グラフィクスとCAD 96－1
（1999. 8. 23）

Progressive Fan 表現によるLOD

鈴木宏正　岡本 基†　金井 崇‡
東京大学大学院工学系研究科 精密機械工学専攻
〒113-8656 東京都文京区本郷 7-3-1
† 任天堂
‡ 理化学研究所 素形材工学研究室
E-mail: *suzuki@cim.pe.u-tokyo.ac.jp*

三角形メッシュモデルを、特に低価格ハードウェア等で、効率的に描画するためには、グラフィックスパイプラインに発行される頂点数を削減することが重要であり、代表的な方法として、PM 表現 (Progressive Mesh) と頂点共有表現がある。前者は、ポリゴンの可視性に注目して、動的にモデルの詳細度を変更する LOD (Level of Detail) の一種であるが、複雑なデータ構造を必要とする。一方後者は、連続する三角形で頂点を共有するもので、データ構造が単純でまたメモリー効率もよく、広く利用されている。しかし、原理的に 1 枚 1 枚の面の生成削除を行う PM に適用することは難しい。本研究では、頂点共有表現の一つである FAN 構造に対して、PM 的な LOD を適用する Progressive Fan 表現法について提案する。試作システムによって 頂点発行数を 40%程度削減できる LOD ができることを示した。

LOD based on Progressive Fan Representation

Hiromasa Suzuki　Motoi Okamoto†　Takashi Kanai‡
Dept. Precision Engineering, University of Tokyo.
7-3-1, Hongo, Bunkyo, Tokyo 113-8656, Japan.
† Nintendo
‡ Materials Fabrication Laboratory, RIKEN
E-mail: *suzuki@cim.pe.u-tokyo.ac.jp*

For efficient rendering of triangular meshes, particularly on a low level graphics hardware, it is critical to reduce the number of vertices passed to the graphics pipeline. Two types of approaches have been used, PM (Progressive Mesh) and vertex sharing by STRIP and FAN structures. The former is a kind of LOD (level of detail) technique, which can dynamically controls the number of vertices according to the visibility of the model. It, however, requires complex polygonal representation. The latter is a traditional technique for representing polygons in a simple data structure of small amount of memory, but does not easily fit to LOD scheme. This paper proposes a LOD technique named *Progressive Fan* based on FAN data structure. A prototype system is developed to demonstrate that the technique can reduce the number of vertices passed to the pipeline by about 40% in LOD process.

1 はじめに

近年、低価格の 3 次元グラフィックスハードウェアの普及によって、一般のユーザが手軽にリアルタイムの 3 次元 CG を楽しめるようになってきた。しかし低価格ハードウェアの描画性能はまだ低く、複雑なポリゴンデータの表示などでは十分な速度が得られない場合がある。一般的な 3 次元の描画処理は、ジオメトリ処理と、ラスタ処理に分けられる。ジオメトリ処理は CPU が担当し、ラスタ処理は 3D グラフィックスカードが担当する構成が主流である。一般に、三角形メッシュの描画では、一つ一つの三角形の頂点座標をグラフィックスライブラリの API に対して「発行」する。ユーザーの管理するモデルの頂点数が n 個だとしても、発行される頂点数は面数の 3 倍になる。面数は経験的に頂点数の約 2 倍といわれているので、発行頂点数は平均 6n 個にも及ぶ。ジオメトリ処理とラスタ処理のどちらがボトルネックになるかは、実際のハードウェア環境によって変わってくるが、PC のようにジオメトリ処理をソフトで行う場合では、グラフィックスカードのラスター化速度に見合うだけのジオメトリ演算能力を持たない場合が多い。そこで、本研究では、描画の効率化＝発行頂点数の低減と考え、三角形メッシュモデルの

描画の効率化を課題とする。

この発行頂点数の節減のためのよく知られた手法は、OpenGL や Direct3D 等のグラフィックスパッケージで用意されている STRIP や FAN 等の頂点共有表現である (図 1)。ここでは、三角形が連続的に並んでいる場合に、前の三角形の 2 個の頂点と新しい 1 個の頂点の指定で、次の三角形を定義できる。また、メッシュを表現するデータ構造としてもメモリー効率が良いなどの特徴を有する優れた方式であり、拡張も行われている [1, 2]。

