2004－CG－117　(6)
2004／11／26

大規模三次元地図情報の高速表示ライブラリ

柿本 正憲[†‡]　芳賀剛士[†]　小原　理[†]　小出雅人[†]
立川智章[†]　小林敏彦[†]　橋本昌嗣[†]

[†]日本 SGI 株式会社
[‡]東京大学

近年、建物や道路などの構造物や地形・海底などの実世界の 3D データ化が進んでいる。そのようにして取得・整備された広域の大規模 3D データをリアルタイムで表示するための手法を考案し、アプリケーションから利用可能なライブラリを開発した。大規模データのファイルアクセスのオーバーヘッドを最小化するため、数 100GB にもなりうる単一ファイルのデータフォーマットを設計し、随時必要な部分を高速にページングする仕組みを実装した。表示処理では、ページングと組み合わせてカリング処理・LOD 処理・テクスチャ制御など最適化処理を行う。種々のアプリケーションに対応するためにはオブジェクト操作・更新が必要となるが、これについては、表示処理時にはページング可能な履歴ファイルとして保存し、後処理で単一ファイルにマージすることで実現した。

Fast Rendering Library for Large 3D Geographical Information

Masanori Kakimoto[†‡]　Takeshi Haga[†]　Osamu Ohara[†]　Masato Koide[†]
Tomoaki Tatsukawa[†]　Toshihiko Kobayashi[†]　Masatsugu Hashimoto[†]

[†]SGI Japan, Ltd.
[‡]The University of Tokyo

Today technologies of 3D data acquisition from real-world objects such as buildings, roads or terrain are in progress. We propose a method to render these types of large scale data sets with interactive rate and developed an application independent library. In order to minimize run time file access overhead for the large data set, we designed a data format for a single large intermediate file which may amount to several hundreds of gigabyte, as well as implemented a mechanism for paging objects from an instance of the file. In the rendering process, optimization techniques such as culling, LOD, texture control are used in combined with paging. To support various types of applications, object modification feature is realized by saving a history file which can also be paged in at run time and can be merged to the single large file at a post processing phase.

1. はじめに

近年、データ取得技術の進歩により、実世界にある物の 3D 形状や色のデータが容易に入手できるようになった。対象物として、人体や工業製品などのように、小規模な装置によりデータ取得できるもののほか、移動撮影や空撮による建築物や道路のデータ取得、衛星による地形データ取得など、広域にわたる実世界 3D データも得られるようになった。

CG による表示という観点で見ると、後者のような大規模な 3D データに関しては部分的な表示であればリアルタイム表示が可能である。しかし、広い領域をカバーしようとすると表示時間がかかるだけでなく、容量が大きすぎて表示すらできない場合もあるという問題があった。広域データの高速表示技術としては、ビジュアルシミュレ

ーション用のライブラリである OpenGL Performer[1]のデータベース・ページング[2]があり、おもに軍事用のフライトシミュレータに使われている。しかしながら、Performer のデータベース・ページングでは一回のページング単位がファイル一つとなっているため、ファイルの数が膨大に増え、ディスクアクセスの速い高価なプラットフォームでの動作に限られていた。

提案手法では、広域の 3D データのページングを効率化するために、全体をメモリ上とほぼ同じデータ構造を持つ単一ファイルとし、任意のサイズを単位として読み込む方式を考案し実装した。

2. 高速表示のための手法：Soarer

提案する広域データの高速表示手法は、地形や建物に関して、近景から遠景まで、視点が自由な高さに上昇してもデータ表示が可能なことから Soarer と名づけ、ライブラリ化した。

2.1. Soarer の処理の流れ

図 1 に Soarer の処理の流れを示す。

入力データとしては、建物・道路・街路樹のような都市構造物のほか、国土全体の標高データや世界全体の海底の深さデータなど、広域 3D 地図データを想定している。

これらの入力データを、それぞれのファイル形式に応じて前処理によって変換し、表示に必要な領域すべてのデータを含む soa フォーマットと呼ぶ形式の単一のファイルを生成する。

