Vol.2010-MPS-77 No.31
2010/3/5

リアルタイムステレオQueryballの開発

勅使河原　佑美[†1]　　石　川　千　里[†1]
高　田　雅　美[†1]　　城　　和　貴[†1]

3次元空間内のオブジェクトを理解するための機能の1つとしてQueryballがある．Queryballはオブジェクトに半透明の球体を重ね合わせることにより，オブジェクトと球体が重なる部分に対して問合せることができ，オブジェクトの表示方法を変えたり，様々な方向からオブジェクトを視認したりできる．しかし，現段階のQueryballのボリュームレンダリングでの表示速度はオブジェクトデータのサイズが大きくなるほど遅くなり，実用には向かない．そこで，Queryballをリアルタイムで実行するために，ボリュームレンダリング部分にGPGPUを用いることによって，高速化を図る．また，オブジェクトをよりリアルに理解するために3次元立体視を行う．

Development of Real-time stereo Queryball

Yumi Teshigawara,[†1] Chisato Ishikawa,[†1]
Masami Takata,[†1] Kazuki Joe[†1] and

Queryball is a tool to understand objects in three-dimensional space. By overlapping a translucentglobe with 3D object, it can quiry to the part where the object and the globe come in succession. It can change the display method with visually checking according to various directions. However, the display speed the volume rendering of Queryball to the present stage slows the large size of the object data, and is not for practical use. Therefore, performance improvement by applying GPGPU to the volume rendering part in real time is necessary. In addition, the system offers three-dimensional stereoscopic vision for user to understand the object more realistically.

†1 奈良女子大学大学院人間文化研究科
Graduate School of Humanities and Sciences, Nara Women's University

1. はじめに

近年，科学技術データの可視化が重要視されている．特に3次元空間では，様々な視点からオブジェクトを見ることができ，数値データや平面データに比べ，直感的にデータを理解することが可能である．そのため3次元可視化技術は，流体，構造，医療などあらゆる分野で活用されている．

3次元空間内のオブジェクトを理解するための機能の1つとしてQueryball[1)2)]が提案されている．Queryballは，ユーザが半透明の球体を操作し，3次元空間に表示されているオブジェクトに重ね合わせることにより，オブジェクトと球体が重なる部分に対して問合せることができるシステムである．オブジェクトに対して，視点変更が可能であり，様々な角度から確認することができる．これにより，新たなデータの特徴や関連を見出すことができる．オブジェクトデータの可視化方法には様々な方法があるが，一般的によく使用される可視化方法の1つに，ボリュームレンダリングがある．ボリュームレンダリングを用いてデータを可視化した場合，データサイズが大きくなるほど，表示速度が遅くなる．本研究では，Queryballをリアルタイムに実行するために，ボリュームレンダリング部分の高速化を図る．高速化には，GPGPU[3)4)]を用いる．

本研究では，更にオブジェクトをリアルに体感するために，アナグリフ方式[5)]によって3次元立体視を行う．3次元立体視を行う上で，左右の目からの画像を交互に表示するため，リアルタイムにQueryballを実行するにあたり，1秒間に片目でそれぞれ30枚，合わせて60枚の描画が必要となる．

第2章では，既存のQueryballの定義と操作方法，問題点について説明する．第3章では，本システムの設計について述べる．ボリュームレンダリングへのGPGPUの適用について詳しく述べ，Queryballシステムの開発方法を説明する．また，3次元立体視の手法とQueryballの操作を容易にするためのGUIを開発について述べる．第4では，本システムの描画速度の評価を行う．第5章では，本稿のまとめを述べる．

2. Queryball

本章ではQueryballのシステム概要について述べる．Queryballとは，ユーザが半透明の球体を操作し，3次元空間内に表示されたオブジェクトに重ね合わせることにより，その重なった部分に対し直接問い合わせることができるシステムのことである．データの読み込み，可視化部分の計算にはvtk(Visual Toolkit)[6)7)8)]が使用されている．

