I-015

グラフィックスハードウェアを用いた 境界表現モデルから Octree モデルへの変換

Generation of Octree Model from Boundary Representation Model by Using Graphics Hardware

井口 恒志† 西尾 孝治† 小堀 研一†
Hisashi Iguchi Koji Nishio Ken-ichi Kobori

1. はじめに

境界表現モデルと Octree モデルにはそれぞれ異なった利点と欠点が存在する．両者の相互変換を行うことができれば，互いのモデルの利点を活かすことができる．しかし，境界表現モデルから Octree モデルへの変換の際には，立方体と形状との交差判定，および内外判定を繰り返して行う必要があるため処理コストが大きい．そこで，本研究では GPU を用いた描画処理によって境界表現モデルから Octree モデルへの変換を高速化する手法を提案する．まず，GPU を利用して木構造の末端のノードを作成する．次に，そのノードの親にあたる部分を作成し，画像に置き換えて GPU 内のメモリに保存する．次に，その画像を基に，ボトムアップで木構造を生成する．提案手法では，Octree 構造を GPU が扱いやすい画像に置き換えて生成するため，高速に変換することができる．

2. 従来法

従来法[1]では，形状を構成するすべての三角形面と Octant との交差判定を行い，境界表現モデルから Octree モデルへ変換を行う．従来法の処理手順を以下に示す．まず，形状の表面を表す境界 Octant を生成する．次に，形状の内部に存在する Octant のみを探索し，各 Octant を形状の外部，内部，境界部に分類する．形状の内部と境界部の Octant で形状を表現する．

3. 提案手法

提案手法では，大きく分けて 2 段階の処理で境界表現モデルを Octree モデルに変換する．図 1 に処理の流れを示す．なお，Level.n-1 はリーフの親ノードを表しており，以降 Octree モデルの木構造のリーフにあたる木の深さを Level.n とし，木の階層を上がるごとに Level.n-i と表記するものとする．i はリーフから木の階層を上がった回数である．

第 1 段階の処理では入力である境界表現モデルからボクセルモデルに変換する．なお，ボクセルの大きさが，Octree モデルのリーフに対応する Octant と同様になるように，解像度を設定するものとする．変換法は，中村らの変換手法[2]をもとに改良したものを用いる．そして，先ほど作成されたデータをもとにリーフの親ノードである Level.n-1 の Octant の状態を決定し，図 2 に示すように画像の状態で保存する．第 2 段階の処理では保存したリーフの親ノードの状態をもとに，まとめ上げ処理によって再帰的にルートまでのノードの情報を決定して画像に保存する．なお，これらの処理は全て GPU で行う．この画像を以後 Octree モデル画像と定義する．なお，リーフの状態は親ノードである Level.n-1 の保存位置にテーブル化して保存されるため，最終的に出力される Octree モデル画像には Level.n-1 のデータからルートまでが保存されることになる．

なお，画像の色の要素を表す 1 チャネルには 1 つの Octant データを保存するものとする．一般に画像中の 1 画素は赤，緑，青，アルファである R，G，B，A の 4 チャネルを持つため，1 画素には 4 つの Octant が保存される．以降，図中の画像内の格子で区切られた領域であるセルはそれぞれ 1 チャネルを表すものとする．つまり，同図では横方向の 4 つのセルが本来の 1 画素を表すことになる．なお，画像の座標系を u，v とする．

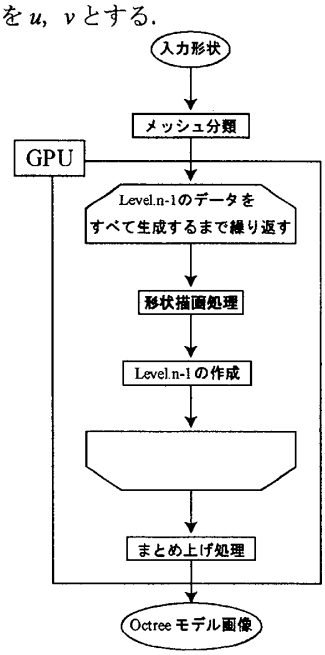

図 1 提案手法の流れ

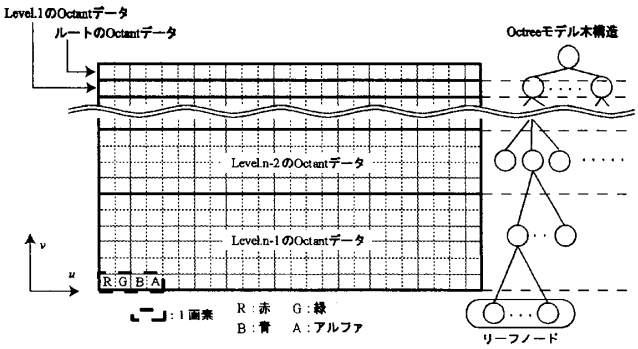

図 2 Octree モデルの画像への保存

3.1 リーフの決定

中村らの手法では，スクリーンを設定して形状を描画すると，そのスクリーンの位置に対応するボクセルデータを

† 大阪工業大学

決定することができる．この手法ではそれぞれのスクリーンの位置で形状全体を描画しなおしているが，提案手法では CPU であらかじめメッシュを分類し，図 3 に示すようにその分類によって描画の必要があるメッシュのみを特定することで，描画コストを軽減している．

