芸術科学会論文誌 Vol. 3 No. 1 pp. 69 – 75

光線ボリュームを用いたボリュームとボリュームの干渉シーンのレンダリング方法

A Method of Rendering Scenes Including Overlapped Volumetric Objects using Ray-Volume Buffers

梶原　景範　　高橋　裕樹　　中嶋　正之

Kagenori KAJIHARA　Hiroki TAKAHASHI　Masayuki NAKAJIMA

東京工業大学 大学院 情報理工学研究科

Graduate School of Information Science & Engineering, Tokyo Institute of Technology

あらまし　ボリュームオブジェクトが互いに干渉したシーンをレンダリングする方法を提案する．本提案方法は，信学ソ大 2001 で提案し，映情学誌，Vol.56,No.10, に採録の光線ボリュームを用いたボリュームとポリゴンの干渉シーンのレンダリング方法を拡張したものである．ボリュームの各ボクセルを通過する光線の色とそこまでの透過度を計算し，光線ボリュームバッファに出力しておくことによって，ボリュームもポリゴンも統一的にテクスチャ付きのポリゴンとしてレンダリングすることを特徴とする．本論文では，ボリュームと干渉するボリュームを半透明ポリゴンの規則的に配列した層と見なして，ボリュームとボリュームの干渉シーンのレンダリングへ拡張した．これによって，ポリゴン用グラフィックパイプラインでボリュームとポリゴンまたはボリュームが互いに干渉したシーンをレンダリング可能である．本提案方式は，光線ボリュームバッファの生成においては，ボリュームとポリゴンの干渉を考慮することなくボリュームを単独で処理すればよいことを特徴とする．

1　はじめに

バーチャルリアリティ(VR) 等で用いられる実時間画像生成は，高性能なビデオアクセラレータや OpenGL が広く使われるようになり，成熟した技術となりつつある．一方，近年ボリュームレンダリングに関する研究が盛んに行われ，小規模のボリュームであれば，ボリュームレンダリング専用ボードを付加した PC により実時間表示が可能になった [12].

今後，VR や訓練用シミュレータ用ビジュアルシステムの映像のリアリティ向上，応用範囲の拡大及び効果の向上のために，ボリュームとポリゴンが混在したシーンの映像生成の必要性が高まると考られる．例えば，訓練用シミュレータにおけるボリュームで出来た煙幕とその中を進む戦車，手術シミュレータにおける身体あるいは臓器と，それに挿入された手術機器, ゲームにおけるボリュームで出来たブッシュとポリゴンで出来た怪獣などの映像である．

VR やシミュレータにおけるモデリングにおいて，車や手術機器のような人工物はポリゴンモデルで比較的忠実に模擬することができる．ブッシュや樹木も，遠景であったりそれらが単独である場合は，透明ポリゴンにそれらの写真をテクスチャマッピングして模擬することができる．しかし，近景であったりブッシュの中を進む戦車のような互いに干渉しその度合いが刻々変化するようなシーンではブッシュはボリュームオブジェクトとする方がより忠実に模擬することができる．

戦車や AH1S などの攻撃用ヘリコプタの訓練用シミュレータにおいて実戦さながらの訓練をするには，自機を敵から隠したり目標機の視認性の悪さを模擬するために，ブッシュや樹木などの複雑な地物に加えて，もうもうたる砂塵，煙幕，火炎や煙などを模擬する必要がある．しかも，それらは互いに干渉し合っていて，干渉の具合は時々刻々変化するのが普通である．このように複雑な，あるいは不定形な物体を含むシーンを模擬するにはボリュームオブジェクトのみならずそれらと干渉し合ったポリゴンオブジェクトをも模擬する必要がある．

Ray-cast 法などによるボリュームレンダリングで，ゲーミングエリア全体を高精彩に実時間映像発生することは，現在では技術的にまだ非現実的である．また，Ray-cast などのボリューム用処理系とポリゴン用処理系を装備することは，システムを非常に複雑にする．特に，両オブジェクトが互いに干渉する場合は両処理系間の情報の交換および制御が複雑になる．

そこで，現在大規模なシミュレータに一般的に使用されているポリゴンベースのビジュアルシステムを基にして，ボリュームとポリゴンの両オブジェクトが混在し互いに干渉するシーンを，光線ボリュームを生成することによって，すべてポリゴンとしてレンダリングする方法を考案した [1][3].

