「はやぶさ 2」運用補助のための可視化手法 —影の描写の検討—

三浦 昭[*1]，山本 幸生[*1]，吉川 真[*1]

Visualization Methods to Support operations of “Hayabusa2” - A Study on rendering shadows -

Akira MIURA[*1], Yukio YAMAMOTO[*1], Makoto YOSHIKAWA[*1]

Abstract

In this paper, we study on algorithms to render shadows precisely and quickly in scenes of the visualization software that we intend to use in operation phase of the space probe ”Hayabusa2” and the asteroid 1999JU$_3$. By means of ray tracing algorithm, we can render shadows more precisely than those of Z-buffering. We have found that, using simplified algorithms of ray tracing and hardware accelerations of GPGPU (OpenCL), it is possible to use ray tracing to render scenes at operations where real-time responses are required. Using a hybrid method of Z-buffering accelerated by OpenGL and ray tracing accelerated by OpenCL, it is possible to render quickly enough in such situations as changing viewpoint using GUI, and so on.

Keywords: Hayabusa2, visualization, ray tracing, OpenGL, OpenCL, GPGPU, Speeding up

概 要

本稿においては，「はやぶさ 2」と小惑星 1999JU$_3$ を想定した，探査機運用者向け可視化ソフトウェアにおける，正確な影（shadow）の描写と，レンダリング速度の向上について検討した．深度バッファを用いた手法に比べてレイトレーシングは影の描写が正確である．本稿においては，レイトレーシングのアルゴリズム簡略化と GPGPU（OpenCL）による高速化を図った結果，リアルタイム性が要求される運用においても，レイトレーシングによるレンダリングが実用的な速度で実現可能であることが示唆された．また OpenCL によるレイトレーシングと OpenGL による深度バッファを用いた計算と組み合わせることにより，リアルタイム性の高い視点変更等の操作にも耐えうるレンダリング速度が得られることが確認できた．

キーワード : はやぶさ 2，可視化，レイトレーシング，OpenGL, OpenCL，GPGPU，高速化

1. はじめに

本稿においては，「はやぶさ 2」と小惑星 1999JU$_3$ を想定した，探査機運用者向け可視化手法における，高精度な影（shadow）の描写と，レンダリング速度の向上について検討する．

筆者らはこれまで，「はやぶさ」の軌道の可視化について，その軌道の推定から各種可視化手法まで検討して来た [1][2]．リアリティを追及した可視化手法としては，レイトレーシング [3] に基づいた CG ソフトウェアが各種提供されており，

筆者らも POV-Ray[*2] を用いた可視化を試みてきた．しかしながら POV-Ray はレンダリング速度・レンダリング機能の両面で，運用に係る要求仕様を満たすことが難しくなった．要求仕様を満たすような商用・非商用 CG ソフトウェ

[*] 平成 27 年 12 月 17 日受付 (Received December 17, 2015)
[*1] 宇宙航空研究開発機構宇宙科学研究所
(Institute of Space and Astronautical Science, Japan Aerospace Exploration Agency)
[*2] Persistence of Vision Raytracer Pty. Ltd. , “POV-Ray - The Persistence of Vision Raytracer”, http://www.povray.org/ (2015 年 11 月現在)

アについても調査を進めたものの，要求仕様に含まれると想定される Hapke モデル [4] 等の機能を実装した製品が本稿執筆時点では見受けられない等，調査は難航している．

そこで筆者らは，要求される機能・性能を満たすような，応答性能の高い可視化ソフトウェア開発に着手した．その開発にあたって課題の一つとなったのは，影（shadow）を高精度・高速にレンダリングすることである．探査機の運用やその模擬・訓練にあたっては，影の描写が重要となるフェーズがあり，本稿においては，要求仕様を満たすような分解能を維持しつつ，実用レベルの応答になることを一つの目安として，影に着目したレンダリング手法について検討を行う．

レンダリングの精度という観点では，画素毎に輝度計算するレイトレーシングが有利であり，今後のアプリケーションにも適用したいところであるが，速度面での優位性は低い．一方で OpenGL[*3] と GPU の組み合わせによるプリミティブ毎の計算は，速度面では有利であるが，精度面での課題がある．

本稿においては，これらの手法を要素毎に分解し，精度の観点と速度の観点の両面から，運用に供する妥当性のある手法を検討し，評価することとした．一つには，精度の面で有利であるレイトレーシングを用いたリアルタイムレンダリングが実現可能であるかについて吟味する．その一方で，速度面でレイトレーシングを補えるよう，速度重視の手法で妥当な精度を実現できるかについても吟味する．

2. 要求仕様

現時点で探査機の運用者向け可視化ソフトウェアに対して要求されると想定される仕様を表 1 に示す．これらの仕様は「はやぶさ 2」運用関係者へのヒアリングやプレゼンテーション等に基づいたものである．可視化の用途によってはこれらの一部が緩和される場合もある．

またこれらはいずれも正式な仕様を表すものではない．表中で「(仮)」と記した箇所は，要求仕様を待つまでの間の，仮の設定値である．

これらの内，陰（shade），テクスチャ，歪については将来制定される要求仕様を待つ段階である．それぞれに対応できる拡張性を意識する必要はあるが, 本稿においては, 形状と影（shadow）を要求仕様に則って描くことに注視する．

2016 年以降，模擬・訓練の時点では，模擬用の小惑星形状や影を 10cm の分解能で表示する必要がある．その際に必要とされる画像解像度は 512×512 ピクセルから 1024×1024 ピクセルである．

2018 年の運用フェーズでは，4K ディスプレイ上での GUI 操作が予定されている．表示する画像としては 1,024×1,024 ピクセルを超える可能性もあるが，本稿の時点では 1,024×1,024 ピクセルを想定して評価する．

表 1 要求仕様

主用途	模擬，訓練	運用
開発時期	2016 年〜	〜 2018 年
UI	CLI/GUI（仮）	GUI
画像解像度	512×512 1024×1024	512×512 1024×1024 その他（仮）
分解能	10cm（精査時）	
レンダリング方法	リアルタイムレンダリング プリレンダリング	リアルタイムレンダリング
表示速度	TBD	精査 : 3 fps（仮） 高速 : 10fps（仮）
表示項目	小惑星 探査機及び探査機の付随物 各種補助表示 (物理量, 方向, 座標等)	
影 (shadow)	上記分解能で影を表示すること．	
陰 (shade)	Hapke モデル等，今後制定される要求仕様に基づいて上記分解能で陰を自動生成するプログラムが可能であること．	
小惑星のテクスチャ	今後制定される要求仕様に基づいて，上記分解能で小惑星のテクスチャを自動生成するプログラムが可能であること．	
歪み	今後制定される要求仕様に基づいて，カメラの光学系の歪み等を再現するプログラムが可能であること．	

[*3] Khronos Group, “OpenGL - The Industry Standard for High Performance Graphics”, https://www.opengl.org (2015 年 11 月現在)

本稿執筆時点の評価ソフトウェアを用いたプレゼンテーション等からの経験則により，10cmの分解能が要求されるような精査時においては，毎秒3フレーム程度以上の表示能力を想定する．

また視点変更等の操作時にはマウス等による指示が迅速に反映されるよう，さらに高速の表示モードが必要と考えられる．本稿においては経験則から，高速表示モードとして10fpsを想定する．高速表示モードにおいては形状や影の精査はしないことを前提に，簡易表示を行うことも現実的な選択肢としてありうる．

「はやぶさ2」は小惑星到着時に，高度20km付近から小惑星を観測する予定となっており[5]，その後に小惑星への降下が予定されている．そのため，可視化ソフトウェアとしては，遠方から至近距離まで破たんのないレンダリングが必要となる．また運用フェーズにおいては運用者が画像の視点・注視点等を自由に設定することが想定される．要求仕様に基づいてレンダリング手法を検討する際は，このような視点・注視点の自由度の高さも考慮する必要がある．

