グラフィクスとCAD 101－10
（2000．12．8）

# モデルの階層的凸近似による仮想空間高速描画手法

小野澤 晃　　北澤 仁志
NTT 生活環境研究所
〒243-0198 神奈川県厚木市森の里若宮 3-1, Email:onoz@aecl.ntt.co.jp

要旨：3次元モデルを少数の凸多面体で近似することを用いたocclusion cullingアルゴリズムについて報告する。3次元モデルを用いた大規模な仮想空間の描画を高速にかつ精度よく行うためには、モデルの正確なシルエットを用いるocclusion cullingアルゴリズムが必須である。本手法では、前処理においてポリゴンモデルを階層的に分割し分割部分を凸包によって近似しておく。walkthrough時にはこれらの凸包のシルエットを用いて隠蔽判定が高速に行われる。実験の結果、少数の頂点からなる凸包によってモデルが精度よく近似され、さらにそれを用いることによってwalkthrough時のフレームレートが大幅に向上することが確かめられた。

# Occlusion Culling Algorithm by Using Hierarchical Convex Simplification of Polygonal Models

Akira Onozawa　　Hitoshi Kitazawa
NTT Lifestyle and Environmental Technology Laboratories

**Abstract**: An occlusion culling algorithm that utilizes hierarchical convex simplification of polygonal models is proposed. Obtaining the silhouette of models precisely is the crucial step in the occlusion culling algorithm. The proposed algorithm in preprocessing provides an approximation of the model's shape as a set of convex hulls. These convex hulls are further partitioned into smaller pieces, providing a few levels of approximations for models. The silhouette of models is approximately generated by projecting the convex hulls. The culling is done by examining the overlap among the silhouettes. The results show that the proposed algorithm provides a good approximation of models by using only a small number of vertices and that walkthrough frame rate is greatly improved.

## 1　はじめに

近年、3次元CG技術による仮想空間を共有することにより種々の通信サービスを提供するアプリケーションが増大している [1, 2]。これらの仮想空間を豊かなものにするためには多数のリアルな物体モデルを用いる必要があり、最近の高速なグラフィックアクセラレータを用いても多大な描画時間を必要とすることがある。仮想空間の構築には種々の方法が考えられるが、本報告では、ポリゴンによって構築されている物体モデルを、3次元的に配置して作るものとする。例えば、建物モデル、アバタ、車モデル等によって構成される街であったり、家具モデル等によって構築される家屋内部等が例である。以下、物体モデルをモデルと略記する。

仮想空間の描画時間を削減するために、カリング *(culling)* と呼ばれる手法が研究されている。カリング手法は、各フレームの描画の直前に視界の外に存在したり他のモデルの影に存在している不可視なモデルを見出して描画の必要なモデルやポリゴンの数を削減する。不可視なモデルを見出すための時間が描画の短縮時間より小さければ、全体として描画にかかる時間が短くなる。仮想空間のカリングは、通常view frustum cullingと呼ばれる方法と、occlusion cullingと呼ばれる方法の2段階からなる。本報告においては、このocclusion cullingに関して新しい手法を提案することを目的とする。

view frustum cullingは、視界（view frustum）の外にあるモデルを見出し、描画時間を削減する。occlusion cullingは、view frustumの中にあって、かつ他のモデルに隠蔽されているモデルを見つけ描画時間をさらに削減する手法である。これは、view frustum cullingより困難な問題であって多くの提

案がされている。一般に、多くのocclusion culling手法は以下の２ステップからなっている。最初のステップでは、他のモデルを隠蔽する可能性の高いoccluderと呼ぶ複数のモデルを選択する。次のステップでは、occluder以外のモデル（occludee）がoccluderのシルエットによって隠蔽されるか否か調べる。以下にoccluderのシルエット生成という観点から従来のカリング手法をまとめる。