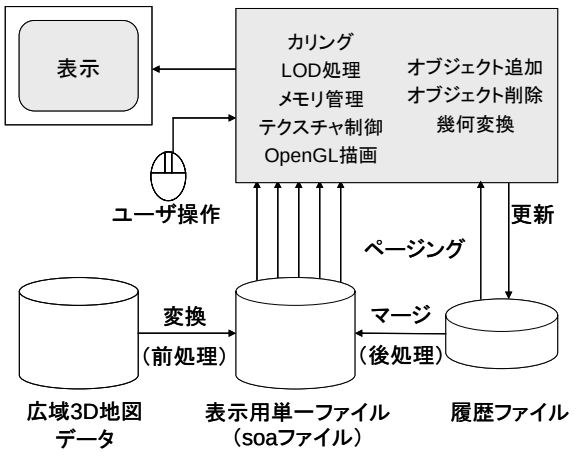

図　1. Soarer の処理の流れ

表示アプリケーション実行時に、Soarer は soa 形式のファイルを入力として必要な部分だけをページ・インして、次節以降で述べる最適化手法を用いて描画を行う。

soa 形式のファイルは、アプリケーション実行時にはつねに open した状態となる。

表示したオブジェクトを変更可能にするために、変更履歴を格納するファイルを用意する。この履歴ファイルは表示アプリケーション終了後に元の大きな soa ファイルにマージして反映させる。

2.2. データ構造

図 2 に、ページングのためのデータ構造を示す。Soarer の中核となるメモリ上のデータ構造は八分木（Octree）データである。地球全体を 24 レベルの八分木で表している。八分木の各ノードは立方体ブロックに対応する。トップレベル（レベル 0）のノードは地球全体を含む一辺 $2^{24}m \fallingdotseq 16800km$ の立方体に対応する。これを 8 分割した立方体が八分木のレベル 1 の 8 つの立方体となる。最も細かいレベルは一辺 $2m$ の立方体で、レベル 23 となる。

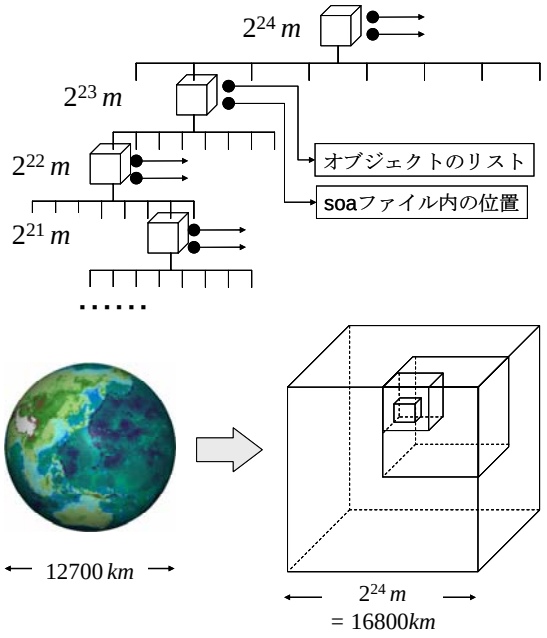

図2. 八分木ノードに対応する立方体ブロックによる表現

八分木の各ノードは、対応するブロックの座標系で定義された複数のオブジェクトを持っている。すべての表示用の物体はオブジェクトという単位で表現され、最終的には OpenGL 表示のための頂点データに分解される。

さらに、各ノードは、対応するオブジェクトデータが soa ファイル内のどの位置に格納されているかという情報を持っている。これにより、オブジェクトがメモリ上になくて読出しが必要な場合に少ないオーバーヘッドでディスクからロードすることができる。

2.3. 表示処理

表示処理は、八分木のノードをトップレベルからたどっていくことによって行う。図 3 に、ノードごとの処理の擬似コードを示す。

「ブロックが存在しない」というのは、soa ファイルの中に、当該ノードに対応するブロックの情報がないことを意味している。ノードは、それ以下の子ノードを持たない。

地球規模のデータではなく、限られた区域内（例えば東京 23 区など）の場合は、最初の数レベルは子ノードが一つずつしかないものをたどっていくことになる（図 2 上の八分木）。それらのブロックも、オブジェクトが存在しないものが続き、対象区域を含むブロックまで降りてきて初めて実際の描画（図 3 の「オブジェクト->draw()」）が実行される。

```
void DrawNode::draw_traversal() {
  if (視界に入っている) draw_block();
  for (８つの子供について) {
    if (ブロックが存在しない) continue;
    if (ブロックがメモリ上にある) {
      if (ブロックが不要) メモリ上から削除;
      else 子供->draw_traversal();
    } else { //メモリ上にない
      if (ブロックが必要) {
        ブロックの位置を読出しリストに登録;
          //あとでまとめて読出し
}}}}
void DrawNode::draw_block() {
  glPushMatrix();
  //ブロック中心位置にローカル座標系を移動
  glTranslate( ..... );
  for (すべてのオブジェクトについて) {
    オブジェクト->draw();
  }
  glPopMatrix();
}
```

