特別研究報告書

視点移動速度に応じた精度制御を行う実時間ボリュームレンダリング処理

指導教官　　富田 眞治 教授

京都大学工学部情報学科

牧田 俊明

平成16年2月2日

視点移動速度に応じた精度制御を行う
実時間ボリュームレンダリング処理

牧田 俊明

内容梗概

ボリュームレンダリングとは，3次元的に中身の詰まった半透明なボリュームデータを2次元平面上に投影し，画像を出力する手法である．この手法は，医療画像分野で人体内部を表現したり，炎や霧，雲などのはっきりした形を持たないものを可視化するのに用いられてきた．

近年では，コンピュータの発達によって従来では成しえなかったような高速なレンダリングが可能になった．しかし，動的に視点を変更しながらボリュームレンダリングしようとすると，計算量が膨大になり，リアルタイムな可視化が難しくなる．

そこで，人間の知覚特性として，動いている物体に対しては，静止している物体に対してよりも認識能力が劣るという性質があるのを利用して，ボリュームレンダリングにおいて，視点の移動速度に応じた精度制御を行う手法を提案する．

本研究では，汎用グラフィクスカードを用いたテクスチャベースボリュームレンダリングにおいて，ボリュームデータの再サンプリング率を利用した手法とボリュームデータの解像度を利用した手法を実装し，それぞれについての描画速度に対する効果や，出力画像の画質などを評価した．また，視点移動速度に応じた精度の制御手法についても評価した．

ボリュームレンダリング処理にはレイキャスティング法を用いた．レイキャスティング法では，投影面のピクセル上から視線ベクトルを飛ばし，一定間隔でサンプリング点を取ってボリュームデータの不透明度に従って混合し，そのピクセルでの値とする．

ボリュームデータの再サンプリング率を利用した手法では，視線ベクトルのサンプリング率を減少させて1ピクセルごとの計算を減らすことと，視線ベクトルの数を減少させてピクセルの数を減らすことで，計算量を減らす．また，ボリュームデータの解像度を利用した手法では，ボリュームデータのメモリ上でのサイズを減らすことで，キャッシュ・メモリの利用効率を上昇させ，メモリ・アクセスの回数を減らすことで高速化する．

ボリュームデータの再サンプリング率による手法では，物体が動いている間はほとんど劣化しているのがわからない程度の画質でも，2 倍程度の描画速度向上が得られることが確認された．また，出力画像の細かな部分の精度を問わないのであれば，データにもよるが 10 倍以上の描画速度向上も可能であった．

ボリューム解像度による手法は，メモリに余裕がなければ用いることができない上，単体で使用しても画質の劣化が激しく，実用性は低かった．しかし，再サンプリング率による手法と組み合わせることで，画質をほとんど劣化させることなく描画速度の向上が見込める手法でもある．

これら 2 種の精度制御により，視点移動速度と出力画像の精度・描画速度の関係について評価した．それにより，なるべく自然な動きの出力を求めるならば，視点が移動している間は常に 30fps 程度を保つように精度制御すべきであることが分かった．また，自然な動きよりも，物体が動いている間でも出力画像の精度を優先するのであれば，物体がゆっくり動いている間は 30fps 未満の中間の精度を持つ出力を用いるのがよいと言える．

今後の課題としては，精度制御の手法として本研究で実装できなかった Early Ray Termination を利用した手法の効果や，メモリに入りきらないような大きなデータを 1 台のマシンで扱った場合と並列化処理した場合においての精度制御の効果の違いを調べたい．

Real-time Volume Rendering Controlled with Level of Accuracy According to Moving Speed of the Viewpoint

Toshiaki Makita

Abstract

Volume rendering is a technique of projecting half transparent volume data, which is filled up in 3 dimensions, on a 2-dimensional plane. This technique has been used for expressing the inside of a human body in a medical image field or visualizing formless things such as clouds, fog, and a flame.

Recently, advanced computer technology have made it possible rendering in high speed that had not been able to be accomplished in former days; However, amount of calculation of real-time volume rendering with a viewpoint changing dynamically is still too huge to carry out it.

To take advantage of the inferiority of man's recognition capability to the moving object to that of the stationary object, I propose the technique by which we control volume rendering with level of accuracy according to moving speed of the viewpoint.

In this research, using texture-based volume rendering on the standard graphics card, I implemented the technique changing resampling rate of volume data and that using multi-resolution volume data, and evaluated the effect over drawing speed and the quality of the output image of each, etc. In addition, I also evaluated the control techniques of level of accuracy according to moving speed of the viewpoint.

I carried out volume rendering by ray-casting. In ray-casting, we fly a ray from a pixel of the projection plane, take sampling points at regular interval, and blend them according to the opacity value of the volume data. Thus, we calculate the value of pixels on the projection plane.

In the technique via resampling rate, we reduce amount of calculation by decreasing sampling rate of a ray so as to reduce calculation in each pixel and by decreasing the number of rays so as to reduce the number of pixels. In the technique of multi-resolution volume data, we accelerates calculation by reducing the size of volume data on the memory so that we use cache memory more efficiently and reduce the number of memory access.

The improvement of about 2 times in drawing speed of volume rendering was obtained by using the technique via resampling rate when the quality of the output image in which object is moving was able to hardly be recognized to become worse. And if the accuracy of the fine portions of the output image does not be required, although it depends on data, the improvement in drawing speed of 10 or more times is also possible.

The technique of multi-resolution volume data is not able to be used if there is no margin in a memory. Moreover, using the technique alone made quality of the output image worse so extremely that it little is practical. But, the combination of technique of multi-resolution volume data and via resampling rate is able to be expected to improve drawing speed of volume rendering with little degrading quality of the output image.

I also evaluated about the relation of moving speed of the viewpoint, the accuracy of the output image, and drawing speed, using these 2 techniques of level of accuracy control. Thereby it was turned out that if it is required that the sequence of the output image has a as natural motion as possible, level of accuracy should be controlled to maintain about 30fps of frame rate of the output image while the viewpoint is moving. On the other hand, if it is demanded that the sequence of the output image is detailed rather than natural while the object is moving, it is good to use middle level of accuracy of that frame rate of the output image is less than 30 fps while the object is moving slowly.

As a future work, I want to investigate the effect of the technique of level of accuracy control such as Early Ray Termination etc. that has not been implemented in this research, and the difference in the effect of level of accuracy control between using parallelized computers and mere a computer to treat such a huge data that it cannot be stored in a memory.

視点移動速度に応じた精度制御を行う
実時間ボリュームレンダリング処理

目次

第1章　序論

ボリュームレンダリングとは，3次元的に中身の詰まった半透明なボリュームデータを2次元平面上に投影し，画像を出力する手法である．この手法は，医療画像分野で人体内部を表現したり，炎や霧，雲などのはっきりした形を持たないものを可視化するのに用いられてきた．

近年では，コンピュータの発達によって従来では成しえなかったような高速なレンダリングが可能になった．しかし，動的に視点を変更しながらボリュームレンダリングしようとすると，視点を変更する度に描画しなおさなければならないので計算量が膨大になり，リアルタイムな可視化が難しくなる．

しかし，人間は動いている物体に対しては，静止している物体よりも，認識能力が低い．また，物体の動きが速いほど，認識能力は低下する．そこで，視点の移動速度に応じて画質を変化させて計算量を増減する手法を提案する．これにより，視点の移動速度が速いときは画質は低いが高速に描画できるようにし，視点が静止している間は描画は遅くても構わないので画質を高くすることができる．これを以下，ボリュームレンダリングの精度制御と呼ぶ．

本研究では，汎用グラフィクスカードを用いたテクスチャベースボリュームレンダリングにおいて2種類の精度制御の手法を実装し，各手法の性質，有効度，組み合わせの効果や，視点移動速度と出力画像の精度や描画速度との関係などを調べ，評価した．

本稿では，以下第2章で背景となるボリュームレンダリングとテクスチャベースボリュームレンダリング，2次元画像の精度制御について述べ，第3章で精度制御の各手法を説明する．その後，第4章で実装，第5章で結果を評価し，第6章で並列化の考察について述べた後，第7章でまとめる．

第2章　背景

2.1　用語説明

まず最初に，以後の背景の説明に必要な用語の解説をしておく．

ボリュームデータ　連続な3次元空間内に分布するマルチスカラ量で表されるデータ．

ボクセル集合　連続なボリュームデータを計算機で扱えるように離散化したもの．ボリュームデータのサンプリングと量子化によって得られる．

ボクセル　ボリュームデータの離散化によって得られたボリューム空間内の最小単位.

フィールド値　ボリューム空間内に分布しているマルチスカラ量のこと.

ボクセル値　ボクセル集合におけるボクセルのある点のフィールド値のこと．元のボリュームデータのサンプリングと量子化によって得られた，離散的に分布するフィールド値.

再サンプリング　ボクセル集合から，元となった連続ボリューム空間内の任意の点におけるフィールド値を求めること．また，再サンプリングによってフィールド値を得た点を再サンプリング点という.

伝達関数　ボリュームデータのフィールド値からカラー値（レッド，グリーン，ブルーの各値）と不透明度値を算出する関数.

OpenGL　汎用グラフィクスカード上でグラフィクス処理を行うための汎用API．本研究ではOpenGLを用いてテクスチャベースボリュームレンダリングを実装している.

アルファ値　OpenGLにおいて不透明度を表現するために用いられる値.

RGBA値　OpenGLにおいてカラー値とアルファ値を表すのに用いられる略記．RGBAがそれぞれレッド，グリーン，ブルー，アルファを表す.

ポリゴン　3次元グラフィクスにおいて物体を構成するための最小単位となる多角形平面．頂点を指定することによって作成する.

テクスチャ　模様のデータ．ポリゴンに貼り付けて用いる．テクスチャをポリゴンに貼り付けることをテクスチャマッピングという．1次元か2次元あるいは3次元の配列でフラグメントのカラー値を決定するのに用いられる．2次元テクスチャは2次元の画像データであり，3次元テクスチャは本研究ではボクセル集合を表現するのに使用されている.

テクセル　テクスチャを構成する要素．テクスチャにとってのピクセルのようなもの.