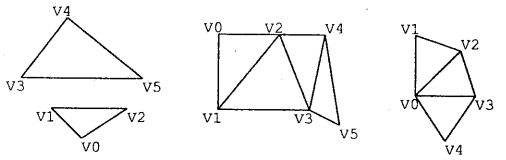

Figure 1: 頂点共有表現

一方、さらにポリゴン数の大きいモデルを表示するには、視覚的にあまり影響しないポリゴンを省略することによって、頂点数そのものの低減することが行われる。このような技術は、LOD (Level of Detail) と呼ばれ、その三角形メッシュに対する代表的なものに、Progressive Mesh (PM) 表現 [3, 4, 6] がある。これは、詳細なモデルを最粗モデルと詳細化情報で表現する手法で、モデルの詳細度 (ポリゴン数) をプログレッシブ (漸進的) に増減させることができる。

しかし、基本的には 1 枚 1 枚の三角形を個別に生成・削除する方式であるため、前述の頂点共有表現には適用することはできない。また、完全な位相データを含む複雑なデータ構造を管理する必要がある [5]。

本研究では、多くの利点を有する頂点共有表現に LOD を適用することを可能にした Progressive Fan (PF) 表現を提案する。これによって、比較的単純なデータ構造での LOD を実現するとともに、効率的な発行頂点数削減を行うことができる。

2 FAN 表現と統合・分割操作

2.1 三角形 FAN

本研究で提案する手法は、FAN という頂点共有表現をベースとしている。図 2 に示すように、三角形 FAN は中心頂点 v_0(FAN 頂点と呼ぶ) の周りに三角形を扇状に並べたものである。最小の FAN は三角形 $\{v_0, v_1, v_2\}$ であり、最大の FAN は v_0 の 1-ring 近傍にある面をすべて接続したものである。ここで、1-ring 近傍とは、ある頂点 v に接続する位相要素全体の集合である。FAN 頂点の 1-ring 近傍の頂点を全部含んでいる FAN を「閉じた」FAN と呼び、そうでない FAN を「開いた」FAN と呼ぶことにする。以下では、1-ring 近傍を簡単に近傍と呼ぶことにする。

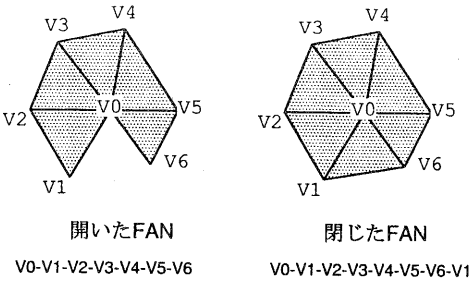

Figure 2: 三角形 FAN

2.2 FAN 統合/分割

PM 表現の稜線消去/頂点分割をベースにした新しいメッシュ操作「FAN 統合/分割」を以下のように定義する (図 2)。

FAN 分割
1 個の FAN 頂点が 2 個の FAN 頂点に分割され、1 つの FAN が 2 つの FAN に増える。

FAN 統合
2 個の FAN 頂点が稜線消去で 1 個の FAN 頂点になり、2 つの FAN が 1 つの FAN に統合される。

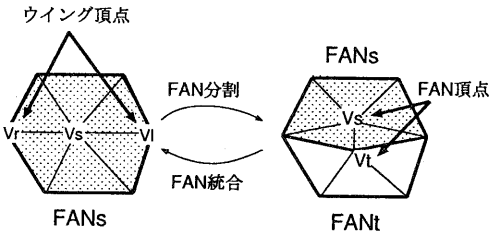

Figure 3: 三角形 FAN 統合/分割

図 3 で、稜線 $\{v_r, v_s\}$ と $\{v_l, v_s\}$ が FAN を分割する切れ目となり、面 $\{v_r, v_s, v_t\}$ と $\{v_l, v_s, v_t\}$ が挿入される。頂点 v_r と v_l は FAN 分割および統合に欠かせない重要な頂点であり、ウイング頂点と名付けて区別する。

