# 小惑星に対する重力場解析計算の GPU による高速化 [a]

河野 郁也, 中里 直人, 平田 成 (会津大), 松本 晃治 (国立天文台)

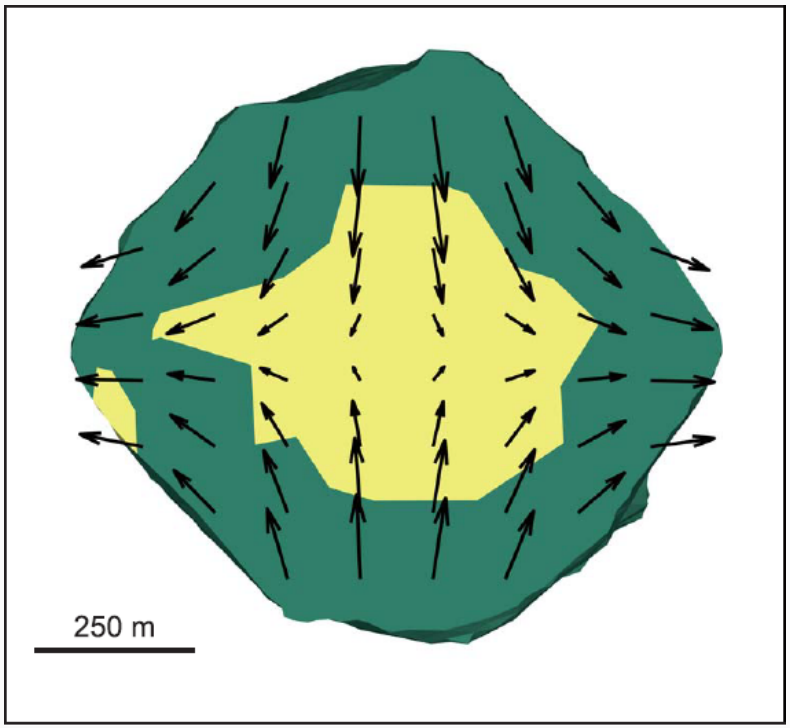
月惑星探査アーカイブサイエンス拠点集会分科会 2021/10/11

[a]本研究は文部科学省特色ある共同研究拠点の整備の推進事業
JPMXP0619217839 の助成を受けています.

# 小惑星 (太陽系小天体) に対する重力場解析

重力場計算コード GFandSlope (会津大) を高速化する.

- 天体形状を多面体で近似した形状モデルに対して計算を行う.
  - リュウグウ表面からの衝突放出物の軌道計算 (Mitsuta ら, 2012)
  - リュウグウ表面における重力勾配計算 (Watanabe ら, 2019)




そもそも，小惑星表面の重力場を数値計算で求めるということは…

- 小惑星の構造や，天体表面での地質活動への影響などを調べる
  - 小惑星の自己重力が弱く，不規則な形状を持っているため，表面の重力場は複雑 → 解析的に求めることは困難
  - 天体の形状モデルを使った数値シミュレーションを行う

# GFandSlope の高速化・並列化

- 重力場中の複雑な形状の天体 (小惑星) を評価
- 多角形を貼り合わせた多面体で近似し，各面において物理量を計算
  - 重力ポテンシャル, 引力, ラプラシアン
- Werner, Scheeres (1996) の論文を元にした計算コード
  - 重力場計算のために，多面体を多角形の集合で近似する理論
  - 具体的な計算処理並列化の展望と，高速化の期待で締められている．