スライス　テクスチャベースボリュームレンダリングにおいてはボリュームデータを視線に対して垂直な断面の重ね合わせにより可視化する．この断面の1つ1つをスライスと呼ぶ．グラフィクスカードを用いる場合には，スライス状のポリゴンにボクセル集合としての3次元テクスチャをマッピングすることでスライスを生成するが，本研究ではマッピングによってスライスを生成するために作成されたスライス状ポリゴンのこともスライスと呼ぶ

こととする．

ラスタライズ　ポリゴンをフラグメントに分解する作業．

フラグメント　ポリゴンを投影されるスクリーン上のピクセルに対応する部分に分解したもの．

フレームバッファ　レンダリングしてスクリーン上に投影された画像データを格納しておくためのグラフィクスカードのメモリ内の領域．ここから読み出された情報がディスプレイに表示される．

オフスクリーンバッファ　レンダリングした画像をディスプレイ上に表示させたくない場合などに，フレームバッファの代わりにレンダリングした画像のデータを格納しておくためのメモリ内の領域．

2.2　ボリュームデータの可視化

2.2.1　ボリュームデータ

連続なボリュームデータは，$\boldsymbol{R}$ を実数全体の集合，n を 1 以上の整数として，$\boldsymbol{R}^3$ から $\boldsymbol{R}^n$ への写像として表現できる．ここでの値域はマルチスカラ量であり，空間の各点が持つ物理的なフィールドに関係することからフィールド値と呼ばれる．本研究で用いたデータは，全て単一スカラのフィールド値を持つものである．

このようなボリュームデータを計算機上で扱うには，定義域を離散化してサンプリング点を取り，各サンプリング点上でのフィールド値を量子化するのが一般的である．このような離散化処理によって得られたボリューム空間の最小単位をボクセルと呼ぶ．また，元のボリューム空間全体を表現するボクセルの集まりを，ボクセル集合と呼ぶ．本研究では，空間の各軸に沿って均等にボリュームデータを離散化した，規則的ボクセル集合を用いた．

2.2.2　ボクセル集合の可視化方式

ボクセル集合を 2 次元ディスプレイ上に表示する手法には，大きく分けて間接方式と直接方式がある．

間接方式は，ボリュームの定義域や値域に制限を設けて部分ボリュームを間接的に可視化するものである．間接方式には，ボリュームの断面を表示する断面生成や，ボリュームの特定のフィールド値に写像される定義域の部分集合を求める等値面化などがある．

また，断面や等値面などの幾何学的データ構造を用いずに可視化する方式を

直接方式という．ボリュームレンダリングは直接方式の可視化手法である．これは無限枚の半透明等値面を合成するに等しい手法であり，ボリューム全体を観察することができる．

2.2.3 ボリュームレンダリング

ボリュームレンダリングとは，ボリュームデータを半透明なゲル状の物体と見なすことにより，空間の各点の厳密な値を見せることを断念するかわりに，ボリューム全体を多色ゼリーのように直観的に見せるものである．有限な計算でこの視覚効果を生むためには，ボリューム空間内で十分な点数の再サンプリングを行い，そこでのフィールド値を伝達関数を用いて適当なカラー値と不透明度値に変換した後，スクリーンに順次投影し混合する．

ボリュームを投影する手法には大きく分けて前方投影と後方投影の 2 種類がある．前方投影は空間内の再サンプリング点をスクリーンに遠いものから順に投影していき，投影点上のカラーに再サンプリング点のカラーを混合していく手法である．後方投影はレイキャスティング法と呼ばれる手法であり，本研究ではこの手法を用いてボリュームを投影した．

2.2.4 レイキャスティング法を用いたボリュームレンダリング

レイキャスティング法では，スクリーンの各ピクセルから発せられた光線（視線ベクトル）上に一定間隔の再サンプリング点をとり，そこでの不透明度値を重みとし，カラー値を加重平均して対応ピクセルのカラー値とする．

再サンプリング点をとる際のフィールド値の計算手法には離散的な手法と連続的な手法がある．前者では，再サンプリング点から最も近いボクセルの値をフィールド値とする．後者では，再サンプリング点の周囲 8 ボクセルの値を用いて 3 重線形補間を行い，フィールド値を求める．本研究では連続的な手法を用いている（図 1）．

2.2.5 平行投影と透視投影

ボリュームデータをスクリーン上に投影する手法には，平行投影と透視投影がある．平行投影では，視線ベクトルはスクリーンと垂直な方向を向いており，全ての視線ベクトルが空間内で平行に位置している．これに対し，透視投影では，遠近法を使用した描画をするために，視線ベクトルは放射状に伸びており，互いに平行ではない．平行投影では視点の近くにあるものも，遠くにあるものも同じ大きさに見えるが，透視投影では近くにあるものは大きく，遠くにあるものは小さく見える．CPU 上でソフトウェアを用いてグラフィクス処理を行う

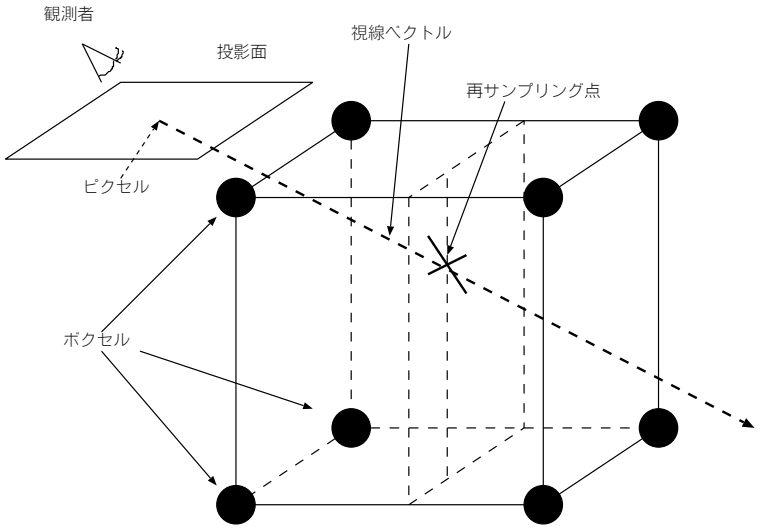


図 1: レイキャスティング法による連続的手法を用いた再サンプリング

場合は，透視投影のほうが処理に大きく時間がかかることが多いが，グラフィクスカード上で処理する場合には，透視投影がハードウェアでサポートされているため，平行投影と比べてもほぼ計算時間に差はない．そのため，本研究では平行投影の場合においてのみ精度制御の評価を行っている．

2.3 テクスチャベースボリュームレンダリング

ボリュームレンダリング処理の実装方法としては，CPU 上のソフトウェアによる実装，ボリュームレンダリング専用のハードウェアを用いた実装，そして汎用グラフィクスカードを用いた実装の 3 つが考えられている．本研究では，汎用グラフィクスカードを用いたテクスチャベースボリュームレンダリングによってボリュームレンダリングを実装した．テクスチャベースボリュームレンダリングとは，最近の汎用グラフィクスカードが持つテクスチャマッピング機能を用いてボリュームレンダリングを行う手法である．

汎用グラフィクスカード上での 3 次元データのレンダリングは，頂点座標を指定してポリゴンを作成し，スクリーンのピクセルに対応する部分に分解（ラスタライズ）した後，テクスチャマッピングやアルファブレンディングなどピクセルごとの処理をするという流れになる．これら一連の処理は，ポリゴンの作成は座標計算パイプライン，ラスタライズ処理はラスタライザ，ピクセルごとの処理はパラレルレンダリングパイプラインによって行われる [1].

テクスチャベースボリュームレンダリングでは，ボリュームを視線方向に垂

直なスライスの重ね合わせにより可視化する．ボリュームデータを 3 次元テクスチャとして表現し，これをスライス状のポリゴンにマッピングする．それらのスライスを視点に遠いものから順に不透明度を表すアルファ値によって足し合わせ，画像を出力する（図 2）．

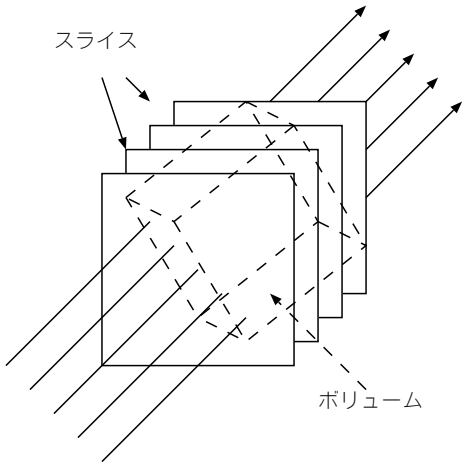


図 2: テクスチャベースボリュームレンダリング

テクスチャベースボリュームレンダリングにおいては，RGBA 値（カラー値とアルファ値）は 3 次元テクスチャデータを構築する際に計算する．よって，テクスチャデータの大きさは元となったボクセル集合の 4 倍のサイズになるが，レンダリング中に伝達関数を用いたテクスチャマッピング処理の計算をしなくてよい．

2.4 ボリュームレンダリングの視点変更

レイキャスティング法でボリュームレンダリングする際には，視線の向きが描画速度に大きな影響を及ぼす．なぜならば，ボリュームのメモリ内でのデータの並びが一定だからである．従って，ある方向から視線ベクトルを飛ばした場合はメモリ参照の局所性が大きくキャッシュを大いに活用できるが，別の方向から見た場合にはキャッシュが全く使われないという可能性がある．

テクスチャベースボリュームレンダリングでも同様の現象が起こる．ポリゴンにテクスチャをマッピングするとき，マッピングに使われるテクスチャの領域がグラフィクスカードのメモリ内で連続している場合はキャッシュを活用できるが，別の方向に視点を移動するとキャッシュがほとんど役に立たず，急激に

描画速度が低下する．1fps 以下程度の描画速度ならば，連続して視点移動しようとすると計算に膨大な時間が必要となり，しばらくのあいだ操作不能に陥ってしまう．従来ではこのような状態を回避するために，バウンディングボックスの表示などの手法が用いられてきた．バウンディングボックスとはオブジェクト同士の衝突判定に用いられる技術で，オブジェクトとその外側の境界面を表す [8]．視点が移動している間はバウンディングボックスのみを表示することで，描画を高速化することができる．また，OpenGL においては処理が間に合わない場合は表示すべき画像の計算を行わないようにする機能がある [2]．しかし，バウンディングボックスの手法ではボリュームデータ自体の外観が認識できない．また，OpenGL の場合でも描画が遅いと表示すべき画像の大部分が抜け落ちて見た目が不自然になってしまう．

本研究では視線の向きによらないメモリの参照局所性の均一化については触れていない．参照局所性を均一化する研究はすでにされていて，それを実装した状態においてまだ上記の問題が解決できない場合について考える．ここで，人間の知覚特性の 1 つとして動いている物体に対しては，静止している物体に比べて認識精度が低いという性質を利用し，視点が移動する速度に応じて出力画像の精度を変える手法を考える．この手法を用いることによって，画質と描画速度のトレードオフを柔軟に設定することが可能になる．

2.5　2 次元画像の精度制御

2 次元ビットマップイメージの表示方法の 1 つに，プログレッシブレンダリングというものがある [9]．これは，インターネット上の画像配信によく用いられる技術の 1 つで，画像全体をまず粗い解像度で表示し，徐々に精細に表示していく．これにより，通信速度が遅くても，最初にイメージ全体が表示されるので，全体像を素早く認識できるようになる．