本論文では，ボリュームとボリュームが干渉するシーンも許すよう上記方法を拡張する方法を提案する．ボリュームどうしのみの干渉のレンダリングは，両オブ

ジェクトが同一様式であるので Ray-cast 系のアルゴリズムでは容易であるが，本論文で提案する方式では，Ray-cast 系の処理系を持たずに，従来のポリゴン用のデプスバッファ方式のグラフィックパイプラインで，ボリュームとポリゴンまたはボリュームの干渉のレンダリングが可能である．

第 2 節で関連研究，第 3 節で光線ボリュームバッファ方式，第 4 節でシミュレーション結果と考察，第 5 節で残された問題について述べる．

2　関連研究

Kreeger らは，ポリゴンが混在した状態をボリュームレンダリングするために，ポリゴンをボリュームの層ごとにスライスすることによって，ボクセルとポリゴンの位置関係を調べて層ごとに Ray-cast でレンダリングしている [7]．彼等の方式の長所は，半透明ポリゴンのデプスソートが不要なことであり，短所はスライスにより 1 つの三角形が多数の小さい三角形に分割されること，および処理が複雑なことである．

Giertsen らは，ボリュームとサーフェスを同時にレンダリングするのに Scan-Plane Buffer(SPB) と Multiple-Layer z-Buffer(MZB) を用いてスキャンラインオーダのレンダリングをした [8]．彼等の方式の特徴は，ボリュームのメッシュ構造を問わないことである．欠点としてはエイリアシングが生じやすいことである．

Levoy は Ray-Trace 法をボリュームレンダリングに拡張し，ボリュームとポリゴンの混在したシーンの映像生成を行う方法を提案した [9]．彼の方法の長所は，リアルな映像生成とアルゴリズムが比較的シンプルであることであり，短所としては計算負荷が大きいこと，エイリアシングが生じやすいことである．

Bitter らは，ボリュームレンダリングにおいて，色の合成計算をボリュームの層ごとに計算し積算する Ray-Slice-Sweep Volume Rendering 法を提案している [10]．しかし，Ray の計算に線形補間を使用し，しかも Sweep しているので，本論文で指摘している光線の拡散が生じ映像はボケるという欠点がある．

3　光線ボリュームバッファ方式

本論文では，投影法は平行投影とする．この節では，まず基本的なアプローチ，光線ボリュームバッファ生成，およびボリュームとポリゴンの干渉の模擬方法について簡単に述べる．詳細は参考文献 [1][2][3][4] を参照されたい．その拡張としてボリュームとボリュームの干渉の模擬について述べる．

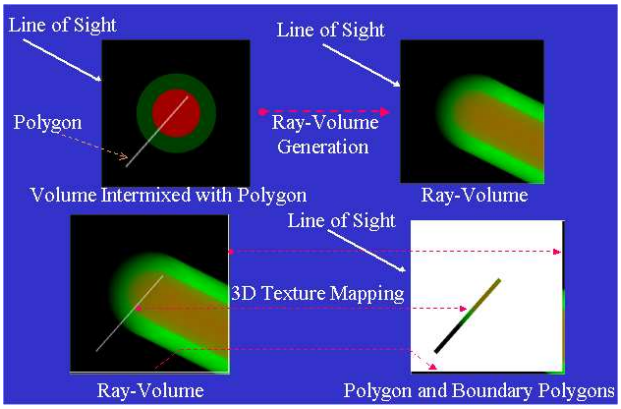

図 1: 本提案方式の概念図

3.1　基本的なアプローチ

ボリュームとポリゴンの混在シーンの，光線ボリュームバッファを用いたレンダリングの基本的なアルゴリズムは次のとおりである．すなわち，フレーム処理の初段でボリュームの中を通過する光線の色とそこまでの透過度をボクセルごとに計算し，その結果を光線ボリュームバッファに格納しておく．ボリュームと干渉するポリゴンは，光線ボリュームバッファを 3D テクスチャとしてマッピングして表現する．ボリュームは，その輪郭ポリゴンに光線ボリュームをマッピングして表現する．図 1 は，本提案方式の概念図であってボリュームにポリゴンが挿入されたシーンを，光線ボリュームを生成しそれをテクスチャマップすることによって，三枚のポリゴンで表現することを表している．

ボリュームと干渉するボリュームは規則的に配列した半透明ポリゴンの層として模擬する．これによって，上記と同じスキームでボリュームとボリュームの干渉シーンのレンダリングが可能になる．