2.3 三角形 FAN 統合の位相的条件と特徴

先の定義に基づいて、FAN 統合が可能な位相的条件は次の 2 つである。

1. FAN 統合する 2 つの FAN 頂点は、お互いにそれぞれの近傍になければならない。(当然、2 つの閉じた FAN は統合できない。)
2. FAN 統合する FAN 頂点の近傍にある面は、統合する 2 つの FAN のいずれかに含まれていなければならない (図 4)。

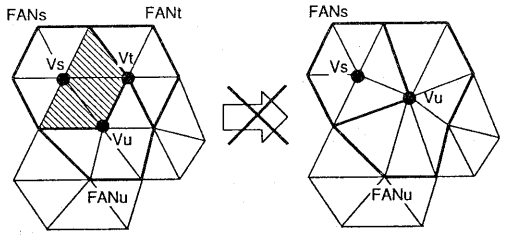

Figure 4: 条件2に反するケース:FAN_t と FAN_u の統合では、FAN 頂点 v_t と v_u のどちらの近傍にも、FAN_t と FAN_u に含まれていない面が存在する (斜線領域)。これを禁止するのは、ここで FAN 統合を行うと、隣接する第3の FAN である FAN_s の面を消去してしまうから。

FAN 統合/分割は、PM 表現の稜線消去/頂点分割と比較してみると、その特徴が浮き彫りになる。頂点分割は新しい頂点1個と新しい面2枚を生成する。しかし操作の影響が周囲に及ぶため、新しく生成された面に隣接する4枚の面の隣接関係、および新しく生成された頂点の周りの面への頂点参照を修正しなければならない。

一方、FAN 分割では、頂点分割同様新しい頂点と面が生成されるが、それは元の FAN の内部にしか影響を与えず、周囲の FAN には影響を及ぼさない。つまり、FAN 統合/分割操作は、周囲の FAN を無視して実行できるという優れた性質を持っている。この性質のおかげで、FAN と FAN の隣接関係の情報を保持する必要が無く、PM 表現に比べ高いメモリ効率性を実現することができる。

3 Progressive Fan 表現

PM 表現と同じように、最粗 FAN メッシュに FAN 分割という詳細化操作を繰り返し適用して、任意の詳細度の FAN メッシュを得られる。最粗 FAN メッシュと、この FAN 分割の列でメッシュを表現するものが、PF 表現である (図5)。一方、PF 表現の生成を生成するには、まず詳細な三角形メッシュを三角形 FAN の集合 (FAN メッシュ) に変換し、初期詳細 FAN メッシュ $\widetilde{FM}(=FM^n)$ を得る。これに n 回の FAN 統合操作を適用して簡略化を行う。

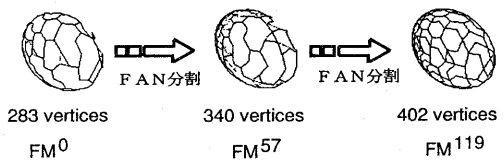

Figure 5: PF 表現の実例 (詳細度レベル数= 120)

3.1 PF 表現の生成

三角形メッシュを入力とし、PF 表現を出力とする手法について説明する。この手法の流れを以下に示す。

1. 初期詳細三角形メッシュを FAN メッシュに変換する (第3.2節)。
2. ユーザが目標の頂点数を指定する。
3. エネルギ関数に基づいて FAN 対を一つずつ統合してゆき、最粗 FAN メッシュを得る。また、統合の過程を履歴として記録する (第3.3節)。
4. 履歴の逆順として FAN 分割シーケンス情報を生成し、最粗 FAN メッシュとともに PF 表現を得る。

3.2 FAN メッシュへの変換

FAN 統合操作には2.3節で示した位相的な制約条件があるため、PM 表現の稜線消去/頂点分割操作と比較すると、簡略化メッシュの取り得る形状の自由度は少ない。このため、最初に三角形メッシュをどのような FAN メッシュに区分するかが、LOD の品質にとって重要である。まず、本研究で用いた FAN の逐次添加アルゴリズムを以下に示す。

