球と接するものの t として得られる．

円柱（楕円柱）

この平面群のうち，円柱（楕円柱）の上面と底面の周囲の二つの円（楕円）と接するものの t の最大値と最小値をそれぞれ上端と下端とする．

円錐（打円錐）

この平面群のうち，円錐（楕円錐）の底面の周囲の円（楕円）と接するものの t と，頂点をスクリーンに投影した位置の中から最大値と最小値を求め，それぞれ上端と下端とする．

凸多面体

多面体を構成する頂点をスクリーンに投影した位置の中から最大値と最小値を求め，それぞれ上端と下端とする．

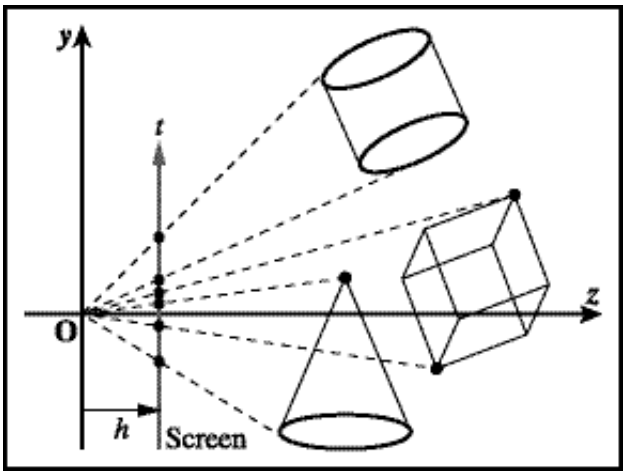


図 5.5 プリミティブの投影像の上下端

プリミティブのスキャンライン上の左端と右端の位置は，そのプリミティブと処理中のスキャンライン平面との交差線の xz 平面への投影像を求め，スキャンライン上の位置 s をパラメータとする直線群 $hx - sz = 0$ に対して，上端・下端を求めたのと同様な方法で s の範囲を求めればよい（図 5.6）．

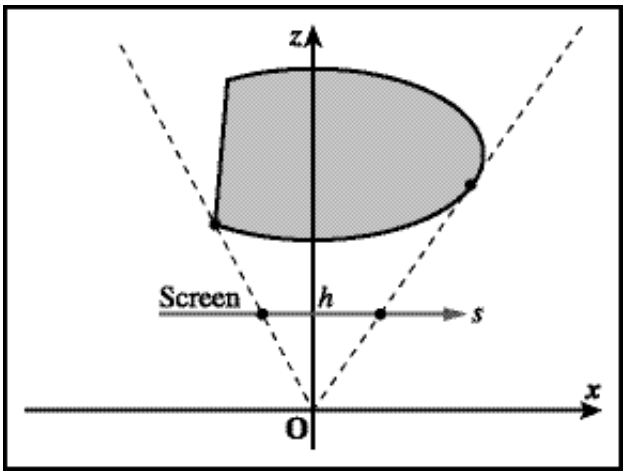


図 5.6 スキャンライン平面上のプリミティブの交差線の左右端

5.5.> 実験>

視線探索法と本手法を用いて陰影画像の生成を行い，処理時間を比較した．実験には NEC 社製 PC-9801VX (CPU i80286, FPU i80287, 8MHz) を用いた．

表 5.1 に，大きさの等しい複数の球を空間中のランダムな位置に置き，それらにブール結合関係を指定しない状態で，160×100 画素の陰影画像を生成するのに要した時間を示す．この結果，本手法は何の高速化手法も用いない視線探索法に対しては，ほぼ 100 倍の処理速度が得られることが示された．

表 5.1 ランダムな球の描画に要した時間

プリミティブ数	計算時間（単位：秒）		速度比
N_p	視線探索法	本手法	
50	364	5	72.8
100	726	8	90.8
150	1,086	11	98.7
200	1,447	15	96.5
250	1,808	18	100.4
300	2,169	20	108.5

表 5.2 は形状データにブール結合関係が含まれている場合の処理時間を示す．これはプリミティブの総数 $N_p = 120$ のときに，N_s 個のセグメントの描画に要し

た時間である（N_p/N_s は 1 セグメントあたりのプリミティブ数）．生成した画像の解像度は 160×100 画素である．この実験では集合演算により可視部分（セグメント）の形状が変化しているために，この結果の各行の間に厳密な関連はないが（N_p/N_s が 1 から 2 に変化した際の処理時間の変化は，集合演算による可視形状の変化が原因である），この場合，本手法は視線探索法に比べ，ほぼ 200 倍の処理速度を得ている．

表 5.2 プリミティブ数固定時のセグメント数に対する処理時間

セグメント数 N_s	N_s/N_p	計算時間（単位：秒） 視線探索法	計算時間（単位：秒） 本手法	速度比
120	1	867	9	96.3
60	2	874	4	218.5
40	3	894	4	223.5
30	4	901	4	225.3

図 5.7 に本手法によって生成した画像の例を示す．なお，図中に凸形状でない一葉双曲面が含まれているものがあるが，本手法は投影像が縦方向に連続していて交差線が凸形状であれば処理可能なので，このような図形も描画可能である．ただし，これは視点との位置関係に依存するため，実用的ではない．