3.2　光線ボリュームの生成

ボリューム内の光線の伝播を模擬し，光線ボリュームを生成する．すなわち，各ボクセルのスクリーンからの奥行きを z とし，そこを通過する光線の色 $C(z)$，およびそこまでの透過度 $t(z)$ を次式により計算し，光線ボリュームバッファに格納しておく．ここで $\psi(\eta)$ および $\phi(\xi)$ は，それぞれ奥行き η におけるボリュームの色および奥行き ξ における不透明度である．

$$C(z) = \int_0^z \psi(\eta) e^{-\int_0^\eta \phi(\xi) d\xi} d\eta \quad (1)$$

$$t(z) = e^{-\int_0^z \phi(\xi) d\xi} \quad (2)$$

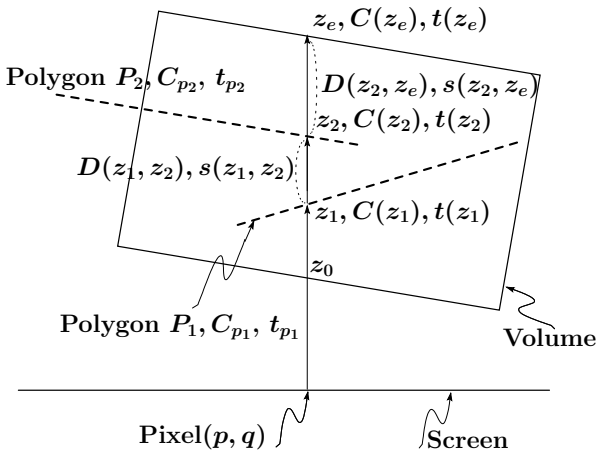

図 2: Polygon の挿入

3.3 ボリュームと干渉したポリゴンの表示

図 2 はスクリーン上の画素 (p, q) において奥行き z_1, z_2 に，それぞれ色 C_{p_1}，透明度 t_{p_1} の半透明なポリゴン P_1，色 C_{p_2}，透明度 t_{p_2} の半透明な P_2 が挿入された状態を示す．

このとき，画素 (p, q) に見える色 C_{pq} は次式により得られる．この式の導出に関しては参考文献 [3] を参照されたい．

$$\begin{aligned} C_{pq} &= C(z_1)(1-t_{p_1}) + t(z_1)C_{p_1} \\ &\quad + t_{p_1}\{C(z_2)(1-t_{p_2}) + t(z_2)C_{p_2}\} \\ &\quad + t_{p_1}t_{p_2}C(z_e). \end{aligned} \tag{3}$$

ここで，$C(z_*)$ および $t(z_*)$ は，式 (1) および (2) で与えられ，奥行き z_* における光線ボリュームの色及び透明度である．z_e は画素 (p, q) におけるボリュームの終端の奥行きである．

3.4 ボリュームとボリュームの干渉

ボリュームと干渉したボリュームを半透明ポリゴン 1，2，. . . ，n が規則的に並んだ層と考えると，画素 (p, q) の色 C_{pq} は次式で与えられる．

$$\begin{aligned} C_{pq} &= \sum_{k=1}^{n}\Big(\prod_{j=0}^{k-1} t_{p_j}\Big)\{C(z_k)(1-t_{p_k}) + t(z_k)C_{p_k}\} \\ &\quad + \Big(\prod_{j=0}^{n} t_{p_j}\Big)C(z_e) \end{aligned} \tag{4}$$

3.5 アルゴリズムの実装

これらのアルゴリズムは，アルファバッファをもったデプスバッファ方式のポリゴングラフィックスで実現することができる．すなわち，式 (4) は，最初アルファバッファの値 α を 0 に初期化しておき，半透明ポリゴンが挿入されるごとに，$(1-\alpha)$ を乗じた $C(z_i)(1-t_{p_i}) + t(z_i)C_{p_i}$ をフレームバッファに加算し，次にアルファバッファの値 α にポリゴンの不透明度 $(1-t_{p_i})$ を加算してアルファバッファを更新しておくことによって実現できる．

このとき，アルファバッファのビット深さがボリュームの透明度の模擬の精度に関係する．単位ボクセル長さ当たりのボリュームの不透明度の分解能が $d\alpha$ とすると，ボリュームの透明度を正確に模擬するためには，アルファバッファのビット深さは $-\log_2 d\alpha$ ビット以上必要である．第 4 節におけるシミュレーション実験では，単精度の浮動小数点を用いたので，アルファバッファのビット深さは 24 ビットである．

