

## 4次元メッシュモデル生成のための3Dアプリケーションからの時系列ボクセルモデル高速生成手法

北海道大学 ○加藤 航平, 小野里 雅彦, 伊達 宏昭, 田中 文基

### 要 旨

本報では, 3DアプリケーションのOpenGL 描画命令をフレームごとに取得し, その命令から時系列3Dボクセルモデルを高速に生成する手法を報告する. 生成した時系列ボクセルモデルは, 4次元メッシュモデル生成に利用される.

### 1. はじめに

サイバーフィールドは, さまざまな実世界の対象をコンピュータ上でモデル化した仮想空間である. サイバーフィールドでは, 3次元形状モデルの動的な状態を4次元時空間モデルで表現する. 本研究では4D-Marching Cubes法による4次元メッシュモデル生成[1]のための3次元ボクセルモデル時系列データ生成の一つの方法として, OpenGL 3Dアプリケーションの動的なアニメーション(以下, OpenGLアニメーション)からボクセルモデル時系列データを生成することを目的とする. 前報[2]では, OpenGLの描画命令からのメッシュモデル生成時にOpenGLアニメーションのフレームレートが低下する, ボクセル生成に時間がかかる, という問題があった.

本報告では高速なボクセル生成手法の実装と, ボクセル生成処理に伴うOpenGLアニメーションのパフォーマンス低下を回避するためにOpenGLの描画命令を他PCに送信し, そこでボクセルモデル生成を行う手法を提案し, 前報[2]との比較を行った.

### 2. ボクセルモデル時系列データ生成手法

#### 2.1 本手法の概要

本手法の概要を図1に示す. まず, OpenGLアニメーションの各フレームからOpenGLの描画命令を取得する. 取得したOpenGL描画命令からボクセルモデルを生成する. この処理により生成されたボクセルモデル時系列データは4次元メッシュモデル生成に使われる. OpenGLの描画命令を取得するために, OpenGL Interceptor[3]を利用した. このツールはDLLプロキシツールであり, OpenGL 3DアプリケーションとOpenGLライブラリ(OpenGL32.dll)の間に割りこんで描画命令を取得する. この描画命令を再度レンダリングに用いることでボクセルモデルを生成する.

#### 2.2 ボクセルモデル生成手法[4]

ボクセルモデルの生成にはEisemannの手法[4]を用いた. この手法は, 図2に示すように, テクスチャメモリの1ピクセルをボクセルのxy領域に対応させ, 各ピクセルのRGBAそれぞれのビット値をボクセルのZ方向の値として扱う. すなわち, ビット値が1であればモデルの内部, 0であればモデルの外部となる. テクスチャメモリを用いたボクセルモデル生成手法を図3に示す. まず, メッシュを構成する三角形をFrame Buffer Object (FBO)を用いてテクスチャメモリにレンダリングを行う. その際, 三角形のデプスに対応するビットよりも手前にあるビットを1にしたカラー値としてピクセルの色を決定する. 次に, 決定されたカラー値と, 今まで処理してきたテクスチャメモリのカラー値のビットとのXOR演算を行う.