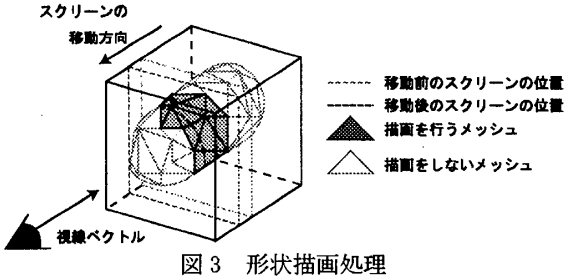

図 3　形状描画処理

以上の処理の結果，生成された Level.n の Octant のうち，Level.n-1 の Octant に対応する 8 つの Octant の状態の組み合わせは 256 通り存在するため，組み合わせを表すインデックス値を画素値とする画像を生成する．

3.2 木構造の生成

前節で生成した Level.n-1 の Octant データから，ルートまでの Octant データをボトムアップで再帰的に生成し，そのつど画像に Octant の状態を保存することで，最終的に画像に木構造を出力する．

Octree 構造では，親ノードの Octant は子ノードの Octant8 つから構成される．よって，親ノードの Octant を作成する際には，Octree モデル画像内の対応する 8 つの子ノードの Octant を参照することで決定できる．このとき，それぞれの Level の Octant は *x,y,z* それぞれの軸方向に順に作成して作成順と同様の順で保存されている．よって，子ノードの Octant は図 4 のように離れた位置に存在する．そこで，まず同図中の破線枠内の 0，および 1 の位置に対応する Octant データを持つ画素を特定する．次に，それらを元に残りの 6 つの Octant を参照する．なお，Level.n-2 以降の Octant は，形状内部と外部が混合している‘Gray’，形状外部のみを表す‘White’，形状内部のみを表す‘Black’の 3 種類の状態で保存する．

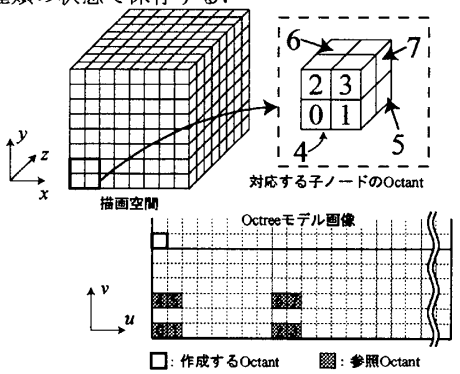

図 4　子ノードの Octant の参照概要

4. 実験と考察

提案手法の有効性を検証するために実験を行った．

この実験では，変換処理の開始から Octree 構造を生成するまでの時間を計測した．CPU に Pentium(R)4 3.2GHz，GPU に GeForce6800 Ultra をもつ計算機を用いた．実験に用いた形状は図 5 に示す 2 形状とし，Level.n は 6 から 9 とした．なお，同図の()内の数字は形状を構成するメッシュ数である．実験結果のグラフを図 6 に示す．なお，処理時間は対数グラフで表した．

dina(1,478,912)　　bunny(144,042)

図 5　実験形状

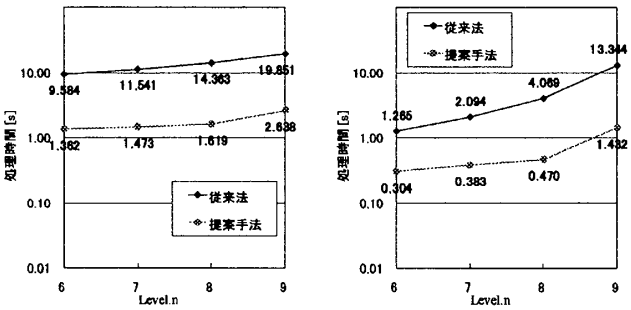

図 6　実験結果

図 6 より，従来法に比べて提案手法が高速に変換できていることがわかる．また実用的と考えられる Level.n が 9 の場合に注目すると，bunny 形状では従来法に比べて処理速度が約 11%になっているのに対し，その約 10 倍の面数である dina では処理速度が約 13%になっている．このことから，提案手法は面数が増加しても効率的に変換できると考えられる．

5. まとめ

本研究は GPU を用いて境界表現モデルから Octree モデルに変換する手法を提案した．提案手法では，中村らの手法を改良することで，Octree モデルの木構造のリーフの状態決定を高速化する手法を提案した．また，GPU 内で繰り返しテクスチャを処理することで，Octree モデルの木構造を画像に置き換えて保存する手法を提案した．これにより，GPU から CPU へ生成途中の木構造のデータを転送する必要が無くなり，高速に木構造を生成することができる．今後の課題として，木構造を画像に保存する際のメモリの使用効率を上げる方法についての検討が挙げられる．

謝辞

本研究を行うにあたり，多大なる助言を頂いた元大阪工業大学大学院，現株式会社東芝の中村徳裕氏に深く感謝する．

参考文献

[1] 小堀研一，石黒和也，久津輪敏郎，”境界表現モデルから Octree 表現への一変換手法”，システム制御情報学会論文誌，Vol.8，No.3，pp.97-105，(1995)

[2] 中村徳裕，井上雄介，西尾孝治，小堀研一，”GPU を用いた境界表現モデルからボクセルモデルへの変換の一手法”，映像情報メディア学会誌,Vol.60,No.10,pp.1624-1627，(2006)