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

A Generation Method from Boundary Representation Model to Binary Voxel Data by Using GPU

学生会員 中村 徳裕[†], 井上 雄介^{††}, 正会員 西尾 孝治^{††}, 正会員 小堀 研一^{††}

Norihiro Nakamura[†], Yusuke Inoue^{††}, Koji Nishio^{††} and Ken-ichi Kobori^{††}

Abstract Boundary representation and spatial partitioning models are widely used in CG. Though boundary representation models require only a small amount of data, manipulating shapes with them is complicated. Spatial partitioning models, on the other hand, have a simple data structure, making shape manipulation easy. However, increasing the precision of the shape model requires ever more data. If these models can be transformed, good use can be made of the advantages of both models. To transform the boundary representation model into the voxel model it is necessary to decide whether each voxel is inside or outside of the shape being manipulated. However, this is a costly process. To solve this problem, we propose a fast method that generates a voxel model from a boundary representation model using GPU. We also describe a method for improving the accuracy of the voxel data.

キーワード: GPU, 形状変換, ボクセルモデル, 境界表現モデル

1. ま え が き

計算機上で形状を表現する際の形状表現法は、大きく分けて境界表現モデルと空間分割モデルに分類できる。

境界表現モデルは平面や曲面の集合で形状を表現するモデルである。このモデルは、数式などで形状を表現するため形状を正確に表現でき、保持するデータ量も比較的少ないという利点がある。しかし、形状編集の際の位相操作が複雑であることや、モデリングの際に頻繁に行われる形状同士の立体集合演算の処理コストが大きいといった欠点がある。一方、空間分割モデルの一種であるボクセルモデルは、立方格子状に分割された3次元空間の各セルの状態が、形状の内部であるか外部であるかを定義することによって形状を表現するモデルである。このモデルは、データ構造が単純で形状操作の際の処理が簡単であり、形状同士の立体集合演算が容易に行えるという利点がある。このことか

ら、近年形状モデラとしても提案されている¹⁾。しかし、形状の表現精度を上げようとする、データ量が膨大になるという問題がある。これらのモデルの欠点を補う方法として、一つのデータ構造で両方のデータを持つ方法がある²⁾³⁾。しかし、この方法でもデータ構造が複雑になることには変わりはない。もう一つの方法として、二つのモデルの相互変換を行うことで、互いのモデルの利点を活かす方法がある⁴⁾。しかし、境界表現モデルからボクセルモデルへの変換の際には、ボクセルの個数の分だけ形状との内外判定を行う必要があるため、処理コストが大きくなる。これを高速化する方法として、形状の境界となるボクセルのみを先に特定し、その後に形状内部を表す部分を探索することによって、内外判定の処理コストを軽減する手法が提案されている⁵⁾。

ところで、近年グラフィックスハードウェア (Graphics Processing Unit, 以下 GPU) を用いることで、さまざまな処理の高速化を図る方法が提案されている^{6)~8)}。GPU とは描画処理に特化したハードウェアであり、近年では、計算処理の高速化や数値計算の計算精度向上だけでなく、GPU の機能の一部を、開発者の要求に応じて自由に変更できるようになってきている。ポリゴンで表現されたモデルからボクセルモデルへの変換に関する手法も提案されているが、これらは主に入力形状の境界部分のみをボクセルに変換する手法を提案しており、形状内部を定義する方法について

2006 年, 情報処理学会第 68 回全国大会で発表

2006 年 3 月 1 日受付, 2006 年 5 月 16 日再受付, 2006 年 6 月 20 日採録

[†]大阪工業大学大学院 情報科学研究科

(〒 573-0196 枚方市北山 1-79-1, TEL 072-866-5184)

^{††}大阪工業大学 情報科学部 情報メディア学科

(〒 573-0196 枚方市北山 1-79-1, TEL 072-866-5184)

[†]Graduate School of Information Science and Technology, Osaka Institute of Technology

(1-79-1, Kitayama, Hirakata, Osaka 573-0196, Japan)

^{††}Department of Information Media, Osaka Institute of Technology

(1-79-1, Kitayama, Hirakata, Osaka 573-0196, Japan)

図 1 従来法の処理の流れ
A flow of conventional method.