図3. 八分木を使った表示処理（八分木のノード一つの表示処理）

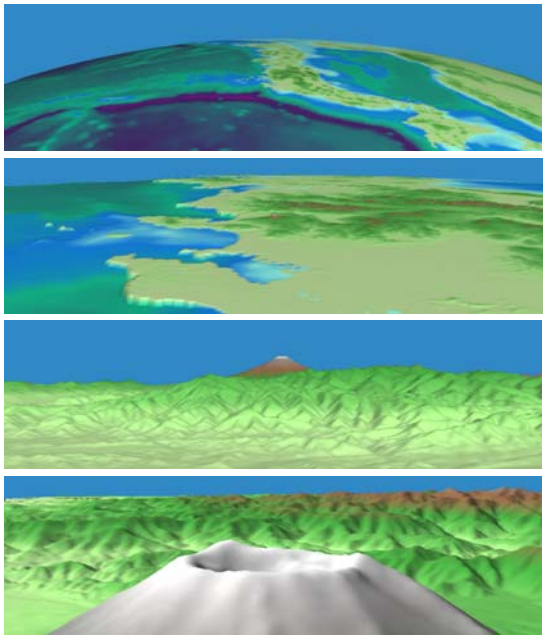

図4. 広域標高データの表示例（国土地理院発行の 50m メッシュの標高データと海洋の球面メッシュを使用）

ブロックが視界（ビュー・フラスタム）に入っているかどうかのチェック、いわゆるビュー・フラスタム・カリングは、不要な描画処理を行わないための一般的な手段で、ブロックの外接球（バウンディング球）を使えば 6 回以下の内積計算で済む。

ブロックが不要かどうか、すなわちメモリ上から削除するかどうかの判断は、視点からの距離とブロックの大きさによって決める。視点から近くて大きなブロックなのにメモリ上にない場合はそのブロックを新たにロードする。一方、小さいわりに視点から遠く、メモリ上にあるブロックはページ・アウト（メモリ上から削除）する。

このようなブロックの要・不要を判定するための閾値を変更することで、表示速度と画質のトレードオフを制御することができる。

図 4 に国土地理院発行の全国標高データ（50m メッシュ）と海洋のメッシュデータを用いた表示例を示す。リアルタイムでストレスなく、地球規

模から近景まで、あるいは全国を移動して表示することができる。

2.4. テクスチャ処理

三次元地図データでは使用するテクスチャデータも膨大な量になる。しかも、視野内の遠景にはそれぞれ異なるテクスチャを持つ多数の建物等が描画されるため、実際に描画に使用されるテクスチャ画像も膨大である。

一般に、テクスチャの拡大・縮小が起こる場合は画像のすべての詳細レベルを保持するミップマップ[3]を用いる。ミップマップは全部のテクスチャがハードウェアのテクスチャ・メモリに格納できることが前提になっている[4]。テクスチャの種類が多くテクスチャ・メモリに入りきらない場合は、別途テクスチャのページングが必要となるが、ミップマップの最も粗いレベルだけが必要な場合（遠景の建物ではひんぱんに起こる）でも詳細度レベルの高い画像も一緒にローディングされる。

テクスチャのページング技術としては、ミップマップの拡張である Clip-Map[5]がある。しかし、これは地形の標高データに衛星画像や航空写真を貼り付けるような、一枚の大きなテクスチャ画像を扱うもので、違うテクスチャを持つ個々の建物を多数扱う場合には向かない。

そこで、OpenGL で用意されたミップマップを使用せず、詳細レベルとページングを同時に処理する独自のテクスチャ管理を実装した。

本方式では、ミップマップの各レベル別にローディングできるようにした。これによって高精細なテクスチャ（例えば 512×512 画素など）を使った建物が遠景にある場合は、必要最小限の粗いレベル（例えば 4×4～16×16）のテクスチャだけを読み込む。このような遠景の建物は多数視野に入るため、効果は大きい。

以上の処理では、ページングするテクスチャの画素数を最小限とすることができる。しかし、レベルの粗い小さいテクスチャが多数存在する場合には、描画の際のテクスチャ切替オーバーヘッドが大きくなる。このため、ある一定レベルより粗い画像だけが必要な場合はテクスチャの使用を打ち切り、あらかじめ建物等のオブジェクトに設定した頂点色で代用した。