Vol.2010-MPS-77 No.31
2010/3/5

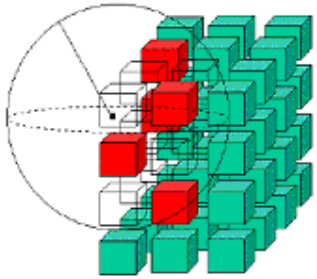

図 1 Queryball の仕組み

Queryball では，領域条件，探索条件，探索条件に該当する部分の表示方法，探索条件に該当しない部分の表示方法を定義することにより，データを可視化する．図 1 は縦に 4 個，横に 3 個，奥行きに 4 個 box が並んでいる 3 次元データに対して，Queryball を適用した例である．各 box には整数値 val があるとする．図 1 では，領域条件は球体内部，探索条件は val の値が 10 以上，探索条件に該当する部分は赤色，探索条件に該当しない部分は，透明で外枠のみ表示と定義されている．

図 2 は，例として地震波のデータの球体外部をレイ・キャスティング法[9] で，球体内部を地震波のベクトルを矢印で表示している Quaryball の GUI である．球体を作成したい場所で左クリックすると，画面上に球体が作成される．この時，マウスがダウンしていた時間の長さで球体のサイズを決められる．つまり長く押すと大きな球が作成される．また球体を作成した後，球体内部をマウスで選択しドラッグすることで，球体の移動が可能であり，球体外部をマウスで選択しドラッグすることにより，オブジェクトの回転が可能である．

Queryball でのボリュームレンダリングはレイ・キャスティング法が使用されている．この計算処理は，データが大きければ大きいほど長くなる．CPU が Genuine Intel(R)，1.6GHz，メモリが 1.99GB のマシンでは，100*100*100 のデータで Queryball を実行する際，1 秒間に 4.9 枚しか表示することができない．

3. システム設計

本章では，本システムをリアルタイムにステレオ表示させるためのシステム設計について述べる．2 章で述べたように，Queryball の問題点はボリュームレンダリングの計算に非常に時間がかかるということである．そこで，ボリュームレンダリングの計算部分に GPGPU

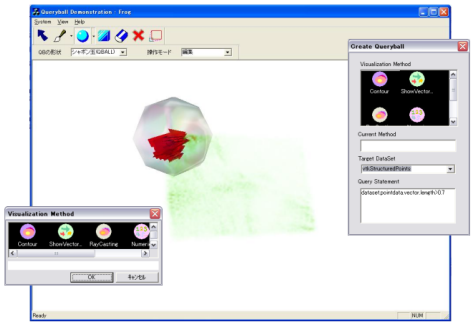

図 2 Queryball の GUI

を用いることにより高速化を図る．また，3 次元データを取り扱うにあたり，データを分かりやすく認識するために 3 次元立体視を行う．

3.1 節では本システムの設計事項について述べる．3.2 節では使用する GPU や，CPU と GPU それぞれでの使用言語，立体視に必要なものなど開発環境について説明する．3.3 節では GPU での計算と高速化について述べ，3.4 節では 3 次元立体視の手法について述べる．3.5 節では，GUI の作成と使用方法について説明する．

3.1 設定事項

本研究では，以下の 3 つの設計を行う．

- GPGPU を用いることによるボリュームレンダリングの高速化
- 3 次元立体視
- GUI

GPU で数値計算するにあたり考慮する点は，CPU と GPU でのデータ転送回数と転送するデータ量，データの保持する場所である．特にボリュームレンダリングする対象となるデータは非常に大きいものが多く，転送時間がかかる．ボリュームレンダリングの高速化は，スクリーンのピクセルごとの計算を GPU で行うことにより，並列計算を行い高速化を図る．この際，ボリュームレンダリングで使用するデータを常に GPU に保持しておくことによりデータ転送時間を削減することができる．

3 次元立体視は最も安価で手軽に行えるアナグリフ方式を用いる．この場合必要なものは赤青メガネのみとなる．3 次元立体視を行うにあたり考慮する点は，本システムを様々な環

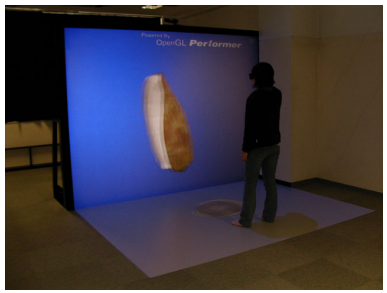

図 3 可視化工房での 3D 表示

境で使用するために，ユーザによる使用状況による設定を可能にすることである．ユーザにより異なる左右の目の幅やスクリーンサイズ，スクリーンと視点の距離の設定を様々状況に対し変更する必要がある．