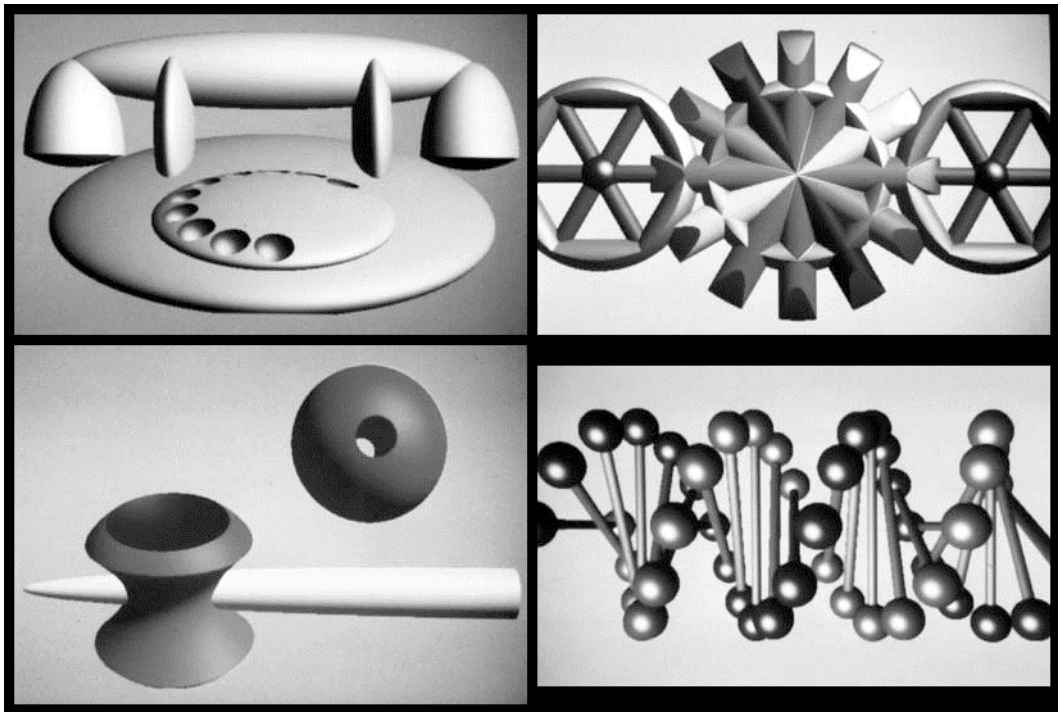


図 5.7 本手法による画像の生成例

5.6.› まとめ›

本章では，任意の凸形状をプリミティブに用いた CSG モデルの陰影画像の効率的な生成手法について述べた．本手法では，前章までに述べた積和形による CSG の記述にもとづく計数による 1 次元集合演算処理をプリミティブのスクリーンへの投影像にも同様に適用することにより，画素ごとに行う可視判定に用いる「第 1 世代の視線」の絞り込みを行った．その結果，何の高速化手法を用いていない視線探索法に対して，最大で 200 倍程度の処理速度が得られた．

しかし，本手法はプリミティブの実装に手間がかかり，多様なプリミティブの実装が困難である．またユーザーがプリミティブを追加することも難しい．曲面を含むプリミティブを扱えるようにするには，本章で述べたアプローチよりも，前章までで述べた多面体の集合演算表示手法を曲面に拡張するほうが，効率的ではないかと考える．また，曲面は多面体と比較して物体形状の一様性が利用しにくい．したがって曲面に対しては，スキャンライン法は必ずしも効率的な隠面消去法ではないかも知れない．CSG モデルの直接表示法として，今後はスキャンライン法以外の手法も検討する必要がある．

第6章> おわりに>

6.1.> スキャンライン法>

本研究では，ユーザーが直感的に理解しやすい形状データの表現形式を用いながら，対話的に形状操作を行える形状モデリングシステムを実現するために，スキャンライン法にもとづいて CSG モデルの陰影画像を直接生成する，高速かつ堅牢なレンダリングアルゴリズムの開発を行った．

スキャンライン法にもとづく隠面消去アルゴリズムは，現在でも高品質な画像を高速に生成する目的に使用されているが，この方法は元々コンピュータの能力が現在に比べてはるかに低かった時代に考案された歴史的な手法である．しかし，乏しいコンピュータリソースの上で必要なパフォーマンスを得るための努力は，この手法に高い処理効率をもたらしている．

そして，この効率向上の鍵となるのが，本論文において「物体形状の一様性」と呼んでいるもの（P. 34 ほか）である．この「一様性」は「一貫性」や「コヒーレンシ」とも呼ばれるもので，「ある点において見えているものは，その点の近傍においても見える可能性が高い」という性質である．

たとえば，可視面を画素単位ではなくサンプルスパンという単位で判定することや，直前のスキャンライン上の各サンプルスパンにおける可視判定結果の再利用などは，この性質を利用したものである．また，スキャンラインにおける稜線の x 座標値を，直前のスキャンラインにおける x 座標値に増分（傾き）を加えて求めることも，多角形がもつ物体形状の一様性が前提になっている．

本研究における第 1 番目の着想は，この「一様性」を用いて集合演算表示の効率化を図ることにある．

既に述べたとおり，視線探索法や Z バッファ法を用いた集合演算表示では，画素単位に独立して可視面の判定を行わなければならない．通常の隠面消去表示とは異なり，集合演算表示では可視面の判定の際に考慮すべき条件がかなり複雑になる．このため，画素ごとに独立して 1 次元集合演算処理を実行して可視面を決定することより，ある画素における判定結果が近傍の画素においても覆らないことを確かめる方が容易なら，後者を採用することで処理を効率化できる．隣接するサンプルスパンの類似性や，隣接するスキャンラインの類似性を利用することによって，ある画素における 1 次元集合演算処理の結果を，一

定の広さの領域に拡大することができる．その結果，理想的には，あるプリミティブが他のプリミティブと交差しない場合，そのプリミティブに対して実行される１次元集合演算処理は実質１回だけで済むはずである．これは，陰影画像生成時に集合演算処理を行っても，集合演算処理自体に要する時間が面の数（データ量）や出力画像の解像度に依存しないことを示す．

実際には，種々の条件によりそのようなプリミティブでも，１次元集合演算処理の実行回数は１回では済まない（ただし，それでもプリミティブを構成する面の数にはほとんど依存しない）．また，面分の交差部分では１次元集合演算処理が実行され，その実行回数はスキャンライン数に比例する．したがって，３章および４章で提案した多面体の直接集合演算表示アルゴリズムの処理時間は，ベースとなっているスキャンライン法による隠面消去処理と同様に，データ量とスキャンライン数にほぼ比例する．