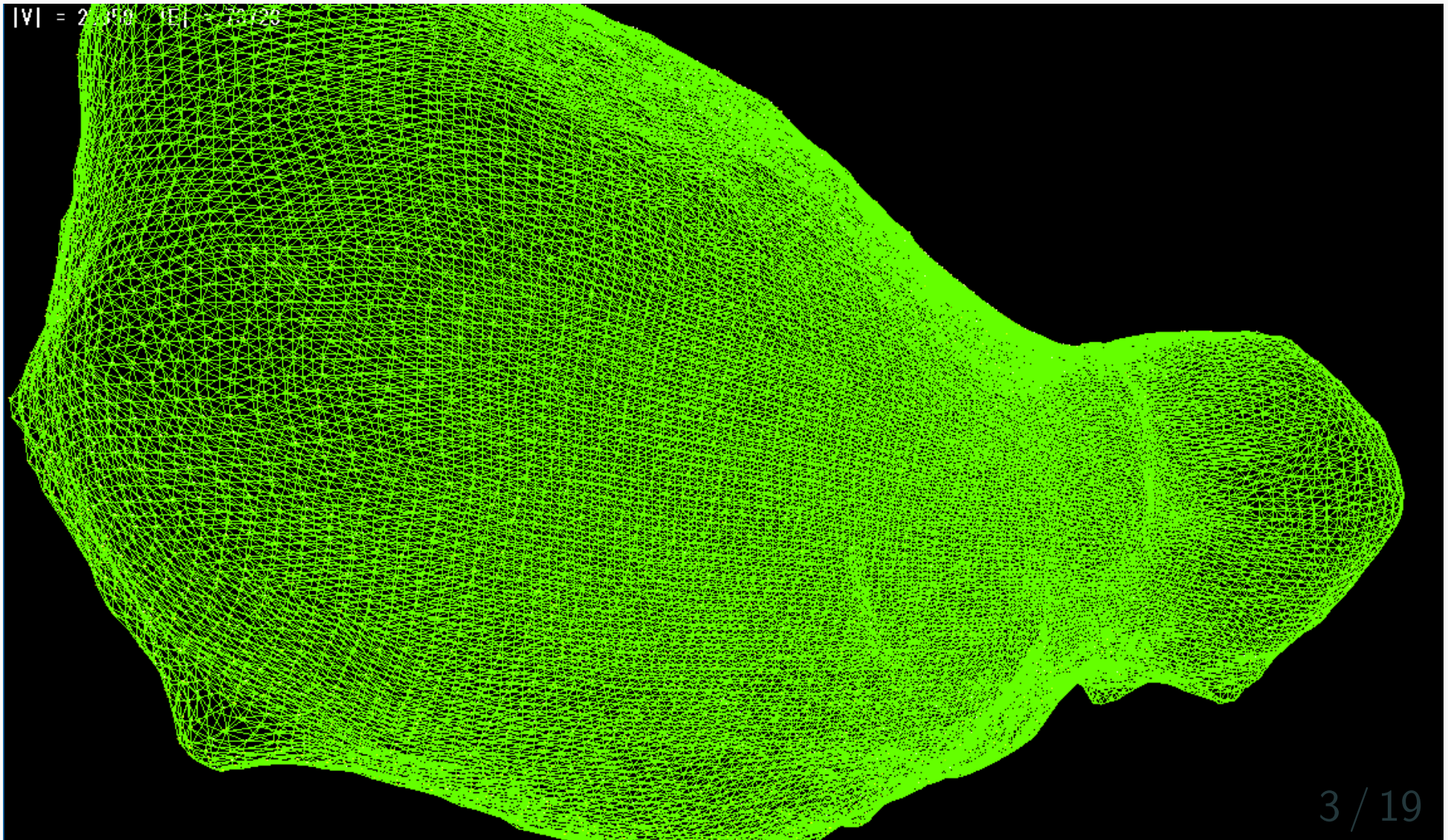
本研究以前の GFandSlope は，
逐次計算の C++コードのみがある

- アルゴリズムとしては N 体計算の部類
- 高解像度の入力モデルに対する大規模計算には厳しい

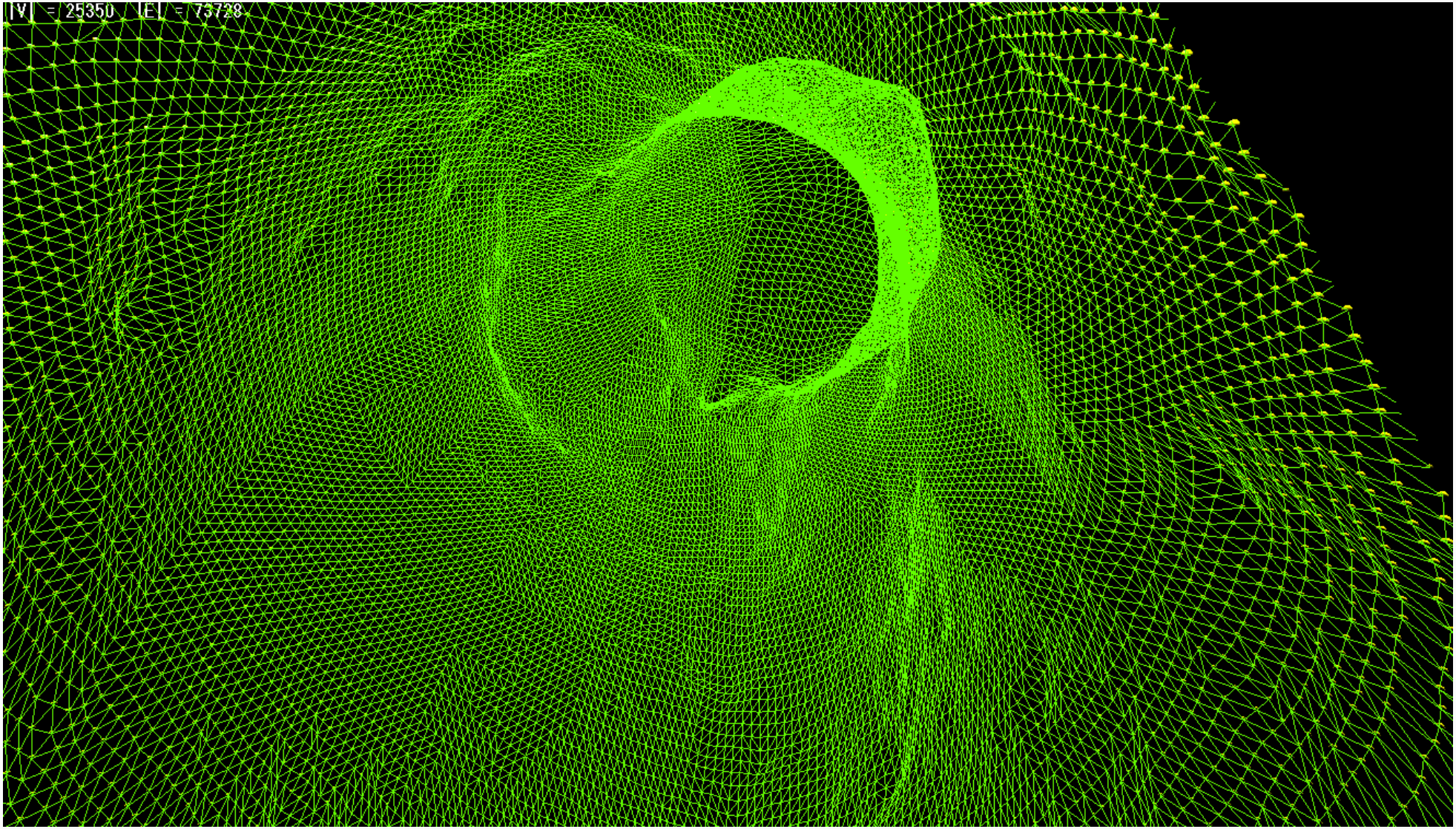
```
1: for f in faces do
2:     for e in edges do
3:         Calculation between f and e
4:     end for
5:     for f′ in faces do
6:         Calculation between f and f′
7:     end for
8: end for
```