また，既存の Queryball ではデータの問合せに使用する球体のサイズの変更，球体の奥行き方向の移動が出来ない．そこで，これらの機能を満たす GUI を新たに開発する．

3.2 開発環境

OS は Windows XP を用いる．Linux でも開発可能ではあるが，Windows XP の場合は本学の可視化工房への転用が可能である．可視化工房とは，本学理学部の各学科から可視化対象を提供してもらい，情報科学科の教育の一環として提供されたデータの可視化を行い，それを提供元に還元して教育研究に役立ててもらうためのコラボレーション・スペースである．可視化工房には，2 面の 2680mm*2010mm の大画面 3DCG 表示装置と 3D スキャナー，グラフィックスカードは NVIDIA Quadro FX4500 を内蔵した可視化サーバーが設置されている．電子式シャッター方式の 3 次元立体視が可能であり，このシステムは WindowsXP にのみに対応している．

GPU 用のプログラミング言語には CUDA[10)11)] を用い，それに伴い CPU でのプログラミングは C 言語を用いる．グラフィックス API は OpenGL[12)13)] を使用する．OpenGL は CUDA との相互運用が可能であり，非常に便利である．

3 次元立体視はアナグリフ方式で行うため，必要なものは赤青メガネのみになる．

3.3 高速化

本節では，本システムでのボリュームレンダリング計算と GPGPU の適用について述べる．3.3.1 項では，CPU から GPU へのデータの転送をどのように行うかについて，3.3.2 項では，ボリュームレンダリングの並列計算と球の内外判定について述べる．

3.3.1 データ転送

データの読み込みは，データファイル名とデータの x 方向，y 方向，z 方向のデータサイズを指定することにより行われる．データサイズにより，GPU にデータファイルを保持する領域を確保し，データを転送する．ここで GPU へ転送したデータはプログラム終了時まで GPU 側に保持し続ける．0〜1 の float 値の 3 次元座標で 3D テクスチャに問い合わせることにより，参照点のデータは自動的に 3 次線形補間された値を得ることができる．そのため，テクスチャデータは 1*1*1 の立方体になるように配置されていると考えボリュームレンダリングの計算が行われる．

そこでデータを GPU に転送する前に，x，y，z 軸方向のデータ数の中で最も大きな値を辺とする立方体のデータを用意する．データの中央に元のデータを配置し，データの存在しない部分には 0 を入力して新たなデータを作成する．元のデータを立方体のデータに作成し直した後，GPU に立方体のデータを転送する．この GPU に転送されたデータを使用して，ボリュームレンダリングを行う．これにより，最初のデータ転送時間はかかるが，それ以降の計算は単純になる．

ボリュームデータをプログラムが終了するまで GPU に保持するのに対し，Queryball の操作において必要な情報や，3 次元立体視における視点位置の情報は描画の度に GPU に転送する．この値はユーザが GUI を用いて指定することになる．

3.3.2 ボリュームレンダリング

レイ・キャスティングでサンプリング点を求める際，視線から一定の距離ごとに分割した点の値を取る．その点の値を用いてボリュームレンダリングの計算を行う．分割数により描画結果の精度が変わる．分割数が大きいほど精度が良くなるが，時間が増えるため，適度な分割数が必要となる．

球の内外判定は，各サンプリング点で行う．既存の Queryball では，球の内外判定を行った後，球内のみのデータと球外のみのデータの 2 つを新たに作成する．これは，VTK の関数を用いてボリュームレンダリングを行っているため，ボリュームレンダリングする前に，球の内外判定を既に行ったデータを準備する必要があるからである．一方，VTK の関数を使用せず，ボリュームレンダリングする前でなく，ボリュームレンダリングの計算中にサンプリング点が球の内部にあるかどうかを判断することにより，新たなデータの作成が必要なくなる．その結果によりサンプリング点の RGB 値と不透明度を変更する．これにより，メモリの節約と読み込み時間の短縮になる．