干渉したボリュームは直方体と仮定し，視線方向に最も平行な稜に垂直な層にスライスしたポリゴンとしてレンダリングする．図 3 は視点座標系，ボリューム座標およびスライス座標の関係を示す．ボリューム座標系は，ボリュームの直交する稜の上に，ボリュームを各軸の正の部分に含み右手系となるように，視点座標系は，視点を原点とし z_e 軸を視線方向に，x_e 軸をスクリーンの右手方向に，y_e 軸を下方に採るものとする．これらの座標系に対して，スライス座標系は，ボリュームの直交する稜の上に，視線と最も直交するボリュームの手前の面の左上の頂点を原点に，z_s 軸を奥行き方向に，x_s 軸及び y_s 軸をその面の辺上に右手系になるように設定するものとする．

視点座標系におけるボリューム原点を $V_{ev_o} = (x_{ev_o}, y_{ev_o}, z_{ev_o}, 1)^T$ とし[1,2]，ボリューム座標系から視点座標系への変換マトリックス T_{ev} が次式で与えられるとする．ただし，位置ベクトルは列ベクトルとする．

$$T_{ev} = \begin{pmatrix} l_{11} & l_{12} & l_{13} & x_{ev_o} \\ l_{21} & l_{22} & l_{23} & y_{ev_o} \\ l_{31} & l_{32} & l_{33} & z_{ev_o} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

ボリューム座標系からスライス座標系への変換マトリックス T_{sv} が，

$$T_{sv} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & x_{sv_o} \\ a_{21} & a_{22} & a_{23} & y_{sv_o} \\ a_{31} & a_{32} & a_{33} & z_{sv_o} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

で与えられるとすると，a_{ij} は次式で与えられる．

$$a_{ij} = \begin{cases} l_{ij}/|l_{ij}| & : |l_{ij}| = \max\limits_{k=1,2,3} |l_{kj}|, \forall k < j, a_{ik} = 0 \\ 0 & : other \end{cases} \tag{7}$$

[1]位置ベクトルの表記において最初の添え字は座標系を表す．
[2]$(x, y, z, 1)^T$ は $(x, y, z, 1)$ の転置行列を表すものとする．

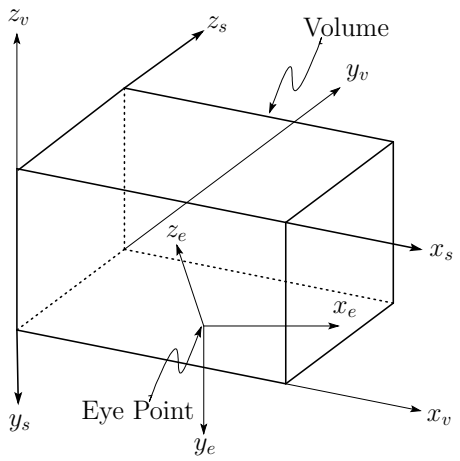

図 3: スライス座標系

また，スライス座標系におけるボリューム原点 $V_{sv_o} = (x_{sv_o}, y_{sv_o}, z_{sv_o}, 1)^T$ は，ボリューム座標系における x_v 方向，y_v 方向および z_v 方向のボリュームサイズを l_{vx} , l_{vy}, l_{vz} として，次式で与えられる．

$$V_{sv_o} = (|l'_{sx}| - l'_{sx}, |l'_{sy}| - l'_{sy}, |l'_{sz}| - l'_{sz}, 2)^T \quad (8)$$

ここで，l'_{sx}，l'_{sy} および l'_{sz} は次式で与えられる．

$$(l'_{sx}, l'_{sy}, l'_{sz}, 0)^T = T_{sv}(l_{vx}, l_{vy}, l_{vz}, 0)^T \quad (9)$$

スライス座標系におけるボリュームのサイズ l_{sx}, l_{sy}, l_{sz} は，$|l'_{sx}|, |l'_{sy}|, |l'_{sz}|$ で与えられる．スライスされたポリゴンの頂点は，スライス座標系で，$V_{s1} = (0, 0, k, 1)^T$，$V_{s2} = (l_{sx}, 0, k, 1)^T$，$V_{s3} = (l_{sx}, l_{sy}, k, 1)^T$，$V_{s4} = (0, l_{sy}, k, 1)^T$ で与えられ，これらをボリューム座標系に変換した点を $V_{v1}, V_{v2}, V_{v3}, V_{v4}$ とする．このスライスポリゴンを $k = 0 \sim l_{sz}$ の順にレンダリングする．