は言及していない⁹⁾。また、ポリゴンとボクセルとの最短距離を計算することで変換を行う方法も提案されている¹⁰⁾が、プログラマブルシェード内で距離計算を行っているため、処理に時間がかかるという問題がある。さらに、境界部分をボクセルに変換する手法の中には、境界部分のボクセルを生成した後、形状の内部を塗りつぶすことで形状内部を定義することもできるとしているもの¹¹⁾もある。しかし、変換後の境界部分を表すボクセルの状態によっては、形状内部を定義する際に、ノイズが発生するというロバスト性の問題がある。また、形状内部を定義する方法まで含めた実験結果は、塗りつぶしを用いて内部を定義するのではなく、不等式が満足されるかを調べることで、個々のボクセルの内外の判定を行う陰関数表現に関するものしか議論されておらず、一般的なポリゴンで表現されたモデルでの処理時間は議論されていない。

そこで本研究では、GPUを用いることによって、ポリゴンで表現された境界表現モデルから、形状内部が定義されたボクセルモデルへの変換を高速化する手法を提案する。提案手法では、ポリゴンで表現された境界表現モデルを入力形状とする。まず、設定した視点で入力形状を描画する。そして、画面に描画されたピクセルデータから特定のカラー成分を読み込むことで、その位置に対応したボクセルデータを生成する。これをボクセル空間の一辺の解像度分だけ繰り返し行うことで、高速にボクセルデータを生成する。また、入力形状を高解像度でラスタライズすることにより、結果のボクセルデータの精度を向上する手法についても提案する。

2. 従来法

ポリゴンで表現された境界表現モデルから空間分割モデルを生成する従来法として文献5)で提案されている手法がある。文献5)はOctreeモデルに変換する手法であるが、ボクセル変換する場合も、ほとんど同様の処理で変換することができる。文献5)と同様に、形状境界部分のボクセルをあらかじめ生成し、その後に形状の内部を表すボクセルを塗りつぶす方法である。従来法の処理の流れを図1に示す。

- (1) 形状を構成しているポリゴンとボクセルとの交差判定を行い、図2のような境界ボクセルを生成する。

図 2 境界ボクセル
Boundary voxels.

図 3 提案手法の概念
Proposed method.

- (2) (1)で生成された境界ボクセルの内部を塗りつぶすため、ボクセル空間の各セルの状態を形状の内部、外部、境界部に分類する。
- (3) 形状の内部ボクセルと境界ボクセルを、形状を表現しているボクセルとする。

3. 提案手法

3.1 提案手法の概要

提案手法では、図3のようにスクリーンを設定して平行投影で形状を描画する。なお提案手法では、初期のクリッピングボリュームは入力形状のバウンディングボックスの最長辺を一辺の長さとする立方体とし、入力形状は初期のクリッピングボリュームの重心と入力形状のバウンディングボックスの重心が重なるように配置するものとする。また、スクリーンの位置とクリッピングボリュームのNearクリッピング面の位置は、常に同じになるように定義するものとし、スクリーンの移動とともにNearクリッピング面も再定義される。Farクリッピング面は変換処理が終了するまで再定義されない。つまり、常にスクリーン前方にある形状のみが描画されることになる。なお、一般にスクリーン、つまりNearクリッピング面とポリゴンが交差する場合には、ポリゴンのうちスクリーンの前方にある部分が描画されることになる。スクリーン上の画素が形状の外部にある場合、その画素の位置から見えるのは形状の表面のみである。また、スクリーン上の画素が形状の内部にある場合、その画素から見えるのは形状の裏面のみである。形状を描画する際には、個々のピクセルから視線と交差するポリゴンを探して描画するため、形状の表面と裏面を異な

図 4 提案手法の処理の流れ
General flow of the method.