Vol.2010-MPS-77 No.31
2010/3/5

球の内外判定により，サンプリング点のデータ値の利用の有無が決まる．あるサンプリング点を (x_j, y_j, z_j) 球の半径を r，球の中心を (x_o, y_o, z_o) とする．更に，球体の膜の厚さを $2a$ として条件分岐を行う．

$$\sqrt{(x_j - x_o)^2 + (y_j - y_o)^2 + (z_j - z_o)^2} < r - a \quad (1)$$

$$r - a \leq \sqrt{(x_j - x_o)^2 + (y_j - y_o)^2 + (z_j - z_o)^2} < r + a \quad (2)$$

$$\sqrt{(x_j - x_o)^2 + (y_j - y_o)^2 + (z_j - z_o)^2} \geq r + a \quad (3)$$

式 (1) が球の内部，式 (2) が球の膜部分，式 (3) が球の外部を表している．例えば球の内部を無色透明に定義すると，式 (1) を満たすサンプリング点の RGB 値と不透明度を全て 0 に，式 (3) を満たすサンプリング点の RGB 値はデータから参照した値をそのまま使用する．球の膜部分の場合は RGB 値には球体の色を入力し，不透明度を低くすることにより，内部が見える半透明の球体ができる．

またサンプリング点の RGB 値を光源からの距離により変更する．光源に近い点は RGB 値を大きくし，遠い点は小さくする．

3.4 3 次元立体視

本節では，本システムへの 3 次元立体視の適用方法について述べる．本システムの立体視はアナグリフ方式により行う．3.4.1 項では左右の目の位置の設定について，3.4.2 項ではアナグリフ方式の適用方法について説明する．

3.4.1 視点の設定

3 次元立体視は左右の目からのオブジェクトの見え方の違いにより行う．左右の目の幅やスクリーンと視点の距離は，ユーザにより異なる．また，ボリュームレンダリングの計算に視点の位置を用いる際，スクリーンのサイズにより，計算上での座標は異なる．そのため，本システムを使用する際，ユーザに左右の目の幅，スクリーンと視点の距離，スクリーンのサイズを設定してもらう必要がある．この値により，ボリュームレンダリングの計算上での視点の座標が異なる．

図 4 は，視点の位置とスクリーン，オブジェクトの xz 平面上での位置関係である．ボリュームレンダリングの計算を行う座標上でのオブジェクトのサイズは 2*2*2，スクリーンのサイズは 3*3 となっている．ボリュームレンダリングの計算を行う座標上でのスクリーンから視点までの距離 $distance$ は，ユーザが本システムを使用する際のスクリーンサイズの 1 辺の長さを $screensize$(cm)，スクリーンから視点までの実際の距離を $distance_{real}$(cm) とすると，

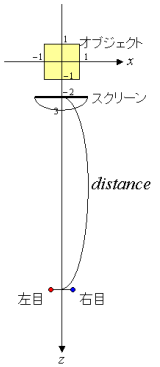

図 4 視点の位置設定

$$distance = distance_{real}/screensize * 3 \quad (4)$$

により，求めることができる．また，ボリュームレンダリングの計算を行う座標上での左右の目の幅 $parallax$ は，実際の距離の左右の目の幅を $parallax_{real}$(cm) とすると，

$$parallax = parallax_{real}/screensize * 3 \quad (5)$$

となる．本システムでは，視点の位置を変更する際，modelview matrix を使用する．OpenGL では，オブジェクト座標系における座標値を，カメラ座標系における座標値へと変換する際に，オブジェクト座標系における座標値に対して左から 4*4 の行列を乗算する．この行列を modelview matrix という．左右の目それぞれに対して，視点の回転の中心がずれる．そのため，OpenGL の物体を平行移動させる関数 glTranslate を用いることにより，回転の中心をずらし，modelview matrix の値を変化させる．

3.4.2 3 次元立体視の適用方法

本システムでは，アナグリフ方式により 3 次元立体視を行う．アナグリフ方式は，左右の目の見え方の画像を用意し，片方の画像から赤色要素のみを取り出した画像と，もう片方の画像から緑色と青色要素を取り出したものを，赤青メガネを用いて同時に見ることにより，立体視する方法である．図 5 の左図は左目から見た場合の画像から緑色と青色要素を取り出したもの，右図は右目から見た場合の画像からを赤色要素のみを取り出したものである．Queryball ではこの 2 枚の画像を高速に交互に表示する．そのため人間の目には図 6 のように 2 枚の画像を乗算合成したもののように見える．赤青メガネを通してこの画像を見る