具体的には次の処理を行う．上記スライスポリゴンをスクリーンに投影したときのポリゴン内部のピクセル位置を視点座標系で $V_{ep} = (p, q, 0, 1)^T$ とし，ボリューム座標系における，視線方向を N_{veye}，スライスポリゴンの法線ベクトルを N_v とすると，ピクセル V_{ep} に対応するスライスポリゴン上の点 V_v は，ボリューム座標系で次式で与えられる．

$$V_v = V_{vp} + \alpha N_{veye} \quad (10)$$

ここで，

$$V_{vp} = T_{ev}^{-1} V_{ep} \quad (11)$$

$$\alpha = (V_{vp} - V_{v1}) \cdot N_v / (N_{veye} \cdot N_v) \quad (12)$$

$$V_{v1} = T_{sv}^{-1} V_{s1}. \quad (13)$$

光線ボリュームのボクセルの中，この点 V_v を囲む上下左右および前後の 8 ボクセルの色と透過度を読み出し，tri-linear の補間をして $C(z_k), t(z_k)$ とし，式 (4) に適用してスライスポリゴンをレンダリングする．

互いに干渉するボリュームのうち，どちらを半透明ポリゴンとして扱いどちらを光線ボリュームとして扱うかの問題であるが，ボクセル間隔とピクセル間隔が同程度であるとすると，原理的には双方の計算時間はほぼ等しいといえる．なぜなら，光線ボリュームの方は手前の層を次の層に順々にマッピングして生成するのに対して，半透明ポリゴンとして扱う場合は各層を順々にスクリーンへマッピングすることになり，どちらも同数枚のポリゴンのマッピングということになるからである．

実際 $160 \times 160 \times 34$ ボクセルのロブスターでイメージサイズをほぼ 160×34 ピクセルにして PentiumIII 700MHz 384MB で計測するとどちらも計算時間は約 3.8 秒であった．イメージのサイズをこれより大きくすると半透明ポリゴンとして扱う方が時間が掛かる．画質については，光線ボリュームの生成に無補間を用いボクセル間隔がピクセル間隔より大きい場合，半透明ポリゴンとして処理した方が画質がいくらかいいようにも見えるが，その差異は非常に小さいものであった．従って，ピクセル面積でボリュームの延べ面積を正規化したとき小さい方を半透明ポリゴンとして扱うのが処理時間的には有利である．

以上の議論は，ボリュームにポリゴンが干渉していないときのことであって，ポリゴンが干渉している側のボリュームは光線ボリュームを生成する必要がある．ポリゴンが両方のボリュームに干渉しているときは両方の光線ボリュームを生成し，後から生成する光線ボリュームは先のボリュームを考慮して光線ボリュームを生成しておき，干渉ポリゴンをレンダリングするとき両方のボリュームがオーバーラップした部分では後者をマッピングする．ボリュームそのものはそれぞれのバンダリーポリゴンで表現する．

互いに干渉するボリュームが 3 個以上のときは一番大きいボリュームを半透明ポリゴンの層として扱い，他のボリュームについて光線ボリュームを作成しておくことによってレンダリングすることができる．いくつのボリュームまで干渉が許されるかは，1 つのポリゴンに何種類のテクスチャが許されるかによって決まる．

次に，光線ボリュームの生成と半透明ポリゴンのレンダリングにおける補間法の違いについてであるが，tri-linear の補間はボリュームが互いに重複している部分において相手方の光線ボリュームに対して行われるので，どちらを半透明ポリゴンとして扱っても，このことによる計算量及び画質の違いはない．

光線ボリュームの生成における補間法が線形補間でない理由は，線形補間を用いてボリューム中の光線の伝播を模擬すると近傍のボクセルの影響が伝播し，光線の拡散を引き起こしボリュームの像のボケが生ずる．無補間で近い方のボクセルを用いると光線のぶれは最大 2 分の 1 ボクセル幅に抑えられ比較的シャープな像が得られる．また，高階の補間法を用いると光線の拡散をそれだけ抑えることができる．これらに関しては文献 [3] に詳細に記述した．

4 シミュレーション実験

本提案手法に基づいて，水素分子の電荷密度分布を表すボリュームの中に差し込まれたロブスターと霧のかかった樹木をシミュレーションした結果を，それぞれ図 4，図 5 に示す．図 6 はロブスターと頭部 CT ボリュームデータの干渉シーンのシミュレーション結果である．

水素分子電子雲ボリュームのサイズは $240\times240\times120$ ボクセル，頭部 CT ボリュームのサイズは $128\times128\times112$ ボクセルである．ロブスターは $160\times160\times34$ ボクセルのボリュームであって，160×34 ピクセルの半透明ポリゴン 160 枚として処理した．樹木のボリュームサイズは 128^3 ボクセル，霧は $220\times100\times50$ である．

