Date: Fri, 05 Apr 2013 18:50:27 +0900
From: Tsukasa NAKANO
To: “TSUCHIYAMA, Akira”, Kentaro Uesugi, “NAKASHIMA, Yoshito”, Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL

みなさま、

GSJ/AIST の中野 司です。古い話ですが、2011 年の夏前にお送りした E-mails で画像や STL データとして与えられた物体像の凸包（convex hull）を抽出するプログラムの話をしました：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/ch.pdf

この中に書いた物体像の凸包（の形状）の「スペクトル解析」用のアルゴリズムについて考えていたところ、以前に会津大学・平田 成くんから教えてもらった内部の密度が一様な多面体による重力の計算に関する論文が目にとまりました：

http://adsabs.harvard.edu/full/1997CeMDA..65..313W

肝心のスペクトル解析用のアルゴリズム開発に行き詰ったので、気分転換にこの論文を読み、その中に記されていた数式に基づいて STL データ（ファイル）として与えられた、内部の密度が一様だと仮定した物体像による重力ポテンシャルと重力加速度（ベクトル）の値を計算するプログラム（群）を書いてみました：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip

この書庫ファイルの中には STL データとして与えられた物体像の表面の正規化した重力ポテンシャル（normalized potential、NP）や鉛直線偏差（deviation of the vertical、DV）を計算し、それらの値に応じて色付けした STL データを作成するプログラムも入っています。

NP = 物体像の重心における値で正規化した重力ポテンシャル（無次元）
DV = 物体表面の内向き法線方向と重力加速度のなす角度（0～180 度）

そして、これらのプログラムを会津大学や JAXA（NASA？）から発表されているイトカワの形状モデルに適用してみました：

http://darts.isas.jaxa.jp/planet/project/hayabusa/shape.pl

その結果の gzip 圧縮した STL ファイルと、それらから描いた鳥瞰図の動画のファイルを SPring-8 の FTP サイトにアップロードしておきました：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/

ただし、今回は以下の 5 種類のイトカワの形状モデルを処理しました：

aizu5.04：会津大学が作成した形状モデル。STL の facet の数は 4285。
f0049152：JAXA（NASA？）が作成した facet 数が 49152 のもの
f0196608：同上。facet 数は 196608
f0786432：同上。786432
f3145728：同上。3145728

前記の FTP サイトにはこれらの形状モデルごとに 6 個のファイルが置いてあります。例えば形状モデル f3145728 のものは以下の通りです：

以前にも紹介したオリジナルの形状モデルの STL ファイル
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/itokawa_f3145728.stl.gz

以前にも紹介したオリジナルの形状モデルの鳥瞰図の動画のファイル
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728.gif

NP や DV の値に応じて色付けした形状モデルの STL
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_np.stl.gz
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_dv.stl.gz

NP や DV の値に応じて色付けした形状モデルの鳥瞰図の動画
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_np.mpg
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_dv.mpg

また、以上のようなイトカワの 5 種類の形状モデルや、それらから計算した重力のデータに関する諸量をこの E-mail に添付した MS-Excel ファイルにまとめておきました。ただし、その 1 行目に記した記号の意味は以下の通りです：

model：形状モデルの名称
N：その形状モデルを構成する STL データの facet の総数
V：その形状モデルが表す物体像の体積
x0、y0 および z0：物体像の重心の座標値
P0：重心における重力ポテンシャルを「重力定数×物体密度」で割った値
NP1 と NP2：その形状モデルの計算で出現した NP の最小値と最大値
DV1 と DV2：その形状モデルの計算で出現した DV の最小値と最大値

今日のところはここまでにします。前記の書庫ファイル gravity.zip に入れたプログラムの詳細を今後の E-mail で説明したいと思っています。とり急ぎ、

添付ファイル “itokawa.xls”（inline table に変換）

model	N	V	x0	y0	z0
aizu5.04	4285	1.845559E-02	-2.252769E-02	1.333747E-03	2.199993E-03
f0049152	49152	1.772544E-02	3.978778E-05	-3.898139E-05	-2.066047E-05
f0196608	196608	1.773134E-02	5.535434E-05	-3.699083E-05	-2.746889E-05
f0786432	786432	1.773314E-02	6.171445E-05	-3.637770E-05	-2.973216E-05
f3145728	3145728	1.773363E-02	6.339775E-05	-3.630267E-05	-3.034454E-05

model	P0	NP1	NP2	DV1	DV2
aizu5.04	1.559526E-01	0.495144	0.821741	0.002871	57.344604
f0049152	1.496021E-01	0.501089	0.798465	0.047213	81.827213
f0196608	1.496274E-01	0.498322	0.798656	0.025490	90.484526
f0786432	1.496351E-01	0.497149	0.798738	0.004928	90.734569
f3145728	1.496370E-01	0.496147	0.798778	0.005168	90.676362
		0.495144	0.821741	0.002871	90.734569

Date: Mon, 08 Apr 2013 12:30:38 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_0

みなさま、

GSJ/AIST のなかのです。重要なことなので、ここに書いておきます。先週末の E-mail で紹介した「STL データとして与えられた、内部の密度が一様だと仮定した物体像による重力ポテンシャルと重力加速度（ベクトル）を計算するプログラム（群）」についてですが、これらが処理する重力ポテンシャルの値は通常使われている値と符号が逆です。例えば、単位の質点から距離 r だけ離れた点の重力ポテンシャル P は以下の通りです：

ぼくが使っている定義式：$P = G / r$
通常使われている定義式：$P = -G / r$

ただし、G は重力定数

このため、通常使われている定義では質量が集中している場所は「重力井戸」になりますが、ぼくのプログラムで計算した重力場はその逆になります。それから、ぼくは P からの重力加速度ベクトル g の計算式でも符号を逆にしています：

ぼくが使っている計算式：$g = \nabla P$
通常使われている計算式：$g = -\nabla P$

ただし、$\nabla = (\partial / \partial x,\ \partial / \partial y,\ \partial / \partial z)$

このため、ぼくのプログラムで計算した g は通常と同じ符号の値になります。

なお、

[1] このような通常とは異なる定義式や計算式は、ぼくが参考にした以下の論文で使われていました（ぼくはこの論文を鵜呑みしてしまった）：

http://adsabs.harvard.edu/full/1997CeMDA..65..313W

[2] 先日の E-mail で紹介した、ぼくのプログラムで計算した「正規化した重力ポテンシャル（NP）」の値は通常の式で計算した値と同じです：

NP = 物体像の重心における値で正規化した重力ポテンシャル（無次元）

とり急ぎ、

Date: Tue, 09 Apr 2013 19:03:09 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_1

みなさま、

GSJ/AIST のなかのです。先日の E-mails に書いた「STL データ（ファイル）として与えられた、内部の密度が一様だと仮定した物体像による重力ポテンシャルと重力加速度ベクトルを計算するプログラム群」に関する以下の話をします：

(1) プログラムの概要
(2) 書庫ファイル gravity.zip の構成
(3) プログラムのインストール
(4) 関数 SurfaceIntegral() について
(5) プログラムの使用法
(6) イトカワの形状モデルに対する NP と DV の計算（デモ）

(1) プログラムの概要

UNIX（Linux や MacOS など）と Windows の端末環境（コマンドプロンプト）で実行可能な以下の 3 種類・8 個の計算機プログラムを書きました：

単純な形状の物体像に対するテスト用のプログラム

cube

内部密度が 1 の立方体による任意の観測点の重力の値を計算する。

globe_[t,o,i]h（3 個のプログラム）

内部密度が 1 の「単位球を模した物体像（GLOBE)」による任意の観測点の重力を計算。GLOBE の作成の仕方が異なる 3 個のプログラムがある。

STL データが表す物体像による重力の値を計算するプログラム

stl_pxyz

任意の観測点もしくは STL データを構成する facet それぞれの重心における重力ポテンシャルと重力加速度を計算し、それらの値をテキストデータとして出力する。

stl_np と stl_dv

これらはそれぞれ STL データを構成する各 facet の重心における正規化した重力ポテンシャル（NP）と鉛直線偏差（DV）を計算し、それらの値に応じて facet に色付けした STL データを作成する。

STL データファイルの処理用のユーティリティプログラム

stl_rgb

STL ファイルの facet それぞれの色データを操作するプログラム。

(2) 書庫ファイル gravity.zip の構成

書庫ファイル http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip には以下の 6 個のディレクトリが格納されています：

src/：プログラムの C 言語ソースファイルなどが入っているディレクトリ
bin/：最初は空。UNIX 用の実行ファイルをインストールするディレクトリ。
exe/：Windows 用の実行ファイルが入っているディレクトリ
doc/：今回の重力の計算で参考にした 2 個の論文の PDF が入っている。
etc/：STL の facet の色付け用カラーマップデータファイルが入っている。
itokawa/

以下で紹介するプログラム群の実行のデモンストレーションに使用する STL ファイル、バッチファイルと C-shell scripts が入っている。

(3) プログラムのインストール

まず、書庫ファイル gravity.zip を適当な場所に解凍・展開して下さい：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip

Windows で gravity/itokawa/ の下のバッチファイル np_stl. bat や dv_stl.bat を実行するだけなら、書庫ファイルを解凍・展開後に何もする必要はありません。gravity/exe/ の下にある実行ファイルを任意の場所から起動したいなら、そのディレクトリ名を実行パスに登録するか、その下にある実行ファイル *.exe を実行パスに登録済みのディレクトリにコピーして下さい。

UNIX の場合、ディレクトリ gravity/src/ の下にある C 言語ソースコードから実行ファイルをコンパイルする必要があります。

```
cd   何とかかんとか/gravity/src
make                ← GNU C Compiler（gcc）でコンパイルするように設定してある。
make  install       ← コンパイルに成功したら実行ファイルを gravity/bin/ にコピー
make  clean         ← 不要になったファイルを消去
```

その後、ディレクトリ gravity/bin/ を実行パスに登録して下さい。

ディレクトリ gravity/itokawa/の下にあるファイル np_bev.*や dv_bev.*は STL データとして与えられた物体像の鳥瞰図をソフトウェアパッケージ「hvd」に含まれるプログラム stl_bev_4 を使って描画します：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/hvd_120301.pdf