また，3 次元のポリゴンレンダリングにおいて LOD 制御という技術がある [10]．これは，物体が視点から遠いとき，表示されるピクセル数に対し頂点数が過剰である場合（表示領域が狭すぎて，画像がつぶれて見えない場合など）に用いられる．ポリゴンの作成は視点からの距離に関わらず頂点数で計算量が決まるため，頂点の数を減らして少ないポリゴン数で物体を構成する．これにより，画質をほとんど下げることなく計算量を減らすことができる．さらに，ポリゴンにマッピングする 2 次元テクスチャに対しても同様で，視点から遠いと

きはテクセル数を減らしたテクスチャに置き換えるミップマップという技術がある．そしてもう 1 つ，注視点とその領域外でレンダリングの精度を変えるという手法もある．

これらの技術は，ボリュームレンダリングに関するものではないが，精度を変更するという点において本研究と関連がある．従って，これらの技術を応用して精度制御に適用することが可能であると考えられる．2 次元画像のプログレッシブレンダリングは，2 次元画像のサンプリング率を変えるということである．これは，ボリュームレンダリングにおいては再サンプリング率を変えるという手法でそのまま適用できる．ただし，2 次元画像においてはスクリーン方向のサンプリング率を変えるだけだったが，ボリュームレンダリングにおいては奥行き方向にも再サンプリング率を変えることができる．しかし，奥行き方向の再サンプリング率の変更は，視線の向きと垂直になるので，スクリーン方向の再サンプリング率の変更とは性質が大きく異なる．この違いについては 4.2 で述べる．また，LOD 制御は遠さによって詳細度を変えるが，アルファ値によってレンダリングの精度を変えることでボリュームレンダリングに適用できる．そして，ミップマップの手法はボリュームの再サンプリング率に合わせてボリュームデータの解像度を変えることで適用が可能である．さらに，注視点と領域外で精度を変える手法は，その他の手法に対して独立に適用可能である．また，2 次元画像の精度制御には使用できないボリュームレンダリング独特の精度制御の手法として，Early Ray Termination（以下 ERT）を利用した手法がある．ERT を用いる場合は，視線ベクトルを飛ばしたとき，視点に近いほうから再サンプリング点を計算していく．再サンプリング点を取っていくと，不透明度が徐々に 0 に近づいていくので，不透明度が 0 に近い一定の値以下になったとき，そこで再サンプリング点を取るのをやめる．そして，そこまでの計算結果をピクセル値として保存する．これを利用して，どの程度不透明度値が 0 に近づいたら計算をやめるかで精度制御ができる．

第 3 章　精度制御の手法

本研究では，ボリュームデータの再サンプリング率を利用した手法と，ボリュームデータの解像度を利用した手法の 2 つの手法を実装した．以下に各手法について説明する．

3.1　ボリュームデータの再サンプリング率を利用した手法

ボリュームデータが投影されるスクリーン平面を xy 平面に平行であるとし，それに垂直な視線ベクトルの方向を z 軸に平行であるとする．このとき，x,y,z 各軸方向の再サンプリング率をそれぞれ R_{s_x},R_{s_y},R_{s_z} と表記することとする．

レイキャスティング法において R_{s_z} を変えるためには，視線ベクトル方向の再サンプリング率を変えればよい．しかし，ただ再サンプリング率を下げるだけだと不透明度が全体的に低くなってしまうため，不透明度に関して補正を行わなければならない．このためには，伝達関数を変えるか，アルファブレンディングの計算方法を変えるなどの必要がある．また，(R_{s_x},R_{s_y}) を変える手法は，視線ベクトル数を変えることで実現できる．

3.2　ボリュームデータの解像度を利用した手法

上記 2 種の手法は再サンプリング点の数を減らすことにより計算量を減らしてきた．別のアプローチとして，ボリュームデータ自体のボクセル数を減らす手法が考えられる．ボリュームデータのサイズを小さくすると，一度にキャッシュ・メモリに格納されるデータ量のボリューム全体に対する割合が増加し，メモリアクセスの回数が減少して描画を高速化することができる．

また，再サンプリング点でのフィールド値の算出方法は周囲 8 ボクセルの 3 重線形補間を用いたので，R_{s_x},R_{s_y},R_{s_z} による手法を使用した場合，全く参照されないボクセルが存在する可能性がある．そのような場合，画像の出力には参照されないボクセル値のデータは全く用いられていないことになる．このとき，ボリュームの解像度を下げる場合に，隣接するボクセルの平均を取って 1 つのボクセルとすると，参照されないデータがなくなり，出力画像も本来のものに近くなると考えられる．従って，この手法を用いるときは前述の 2 種類の手法に併せて用いるのがよい場合が多い．

しかし，この手法はメモリを余分に消費してしまうという点で問題がある．各辺 $\frac{1}{2}$ のサイズのデータを作成すると，元にしたデータの 12.5%余分にメモリを消費することになる．よって，もしメモリが足りない場合や余分なメモリを消費したくない場合にはこの手法を用いることはできない．

この手法は，キャッシュのサイズなどハードウェアの特徴にその効果が依存する．また，元からボリュームデータのサイズが小さい場合にはほとんど効果

がない可能性がある．

以下，この手法をボリューム解像度による手法と呼ぶ（図 3）．

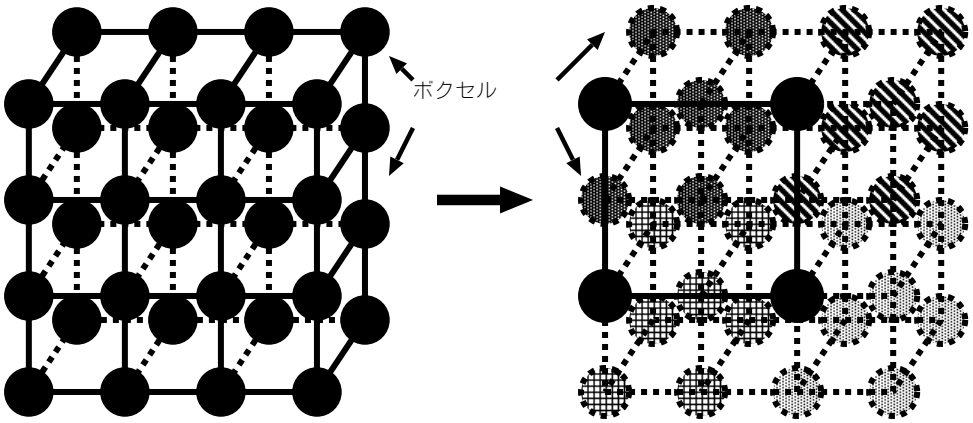


図 3: ボリューム解像度による手法

3.3 Early Ray Termination を利用した手法

ERT の Terminate 率によって精度制御が可能である．

この手法は，全体的に不透明度が高くなるような伝達関数を用いている場合，効果が大きくなると考えられる．

3.4 精度制御手法の組み合わせ

R_{s_x},R_{s_y},R_{s_z} による手法，ボリューム解像度による手法，そして ERT による手法は，互いに独立に実装できるので，組み合わせて用いることが可能である．また，注視点とその領域外で精度を変える手法や，不透明度が高くなるほど精度を低くする手法は，R_{s_x},R_{s_y},R_{s_z} と ERT による手法の応用として用いることができると考えられる．

3.5 視点の移動速度と精度制御

本研究では視点の移動速度に応じてレンダリングの精度を変えるので，移動速度によってどのようにレンダリングの精度を変えればよいか考える必要がある．

人間の目に自然に見える描画速度は 30fps 程度であると言われている．従って，静止している間は最高精度でレンダリングし，視点が移動しているときは 30fps 程度の描画速度が出るように精度を落とす手法が考えられる．また，ゆっ

くり動いているときは 30fps よりも遅い描画速度でもいいからより高い精度でレンダリングするという手法も考えられる．

第 4 章　実装

4.1　テクスチャベースボリュームレンダリングの実装

本研究では汎用グラフィクスカード上でグラフィクス処理を行うための汎用 API である OpenGL を用いてテクスチャベースボリュームレンダリングを実装した．テクスチャベースボリュームレンダリングはグラフィクスカード上で以下のように実装されている．

テクスチャマッピングされてスライスとなるためのポリゴンは視点に遠いものから座標計算パイプラインで作成され，ラスタライザでラスタライズされてフラグメントとなる．このフラグメントが再サンプリング点を表している．フラグメントはボクセル集合を表す 3 次元テクスチャのマッピングによって，その RGBA 値が決定される．このフラグメントの RGBA 値は，再サンプリング点のフィールド値から伝達関数で計算されたカラー値と不透明度値を表す．各フラグメントは対応するピクセルごとに，パラレルレンダリングパイプラインによって視点に遠いものから順にアルファ値を用いて混合処理され，出力画像のピクセルデータとなる．ピクセルデータはグラフィクスカードのメモリ内のフレームバッファという領域に保存される．フラグメントの混合処理は，新規に入ってくるフラグメント（ソース）の RGBA 値と，現在フレームバッファに保存されている対応するピクセル（デスティネーション）の RGBA 値，そして混合係数を用いた演算により行われる．具体的には，ソースとデスティネーションの RGBA 値をそれぞれ (R_s, G_s, B_s, A_s),(R_d, G_d, B_d, A_d) とし，ソースとデスティネーションの混合係数を各々(S_r, S_g, S_b, S_a),(D_r, D_g, D_b, D_a) とすると，

$$(R_sS_r + R_dD_r, G_sS_g + G_dD_g, B_sS_b + B_dD_b, A_sS_a + A_dD_a)$$

が混合処理後のデスティネーションの新たな RGBA 値となる．このとき，ソースのアルファ値が 0 ならばフラグメントは拒否され，混合の計算は行われない（付録 A.2 参照）．全てのフラグメントの混合処理が終了すると，最終的な画像が出力される．

なお，スライスは 3 次元テクスチャが回転したとき端点の描き残しがないように，3 次元テクスチャの最も長い辺の $\sqrt{3}$ 倍の長さをもつ正方形とする．ま

た，最も手前のスライスから最も奥のスライスまでの距離は，スライスの一辺の長さと等しいとする．しかし，最終的にはテクスチャがマッピングされない領域にあるポリゴンは，クリッピングしてレンダリングされないようにしている．よって，仮にスライス数が 512 枚ならば実際に 3 次元テクスチャデータがマッピングされるスライスの数は，スライスが 3 次元テクスチャのいずれかの面に対し平行なときに最小で 295 枚であり，そこから xyz 各軸方向に 45 度回転したときに最大で 512 枚である．従って，256^3 の大きさを持つデータならばスライス 512 枚程度で視線ベクトル方向の再サンプリング数とボクセル数が同数に近くなる．