シミュレーションは，PentiumIII 700MHz 384MB を用いて，光線ボリュームの生成，3D テクスチャマッピングおよびポリゴンのレンダリングすべてソフトウェアで行った．また，光線ボリュームの生成において，式 (1)，(2) を数値的に積分するときに線形補間を行うと，光線の発散が生じ像がボケるので無補間または高階の補間を用いる必要があることを参考文献 [3] で述べたが，ここではすべて無補間の方法で行った．

図 4 のシミュレーションにおいて，光線ボリュームの生成に約 15 秒，640×480 ピクセルのスクリーンへのレンダリングに約 23 秒を要した．図 5 のシミュレーションにおいては，光線ボリュームの生成が約 5.7 秒，640×480 ピクセルのスクリーンへのレンダリングが約 46.3 秒であった．霧が流れる様子を模擬したが，霧の懸かり具合によってレンダリング時間は 26.6 秒〜50.5 秒と変動した．図 6 のシミュレーションにおける光線ボリューム生成及び同上のスクリーンへのレンダリング時間は，それぞれ約 6.0 秒及び約 35.8 秒であった．

以上のことから，光線ボリュームの生成時間はベースになるボリュームのサイズにほぼ比例し，レンダリング時間は干渉するボリュームサイズに関係し，干渉の度合いにより変動する．この理由については次節で述べる．

5 考察

5.1 計算量に関する考察

光線ボリュームバッファを用いることにより，ボリュームとポリゴンが干渉したシーンをボリュームとポリゴンを統一的にレンダリングでき，その手法をボリュームとボリュームの干渉シーンに拡張できることを示した．

本論文では，手法の有用性の実証を目的とし，高速化のためのプログラミングにはあまり意を払わなかった．例えば，半透明ポリゴンを予め干渉している部分だけにクリッピングしておくと計算時間は少なくなるが，ポリゴン全体をレンダリングした．それでも式 (10) の V_v がボリューム内にない場合は計算量が少なく，干渉の度合いが少ない場合はレンダリング時間が短くなる．

処理量について Kreeger らの方法と比較検討する．Kreeger らのレンダリングはボリュームの各層に対して次のステップを繰り返す．

Step 1: ボリュームのスライスをテクスチャ付きポリゴンとして，デプスチェック付きで表示

Step 2: bucket ソートにしたがって次のスライスとの間のポリゴンをスライスおよびその三角形化

Step 3: 三角形をデプスチェック付きでテクスチャマッピングして表示

Step 4: bucket ソートシステムの更新

Step1 は本提案方式の光線ボリュームの生成に対応し，本提案手法の場合デプステストは不要である．デプステストなしのポリゴンのレンダリングは，実験によると 10%早くなった．文献 [7] によればポリゴンは数が 2 ないし 3.5 倍に細分化されるので，ポリゴンのシェーディング面積は一定としても，三角形のセットアップ処理[3]は，本提案手法に対してそれだけ掛かることになる．ポリゴンが比較的小さいとして[4]，三角形のセットアップ処理とシェーディング処理が同等とすると，Kreeger らの方法は本提案方式に対して 3/2 から 4.5/2 倍のポリゴン表示時間を要することになる．また，Step2 および Step4 は本提案方式では不要である．

処理量は Step1 と Step3 が全体の大部分を占めると考え，Step1 と Step3 の割合は挿入されたポリゴンの延べ面積に依存するが大雑把に同程度とし，各ステップの処理量の割合を次のように仮定した．Step1 および Step3 がそれぞれ全処理の 50%および 40%，Step2 と Step4 で 10%とすると，本提案方式は Kreeger らの方法に対して 50%×0.1+40%×0.5×{1-