Vol.2010-MPS-77 No.31
2010/3/5

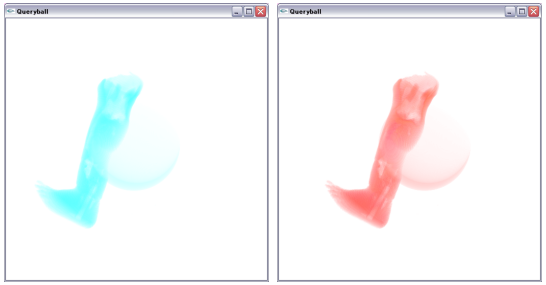

図 5 左右それぞれの目から見た画像

図 6 5 の 2 枚の画像の合成結果

ことにより，立体的に見ることができる

3.5 GUI の作成

本システムでは，GUI を作成するにあたり，GLUI を使用する．作成した項目は大きく分けて以下の 5 つである．

- 環境設定
- マウス操作
- 球体のサイズ
- 初期化ボタン
- 終了ボタン

ユーザの左右の目の幅，スクリーンのサイズ，スクリーンと視点の距離により，見え方が異なるため，これらの値を入力する上下ボタンを作成する．キーボードからの入力も可能である．入力された値により，式 (4) と式 (5) を用いて，ボリュームレンダリングの計算での視点の座標を計算する．

データ	データサイズ	1 秒間の描画枚数 (fps)
bucky.raw	32*32*32	59.81
hydrogenatom.raw	128*128*128	59.81
skull.raw	256*256*256	59.81

表 1 データサイズによる計算処理速度比較実験

マウス操作では，球体の移動，オブジェクトの回転，光源の移動のどれを操作するか選択を行う．更に球体の操作では，奥行き方向の移動を追加する．

球体のサイズは既存のものと異なり，球体のサイズは，上下ボタンでの数値変更や，キーボードからの入力に対応させる．既存の Queryball は一度作成した球体のサイズを変更出来ないが，この手法では変更可能である．初期値は 0.3 となっており，1.0 でオブジェクト全体を球がほぼ覆いつくすように設定する．

初期化ボタンは，球体の位置とオブジェクトの方向を初期の状態に戻すためのものである．オブジェクトの回転は，視点の移動により行っている．視点の移動は，x 軸と y 軸での回転で行われておりその回転角度を常に保持することにより，好きなときに視点位置を初期状態に戻すことが可能である．

終了ボタンでは，システムを終了されると同時に，CPU と GPU でそれぞれ確保されているメモリを解放する．

4. 評　　価

本システムの描画速度を計測する．はじめにデータサイズによる比較を行う．GPU は Geforce 8800 GTS 512 を使用する．データは CUDA のサンプルデータと New ”Real World” medical datasets available[14] に掲載されているデータを用いる．スクリーンのピクセル数は 512*512，3.3.2 項で述べた分割数は 250 とする．表 1 はデータサイズによる 1 秒間の描画枚数を比較した結果である．

この結果より，データサイズによる描画速度の違いはない．これは，ボリュームレンダリングでのサンプリング回数がほぼ変わらないからである．

そこで，分割数による比較を行う．本システムでのボリュームレンダリングは，最大で分割数回のサンプリングを行うことになる．オブジェクトの 3 次元空間内での大きさは 2*2*2 である．表 2 は分割数による 1 秒間の描画枚数を比較した結果である．

分割数	分割幅	1 秒間の描画枚数 (fps)
100	0.05	60.39
250	0.02	59.81
500	0.01	57.14
1000	0.005	30.05
2500	0.002	19.94
5000	0.001	10.02

表 2 分割数による計算処理速度比較実験

分割数が多いほど処理時間がかかることが分かる．オブジェクトがきれいに見えるためには分割数を最低 250 にしなくてはならない．この場合，1 秒間の描画枚数は 59.81 枚となる．この速度は，CPU で計算を行っていた場合の約 600 倍の速度である．また，1 秒間におよそ 60 枚描画しているため，リアルタイムな可視化が可能である．