一方，６章で述べた凸形状プリミティブの集合演算表示アルゴリズムでは，画素単位に可視面を判定しているために，処理時間はスクリーン上の画素数に比例する．この方法において，ある画素における可視面（プリミティブ）の判定後，x-Occupation List に登録されているプリミティブの次の画素における奥行き値を求め，バブルソート法により x-Occupation List のプリミティブの順序の並べ替えを行うようにする．こうすれば，プリミティブの順序の入れ替えが発生しなかったことを検出して，直線の可視プリミティブを現在の画素における可視プリミティブとすることができる．しかし，この方法における奥行き値の算出は，計数による１次元集合演算処理と比較すれば，逆に手間のかかる処理となる．そこで，この研究ではスキャンライン上で投影像の論理演算を実行して，奥行き値の算出の対象となるプリミティブを絞り込む手法を採用した．

本研究における第２番目の着想は，CSG の積和形記述の論理積項であるセグメントが表す部分物体の「局所性」を用いて，集合演算処理の効率化を図ることにある．

この「局所性」とは，いくつかのプリミティブの論理積である可視形状，すなわちセグメントが表す部分物体が，それらのすべてのプリミティブに内包されるという性質である．

CSG モデルをプリミティブの積和形で表現することによって，スキャンライン法を含むいくつかの隠面消去アルゴリズムが和集合演算を表示可能であるという性質が利用できる．これによって，可視面の決定に必要な集合演算処理を

論理積に統一することができる．さらに，セグメントが表す部分物体は，視点から実際の可視面に到達するまでに，そのセグメントを構成するプリミティブの境界面をすべて通過する必要がある．この性質を利用することによって，奥行き方向の１次元集合演算処理において通常行われるプリミティブの内外判定を，単純な計数処理に置き換えることが可能になる．ここで負のプリミティブについては，その「内部」に侵入することが，論理演算上は「外部」に脱出することになるため，単に計数時の符号を反転するだけで処理可能になる．

このように単純な計数処理で１次元集合演算処理が可能になるのには，それに関係するプリミティブの表面が既に奥行きの順に並べ替えられているという，「物体形状の一様性」を利用したスキャンライン法の特徴が寄与している．

6.2.> ハードウェアとソフトウェア>

スキャンライン法が乏しいハードウェアリソースの上で効率を最大限に追求したアルゴリズムであるという議論に対して，現在のように潤沢なハードウェアリソースが非常に低価格で入手できる状況で，その意義を考えてみたい．

過去には，スキャンライン法による隠面消去アルゴリズムをハードウェアで実装し，フライトシミュレーターなどのリアルタイム映像生成に応用した例もあった．しかし現在では，そのような目的のハードウェアには，一般的に Z バッファ法が用いられる．

Z バッファ法が図形処理ハードウェアに採用される理由は, この方法が画素単位の奥行き比較のみで実現できるために，ハードウェアによる実装が容易だからである．そして，そのような図形処理ハードウェアの性能は，単位時間あたりに描画可能な多角形（三角形ないし四角形）の数を競ってきた．ところが図形処理ハードウェアの性能向上に伴って，多角形を Z バッファやフレームバッファなどのメモリーに書き込む際の帯域幅がボトルネックになり始めている．

良く知られているように，メモリー素子の速度向上は CPU などの演算ハードウェアの速度向上に比べて格段に遅い．このため走査変換を高速に行えるハードウェアを用意しても，メモリーへの展開が足かせとなる場合がある．さらに Z バッファ法による隠面消去では, データ量が多いほど多くの多角形が上書きされて消されてしまう．つまり，大きな帯域が表示されない多角形を描くために消費されているのである．このため，あらかじめ可視となり得ない物体をソフトウェア的に取り除いておくカリングという処理が行われる場合もある．

この問題は，集合演算表示においてより顕著となる．一般的な図形処理ハードウェアは，集合演算表示に直接対応した機能を備えていないが（文献 15 の手法は Z バッファに 1 画素を書き込むごとに，ソフトウェア的に集合演算処理を実行している），ステンシルバッファと呼ばれる第 3 のメモリーを持つものは存在する．このステンシルバッファと Z バッファを組み合わせることによって，集合演算表示にハードウェアの支援を利用できる．

しかし，この方法はプリミティブが凸形状の多面体に限られるほか，一つの面を Z バッファとステンシルバッファに複数回描き込む，いわゆるマルチパスレンダリングの手法を採らざるを得ない．このことは，この手法による集合演算表示が，より大きなメモリー帯域を要求することを示している．

これに対して，本研究の手法はソフトウェアのみによる実装ではあるが，前述のようなメモリー指向のアプローチを採用せず，画面に表示されるものだけを効率よく抽出することに注力している．また AEL や APL によって処理対象となるデータの管理を局所化しているため，近年の CPU の性能向上の重要な要素であるキャッシュメモリーとの整合性がよい．実際，5 章で述べた手法では，図形処理ハードウェアの支援が無くとも，ほぼリアルタイムに集合演算表示を行うことができた．

また，現時点の図形処理ハードウェアの動作周波数が 300MHz に達していないのに対し，CPU の動作周波数は 2GHz に到達し，さらに速度を上げようとしている．また図形処理ハードウェアは内部の並列処理によって実行性能を高めているが，CPU も「マルチメディア命令」と称して SIMD 型の命令を追加するなどして，高い性能を実現している．したがって，集合演算表示のように可視面決定に複雑な手続きを用いなければならないような場合，本研究のようなソフトウェア的なアプローチでも，当面の需要には対応できると考えている．

しかし，ゲームやテーマパークのアトラクション，あるいはバーチャルリアリティなどの用途では，リアルタイム性に加えて生成される画像の品質（リアリティ）も問われるようになってきた，そのために図形処理ハードウェアの機能もさらに高度化しつつあり，最近ではプログラマブルな機能を持つものも登場している．そのような図形処理ハードウェアは，CPU に対して GPU (Graphics Processing Unit) と名付けられている．したがって今後は，このようなプログラマブルな図形処理ハードウェアを用いて，集合演算表示の可視面判定にハードウェアの支援を利用する手法も視野に入れる必要がある．