るカラー値で描画しておき、個々のピクセルが表面と裏面のどちらのカラー値になっているかを調べることで、そのピクセルの位置が形状の内部か外部かを調べることができる。これは、ピクセルの中心をボクセルの中心と考えた場合、個々のボクセルが形状の内部に入っているかどうかを調べることと同じ意味を持つ。このカラー値を取得することで、現在のスクリーン位置に対応するボクセルデータを生成することができる。これにより、従来法のように境界部分のみを変換する必要なく、一度に形状内部まで決定することができる。この処理を、同図のスクリーンを矢印の方向に移動させながら繰り返し行うことで、形状をボクセルデータに変換する。

GPU は、入力された形状データを画面に表示できるように離散化し、形状データの色情報から各画素のカラー値を決定する。GPU はこの処理に特化した演算ユニットを持つため、同様の処理を CPU で計算するよりも高速に行うことができる。提案手法の処理の流れを図 4 に示す。

提案手法の処理手順を以下に示す。なお、以降カラー値は (R,G,B) の形式で表記する。また、括弧内の R,G,B をそれぞれカラー成分とする。

- (1) 描画領域を変換後のボクセル空間の解像度と同じ解像度に設定する。
- (2) 入力形状の表面のカラー値を赤 (1.0,0,0)、裏面を青 (0,0,1.0) に設定し、初期の視点を決定する。
- (3) スクリーンを移動した後、入力形状を描画する。
- (4) 図 3 のように、移動後のスクリーンの位置で画面に描画されたピクセルデータから、カラー値を取得する。読むカラー成分は、同図に示すように青成分 (B) とする。GPU から読込んだ青成分のデータを直接計算機上のメインメモリー内に書き込むことで、その位置に対応するボクセルデータを生成する。

図 5 スクリーンの位置
Screen position.

図 6 スクリーンの位置と描画結果の関係
Relation between screen position and drawing result.

- (5) ボクセルデータがすべて生成されるまで (3), (4) を繰り返す。

3.2 スクリーンの設定

図 5 に示すように、描画領域を Z 軸に対して垂直に分割する。このときの分割数は、ボクセル領域の一辺の分割数と同じとする。そして、描画領域を分割している各分割線 (以下、プレーン) の位置を描画の際のスクリーンの位置とする。このとき、隣り合うプレーン間の距離は、ボクセルの一辺の長さと同じとなる。例えば、ボクセル領域を $256 \times 256 \times 256$ に設定した場合、描画領域を Z 軸に対して垂直に 256 分割することになる。

Z 軸の負の方向に対して、プレーン単位でスクリーンを移動し、その都度平行投影で形状を描画する。その際、プレーン毎にカラー値取得の処理を行う。

3.3 カラー値の取得

前節で設定した各プレーンで入力形状を描画し、画面に描画されたピクセルデータを読み込む。あるプレーンから形状を描画したときの描画面面の一例を図 6 に示す。

提案手法では、裏面を (0,0,1.0) に設定しているため、図 6(a) の形状に対して、同図 (b) の太線で示す部分が同図 (c) のように (0,0,1.0) で描画される。つまり、(0,0,1.0) で描画されるピクセルの位置は形状の内部ということになる。ここで、表面は (1.0,0,0) で描画されているため、表面を表す

図 7 ボクセルデータの生成
Generation of binary voxel data.