すべての頂点を FAN 頂点候補とする。
while FAN 頂点の候補が存在する *do begin*
  FAN 頂点候補から FAN 頂点 v を適当に選択し、その頂点を FAN 頂点候補から除く (図 6a)。
  if v の近傍に既存の FAN 頂点が存在しない
    then 閉じた FAN を形成する (図 6b)。
  else if 近傍に既存の FAN 頂点が存在する
    then 既存の FAN 頂点との間で面の分配を行い、FAN を形成する (4.2.2 節)。2 つの FAN のうちサイズ (内部に含む頂点の個数) の小さい方が面を取る (図 6c)。その後、FAN 頂点候補からウイング頂点 v_r と v_l を除く。
  次の FAN 頂点候補へ
三角形メッシュ上に残る孤立三角形を頂点数3個の FAN に変換する
end

閉じた FAN 同士は FAN 統合できないので、このアルゴリズムでは、閉じた FAN 同士が密集することのないような FAN メッシュを生成する。これについて以下で説明する。

3.2.1 FAN の重なり

三角形メッシュ上に FAN 頂点を添加していき、FAN 頂点は近傍の FAN 頂点に邪魔されない限り、自分の周囲に面を張る。ステップが進み、FAN 頂点が密集するにつれて、FAN 頂点同士が近傍に隣接し、お互いに面を張ろうとするようになる。この時、どちら

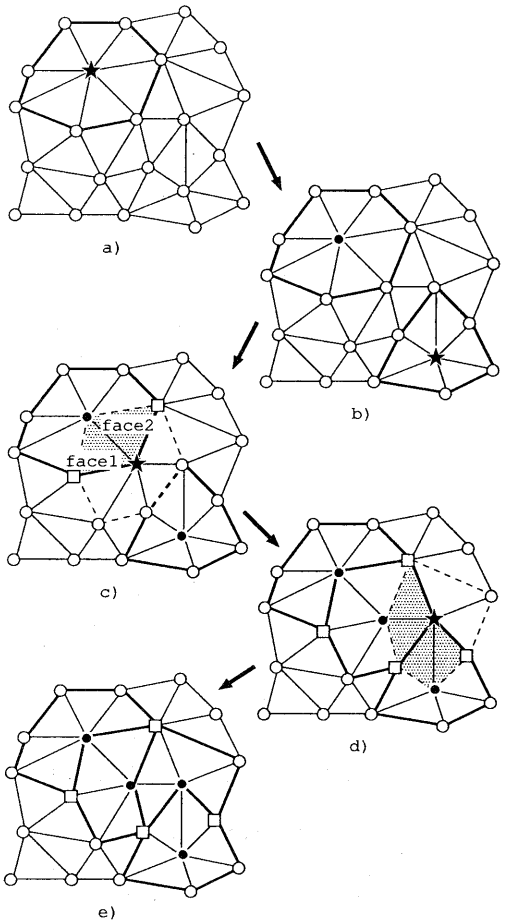

Figure 6: FAN の逐次添加 (a) : 最初の FAN b) : 近傍に FAN 頂点がない場合 c) : 近傍に FAN 頂点がある場合 d), e) : 複数の既存 FAN 頂点と面を取り合う場合。 図の○は FAN 頂点候補、★は追加された FAN 頂点、●は FAN 頂点、□はウイング頂点。

の FAN 頂点が面を張る権利を得るのか決定しなくてはならない。

FAN 頂点と FAN 頂点が取り合う面の数は、2 多様体では 2 枚の場合と 1 枚の場合があるが、本研究のアルゴリズムは以下のようにして必ず 2 枚の面を取り合うようにしている。

図 7 の a) では 2 個の FAN 頂点 v_a と v_b が 2 枚の面を取り合い、v_a と v_c が 1 枚の面を取り合っている (図 7a の点塗りつぶし部分)。この場合、FAN_a と FAN_b は FAN 統合できるが、FAN_a と FAN_c の FAN 統合は許可されない。このように 3 個の FAN 頂点が互いに近傍に存在する状況では、FAN 統合が許可されない FAN 対が生まれてしまう。

このような位置に FAN 頂点を添加するのは効率が悪い。そこで図 7b) のように、FAN 頂点 v_b を添加した時、ウイング頂点 v_r と v_l を FAN 頂点候補から除去して、2 つの FAN 頂点が必ず 2 枚の面を取り合うように保証する。