データサイズが大きくなると，分割数を増やした，より精密な描画が求められる．そこで，Geforce 8800 GTS 512 よりもスペックの良い TESLA C1060 を用いて実験する．TESLA C1060 は描画処理が不可能であるため，GPU での計算速度を計測する．1 回の描画での GPU 部分の計算時間は，Geforce 8800 GTS 512 では 3.800ms であるのに対し，Tesla C1060 では 0.009ms となる．これは，Geforce 8800 GTS 512 の約 400 倍の速さである．よって，性能の良い GPU を使用すれば，分割数を増やしても 1 秒間に 60 枚以上の描画が可能である．

5. ま と め

本研究では，Queryball の問題点であるボリュームレンダリングの描画速度を解決するために，GPGPU を用いて高速化を行った．

ボリュームレンダリングでの描画結果を表示するスクリーンのピクセルごとの計算を GPU で並列処理した．Queryball の球の内外判定と光の影響を，GPU でボリュームレンダリングのサンプリング点での計算と同時に行うことにより，計算時間とメモリを節約できる．また，オブジェクトデータを GPU に転送する前に，あらかじめ立方体のデータに変更することにより，GPU 上での計算が容易となる．

より直感的にデータを理解するために，3 次元立体視をおこなった．3 次元立体視には，最も安価でどこでも簡単に実装できるアナグリフ方式を使用する．左右の目の画像を交互の表示したものを，赤青メガネを用いて見ることにより，3 次元立体視を実現している．

使用環境の設定，球体のサイズ変更，球体の奥行き方向の移動，光源の移動などの機能を追加するために，新たに GUI を作成した．GUI には描画速度にあまり影響がない GLUI を用いた．球体の移動方向として，新たに奥行きへの移動をマウスで操作可能になるよう追加した．また，球体のサイズ変更を上下ボタンやキーボード入力で変更でき，光源の移動もマウスで操作できる．同様に，立体視で用いる左右の目の幅や，スクリーンサイズ，スクリーンと視点の距離も上下ボタンやキーボード入力でユーザが設定可能である．これにより，様々な環境での 3 次元立体視が可能である．

オブジェクトデータのサイズにより，描画速度の比較を行った．美しく描画できていると認識できる間隔は 0.02 以下で，このときの描画速度は 59.81fps である．つまり，描画速度は CPU に比べ，約 600 倍の速度での描画が可能となった．1 秒間に約 60 枚描画しているため，リアルタイムな可視化が可能である．

本システムを本学の可視化工房で実装することが今後の課題である．

参 考 文 献

1) 渡辺知恵美, 増永良文, 城和貴：Queryball: 没入型 VR システムのための対話的な問合せモデル，日本データベース学会 Letters, Vol.2, No.2, pp.25–28 (2003).
2) Ai Ishida, Chiemi Watanabe and Kazuki Joe.: A Query Description Model and its Implementation as an Interactive Query Tool for Visualization Systems, The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications, Vol.I, pp.359–365 (2004)
3) GPGPU : http://gpgpu.org/
4) 編者 Matt Pharr, シリーズ編集 Randima Fer-nando, 監訳 中本浩：GPU Gems2 日本語版 ハイパフォーマンス グラフィックスと GPGPU のためのプログラミング テクニック, 株式会社ボーンデジタル (2005/11).
5) 佐藤崇徳, 後藤秀昭：アナグリフによる地形実体視と地理教育での利用, 日本国際地図学会平成 18 年度定期大会発表論文・資料 pp.16–17 (2006)
6) The Visualization ToolKit : http://public.kitware.com/VTK/
7) Will Schroeder, Ken Martin, Bill Lorensen.: Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition, Kitware (2006/12)
8) Inc. Kitware : VTK User's Guide Version 5, Kitware (2006/9)
9) S. Handa, T. Takada : Visual Computing: Integrating Computer Graphics with Computer Vision, Springer-Verlag (1992/06)
10) CUDA ZONE : http://www.nvidia.co.jp/object/cuda_home_jp.html
11) 青木尊之, 額田彰：はじめての CUDA プログラミング, 工学社 (11/2009)
12) OpenGL : http://www.opengl.org/
13) 林武文, 加藤清敬：OpenGL による 3 次元 CG プログラミング, 工学社 (2009/11)
14) New "Real World" medical datasets available: http://volvis.org/