[3]エッジの傾斜や輝度の変化率の計算等

[4]大きければポリゴン数の増加率は 2 ないし 3.5 倍より遥かに大きくなる

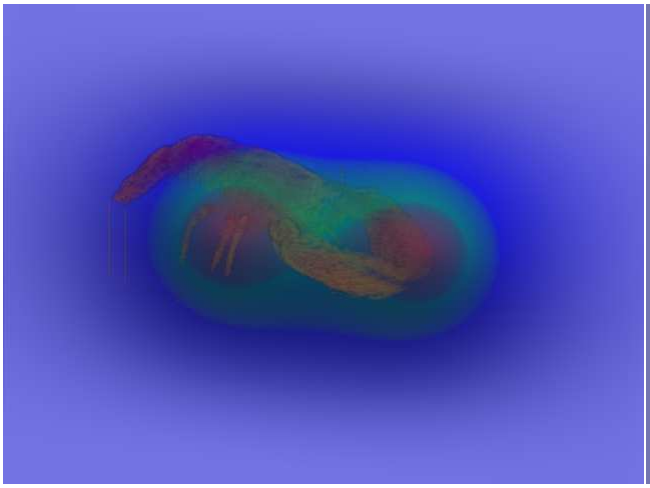

図 4: 水素分子とロブスター

図 5: 樹木と霧

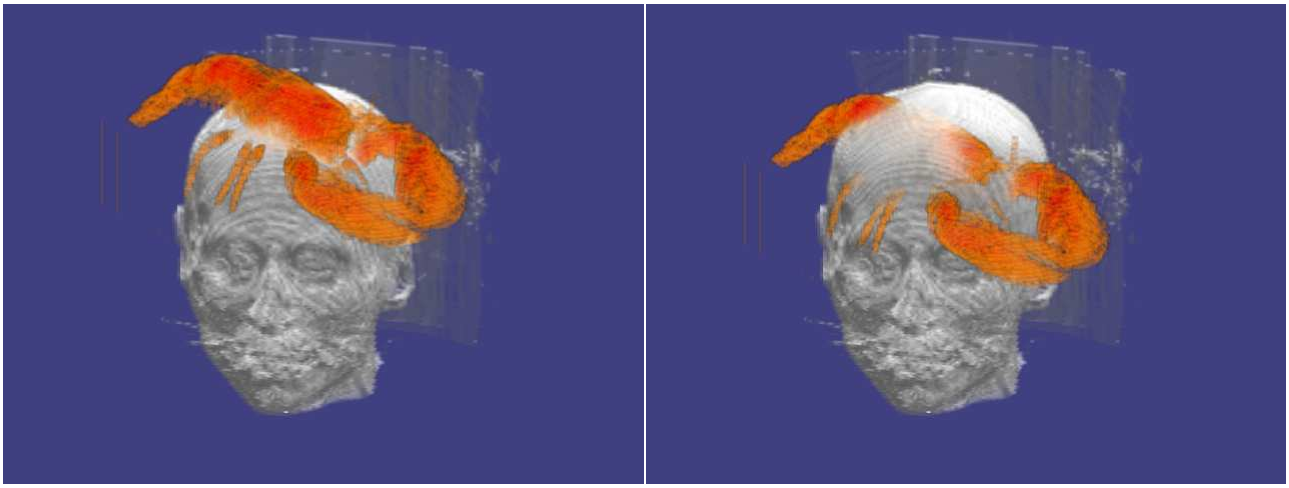

図 6: 頭部 CT とロブスター

(1/2∼1/3.5)}+10%=25∼29% 処理が軽いことになる．

5.2 本提案方式の特徴

ポリゴンと干渉したボリュームをレンダリングする場合，ボリュームの，ポリゴンの手前にある部分と後ろ側にある部分を知る必要がある．関連研究では，それぞれこのための処理に工夫がなされている．すなわち，Kreeger らはボリュームの層ごとにボリュームをポリゴンとともにスライスし，Giertsen らはスキャンラインごとにビュウボリュームをスキャンラインプレーンでスライスし，Levoy は Ray-Trace 法によって，ボリューム中のポリゴンの位置を確認しながらレンダリングしている．

本提案手法では，このための処理を必要とせず，光線ボリュームバッファと 3D テクスチャマッピングによってこの機能を実現する．光線ボリュームバッファの生成は，ポリゴンとの干渉を全く考慮する必要がなく，単純なボリュームレンダリングと等価な計算量で可能ある．また，3D テクスチャマッピングは，最近のグラフィックハードウェアでは，レンダリング速度に対するペナルティなしにポリゴンのレンダリングができる．

本提案手法の利点は，光線ボリュームの生成部分を 3D テクスチャジェネレータとして局所的に追加することによって，従来型のグラフィックスシステムでボリュームオブジェクトが描画できるよう機能拡張が可能であることである．さらに，光線ボリュームの生成においては，ポリゴンとの干渉を考慮する必要がないので，構造が単純で LSI 化に向いていることである．また，ボリュームもポリゴンとして表示するが，全ボクセルを処理して得たテクスチャをマッピングするの

で，同じくボリュームをポリゴンで表現する Marching Cube 方式が等値面のみを表示するのとは異なり [11]，ボリュームの持つ質感・量感も充分に表現できることである．また，逆にポリゴンをボクセルに変換することもないので量子化の問題もない [5].