3. 評価に用いる環境・データ

3.1. ハードウェア・ソフトウェア環境

本稿で評価に用いるハードウェア及びソフトウェアについて，以下に示す．以後本稿においては，「Mac Pro」，「MacBook Pro」，「Mac mini」の呼称は以下の具体的な環境に対する固有名詞として使用する．

用いるフレームワークはいずれもXcode[*4]に標準で添付されているものである．その他のAPI，ライブラリ等は自製のものを使用する．

本稿の手法を具体的に実装するプラットフォームは未定であるため，幅広いプラットフォームに適用可能なプログラム手段を採用する．検討対象としている手法自体のコーディングは，CPU側のプログラムはC++を用い，GPUによる高速化はOpenGL[*3]とOpenCL[*5]を用いる．プラットフォーム依存性があるObjective C++とCocoaはベンチマーク用の初期設定を行うための利用に留める．

(a)Mac Pro環境
- 筐体 : Mac Pro (Late 2013)
- CPU: Xeon E5 プロセッサ 3.5GHz 6コア
- GPU: AMD FirePro D700[*6] 2式
  - 6GB GDDR5 VRAM
  - 2048 stream processors
  - 3.5TFLOPS
- 主記憶 : 64GB
- OS: OS X Yosemite
- 開発環境 : Xcode 7.1(7b91B)
- フレームワーク
  - Cocoa
  - OpenGL (version 2.1)
  - OpenCL (version 1.1)
- コンパイラ :
  - clang (C++，Objective C++):
    - LLVM version 7.0.0
  - openclc (OpenCL): LLVM version 3.2

(b)MacBook Pro環境
- 筐体 : MacBook Pro (Retina, 15-inch, Mid 2014)
- CPU: Core i7 プロセッサ 2.8GHz 4コア
- GPU1: Intel Iris Pro Graphics 5200[*7] (CPU内蔵)
  - 1.5GB SDRAM
  - 40 execution units, 280 threads
  - 832GFLOPS @1.3GHz
- GPU2: NVIDIA, GeForce GT 750M[*8]
  - 2GB VRAM
  - 384 unified shaders
  - 722.7GFLOPS
- 主記憶 : 16GB
- OS: OS X Yosemite
- 開発環境 : Xcode 6.3(6D570)
- フレームワーク
  - Cocoa

[*4] Apple Inc., “Xcode - What’s New - Apple Developer”, https://developer.apple.com/xcode/index.html (2015年11月現在)
[*5] Khronos Group, “OpenCL - The open standard for parallel programming of heterogeneous systems”, https://www.khronos.org/opencl/ (2015年11月現在)
[*6] Apple Inc., “Mac Pro - Technical Specifications - Apple”, http://www.apple.com/mac-pro/specs/ (2015年11月現在)
[*7] Intel Corporation, “The Compute Architecture of Intel® Processor Graphics Gen7.5”, https://software.intel.com/sites/default/files/managed/f3/13/Compute_Architecture_of_Intel_Processor_Graphics_Gen7dot5_Aug2014.pdf, pp.12-13 (2015年11月現在)
[*8] From Wikipedia, the free encyclopedia, “GeForce 700 series”, https://en.wikipedia.org/wiki/GeForce_700_series (2015年11月現在)

OpenGL (version 2.1)
OpenCL (version 1.1)
コンパイラ :
clang (C++，Objective C++):
LLVM version 6.1.0
openclc (OpenCL): LLVM version 3.2

(c)Mac mini 環境
筐体 : Mac mini Late 2012
CPU: Core i7 プロセッサ 2.6GHz 4 コア
GPU: Intel HD Graphics 4000[*9] (CPU 内蔵)
1GB SDRAM
16 execution units, 112 threads
256GFLOPS @1GHz
主記憶 : 16GB
OS: OS X Mavericks
開発環境 : Xcode 6.2(6C131e)
フレームワーク
Cocoa
OpenGL (version 2.1)
OpenCL (version 1.1)
コンパイラ :
clang (C++，Objective C++):
LLVM version 6.0
openclc (OpenCL): LLVM version 3.2

3.2. 表示すべきオブジェクト

本稿において表示すべきオブジェクトを表 2 に示す．以後，レンダリングにあたっては，ひとまとまりの形状をオブジェクトと称し，そのオブジェクトを構成する個別の原始的な形状をプリミティブと称する．

各種物理量や補助線等は，運用者を補助するために，各々の特徴に応じたモデリングが想定される．本稿においては，その例として LRF データを扱う．LRF は探査機から幾つかの方向にビームを放って測距する仕様となっており，得られた距離とビーム方向から，小惑星表面の位置を推定することができる．本稿で用意した LRF オブジェクトは四角錐で，錐の先端が測距点を示す．

本稿においては，探査機は 1 つのオブジェクトで表現する．小惑星のモデルは同一形状でありながら解像度の異なる 4 種類を用いるが，評価に当たっては，シーン毎に 1 つのオブジェクトを選択する．LRF データは 4 方向分を可視化するものとして，4 オブジェクト用意する．これらを合計して，1 つのシーンで同時に用いるオブジェクト数は 6 となる．

表 2 速度比較に用いるオブジェクト

表示項目	オブジェクトの概要	プリミティブ数	
		略記	総数
探査機	「はやぶさ 2」		213,405
小惑星	Gaskell イトカワ形状モデル (右記いずれかを選択)	49K	49,152
		196K	196,608
		786K	786,432
		3145K	3,145,728
LRF	四角錐 (4 式)		4×4 式

これらの形状データは，いずれも実際の「はやぶさ 2」や小惑星等を正確に再現したものではない．本稿においては，与えられた形状データに基づいて適切なレンダリングができることを確認するために，これらのデータを用いている．その範囲内においては，形状データ自体と現実の探査機等との形状の相違は本質的な問題ではない．

2016 年の模擬・訓練以降は，現実の探査機の形状を反映したレンダリングを行う必要があり，フライトモデルの形状をフィードバックすることは，今後の課題として残されている．同様に小惑星 $1999JU_3$ の詳細な形状モデルも得られていないため，小惑星イトカワの Gaskell 形状モデル [*10] を借用した．2016 年以降の模擬・訓練にあたっては詳細な仮想小惑星モデルを用いることが想定される．また実運用にあたっては，随時小惑星の推定形状がアップデートされることとなる．

「はやぶさ 2」の小惑星接近時の実データはまだ取得されておらず，LRF をはじめとした，本稿で用いる位置・姿勢等の諸データは，模擬データである．

[*9] David Cowperthwaite, David Hoff, “Intel® Processor Graphics: Now Serving the Cloud”, https://software.intel.com/sites/default/files/managed/a6/e3/Now%20Serving%20the%20cloud˙Cowperthwaite%20.pdf, p.6 (2015 年 11 月現在)
[*10] HAYABUSA PROJECT SCIENCE DATA ARCHIVE, “Shape Model”, https://darts.isas.jaxa.jp/planet/project/hayabusa/shape.pl (2015 年 11 月現在)

4. レンダリング手法

要求仕様の中で，プリレンダリングについては，時間的な制限が緩やかであるため，レイトレーシングを用いるものとする．一方でリアルタイムレンダリングについては，精度と時間的な制約とのトレードオフを検討する必要がある．本章においては，OpenGL 等で多用されているプリミティブ毎のレンダリング手法と，レイトレーシングに代表される画素毎のレンダリング手法とを，要素に分解して整理する．

探査機や小惑星の陰 (shade) は，本稿における評価ではランバート反射によるフラットシェーディングを用い，テクスチャについては単色とする．光学系の歪みは考慮しない．これらは，係る要求仕様が定まった時点で検討・評価するものとする．