ピクセルの青成分は 0 となる。そのため、現在のスクリーンの位置の Z 座標を Z_i とすると、図 7 のように描画結果のピクセルデータから青成分のみを取得したピクセルデータは、現在の Z 座標でのボクセルデータと考えることができる。これを各プレーンに対して繰り返すことで、ボクセルデータに変換することができる。

3.4 変換精度の向上

近年、GPU の性能の向上により、描画処理の一部を開発者の要求に応じて変更できるプログラマブルシェーダといわれる機能が登場している¹²⁾¹³⁾。以下では、この機能を用いることで、変換後のボクセル形状の誤差を抑制するための手法も提案する。

GPU でスクリーンに出力するピクセルデータを生成する際には、画素の中心を基準として形状をラスタライズする。そこで、GPU 内部で解像度を上げてラスタライズし、個々のボクセルの状態を決定するためのサンプリング点数を増やすことで、出力されるボクセルデータの精度を上げる。以降の説明では、GPU 内部で処理する際の解像度を、要求されている解像度の 2 倍にする場合を例にあげて説明する。なお、最も視点に近いプレーンを 1 番目のプレーンとしたときの奇数番目のプレーンを奇数プレーン、偶数番目のプレーンを偶数プレーンとする。

まず、入力形状を要求されている解像度の 2 倍の解像度でラスタライズする。プレーンの数も 2 倍に設定し、偶数プレーンと奇数プレーンの 2 枚のプレーンで、ある Z 座標でのボクセルデータが決定されることになる。このとき、図 8 の破線で示す奇数プレーンでは、入力形状の表面と裏面を 3.1 節で説明したカラー値に設定して描画する。その描画結果を GPU 内のメモリーに残したまま、一点鎖線で示す偶数プレーンで形状を重ねて描く。なお、このときの偶数プレーンでの描画結果は、GPU 内に保存しておいた奇数プレーンの描画結果に出力するものとする。このとき、同一のカラー値を設定して描画すると、奇数プレーンでの描画結果が上書きされてしまうので、青成分にカラーマスクを設定して書き込みが行われないようにした後、形状の裏面のカラー値を緑 (0,1.0,0) に設定して描画する。これにより、同図の閾値処理の部分に示すように、二つのプレーンでの描画結果をひとつにまとめた結果が描画されている

図 8 精度向上処理
Improvement of accuracy of voxel data.

図 9 閾値処理の例
An example of decision of the voxel.

ことになる。この描画結果のデータに対して閾値処理を行い、最終的な出力データを生成する。ここまでの処理は、スクリーンの移動を除いて、すべて GPU 内で行われ、GPU 外のメモリーへのデータ転送は行われない。以降、解像度を上げて生成された、二つのプレーンでの描画結果をひとつにまとめたバッファを一時記憶バッファ、要求されている解像度と同一の解像度で、閾値処理後の結果データを保存するバッファを、最終出力バッファとして説明する。なお、これらのバッファはどちらも GPU 内に作られる。同図の太枠内に示す 4 ピクセルは、最終出力バッファの 1 ピクセルに対応するので、元の解像度で一つのボクセルと考えることができる。この四つのピクセルの青成分と緑成分を調べ、それぞれの成分の値を合計する。この値が、太枠内の個々のピクセルのうち形状内部と判定されたピクセル数となり、最大で 8.0 となる。この合計値が閾値以上の場合、元の解像度の対応するピクセルは内部ボクセルを表すこととする。この閾値処理を、最終出力バッファのそれぞれのピクセルのカラー値がすべて決定されるまで繰り返すことで、ある Z 座標でのボクセルデータが決定される。この閾値処理は、ピクセルシェーダを用いて GPU 内で行われる。図 9 に閾値処理の一例を示す。

同図のように、元の解像度の 1 ピクセルに対応する一時記憶バッファの 4 ピクセルの青成分と緑成分のカラー値を合計する。同図では合計値が 5.0 となっており、閾値を最大値の半分の 4.0 に設定した場合、元の解像度の対応する

図 10 精度向上処理の流れ
A flow of improving accuracy of voxel data.