また，OpenGL の制約でテクスチャは各辺が 2 の累乗であることが推奨されている．従って，ボリュームデータには各辺が 2 の累乗であるものを用いるのがよい．

4.2 R_{s_x},R_{s_y},R_{s_z} による手法

R_{s_z} による手法と (R_{s_x},R_{s_y}) による手法は実装方法が大きく異なるので，以下に項目を分けて説明する．

4.2.1 R_{s_z} による手法

R_{s_z} による手法をテクスチャベースボリュームレンダリングで実装するには，スライスの数を減らせばよい（図 4）．

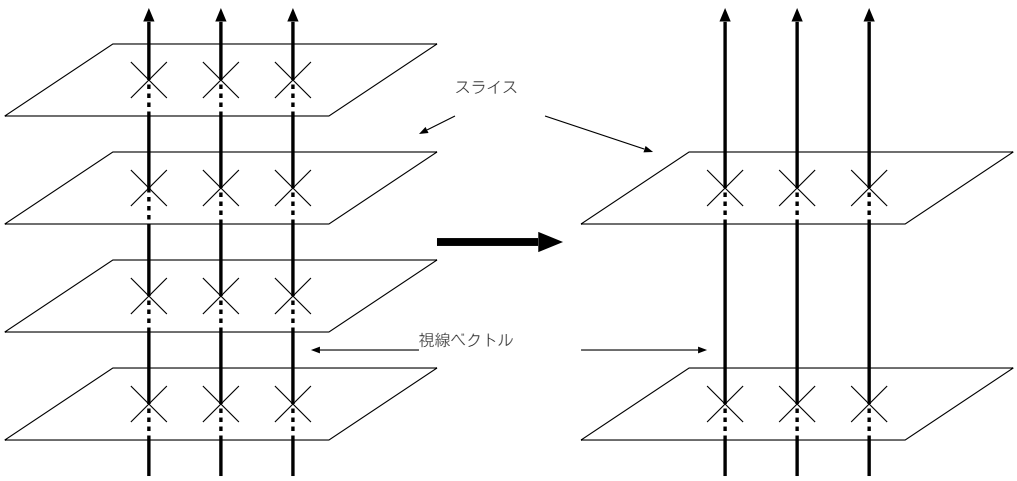


図 4: R_{s_z} による手法

しかし，この手法では単にスライス数を減らすだけでは全体的に透明度が減少してしまうという問題がある．フラグメントの混合処理をするときに，同じスライスが 2 枚あるのと同じように計算するようにすれば，透明度が低いという問題は解決できると考えられる．このためには，ボリュームデータとしてのテクスチャ自体の RGBA 値をスライス数に合わせて変えたものを用意しておくか，混合処理の際の混合係数を変更するという方法が考えられる．前者の場合，テクスチャデータを複数用意しておくのは可能であるが，ただでさえ大きいボリュームデータを複数保持しておくのは非常にメモリ効率が悪いうえ，スライスの枚数に対する制限が大きくなってしまう．さらに，ボリューム解像度による手法と組み合わせた場合には非常に多種類のテクスチャデータが必要となるので，この方法は現実的ではない．また，後者の場合も，混合係数は極めて単純なものしかサポートされていないので，スライス数に合わせて適切に混合係数を変更するのは難しい．

本研究ではこの問題はまだ解決されていないため，R_{s_z} による手法を用いて精度を下げたときの出力画像は全体的に透明度が高い．従ってこの手法はまだ実装が不完全である．しかし，混合係数を変更するのみで実装できた場合は現在の不完全な実装と描画時間は変わらないと考えられるので，不完全ではあるが画像と描画時間を出力し，評価した．

4.2.2 (R_{s_x},R_{s_y}) による手法

OpenGL では出力画像のピクセル数がそのまま視線ベクトル数になるため，出力画像を表示するウィンドウのサイズを小さくすることで単純に視線ベクトルの数を減らすことができる．しかし，ピクセル数を減らしたままディスプレイに表示しようとすると，出力画像のサイズが小さくなる．このままでは精度を変えると出力画像の大きさも変化してしまうので，出力画像の拡大処理を行わなければならない．

出力画像の大きさを変えずに解像度のみを下げる手法として，本研究ではオフスクリーンバッファというメモリ領域を用いた．OpenGL では出力画像に関するデータは本来フレームバッファに書き込まれる．しかし，オフスクリーンバッファを用いることにより，フレームバッファに書き込む内容をオフスクリーンバッファにリダイレクトすることができる．そして，一旦オフスクリーンバッファに書き込まれた画像を 2 次元テクスチャとして取り込み，新たに作成した矩形ポリゴンにマッピングしてこれを出力画像としてフレームバッファに書き

込む．これにより，オフスクリーンバッファを経由するオーバーヘッドはあるものの，出力画像のサイズを変更することなしに解像度のみを下げることができる（図5）．

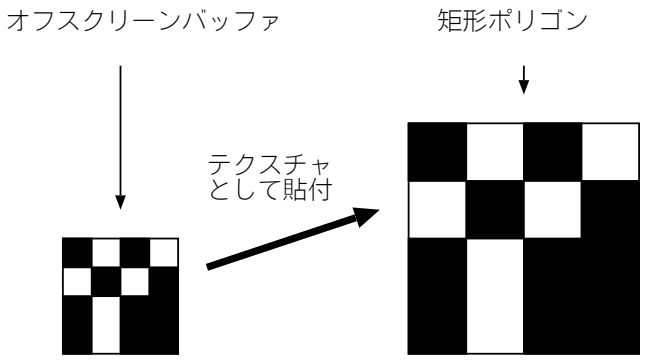


図5: オフスクリーンバッファを用いた (R_{s_x},R_{s_y}) による手法

なお，2次元テクスチャとして用いるビットマップ画像データは，各辺が2の累乗であったほうがよいので，オフスクリーンバッファに書き込む際の画像サイズも各辺を2の累乗とする．

また，テクスチャとして取り込んだオフスクリーンバッファの内容を矩形ポリゴンにマッピングする際には，拡大処理をする必要がある．このときより滑らかな外観を得るため，ピクセル値にはテクスチャのそのピクセルに対応する位置に最も近い 2×2 のテクセル配列の2重線形補間を用いた．

4.3 ボリューム解像度による手法

ボリューム解像度による手法を用いる場合は，ボリュームとしての3次元テクスチャデータをメインメモリから読み込んだあと，初期処理で解像度の低い3次元テクスチャデータをCPUによって作成し，グラフィクスカードのメモリ内に格納しておく．3次元テクスチャのどれを用いて描画するかによって精度を変える．

元テクスチャの各辺は2の累乗であると想定して，解像度の低いボリュームデータを作成する方法は，各辺を2の累乗とするために，xyzの各軸方向に隣接する8ボクセルを1つのボクセルとすることで，$(\frac{1}{2})^3$ のデータを作ることを繰り返す．このとき，8つのボクセルをどのような計算で1つのボクセルにするかによって出力画像の見え方が変化する．典型的には8つのうち1つを代表値と

する手法と，8 つのボクセルの平均値をとる手法がある．前者の場合，代表値を除く 7 つのボクセルデータが損なわれてしまう．また後者の場合，物理的な境界などを表す部分で，隣り合うボクセル同士の値が大きく異なるとき，平均することで中間の値になってしまって境界があいまいになる可能性がある．特にボクセル値 0 のデータとそれ以外のデータが明確に分離されている場合，物体の表面のフィールド値が減少してしまう現象が起こる可能性が高い．これを回避したい場合，8 ボクセルのうち最大値を代表値とする手法や，8 ボクセルの中にボクセル値 0 のボクセルを含む場合で 8 つ全てのボクセルがボクセル値 0 でない場合にはボクセル値 0 のボクセルを除いたボクセルでのみ平均値をとって新たなボクセル値とする手法などが考えられる．前者の場合，ボクセル値 0 以外のボクセルデータも失われ，逆に全体的にフィールド値が増加してしまう．また，後者の場合はボクセル値 0 との境界にしか使えず，それ以外の場合では同じ問題が残る．いずれにしても目的とデータに応じた手法を用いる必要があり，それによってはこの手法は全く利用できない可能性がある．

本研究ではサンプルデータによって用いる方法を変えた．どのデータにどの方法を用いたかは 5.1.3 に示す．

4.4 Early Ray Termination を利用した手法

OpenGL を用いて ERT を実装する場合には，フラグメントの混合計算をする際に現在のフレームバッファのアルファ値を調べる必要があると考えられる．現在のフレームバッファの内容によってフラグメントの取捨選択をするのは現在では複雑で実装が難しく，また実装できたとしてもオーバーヘッドが大きすぎて ERT による描画速度向上の効果が出ない可能性がある．従って，本研究では ERT による手法は実装していない．しかし，CPU 上のソフトウェアを用いてボリュームレンダリングする場合には，簡単に実装できて有効な方法であると考えられる．

4.5 その他の手法

不透明度が高くなるほど精度を低くする手法については，ERT と同じ理由から実装していない．また，注視点とその領域外で精度を変える手法についても，本研究ではまだ実装されていない．

本研究では，R_{s_x},R_{s_y},R_{s_z} とボリューム解像度による手法やその組み合わせを

実装した.

4.6 視点の移動速度と精度制御

本研究では，マウスをドラッグする間のマウスポインタの位置の変位量とボリュームの回転角が比例するような OpenGL プログラムを用いてボリュームレンダリングを行った．ボリュームの回転速度に応じて精度を変更する方法としては，最大で 30fps 程度出るようにし，視点が動いている間は常に 30fps とするものと，視点の移動速度が速いときは最大で 30fps で，移動速度が遅くなると徐々に精度をよくしていくものを実装した.

4.7 グラフィクスカード

グラフィクスカードとして Radeon9700pro を搭載した計算機にて精度制御を実装した．Radeon9700pro は ATI 社のグラフィクスカードであり，その主な性能を表 1 に示す [5].