3.2.2 面の分配

前節の工夫によって、新しい FAN を添加した時に古い FAN と取り合う面の数は 2 個と保証される。すると面の分配のしかたは (対称性を考慮すると)2 通りに限定される (図 8)。描画の効率性という点でいうと、ケース (2) では隣り合う FAN で重複する頂点数が多くなり無駄である。そこでケース (1) の分配に限定する。どちらの FAN に面を与えるかは、FAN のサイズ (内部に含む頂点数) を可能な限り平均化するように決定する。

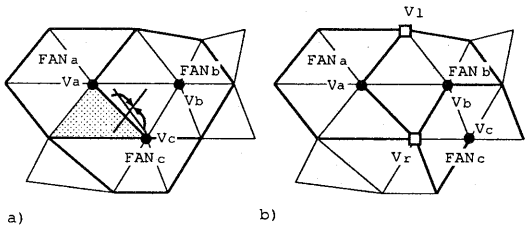

Figure 7: FAN の重なり

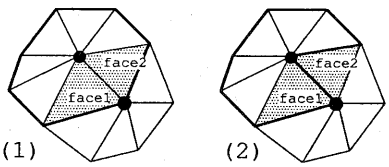

Figure 8: 面の分配

3.3 FAN 統合アルゴリズム

以上で、詳細三角形メッシュに対する FAN 分割が得られたので、次に、FAN を 2 つずつ順番に統合してゆき、その履歴を記録する。統合する順番は、後述するエネルギ関数に基づいて決められる。このようにして、それ以上 FAN の統合ができなくなるか、ユーザーが指定した目標の頂点数に達した時点で簡略化を打ち切り、その時の FAN メッシュを最粗 FAN メッシュとする。FAN 統合の逆操作が FAN 分割なので、この FAN 統合の履歴と最粗 FAN メッシュから PF 表現 (最粗 FAN メッシュ + FAN 分割シーケンス) を作成できる。

3.3.1 FAN 統合順序の決定

統合の順序、すなわち、FAN 対の選択には、初期詳細 FAN メッシュをもとに簡略化 FAN メッシュの精度を表すエネルギー関数 E を定義し、1 回 FAN 統合を実行した時のエネルギ関数の差分 ΔE が最小になる FAN 対を選択し、FAN 統合を実行する。2 個の

FAN 頂点 v_s と v_t が統合されて生まれる新しい FAN 頂点 v_u の座標は、$\{v_s, 1/2v_s + 1/2v_t, v_t\}$ の中から最適なものを選択する。また、FAN 統合の際に三角形 FAN の中の三角形がひっくり返ってしまう可能性があるので、そのような FAN 対の統合は回避する。

FAN メッシュFM のエネルギ関数を $E(FM) = E_{dist}(FM) + E_{spring}(FM)$ と定義する。E_{dist} と E_{spring} のの 2 項は簡略化の過程で FAN メッシュの形状を保存するためのものである。まず、初期詳細 FAN メッシュ $\widehat{FM}$ 上に点群 $\mathbf{X}$ を取る。最小限、$\widehat{FM}$ の各頂点を抽出して点群 $\mathbf{X}$ とする。ユーザが必要ならば、$\widehat{FM}$ 上の点を任意に抽出し、点群 $\mathbf{X}$ に追加する。点群に含まれる各点を $\mathbf{x}_i$ と表す (但し、抽出した点の個数を n として $0 \leq i \leq n-1$)。点 $\mathbf{x}_i$ から簡略化メッシュFM 上へ下ろした垂線の足を点 $\mathbf{b}_i$ とする。$\mathbf{b}_i$ のと点 $\mathbf{x}_i$ との最短距離 $d^2(\mathbf{x}_i, FM)$ を求める。E_{dist} は点群 $\mathbf{X}$ 全体についての距離エネルギ $d^2(\mathbf{x}_i, FM)$ の総和である。

また、バネエネルギ関数 E_{spring} は FAN メッシュの各稜線 $\{j,k\}$ についてのバネエネルギ ($\kappa \parallel \mathbf{v}_j - \mathbf{v}_k \parallel^2$, κ はバネ係数) の総和であり、頂点の移動量を抑えるための関数である。