6.3.> デザイナーのアプローチとエンジニアのアプローチ>

本研究は，あるデザイナーが「CSG によるモデリングはわかりやすくて面白いが，結果を得るまでに時間がかかったり，思ったような結果が得られなかったり，取り消しが効かなかったりするところに難がある」と評していたところに端を発する．

1983 年末，当時のスーパーミニコンピュータである DEC 社製 VAX11/780 につながれた文字端末 (VT-100) に向かって，デザイナーが形状モデリング作業に取り組んでいるのを目の当たりにした．それから 18 年の歳月を経て，コンピュータや CG の環境は大きく進化した．それにもかかわらず，その頃から感じていた形状モデリング作業の際の**もどかしさ**は，様々な道具が整備された現在でも，未だに消えずに残っているように感じられる．

形状データの表現形式が，デザイナーにとっては造形の素材であると同時に，空間を認知して形状を操作するためのスキームとしても用いられるという見解は既に述べた．しかし CSG とて，そのイメージの世界を記述するのに十分な能力を持っているとは言えない．

その一方で，今後も「メディア系コンテンツ」に対する需要は高まる一方だろう．したがって，今後も形状データの表現形式やモデリング手法に対する要求が発生してくるものと予想される．実際，補遺に挙げたもの以外にも，様々な形状データの表現形式や，ユーザーインターフェースとしての形状の記述方式が提案されており，それらは適材適所で，あるいは互いに組み合わせて提供されている．

とはいえ，小刀一本で様々な造形を行う作家がいるように，機能の膨大な組み合わせがすべてを満足できるとは限らない．スケッチベースのモデリング[31]のように，見かけはシンプルながら斬新なモデリング手法も提案されている．しかし，我々は未だ「実体のような実物感を持ちながら，自由に，思ったように造形できる素材」を提供するに至っていないように思う．

そのような状況の中でも，デザイナーは既存の道具を駆使して，どんどん新しい表現や造形を生み出している．我々は彼らに対して，彼らの感性を触発し，さらに新しい発想を生み出すような素材を今後も提供し続ける義務がある．

謝辞>

本論文は，大阪大学産業科学研究所 北橋 忠宏 教授のご指導のもとに，筆者が豊橋技術科学大学大学院工学研究科情報工学専攻修士課程在学中，和歌山大学経済学部産業工学科，大阪大学産業科学研究所音響材料部門（北橋研究室，内地研究員），および和歌山大学システム工学部デザイン情報学科において行った研究の成果をまとめたものです．

最初に，20 年の長きにわたり，懇切なるご指導，ご鞭撻，ならびに公私にわたるご厚情を賜りました 北橋 忠宏 先生に対し，深甚なる感謝の意を表します．本研究も全行程を通じて北橋先生ご自身によるご指導，ご助言，そして数々のご支援を戴きました．重ねて深く御礼申し上げます．

また，本論文の執筆にあたり貴重なご教示を戴きました大阪大学大学院基礎工学研究科 萩原 兼一 教授，ならびに大阪大学サイバーメディアセンター 竹村 治雄 教授に心から感謝いたします．

本研究の端緒にあたり多くのご討論やご示唆を戴いた，豊橋技術科学大学 旧北橋研究室の皆様に深く感謝いたします．また同研究室のメンバーでもあったブラザー工業（株）研究開発センター 角谷 裕司 主任研究員には，本研究と２次元図形の論理演算との関連について重要なご示唆を戴きました．厚く御礼申し上げます．

本研究の中盤の遂行にあたり多くのご討論やご示唆，ならびに研究遂行上のご支援を戴きました大阪大学産業科学研究所北橋研究室の皆様に深く感謝いたします．中でも 平井 誠 先生（現 大阪市立大学 助教授）には，豊橋技術科学大学時代から懇切なるご指導，ご助言を戴きました．厚く御礼申し上げます．

本研究を遂行する機会を与えていただき，あたたかいご支援を賜りました和歌山大学経済学部 竹内 昭浩 教授，ならびに 八丁 直行 助教授に深く感謝いたします．

本研究の国際会議での発表に際し，懇切なるご指導とご支援を賜りました和歌山大学教育学部 Leonard Lundmark 教授に深く感謝いたします．Lundmark 先生には公私にわたってご厚情を戴きました．重ねて御礼申し上げます．

また日頃から筆者の研究活動全般において，ご討論やご支援を戴いている和歌山大学システム工学部デザイン情報学科，ならびに同大学大学院システム工

学研究科グラフィックスクラスタの皆様に深く感謝いたします．なかでも 吉本 富士市 教授には，本研究の遂行にあたってご助言や多大なご支援を戴きました．深く感謝いたします．また 河原 英紀 教授には，研究に対する重要なご助言とご指導を賜りました．深く感謝いたします．

そのほか，筆者の研究活動全般において，非常に多くの方々からご協力，ご助言，ご支援を戴きました．筆者の研究活動に関係するすべての方々に，個々に記して感謝の意を表します．

最後に，知的障害を持つ息子を育てる傍ら，筆者の健康を気遣い，生活全般にわたって筆者を支援してくれる妻と，その妻と兄を明るい笑顔と持ち前の元気さで支えてくれる我が娘，そして，その障害をもって筆者に「さまざまな『もの』の見方」に関する知見を与えてくれる我が息子に感謝します．

補遺› 形状データの表現形式›

境界による表現›

立体の形状を，その内部と外部を隔てる境界，すなわち面や，面を構成する稜線・頂点の情報（境界情報）を用いて表す場合を，一般に境界表現 (Boundary Representation, B-reps)と呼ぶ（図 補遺 1）．形状に曲面が含まれている場合は，その部分を２次曲面やパラメトリック曲面で表現する場合もあるが，多角形近似を行う場合も多い．下図の例では，直方体の 8 個の頂点を合計 12 本の稜線で結び，さらに 1 面を 4 本の稜線（辺）で表現している．これを頂点表，稜線表，多角形表の三つの表からなるデータ構造を用いて記述している．