# GFandSlope で扱う入力データ (を可視化したもの)

- 小惑星イトカワ (25143 Itokawa)
  - 多面体の頂点座標，辺・面の構成情報を持つファイルから取得



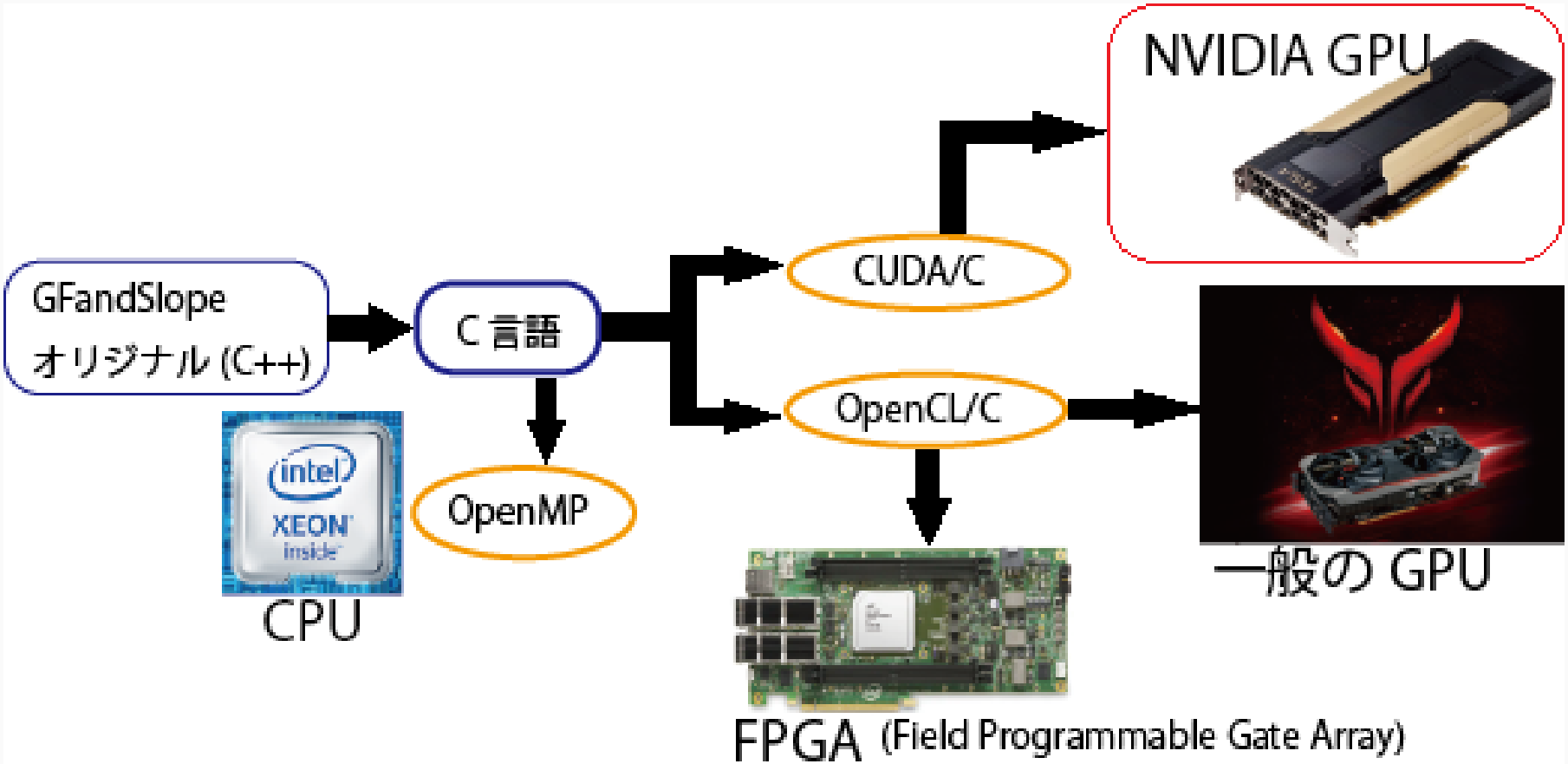
# 可視化した形状モデルの内部




- 表面を三角形のタイルで貼り合わせたモデル
  - 頂点数 25350, 辺の数 73728, 三角形の数 49152

# GFandSlope の高速化・並列化

- 「並列処理」によって、重力場計算を高速化する.
- 重力場計算は並列性が高く，特に GPU の性能を活かした高速化が可能




一般の研究目的で普及している NVIDIA Tesla V100 での利用を想定して，CUDA/C コードの実装と精度評価を行う.

# GFandSlope におけるポテンシャル計算

$G$: 重力定数, $\sigma$: 密度定数 (GFandSlope では密度分布が一定と仮定),
$\mathbf{r}$: 計算を行う面の重心を基準とした，各辺・面への位置ベクトル

- ポテンシャル

$$U = \frac{1}{2} G\sigma \sum_{e \in edges} \mathbf{r}_e \cdot \mathbf{E}_e \cdot \mathbf{r}_e \cdot L_e - \frac{1}{2} G\sigma \sum_{f \in faces} \mathbf{r}_f \cdot \mathbf{F}_f \cdot \mathbf{r}_f \cdot \omega_f \tag{1}$$