4 基本データ構造

紙面の都合上詳しい説明は割愛するが、PF 表現の基本となる FAN メッシュ表現のデータ構造を以下に示す。

```
class Vertex {
  Point  point      座標値 (x,y,z)
  Vector normal     法線ベクトル (n_x,n_y,n_z)
}
class ListNode {
  int  vertex       頂点インデクス
  ListNode* next    次のノードへのポインタ
}
class FAN {
  int  center       中心頂点のインデクス
  ListNode* vptrs   頂点の循環リストの開始ノード
  short loop        FAN の開閉を示すフラグ
}
class FanMesh {
  Array<Vertex> vertices   頂点配列
  Array<Fan> fans          FAN 配列
}
```

次に、PF 表現と Fan 分割 (FanSplit) のデータ構造を示す。

```
class PFanMesh
  FanMesh base_fm      最粗 FAN メッシュ FM^0
  Array<FanSplit> fsplits   FAN 分割配列
}
class FanSplit {
  Vector ΔV      差分座標ベクトル (dx,dy,dz)
  Vector ΔN      差分法線ベクトル (dnx,dny,dnz)
  short l,m      分割インデクス
  struct{
    short loop-predict : 2     FAN の開閉の変化
    short g-predict : 2        分割後の FAN 頂点の位置
  } code     2bit フィールド (合計 4bit)
}
```

5 適用例と評価

5.1 例題

以上のアルゴリズムに基づき、システムを試作した。実行環境は、ノート型 PC(CPU:K6-2/233MHz) である。図 5 は、楕円体形状を PF 表現に変換したものであり、その結果を表 1 に示す。ここで、PF 表現は、120 段階の詳細度をもち、孤立三角形が 44 枚発生している。この表には、頂点発行数を、詳細メッシュと最粗メッシュ、そしてその中間のメッシュの 3 つの詳細度に対して示してある。

同様に、ビーナス像の例を図 9 と表 2 に、ブタ形状の例を図 10 と表 3 に示す。

詳細度	頂点数	FAN 数	発行頂点数	孤立三角形
0	283	92	790	44
58	341	150	1080	44
119	402	211	1222	44

Table 1: 楕円体への適用

詳細度	頂点数	FAN 数	発行頂点数	孤立三角形
0	530	270	1764	168
80	610	350	2164	168
181	711	451	2320	168

Table 2: ビーナス像への適用

詳細度	頂点数	FAN 数	発行頂点数	孤立三角形
0	2517	1259	8437	889
441	2958	1700	10642	889
1005	3522	2264	11568	889

Table 3: ブタ形状への適用

モデル	頂点数 (n)	N_0	N_{PF}
楕円体	402	5.97n	2.79n〜3.17n
ビーナス像	711	5.98n	3.33n〜3.54n
ブタ	3522	6.0n	3.28n〜3.35n

Table 4: 頂点発行数 (N_0) は、もとの三角形メッシュの頂点発行数、N_{PF} は、PF 表現の頂点発行数。

5.2 頂点発行数の評価

以上の適用例の結果から、実際にどの程度発行頂点数が削減できたかを評価する。モデルの頂点数 n に対して、描画速度が約 1.5 倍になるレベル、すなわち、発行頂点数が 4.0n 以下になることを効率化の評価の目安とする。結果を表 4 に示す。楕円体の例では、発行頂点数は、頂点数 n に対して 2.79n〜3.17n に留まっており、元の三角形メッシュの発行頂点数は 5.97n であり、47〜53 % の発行頂点を削減できた。約 2 倍の効率化がなされており、大変良好な結果といえる。他の例でも、若干削減の割合は低いものの、

同様の結果が得られている。この楕円体が解くに良かったのは、そのメッシュの規則性にあると思われる。不規則なメッシュよりも規則的なメッシュの方が経験的に孤立三角形の発生数が少なく、結果として発行頂点数が効率的に削減できている。

6 結論と展望

発行頂点数と詳細度表現という2つの軸で評価した結果、PF表現はそれぞれの軸において、頂点共有表現とPM表現ほどの効果は発揮できないが、両方の利点をある程度兼ね備えているという点で、三角形メッシュ表現に新しい選択肢を加えることができたといえる。PF表現の有用性を以下にまとめる。