多くの手法 [3, 4, 5] は、対象を建築物内部に限定することでシルエット生成を単純化している。文献 [6] においては、空間の階層構造を生成する基本的な考え方が述べられている。しかし、ここではモデルのシルエット生成の方法や実験結果が述べられていない。文献 [7] はoccluderが凸であるか、二つの凸なモデルの和である場合について論じているが、非凸なモデルをoccluderとする方法については述べていない。文献 [8] は、建築物を念頭に上と同様な仮定をしている。文献 [9] では、Hierarchical Occlusion Map (HOM) と呼ばれるtexture mapをoccluderのシルエットに採用する。occluderを一度描画処理し、テキスチャメモリを用いてHOMが生成される。シルエットは正確に求まるが、ハードの利用を前提としている。文献 [10] もハードの利用を前提としている。文献 [11] は動的シーン（モデルが可動）を扱っているが、シルエットを求めることは静的カリング手法の問題として論じていない。

すなわち従来のocclusion cullingにおけるシルエット生成は、粗すぎるかモデルの種類を限定するかあるいはハードの特別な機能を必要としていた。本報告では、任意形状のoccluderのシルエットを高速にかつ精度よく見積れる手法を提案する。提案の手法は、前処理において予めモデルを分割しその結果生じた部分を頂点数の少ない凸包で近似しておくことによって、特別なハードを用いずに精度よくかつ高速にシルエットを求められる。モデルの構造やユーザの視点位置に関しても特別な仮定はしない。また、この凸包はシーンには現れない。実験の結果、本手法によってモデルの精度の良いシルエットが得られ、かつ仮想空間のwalkthrough時の描画時間がview frustum cullingのみの場合に比べて大幅に減少することがわかった。

## 2 モデルの階層的凸近似

occlusion cullingを行うには、occluderモデルのシルエットを正確にかつ高速に求め、他のモデルのシルエットの隠蔽判定をする必要がある。本節では、仮想空間内のモデルのシルエット生成に用いる近似を作成する手法について述べる。

本手法は、モデルを複数の部分に階層的に分割し、各分割部分を近似する凸多面体（凸包）を予め計算しておくというものである。隠蔽判定はこの凸包群を用いて行われる。凸包は必ずしもモデルに内包されるわけではないが、適当数用いるとシーンの不具合（本来可視なモデルが隠蔽される等）は実際にはほとんど生じない。計算幾何学において多面体を凸多面体に分割するアルゴリズムが提案されている [12, 13, 14] が、これらは大量の錐を生成し実用的ではない。また、本報告で扱うモデルはポリゴンの集合であり必ずしも閉多面体ではない。また、この問題はポリゴンモデルの簡単化 [15] やsubdivision [16] の範疇にも無い。文献 [17] は曲率に基づく３次元面の分割手法を提案しているが、分割の結果生じた部分は一般には凸にはならない。

３次元モデルの凸近似問題を以下のように定義する。モデルは、ポリゴンの集合 $S$ で表現されているとする。$S$ の全ポリゴンの全頂点の集合を $V(S)$ と書く。また、$V(S)$ の凸包を $C(V(S))$ と書く。凸包 $C(V(S))$ の頂点（extreme point）の集合は、$X(C(V(S)))$ と書く。$X(C(V(S))) \subseteq V(S)$ なので、モデルを移動や回転しても凸包の頂点座標に関して余計な計算は要らない。以下では、ポリゴン集合 $S$ を分割し、分割の結果生じた部分を凸包で近似することを考える。すなわち、$S$ を適当に $n$ 分割：

$$S = \bigcup_{i=0}^{n-1} S_i,\ s.t.\ S_i \cap S_j = \phi\ (i \neq j), \qquad (1)$$

し、$S$ を $V(S_i)$ の部分集合 $U_i \subseteq V(S_i)$ の凸包 $C(U_i)$ の和（$S$ の"凸近似"と呼ぶ）$\bigcup_{i=0}^{n-1} C(U_i)$ で近似することを考える。$\Pi = (U_0, U_1, \cdots, U_{n-1})$ とおいたとき、$S$ の凸近似を $\Gamma(S, \Pi)$ と書く。