- 引力

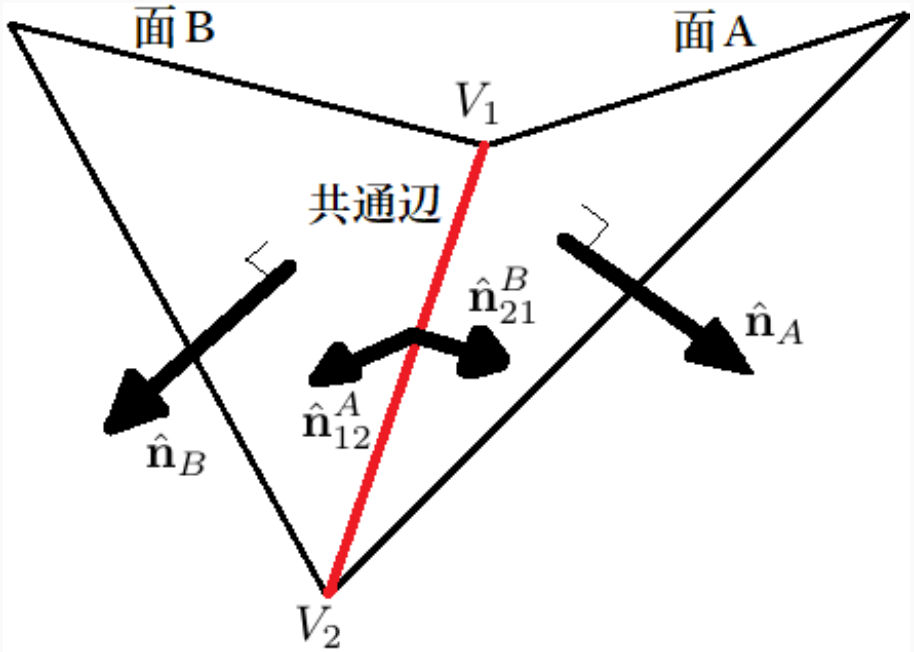
$$\nabla U = -G\sigma \sum_{e \in edges} \mathbf{E}_e \cdot \mathbf{r}_e \cdot L_e + G\sigma \sum_{f \in faces} \mathbf{F}_f \cdot \mathbf{r}_f \cdot \omega_f \tag{2}$$

- ラプラシアン

$$\nabla^2 U = -G\sigma \sum_{f \in faces} \omega_f \tag{3}$$

# GFandSlope におけるポテンシャル計算

**E (F)**: 任意の 2 つの面 (三角形) が共有する 2 頂点を結ぶ辺に対して，
面・辺の法線ベクトルの直積をとったもの ($3 \times 3$ 行列形式)




$$\mathbf{E}_e = \mathrm{E}_{12} = \hat{\mathrm{n}}_A(\hat{\mathrm{n}}_{12}^{A})^T + \hat{\mathrm{n}}_B(\hat{\mathrm{n}}_{21}^{B})^T \tag{4}$$