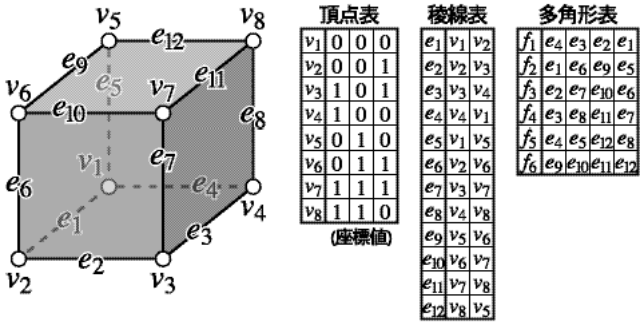


図 補遺 1 境界情報を用いた形状の表現

実際のシステムでは，立体形状データの表現に WED (Winged Edge Data) というデータ構造がよく用いられる（4.1. 節参照）．

境界表現は複雑な立体の形状を正確に表現することが可能である．また立体の表面形状の情報を明示的に保持しているため，画像の生成を効率良く行うことができる．このように境界表現はコンピュータにとって効率の良い形状データの表現形式であるとされ，多くの形状モデリングシステムで用いられている．

反面，この方法はデザイナーが対話的に形状を作成する場合には，必ずしも効率の良いものであるとは言えない．この方法で表現された形状の作成や修正は，基本的に頂点や稜線，あるいは面などの細かな形状要素を一つ一つ指定して操作する局所変形操作 (Local Operation) によって行われる（図 補遺 2）．

この局所変形操作による形状作成は，ディティールの作り込みには有効だが，全体を見通した形状変更は必ずしも容易ではない．また意図した形状を得るためにデザイナーが複雑な手順を考える必要があるなど，造形手法が現実世界で

用いられているものから程遠い．このため多くの形状モデリングシステムでは，ユーザーインターフェースとして他の形状データの表現形式を用い，作成された形状を境界表現による内部データに変換できるようになっている．

なお指定した稜線や頂点付近の角を取る丸め変形操作も局所変形操作の一つだが，これは単純な局所変形操作の組み合わせで実現される，高度な形状操作うちの一つである．

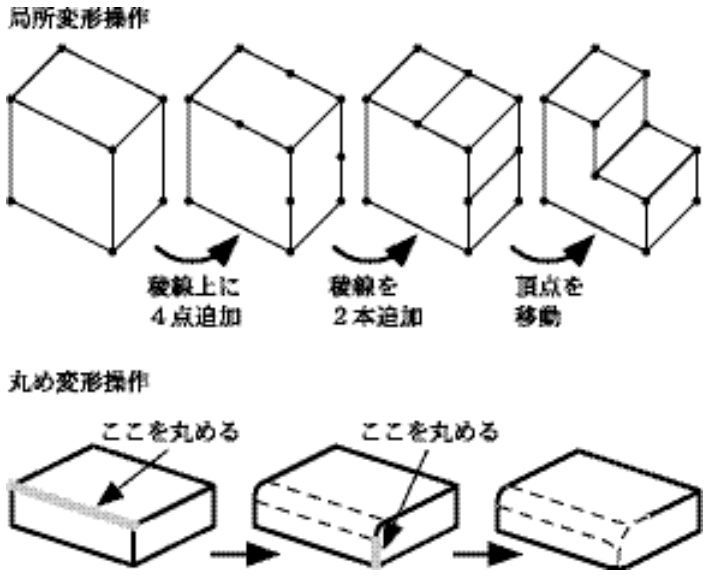


図 補遺 2 局所変形操作と丸め変形操作

立体の組み合わせによる表現>

表現しようとする形状が立体であれば，それをあらかじめ形状が明らかな別の立体の組み合わせで表現するという方法も採用できる．

プリミティブインスタンス法

工業製品のような人工物の形状は，単純な形状の組み合わせで表現されていることが多い．そこで積み木のように単純な形状を持つ基本立体（プリミティブ）をいくつか用意しておき，それらの組み合わせで形状を定義することができる（図 補遺 3）．この方法をプリミティブインスタンス法と呼ぶ．プリミティブは手続きによって自動生成することも可能であり，大きさや位置，回転角などの情報の他，形状の特徴（たとえば歯車であれば歯数など）のパラメータを与えれば，同じ種類の異なる形状のインスタンスを生成される．この方法は，用意されたプリミティブの種類が作成しようとする形状に適合していれば効率の良い形状作成が可能だが，適合していない場合の形状はうまく表現できない．