異なる手段においても，ソースコードや座標変換等の手順を共有できるよう，パイプラインの各段における，座標変換，視点変換，クリッピング等に係る行列の定義は OpenGL で用いられる行列に揃え，各種座標変換に係る配列要素の計算方法も OpenGL API 互換とする．

4.1. レンダリング手法の簡素化

本章の各手法は，各節に掲げる参考文献等に準じているが，以下のような簡素化を前提として高速化を図る．また今後のソフトウェア開発や不具合修正等の維持管理に係るコストを抑えるため，コーディングが徒に複雑とならないことも考慮する．

(a) 扱うプリミティブは三角形のみとする

各オブジェクトは，三角形の集合体として考える．小惑星の形状モデルはポリゴンモデルとして提供されており，曲面等を考慮する必要はない．探査機の形状等も，ポリゴンモデルに変換して使用する．

(b) 鏡面反射や透過は考慮しない

オブジェクトのレンダリングにあたって，鏡面反射や透過等は，運用者向けの可視化としては本質ではないため，割愛する．

(c) 小惑星や探査機を描く際の光源は太陽のみとする

陰 (shade) や影 (shadow) の計算にあたって，光源は 1 つのみとする．探査機のライトを用いた撮影等，複数光源の使用は特殊事例であり，現時点では考慮しない．

(d) 座標変換等はオブジェクト単位とする．

各プリミティブ毎の異なる座標変換は行わず，座標変換等が必要な場合はオブジェクトを単位とする．今後構築されるアプリケーションにて，探査機のアンテナ等，可動部分を取り入れる際は，稼働部分を独立したオブジェクトとして扱うことを想定する．

4.2. 形状を計算する手法

本稿においては，形状を計算する手法として，プリミティブ毎に計算する手法と画素毎に計算する手法を検証する．

4.2.1. プリミティブ毎に計算する手法

レンダリング対象のプリミティブ毎に，画面上の位置と色を計算する手法である．

本稿においては，計算するシーンの奥行き情報を参照する手法（Z-Buffering[*11]）を採用する．以後，奥行き情報を参照するバッファを深度バッファと記す．この深度バッファに記録された奥行き情報と新たに計算するプリミティブの奥行き情報を比較し，それまで計算した中で最短の奥行きとなる部分については，深度バッファに新たな奥行き情報を記録するとともに，その部分に表示されるべき色を，当該プリミティブと光源との位置関係等に基づいて計算する．

本稿においては，プラットフォーム間の互換性が高い高速化手段として，OpenGL と GPU の組み合わせを採用する．OpenGL には，glVertexPointer() や glDrawArrays() 等 [*12] の，予め定義された頂点配列等を一括して扱うための API が用意されており，本稿においても，これらの API を用いて，オブジェクト単位でプリミティブの頂点配列を作成の上，レンダリングを行う．そのため，上記深度バッファに係る計算式は，ソースコード上には表れない．

陰やテクスチャ，歪みに係る将来の要求仕様に対しては，プログラマブルシェーダでの対応が想定されるが，現時点では固定機能シェーダ

[*11] From Wikipedia, the free encyclopedia, “Z-buffering”, https://en.wikipedia.org/wiki/Z-buffering (2015 年 11 月)
[*12] Khronos Group, “OpenGL 2.1 Reference Pages”, https://www.opengl.org/sdk/docs/man2/ (2015 年 11 月現在)

を用いる．

この手順だけでは他のプリミティブと光源との位置関係を知る方法が無いため，影を計算することはできない．

4.2.2. 画素毎に計算する手法

レンダリング対象の画素毎に，その画素に表示されるべき色を計算する手法であり，本稿で述べる手法はレイトレーシング [(3)] の手法の一部である．

この手法においては，各画素の対応した視線（レイ）を追跡し，レイとプリミティブとの衝突位置を計算する．レイと衝突する全てのプリミティブに対して，その衝突位置までの距離を計算し，最も距離の近いものについて，その部分を構成するプリミティブと光源との位置関係等に基づいて陰を計算する．本稿においては，透過を考慮していないため，レイの追跡は 1 回のみである．

本稿においては，GPU を用いた高速化手法として，OpenCL を採用する．OpenCL は異なるアーキテクチャの GPU に対して同じソースを適用できるメリットがあると期待される [*13]．

陰やテクスチャ，歪みに係る将来の要求仕様に対しては，関連するソースコードの追加で対応可能である．

プリミティブ毎に計算する手法と同様，この手順だけでは他のプリミティブと光源との位置関係を知る方法が無いため，影を表現することはできない．

4.3. 影を計算する手法

影を計算するには，光源と他のプリミティブとの位置関係を計算する必要がある．本稿においては前節 4.2.1，4.2.2 の手法を光源方向に適用して計算する．また本稿においては，単一光源を前提としているため，いずれの手法における計算も単一光源に対して 1 式のみとなる．

4.3.1. プリミティブ毎に計算する手法

シャドウマッピング [(6)] の手法である．本稿においては，4.2.1 との組み合わせをシャドウマッピングとして評価する．

4.2.1 においては視点と各プリミティブとの位置関係から深度バッファを計算したため，その深度バッファは視点との距離情報を有する深度バッファとなる．本節においては，この方式を，光源と各プリミティブの位置関係に置き換えて計算する．すなわち光源を影計算の視点位置として，最も光源位置に近いプリミティブの深度を計算する．この深度バッファは光源との距離情報を有する深度バッファであり，シャドウマップと呼ばれる．

シャドウマッピングにおいては，視点との距離情報を有する深度バッファと，シャドウマップとを比較することで，それぞれの部分が最も光源に近いか，もしくは他のプリミティブにさえぎられて影になっているかを判定する．この手法においては，シャドウマップは常に影のレンダリング対象となるオブジェクトを包含するように設定する必要がある．

4.3.2. 画素毎に計算する手法

4.2.2 と共にレイトレーシング [(3)] の手法になる．

各画素について視点とプリミティブとの最短距離が求まったところで，その距離に相当する位置を新たな視点位置として，4.2.2 と同様の手法で光源方向のレイを追跡する．4.2.2 においてはレイと衝突する全てのプリミティブとの距離計算が必要であったが，本稿においては影 (shadow) の判定に透過を考慮していないため，レイがいずれかのプリミティブと衝突することが判明した時点で追跡を打ち切る．

また本稿においては，4.2.1 との組み合わせも併せて評価する．4.2.1 と 4.3.2 の組み合わせの名称については，該当する文献を見出すことができなかったため，仮に「ハイブリッド法」と称する．

5. 各手法の比較概略

前章で述べた手法について，精度や速度の観点での要約を表 3 に示す．表の中で，「○」は要求仕様を満たすための優位性があることを示す．「△」は条件次第で要求仕様を満たすことを示す．

[*13] Khronos Group, “Conformant Products”, https://www.khronos.org/conformance/adopters/conformant-products#opencl (2015 年 11 月現在)

表 3 速度・精度面の比較

計算手法	プリミティブ毎	画素毎
形状の精度	△	○
影の精度	△ (*1)	○
GPU を用いた高速化	OpenGL	OpenCL
計算速度	○	△

(*1): 精度を要求されない用途での高速表示目的に限定される.

次に，評価対象として想定する手法の組み合わせと，本稿においてこれらの手法を区別するための名称を表 4 に示す．次章以降では，これらの手法について精度，速度の面でなされた検討の詳細を述べ, 性能評価を行う．なお，形状を画素毎に計算し影をプリミティブ毎に計算する手法については，速度面でも精度面でも優位性が見いだせなかったため，本稿では割愛する.

表 4 評価対象として想定する手法の組み合わせ

形状の計算	影の計算	本稿での名称
プリミティブ毎	N/A	シェーディング (shading)
プリミティブ毎	プリミティブ毎	シャドウマッピング (shadow mapping)
プリミティブ毎	画素毎	ハイブリッド法 (hybrid method)
~~画素毎~~	~~プリミティブ毎~~	N/A
画素毎	画素毎	レイトレーシング (ray tracing)

6. 精度に係る検討

本章では，各計算手法の精度に係る課題を検討する.