- PM表現同様、ポリゴン数を制御できるLOD表現が実現できた。
- 頂点共有表現のもつ利点を取り入れ、発行頂点数を大幅に削減できた。

今後の課題としては、以下のものが挙げられる。

- FAN統合/分割の定義を拡張し、操作が可能な位相的制約を緩和して、簡略化のステップ数を上げ、詳細度のレベル数を多くする。
- FAN統合/分割は、稜線消去/頂点分割と比べて簡略化形状の自由度が小さいので、形状の保存性が悪い。
- PF表現は、FAN同士の隣接関係の情報を保持しなくて済むので、PM表現や、Half Edge構造などのB-Repモデルと比較して、データ構造が単純化できる。PM表現はデータ圧縮性をもつことが知られているが、現在、PF表現のメモリ効率について検討を行っており、PM表現とほぼ同等かそれ以上のデータ圧縮性が得られると予想している。また文献[7]との関連性についても検討する必要がある。

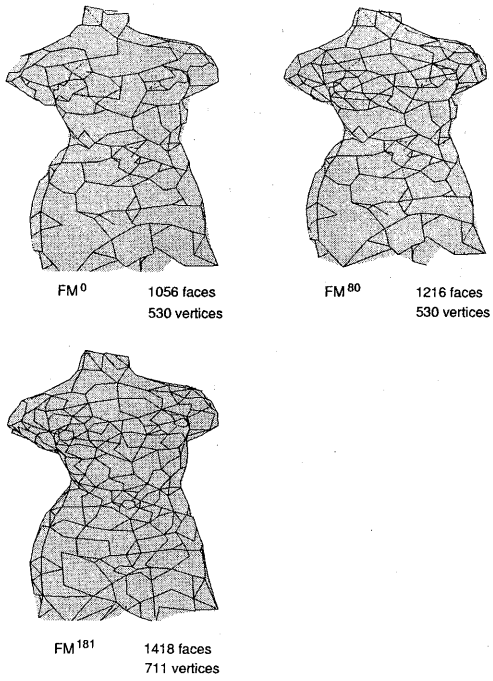

Figure 9: ビーナズ像への適用

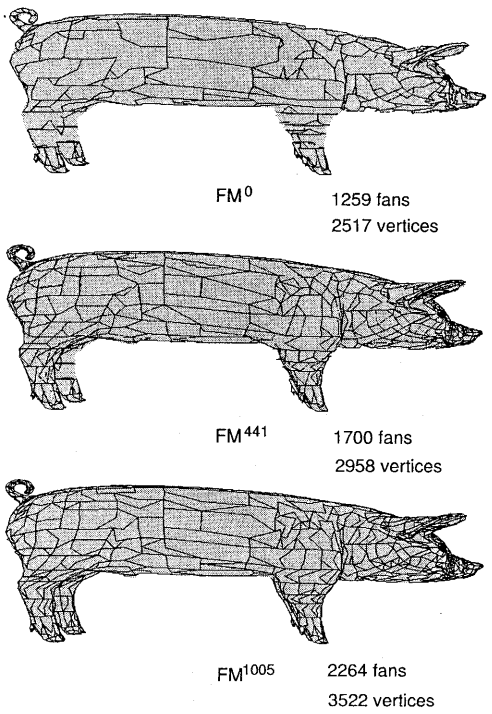

Figure 10: ブタ形状への適用

参考文献

[1] M. Deering. Geometric compression. ACM SIGGRAPH '95, pp. 13-20.

[2] M. Deering. Hardware geometry compression specificaion from java3d. Technical report, Sun Microsystems, 1998.

[3] H. Hoppe. Progressive meshes. ACM SIGGRAPH '96, pp. 99–108.

[4] H. Hoppe. View-dependent refinement of progressive meshes. ACM SIGGRAPH '97, pp. 189–198.

[5] H. Hoppe. Efficinet implementation of progressive mesh. In *Computers&Graphics*, Vol. 22, pp. 27–36, 1998.

[6] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain. In *IEEE Visualization*, pp. 35–42, 1998.

[7] G. Taubin et al. Progressive forest split compression. ACM SIGGRAPH '98, pp. 123–132.