ピクセルには (0,0,1.0) が設定される。この閾値処理の後、3.3 節と同様の処理によって最終出力バッファをメインメモリに出力する。このときのみ、GPU 外のメモリにデータ転送が行われる。

奇数プレーンでの描画からカラー値の取得までの一連の処理を、3.2 節と同様に解像度と同じ回数だけ繰り返すことで、変換後のボクセルデータを生成する。以上の精度向上処理の流れを図 10 に示す。

これにより、変換の際に倍の解像度で形状が評価されるため、精度を向上させることができる。なお、説明の簡単のため形状を描画する空間を 2 倍に設定しているが、GPU で処理する場合は、アルファ値を用いることで 3 倍までは上記の手順を変えることなく処理が可能である。それ以上の倍率でも、3 枚のプレーン进行处理するごとに一時的な閾値処理の結果を保存するバッファを用意することで、同様に処理が可能である。

4. 実験と考察

4.1 変換速度の比較

提案手法の有効性を検証するため実験を行った。この実験では、変換処理の開始からすべてのボクセルデータが生成されるまでの処理速度を計測した。従来法には、2 章で説明した手法を用いた。実験には、CPU に Pentium(R)4

図 11 実験形状
Experimental shapes.

図 12 レベルと変換速度の関係
Relation of the level to the generation cost.

3.2GHz, GPU に GeForce6800 Ultra をもつ計算機を用いた。実験に用いたポリゴンで表現された形状は図 11 に示す 4 形状とし、今回の実験でボクセルの解像度であるレベルはレベル 7 から 9 とした。また、精度向上処理の際の解像度の倍率は 2 倍とし、閾値は 4 とした。

なお、レベルとはボクセル空間の一辺の分割数を 2^n としたときの n の値である。また、() 内の数字はポリゴン数である。実験結果を図 12 に示す。処理時間は対数グラフとなっている。

図 12(a) から (d) のいずれにおいても、レベルが高くなるにつれて、提案手法の効果が顕著に出ていることがわかる。レベル 9 に注目した場合、精度向上処理を行った場合でも、従来法に比べて処理時間が約 20~35 % になっていること

表 1 処理の内訳
Classification of the process.

	カラー値の取得	閾値処理	形状描画処理
レベル 7	4.71	4.83	90.46
レベル 8	12.31	10.06	77.63
レベル 9	28.40	21.21	50.39

単位: %

がわかる。また、形状を構成する面数が増加しても、従来法に比べて高速に処理できていることもわかる。このことから、提案手法の有効性を確認することができた。さらに、図 11 の形状の中で、最も面数の多い Apple 形状について、精度向上を行った時の処理時間の内訳を調べた。結果を表 1 に示す。なお、表中の数値は全体の処理時間を 100 % としたときの割合である。

表 1 より、レベル 9 に変換した際でも、全体の処理時間の 50 % 以上を形状の描画が占めていることがわかる。提案手法では、精度向上処理を除き形状を一度だけ描画する時間が、そのままある Z 座標でのボクセルデータを生成する処理時間となる。また提案手法では、スクリーンが増加するという事実と、解像度が大きくなるということは同じ意味であるため、提案手法の処理時間は解像度に依存すると考えられる。また、今回の実験結果は、CPU の性能に依存しない形でアルゴリズムの性能を評価するという観点から、CPU でスクリーン後方にあるポリゴンを分類し、省略して描画するなどの方法はとっていない。しかし、CPU の性能しだいでは、スクリーンの後方に存在するポリゴンは、描画しないようにする方法も考えられる。この場合、描画に必要なポリゴン数は段階的に減少していくため、形状を構成するポリゴンの状態にもよるが、処理全体におけるポリゴンの描画枚数に関する効率は、ある程度向上すると考えられる。それでも個々のポリゴンは、複数回描画する必要があるが、一般に用いられている GPU の描画性能は高く、形状の描画は GPU の本来の用途であるため、今後も GPU の性能の中でも描画性能の向上が著しいと考えられる。そのため、パーソナルコンピュータで扱える解像度のボクセルデータに変換する際には、特に問題にならないと思われる。さらに、最も GPU で最適化されている処理は、形状描画であると考えられるため、提案手法は、GPU の性能の向上の恩恵を受けやすいという点で、GPU の利用に最適化されたアルゴリズムであると考えられる。なお提案手法では、スクリーンの移動に伴うクリッピングボリュームの再定義と、最終出力バッファからの閾値処理後のカラー値の取得処理以外の処理は、すべて GPU 内で行われる。また、閾値処理はピクセルシェードで行われる。そのため、カラー値の取得は、GPU から CPU へデータを転送する際の転送帯域に依存しており、閾値処理は GPU のプログラマブルシェードに関する処理速度に依存していると考えられる。