レイトレーシングを OpenCL で実装する場合，32 ビット浮動小数点で計算するのが現実的である．32 ビット浮動小数点は 10 進数で 7 桁程度の精度を有している．これは 20km のレンジに対して 2mm 程度の精度であり，小惑星近傍のミッションを任意の視点から描いたとしても，要求仕様である 10cm の精度を満たす値である.

一方，プリミティブ毎に計算する手法では，深度計算に起因する精度不足や，深度バッファの解像度不足が生じる場合がある．以下に不具合の例を示す．位置・姿勢等のパラメータは表 5 に示す．その模式的な位置関係を図 1 に示す．この配置は，様々な不具合を一つのシーンに例示するための配置であり，現実の運用を反映したものではない.

表 5 不具合が生じる例の位置・姿勢等

探査機	
位置	(0.06028, -0.10510, 0.06881)
姿勢	(0.80242; 0.42986, -0.35257, 0.21684)
小惑星	
位置	(0, 0, 0)
姿勢	(0.13272; 0.65282, 0.73086, 0.14845)
太陽	レイトレーシング用（点光源）
位置	$(1.04389\times10^{8},0.90274\times10^{8},0.37203\times10^{8})$
太陽	シャドウマッピング用（平行光源）
位置	(1.04389, 0.90274, 0.37203)
方向	(-1.04389, -0.90274, -0.37203)
カメラ	
位置	(0.11504, 0.07638, 5.00000)
方向	(0,0,1)
視野角	0.23097 度

図 1 模式的な位置関係

位置の実際の単位は [km] であるが，本稿においては 1km がプログラム上の表現で 1 となるように各表示すべきオブジェクト等のスケールを合わせている．姿勢は SPICE Toolkit[*14] が提供するクォータニオンである．太陽は点光源もし

[*14] Navigation and Ancillary Information Facility, NASA, “SPICE Conventions”, https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/05_conventions.pdf, p.13 (2015 年 11 月現在)

くは平行光源として扱い，白色光とする．影の計算にあたって，画素毎に計算する手法では点光源の値を用い，プリミティブ毎に計算する手法では，平行光源の値を用いる．カメラの視野角は，現実のカメラでは想定し難い値に設定しているが，GUI 操作により自由に拡大・縮小するような運用においては起こりうる値であり，可視化手法の中でも考慮しておく必要がある．

この配置に基づいたレイトレーシングによるレンダリング例を図 2 に，シャドウマッピングによるレンダリング例を図 3 に示す．

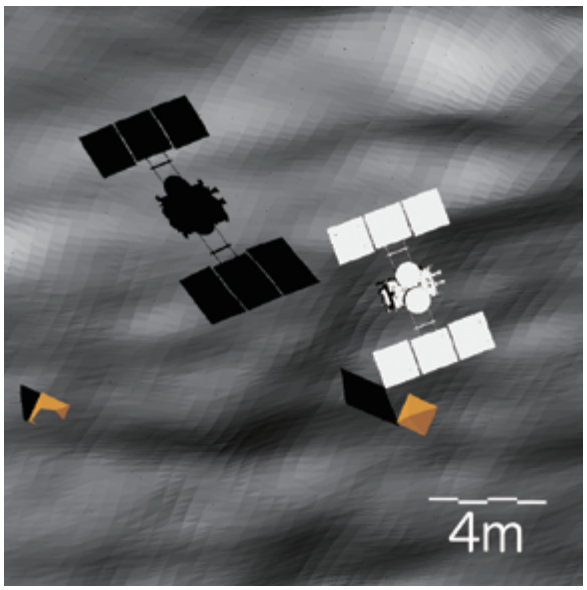


図 2 レイトレーシングのレンダリング例

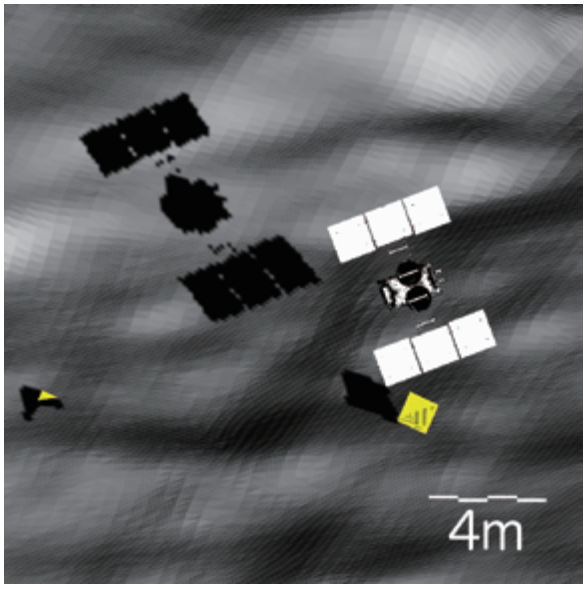


図 3 シャドウマッピングのレンダリング例

各図において，中央右付近のモデルが探査機，背景の灰色の多面体が小惑星を表している．四角錐は LRF による測距値に基づいて計算した小惑星面の推定位置を表しており，図中には 2 つ描かれている．右下の「4m」の表示は，探査機の位置での大まかな 4m の長さを表している．横棒 1 本は 1m を表している．

これらの図から読み取れる問題点は，次の通りである．

6.1.1. 深度バッファの精度不足

OpenGL における深度バッファは，奥行きの最小値と最大値を設定する必要があり，レンダリングされるのは，この設定範囲内のオブジェクトのみである．また透視投影を表現するためのパースペクティブ行列と深度バッファを併用する場合，深度バッファは近距離において高精度となり遠距離においては低精度となる．環境によっては深度バッファが 32 ビットではなく，24 ビットの精度しか得られない場合もあり [*15]，注意が必要である．

このような条件下においては，直近のオブジェクトと遠方のオブジェクトを同時に描くことが困難な場合がある．深度バッファの精度に関する課題は，この例に限らず，深度バッファとパースペクティブ行列を併用する場合に共通に発生しうる．

探査機と小惑星のランデブーについて，概ね 20km 程度の距離からタッチダウンまで描くと想定すると，視点や注視点としては 1m オーダから 10km オーダの範囲でオブジェクトを描くことになると考えられる．単純にこの想定から OpenGL の深度範囲を 1m 〜 10km 相当に設定して描いたものが図 3 である．

図 3 は，探査機の形状が崩れており，また周囲の LRF に係る四角錐も不適切なレンダリングになっている．不具合の拡大例を図 4 に示す．各図共に，レイトレーシング側の画像は，適切に形状を反映している．

[*15] Apple Inc., “OpenGL Capabilities Tables”, https://developer.apple.com/opengl/capabilities/ (2015 年 11 月現在)

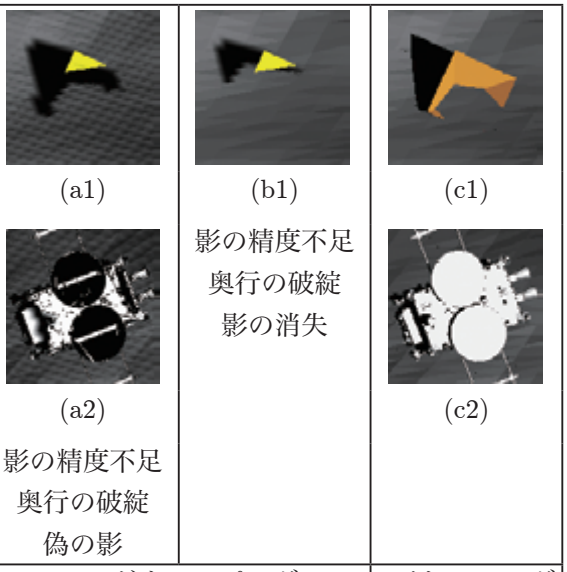


図 4 精度に係る不具合例（拡大）

図 4(a1) においては，LRF に係る四角錐と小惑星との奥行関係が破綻して，本来描かれるべき錐の一部が消えている．この例では，四角錐のサイズは 1 辺が 1m 程度であるので，そのサイズに匹敵する部分が消失しうることが見て取れる．図 4(a2) においては，探査機内のプリミティブ同士の奥行関係が破綻して，本来奥に隠れているはずの構造が露呈している．