表 1: Radeon9700pro の性能

メモリ	DDR-SDRAM 128MB
メモリクロック	620MHz
メモリインターフェイス	256bit
メモリバンド幅	19.8GB/sec
コアクロック	325MHz
パラレルレンダリングパイプライン数	8
座標計算パイプライン数	4

4.8 サンプルデータ

レンダリングに用いたボリュームデータは以下のとおりである．各データのボクセル値は 8bit で表される.

skull サイズ 128^3，人体頭部の CRT のボリュームデータ．3 次元テクスチャとして格納した場合のメモリ内でのサイズは 8MB．glut[1)] のサンプルデータ

[1)] http://www.opengl.org/resources/libraries/glut.html

の CRT を元に作成した．

bonsai　サイズ 256^3，分厚い土の部分と細い枝，薄い葉からなる盆栽のボリュームデータ．テクスチャサイズ 64MB．volvis[1] より利用させて頂いた．

poisson　サイズ 128^3，立方体の各面に温度を与えたときの熱伝導の様子を可視化したもの．なだらかに値の変化するボリュームデータ．テクスチャサイズ 8MB．ポアソン方程式より作成した．

engine　サイズ $256^2 \times 128$，エンジンのデータ．ボクセル値の変化が少ない．テクスチャサイズ 32MB．volvis より利用させて頂いた．

mrbrain　サイズ $256^2 \times 128$，人体頭部の MRI のボリュームデータ．ボクセル値は小さな幅の中に密集している．テクスチャサイズ 32MB．

poisson を除くと，サンプルデータは中心にオブジェクトがあり，その外側はボクセル値 0 のボクセルで占められている．poisson はボリューム空間のほとんどがボクセル値が 0 でないボクセルで占められている．

グラフィクスカードのメモリは 128MB なので，bonsai などの 256^3 のサイズのデータは格納できる．しかし，512^3 のサイズのデータになると入りきらなくなるので，ボリュームデータを分割して 1 台で順番にレンダリングするか，並列化してレンダリングする必要がある．

本来 512^3 程度のメモリに入りきらないような大きなデータでこそ，描画速度が非常に遅くなるので精度制御が有効なのだが，本研究ではボリュームを分割してレンダリングする機能や並列化を実装していないので，そのような大きなデータは扱っていない．

第 5 章　結果と評価

5.1　結果

以下に，ウィンドウサイズを 512^2 としたときの，各手法による描画時間の結果と出力画像を示す．以下では，スライスの数を S_z，スクリーンの解像度（ピクセルの数，すなわち視線ベクトルの数）を S_{xy}，ボリュームデータの解像度を表すために，元データ（4.8 で示したデータ）に対する各辺の比（bonsai なら 256^3 のとき 1^3，128^3 のとき $(\frac{1}{2})^3$ と表記する）を R_v と表記する．

[1] http://www.volvis.org/

5.1.1 描画時間の測定方法

メモリの参照局所性が平均化されていることを想定して，さまざまな方向から見た場合の描画時間の平均を用いるため，以下のような手法で描画時間を測定する．

まず，スライスに平行に 3 次元テクスチャデータが並んでいて，最小の描画時間が出る向きから描画を開始する．そして，その向きから x,y 軸中心にそれぞれ $\frac{180}{256}$ 度ずつ回転させた画像を描画させる命令を 256 回繰り返す．最初の命令を発行させてから，実際に全ての描画が終了するまでの時間を 256 で割り，描画時間とする．直観的には x,y 軸中心にそれぞれ同じ速さで 180 度ずつ回転させる間の平均描画時間と同じである．

5.1.2 描画時間

表 2,3 に，R_{s_z},(R_{s_x},R_{s_y}) による手法の描画時間を示す．また，表 4 にボリューム解像度による手法の描画時間を示す．

表 2: S_z と描画時間（S_{xy}:512^2,R_v:1^3）

S_z	描画時間 (ミリ秒)				
	skull	bonsai	poisson	engine	mrbrain
512	66.3	185	75.8	93.1	95.2
384	49.8	139	57.6	69.9	71.6
256	33.2	92.4	38.5	46.5	47.8
128	16.8	46.5	19.4	23.6	24.0

表 3: S_{xy} と描画時間（S_z:512,R_v:1^3）

S_{xy}	描画時間 (ミリ秒)				
	skull	bonsai	poisson	engine	mrbrain
512^2	101	218	110	127	130
256^2	60.2	134	62.8	72.4	76.3
128^2	38.9	57.6	39.4	31.0	37.3
64^2	17.2	13.7	17.2	8.43	10.2

表 4: R_v と描画時間（S_z:512,S_{xy}:256^2）

R_v	描画時間 (ミリ秒)				
	skull	bonsai	poisson	engine	mrbrain
1^3	66.3	185	75.8	93.1	92.4
$(\frac{1}{2})^3$	29.8	67.5	38.5	34.6	34.5
$(\frac{1}{4})^3$	24.3	30.3	30.3	15.8	15.7
$(\frac{1}{8})^3$	23.3	24.9	28.4	13.0	12.8

5.1.3 出力画像

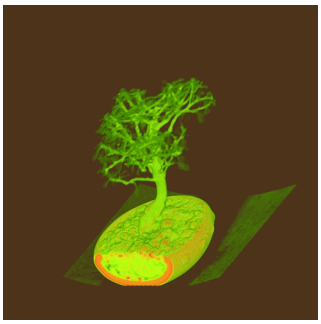
bonsai に各精度制御の手法を実装したときの出力画像を図 6,7,8 に示す．分量が多いので他のデータの出力画像は付録 A.1 に記載する．なお，ボリューム解像度による手法において，8 ボクセルを 1 ボクセルにする際の計算方法については，bonsai,poisson は 8 ボクセルの平均値を，その他は 8 ボクセルの内，ボクセル値が 0 でないボクセルのみの平均値を使用した．bonsai, poisson 以外は物体の表面のフィールド値が下がらないようにするための計算方法である．poisson は全体的に一様になだらかな変化をするボクセル値を持つので平均値を用いた．bonsai は植木鉢や土，幹の表面などにも薄い部分がたくさんあり，ボクセル値 0 以外の平均値を用いると著しく画質が劣化したので，単に平均値を用いた．従って bonsai の図 8 では表面が黒ずんで見える．

5.2 評価

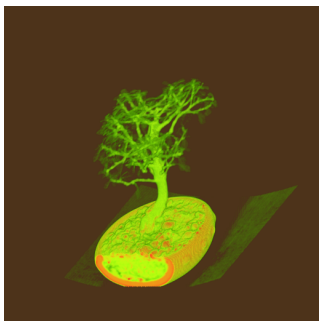
5.2.1 データごとの描画時間

まず初めに精度制御をしない場合の描画時間について各データで比較する．表 2 の S_z:512 の行を比較すると，skull が最も小さく，次に poisson，engine，mrbrain，bonsai と続く．ボリュームデータのサイズが小さいほうが描画が速いのは，1 度のメモリアクセスでキャッシュに格納できるデータ量がボリュームデータ全体に対して大きいためメモリアクセスの回数が少なくなるからと考えられる．ボリュームのサイズが同じならば，同じ条件でレンダリングすると，アルファ値が 0 となるフラグメントの数によって描画時間が変化する（付録 A.2 参照）．skull の描画が poisson より速いのは，アルファ値 0 のフラグメントが poisson より遥かに多いからである．

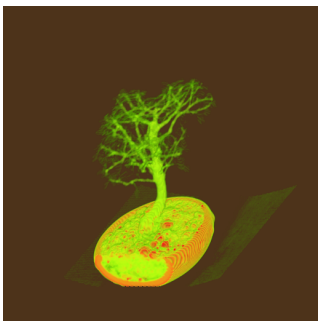
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:384,S_{xy}:512^2,R_v:1^3



(c)S_z:256,S_{xy}:512^2,R_v:1^3



(d)S_z:128,S_{xy}:512^2,R_v:1^3

図 6: S_z と出力画像 (bonsai)

(a)S_z:512,S_{xy}:512^2,R_v:1^3

(b)S_z:512,S_{xy}:256^2,R_v:1^3

(c)S_z:512,S_{xy}:128^2,R_v:1^3

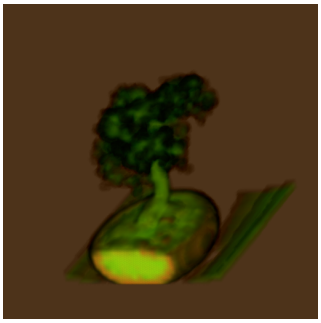
(d)S_z:512,S_{xy}:64^2,R_v:1^3

図 7: S_{xy} と出力画像 (bonsai)

(a)S_z:512,S_{xy}:512^2,R_v:1^3

(b)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{2})^3$

(c)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{4})^3$



(d)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{8})^3$

図 8: R_v と出力画像 (bonsai)

また engine や mrbrain はスライスの配置されている空間に対して他の立方体のデータよりもマッピングされる領域が小さいので，データの大きさに比べて描画時間は小さい．engine や mrbrain は非立方体のボリュームデータであり，その各辺の比から立方体のデータに比べてマッピングされる領域は $\frac{1}{2}$ になる．engine の場合や mrbrain はマッピングされる領域は半分であるが，スライスの元となるポリゴンは一度作成された後にマッピングされない部分のみが消去されるので，単にスライスを半分に減らすよりは計算量が多い．

5.2.2 R_{s_z} による手法

描画時間の面では，表 2 において S_z と描画時間を比べてみると正比例関係にあることがわかる．スライスを半分にすることは描画されるポリゴンが半分になることである．ラスタライズやアルファブレンディングの計算量も半分になる．従って，単純に描画時間が半分になることは容易に予測できることである．これは，データの種類には依存しない性質である．

次に，各データによる出力画像とスライス数の関係を考察する．bonsai や skull の S_z:128 の図（図 6, 図 A.1）を見ると確認できるが，スライスの不足によりエ

イリアシングアーチファクトによる縞模様が見える．このエイリアシングはボリュームデータの表面と中身の値が大きく違うようなデータに顕著に見られる．bonsai は植木鉢や幹の中身が剥き出しになり，また葉がいくらか減少している．skull のほうでは頭蓋骨の耳の辺りが見えなくなって中の空洞が見えている．しかし，engine や mrbrain のようなボリュームはだいたい均一なデータを持っているので画質の劣化が少ない．poisson もなだらかに変化する部分は問題ないが，赤い面の近くなどは色が急激な変化を持つのでその辺りにはエイリアシングがみられる．

つまり，描画時間の短縮の面ではデータによる差はないが，出力画像には画質の差が表れ，bonsai のような薄い部分を持つデータには特にこの手法は向いていない．逆に engine のようなほとんど均一なデータで構成されているボリュームには有効であると考えられる ．

5.2.3 (R_{s_x},R_{s_y}) による手法

描画時間については，解像度を $\frac{1}{2} \times \frac{1}{2}$ 下げるごとにピクセルごとの計算量が $\frac{1}{4}$ になるが，スライスの元となるポリゴンを作成やクリッピング処理に要する時間は解像度を下げても変わらないので単純に $\frac{1}{4}$ にはならない．また，オフスクリーンバッファを経由するオーバーヘッドも存在する．さらに，解像度を下げすぎるとピクセルごとの計算量がポリゴンの作成やクリッピング処理に比べて相対的に小さくなり，ほとんど描画時間が下がらなくなる．

また，(R_{s_x},R_{s_y}) による手法の効果はボリュームデータが空間上にマッピングされる際のボクセルの視体積に対する分布密度に関係がある（詳細については付録 A.3 参照）．そのため，表 3 において，skull・poisson と bonsai・engine・mrbrain とを比べると描画時間減少の効果が異なる．

オフスクリーンバッファを経由してレンダリングした際のオーバーヘッドは，表 2 の S_z:512 の行と表 3 の S_{xy}:512^2 の行を比較するとその差を確認できるが，S_{xy}:512^2 で約 34 ミリ秒と，無視できるほど小さなものではない．解像度が最も高い場合のみオフスクリーンバッファを用いないようにすることはできるが，解像度を低くするとこの方式ではやはり用いる必要がある．もしオフスクリーンバッファを用いることなしにスクリーン解像度を下げることができれば，さらに高速化できると考えられる．