ポリゴンの集合 $S$ とその凸近似 $\Gamma(S, \Pi)$ に対して、"正射影誤差"を以下のように定義する。$S$ の全ポリゴンをxy, yz, zxの各座標面に正射影したときの総シルエット面積を $A(S)$ とする。シルエット面積の近似値は、後述する隠蔽判定で用いるスラブ線を使えば $O(|S|)$ の手間で求められる。同様に、凸近似についてもその凸包のfaceを正射影してシルエット面積をもとめ、$A(\Gamma(S, \Pi))$ とする。ポリゴンの集合 $S$ と、その凸近似 $\Gamma(S, \Pi))$ の"正射影誤差" $E(S, \Gamma(S, \Pi))$ は、

$$E(S, \Gamma(S, \Pi)) = |A(S) - A(\Gamma(S, \Pi))| \qquad (2)$$

と定義される。ちなみに、一般に $N$ 点に対して凸包を作成するには平均で $O(N \log N)$ を要する [12]。

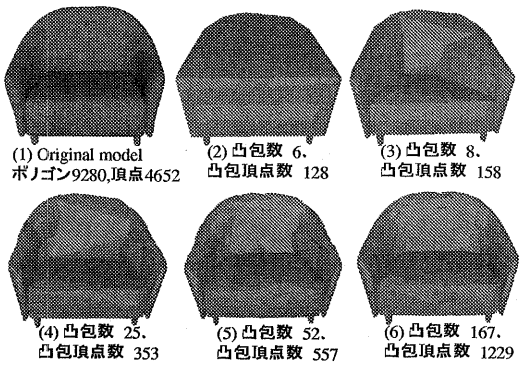


図 1: モデル (Sofa) の凸近似の例

**問題 1** モデルの凸近似問題

正整数 $M$ を、凸包集合の総頂点数の上限値とする。ポリゴンの集合 $S$ で表現されたモデルに対して、以下の条件を満たす $S$ の分割（と分割数 $n$）：

$$S = \bigcup_{i=0}^{n-1} S_i,\ s.t.\ S_i \cap S_j = \phi\ (i \neq j), \qquad (3)$$

を求めよ。

条件：$U_i$ $(\subseteq V(S_i))$ を、$|\bigcup_{i=0}^{n-1} X(C(U_i))| \leq M$ となるように定めたとき、ポリゴン集合 $S$ とその凸近似 $\Gamma(S,\Pi)$ $(\Pi = (U_0, U_1, \cdots, U_{n-1}))$ の正射影誤差 $E(S,\Gamma(S,\Pi))$ が、あらゆる $S_i$、$U_i$ の取り方の中で最小である。

問題 1は、与えられた頂点数の上限値 $M$ の下で、ポリゴン集合 $S$ で表されるモデルの最小正射影誤差の凸近似を求めることを意味する。$M$ を小さくすると、凸近似の頂点数が減り結果的に隠蔽判定が高速になるが、近似の精度は低下する。$M$ を大にすると、精度は向上するが隠蔽判定の時間が増大する。一つのモデルに対して、複数の $M$ の値について精度の異なる凸近似を作成し、$M$ の昇順にレベル番号を付加する。モデルの凸近似の例を図 1に示す。$M$ をモデルの総頂点数以上にすると、分割はポリゴン一つになるまで行われ誤差は 0 になる。