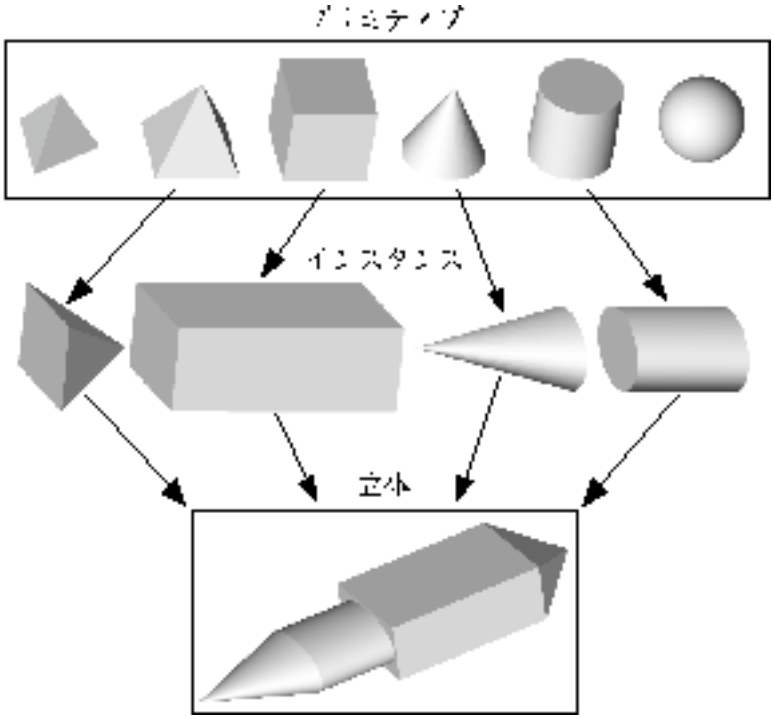


図 補遺 3 プリミティブインスタンス法

Voxel 表現

Voxel 表現は空間を等分分割した立方体の集合によって形状を表現するものである（図 補遺 4）．この立方体を Voxel という．この形式の形状データはコンピュータ断層写真などから得られることが多いが，彫塑と同様な方法で形状を作成する場合にも用いられている[32]．等分分割された個々の部分空間を物体が占有するか否かで形状を表すため，個々の Voxel の有無は 1bit で表現できる. コンピュータ断層写真から作成したデータの場合は，Voxel ごとに濃度が与えられていることもある．その場合，画像を生成するには，濃度の境界を検出して実際に見える表面を判定する必要がある[33].

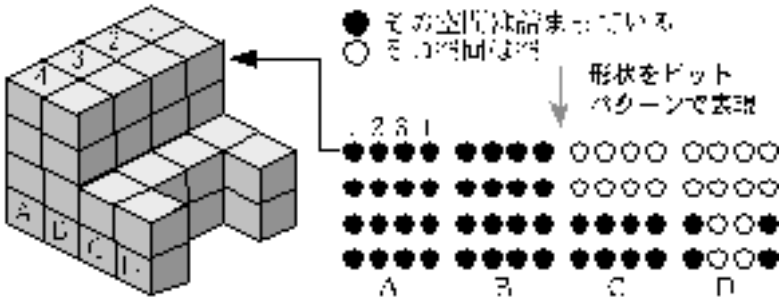


図 補遺 4 Voxel 表現

Oct-Tree (Octree) 表現

Voxel 表現による形状データを階層的に構成することによって，データ量の削減を行ったものである（図 補遺 5）．空間を 8 個に等分分割し，個々の部分空間を，○全部空である●全部埋まっている◎一部埋まっている，の三つに分類する．そして，◎一部埋まっている部分区間について同様の分割を行い，これらの部分空間の包含関係を 8 分木で表現する．形状の作成や修正には Voxel 表現とほぼ同じ手法を用いることができるが，その手続きは多少複雑化する．ただしデータ量が圧縮されているため，処理時間は短縮することが可能である．また，部分空間の前後関係を 8 分木から求めることができるため，軸測図などが短時間で生成できる．

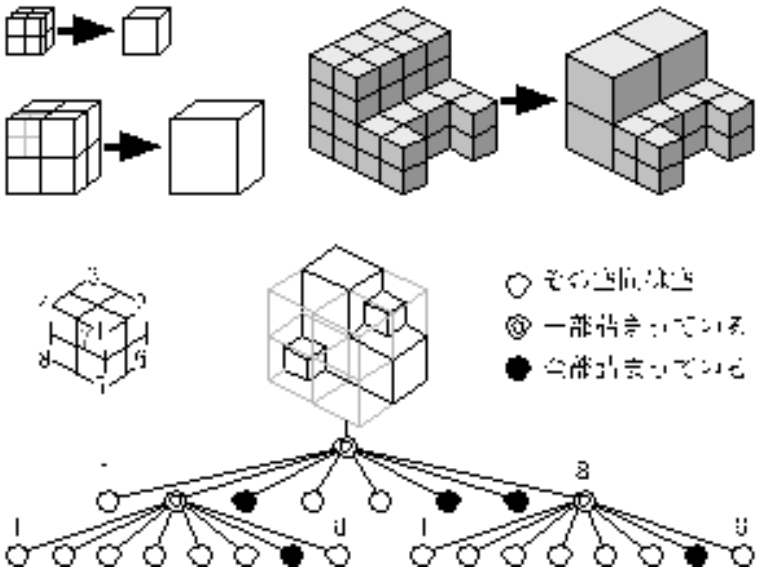


図 補遺 5 Oct-Tree 表現

これらの方法は基本立体の形状が明らかなので，体積などのマスプロパティの算出が容易である．また個々の基本立体の画像を生成する手法が存在すれば，定義された形状の画像を生成することは難しくない．反面，基本立体の種類が限られているため任意の形状を正確に表現することが難しく，近似形状を作成する場合は精度を上げるにしたがってデータ量が増大してしまう．特に Voxel 表現では精度を上げるとデータ量が膨大となり，形状の変更や画像の生成に時間がかかるようになる．これは Oct-Tree 表現においても同様だが，セル形状を拡張することによって，この問題を改善する手法が提案されている[34]．

生成過程による表現>

生成過程による表現とは，目的の形状を得るに至る（生成する）ための手段や手続きを形状の記述に用いるデータとして用いる方法である．

局所操作の組み合わせによる表現

局所操作は境界表現における形状の作成および修正の手段だが，目的の形状を得るまでの手順をすべて記録しておくことにより，それを再生して同じ形状を得ることができる．これは，この手順の組そのものを形状データとして利用できることを意味している．また境界表現データに局所操作による変更を加えた場合，変更の取り消し (undo) を行うには以前のデータを回復する手段が必要になるが，局所操作の履歴が保存されていれば，任意の時点にさかのぼって形状を復旧することが可能になる [25]．さらに，任意の時点での局所操作のパラメータを変更して，形状を修正することも可能である．したがって，これは試行錯誤をともなう対話的なモデリングに適した形状の表現方法である．しかし，このデータから画像を生成するためには，一旦境界表現などによる形状データを作成する必要がある．

CSG (Constructive Solid Geometry)