$$\mathbf{F}_f = \hat{\mathrm{n}}_f(\hat{\mathrm{n}}_{12}^{f})^T \tag{5}$$

面 ($f \in faces$) に対しては，面の法線ベクトル同士で直積をとる．

# ポテンシャル計算: 辺 (または面) に対するファクター

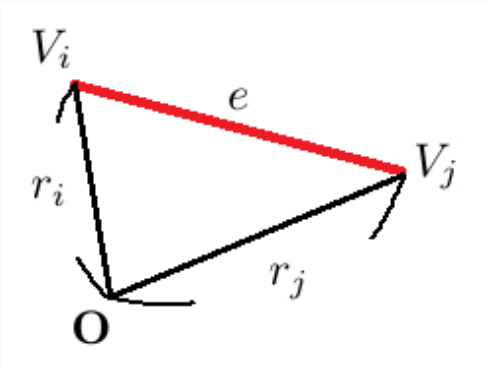
$L(\omega)$: ポテンシャルの式を導出時に現れる積分項の置き換え

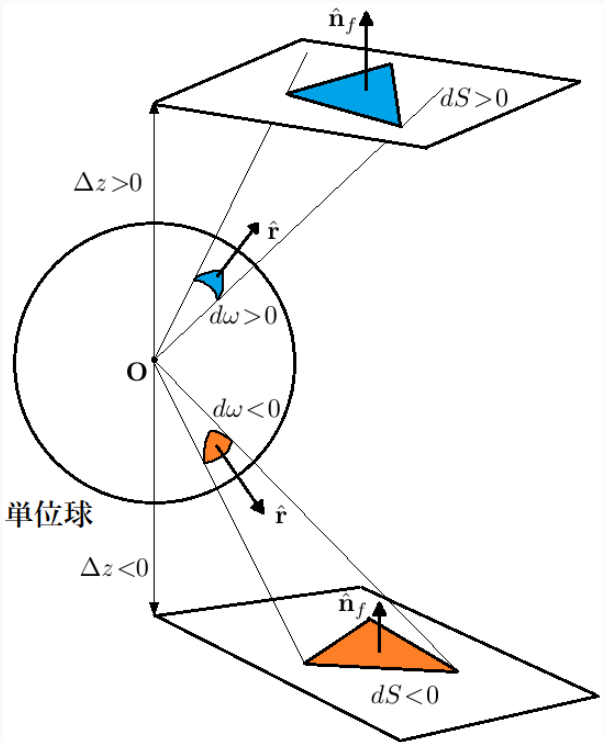
- 面 $f$ を構成する辺 $e = (i, j)$ に対する積分

$$L_e = \int_e \frac{1}{r} ds = log \frac{r_i + r_j + e_{ij}}{r_i + r_j - e_{ij}} \tag{6}$$

- 面に対する立体角 (= 単位球面上へ射影した時の符号付き面積)

$$\omega_f = \iint_{triangle} \frac{\Delta z}{r^3} ds = 2 \arctan \frac{\mathrm{r}_i \cdot \mathrm{r}_j \times \mathrm{r}_k}{r_i r_j r_k + r_i(\mathrm{r}_j \cdot \mathrm{r}_k) + r_j(\mathrm{r}_k \cdot \mathrm{r}_i) + r_k(\mathrm{r}_i \cdot \mathrm{r}_j)} \tag{7}$$



# GFandSlope のアクセラレータ向けコード化のための検討

オリジナルの GFandSlope コードが C++での実装されている

- STL コンテナ (vector, list), イテレータによるデータへのアクセス
- 外部ライブラリの使用, spice(NASA), vector3(銭谷)
- クラスを用いたオブジェクト指向型の実装

→ 並列処理フレームワーク (CUDA/OpenCL) との相性が良くない

入出力や重力場計算本体，その他処理が混在しており，アクセラレータ上で高速化したいコードの抽出が困難

- データ構造の変更を行い，C 言語へ一旦移植
  - クラス構造を分解 (単純に構造体にする)
  - STL コンテナは純粋は配列で実装し直す
  - データ読み込み時に，各三角形が使用する頂点・面 (隣接情報) をまとめる
    - 予め隣接リストとして抽出し，整列する

# GFandSlope のアクセラレータ向けコード化のための検討

- 座標は構造体で作成
- spice, vector3 などの外部ライブラリの置き換え
  - GPU カーネルにそのまま組み込めない (vector3 は C++用)
  - 基本的な代数操作しか使っていないので，独自に実装しなおす
    - 3 要素の 1 次元配列を 3 次元幾何ベクトルとして見た演算
- IO の計算ループ外への追い出し
  - 2 重ループ内側で逐次入力 (fscanf) がある
  - GPU カーネル化にも邪魔になるため，計算ループ外に独立させる

CPU での事前評価: Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz
コンパイラは gcc (ver.5.4.0) で実行　(入力ポリゴン数 49152)

- オリジナル C++コード → 4900 秒 $\sim$ 1 時間 20 分
- 移植 C コード → 1323 秒

# GPU 実装と評価

作成した C コードを GPU で計算できるように変更 (CUDA/C)