問題 1の最適解を前処理の限られた時間の中で求めるのは難しいので、ここではheuristics を提案する。本アルゴリズムは、ボトムアップなクラスタリングとトップダウンな分割からなる。ボトムアップ処理においては、正射影誤差を考慮しながらポリゴンが階層的にクラスタリングされ、自然にクラスタの木（クラスタ木と呼ぶ）が構成される。トップダウン処理では、クラスタ木を根（モデル全体に対応）から下りながら総頂点数の上限値 $M$ の範囲で正射影誤差が可能な限り小さい分割を探る。クラスタ木を $T$、その根を $r$ とする。$T$ のノードはクラスタに対応し、枝 $(u,v)$ は、$u$ に対応するクラスタが、$v$ に対応するクラスタを内包することを意味している（$u$ が上位）。$T$ のリーフは、ポリゴンに対応する。クラスタ木のノードの集合 $Q$ に対応するポリゴン集合の正射影誤差を $E(Q)$、$Q$ 内の各ノードに対応するポリゴン集合を凸包で近似したときの頂点数の総和を $m(Q)$ とする。あるノード $v$ に対して、$T$ におけるその子の集合を $D(v)$ と書く。ノード $v$ の、凸包を求める対象となる頂点の集合を $U(v)$ と書く。$U(v)$ は、Bounding Sphere 上の一定角度間隔の緯線・経線の各交点に最も近い点を選び生成する。

**Algorithm 1** モデルの凸近似生成

**Input** $S$, $M$.

**Output** $\Gamma(S,(U_0,U_1,\cdots,U_{n-1}))$。

1. Bottom-up clustering phase: クラスタ数が 1 になるまで以下を実行。

step 1 Algorithm 2を用いて、クラスタ $L_i$ $i = 0,1,\cdots,k-1$ を生成。

step 2 各 $V(L_i)$ から、凸包生成の対象となる頂点集合 $U_i$ を決定。

step 3 各 $U_i$ に対して凸包を生成。$T$ 内に対応するノード $v_i$ を生成。

2. Top-down partitioning phase:

step 1 $Q, Q_{min} \leftarrow \{r\}$、$E_{min} \leftarrow E(Q)$。

step 2 $Q'_j \equiv Q \cup D(v_j) - \{v_j\}$ $(\forall v_j \in Q)$。$j_0$ を $min_{\{j|m(Q'_j)\leq M\}} E(Q'_j)$ を実現する $j$ とする。そのような $j_0$ が存在しなければ return$(Q_{min})$。

step 3 $Q \leftarrow Q'_{j_0}$。$E(Q) < E_{min}$ ならば $Q_{min} \leftarrow Q, E_{min} \leftarrow E(Q)$。step 2へ。

以下のアルゴリズムは、現クラスタの集合から新クラスタの集合を生成する。初期クラスタは各ポリゴンである。あるクラスタ $L$ と頂点を共有するクラスタの和（$L$ 自身も含む）を $NE(L)$ と書く。

**Algorithm 2** クラスタリング

**Input** クラスタ集合 $\{L_i \subseteq S | i = 0,1,\cdots,k\}$。

**Output** 新クラスタ集合$\{L'_i \subseteq S | i = 0, 1, \cdots, k'\}$。$(k' < k)$

step 1 各$NE(L_i)$に対応するポリゴン集合の正射影誤差（$E(L_i)$と略記）を計算。

step 2 全ての$L_i$を$E(L_i)$の昇順にソート。

step 3 ソート順に見て他の$NE$と重ならない$NE(L_i)$を新クラスタ$L'_j$として選択。

以下の事実が成り立つ。

**命題 1** *Top-down partitioning*において、$Q' = Q \cup D(v) - \{v\}$ $(\forall v \in Q)$とすると、$m(Q) \leq m(Q')$である。

つまり、Algorithm 1は、分割の進行につれ凸近似の頂点数を（$M$の範囲で）増加しながら、正射影誤差が小さくなる分割を探索している。また、正射影誤差は分割の進行につれ減少する傾向がある。