また，bonsai や mrbrain の場合，解像度を下げると盆栽の葉や脳のしわなどの細かい部分の画質が劣化する様子が確認できる．しかし，skull や engine など

では解像度を最大から $\frac{1}{2} \times \frac{1}{2}$ に下げた程度では，ほとんど見た目が変わらない．視点が連続で変更される場合には，画質の劣化はほとんど確認できない．これは，skull や engine が，ボクセル値が小刻みに変化する部分をほとんど持たないためである．また，poisson では解像度を $\frac{1}{2} \times \frac{1}{2}$ に下げても，角度によっては静止していても肉眼では解像度が変化したことがほとんど確認できない．poisson のデータはフィールド値の変化が一様になだらかで 2 次元の補間がよく働くため，解像度を下げても非常に画質がよい．

従って，bonsai や mrbrain のような出力画像のピクセル値が小刻みに大きく変化するようなデータにはこの手法は向いておらず，poisson のように一様になだらかな変化をするフィールド値を持つデータには特に有効であるといえる．

5.2.4 ボリューム解像度による手法

この手法は単体で用いてもデータの劣化が激しく，ほとんど役に立たない．また，境界があいまいになるという問題もあるので，ボリューム解像度を大きく減少させると，元データに比べて著しく異なる画像が出力される場合が多くなる．従って，他の 2 種の手法を用いたときに再サンプリング点の数が少なくなりすぎた場合用いるのがよいと考えられる．また，この手法はボクセル値がなだらかに変化するデータ（隣り合うボクセル値が急激に変化することのないデータ）に対しては有効なので，poisson などのデータならば単体でも問題なく利用できると考えられる．

5.2.5 視点の移動速度と精度制御

以上では精度を変更したときの静止画に関する評価を行ってきた．本研究の目的は，視点移動速度に応じた精度制御であるので，動画に関する評価を行わなければならない．

視点移動速度に応じた精度制御として 4.6 に示した 2 種類の手法を，サンプル中でレンダリング時間が最も長い bonsai を用いて実装した．レンダリングにおける最高精度は S_z:512,S_{xy}:512^2,R_v:1^3，最低精度は S_z:256,S_{xy}:256^2,R_v:$(\frac{1}{2})^3$ とし，オフスクリーンバッファは常に用いてレンダリングすることとする．最高精度と最低精度での描画速度は，それぞれ 4.59fps,28.3fps である．さらに，中間精度を 1 つ用意し，S_z:384,S_{xy}:256^2,R_v:1^3 とする．このときの描画速度は 9.66fps である．

静止しているときは最高精度でレンダリングし，秒間 50°以上で回転しているときは最低精度でレンダリングする．回転速度が秒間 0~50°のとき，最低精

度でレンダリングする手法と，中間精度でレンダリングする手法について比較した．まず，秒間 50°以上で回転しているときは，回転の動きが速いため最低精度でも画質の劣化はあまり気にならなかった．また，動きもスムーズで，不自然な動きには見えなかった．秒間 0~50°のときは，最低精度でも物体が動いているため細部がよく見えないので，不自然には感じられなかった．しかし，中間精度を用いた場合は，1 枚 1 枚の画像は最低精度より画質がよいし，描画が遅くてマウスの動きについてこられないということはなかったが，動きがかくかくしているように見えて不自然さが感じられた．また，特に中間精度を用いない手法に言えることであるが，精度が突然変わる場面では強く不自然さが感じられた．よって，中間精度を用いない場合でも，描画がマウスについてこられないほど遅くないのであれば，徐々に精度を変えるようにするのがよいと考えられる．

まとめると，精度制御の手法は用途によるが，なるべく自然な動きを求めるのならば，視点が移動している間は常に 30fps 程度を保つのがよいように思われた．また，動きが不自然であってもよいから，描画速度がマウスの動きについてこられる程度の速さで，より出力画像の精度を求めるのであれば，30fps 未満の中間精度を用いるのもよいと考えられる．

第 6 章　並列化

メモリに入りきらないような大きなボリュームデータを高速に描画するためには，計算機を並列化してクラスタシステム上でボリュームレンダリングする必要がある．並列化を実装したときの例としては，ボリュームデータを分割して各ノード上でレンダリングする手法がある．最終的には各ノードでレンダリングされた画像を 1 つのノードに集めて合成し，出力画像とする．このとき，R_{s_z} による手法やボリューム解像度による手法の実装は変わらないが，(R_{s_x}, R_{s_y}) による手法を実装する際は，少し異なる．

最終画像を合成するノードをサーバーとして，残りのノードをクライアントとすると，クライアントでの出力画像はクライアントのディスプレイに表示されるため，サーバーを操作するユーザーには見えない（見えてもよい）．そのため，わざわざオフスクリーンバッファに解像度の低い画像を書き込む必要がなくなり，クライアントでレンダリングした画像をサーバーでテクスチャとし

て取り込んで拡大してマッピングすればよい．このとき，オフスクリーンバッファを用いるためのオーバーヘッドはクライアントからサーバーへの通信のオーバーヘッドに完全に置き換えられることになる．この通信のオーバーヘッドは他の手法を用いるときはもちろん，精度制御を用いないときでもかかってしまう．これは並列化のオーバーヘッドであるが，これのために (R_{s_x},R_{s_y}) による手法のオーバーヘッドが相殺され，相対的に他の手法に比べて (R_{s_x},R_{s_y}) による手法の有効性が上昇する．

また，現在の汎用グラフィクスカードのメモリに入りきるようなサイズのボリュームデータでは，精度を高くしても 5fps 程度の描画速度は出るので，精度制御が有効なのはむしろメモリに入りきらないような，並列化を必要とする巨大なボリュームデータということになる．この場合，ただでさえメモリ領域が不足しているから並列化しているのに，さらにメモリ領域を必要とするようなボリューム解像度による手法はナンセンスであり，有効でない可能性が高い．並列化の実装の際には，なるべくボリューム解像度による手法以外を組み合わせて精度制御できるようにするべきである．

本研究では，並列化してボリュームレンダリングする際の精度制御はまだ実装されていない．並列化を実装すると，通信時間がオフスクリーンバッファのオーバーヘッドと同様のものになると考えられる．ただし，通信にかかる時間はオフスクリーンバッファを経由するよりは大きくなると思われる．そのことを除けば並列化処理する場合と一台のマシンで処理する場合には大きな差はないと考えられる．

第 7 章　結論

人間の知覚特性として，動いている物体を認識する能力は静止している物体に対して低いということを利用して，視点移動速度に応じた精度制御をボリュームレンダリング処理に対して実装した．精度を変える手法として，R_{s_x},R_{s_y},R_{s_z} によるものと，ボリュームの解像度によるものを実装し，視点移動速度に応じた精度制御の手法として中間精度を用いるものと，最高精度と最低精度の 2 種類のみを用いるものを実装し，評価した．

静止画における評価については，(R_{s_x},R_{s_y}) による手法は $\frac{1}{2}\times\frac{1}{2}$ 程度ではほとんど画質の劣化がなく，さらに再サンプリング率を下げると大きく描画時間を

減少させることができ，有効な手段であった．また，R_{s_z} による手法も未完成ではあったがスライス数に対して線形にレンダリング時間を減少させることができた．さらに，ボリューム解像度による手法も，メモリに余裕があるならばという条件の下で，R_{s_x},R_{s_y},R_{s_z} と組み合わせて効果が期待できることが分かった．

動画における評価については，中間精度を用いる場合は動きに不自然さはあるが実時間で精度の高い出力画像を得られた．また，中間精度を用いない場合も，精度は低いが動きの滑らかな，自然な出力画像を得ることができることが分かった．

今後の課題としては，ERT など本研究で実装できなかった精度制御の実装や，グラフィクスカードのメモリ内に入りきらないような大きなデータで，ボリュームを分割して一台で逐次的にレンダリングする場合と，並列化処理でレンダリングする場合の精度制御の効果の違いについて調べたい．

謝辞

本研究の機会を与えてくださった富田眞治教授に深甚の謝意を表します．

また，本研究に関して適切なご指導を賜った中島康彦助教授，森眞一郎助教授，五島正裕助手，津邑公暁助手に心から感謝いたします．

さらに，日頃からご助力頂いた京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝いたします．

参考文献

[1] Woo, M. and Neider, J. and Davis, T.（アクロス訳）: OpenGL プログラミングガイド（原著第 2 版）, ピアソン・エデュケーション (1997).

[2] Angel, E.（滝沢徹, 牧野祐子訳）: OpenGL 入門, ピアソン・エデュケーション (2002).

[3] 中嶋正之, 藤代一成: コンピュータビジュアリゼーション, 共立出版 (2000).

[4] 千葉則茂, 土井章男: 3 次元 CG の基礎と応用, サイエンス社 (1997).

[5] ATI Technologies Inc. http://www.ati.com/.

[6] Gelder, A. V. and Kim, K.: Direct volume rendering with shading via three-dimensional textures, *Proceedings of the 1996 symposium on Volume visualization*, San Francisco, California, United States, pp. 23–ff (1996).

[7] Bhaniramka, P. and Demange, Y.: Multi-resolution representations: OpenGL volumizer: a toolkit for high quality volume rendering of large data sets, *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, Boston, Massachusetts, pp. 45–54 (2002).

[8] Zhou, Y. and Suri, S.: Analysis of a bounding box heuristic for object intersection, *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, Baltimore, Maryland, United States, pp. 830–839 (1999).

[9] Woolley, C. and Luebke, D. and Watson, B. and Dayal, A.: Session 7: rendering: Interruptible rendering, *Proceedings of the 2003 symposium on Interactive 3D graphics*, Monterey, California, pp. 143–151 (2003).

[10] Luebke, D. and Reddy, M. and Cohen, J. D. and Varshney, A. and Watson, B. and Huebner, R.: LEVEL of DETAIL FOR 3D GRAPHICS, Morgan Kaufmann (2003).