- 高速化したい処理 (重力場計算) を抜き出し，GPU コードを作成 (1 つの面の計算を 1 スレッドが担当)
- それ以外 (入出力やその他処理) は CPU 上で実行し，重力場計算に関するデータのみ GPU と通信・転送
- 浮動小数点演算は全て倍精度 (元の C++コードの実装に準拠)

| | Tesla K20 | Tesla V100 | GeForce RTX 3090 |
|---|---|---|---|
| GPU コア数 | 2496 | 5120 | 10496 |
| 周波数 | 706MHz | 1370MHz | 1395MHz |
| 倍精度理論性能 (DP) | 1.17TFlops | 7.0TFlops | 556.0GFlops |
| 単精度理論性能 (SP) | 3.51TFlops | 14.0TFlops | 35.58TFlops |

**Table 1:** CUDA コードの性能評価に用いる NVIDIA GPU とそのスペック

- 元の C++コード (4900s) に比べて，どの程度高速に計算できるか？
- イトカワの複数の解像度モデルを入力データとして，計算時間を計測
  - ポリゴン数: 50K, 200K, 780K, 3.1M

# GPU による並列化 GFandSlope の計算時間

| GPU \ モデル解像度 | 50K | 200K | 780K | 3.1M |
|---|---|---|---|---|
| Tesla V100 | 0.45s | 7.07s | 1m52s | 29m23s |
| Tesla K20 | 3.54s | 56.85s | 14m58s | 3h59m |
| RTX3090 | 5.12s | 1m18s | 20m38s | 5h30m |

**Table 2:** 各入力モデルに対する NVIDIA GPU 上での計算時間 (倍精度演算)

- v100 で 49152 ポリゴンの形状モデルの計算が 0.45s で完了
- CPU で実行した元の C++コードと比べる (4900s) と比べると，約一万倍高速に計算できていることになる
- 計算時間の増加傾向はほぼ $O(n^2)$
- RTX は倍精度演算における理論性能自体が低いため，旧世代 GPU の K20 にも劣る

# 倍精度コードの Tesla V100 上での実行演算性能

最も短時間で計算を終えた NVIDIA Tesla V100 GPU での計算について，実行時間から演算性能 $N_{perf}$(1 秒間あたりに実行した浮動小数点演算数) を算出.

- カーネルコードに記述した演算数を計上，除算・平方根は 2 演算扱い

GPU 上で行われる浮動小数点演算の数は

ポリゴン数 $N_P = 3145728$, 辺の数 $N_E = 4718592$ (3.1M モデルの場合) の時，

$$N_{op} = (197 + N_E * 175 + N_P * 174) * N_P \tag{8}$$

v100 上で 30 分で計算を終えていることから，演算性能は

$$N_{perf} = N_{op}/1800(sec.)/10^{12} = 2.4\mathrm{TFlops} \tag{9}$$

v100 の倍精度理論性能: $\sim 7\mathrm{TFlops}$

→ ピーク性能の 30%弱　(FMA を明示的に使っていないため)

# 演算の単精度化による高速化の検討

- 安直に全てを単精度 (float 型変数) に変えて実行する
- 計算結果の精度が維持できるならば約 2 倍速くなる (RTX では 64 倍)

→ RTX で高速に計算できるならば，計算資源の用意 (金銭コスト) も抑えられる．

| GPU \ モデル解像度 | 50K | 200K | 780K | 3.1M |
|---|---|---|---|---|
| Tesla V100 | 0.23s | 3.77s | 49.35s | 12m45s |
| Tesla K20 | 1.65s | 24.75s | 6m38s | 1h44m |
| RTX3090 | 0.17s | 2.27s | 35.00s | 9m22s |