← この PDF の 11 ページ目以降に hvd の簡単な説明があります。

hvd のインストール法は先の gravity.zip のものと概ね同じです。そのためにまず、hvd 用の書庫ファイル hvd.zip を解凍・展開します：

http://www-bl20.spring8.or.jp/~sp8ct/tmp/hvd.zip

その後、文字列"gravity"を"hvd"に読み替えて、先に説明した手法で hvd のプログラム群をインストールして下さい。なお、書庫ファイル hvd.zip から展開したディレクトリ hvd/ を gravity/ と並んだ位置に配置すれば、前記の gravity/itokawa/*_bev.* の実行前に実行パスを設定する必要はありません。

(4) 関数 SurfaceIntegral() について

単純な形状の立方体や GLOBE に対するものも含めて、ここで紹介している STL データとして与えられた物体像による重力の値を計算するプログラムはすべて、gravity/src/si.c に記した C 言語の関数 SurfaceIntegral () を使っています。この関数は STL の facet それぞれに対して論文

gravity/doc/Werner_1996.pdf
http://adsabs.harvard.edu/full/1997CeMDA..65..313W

に載っていた（Gauss の発散定理と Stokes の公式を用いて体積積分から変換した）面積分の式をそのまま使っているため、処理速度に難があります。個々の facet の形状だけに依存する量（現在の si.c のコードで大文字で記してある auto 変数 D??、N? や O??? の値）を最初にまとめて計算してメモリに格納し、それらを面積分で使うようにすれば、現状の倍近い処理速度を達成できると思われます。

なお、これは現在の si.c のコードで対処済みの話ですが、観測点が facet の辺の上にある場合に Werner_1996.pdf の数式はそのまま使えません。その場合は例外として処理する必要があります。しかし、それには余分な処理時間が必要になるので、それを避けるために現在の si.c のコードでは facet の重心を観測点とするプログラム用に「観測点が facet の辺の上には来ないこと」を宣言するプリプロセッサ用定数 N_O_E（not on edge）を導入しています。ソースコードからのコンパイル時に定数 N_O_E が define されていると、上記の例外処理を含まない、少しだけ高速な実行コードが生成されます。

(5) プログラムの使用法

プログラム cube

機能

x、y、z 座標値の絶対値がいずれも 1 以下の領域を占めている密度 1 の 立方体による任意の位置の観測点の重力ポテンシャルの値と重力加速度のベクトルの 3 成分の値を計算するテスト用のプログラム。

起動法

cube（起動パラメータなし）

処理内容

空白、タブもしくは改行コードで区切られた観測点の x、y、z 座標値を標準入力（通常はキーボード）から読み込み、下記の 7 個の値をタブコード区切りで並べたテキスト行を標準出力（端末画面）に書き出す。

[1,2,3] 観測点の[x,y,z] 座標値

[4] 重力ポテンシャルの値 / 重力定数 = P(x,y,z)

[5,6,7] 重力加速度の [x,y,z] 成分の値 / 重力定数

なお、標準入力のデータが尽きる（キーボード入力の場合は「^D」もしくは「^Z（Windows）」のみを入力する）とプログラムは終了する。

備考

P(x,y,z) の解析解

P(0,0,0) = 24 Ln((1+√3)/√2) - 2 π ～ 9.52031 ...

P(1,1,1) = P(0,0,0) / 2 ～ 4.76015 ...

P(0,0,1) = 長いので式は省略 ～ 7.17124 ...

P(0,1,1) = 4 Ln(10) - ArcTan(44/117) − π ～ 5.70904

これらの解析解は論文

gravity/doc/Seidov_2000.pdf

http://arxiv.org/abs/astro-ph/0002496

に載っています。

プログラム globe_th、globe_oh および globe_ih

機能

内部密度が 1 の座標原点を中心とした「単位球を模した物体像（GLOBE）」による任意の位置の観測点の重力ポテンシャル値と重力加速度ベクトルの 3 成分の値を計算するテスト用のプログラム。

起動法

globe_[t,o,i]h {level} {lon lat rot}

ただし、

level（0 以上の整数）

初期多面体の表面の三角形の細分化レベル。指定省略時は 0 が用いられる。

lon、lat および rot（3 個とも度単位の角度）

すべての処理前に原点を中心とした 3 次元回転を初期多面体に施す場合の 3 個の回転角。指定省略時は多面体を回転しない。

処理内容

globe_[t,o,i]h はそれぞれ、単位球に内接している下記の正多面体を初期多面体とし、添付した PDF ファイル[t,o,i]h.pdf のようにしてそれらの表面の正三角形を細分化して構築した GLOBE を使用する。

th：正 4 面体（tetrahedron）

oh：正８面体（octahedron）

ih：正 20 面体（icosahedron）

３個の回転角が指定された場合はまず初期多面体の３次元回転を行う。次に、指定された level の値に応じた GLOBE を構築してその体積と重心の座標値を計算し、それらの４個の値をタブコードで区切った１行にまとめて標準出力に書き出す：

[1] GLOBE の体積 = V

[2,3,4] GLOBE の重心の[x,y,z]座標値

その後、空白、タブもしくは改行コードで区切られた観測点の x、y、z 座標値を標準入力から読み込み、下記の７個の値をタブコード区切りで並べたテキスト行を標準出力に書き出す。

[1,2,3] 観測点の[x,y,z]座標値

[4] 重力ポテンシャルの値 / 重力定数 = P(x,y,z)

[5,6,7] 重力加速度の[x,y,z]成分の値 / 重力定数

備考

単位球の体積

V = (4/3) π ～ 4.18879 ...

単位球の P(x,y,z) の解析解

r = √(x^2+y^2+z^2) として、

P(x,y,z) = V × (3 - r^2) / 2　　r ≦ 1

= V / r　　r ≧ 1

P(0,0,0) = V × 3/2 = 2 π ～ 6.283185 ...

P(0,0,1) = V = (4/3) π ～ 4.18879 ...

P(0,0,2) = V / 2 = (2/3) π ～ 2.09435 ...

プログラム stl_pxyz

機能

STL データ（ファイル）として与えられた内部密度が１の物体像による任意の位置の観測点における、もしくは、その STL データを構成している facet（三角形）それぞれの重心における重力ポテンシャルの値と重力加速度の３成分の値を計算し、テキストデータとして出力する。

起動法

stl_pxyz　STL　{point}

ただし、

STL

STL ファイル（ASCII と binary の両形式を処理可能）の名前。ただし、"-" を指定すると標準入力からデータを読む。

point

各行に空白もしくはタブコード区切りで重力の値を計算する観測点の x、y、z 座標値の３個の値（４個以上の値があっても良いが、それらは読み飛ばされる）を列記したテキストファイルの名前。ただし、"-" を指定すると標準入力からデータを読む。この

ファイル名の指定を省略すると与えられた STL データを構成している facet それぞれの重心を観測点とする。

処理内容

ファイルから STL データを読み込んだ後、まず、物体像の体積と重心の座標値を計算し、それら 4 個の値をタブコードで区切った 1 行にまとめて標準出力に書き出す：

[1] STL データが表す物体像の体積

[2,3,4] その物体の重心の [x,y,z] 座標値

その後、起動時の指定に応じてファイルから読み取った座標値の観測点における、もしくは、facet それぞれの重心における重力ポテンシャルと重力加速度の成分値を計算し、それらを含む以下の 7 もしくは 13 個の値をタブコードで区切って並べた行を標準出力に書き出す：

観測点のファイル（point）が指定された場合

[1,2,3] 観測点の[x,y,z]座標値

[4] 観測点における重力ポテンシャルの値 / 重力定数

[5,6,7] 重力加速度の[x,y,z]成分の値 / 重力定数

観測点のファイルが指定されなかった場合

[1,2,3] STL の facet を構成する頂点 1 の[x,y,z]座標値

[4,5,6] STL の facet を構成する頂点 2 の[x,y,z]座標値

[7,8,9] STL の facet を構成する頂点 3 の[x,y,z]座標値

[10] facet の重心における重力ポテンシャルの値 / 重力定数

[11,12,13] 重力加速度の[x,y,z]成分の値 / 重力定数

備考 1

facet の頂点の座標値やその重心における重力の値が列記された行は STL ファイルに格納されている facet と同じ順で出力される。これらの各行に記されている値からその facet に付ける色の R、G、B 成分の値を計算すれば、次に説明するプログラム stl_rgb を使ってそれらの facet ごとの色のデータを元の STL ファイルに埋め込むことができる。

備考 2

facet 数が非常に多い場合、後述するプログラム stl_np や stl_dv でその STL データを構成するすべての facet の重力の値をまとめて計算すると膨大な処理時間が必要になる。複数の CPUs を利用できるなら、以下の安易な手法でそれをかなり軽減できる：

[a] 書庫ファイル hvd.zip に入っているプログラム stl_dump を使って STL ファイル上の facet それぞれの 3 頂点の座標値をテキストデータとしてダンプする。その後、facet それぞれの重心の座標値を計算し、それらの値を複数のテキストファイルに分割して書き込む。

[b] これらの複数のファイルそれぞれを起動パラメータ point として指定した stl_pxyz(の複数のプロセス）を複数の CPUs を使って並列に実行する。

[c] こうして計算した facet それぞれの重心における重力の値を入れたファイルすべてを連結し、それと同時にそれぞれの値に応じた色データを計算する。その後、プログラム stl_rgb を使ってこれらの色データをもとの STL ファイルに埋め込む。

プログラム stl_rgb

機能

STL ファイルのそれぞれの facet の色データを操作するプログラム。具体的には起動パラメータの指定に応じて以下の４種類の動作を行う：

[1] それぞれの facet の色データを抽出する。

[2] それぞれの facet の色データを削除する。

[3] それぞれの facet に個別の色データを埋め込む。

[4] すべての facet に単一の色データを埋め込む。

起動法

[1] stl_rgb　orgSTL　> newRGB

[2] stl_rgb　orgSTL　newSTL

[3] stl_rgb　orgSTL　orgRGB　newSTL

[4] stl_rgb　orgSTL　R　G　B　newSTL

ただし、

orgSTL：もとの STL データファイルの名前

newRGB

orgSTL の facet それぞれに付いている色の R、G、B 成分の 値（0～ 255 の整数値）をタブコード区切りの１行にまとめて書き込むテキストファイルの名前。ただし、それらの行数は orgSTL の facet 数と同じで、色データが付いていない facet に対応した行は空行になる。

newSTL：新しい STL データファイルの名前

orgRGB

orgSTL のそれぞれの facet に埋め込む色の R、G、B 成分の　値（0～255 の値；浮動小数点数でも良い）を各行に空白もしくはタブコード区切りで書き込んだテキストファイルの名前。