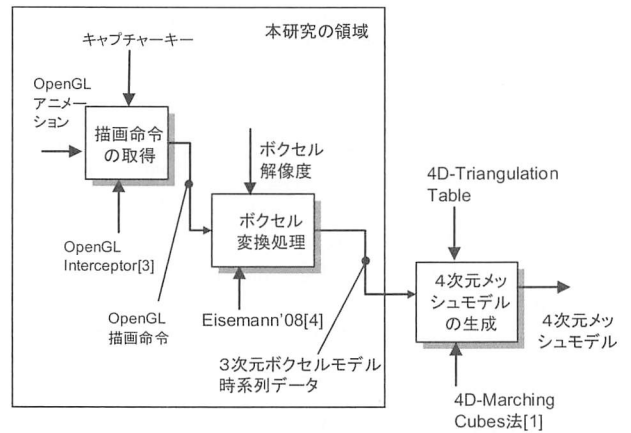


図1 ボクセルモデル時系列データ生成手法の概要

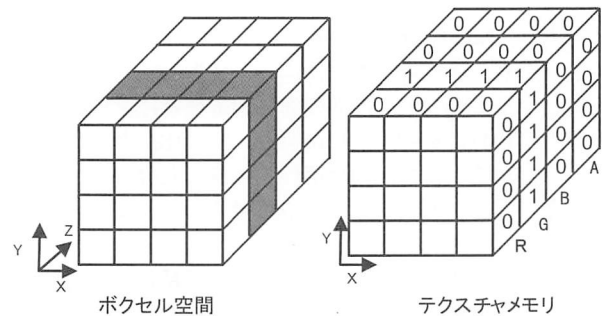


図2 テクスチャメモリによるボクセル表現

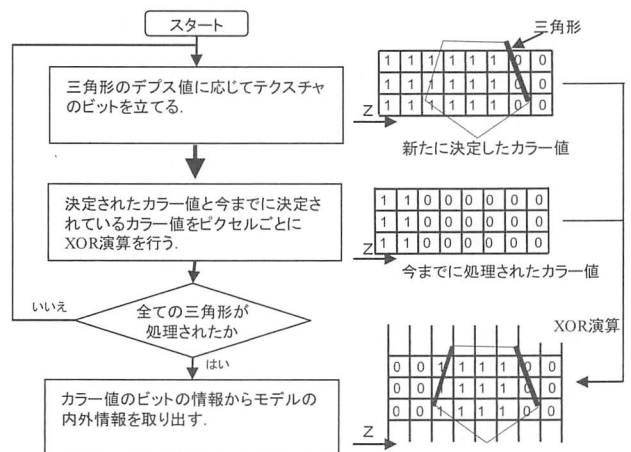


図3 ボクセル変換手法

この処理を、メッシュモデルを構成する三角形が全て処理されるまで行うことで、ボクセルモデルが生成される。

テクスチャメモリ1枚のカラー値は OpenGL の拡張ライブラリのひとつである GLEW ライブラリ [5] を用いて 32 ビット×4 の 128 ビット分確保できる。更に Z 方向に高解像度のボクセルモデルを生成するには、複数のテクスチャメモリに同時にレンダリングを行える Multiple Render Target (MRT) を組み合わせる必要がある。本研究で使用した GPU (NVIDIA Geforce 8500 GT) では MRT は 8 枚使えるので、 $128 \times 8 = 1024$  分割を Z の解像度とすることが出来る。

### 3. 処理の分散化

OpenGL アニメーションとボクセルモデル生成を同一の計算資源 (CPU と GPU) で行うと、ボクセルモデル生成の処理で 3D アプリケーションのパフォーマンスが低下する恐れがある。そこで、OpenGL Interceptor で取得した OpenGL 描画命令を、ソケット通信を用いて別 PC に転送し、ボクセルモデル生成の処理を行う。図 4 にその概略を示す。OpenGL 3D アプリケーションと OpenGL32.dll の間に割り込ませた OpenGL Interceptor を用いて取得した OpenGL 描画命令をバッファリングして、ある程度のサイズごとにボクセルモデル生成を行う PC に送信する。命令を受け取った PC は、フレームごとに OpenGL 描画命令をまとめ、前述の手法でボクセルモデルを生成する。なお、高速で通信を行うため、通信方式は UDP (User Datagram Protocol) を用いた。

### 4. ボクセルモデル時系列データ生成結果

本手法において、メッシュモデルの回転アニメーション (三角形数: 38380) を対象とし、実際にボクセルモデルを生成した結果を図 5 に示す。OpenGL アニメーション (図 4 (a)) から描画命令を取得して送信、描画に用いてボクセルモデル時系列データ (図 5 (b)) を生成した。3D アプリケーションシステムは、CPU: Core2duo6420, 2.13GHz, RAM: 2GB, GPU: Geforce 8500GT, ボクセル生成サーバは、CPU: Pentium4 3.8GHz RAM: 2GB GPU: Geforce8400GS である。ボクセル生成プログラムには、グラフィック言語 cg も用いた。