**Table 3:** 各入力モデルに対する NVIDIA GPU 上での計算時間 (単精度演算)

# 演算の単精度化による高速化の検討

完全倍精度演算における計算結果との相対誤差を調べる．

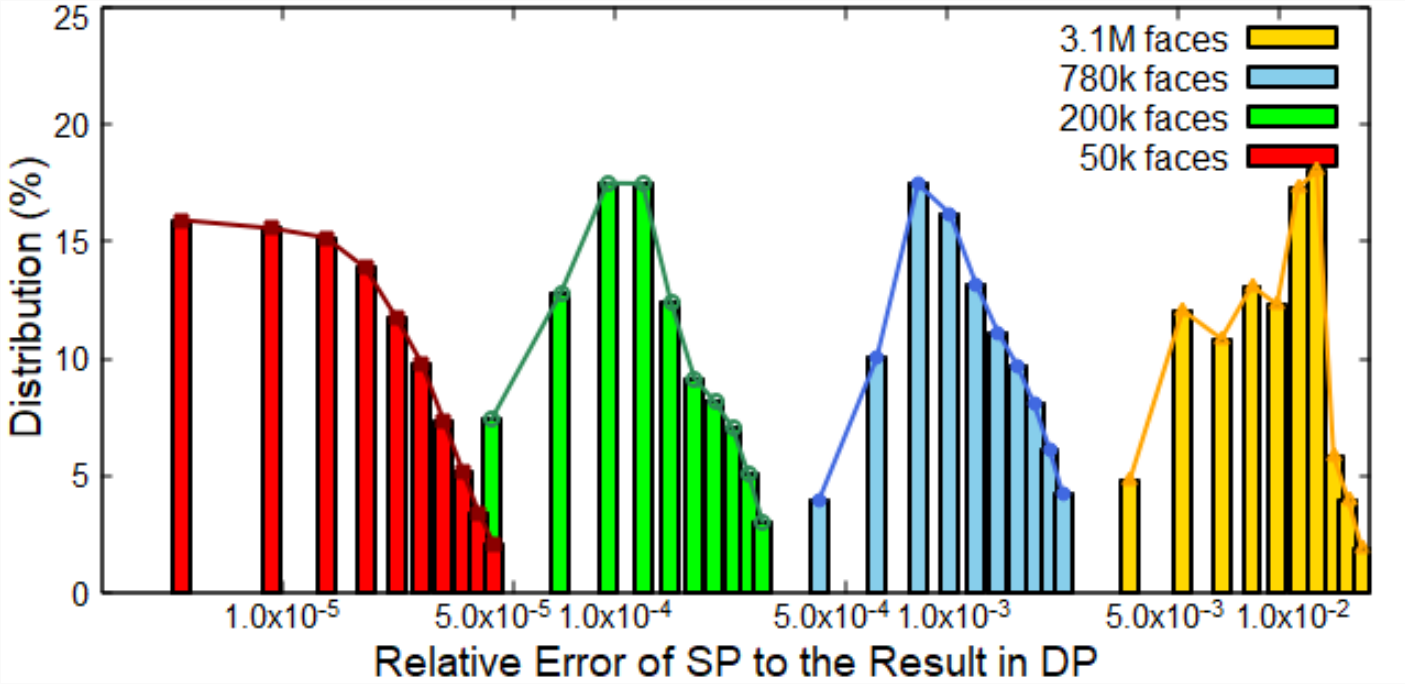



**Figure 1:** 完全単精度演算による計算結果 (ポテンシャル) の相対誤差

- 最も小さい入力モデルでは，各値の相対誤差は $5.0 \times 10^{-6}$ 付近に集中
- モデルの解像度を上げると誤差の増大が目立つ．
- 3.1M ポリゴンでは，40%のポテンシャル値の相対誤差が 0.01 以上
- 高速にはなるが，計算精度の面で実用性に欠く

# 混合演算 (単精度+倍精度) による高速化

- 倍精度演算が必要な場所を特定して，一部を倍精度で行う．
- 式 (1)〜(3) に現れる，総和計算に関してのみ倍精度で処理する．

| GPU \ モデル解像度 | 50K | 200K | 780K | 3.1M |
|---|---|---|---|---|
| Tesla V100 | 0.30s | 5.19s | 1m7s | 17m55s |
| RTX3090 | 1.87s | 28.14s | 7m29s | 2h6m |

**Table 4:** 計算時間 (単精度+総和部分のみ倍精度演算)

- 一部倍精度演算になることにより，完全単精度よりは計算時間はやや長くなる (完全倍精度より速く，完全単精度より遅い)