CSG はプリミティブインスタンス法において，プリミティブ間にブール代数にもとづく和・差・積の結合関係（ブール結合関係）を記述することによって形状を表現する方法である（図 補遺 6）．このブール結合関係の記述には木構造を用いることが多いが，内部的にブール演算式を加法標準形で保持することもある [24]．この方法はプリミティブインスタンス法に比べて柔軟に形状を表現できる．またこの方法は粘土をくっつけたり削ったりする感覚でモデリングできるため，デザイナーにとっても理解しやすい．さらに木構造による表現は，形状作成の手順を保持するため取り消し処理が容易に実現でき，プリミティブのインスタンスのパラメータ変更やプリミティブの取り替えによる形状の変更も容易である．したがってこの方法も対話的なモデリングに適した形状の表現方法と言える．しかしデータに最終的な表面形状が明示的に含まれていないため，画像を生成するには集合演算処理を実行して実際の表面形状を求める必要がある．

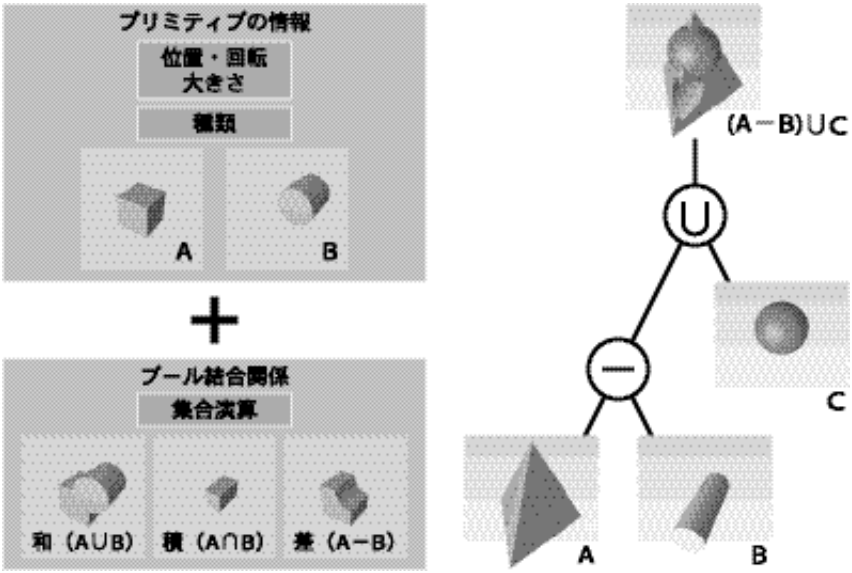


図 補遺 6 集合演算と CSG の木構造による表現

スイープ

２次元図形を３次元中で移動した軌跡を用いて形状を表現する方法である．クロスセクション（２次元断面図形）と，それを移動するパス（経路）を形状データとして用いる（図 補遺 7）．この方法は単独でも多様な形状を表現できるが，CSG などのプリミティブとして用いられることも多い．クロスセクションとパスは互いに独立して修正できる．また，経路中に形の異なる複数のクロスセクションを置くことにより，非常に柔軟に形状が表現できる．これは VRML でも採用されている．この方法も実際の表面形状を保持していないため, 画像を生成する際に表面形状を求める必要がある．

図 補遺 7 スイープ

現実の造形では，素材に形状操作を繰り返し適用して目的の形状を得る．生成過程による表現は，この形状操作自体を形状記述に用いるため，現実の造形手法との親和性が良い．反面この方法は形状記述自体に実際の表面形状を明示的な形で保持しないため，画像などの出力を作成する際には何らかの方法で表

面形状を求める必要があり，その分画像の生成に時間がかかる場合がある．このため形状モデリングシステムの内部で境界表現データも併用し，インタラクティビティの向上を図ることが多い．またこれらは境界表現による形状モデリングシステムのユーザーインターフェースの一部に用いられる．

密度や濃度による表現>

自然界には煙のように姿を見ることはできても，正確な形状を表現することが難しいものも存在する．以下の表現形式は人工物の表現に用いられることは少ないが，３次元 CG ではリアルなシーンの表現に欠かせないものである．

パーティクル

パーティクルは煙や炎，葉の生い茂った木など，明確な表面形状を定義することが困難な形状の表現に用いられる．これは大きさを持たない微小形状（粒子）の密度分布を用いて形状を表現する．木の葉などの場合は有限の大きさを持つ微小形状の集合も用いられる．これは密度分布や微小形状の集合体全体の「見え方」をモデル化することによって画像を生成する．

メタボール

メタボールは濃度球あるいはブラブとも呼ばれ，空間中に分布する濃度の等値面（等濃度面）を用いて形状を表現する方法である．濃度は空間中の点電荷による電位分布のように，ある位置からの距離に対して単調に減少する関数を用いて表現する．この方法は滑らかな曲面を対話的に作成するのに適するが，平面や角，あるいはあらかじめ形状が決まっている曲面を正確に再現することは難しい．また画像を生成する場合には，濃度がある閾値に達する点（面）を検出して表面（等値面）を判定する必要がある．

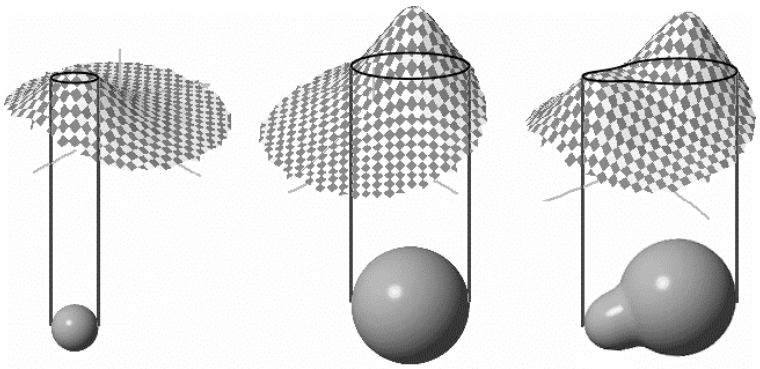


図 補遺 8 メタボール

参考文献>

[1] Coons, S.: “An outline of the requirements for the computer aided design,” Proc. of Spring Joint Computer Conference, 1963.