生成したボクセルモデル時系列データは 30 個、図 5 (b) のボクセル解像度は xyz でそれぞれ 512 とした。前報 [2] との比較結果を表 1 に示す。ボクセル生成処理が前報で使用していた手法に比べて高い解像度でも高速で処理できるようになった。一方、もう一つの問題点である、3D アニメーションの描画フレームレートの低下はほとんど改善されなかった。その原因としては、送信処理と送信のための文字列変換処理のオーバーヘッドが挙げられる。

### 5. 結論

OpenGL 3D アプリケーションの描画命令を取得することによって、フレームごとのボクセルモデルの時系列データを生成するシステムにおいて、高速なボクセル生成処理と、OpenGL の描画する PC とボクセル変換を行う PC を分け、その間で通信を行い、ボクセル生成時に実際の 3D アニメーションのパフォーマンスの低下を防ぐ処理を実装した。また、前報 [2] との比較を行い、実装手法の有効性を検討した。

今後の課題は、フレームレートの低下を抑制するために、送信処理と送信文字列変換処理の効率化である。

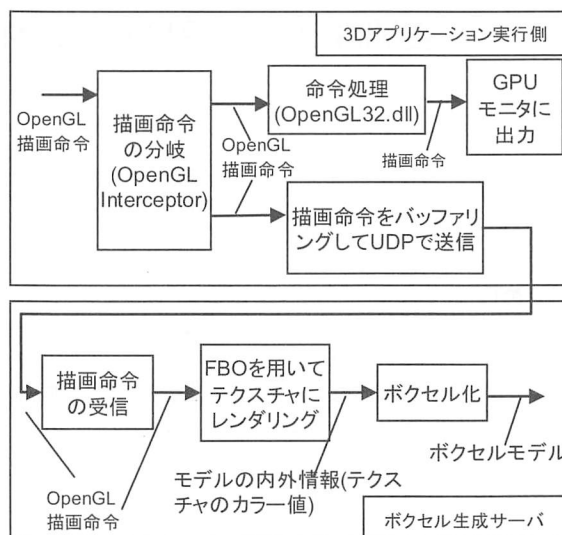


図 4 ボクセル生成処理の分散化

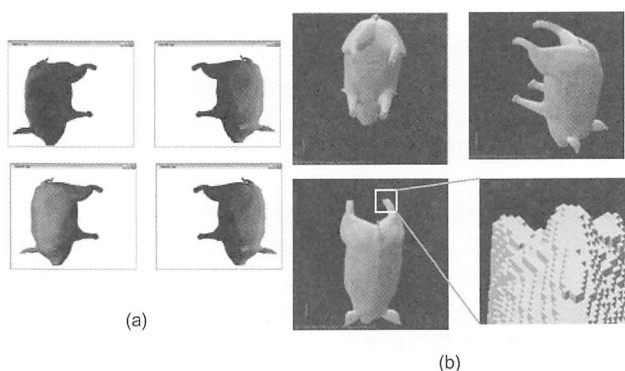


図 5 ボクセルモデル時系列データ生成結果

表 1 前回手法 [2] とのボクセル生成時間の比較

	OpenGL アニメーションのフレームレート [FPS]	ボクセル生成時間 [s]		
		128 <sup>3</sup>	256 <sup>3</sup>	512 <sup>3</sup>
前回手法	0.71	0.32	0.45	1.1
本手法	0.98	0.015	0.031	0.078

### 参考文献

- [1] 川岸良次他, サイバーフィールドのための 4 次元形状モデリングに関する研究 (第 2 報), 2008 年度精密工学会秋季大会学術講演会講演論文集, J03, 2008. 9
- [2] 加藤航平他, サイバーフィールドのための 4 次元形状モデリングに関する研究 (第 4 報), 2009 年度精密工学会秋季大会学術講演会講演論文集, 003, 2009. 3
- [3] D. Trebilco, GLintercept <http://glintercept.nutty.org/>
- [4] Eisemann 他, Single-Pass GPU Solid Voxelization for Real-Time Applications, Proceedings of graphics interface 2008, pp73-80
- [5] GLEW ライブラリ <http://glew.sourceforge.net/>