図5. 東京・渋谷の遠景（上）と近景（下）
（データ提供：株式会社ジオ技術研究所）

3. ライブラリ化と実装

実在する都市や地域の三次元地図データをインタラクティブに表示するソフトウェアの応用範囲は広い。例えば、景観シミュレーションをはじめとする都市・地域開発計画は代表的な応用例である。二次元地図表示との連動など、GIS のアプリケーションとの組み合わせも効果的な応用である。また、防災・交通シミュレーション、放送・映画制作への応用（マラソン中継、事故現場中継・ロケなどの撮影プランニング）、エンタテインメント応用なども考えられる。

このような多岐にわたるアプリケーションソフトを可能にすることを目的に、大規模三次元地図データの表示部分をアプリケーションソフト開発者が簡単に利用できるようライブラリ化し、ソフトウェア開発ツールキット（SDK）とした。

3.1. OpenGL アプリケーションとの役割分担

2 章で述べた表示技術を集約した Soarer ライブラリとアプリケーションの役割分担について述べる。

Soarer ライブラリは都市・地域の三次元地図データの高速表示を担当し、アプリケーション側はその上に必要な三次元物体を、やはり OpenGL を用いて描画する。アプリケーションから見ると、描画したい対象物の背景をクリアする代わりに Soarer に対して景観を描くように指示するような使い方となる。

Soarer では、アプリケーションとの座標系を合わせるためのユーティリティを用意しており、ちょうどアプリケーションが gluLookAt 関数で視点と視野を定義するのと同じ感覚で、実在の緯度経度上に視点と視野を設定できる。

また、Soarer では、表示中の三次元地図データのオブジェクトに対して、任意の直線との当たり判定や任意の点からの最近接オブジェクトの検出を行うことができる。

一方、GUI のようなユーザインタフェースは、すべてアプリケーション側が用意することになり、Soarer は GUI を持たない。

3.2. データ形式依存のライブラリ

実在の広域三次元地図データにはいくつかの種類がある。Soarer は、地図データの種類とは独立の表示技術であるが、実際にアプリケーションから三次元地図データを利用する場合は表示だけではなく、データに付随する各種情報を利用することが多い。その部分はデータの種類に依存せざるを得ない。

そこで、Soarer ライブラリは表示用のコアライブラリと位置づけ、アプリケーションから実際に見える API（Application Programming Interface）はデータ依存部分も含んだものとした。

ジオ技術研究所が提供している三次元デジタル地図 Walkeye Map[6]に対応するライブラリとして、Soarer を核とする GEO-Element をジオ技術研究所と共同開発した。アプリケーション開発者からは Soarer の API ではなく、GEO-Element の API が見えることになる。各ソフトウェア間の関係を図 6 に示す。

GEO-Element では、soa ファイルではなく ele ファイルという形式の単一ファイルに三次元地図データを格納する。これは soa ファイルの情報に加えて、Walkeye Map 特有の属性情報も含んでいるファイル形式である。

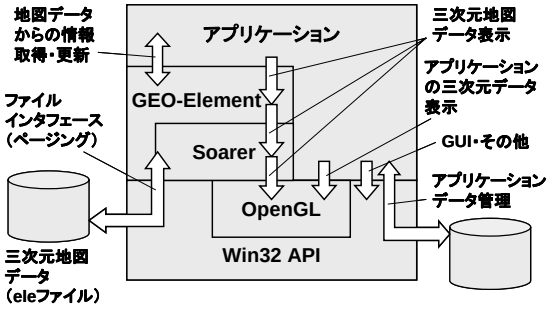

図　6. GEO-Element のソフトウェア構成

3.3. 柔軟性と高速性の両立

三次元地図のアプリケーションを考えた場合、単純に景観を表示するだけでは実用にはならない。建物等のオブジェクトの追加・変更・削除という基本的な編集機能を Soarer が持っている必要がある。これを実現し、アプリケーションが GEO-Element ライブラリを使って利用できるようにした。以下その方法について説明する。

GEO-Element で扱う ele 形式のファイルはサイズが大きい上に高速読み出し用に最適化されている。そのため、書き込みには時間がかかってしまい、アプリケーション実行中に書き込みを行うのは現実的ではない。

オブジェクトを変更する柔軟性と表示の高速性を大規模データに対して実現するために、1 回のアプリケーション実行中は、オブジェクトの変更履歴を記録する別のファイルを用いることにした（図 1 の右下）。