4.2 精度の検証

精度向上処理の効果を検証するための実験を行った。精

度向上を行わない場合のボクセルデータと、精度向上処理後のボクセルデータに対して、入力形状との誤差を比較した。なお、変換後のボクセルデータのうち、形状の境界を構成する個々のボクセルの重心から入力形状の表面までの最短距離を誤差とした。ここでの形状の境界を構成するボクセルとは、形状内部ボクセルのうち、そのボクセルを構成する 6 面に接するいずれかのボクセルが形状外部ボクセルを示すものとする。これは、変換後のボクセルデータの表面を構成するボクセルが、入力形状の境界を表しているほど誤差が小さいと考えるためである。すべての境界ボクセルに対して誤差を調べ、その中の最大誤差 $MaxError$ 、および平均誤差 $AveError$ を求めた。誤差の算出式を式 (1) に示す。

$$MaxError = \max_{voxel \in V} (MinDist(voxel))$$

$$AveError = \frac{1}{N} \sum_{voxel \in V} (MinDist(voxel))$$

$$MinDist(voxel) = \min_{face \in F} (Dist(face, voxel)) \quad (1)$$

ここで、 V は形状の境界を構成するボクセルの重心の座標の集合を表し、 F は入力形状を構成する面の集合を表す。また、 N は集合 V の要素数である。 $Dist(face, voxel)$ は面 $face$ とボクセルの重心の座標 $voxel$ との最短距離を求める関数であり、 $MinDist(voxel)$ は入力形状を構成する個々の面とボクセルの重心の座標 $voxel$ とのそれぞれの最短距離のうち、最小のものを求める関数である。なお、ここでの誤差の単位は、1 ボクセルの一辺の長さを 1 としたときの距離である。実験には図 11 の形状をレベル 8 で変換した結果を用いた。また、4.1 節と同様に精度向上処理の際の解像度の倍率は 2 倍とし、閾値は 4 とした。その結果を図 13 に示す。

図 13 より、平均誤差が約 0.05~0.45 改善されていることがわかる。また、すべての形状において最大誤差も改善されている。同図より、精度向上処理は Hourglass や Chair などの複雑な凹凸を持つ形状の平均誤差に、より効果が現れている。これは、複雑な凹凸や面同士の角度の急峻な部分が多いほど、画素の中心のみをサンプリングした結果と入力形状との誤差が大きくなるためであると考えられる。

5. む す び

本稿では、GPU を用いてポリゴンで構成された境界表現モデルからボクセルモデルへの変換を高速化するための手法を提案した。提案手法では、変換元形状の表面と裏面に異なるカラー値を設定して描画し、描画結果から特定のカラー成分のみを取得することで、変換処理を実現した。提案手法は 2 次元での処理を基準としているため、GPU 内の処理の過程で必要とするビデオメモリーが少なくすむ。そのため、高解像度のボクセルデータでも適用することができる。また、GPU 内で従来の描画処理とプログラマブ

図 13 精度の比較

Comparison of accuracy of generated voxel data.