R、G および B

orgSTL の facet すべてに埋め込む単一の色の R、G、B 成分の値（0～255 の値；浮動小数点数でも良い)。

プログラム stl_np と stl_dv

機能

STL データファイルが表す内部の密度が１の物体像によるそれぞれの facet の重心における重力ポテンシャルや重力加速度から stl_np では NP の値を、stl_dv は DV の値を計算する：

NP = 物体像の重心における値で正規化した重力ポテンシャル（無次元）

DV = 物体表面の内向き法線方向と重力加速度のなす角度（0～180 度）

その後、これらの値を 0～32767 の色番号に変換する。指定されたカラーマップに従ってその番号を facet の色データに変換し、それを新しい STL データファイルに埋め込む。

起動法

stl_np orgSTL {{min max} CM_STL newSTL}

stl_dv orgSTL {{min max} CM_STL newSTL}

ただし、

orgSTL：もとの STL データファイルの名前

min と max

計算で得た NP や DV の値を色番号に変換する際の値の範囲。min 以下の値は色番号０に、max 以上は 32767 に変換される。

CM_STL

STL データファイル上の各 facet が保持可能な０～32767 の色番号に応じた色のデータを記したカラーマップデータのテキストファイルの名前。空白もしくはタブコードで区切った以下の２もしくは４個の値を各行に書き込む：

[1] 以下のような色番号の範囲を示す range list

0：番号 0

0,2：番号０と２

0－2：番号０～２

－2,32766－：番号２以下と 32766 以上

[2,3,4]

色の R、G、B 成分値（０～255 の値；浮動小数点数でもかまわない)。１個の値だけを指定すると、R、G、B 成分値すべてがその値だと見なす。

newSTL：新しい STL データファイルの名前

処理内容

指定した起動パラメータの個数によらず stl_np や stl_dv は orgSTL の facet すべての NP や DV の値を計算する。その後、orgSTL だけの指定で起動した場合はタブコード区切りで２もしくは３個の値が並んだ以下の１行を標準出力に書き出してプログラムが終了する。

stl_np の出力行には３個の値が並ぶ：

[1] 物体像の重心における重力ポテンシャルの値 / 重力定数

[2,3] 物体像の facet すべての NP の最小値と最大値

stl_dv では２個の値のみ：

[1,2] 物体像の facet すべての DV の最小値と最大値

起動パラメータ min と max が指定された場合はそれらを下限と上限値として、また、指定されなかった場合は上記の最小値と最大値を使って、facet それぞれの NP や DV の値を正規化した０～ 32767 の範囲の色番号を決める。そして、その番号に応じた CM_STL で与えられた facet の色を埋め込んだ新しい STL データファイル newSTL を作成する。

(6) イトカワの形状モデルに対する NP と DV の計算（デモ）

ディレクトリ gravity/itokawa/ には以下の３種類のファイルが入っています：

aizu5.04.stl

会津大学が作ったイトカワの形状モデルの STL ファイル

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/itokawa_aizu5.04.stl.gz

np_stl.bat、np_stl.csh、dv_stl.bat および dv_stl.csh

形状モデルの表面に NP や DV に応じた色付けした以下の２個の新しい STL ファイルを作るためのバッチファイルや C-shell scripts：

aizu5.04_np.stl

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/aizu5.04_np.stl.gz

aizu5.04_dv.stl

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/aizu5.04_dv.stl.gz

np_bev.bat、np_bev.csh、dv_bev.bat および dv_bev.csh

hvd のプログラム stl_bev_4 を使って以下の２個の物体像のカラーの鳥瞰図の動画を描くためのバッチファイルと C-shell scripts：

aizu5.04_np.gif

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/aizu5.04_np.mpg

aizu5.04_dv.gif

http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/aizu5.04_dv.mpg

注

プログラム stl_bev_4 で作った animation GIFs に対して余白のトリミングやカラーバーの付加などのちょっとした加工を加えた結果が上記の MPEGs です。

これらのバッチファイルや C-shell scripts の起動法は以下の通りです：

Windows

ファイルブラウザで *_stl.bat、*_bev.bat の順にファイルをクリック。

UNIX

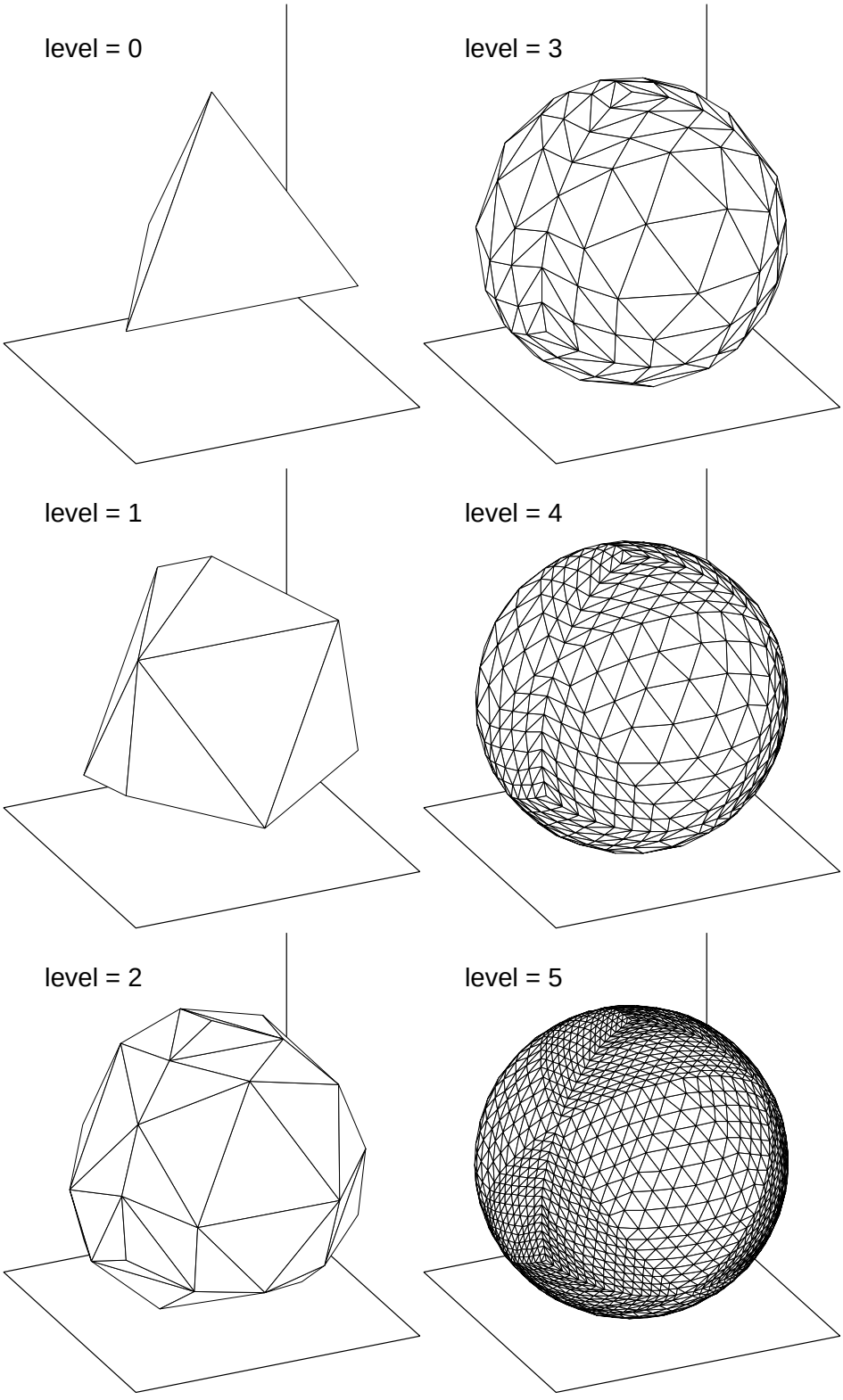
```
cd　何とかかんとか/gravity/itokawa
csh　np_stl.csh　&&　csh　np_bev.csh
csh　dv_stl.csh　&&　csh　dv_bev.csh
```

こちらの Windows7 PC（ThinkPad T400s、Core2 Duo P9400、2.40GHz）を使うと上記の４個のバッチファイルの実行時間の合計は１分以下でした。