Algorithm 1の手間は、平均で$O(|S| \log^2 |S|)$であるということができる（証明略）。

## 3 Occlusion Culling

walkthrough時には、view frustum cullingとocclusion cullingがこの順に実行される。view frustum cullingには、文献 [6] に類似の方法を使用しているのでその説明は省略し、モデルの凸近似を用いたocclusion cullingについて以下に説明する。

occluderの選択、凸包の射影と隠蔽判定の方法の詳細については後述する。occludeeに選ばれたモデルのシルエットはBounding boxで近似する。

**Algorithm 3** *Occlusion Culling*

**Input** モデルの集合とそれらの凸近似。

**Output** モデル毎の可視性。

step 1 occluderを選択（後述）。選択できなければレンダリングへ。$L_R \leftarrow 0$。

step 2 occludeeを選択（後述）。occludeeを仮想スクリーンに射影（後述）。

step 3 while $(L_R < L_R^{max})$ do:

1. occluderのレベル$L_R$の凸包を射影。
2. 隠蔽判定（後述）。$L_R++$。

step 1へ行く。

$L_R^{max}$は、occluderの凸近似の最大レベルを表している。この値が高いほど精度の良い隠蔽判定が行われる。これは視点からモデルへの距離でLODのように増減も可能である。

### 3.1 OccluderとOccludeeの選択

カリングを効率的に行うためには、「良い」occluderを効率的に選択することが重要である。これに関しては過去に多くの提案がある [7, 9]。

本手法では、前処理で選んだoccluderの候補から、walkthrough時にview frustum内のoccluderを選択する。一般に、occluderは、表面積が大であるものが望ましい。これを各モデルに対して以下に定義するoccluder度で評価する。ポリゴン集合$S$で表されるモデルのポリゴンの座標面への正射影総面積を$A(S)$[1]、視点からモデル$S$までの距離を$d(S)$、Far cullingの値を$Far$としたとき、walkthrough時のoccluder度$\alpha(S)$は、

$$\alpha(S) = (1 - \frac{d(S)}{Far}) \frac{A(S)}{\max A} \quad (4)$$

となる。occluderは、この値の大きい順に$\alpha$の値が指定値に以下になるまで採用する。これは、sortingを用いなくても簡単なhashingによってモデル数に比例する手間で行える。

選択されたoccluderに対して、occludeeを以下のように選択する。過去これには種々の提案があるが [8, 4, 3] 本報告では、以下のような簡単な方法をとることにする。モデル$u$がoccluderである場合、モデル$v$は、$u$からの距離がfar cullingの以内であって、かつ線分$uv$が視点-$u$間の線分と成す角が一定値以内であれば、occludeeとして選ばれる。以上は、モデル間の相対的位置関係を表すグラフを予め作っておいて行う。詳細はここでは省略する。

### 3.2 スラブ線を用いた隠蔽判定

occluderとoccludeeが選択された後、それらを近似する凸包の集合が、仮想スクリーンに射影される。仮想スクリーンは、実際のスクリーンと同じ位置にあり、同じサイズである。3次元凸包の射影は2次元凸包であるが、本手法ではoccluderを構成する2次元凸包の和が、occludeeの凸包の和を包含するか否か調べる必要がある。ここではこれを近似的にしかし高速に解く方法を提案する。

まず、仮想スクリーン上に等間隔にならぶスラブ線を用意しておく（図 2）。3次元凸包の頂点は、仮想スクリーン上の射影座標を求めたあと、そこに最も近いスラブ線に割り当てられる。スラブ線は、我々の実験においては100本程度で十分であった。仮想スクリーン上に射影された頂点の2次元凸包