本提案手法の欠点は，ボリュームデータを直接レンダリングする方式に比べて，光線ボリュームバッファ用の大量のメモリを必要とすることである．例えば，図 4 の水素分子の電荷密度分布を表すボリュームの光線ボリュームのサイズは，ボクセル当たり RGB と透明度の 3 + 4 Byte で，全体として $7 \times 240 \times 240 \times 120 \approx 48.4$MByte である．

また，Ray-cast 法においては，光線上の不透明度の累積がほぼ 1 に達すると処理を打ちきることができるのに対して，光線ボリュームは，ボリューム全体を最後まで処理する必要があることである．本提案手法においては，半透明ポリゴンはデプスソートされている必要があるので，BSP(Binary Space Partitioning) 等によるソートが必要である．これは，本提案手法の欠点というより，デプスバッファ方式のグラフィックスシステム本来の制限であると考える．

6 おわりに

本論文では，先に提案したフレーム処理の初段で光線ボリュームバッファを生成しておくことにより，ボリュームとポリゴンを統一的にレンダリングする方法 [3] を拡張しボリュームとボリュームの干渉シーンをレンダリングする方法を提案した．ボリュームとポリゴンまたはボリューム相互の干渉の計算には，複雑な処理を必要とせず，テクスチャマッピングの手法を用いて実現したことが特徴である．

今後の課題としては，本論文では平行投影に限定した方式を提案したが，訓練用シミュレータ等に応用するには透視投影に拡張する必要がある．特に視点がボリュームの中に入った場合も許すようなアルゴリズムを検討する予定である．また，グラフィックスハードウェアの利用及び PC クラスタを用いた並列化によるアルゴリズムの高速実装を検討したいと考えている．

参考文献

[1] 梶原, 高橋, 中嶋, “光線ボリュームバッファを用いたボリュームとポリゴンの混在シーンの生成方法”, 信学ソ大 D-11-79，2001.

[2] 梶原, 高橋, 中嶋, “光線ボリュームバッファを用いたボリュームとポリゴンの混在シーンの生成方法”, 映情学技報 VOL.25，No.85, pp.19-24, 2001.

[3] 梶原, 高橋, 中嶋, “光線ボリュームバッファを用いたボリュームとポリゴンの混在シーンのレンダリング方法”, 映情学誌, Vol.56, No.10, pp.1607-1612, 2002.

[4] 梶原, 高橋, 中嶋, “光線ボリュームバッファを用いたボリュームとポリゴンの混在シーンのレンダリング方法— ボリュームとボリュームの干渉シーンのレンダリングへの拡張 —”, 第 18 回 NICOGRAPH 論文コンテスト論文集, pp.33-38, 2002.

[5] A. Kaufman, D. Cohen, R. Yagel, “Volume Graphics”, IEEE Computer, vol.26, no.7, pp.51-64, July 1993.

[6] A. Kaufman, D. Cohen, R. Yagel, “サーフェスグラフィックスからボリュームグラフィックスへ”, 日経 CG, 1 月号，pp148-155，2 月号 pp.130-137, 1994(上記邦訳).

[7] K. Kreeger, A. Kaufman, “Mixing Translucent Polygons with Volumes”, Proceedings Visualization ’99, pp.191-198, 1999.

[8] C. Giertsen, A. Tuchman, “Fast Volume Rendering with Embedded Geometric Primitives”, Visual Computing —Integrating Computer Graphics with Computer Vision, T.L. Kunii(ed), Springer Verlag, pp.253–271, 1992.

[9] M. Levoy, “A Hybrid Ray Tracer for Rendering Polygon and Volume Data ” IEEE Computer Graphics & Applications 10(2), 33-40, Mar. 1990.

[10] I. Bitter, A. Kaufman, “A Ray-Slice-Sweep Volume Rendering Engine”, In Proc. of Eurographics/SIGGRAPH workshop on graphic hardware 1997, pp.121-130, 1997.

[11] W. Lorensen, E.H.Cline, “Marching Cubes: A High Resolution 3-D Surface Construction Algorithm”, In Proc. of SIGGRAPH ’87 in Computer Graphics 21,4, July 1987.

[12] H. Pfister, J. H. Hardenburg, H. Lauer, S. Sailor, “The VolumePro Real-Time Ray-Casting System” Proceedings of the ACM SIGGRAPH ’99 Conference pp131-138, Aug. 1999.