履歴ファイルは、ele ファイルと同様ページングに対応している。GEO-Element のページング処理の部分では元の ele ファイル以外に履歴ファイルもチェックする必要があるため、履歴ファイルが大きくなっていくと処理オーバーヘッドも大きくなるという問題がある。そのため、1 回のアプリケーションでの変更部分がたまってきたら、一度アプリケーションを終了し、履歴ファイルの内容を元の ele ファイルに反映させる必要がある。

履歴ファイルを ele ファイルに反映させる（マージする）エンドユーザ向けツールとして GeoMerge を用意した。GeoMerge を実行することにより、元の ele ファイルとは別に、履歴を反映した ele ファイルが生成される。GeoMerge の処理時間は、履歴の大きさにもよるが、10 秒～数分程度である。

3.4. 表示性能

Soarer あるいは GEO-Element の表示速度については、ページングの度合いによって変動する。ページングの影響がない場合には 15fps～60fps のフレームレートが得られるが、容量の大きいデータだと表示速度に影響が出て最悪で 2fps 程度まで落ちる。

Walkeye Map の東京・山手線内部及び周辺のデータを ele ファイルに変換すると 89GB になる。これを表示（図 7 上）すると、視野の奥行き方向にオブジェクトが少ない場合は 30fps 程度の表示速度が得られるが、奥行き方向にオブジェクトが多い方向を向いて視点・視線を変化させるとページングが起こり 2fps 程度となる。これは主にテクスチャデータのローディングまたは切り替えオーバーヘッドが原因と考えられる。

また、渋谷周辺の ele ファイルはサイズが約 2GB で、その表示（図 5 および図 7 下）では、15fps～60fps のフレームレートが得られた。

以上の表示性能評価で使用したハードウェアは、Xeon 3.06GHz、メインメモリ 2GB、グラフィックスは 3Dlabs 社製 Wildcat VP990 Pro（メモリ 512MB）、表示解像度 1280×1024 である。

4. まとめと今後の課題

広域の三次元地図データをリアルタイムで表示するための、ファイル形式・表示データ管理・テクスチャ処理などの高速化手法について述べ、それらを実用化するためのライブラリの機能について述べた。

今後の課題として、さらなる最適化が挙げられる。現在の実装では、テクスチャ処理が依然としてボトルネックになっており、特にページングが大量に発生する際に顕著となっている。これを解決することが第一の課題となっている。

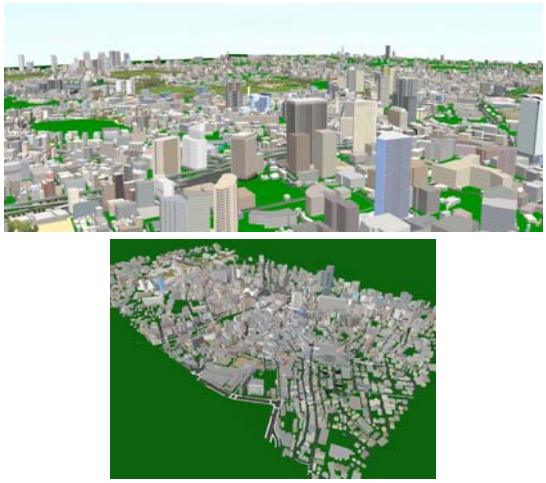

図7. 山手線内部および周辺データの一部（上）と渋谷周辺の全体データ（下）
（データ提供：株式会社ジオ技術研究所）

謝辞

GEO-Element の実装に際して多くのアドバイスをいただいたジオ技術研究所の田中敏夫氏に感謝いたします。

参考文献

[1] J. Rohlf and J. Helman: "IRIS Performer: A High-Performance Multiprocessing Toolkit for Real-Time 3D Graphics", *Proc. SIGGRAPH 1994*, pp. 155-162 (July 1994)

[2] Silicon Graphics, Inc.: “OpenGL Performer Programming Guide”

[3] L. Williams, “Pyramidal Parametrics,” *Proc. SIGGRAPH 1983*, pp. 1-11 (July 1983).

[4] OpenGL Architecture Review Board, “*OpenGL プログラミングガイド 第2版*,” アジソン・ウェスレイ，ISBN4-7952-9710-X (1997).

[5] J. Montrym, D. R. Baum, D. L. Dignam, C. J. Migdal, “Infinite Reality: A Real-Time Graphics System,” *Proc SIGGRAPH 1997*, pp. 293-302 (July 1997).

[6] 入江，藤，内海：”実画像と 3D 地図間のカメラ位置推定に関する研究,” 情報処理学会研究報告 2003-CG-117 (November 2004)(掲載予定).