ルシェーダを用いることで、解像度を上げてサンプリングすることで変換精度を向上させる手法も提案した。提案手法で利用している描画処理は、GPU の本来の用途であるため、急速に発展している GPU の高性能化の恩恵を受けやすいと考える。今後の課題として、精度向上処理でサンプリング点数を増加させたときの、入力形状の複雑さと精度向上の度合い、および処理時間の関係からの最適なサンプリング点数の決定法や精度向上処理の高速化などが挙げられる。

【文 献】

- 1) Kase K., Teshima Y., Usami S., Ohmori H., Teodosiu C. and Makinouchi A.: "Volume CAD", Volume Graphics 2003 Eurographics / IEEE TCVG Workshop Proceedings, I. Fujishiro, K. Mueller, A. Kaufman (eds.) in cooperation with ACM SIGGRAPH, Tokyo, pp.145-150, pp.173(2003)
- 2) 米川和利, 小堀研一, 久津輪敏郎: "空間分割モデルを用いた形状モデラ", 情処学論, **37**, 1, pp.60-69(1996)
- 3) P. Brunet, and I. Navazo: "Solid Representation and Operation Using Extended Octree", ACM Transaction on Graphics, **9**, 2, pp.171-197(1990)
- 4) 中村徳裕, 西尾孝治, 小堀研一: "2 値ボクセルからの境界表現モデルの自動生成に関する一手法", 映情学誌, **59**, 10, pp.1445-1453(2005)
- 5) 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論, **8**, 3, pp.97-105(1995)
- 6) Onoue K., and Nishita T.: "Virtual Sandbox", Proceedings of IEEE 2003 Pacific Conference on Computer Graphics and Applications, pp. 252-259(Oct. 2003)
- 7) 山地秀美, 新藤義昭: "Z バッファ法を利用したオブジェクトおよび衝突検出", 情処学論, **43**, 6, pp.1899-1909(2002)
- 8) 山本修身: "グラフィックスハードウェアを用いた Delaunay 三角形分割の高速計算", 日本応用数理学論, **14**, 4, pp.235-266(2004)
- 9) Z. Fan, W. Li, X. Wei, and A. Kaufman: "GPU-based Voxelization and its Application in Flow Modeling", ACM Workshop on General-Purpose Computing on Graphics Processors(2004) <http://www.cs.sunysb.edu/~vislab/projects/gpgpu/Jellyfish.pdf>
- 10) H. Hsieh, Y. Lai, W. Tai, and S. Chang: "A Flexible 3D Slicer for Voxelization Using Graphics Hardware", Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp.285-288(2005)
- 11) Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng: "Real-time

Voxelization for Complex Polygonal Models", Computer Graphics and Applications, 12th Pacific Conference on (PG'04), p.43-50(2004)

- 12) F. Randima and J.K. Mark: "The Cg Tutorial 日本語版 - プログラム可能なリアルタイムグラフィックス完全ガイド", 株式会社ボーナデジタル, pp.61-69, pp.194-196(2003)
- 13) CgUsersManual: http://developer.nvidia.com/object/cg_toolkit.html

なかむら のりひろ
中村 徳裕 2004 年, 大阪工業大学大学院博士前期課程修了。現在, 同大学院博士後期課程にて CG, CAD に関する研究に従事。学生会員。

いのうえ ゆうすけ
井上 雄介 2006 年, 大阪工業大学情報科学部情報メディア学科卒業。CG, CAD に関する研究に従事。

にしお こうじ
西尾 孝治 2000 年, 大阪工業大学大学院博士後期課程修了。2000 年, 豊橋技術科学大学助手を経て, 2002 年, 大阪工業大学情報科学部情報メディア学科専任講師。3 次元画像処理に関する研究に従事。博士 (工学)。正会員。

こぼり けんいち
小堀 研一 1975 年, 山梨大学大学院修士課程修了。1975 年, シャープ (株) 入社。以後, 一貫して CAD, CAM, CG に関する研究開発に従事。1991 年, 大阪工業大学に奉職。現在, 同大学情報科学部情報メディア学科教授。CAD, CG, バーチャルリアリティの研究に従事。工学博士。正会員。