付録

A.1 精度制御の各手法による出力画像

(a)S_z:512,S_{xy}:512^2,R_v:1^3

(b)S_z:384,S_{xy}:512^2,R_v:1^3

(c)S_z:256,S_{xy}:512^2,R_v:1^3

(d)S_z:128,S_{xy}:512^2,R_v:1^3

図 A.1: S_z と出力画像 (skull)

(a)S_z:512,S_{xy}:512^2,R_v:1^3

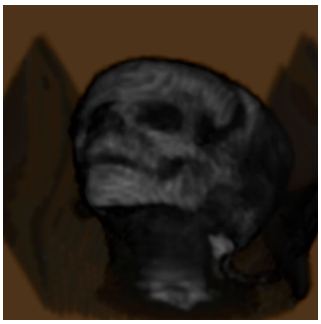
(b)S_z:512,S_{xy}:256^2,R_v:1^3

(c)S_z:512,S_{xy}:128^2,R_v:1^3

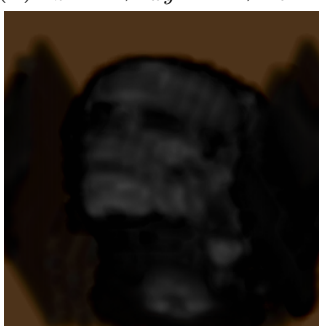
(d)S_z:512,S_{xy}:64^2,R_v:1^3

図 A.2: S_{xy} と出力画像 (skull)

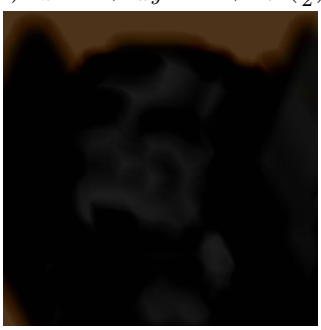
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{2})^3$

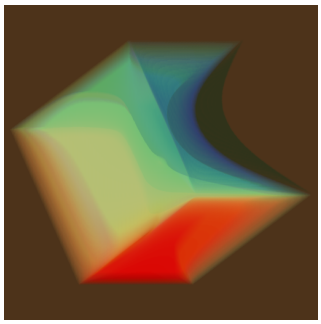


(c)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{4})^3$

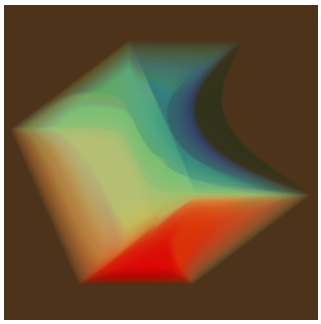


(d)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{6})^3$

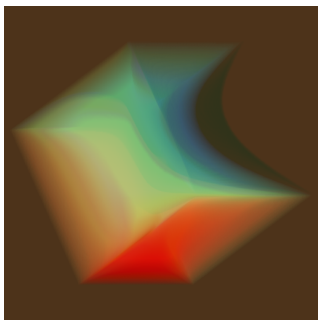
図 A.3: R_v:と出力画像 (skull)



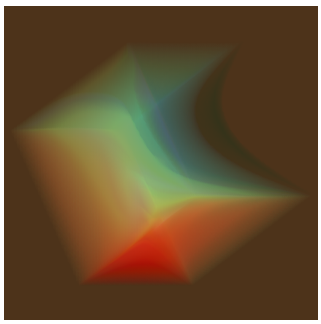
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:384,S_{xy}:512^2,R_v:1^3

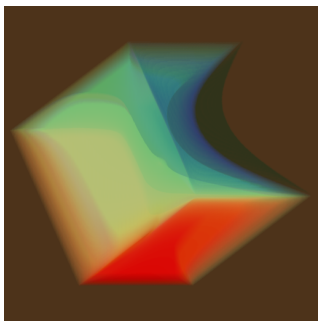


(c)S_z:256,S_{xy}:512^2,R_v:1^3

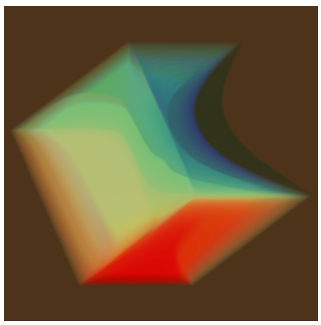


(d)S_z:128,S_{xy}:512^2,R_v:1^3

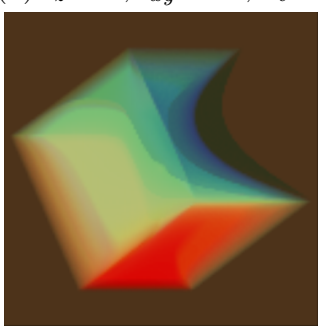
図 A.4: S_z と出力画像 (poisson)



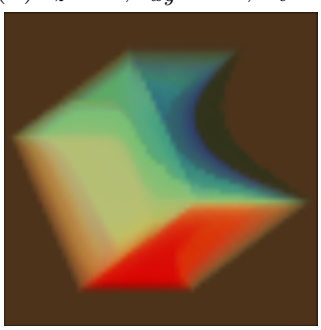
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:256^2,R_v:1^3

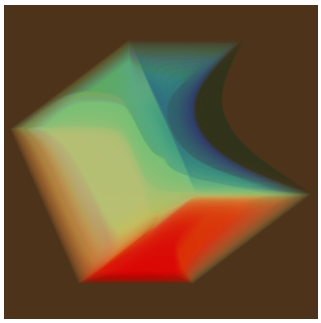


(c)S_z:512,S_{xy}:128^2,R_v:1^3

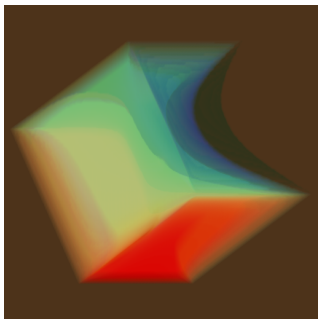


(d)S_z:512,S_{xy}:64^2,R_v:1^3

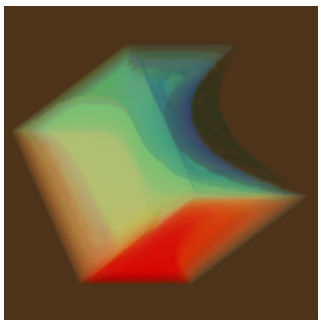
図 A.5: S_{xy}:と出力画像 (poisson)



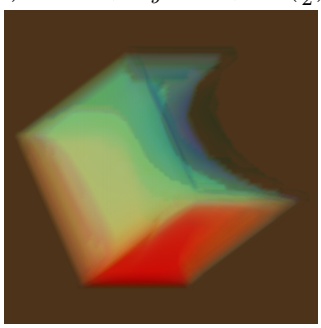
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{2})^3$

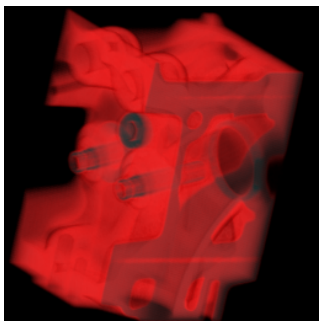


(c)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{4})^3$

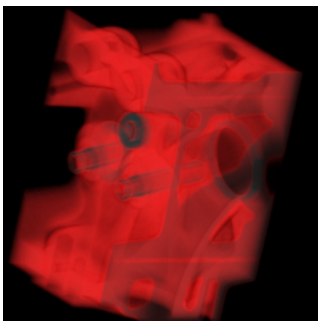


(d)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{8})^3$

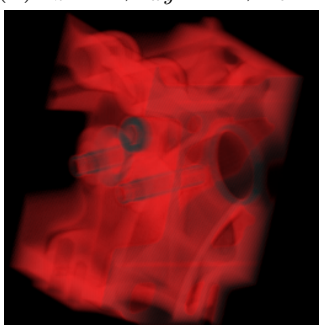
図 A.6: R_v:と出力画像 (poisson)



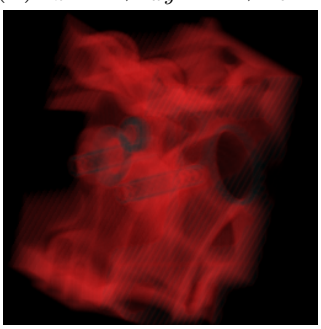
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:384,S_{xy}:512^2,R_v:1^3

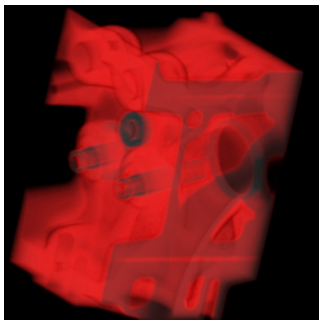


(c)S_z:256,S_{xy}:512^2,R_v:1^3

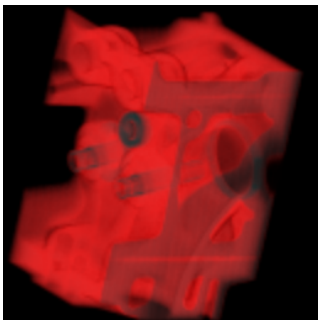


(d)S_z:128,S_{xy}:512^2,R_v:1^3

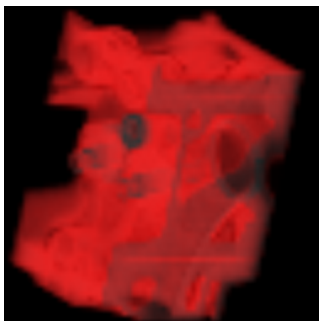
図 A.7: S_z と出力画像 (engine)



(a)S_z:512,S_{xy}:512^2,R_v:1^3



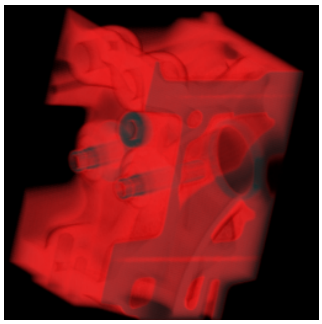
(b)S_z:512,S_{xy}:256^2,R_v:1^3



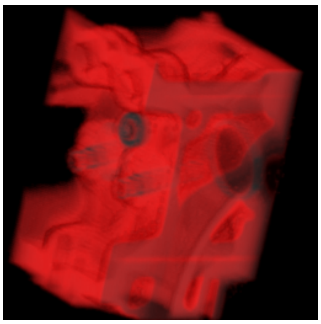
(c)S_z:512,S_{xy}:128^2,R_v:1^3

(d)S_z:512,S_{xy}:64^2,R_v:1^3

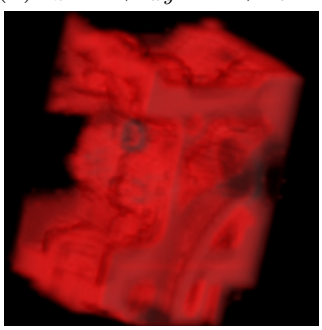
図 A.8: S_{xy}:と出力画像 (engine)



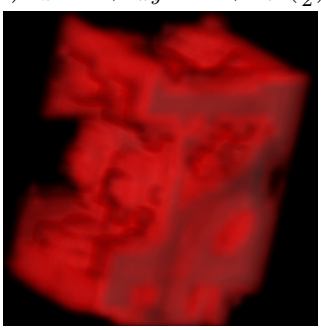
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{2})^3$