長い E-mail になってすみません。とりあえず以上です。

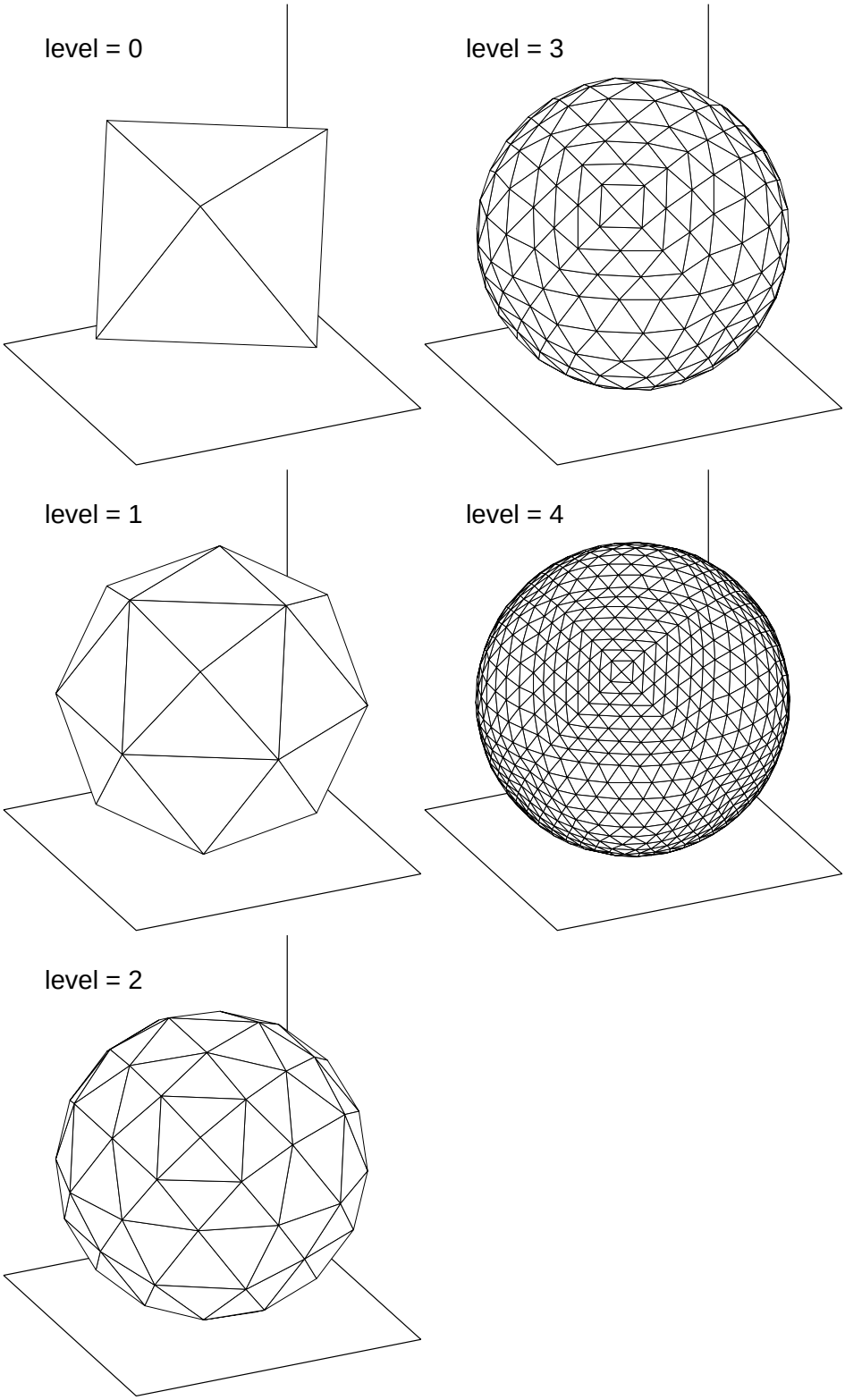
添付ファイル "th.pdf"

globe_th



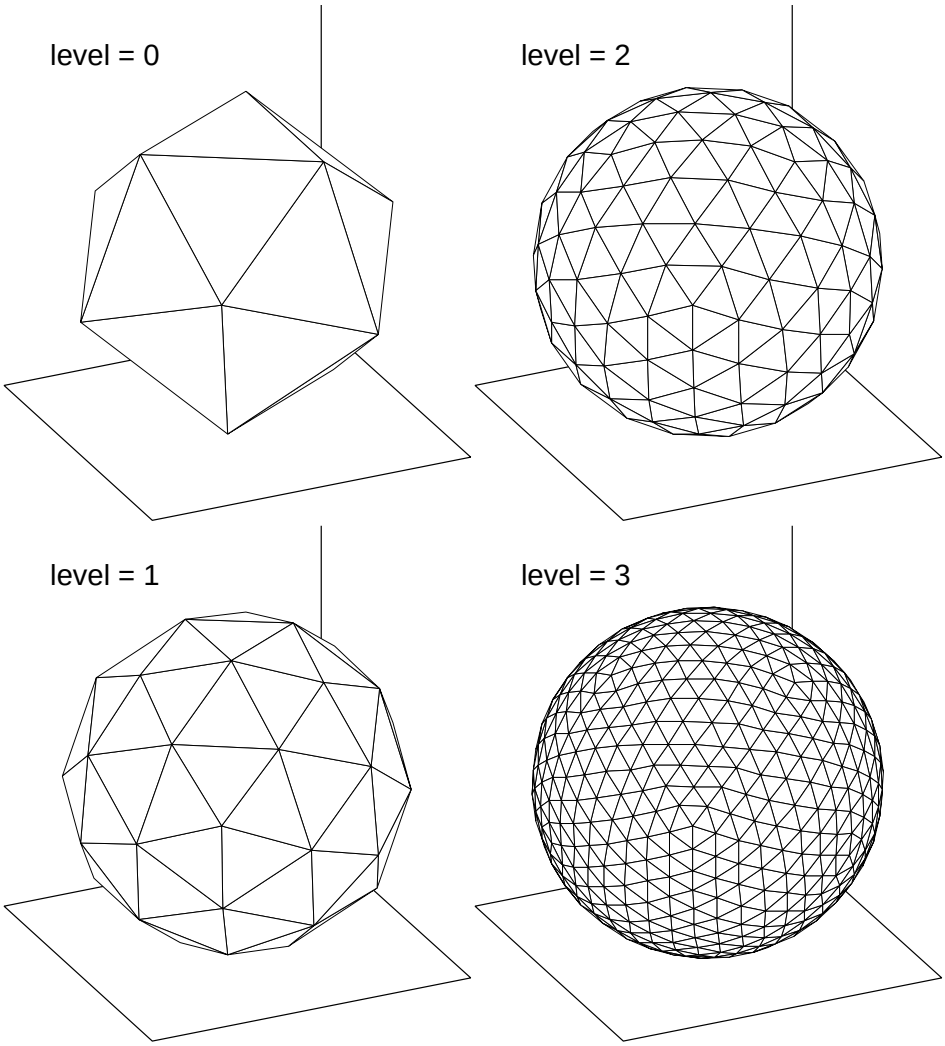
添付ファイル "oh.pdf"

globe_oh



添付ファイル "ih.pdf"

globe_ih



Date: Tue, 09 Apr 2013 20:10:14 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_2

みなさま、

なかのです。「STL データ（ファイル）として与えられた、内部の密度が一様だと仮定した物体像による重力ポテンシャルと重力加速度ベクトルを計算するプログラム群」に関して書き忘れたことがありました。プログラム stl_np で得た NP の値と stl_dv で得た DV の値などについてです：

重心の値で正規化した重力ポテンシャル（normalized potential）
NP= 物体像の重心における値で正規化した重力ポテンシャル（無次元）

地球のような大型天体では多少の凸凹があるとは言え、その表面は概ね等ポテンシャル面に一致します。しかし、イトカワのような小天体ではそうはなりません。地球では（通常の定義の）重力ポテンシャルは地形の標高と比例するので、NP を「重力的な意味の標高」と解釈するのが自然でしょうね。ただし、NP は質量の集中している点で正の値（物体の重心で値１）、無限遠で０なので、NP の値が小さい点ほど重力的な意味の標高が高いことになります。

鉛直線偏差（deviation of the vertical）
DV = 物体表面の内向き法線方向と重力加速度のなす角度（０～180 度）

実は、地球の DV で使う法線方向は地球の形を良く近似する回転楕円体のもので、近隣の点で大きく変わりません。このような「準拠楕円体」のようなモノをイトカワに設定できないので、ぼくは STL の各 facet の重心の DV の計算に（近隣の点で大きく異なることもある）facet の法線方向を使いました。このようにして計算した DV は「重力的に見た表面の勾配」ですね。DV が 45 度以上の場所にモノを置くと、それはそこでの表面に沿って勝手に滑ります。ハヤブサのタッチダウン地点の選定では（後述するイトカワの自転に伴う遠心力を繰り込んだ）DV の空間分布の検討が重要だったのでしょうね。

上にも書きましたが、イトカワの表面の DV の議論ではイトカワの自転（もしかしたら公転も？）に伴う遠心力を考慮しないといけません。

遠心力による加速度 = 回転軸からの距離 × 自転の角速度^2

回転軸の位置がわかればこの値は容易に計算できますね。プログラム stl_pxyz を使えば観測点の座標値や重力加速度を重力定数で割った値がテキストデータとして得られるので、遠心力を考慮した DV の値の計算も容易です。とり急ぎ、

Date: Thu, 13 Jun 2013 13:35:38 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_3

みなさま、

GSJ/AIST のなかのです。以前に紹介した「STL データとして与えられた物体像に対してその内部の密度が一様と仮定した場合の重力ポテンシャルや重力加速度ベクトルの値を計算するプログラム」のうちの主要なものに対して以下の４種類の高速化を行いました。

(1) 「段取り」の見直しによる NP と DV の計算の高速化（省力化？）
(2) 面積分の計算アルゴリズムの改良による高速化
(3) マルチスレッド処理による高速化
(4) CUDA GPU 用のプログラム

注

STL（STereo Lithography）データ
３次元物体像の表面を多数の三角形で表現した形状データ
http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.pdf

NP（Normalized Potential）
物体像の重心における値で正規化した重力ポテンシャル（無次元）
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_np.mpg

DV（Deviation of the Vertical、鉛直線偏差）
物体表面の内向き法線方向と重力加速度のなす角度（０～180 度）
http://www-bl20.spring8.or.jp/~sp8ct/tmp/itokawa/f3145728_dv.mpg

高速化したプログラムのソースファイルや Windows 用実行ファイルなどは以前と同じ書庫ファイル http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip の中に入れてあります。また、CUDA GPU 用のものを除いて､新しいプログラムのソースファイルなどは以前のものと同じディレクトリ gravity/src/ や exe/ の下に格納されています。そして、新しいプログラムのインストール法も以前のものとまったく同じです。

ただし、Windows 用実行ファイルをコンパイルし直す場合はコンパイル用のスクリプト gravity/src/Makefile の設定を変える必要があります。MinGW（Minimalist GNU for Windows）の環境でコンパイルするためのスクリプト gravity/src/Makefile.MinGW を参考にして下さい。

なお、新たに設けたディレクトリ cuda/ の下にある CUDA GPU 用のプログラムのソースファイルやそれらのインストール法については後で改めて説明します。

以下に高速化したプログラムの詳細などについて書きます。

(0) 以前に紹介したプログラムの概要（復習）

これについては以前の E-mail に書いた説明の引用で済ませます。ただし、下記のテスト用プログラム cube、globe_* とユーティリティプログラム stl_rgb は今回の高速化の対象外です。

On Tue, 09 Apr 2013 19:03:09 +0900 Tsukasa NAKANO wrote:

> 単純な形状の物体像に対するテスト用のプログラム
>　　cube
>　　内部密度が 1 の立方体による任意の観測点の重力の値を計算する。
>　　globe_[t,o,i]h（3 個のプログラム）
>　　内部密度が 1 の「単位球を模した物体像（GLOBE)」による任意の観測点の重力を計算。
>　　GLOBE の作成の仕方が異なる 3 個のプログラムがある。
>
> STL データが表す物体像による重力の値を計算するプログラム
>　　stl_pxyz
>　　任意の観測点もしくは STL データを構成する facet それぞれの重心における重力
>　　ポテンシャルと重力加速度を計算し、それらの値をテキストデータとして出力する。
>　　stl_np と stl_dv
>　　これらはそれぞれ STL データを構成する各 facet の重心における正規化した重力
>　　ポテンシャル（NP）と鉛直線偏差（DV）を計算し、それらの値に応じて facet に
>　　色付けした STL データを作成する。
>
> STL データファイルの処理用のユーティリティプログラム
>　　stl_rgb
>　　STL ファイルの facet それぞれの色データを操作するプログラム。

(1) 「段取り」の見直しによる NP と DV の計算の高速化（省力化？）

プログラム stl_np と stl_dv はそれぞれ、同一の関数 SurfaceIntegral() を用いて物体像表面の重力ポテンシャル P や重力加速度ベクトル G の値を計算し､その後､それらを NP や DV の値に変換しています。ただし、その際に、stl_np は G の、また、stl_dv は P の計算結果を捨てているので、NP と DV の両方のデータを得るために stl_np と stl_dv の両方を個別に実行するのはムダです。プログラム stl_pxyz を使えば P と G の両方のデータを計算・出力できるので、それを一度だけ実行し、その後、その出力から NP や DV を算出するのが経済的です。このようなムダのない計算の「段取り」については以前にも紹介しました。

On Tue, 09 Apr 2013 19:03:09 +0900 Tsukasa NAKANO wrote:

> プログラム stl_pxyz

> 機能
> STL データ（ファイル）として与えられた内部密度が 1 の物体像による任意の位置の
> 観測点における、もしくは、その STL データを構成している facet（三角形）それぞれ
> の重心における重力ポテンシャルの値と重力加速度の 3 成分の値を計算し、テキスト
> データとして出力する。
> 起動法
> stl_pxyz　STL　{point}
> ...
> 備考 2
> facet 数が非常に多い場合、後述するプログラム stl_np や stl_dv でその STL データを
> 構成するすべての facet の重力の値をまとめて計算すると膨大な処理時間が必要になる。
> 複数の CPUs を利用できるなら、　以下の安易な手法でそれをかなり軽減できる：
> [a] ...
> [b] ...
> [c] ...

問題は上記の[a,b,c]の処理内容がやや複雑なことです。これを改善するため、新たに 2 個の（C 言語）プログラムと 6 個の awk スクリプトを書きました。

プログラム stl_pg

機能

stl_pxyz の機能のうち「任意の観測点における値の計算」を削除したプログラム。物体像表面の facet それぞれの重心における重力ポテンシャルや重力加速度ベクトルを計算し、テキストデータとして出力する。

起動法

stl_pg　STL

ただし、STL は入力する STL データファイルのパス名。

処理内容

ファイルから STL データを読み込んだ後、stl_pxyz と同様にその物体像の体積と重心の位置の 4 個の値を計算し、タブコード区切りの 1 行にまとめて標準出力に書き出す：

[1] STL データが表す物体像の体積

[2,3,4] その物体の重心の[x,y,z]座標値

次に、stl_pxyz とは異なり、物体像の重心における重力ポテンシャルと重力加速度ベクトルの値を計算し、1 行にまとめて出力する：

[1] 物体像の重心における重力ポテンシャルの値 / 重力定数

[2,3,4] そこでの重力加速度の[x,y,z]成分の値 / 重力定数

その後、stl_pxyz を観測点のファイルの指定をせずに起動した場合と同様に、物体像の表面の facet それぞれの重心の重力ポテンシャルと重力加速度の値を計算し、それらを facet の 3 頂点の座標値とともに 1 行にまとめて出力する（facet ごとに合計 13 個の値を出力する）：

[1,2,3] STL の facet を構成する頂点１の[x,y,z]座標値
[4,5,6] STL の facet を構成する頂点２の[x,y,z]座標値
[7,8,9] STL の facet を構成する頂点３の[x,y,z]座標値
[10] facet の重心における重力ポテンシャルの値 / 重力定数
[11,12,13] そこでの重力加速度の[x,y,z]成分の値 / 重力定数

t2stl

機能

テキスト形式の STL データをバイナリ形式（正確には binary or color STL 形式）のデータファイルに変換するユーティリティプログラム。

起動法

t2stl {CM_STL} STL

ただし、

CM_STL

stl_np や stl_dv の起動パラメータとして指定できるものと同じ STL データ用のカラーマップのデータファイルのパス名。このカラーマップは後述する facet 記述行の「フラグ値」を facet の色に変換する際に用いられる。CM_STL の指定を省略すると facet 記述行の上のフラグ値が STL データファイルにそのまま転記される。

STL

新しい STL データファイルのパス名

処理内容

物体像表面を覆い尽くす facet それぞれのデータを記述している複数の行を標準入力から読み込み、それらを変換して STL データファイルに書き込む。それぞれの facet の記述行には空白もしくはタブコード区切りで以下のいずれかの個数の値が並んでいる必要がある。

９個の値が並んでいる記述行

[1,2,3] facet の頂点１の[x,y,z]座標値
[4,5,6] facet の頂点２の[x,y,z]座標値
[7,8,9] facet の頂点３の[x,y,z]座標値

10 個

[1-9] facet の３頂点それぞれの[x,y,z]座標値
[10] その facet のフラグ値（０～ 32767 の整数値）

12 個

[1-9] facet の３頂点それぞれの[x,y,z]座標値
[10,11,12] その facet の色の[R,G,B]成分値（０～255）

備考１

t2stl は入力した STL データの「整合性」をチェックしない。つまり、入力した facet が物体像を覆い尽くしているか否かはもちろんのこと、入力した facet の総数（= facet 記述行の行数）の確認すら行わない。

備考２

color STL 形式データファイルのそれぞれの facet には 32768 種類の色情報を埋め込むことができるが、t2stl ではそれをカラーマップ形式で指定することも、R、G、B 成分値そのもので指定することもできる。facet それぞれの記述行に「フラグ値」が指定された場合、それは起動パラメータ CM_STL のカラーマップのインデックスと見なされる。

awk スクリプト pg2np_[0,1,2].awk と pg2dv_[0,1,2].awk

機能と処理内容

stl_pg が出力したデータから NP や DV の値を計算するスクリプト。NP の計算用の３個のスクリプト pg2np_[0,1,2].awk は stl_np の、また、DV の計算用の pg2dv_[0,1,2].awk は stl_dv の３通りの起動法のそれぞれに対応している（後述）。

起動法

後述する新しい段取りの実行例をご覧下さい。

これらを使って stl_np や stl_dv と同じ内容の処理を実行する段取りは以下の通りです。まず、stl_pg を起動してSTLデータファイルorg.stlが表現している物体像の表面の重力ポテンシャルや重力加速度ベクトルの値すべてを計算し、その結果を作業用のテキストファイル pg.txt に書き込んでおきます。

```
stl_pg　org.stl > pg.txt
```

その後、stl_np や stl_dv それぞれの３種類の起動法に応じた処理を行います。

NP の計算

物体像表面の NP の最小値と最大値を調べて標準出力に書き出す。

従来の入力：stl_np　org.stl

新しい手法：awk　-f　pg2np_0.awk　pg.txt

出現した最小値と最大値で正規化した NP の値に応じてカラーマップ CM_STL で色づけした物体像を STL データファイル np.stl に書き込む。

```
stl_np　org.stl　CM_STL　np.stl
awk　-f　pg2np_1.awk　pg.txt　|　t2stl　CM_STL　np.stl
```

指定した最小値 MIN と最大値 MAX で正規化した NP の値に応じて CM_STL で色づけした像を np.stl に書き込む。

```
stl_np　org.stl　MIN　MAX　CM_STL　np.stl
awk　-v　min=MIN　（改行しない）
　　-v　max=MAX　（改行しない）
　　-f　pg2np_2.awk　pg.txt　|　t2stl　CM_STL　np.stl
```

DV の計算

NP → DV かつ np → dv と読み替えれば上記の NP の場合と同様です。

なお、awk スクリプト用のインタプリタの実行ファイルとして"awk"ではなく"gawk"（GNU awk）を指定する必要があるかもしれません。また、それを使って実行する awk スクリプトのファイル名は絶対パスで指定する必要があります。

(2) 面積分の計算アルゴリズムの改良による高速化

以前の E-mail に書いた以下の部分を改良しました。

On Tue, 09 Apr 2013 19:03:09 +0900 Tsukasa NAKANO wrote:

> (4) 関数 SurfaceIntegral() について
>
> 単純な形状の立方体や GLOBE に対するものも含めて、ここで紹介している STL データ
> として与えられた物体像による重力の値を計算するプログラムはすべて、gravity/src/si.c
> に記した C 言語の関数 SurfaceIntegral () を使っています。この関数は STL の facet それ
> ぞれに対して論文
> gravity/doc/Werner_1996.pdf
> http://adsabs.harvard.edu/full/1997CeMDA..65..313W
> に載っていた（Gauss の発散定理と Stokes の公式を用いて体積積分から変換した）面積分
> の式をそのまま使っているため、処理速度に難があります。個々の facet の形状だけに依
> 存する量（現在の si.c のコードで大文字で記してある auto 変数 D??、N? や O??? の値）を
> 最初にまとめて計算してメモリに格納し、それらを面積分で使うようにすれば、現状の倍
> 近い処理速度を達成できると思われます。

改良したコードはインクルードファイル gravity/src/si.h に書き込みました。重力の計算の面積分を行う新しい関数の名前は CalculateSI()で、その実行前後に初期化用の関数 BeginSI()や後始末用の関数（実態はマクロ）EndSI() を呼び出す仕様にしました。また、コンパイル時に定数（マクロ）SI_P と SI_G を宣言することにより、重力ポテンシャルと重力加速度ベクトルのそれぞれの値だけを計算する実行コードを生成できます（これにより、NP や DV だけを計算する場合に若干の高速化が望めます)。