[1] Oriented-Bounding-Box (OBB) [18] の面への射影でも可。

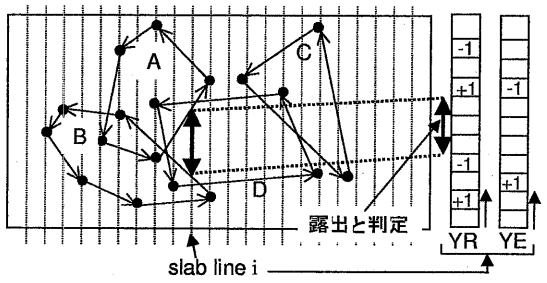


図 2: スラブ線を用いた隠蔽判定

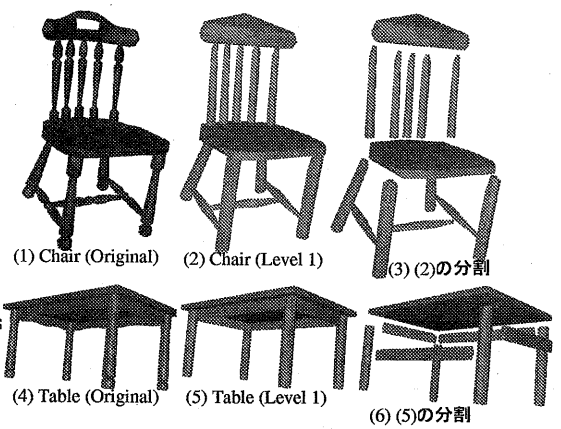


図 3: モデルの凸近似の実験結果

は、詳細は省略するが頂点数を $m$ とすると $O(m)$ で求められる。この2次元凸包には、閉路として半時計回りの向きを与える。

隠蔽判定は、この閉路グラフを用いて概略以下のように行われる。スラブ線を左から右へ順に調べて、配列 $X$ にそのスラブ線に交差する閉路グラフの枝を格納する。各スラブ線においては、$X$ 中の枝のスラブ線での交点を hash した配列 $Y$ を設ける。あとは $Y$ を下から調べることにより occludee の閉路グラフであって、occluder に隠蔽されていない部分があるか調べればよい。図 2に以上の概要を示す。凸包の射影を含めた以上の手間は、枝は $X$ に一度だけ入りかつ枝の数は頂点数に等しいことを考えると、$m$ を射影された頂点数とすると、$O(m)$ であると言える。また、凸包の頂点数はその生成時に定数 $M$ で制限されているので、カリング全体の手間は view frustum 内のモデルの数に比例すると言える。また、以上の手続きに [9] と同様にハードを用いれば、さらなる高速化も期待できる。

## 4 実験結果

以上の occlusion culling アルゴリズムを、ネットワークベース空間シミュレータ WebReality に implement した [1]。WebReality は、ネットワーク経由で取得した VRML 形式のモデルデータをレイアウトすることによって仮想空間を構築しシミュレーションできるシステムである。VC++ により作成し Windows NT PC(single Pentium III(733MHz), ELSA Gloria II) 上で実験を行った。

まず凸近似の結果について示す（図 3及び表 1参照）。凸包頂点数の上限値 $M$ はモデルの総頂点数に応じて適当に指定した。Level 2 の方が 1 よりも $M$ の値を大に指定している。Level 2 の方が、凸包数は増えるが正射影誤差が少し減少していることがわかる。図 3に凸近似の例を示す。これらは、描画はされず形状（シルエット）のみが重要であることに注意する。Level 2 の形状でカリング用には十分である。

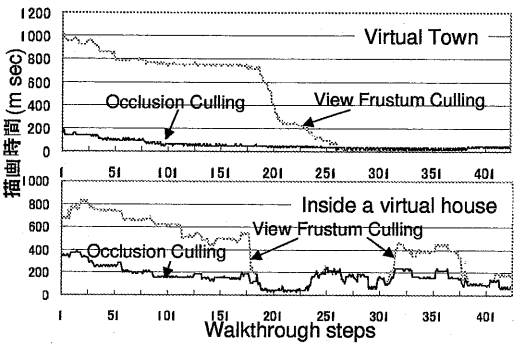


図 4: 仮想空間の描画時間

次に、大規模な仮想空間の walkthrough 時の描画時間を図 4示す。179 モデルを含む街の仮想空間（総ポリゴン数 973,103）と 124 モデルを含む室内仮想空間（総ポリゴン数 629,336）について実験した。walkthrough 経路は仮想空間を一方の端から他方の端まで渡るよう設定し、frame 毎の描画時間を測定した。図 4に view frustum のみの場合も含めて示す。提案の occlusion culling を用いるといずれの場合にも大幅に描画時間が短縮されることがわかる。