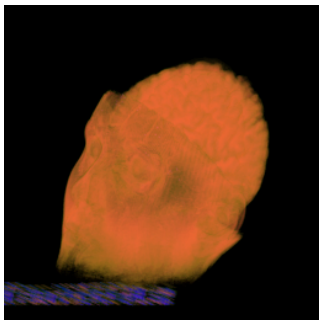


(c)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{4})^3$

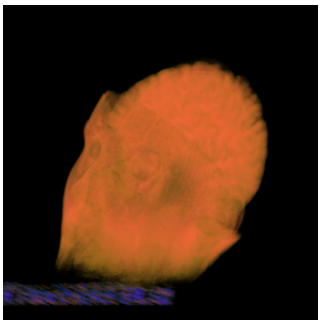


(d)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{8})^3$

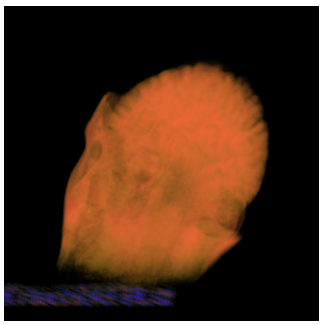
図 A.9: R_v:と出力画像 (engine)



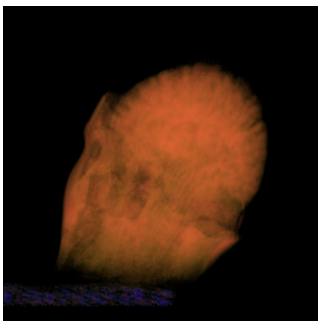
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:384,S_{xy}:512^2,R_v:1^3

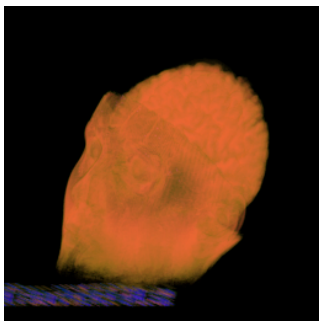


(c)S_z:256,S_{xy}:512^2,R_v:1^3

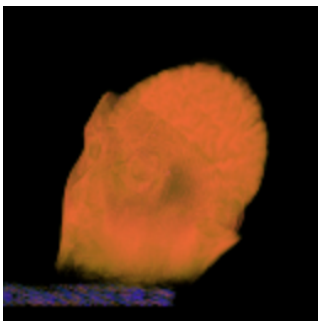


(d)S_z:128,S_{xy}:512^2,R_v:1^3

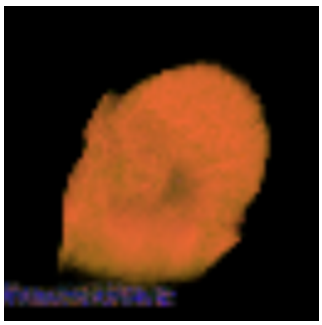
図 A.10: S_z と出力画像 (mrbrain)



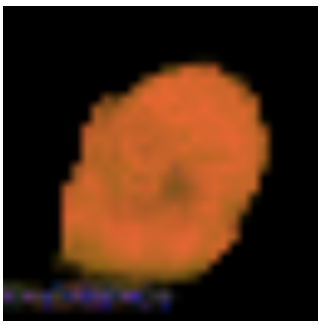
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:256^2,R_v:1^3

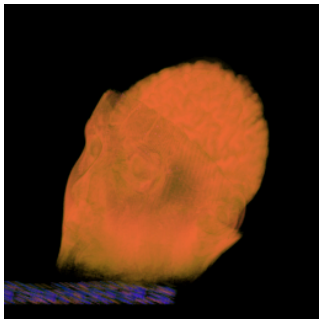


(c)S_z:512,S_{xy}:128^2,R_v:1^3

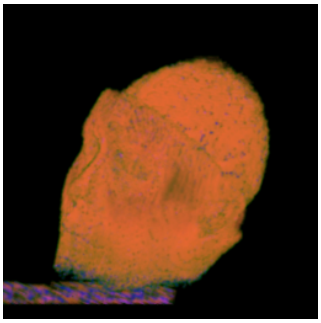


(d)S_z:512,S_{xy}:64^2,R_v:1^3

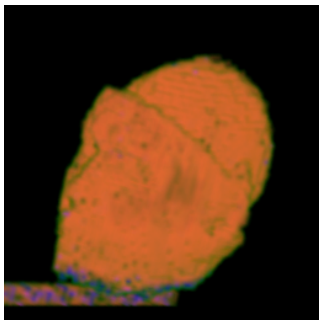
図 A.11: S_{xy}:と出力画像 (mrbrain)



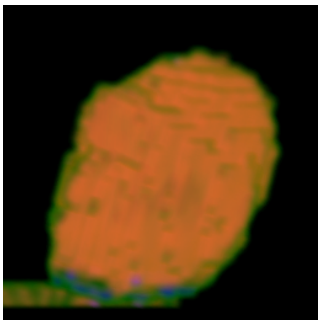
(a)S_z:512,S_{xy}:512^2,R_v:1^3



(b)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{2})^3$



(c)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{4})^3$



(d)S_z:512,S_{xy}:512^2,R_v:$(\frac{1}{8})^3$

図 A.12: R_v:と出力画像 (mrbrain)

A.2　アルファ値 0 のフラグメント数と描画時間の関係

アルファ値 0 のフラグメントの数と描画時間の関係を調べるため，128^3 のサイズのボリュームデータにおいて，全てのテクセルの RGB 値は 1（最大 255）とし，アルファ値 0 のテクセルと 1 のテクセルを以下のように分布させて描画時間を測定した．描画時間の測定方法は，5.1.1 のとおりである．精度は S_z:256,S_{xy}:512^2,R_v:1^3 とした．なお，テクスチャはボリュームデータの中心を原点として，x 軸方向にデータが並んでおり，次に y 軸方向，そして z 軸方向に並んでいるものとする．

1. アルファ値全て 0.
2. x>0 でアルファ値 0，x≤0 でアルファ値 1.
3. y>0 でアルファ値 0，y≤0 でアルファ値 1.
4. z>0 でアルファ値 0，z≤0 でアルファ値 1.
5. アルファ値全て 1.

表 A.1 に各データでの描画時間を示す．

表 A.1: アルファ値 0 のフラグメント数と描画時間の関係

テクスチャ	1	2	3	4	5
描画時間 (ミリ秒)	32.3	35.9	35.9	35.9	39.2

表 A.1 より，アルファ値 0 のフラグメントの数が同じだと描画時間も同じになると予想される．また，アルファ値が 1 のテクセルをアルファ値 255 に変えて描画時間を計測しても結果は同じであった．さらに，OpenGL の機能アルファ・テストにより全てのフラグメントの受け入れを拒否したときと，全てのテクセルのアルファ値が 0 であったときの描画時間が等しかった．これにより，アルファ値 0 のフラグメントは，アルファブレンディングの際に自動的に拒否されると考えられる．よって，アルファ値 0 のフラグメント数が描画時間に影響を及ぼすと言える．

A.3　(R_{s_x},R_{s_y}) による手法とボリューム解像度の関係

(R_{s_x},R_{s_y}) による手法の効果は，ボリュームを空間内にマッピングしたときの，視体積（スライスの配置されている空間）に対するボクセルの分布する密度によって変化する．本研究で用いたデータの中では，ボリュームのサイズが 256^3

と $256^2 \times 128$ のとき，ボクセルの密度が最大になる．このときボリューム解像度による手法によってボクセル密度を減少させて，それぞれに (R_{s_x},R_{s_y}) による手法を適用した場合の描画時間を示したものが表 A.2 と A.3 である．bonsai と engine を選んだのは，ボリュームのサイズは違うが，ボクセルが分布する密度は同じであるため，(R_{s_x},R_{s_y}) による手法の効果を比較してこれが等しいことを示すためである．これらの表のスクリーン解像度を変えたときの，描画時間の比を計算したものを表 A.4,A.5 に示す．これらの表の項目の値は，それぞれ S_{xy} を変化させたときの描画時間の比であり，例えば $512^2 \rightarrow 256^2$ の行ならば，S_{xy}:512^2 のときの描画時間に対する S_{xy}:256^2 のときの描画時間の比を表す．表 A.4,A.5 の各項目を比べると，その値は完全にばらばらではなくある程度似通っている．また，各表の項目はその左上の項目や右下の項目と値が近い．これは，一度のメモリアクセスでキャッシュに格納された領域によって，何個の再サンプリング点の値を計算できるかが関係している．ボリュームのボクセル密度が粗いと，一度のメモリアクセスで多くの再サンプリング点の計算ができる．しかし，再サンプリング点の間隔が広がってもメモリアクセスの回数はあまり減らない．ボクセル密度が細かいと，一度のメモリアクセスで計算できる再サンプリング点の数は少ないが，再サンプリング点の間隔が広くなると，計算に用いないボクセルが増加してきてメモリアクセスの回数が大幅に減少する．ただし，この場合は上述したように参照されるボクセルがなくなるようにボリュームデータを小さくしたほうが，計算も速くなるうえ見た目も綺麗になる可能性が大きい．

表 A.2: S_{xy},R_v による bonsai の描画時間（ミリ秒）

	R_v			
S_{xy}	1^3	$(\frac{1}{2})^3$	$(\frac{1}{4})^3$	$(\frac{1}{8})^3$
512^2	218	102	63.8	58.8
256^2	134	60.1	25.1	18.0
128^2	57.6	38.4	14.5	7.66
64^2	13.7	17.3	9.67	4.51

表 A.3: S_{xy},R_v による engine の描画時間（ミリ秒）

S_{xy}	R_v			
	1^3	$(\frac{1}{2})^3$	$(\frac{1}{4})^3$	$(\frac{1}{8})^3$
512^2	127	68.9	49.6	46.6
256^2	72.4	35.7	18.1	14.6
128^2	31.0	21.6	9.57	6.25
64^2	8.43	10.2	6.37	3.91

表 A.4: S_{xy} を変化させたときの描画時間の比（bonsai）

S_{xy}	R_v			
	1^3	$(\frac{1}{2})^3$	$(\frac{1}{4})^3$	$(\frac{1}{8})^3$
$512^2 \rightarrow 256^2$	0.61	0.59	0.39	0.31
$256^2 \rightarrow 128^2$	0.43	0.64	0.58	0.43
$128^2 \rightarrow 64^2$	0.24	0.45	0.67	0.59

表 A.5: S_{xy} を変化させたときの描画時間の比（engine）

S_{xy}	R_v			
	1^3	$(\frac{1}{2})^3$	$(\frac{1}{4})^3$	$(\frac{1}{8})^3$
$512^2 \rightarrow 256^2$	0.57	0.52	0.36	0.31
$256^2 \rightarrow 128^2$	0.43	0.61	0.53	0.43
$128^2 \rightarrow 64^2$	0.27	0.47	0.70	0.63