このコードを組み込んだ、従来の stl_pxyz、stl_pg、stl_np および stl_dv と完全互換な機能を持つプログラム群を作成しました。

```
stl_pxyz_s  orgSTL  {point}
stl_pg_s    STL
stl_np_s  orgSTL  {{min  max}  CM_STL  newSTL}
stl_dv_s  orgSTL  {{min  max}  CM_STL  newSTL}
```

なお、これらのプログラム名の語尾の"_s"は single を意味しています（次に紹介するマルチスレッド版のプログラムの名前を"*_m"としたため)。

(3) マルチスレッド処理による高速化

前記の si.h のコードを利用したマルチスレッド版のプログラム群（ここまでに説明したものと完全互換な機能を持つプログラム群）も作成しました。

stl_pxyz_m　orgSTL　{point}
stl_pg_m　　STL
stl_np_m　orgSTL　{{min　max}　CM_STL　newSTL}
stl_dv_m　orgSTL　{{min　max}　CM_STL　newSTL}

これらはインクルードファイル graivty/src/mt.h に書き込んだコードを用いて物体像の重力の値を並列に計算します。ただし、このコードは UNIX 系 OS では pthread ライブラリを、また、Windows では WinThread を使用します。

マルチスレッド版のプログラムは重力値の計算に使用するスレッドの個数を環境変数 THREADS によって指定できるようになっています。つまり、プログラムの実行の直前に以下の端末入力で計算に使うスレッド数（自然数）を指定できます。

UNIX で C-shell 系 shell（csh や tcsh）を使っている場合
　setenv　THREADS　スレッド数
B-shell（sh や bash）
　THREADS=スレッド数 &&　export　THREADS
Windows のコマンドプロンプト
　set　THREADS=スレッド数

さて、ここまでに紹介した「STL データとして与えられた物体像に対してその内部の密度が一様と仮定した場合の重力ポテンシャルや重力加速度ベクトルの値を計算するプログラム群」のベンチマークテストの結果を紹介します。添付した PDF ファイル cpu.pdf をご覧下さい。そこには 4 台の計算機の上で STL データや計算に使用するスレッド数を変えて実行した stl_pxyz*、stl_pg*、stl_np*と stl_dv*の処理時間や「計算結果の同一性」を表す値などが示されています。ただし、cpu.pdf に記されている項目の意味は以下の通りです。

host
　ベンチマークテストを実行した Linux 計算機のホスト名
　　gsjgix：Dell Precision T7600、Xeon E5-2687W CPU（８ cores）
　　sp8ct：Dell Precision T7400、Xeon E5420 ×２（合計８ cores）
　　gsjvix：Dell Precision T7500、Xeon X5570 ×２（合計８ cores）
　　gsjkic：Dell Precision 390、Core2 Duo 6700（２ cores）
model
　STL データ（小惑星イトカワの形状モデル）の名前
　　aizu5.04：会津大学のモデル。facet の総数は 4285。
　　f0049152：JPL の Gaskell のモデル。facet の総数は 49152。

f0196608：同上。ただし、facet の総数は 196608 = 49152 ×４。

run

実行したプログラムの名前 stl_*。ただし、語尾に"_g"が付いているものは後で説明する CUDA GPU 用のプログラム。

threads

計算に使用したスレッド数。ただし、"-" は single thread。また、K20、GTX_680、C2070 と C1060 は CUDA GPU の機種。

sec.

プログラムの秒単位の実行時間（起動から終了までの経過時間）

CRC

出力データの Cyclic Redundancy Check コード（いわゆる check sum コード)。この数値が同じデータは完全に同じものであると見なせる。

bytes

出力データの総量（バイト数）

cpu.pdf に示されている結果から以下のことがわかりました。

[1] 新しいプログラム（例えば stl_pxyz_s）は以前のもの（stl_pxyz）の２倍（以上）の処理速度になっている。
[2] 新しいプログラムは以前のものと完全に同じ結果を返している。
[3] マルチスレッド処理ではスレッド数を２倍にすると処理速度も概ね２倍になっている。つまり、並列化の効率が非常に良い。
[4] マルチスレッド版のプログラムも以前のものと完全に同じ結果を返す。
[5] 多数の facet からなる STL データに対して CUDA GPU を使うと以前のものの 100 倍以上高速に処理を行える。ただし、その処理結果は以前のものと完全には一致しない。

長くなったので今回はここまで。予告した「(4) CUDA GPU 用のプログラム」に関する話は次便以降の E-mail に回します。とり急ぎ、

添付ファイル "cpu.pdf "（1 / 4）

host	model	run	threads	sec.	CRC	bytes
gsjgix	aizu5.04	pxyz	-	2.681972	3062843378	750924
gsjgix	aizu5.04	pxyz_s	-	0.993060	3062843378	750924
gsjgix	aizu5.04	pxyz_m	1	0.983596	3062843378	750924
gsjgix	aizu5.04	pxyz_m	2	0.537792	3062843378	750924
gsjgix	aizu5.04	pxyz_m	4	0.299867	3062843378	750924
gsjgix	aizu5.04	pxyz_m	8	0.175185	3062843378	750924
gsjgix	aizu5.04	pxyz_g	K20	0.677083	1614961764	750924
gsjgix	f0049152	pxyz	-	336.893807	512884516	8606574
gsjgix	f0049152	pxyz_s	-	124.210253	512884516	8606574
gsjgix	f0049152	pxyz_m	1	123.667104	512884516	8606574
gsjgix	f0049152	pxyz_m	2	65.461036	512884516	8606574
gsjgix	f0049152	pxyz_m	4	33.861472	512884516	8606574
gsjgix	f0049152	pxyz_m	8	17.614100	512884516	8606574
gsjgix	f0049152	pxyz_g	K20	2.205713	3014388987	8606574
gsjgix	f0196608	pxyz_g	K20	20.482203	395161540	34425798

host	model	run	threads	sec.	CRC	bytes
gsjgix	aizu5.04	pg	-	2.586738	837575810	750978
gsjgix	aizu5.04	pg_s	-	0.833708	837575810	750978
gsjgix	aizu5.04	pg_m	1	0.834786	837575810	750978
gsjgix	aizu5.04	pg_m	2	0.461044	837575810	750978
gsjgix	aizu5.04	pg_m	4	0.257810	837575810	750978
gsjgix	aizu5.04	pg_m	8	0.157248	837575810	750978
gsjgix	aizu5.04	pg_g	K20	0.644268	4060343442	750978
gsjgix	f0049152	pg	-	318.973911	590664062	8606628
gsjgix	f0049152	pg_s	-	104.414618	590664062	8606628
gsjgix	f0049152	pg_m	1	104.467494	590664062	8606628
gsjgix	f0049152	pg_m	2	55.297099	590664062	8606628
gsjgix	f0049152	pg_m	4	28.696931	590664062	8606628
gsjgix	f0049152	pg_m	8	14.962336	590664062	8606628
gsjgix	f0049152	pg_g	K20	2.336241	1008550782	8606628
gsjgix	f0196608	pg_g	K20	20.253148	986363691	34425853

host	model	run	threads	sec.	CRC	bytes
gsjgix	aizu5.04	np	-	2.578718	1899975242	214334
gsjgix	aizu5.04	np_s	-	0.806747	1899975242	214334
gsjgix	aizu5.04	np_m	1	0.806569	1899975242	214334
gsjgix	aizu5.04	np_m	2	0.439603	1899975242	214334
gsjgix	aizu5.04	np_m	4	0.238953	1899975242	214334
gsjgix	aizu5.04	np_m	8	0.141873	1899975242	214334
gsjgix	aizu5.04	np_g	K20	0.754567	2500797220	214334
gsjgix	f0049152	np	-	318.111434	4202476463	2457684
gsjgix	f0049152	np_s	-	102.229369	4202476463	2457684
gsjgix	f0049152	np_m	1	102.191257	4202476463	2457684
gsjgix	f0049152	np_m	2	54.092022	4202476463	2457684
gsjgix	f0049152	np_m	4	27.947149	4202476463	2457684
gsjgix	f0049152	np_m	8	14.498661	4202476463	2457684
gsjgix	f0049152	np_g	K20	2.144160	829119505	2457684
gsjgix	f0196608	np_g	K20	20.395434	3715266744	9830484

host	model	run	threads	sec.	CRC	bytes
gsjgix	aizu5.04	dv	-	2.597524	3202635282	214334
gsjgix	aizu5.04	dv_s	-	0.830142	3202635282	214334
gsjgix	aizu5.04	dv_m	1	0.824657	3202635282	214334
gsjgix	aizu5.04	dv_m	2	0.447442	3202635282	214334
gsjgix	aizu5.04	dv_m	4	0.244174	3202635282	214334
gsjgix	aizu5.04	dv_m	8	0.174619	3202635282	214334
gsjgix	aizu5.04	dv_g	K20	0.708261	3202635282	214334
gsjgix	f0049152	dv	-	318.100209	4259107353	2457684
gsjgix	f0049152	dv_s	-	104.606342	4259107353	2457684
gsjgix	f0049152	dv_m	1	104.480619	4259107353	2457684
gsjgix	f0049152	dv_m	2	55.264998	4259107353	2457684
gsjgix	f0049152	dv_m	4	28.517266	4259107353	2457684
gsjgix	f0049152	dv_m	8	14.806436	4259107353	2457684
gsjgix	f0049152	dv_g	K20	2.164367	2083893298	2457684
gsjgix	f0196608	dv_g	K20	19.483657	2193105709	9830484

host	model	run	threads	sec.	CRC	bytes
sp8ct	aizu5.04	pxyz	-	4.286982	3062843378	750924
sp8ct	aizu5.04	pxyz_s	-	2.169572	3062843378	750924
sp8ct	aizu5.04	pxyz_m	1	2.167826	3062843378	750924
sp8ct	aizu5.04	pxyz_m	2	1.124208	3062843378	750924
sp8ct	aizu5.04	pxyz_m	4	0.584597	3062843378	750924
sp8ct	aizu5.04	pxyz_m	8	0.317376	3062843378	750924
sp8ct	aizu5.04	pxyz_g	GTX_680	0.341376	1614961764	750924
sp8ct	f0049152	pxyz	-	582.084603	512884516	8606574
sp8ct	f0049152	pxyz_s	-	324.629597	512884516	8606574
sp8ct	f0049152	pxyz_m	1	340.396439	512884516	8606574
sp8ct	f0049152	pxyz_m	2	167.157398	512884516	8606574
sp8ct	f0049152	pxyz_m	4	94.311451	512884516	8606574
sp8ct	f0049152	pxyz_m	8	59.337903	512884516	8606574
sp8ct	f0049152	pxyz_g	GTX_680	2.266897	3014388987	8606574
sp8ct	f0196608	pxyz_g	GTX_680	25.183020	395161540	34425798