このような問題を回避するためには，探査機や小惑星等，描くオブジェクトと視点との位置関係を常に把握しながら，レンダリングするシーンの奥行き範囲を適切に決定する必要がある．また場合によっては，精度低下が懸念されるような視点やオブジェクトの配置を禁止するような工夫も必要になると考えられる．この留意点は表 4 に掲げたシェーディング，シャドウマッピング，ハイブリッド法に共通するものである．

6.1.2. シャドウマップの精度不足

シャドウマップに起因する課題も見受けられる．シャドウマップも深度バッファであるため，6.1.1 で述べた課題が発生しうるが，これはシャドウマップを計算する際に並行投影（並行光線）を用いることで軽減しうる．その他の課題を以下に述べる．

例えば本稿で用いたシステムにおいては，シャドウマップを計算するための深度バッファのサイズは，4096×4096 ピクセルが限界であった．「はやぶさ 2」が目的としている小惑星 1999 JU_3 は，直径が 1km 弱と推定されており，これで小惑星の存在範囲をカバーするとなると，深度バッファ 1 ピクセル当たりのサイズは 25cm 四方程度となり，精査目的では要求仕様を満たさない．

また本来はありえないテクスチャが図 3 の随所に見受けられる．図 4(a1)，(a2) において，小惑星上の細かい模様は，シャドウマップの量子化誤差に起因する偽の影であり，実際には存在しないものである．本稿においてはテクスチャを単色でレンダリングしているため偽の影を見分けることが容易であるが，将来的に小惑星のテクスチャを付加した画像においては，それが本来のテクスチャであるのか量子化誤差に起因するのかを見分けるのは困難となる．

このような量子化誤差の影響を軽減するためには，シャドウマップに若干の不感域を設けて，影の判定が際どい領域では影と判定しない等の対策が必要となる．しかしながらこのような対策を講じると，逆に本来は影であるべき領域が影と判定されなくなる副作用がある．図 4(b1) は，小惑星上の偽の影が目立たなくなる程度に不感域を設けた場合の例である．LRF に係る四角錐の影は，図 4(c1) のように描かれるべきところであるが，その多くの部分が消失している．

シャドウマッピウングを用いて影を描く際には，これらの誤差を考慮しながら，影が適切に描かれるようパラメータを調整しなくてはならないが，実際問題として，運用者がこの調整を行うことは現実的ではない．

このように，シャドウマッピングは影の精度が要求される場面では利用が困難である．

7. 高速化に係る検討

OpenGL を用いてプリミティブ毎に計算する手法であれば，GPU による高速化が容易であり，速度に関する懸念は小さい．シャドウマッピングも，画像を精査する必要の無い場面では，速度面の優位性がある．またシャドウマッピングはソースコードが比較的シンプルであり，保守のメリットも勘案しつつ，精度を要求しない場面での簡易かつ高速表示の手段として候補に残すこととする．

これに対して画素毎に計算する手法は，それ

ぞれの画素に対して各プリミティブとの衝突判定を行うため，計算時間を要する．以下に，画素毎に計算する手法の高速化について検討する．

7.1. ボクセル分割

一般のレイトレーシングと同様に，4.2.2 と 4.3.2 で示した，画素毎に計算する手法においては，ボクセル分割 [7] による高速化を行う．本稿においては，下位層のボクセルは上位層のボクセルを各軸方向に二等分する [8] 方式を採用する．すなわち，上位層のボクセルから，夫々 8 個の下位ボクセルを生成する，8 分木の構成となる．

ボクセル分割は，シーンの空間に沿って分割して階層化する方法やオブジェクトに沿って分割して階層化する方法等が考えられる．シーンの空間に沿って分割する場合，シーンのレンダリング毎にボクセル分割する必要があるが，ボクセルの階層化や，階層化されたボクセルにプリミティブを割り当てる作業は OpenCL によって並列度を上げることが困難である．

本稿で扱うオブジェクトについて考えると，個々のオブジェクトの座標系において，それに含まれるプリミティブが変形することは考慮しなくて良い．個々のオブジェクトに沿ってボクセル分割する場合，階層化処理はオブジェクト毎に 1 度実施すれば，一連のシーンのレンダリングに渡って，そのオブジェクトのボクセル構成は不変であるとみなすことができる．また本稿で想定するシーンは，プリミティブ数は多いが，オブジェクト数は少ないため，オブジェクト毎にボクセル階層を持つオーバーヘッドは小さいと考えられる．

そこで本稿においては，オブジェクトに沿って 8 分木に階層化する方法を採用する．階層化されたボクセルの内，最下層のボクセルには，それぞれに（全体もしくは一部が）内包されるプリミティブを割り当てる．8 分木による階層化の例を図 5 に示す．

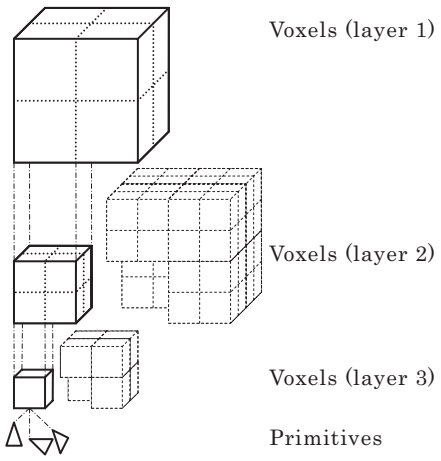


図 5 ボクセル分割の概要

本稿で採用するボクセルの階層構造においてプリミティブとレイとの衝突位置を計算するにあたっては，最初に計算対象となるレイの始点とレイの方向を，当該オブジェクトの座標系に変換する．各オブジェクトの座標系内において，最上位のボクセルからレイの衝突判定を行い，レイと衝突したボクセルについては，下位層のボクセルとの衝突判定を順次実施する．衝突判定された最下層のボクセルについては，そこに含まれるプリミティブに対してレイの衝突判定を行う．この手順を，表示すべき全オブジェクトに対して実施する．

本稿で用いたボクセル探索の概要を図 6 に示す．

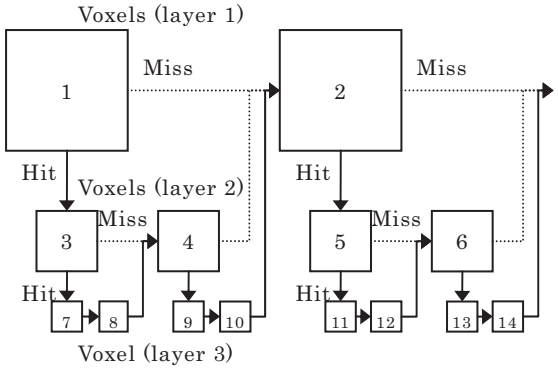


図 6 ボクセル探索の概要

図中では簡単のため 2 分木を 3 階層で表現しているが，実際の探索には 8 分木が用いられ，階層数もオブジェクトによって異なる．

また OpenCL においては，ホスト側のポインタ変数を GPU では参照できないため，ボクセルの探索には，インデックスによる間接参照を採

用した．図 6 の探索をインデックスで表現した例を表 6 に示す．

表 6 ボクセル探索のインデックス表現の例

Voxel #	1	2	3	4	5	6	7	8
Next voxel # (hit)	3	5	7	9	11	13	8	4
Next voxel # (miss)	2	...	4	2	6	...	8	4