[2] Sutherland, I. E.: “SKETCHPAD: A Man-Machine Graphical Communication System,” Ph. D. dissertation, MIT, January 1963. Abridged version is SJCC 1963, Spartan Books, Baltimore, Md., P. 329.

[3] 山口富士夫: “コンピュータディスプレイによる図形処理工学”, 日刊工業新聞社, P. 5, 1981.

[4] 和歌山大学システム工学部デザイン情報学科: “デザイン情報学入門,” P. 91, 日本規格協会, 2000.

[5] 技術系 CG 標準テキストブック編集委員会監修: “コンピュータグラフィックス,” P. 117, 画像情報教育振興協会, 1995.

[6] 大野義夫: “線分の発生,” PIXEL, No.11, pp. 101-107, 1983.

[7] Bresenham, J. E.: “Algorithm for Computer Control of a Digital Plotter,” IBM System Journal, Vol. 4, No. 1, pp. 25-30, 1965.

[8] Foley, J. D. et al.: “Computer Graphics Principles and Practice (2nd Edition),” Addison-Wesley, pp. 92-99, 1990.

[9] 大野義夫: “多角形のスキャンコンバージョン(2),” PIXEL, No. 16, pp. 136-141, 1984.

[10] Sutherland, I. E. and Hodgman, G. W.: “Reentrant Polygon Clipping,” CACM, Vol. 17, pp. 32-42, 1974.

[11] Weiler, K.: “Polygon Comparison Using a Graph Representation,” ACM Computer Graphics, Vol. 14, pp. 10-18, 1978.

[12] 床井浩平, 北橋忠宏: “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.

[13] 床井浩平, 北橋忠宏: “凸な立体の集合演算によって定義された形状のスキャンライン法による陰影画像生成,” 情報処理学会論文誌, Vol. 30, No. 1, pp. 81-90, 1989.

[14] Roth, S. D.: “Ray Casting for Modeling Solids,” CGIP, Vol. 18, pp. 109-144, 1982.

[15] Rossignac, J. R, and Requicha A. A. G.: “Depth Buffering Display Techniques for Constructive Solid Geometry”, IEEE CG&A, Vol. 6, No. 9, pp. 29-39, 1986.

[16] Okino, N., Kakazu, Y, and Morimoto, M.: “Extended Depth-Buffer Algorithms for Hidden-Surface Visualization”, IEEE CG&A, Vol. 4, No. 5, pp. 79-88, 1984.

[17] 川島泰正, 太田吉美, 徳増真司: “CAD 対話インタフェースのためのソリッドモデル隠面処理法,” 情報処理学会論文誌, Vol. 28, No. 2, pp. 147-155, 1987.

[18] Atherton, P. R.: “A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry”, ACM Computer Graphics, Vol. 17, No. 3, pp. 73-82, 1983.

[19] 山口富士夫: “コンピュータディスプレイによる図形処理工学”, 日刊工業新聞社, pp. 281-292, 1981.

[20] Jansen, F. W.: “Depth-order point classification techniques for CSG display algorithms,” ACM Transaction on Graphics, Vol.10, No.1, pp. 40-70, 1991.

[21] Glassnar, A.S: “Space subdivision for fast ray tracing,” IEEE CG&A, Vol.4, No.10, pp.15-22, 1984.

[22] Fujimoto, A. and Iwata, K.: “Accelerated Ray Tracing,” Proceedings of CG Tokyo ’85, pp. 41-65, Springer Verlag, 1985.

[23] Goldfeather, J. and College, C.: “Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning”, IEEE CG&A, Vol. 9, No. 3, pp. 20-28, 1989.

[24] 木下正博: “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.

[25] 千代倉弘明: “ソリッドモデリング,” 工業調査会, 1985.

[26] Yamaguchi, F. and Tokieda, T.: “Development of Hidden Line Elimination Algorithm and Hidden Surface Elimination Algorithm by Classification Method,” Proceedings of Graphics and CAD Symposium, pp. 133-140, 1983.

[27] 床井浩平: “スキャンライン法によるリアルタイム集合演算表示,” 映像情報メディア学会冬季大会 (ITE ’97) 講演予稿集, P. 71, 1997.

[28] 床井浩平, 北橋忠宏: “スキャンライン法による会話的 CSG 表示アルゴリズム,” 映像情報メディア学会誌, Vol. 54, No. 7, pp. 1061-1068, 2000.

[29] Tokoi, K. and Kitahashi, T.: “An Improved Scan-Line Algorithm for Display of CSG Models,” Proceedings of Information Visualization (IV2001), pp. 477-482, 2001.

[30] Blinn, J. F.: “A Scan-Line Algorithm for Displaying Parametrically Defined Surfaces,” ACM Computer Graphics, Vol. 12, No. 3, P. 27, 1978.

[31] Igarashi, T., Matsuoka, S., Tanaka, H.: “Teddy: A Sketching Interface for 3D Freeform Design,” SIGGRAPH 99 Conference Proceedings, pp. 409-416, 1999.

[32] 水野慎士, 岡田稔, 鳥脇純一郎, 横井茂樹: “仮想彫刻－仮想空間における対話型形状生成の一手法,” 情報処理学会論文誌, Vol. 38, No. 12, 1997.

[33] Lorensen, W. E. and Cline, H. E.: “Marching Cubes – A High Resolution 3D Surface Construction Algorithm,” ACM Computer Graphics, Vol. 21, No. 4, pp. 163-169, 1987.

[34] 米川和利, 小堀研一, 久津和敏郎: “空間分割モデルを用いた形状モデラ,” 情報処理学会論文誌, Vol. 37, No. 1, pp. 60-69, 1996.