host	model	run	threads	sec.	CRC	bytes
sp8ct	aizu5.04	np	-	3.986953	1899975242	214334
sp8ct	aizu5.04	np_s	-	1.735501	1899975242	214334
sp8ct	aizu5.04	np_m	1	1.733718	1899975242	214334
sp8ct	aizu5.04	np_m	2	0.910165	1899975242	214334
sp8ct	aizu5.04	np_m	4	0.461533	1899975242	214334
sp8ct	aizu5.04	np_m	8	0.249427	1899975242	214334
sp8ct	aizu5.04	np_g	GTX_680	0.320758	2500797220	214334
sp8ct	f0049152	np	-	520.571663	4202476463	2457684
sp8ct	f0049152	np_s	-	264.006507	4202476463	2457684
sp8ct	f0049152	np_m	1	265.213113	4202476463	2457684
sp8ct	f0049152	np_m	2	134.495656	4202476463	2457684
sp8ct	f0049152	np_m	4	78.705161	4202476463	2457684
sp8ct	f0049152	np_m	8	50.626403	4202476463	2457684
sp8ct	f0049152	np_g	GTX_680	2.252894	829119505	2457684
sp8ct	f0196608	np_g	GTX_680	24.484158	3715266744	9830484

host	model	run	threads	sec.	CRC	bytes
sp8ct	aizu5.04	pg	-	3.937157	837575810	750978
sp8ct	aizu5.04	pg_s	-	1.796343	837575810	750978
sp8ct	aizu5.04	pg_m	1	1.777643	837575810	750978
sp8ct	aizu5.04	pg_m	2	0.907606	837575810	750978
sp8ct	aizu5.04	pg_m	4	0.480670	837575810	750978
sp8ct	aizu5.04	pg_m	8	0.264500	837575810	750978
sp8ct	aizu5.04	pg_g	GTX_680	0.339634	4060343442	750978
sp8ct	f0049152	pg	-	513.739871	590664062	8606628
sp8ct	f0049152	pg_s	-	268.320953	590664062	8606628
sp8ct	f0049152	pg_m	1	260.783039	590664062	8606628
sp8ct	f0049152	pg_m	2	135.279980	590664062	8606628
sp8ct	f0049152	pg_m	4	79.327020	590664062	8606628
sp8ct	f0049152	pg_m	8	51.497196	590664062	8606628
sp8ct	f0049152	pg_g	GTX_680	2.434625	1008550782	8606628
sp8ct	f0196608	pg_g	GTX_680	25.236561	986363691	34425853

host	model	run	threads	sec.	CRC	bytes
sp8ct	aizu5.04	dv	-	3.946017	3202635282	214334
sp8ct	aizu5.04	dv_s	-	1.748783	3202635282	214334
sp8ct	aizu5.04	dv_m	1	1.739281	3202635282	214334
sp8ct	aizu5.04	dv_m	2	0.894958	3202635282	214334
sp8ct	aizu5.04	dv_m	4	0.464259	3202635282	214334
sp8ct	aizu5.04	dv_m	8	0.249775	3202635282	214334
sp8ct	aizu5.04	dv_g	GTX_680	0.324601	3202635282	214334
sp8ct	f0049152	dv	-	517.498430	4259107353	2457684
sp8ct	f0049152	dv_s	-	266.480450	4259107353	2457684
sp8ct	f0049152	dv_m	1	269.697182	4259107353	2457684
sp8ct	f0049152	dv_m	2	134.091549	4259107353	2457684
sp8ct	f0049152	dv_m	4	73.865344	4259107353	2457684
sp8ct	f0049152	dv_m	8	44.867669	4259107353	2457684
sp8ct	f0049152	dv_g	GTX_680	2.051323	2083893298	2457684
sp8ct	f0196608	dv_g	GTX_680	23.664590	2193105709	9830484

添付ファイル "cpu.pdf "（3 / 4）

host	model	run	threads	sec.	CRC	bytes
gsjvix	aizu5.04	pxyz	-	3.617518	3062843378	750924
gsjvix	aizu5.04	pxyz_s	-	1.674990	3062843378	750924
gsjvix	aizu5.04	pxyz_m	1	1.673354	3062843378	750924
gsjvix	aizu5.04	pxyz_m	2	0.961170	3062843378	750924
gsjvix	aizu5.04	pxyz_m	4	0.456683	3062843378	750924
gsjvix	aizu5.04	pxyz_m	8	0.262832	3062843378	750924
gsjvix	aizu5.04	pxyz_g	C2070	0.216365	2439447421	750924
gsjvix	f0049152	pxyz	-	469.589264	512884516	8606574
gsjvix	f0049152	pxyz_s	-	219.924151	512884516	8606574
gsjvix	f0049152	pxyz_m	1	219.308322	512884516	8606574
gsjvix	f0049152	pxyz_m	2	111.395830	512884516	8606574
gsjvix	f0049152	pxyz_m	4	58.129607	512884516	8606574
gsjvix	f0049152	pxyz_m	8	28.680794	512884516	8606574
gsjvix	f0049152	pxyz_g	C2070	2.840807	1280287708	8606574
gsjvix	f0196608	pxyz_g	C2070	37.386007	2377971899	34425799

host	model	run	threads	sec.	CRC	bytes
gsjvix	aizu5.04	pg	-	3.548098	837575810	750978
gsjvix	aizu5.04	pg_s	-	1.419824	837575810	750978
gsjvix	aizu5.04	pg_m	1	1.413682	837575810	750978
gsjvix	aizu5.04	pg_m	2	0.731813	837575810	750978
gsjvix	aizu5.04	pg_m	4	0.390173	837575810	750978
gsjvix	aizu5.04	pg_m	8	0.225424	837575810	750978
gsjvix	aizu5.04	pg_g	C2070	0.267401	581682643	750978
gsjvix	f0049152	pg	-	462.374084	590664062	8606628
gsjvix	f0049152	pg_s	-	185.719775	590664062	8606628
gsjvix	f0049152	pg_m	1	185.226964	590664062	8606628
gsjvix	f0049152	pg_m	2	92.540368	590664062	8606628
gsjvix	f0049152	pg_m	4	49.266831	590664062	8606628
gsjvix	f0049152	pg_m	8	24.254286	590664062	8606628
gsjvix	f0049152	pg_g	C2070	2.694715	1505280642	8606628
gsjvix	f0196608	pg_g	C2070	34.265602	2068349057	34425853

host	model	run	threads	sec.	CRC	bytes
gsjvix	aizu5.04	np	-	3.529473	1899975242	214334
gsjvix	aizu5.04	np_s	-	1.376690	1899975242	214334
gsjvix	aizu5.04	np_m	1	1.359297	1899975242	214334
gsjvix	aizu5.04	np_m	2	0.702892	1899975242	214334
gsjvix	aizu5.04	np_m	4	0.367286	1899975242	214334
gsjvix	aizu5.04	np_m	8	0.203089	1899975242	214334
gsjvix	aizu5.04	np_g	C2070	0.265409	1899975242	214334
gsjvix	f0049152	np	-	461.377910	4202476463	2457684
gsjvix	f0049152	np_s	-	182.100676	4202476463	2457684
gsjvix	f0049152	np_m	1	198.760964	4202476463	2457684
gsjvix	f0049152	np_m	2	91.038664	4202476463	2457684
gsjvix	f0049152	np_m	4	45.425167	4202476463	2457684
gsjvix	f0049152	np_m	8	23.735029	4202476463	2457684
gsjvix	f0049152	np_g	C2070	2.505152	2990482038	2457684
gsjvix	f0196608	np_g	C2070	34.232158	4085435141	9830484

host	model	run	threads	sec.	CRC	bytes
gsjvix	aizu5.04	dv	-	3.532582	3202635282	214334
gsjvix	aizu5.04	dv_s	-	1.398293	3202635282	214334
gsjvix	aizu5.04	dv_m	1	1.397031	3202635282	214334
gsjvix	aizu5.04	dv_m	2	0.718765	3202635282	214334
gsjvix	aizu5.04	dv_m	4	0.378144	3202635282	214334
gsjvix	aizu5.04	dv_m	8	0.216956	3202635282	214334
gsjvix	aizu5.04	dv_g	C2070	0.197959	3202635282	214334
gsjvix	f0049152	dv	-	461.037828	4259107353	2457684
gsjvix	f0049152	dv_s	-	184.738263	4259107353	2457684
gsjvix	f0049152	dv_m	1	184.657898	4259107353	2457684
gsjvix	f0049152	dv_m	2	93.101940	4259107353	2457684
gsjvix	f0049152	dv_m	4	47.249561	4259107353	2457684
gsjvix	f0049152	dv_m	8	24.254719	4259107353	2457684
gsjvix	f0049152	dv_g	C2070	2.407201	1749090230	2457684
gsjvix	f0196608	dv_g	C2070	32.648322	1859481159	9830484

host	model	run	threads	sec.	CRC	bytes
gsjkic	aizu5.04	pxyz	-	6.635678	3685333278	750924
gsjkic	aizu5.04	pxyz_s	-	4.248771	3685333278	750924
gsjkic	aizu5.04	pxyz_m	1	4.324267	3685333278	750924
gsjkic	aizu5.04	pxyz_m	2	2.226369	3685333278	750924
gsjkic	aizu5.04	pxyz_m	4	2.221015	3685333278	750924
gsjkic	aizu5.04	pxyz_m	8	2.221544	3685333278	750924
gsjkic	aizu5.04	pxyz_g	C1060	0.161420	2478090178	750924
gsjkic	f0049152	pxyz	-	863.324678	876836119	8606574
gsjkic	f0049152	pxyz_s	-	557.441794	876836119	8606574
gsjkic	f0049152	pxyz_m	1	562.337855	876836119	8606574
gsjkic	f0049152	pxyz_m	2	281.375571	876836119	8606574
gsjkic	f0049152	pxyz_m	4	281.316934	876836119	8606574
gsjkic	f0049152	pxyz_m	8	281.229684	876836119	8606574
gsjkic	f0049152	pxyz_g	C1060	3.100790	2679581633	8606574
gsjkic	f0196608	pxyz_g	C1060	36.916614	1003998830	34425798