7.2.GPU による高速化の範囲

4.2.1 と 4.3.1 で述べた，プリミティブ毎に計算する手法については，OpenGL と GPU の組み合わせで高速化する．4.2.2 と 4.3.2 で述べた画素毎に計算する手法については，OpenCL と GPU の組み合わせで高速化する．

GPU と OpenGL や OpenCL の互換性については，OpenGL，OpenCL 共に，3.1 に記した AMD 社，NVIDIA 社，Intel 社の各 GPU について動作検証を行う．

評価対象となる手法の組み合わせと，本稿で用いる各組み合わせの名称は，前述の表 4 に示した通りであるが，この中でハイブリッド法は，OpenGL と OpenCL という異なる手段でレンダリングを分担する手法となっている．すなわち，OpenGL で形状を計算した結果である画像の配列と深度バッファの配列を OpenCL に引き継いで影の計算を実施する．この引き継ぎ過程が入るため，GPU での計算時間が短い場合は引き継ぎによるオーバーヘッドが相対的に大きくなると考えられる．

OpenGL での処理と OpenCL での処理とで座標計算を一致させる必要もあるが，本稿においは座標計算用の行列を両者互換となるよう設計しているため，座標計算に関しては特にオーバーヘッドは発生しない．

8. 所要時間等の比較

8.1. 比較対象

所要時間の比較は，表 4 に掲げた手法に対して実施する．計測にあたっては原則として GPU による高速化を行うが，レイトレーシングについては，CPU による所要時間も併せて評価する．これは GPU 搭載のメモリでは処理できないような大規模形状データのプリレンダリングも模擬・訓練の中で想定しうるためである．CPU によるレイトレーシングについては，Grand Central Dispatch (GCD)[*16] によるマルチスレッド化を行う．GCD は本稿で用いたプラットフォームに限定されるものではないが，他の OS で係るアプリケーション開発を行う場合は，その互換性に留意する必要がある．またプリレンダリングにおいてはパラメータ並列の計算も可能であり，レンダリングする一連のシーン全体としての高速化にマルチスレッド化が必須とは限らない．

8.2. 比較に用いるシーン

比較に際して表示すべきオブジェクト等の配置を表 7 に示す．各値の定義等は，表 5 と同一である．これらから描かれるシーンの例を図 7，図 8 に示す．シーンのサイズは，1,024×1,024 とする．

表 7 表示すべきオブジェクト等の位置・姿勢

探査機	各シーン共通
位置	(0.04061,0.13302,0.07361)
姿勢	(0.78839; 0.51098, -0.30285, 0.16005)
小惑星	各シーン共通，4 モデル共通
位置	(0, 0, 0)
姿勢	(0.10212; 0.50216, 0.84153,0.17096)
太陽	点光源
位置	$(1.04455\times10^{8},0.90215\times10^{8},0.37176\times10^{8})$
太陽	平行光源
位置	(1.04455, 0.90215, 0.37176)
方向	(-1.04455, -0.90215, -0.37176)
カメラ a	シーン a 用
位置	(0.04306, 0.05716, 5.00000)
方向	(0,0,1)
視野角	0.41815 度
カメラ b	シーン b 用
位置	(0.00977, 0.03872, -5.00000)
方向	(0,0,1)
視野角	4.93447 度

[*16] Apple Inc., “Grand Central Dispatch (GCD) Reference ”, https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html (2015 年 11 月現在)

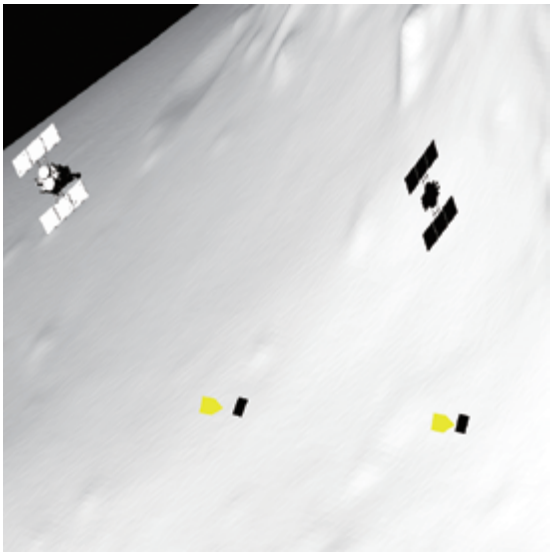


図 7 所要時間比較に用いるシーン a (近景)

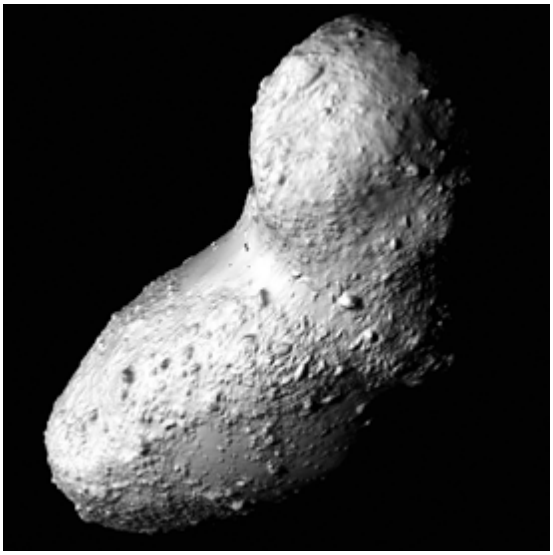


図 8 所要時間比較に用いるシーン b (遠景)

8.3. ボクセル分割の階層数

画素毎に計算する手法で用いたボクセル分割の階層数やボクセル数を, 表 8 に示す. 階層数は, 現在のところ経験則で高速となる値を求めており, 最適な階層数を自動的に求める手法については, 今後の課題である.

なお Mac mini と 3,145,728 ポリゴンモデルの組み合わせにおいては, OpenCL を用いたレンダリングで GPU のメモリ容量の限界を超えたため, OpenCL を用いたレイトレーシング及びハイブリッド法の階層数を I つ減じている. ケース 2 については GPU のメモリ容量を超える事例がさらに多く見受けられたため, OpenCL を用いる手法については GPU で正常に処理できるもののみ計測した. GPU のメモリ容量を勘案した階層数の設定も, 今後の課題である.

表 8 ボクセル分割の階層数・ボクセル数

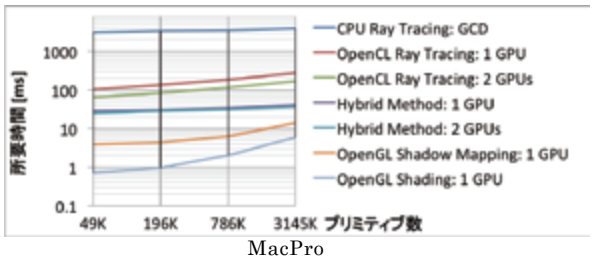
モデル	プリミティブ数	ケース 1		ケース 2	
		階層	ボクセル	階層	ボクセル
小惑星	49,152	8	37,874	9	150,515
	196,608	9	151,186	10	600,321
	786,432	10	601,285	11	2,384,543
	3,145,728	11	2,387,458	12	9,460,613
探査機	213,405	9	114,047	10	577,835

8.4. 比較結果

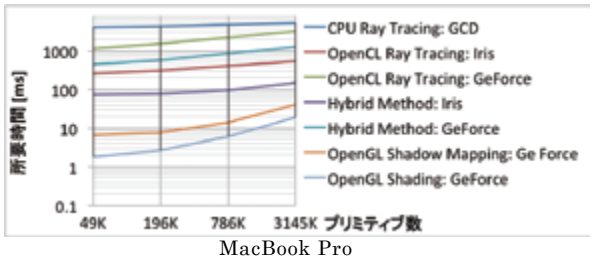
各手法においてレンダリングに要する時間の比較結果を図 9, 図 10 に示す. 横軸は小惑星のポリゴン数, 縦軸はシーン 1 回のレンダリング時間を表している. 小惑星のポリゴン数は, 1 段詳細化されるごとに 4 倍のプリミティブ数となっているため, 事実上横軸はログスケールとなっている.