表 1: 凸近似の実験結果

| モデル | Original | | Level 1 | | | Level 2 | | | CPU (s) |
|---|---|---|---|---|---|---|---|---|---|
| | ポリゴン数 | 頂点数 | 凸包数 | 凸包頂点数 | 誤差 | 凸包数 | 凸包頂点数 | 誤差 | |
| Table | 3948 | 2080 | 9 | 210 | 0.75% | 17 | 282 | 0.75% | 0.36 |
| Chair | 4617 | 2390 | 14 | 286 | 6.6% | 24 | 396 | 6.4% | 0.40 |
| Sofa | 9280 | 4652 | 8 | 158 | 3.7% | 52 | 557 | 3.2% | 1.8 |

「誤差」は、正射影誤差の、モデルの正射影面積に対する割合。

## 5 おわりに

モデルを凸近似することにより仮想空間の描画時間を削減するocclusion culling手法について述べた。本提案の手法は、前処理時にモデルの凸近似を生成することによりocclusion culling時に必要なシルエットを任意形状のoccluderモデルに対して高速にかつ正確に求めることができる。凸近似は近似の度合いに応じて複数（階層的に）用意される。実験の結果、本手法を用いるとoccluderモデルのシルエットを正確に求めることができるとともに、描画時間が大幅に削減できることが確かめられた。

## 謝辞

日頃ご指導頂くNTT生活環境研究所小倉武部長に感謝する。

## 参考文献

[1] 北澤, 小野澤, and 佐藤. Webreality: ネットワークベース仮想空間レイアウトシステム. In 情処全大, pages 4N–8, 10月 1998.

[2] 一之瀬進. 新しい映像ネットワーク技術とサイバーソサエティ. 映情学誌, 52(12):1756–1761, 1998.

[3] J. Airey, J. H. Rohlf, and F. P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In *Proc. 1990 Symposium on Interactive 3D Graphics*, pages 41–50, 1990.

[4] S. J. Teller and C. H. Séquin. Visibility processing for interactive walkthroughs. In *Proc. SIGGRAPH91*, pages 61–69, 1991.

[5] T. A. Funkhouser, C. H. Séquin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics*, pages 11–20, 1992.

[6] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[7] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Computational Geometry 97*, pages 1–10, 1997.

[8] S. Coorg and S. Teller. Real time occlusion culling for models with large occluders. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[9] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proc. SIGGRAPH97*, pages 77–88, 1997.

[10] N. Greene and M. Kass. Hierarchical z-buffer visibility. In *Proc. SIGGRAPH93*, pages 231–240, 1993.

[11] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *EUROGRAPHICS'96*, 15(3):C–249 – C–258, 1996.

[12] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 1997.

[13] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal of Computing*, 13:488–507, 1984.

[14] B. Chazelle and L. Palios. Triangulating a nonconvex polytope. *Discrete and Computational Geometry*, 5:505–526, 1990.

[15] J. Rossignac and G. Taubin, editors. *3D Geometric Compression*. SIGGRAPH'99 Course Notes 22, 1999.

[16] D. Zorin and P. Schroeder, editors. *Subdivision for Modeling and Animation*. SIGGRAPH'99 Course Notes 37, 1999.

[17] A. P. Mangen and R. T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[18] S. Gottshalk, M. C. Lin, and D.Manocha. OBBTree: A hierarchical structure for rapid interface detection. In *Proc. SIGGRAPH96*, pages 171–180, 1996.