# 混合演算 (単精度+倍精度) による高速化

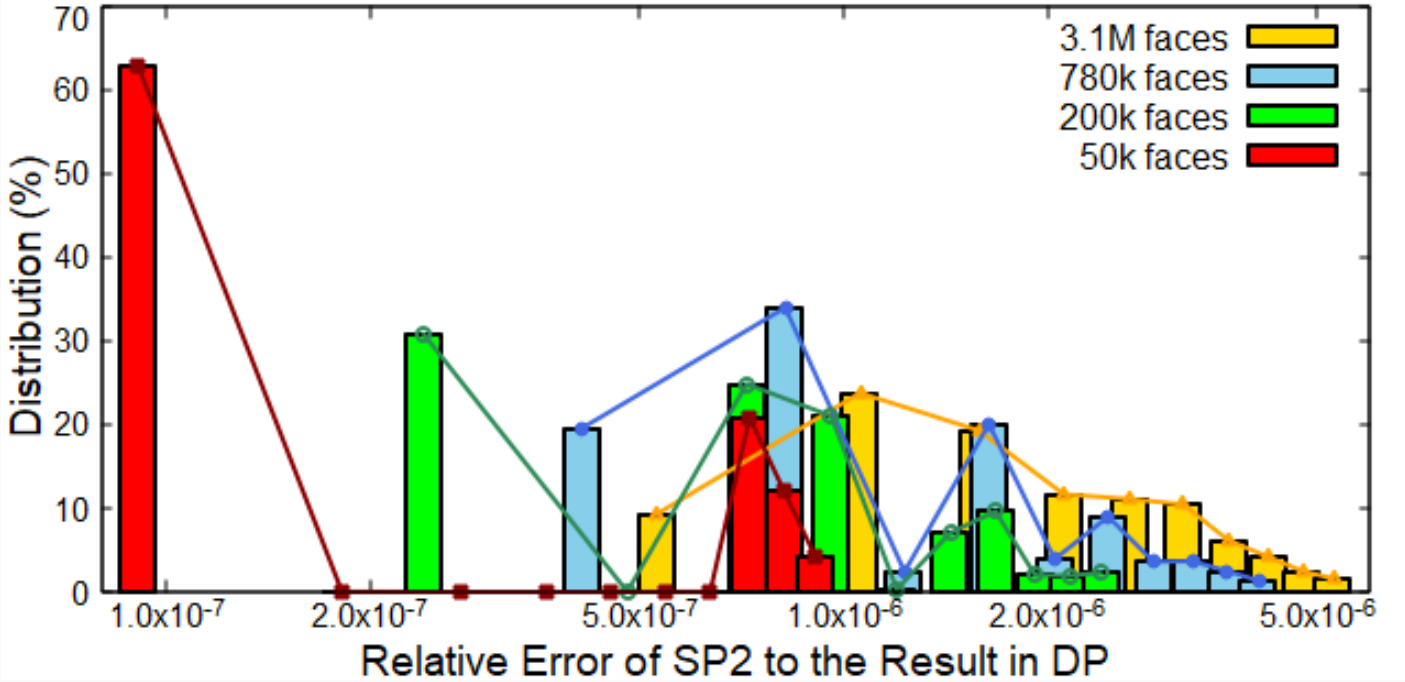
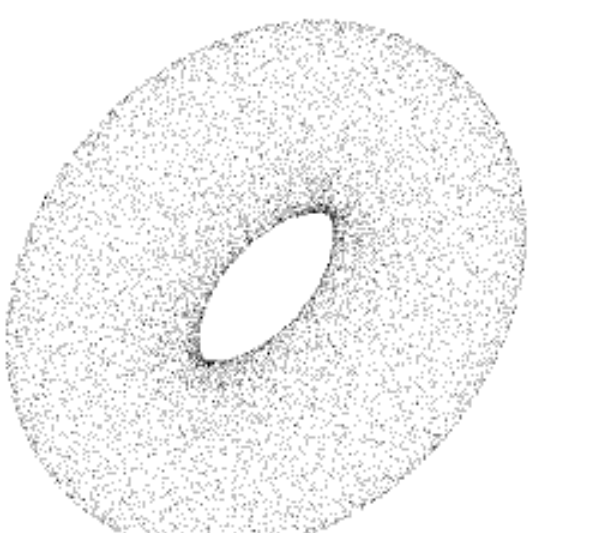



**Figure 2:** 総和部分のみ倍精度の計算結果 (ポテンシャル) の相対誤差

- 各モデル (解像度別) の誤差が大きく重なって見える → 入力モデルの高解像度化による誤差の増大が緩和された
- 多くの相対誤差は $10^{-6}$ 付近に分布し，実用上十分小さい
- かつ，完全倍精度に比べて 1.64 倍速い (V100 で 3.1M モデルを 18 分)

# 高速化のための考察・検討

- GPU によるオリジナルコードの高速化 (モデルの密度一定) には成功
  - 元計算を拡張し，密度分布を考慮したより実用的な計算がしたくなる．
- 重力場計算アルゴリズムが複雑化することで，並列処理の演算量も増加
  - 特に，高解像度モデルにおける計算時間のさらなる増大は避けられない
  - 例えば，リュウグウ (10M ポリゴン〜) の解析には所要時間が大きすぎる
  - 並列処理における高速化・最適化も必要
- ツリー法 ($N$ 体計算そのものの計算量を減らす方式) の導入
  - 領域分割し，距離の離れた (重力作用の小さな) 粒子らをまとめて計算
  - モデル上の多角形同士の重力計算となる **GFandSlope** には適さない
  - 点群 (形状モデルを点の集合に変換) の利用 (Kanamaru ら, 2019)



- 混合精度演算についても再度検討
  - arccos などの複雑な演算があるため，「総和だけ倍精度」がこの問題にとって最適かは不明
  - 自動チューニングの手法に倣って，単精度/倍精度の配分を最適化

# おわりに

- 小惑星研究のための重力場解析アプリケーション GFandSlope を，GPU を用いた並列処理により，従来よりも飛躍的に高速に実行できるように
  - データ構造を改めてライブラリ依存をなくし，1GPU 上で解析を手軽に実行できるようになった
  - 従来の逐次計算に比べて数千倍以上の高速化に成功し，小惑星研究の円滑な遂行に貢献する
- 今回の並列化・高速化によって，GFandSlope の解析アルゴリズムそのものの更新も検討
  - 内部構造を持った (密度が一定ではない) 形状モデルの解析へ
  - より大きな形状モデルを高速に計算できるよう，並列処理の観点から GPU カーネルコードの最適化を進める

- 本研究の成果となる GPU コードは，オリジナル版 GFandSlope と同じく公開の形にできるように準備を進めている
- 詳しい結果をまとめた論文は，国際会議 MCSoC2021 (12 月) で発表予定