レンダリング時間の計測にあたっては, 各条件ごとに同一のレンダリング計算を 100 回連続して実施し, その平均をとった.

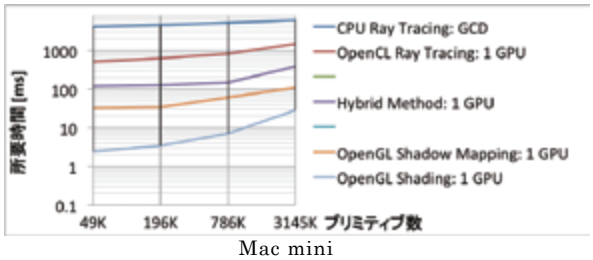
ケース 2 で正常にレンダリングできた事例については, 図 11, 図 12 に, ケース 1 との所要時間比を示す.



MacPro

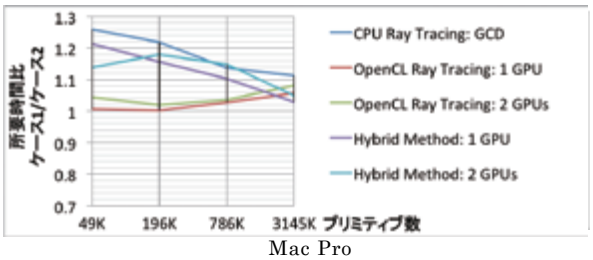


MacBook Pro

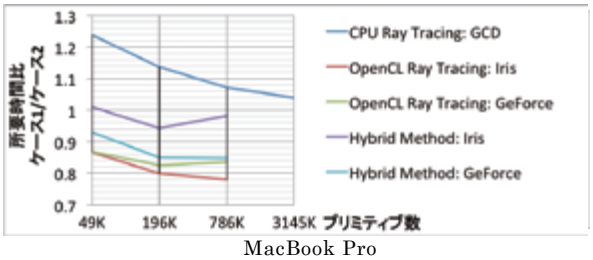


Mac mini

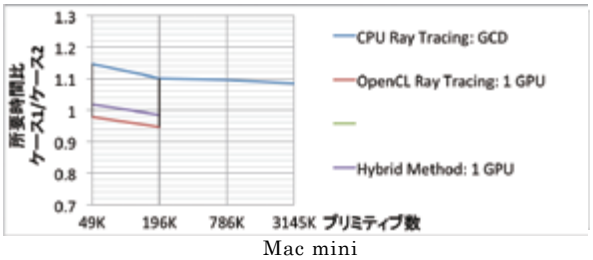
図 9 レンダリング速度比較（ケース 1, シーン a）



Mac Pro

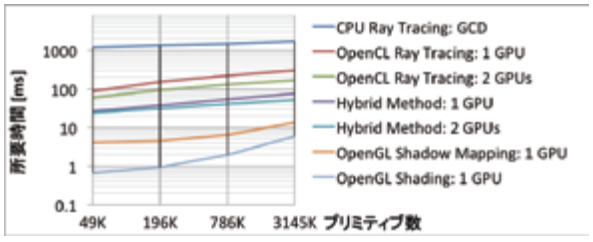


MacBook Pro

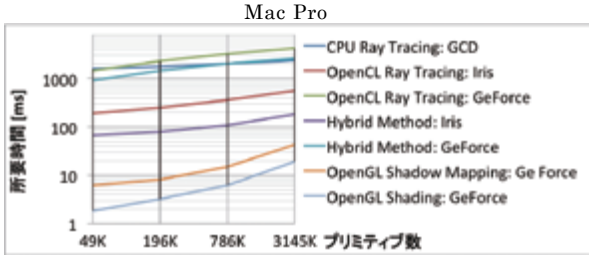


Mac mini

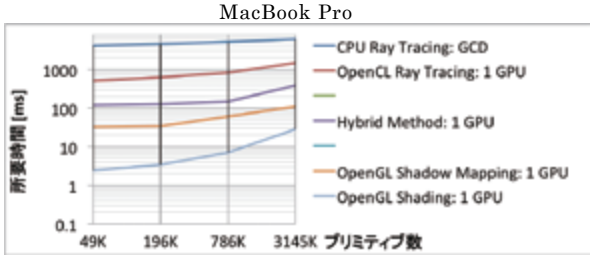
図 11 ボクセル階層数による所要時間比較(シーン a)



Mac Pro

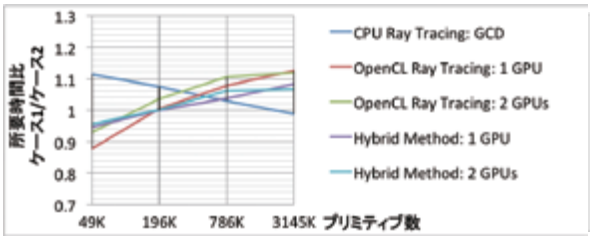


MacBook Pro

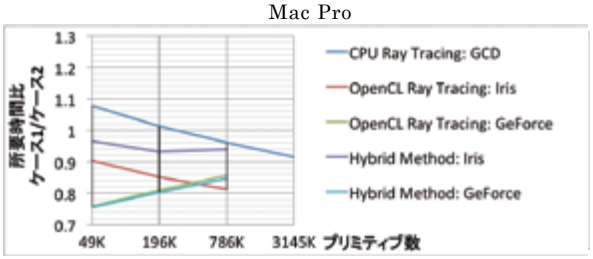


Mac mini

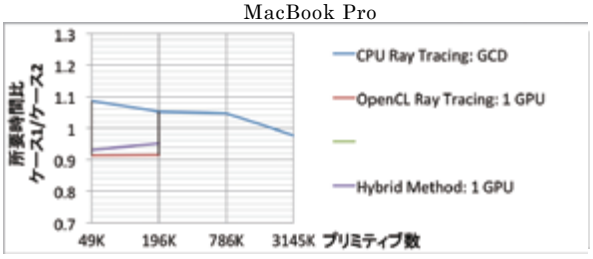
図 10 レンダリング速度比較(ケース 1, シーン b)



Mac Pro



MacBook Pro



Mac mini

図 12 ボクセル階層数による所要時間比較(シーン b)

いずれのケースが高速にレンダリングできているかについては，実行条件によって様々であり，一概にいずれの階層数が優れていると結論付けるのは難しい．階層数等，ボクセル分割に係るチューニングは，今後ソフトウェアを開発するプラットフォームが決定した後の検討事項になると考えられる．

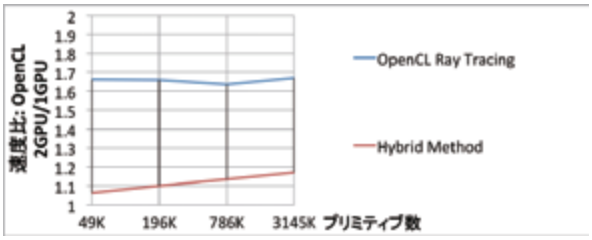
次に測定環境毎の結果を述べる．

(a)Mac Pro 環境

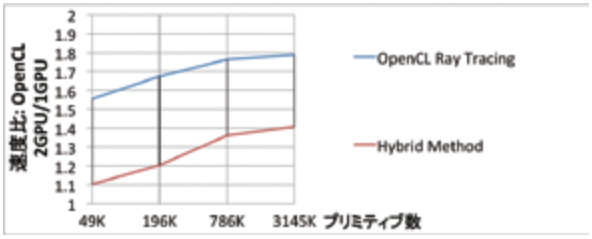
Mac Pro は同一型の GPU を 2 基搭載しているため，OpenCL を用いた高速化の評価にあたっては，1GPU 単独の評価と 2 GPU 並列の評価を行った．

本稿における評価の中で最もポリゴン数の多い 3,145,728 プリミティブの場合でも，1GPU の場合，ハイブリッド法が高速表示の要件（10fps）を満たしており，OpenCL を用いたレイトレーシングもかろうじて精細表示の要件（3fps）を満たしている．ハイブリッド法において，深度バッファに係るパラメータ設定を適切に行えば，あらゆる場面で高精細かつ高速の表示が実現する可能性がある．2GPU のレイトレーシングは余裕を以て精細表示の要件を満たしている．

1GPU と 2GPU との計測結果を比較し，2GPU 時の性能向上比を求めたものを図 13 に示す．レイトレーシングにおいては，性能向上の効果が大きいが，ハイブリッド法においては，OpenGL でのレンダリングが 1GPU に限定されることと OpenGL・OpenCL 間のデータ引き継ぎのオーバーヘッドもあって，性能向上の効果は低い．



シーン 1



シーン 2

図 13 1GPU と 2GPU での速度比

CPU を用いたレイトレーシングにおいては，1 秒〜数秒で 1 枚のレンダリングができており，大規模形状モデルを用いたプリレンダリングにおいては，64GB のメモリを有効に活用できるものと考えられる．

(b)MacBook Pro 環境

MacBook Pro 環境においては，OS の制約により，ディスプレイ表示用（OpenGL 実行用）の GPU は自動的に GeForce が割り当てられた．図 9 から図 12 におけるハイブリッド法の GPU は，OpenCL を実行した GPU を表している．

ここでは Iris Pro を用いたハイブリッド法が精細表示の要件を満たしている．速度面ではシャドウマッピングもしくは Iris Pro を用いた低ポリゴン数のハイブリッド法が高速表示の要件を満たしている．このようなモバイル環境でのプレゼンテーション等においては，シャドウマッピングとハイブリッド法を適切に使い分けることにより，実用的な表示が実現できる可能性がある．

MacBook Pro と，Iris Pro の低速版にあたる GPU を搭載した Mac mini とで，シェーディングの所要時間差は 10ms 程度である．MacBook Pro にて OpenGL を含めて Iris Pro で実行したとしても，本稿で計測した所要時間と比較して，その差は僅かであると想像される．これは GPU 内蔵の CPU パッケージでリアルタイムレンダリングが実用化されうることを示唆している．

GeForce を用いた OpenCL 関連の所要時間が大きくなっているが，これは OpenCL と GeForce のアーキテクチャの不整合によるものなのか，ディスプレイ表示と OpenCL を兼用したことによるものなのかについて，精査する余地がある．MacBook Pro の OS には，ディスプレイ表示用の GPU を手動で切り替えるような環境設定が用意されていないため，原因切り分けのためには，別途 GeForce を OpenCL 専用に供した場合の性能評価を行う必要があると考えられる．また NVIDIA 社のアーキテクチャ専用の CUDA[*17] による評価も選択肢の一つであろう．

(c)Mac mini 環境

Mac mini 環境の性能やメモリ割当では実利用

[*17] NVIDIA Corporation, “Parallel Programming and Computing Platform | CUDA | NVIDIA | NVIDIA”, http://www.nvidia.com/object/cuda_home_new.html (2015 年 11 月現在)

に供することは困難と考えられるが，開発時の機能確認や互換性確認の用途には利用可能であると考えられる．

9. おわりに

探査機運用者向けの可視化手法における，影の正確な描写と応答速度の向上について検討した．

高精度の描写に長けているレイトレーシングについては，インタラクティブなレンダリングにおいても，環境によっては実用的な応答時間で利用可能であることが示唆された．本稿においては，そのために不要な計算要素の省略や，レイの追跡アルゴリズムの簡略化を検討し，また OpenCL による高速化を施した．

ハイブリッド法は，本稿で述べた運用者向け可視化の範囲においては，深度バッファのパラメータを考慮することで精度を維持しつつ，レスポンスの向上も期待でき，一層の調査・研究が有望であると考えられる．

今後はアルゴリズム及び GPU 選定の双方から調査・検討を行い，高精度な影の描写をリアルタイムに実現できるように改善を図って行く予定である．

また執筆時点で用いているレイトレーシングやハイブリッド法のアルゴリズムは，係る行列計算等を忠実にコーディングしたものであり，GPU の特性を考慮したチューニングには至っていない．可視化ソフトウェアを搭載する機種が選定された後は，GPU の特性を考慮したチューニングを行うことでさらなる速度向上が期待できる．

また本稿で評価に用いた GPU は，最速のもの（FirePro D700）でも，執筆時点で既に販売開始から 2 年経過した製品であり，執筆時点の性能としては，この 2 倍以上の演算性能を実現した製品が見受けられる [*18,*19]．ソフトウェアの改良のみならず，最新のハードウェアを用いることによって，2018 年の運用時点では，さらなる速度向上が見込めるものと期待される．

また，本稿の評価の中には，ディスプレイ表示に割り当てられている GPU を OpenCL に供した場合の評価も含まれている．その場合，速度評価のプログラム実行中は他のウィンドウ操作や文字入力等に遅延が見受けられることがあった．本稿のような機能・性能を評価する段階では他の操作を控えての実行が可能であるが，実運用環境で GUI 操作をする場合，操作に係る遅延も考慮する必要がある．速度向上と操作性の向上とのトレードオフは今後の課題である．

参考文献

1. 「はやぶさ」の小惑星イトカワ探査軌道・姿勢の可視化 . 三浦昭 , 山本幸生 , 吉川真 . 2014 年 , 宇宙科学情報解析論文誌 第 3 号 , ページ : 7-15.
2. はやぶさの軌跡の可視化 : タッチダウン時の位置推定 . 三浦昭 , 山本幸生 , 吉川真 . 4, 2015 年 , 宇宙科学情報解析論文誌 第 4 号 , ページ : 173-183.
3. Some techniques for shading machine renderings of solids. Appel, Arthur. 1968, AFIPS '68 (Spring) Proceedings of the April 30--May 2, 1968, spring joint computer conference, pp. 37-45.
4. Bidirectional Reflectance Spectroscopy 1. Theory. Hapke, Bruce. B4, Journal of Geophysical Research, Vol. 86, pp. 3039-3054.
5. 再び宇宙大航海へ臨む「はやぶさ 2」第 5 回航法誘導制御 . 照井冬人 . 399, 2014 年 , ISAS ニュース , ページ : 5.
6. Casting Curved Shadows on Curved Surfaces. Williams, Lance. 1978, SIGGRAPH '78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques, pp. 270-274.
7. Reddy, Dabbala Rajagopal and Rubin, S. Representation of three-dimensional objects. s.l. : Department of Computer Science, Carnegie Mellon University, 1978. pp. 1-16.

[*18] From Wikipedia, the free encyclopedia, "GeForce 900 series", https://en.wikipedia.org/wiki/GeForce_900_series (2015 年 11 月現在)
[*19] From Wikipedia, the free encyclopedia, "AMD Radeon Rx 300 series", https://en.wikipedia.org/wiki/AMD_Radeon_Rx_300_series (2015 年 11 月現在)

8. Space Subdivision for Fast Ray Tracing. Glassner, Andrew S. 1984, Computer Graphics and Applications, IEEE, pp. 15-24.
9. Shadow Algorithms for Computer Graphics. Crow, Franklin C. 1977, SIGGRAPH '77 Proceedings of the 4th annual conference on Computer graphics and interactive techniques, pp. 242-248.