host	model	run	threads	sec.	CRC	bytes
gsjkic	aizu5.04	np	-	6.553352	1899975242	214334
gsjkic	aizu5.04	np_s	-	3.962813	1899975242	214334
gsjkic	aizu5.04	np_m	1	3.995633	1899975242	214334
gsjkic	aizu5.04	np_m	2	2.055203	1899975242	214334
gsjkic	aizu5.04	np_m	4	2.045914	1899975242	214334
gsjkic	aizu5.04	np_m	8	2.048268	1899975242	214334
gsjkic	aizu5.04	np_g	C1060	0.134734	3662596069	214334
gsjkic	f0049152	np	-	852.884612	4202476463	2457684
gsjkic	f0049152	np_s	-	519.194607	4202476463	2457684
gsjkic	f0049152	np_m	1	519.788449	4202476463	2457684
gsjkic	f0049152	np_m	2	260.221720	4202476463	2457684
gsjkic	f0049152	np_m	4	260.328830	4202476463	2457684
gsjkic	f0049152	np_m	8	260.147047	4202476463	2457684
gsjkic	f0049152	np_g	C1060	2.579625	2946272878	2457684
gsjkic	f0196608	np_g	C1060	32.634672	224297550	9830484

host	model	run	threads	sec.	CRC	bytes
gsjkic	aizu5.04	pg	-	6.547848	302857074	750978
gsjkic	aizu5.04	pg_s	-	4.016005	302857074	750978
gsjkic	aizu5.04	pg_m	1	4.045816	302857074	750978
gsjkic	aizu5.04	pg_m	2	2.077185	302857074	750978
gsjkic	aizu5.04	pg_m	4	2.077301	302857074	750978
gsjkic	aizu5.04	pg_m	8	2.102174	302857074	750978
gsjkic	aizu5.04	pg_g	C1060	0.158298	2703475146	750978
gsjkic	f0049152	pg	-	853.333844	297303935	8606628
gsjkic	f0049152	pg_s	-	523.539742	297303935	8606628
gsjkic	f0049152	pg_m	1	523.890872	297303935	8606628
gsjkic	f0049152	pg_m	2	262.342495	297303935	8606628
gsjkic	f0049152	pg_m	4	262.337049	297303935	8606628
gsjkic	f0049152	pg_m	8	262.305610	297303935	8606628
gsjkic	f0049152	pg_g	C1060	2.906979	4141494098	8606628
gsjkic	f0196608	pg_g	C1060	34.168412	1567380255	34425853

host	model	run	threads	sec.	CRC	bytes
gsjkic	aizu5.04	dv	-	6.703684	3202635282	214334
gsjkic	aizu5.04	dv_s	-	3.947847	3202635282	214334
gsjkic	aizu5.04	dv_m	1	3.989153	3202635282	214334
gsjkic	aizu5.04	dv_m	2	2.035459	3202635282	214334
gsjkic	aizu5.04	dv_m	4	2.039595	3202635282	214334
gsjkic	aizu5.04	dv_m	8	2.048720	3202635282	214334
gsjkic	aizu5.04	dv_g	C1060	0.135192	4195040585	214334
gsjkic	f0049152	dv	-	877.174596	4259107353	2457684
gsjkic	f0049152	dv_s	-	518.669955	4259107353	2457684
gsjkic	f0049152	dv_m	1	518.836269	4259107353	2457684
gsjkic	f0049152	dv_m	2	259.648293	4259107353	2457684
gsjkic	f0049152	dv_m	4	259.559340	4259107353	2457684
gsjkic	f0049152	dv_m	8	259.497179	4259107353	2457684
gsjkic	f0049152	dv_g	C1060	2.628079	3430372380	2457684
gsjkic	f0196608	dv_g	C1060	32.977599	1558741057	9830484

Date: Sat, 06 Jul 2013 15:36:46 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_4

みなさま、

GSJ/AIST のなかのです。かなり間が空きましたが、先日の E-mail の続きです。

On Thu, 13 Jun 2013 13:35:38 +0900 Tsukasa NAKANO wrote:

> みなさま、
>
> GSJ/AIST のなかのです。以前に紹介した「STL データとして与えられた物体像に対して
> その内部の密度が一様と仮定した場合の重力ポテンシャルや重力加速度ベクトルの値を計算
> するプログラム」のうちの主要なものに対して以下の 4 種類の高速化を行いました。
>
> (1) 「段取り」の見直しによる NP と DV の計算の高速化（省力化？）
> (2) 面積分の計算アルゴリズムの改良による高速化
> (3) マルチスレッド処理による高速化
> (4) CUDA GPU 用のプログラム
> ...
> 長くなったので今回はここまで。予告した「(4) CUDA GPU 用のプログラム」に関する話は
> 次便以降の E-mail に回します。とり急ぎ、

(3) マルチスレッド処理による高速化（補足）

以前の E-mail にマルチスレッド処理で何をどのように並列計算するかの説明を書き忘れていました。これは後述する CUDA GPU 用のプログラムの処理内容とも関係するので、ここで少し説明しておきます。

ここで紹介しているプログラムは重力の源である物体の表面を構成している多数の三角形 Tm（m = 1～M）のそれぞれに関する面積分 I() を積算して、指定した観測点 O での重力の値（例えば、重力ポテンシャル P）を計算しています。

$$P(O) = I(T1,O) + I(T2,O) + ... + I(TM,O)$$

この計算は観測点 Ol（l = 1～L）ごとに個別に行えるので、マルチスレッド版のプログラムは指定したスレッド数に応じた複数個の観測点における値 P(Ol) をマルチスレッドで並列に処理するようになっています。また、CUDA GPU 用のプログラムでは、観測点すべての値を同時に計算します（理論的には）。

話が少し脱線しますが、重力計算用のプログラムの処理速度を測る「時定数」についても書いておきます。上に書いたように、これらが行う計算処理は観測点の総数 L と物体像表面の三角形の総数 M の積に比例した手間を要します。そして、これらの処理時間を L×M で割ることにより、処理に用いるデータの量に依存しない時定数の値を得ることができます。ただし、プログラム stl_pxyz* に観測点の位置を陽に指定した場合以外は L ～ M（物体像表面の三角形の重心が観測点）なので、結局、観測点の位置を指定しないで起動した stl_pxyz*、stl_pg*、stl_np* および stl_dv* の時定数の値を以下の式で計算できます。

時定数 ～ プログラムの実行時間 / (物体像表面の三角形の総数 M)^ 2

添付した PDF ファイル ns.pdf をご覧下さい。これらの表の nano sec. の欄に示した値がナノ秒（10^-9 秒）単位の時定数です。それぞれのページの上の表に以前に紹介したベンチマークテストの実行時間から算出した値を記しました。

On Thu, 13 Jun 2013 13:35:38 +0900 Tsukasa NAKANO wrote:

> host
> ベンチマークテストを実行した Linux 計算機のホスト名
> gsjgix : Dell Precision T7600、Xeon E5-2687W CPU（８ cores）
> sp8ct : Dell Precision T7400、Xeon E5420 ×２（合計８ cores）
> gsjvix : Dell Precision T7500、Xeon X5570 ×２（合計８ cores）
> gsjkic : Dell Precision 390、Core2 Duo 6700（２ cores）
> model
> STL データ（小惑星イトカワの形状モデル）の名前
> aizu5.04 : 会津大学のモデル。facet の総数は 4285。
> f0049152 : JPL の Gaskell のモデル。facet の総数は 49152。
> f0196608 : 同上。ただし、facet の総数は 196608 = 49152 ×４。
> run
> 実行したプログラムの名前 stl_*。ただし、語尾に "_g" が付いている
> ものは後で説明する CUDA GPU 用のプログラム。
> threads
> 計算に使用したスレッド数。ただし、"-" は single thread。また、
> K20、GTX_680、C2070 と C1060 は CUDA GPU の機種。
> sec.
> プログラムの秒単位の実行時間（起動から終了までの経過時間）

また、ns.pdf のそれぞれのページの下の表には後で説明する「チューニング・パラメータ」を様々に変えてコンパイルした CUDA GPU 用プログラムそれぞれの実行速度と時定数の値を記しました。これらは、GPU の欄に記した GPU の上で object の欄に記した CUDA GPU 用プログラム stl_*_g を実行して、ns.pdf の上の表の最後の行のものと同一の形状モデル f0196608（M = 196608）を処理した結果です。ただし、括弧で囲んだホスト名が GPU の欄に記されている行は８個のスレッドを並列稼動したマルチス

レッド版プログラム stl_*_m による結果で、比較のために載せました。そして、NVCC と DEFS の欄に記したものが具体的なチューニング・パラメータなどですが、これらについては後で説明します。

チューニング･パラメータを変えると CUDA GPU 用プログラムの処理速度は大きく変わりますが、これは後で説明する計算精度とのトレード・オフが関係し、今のところ ns.pdf の下の表に太字で記した「ほどほどの性能」になるパラメータを採用しています。なお、ns.pdf の上の表でもこのパラメータを採用した CUDA GPU 用プログラムを使いました。

(4) CUDA GPU 用のプログラム

書庫ファイル http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip の中のディレクトリ gravity/cuda/ の下に入れた CUDA GPU 用のプログラムのソースファイルなどは Linux 用の CUDA toolkit を使ってコンパイル・インストールすることを想定しています。さらに、Linux 上でも gravity/cuda/Makefile に書き込まれている CUDA GPU の機種に応じた「チューニング・パラメータ」などを再設定する必要があると思われます。このようなコンパイル・インストールに関する話は後回しにして、プログラムの使用法などについてまず説明します。

CUDA GPU の上で STL データを使って重力を計算するプログラムにはこれまでに紹介した（通常 CPU 用の）旧版、新版およびマルチスレッド版のプログラムと完全互換な以下の 4 個があります。

stl_pxyz_g　orgSTL　{point}
stl_pg_g　 STL
stl_np_g　orgSTL　{{min　max}　CM_STL　newSTL}
stl_dv_g　orgSTL　{{min　max}　CM_STL　newSTL}

ただし、ホスト計算機に装着した複数の CUDA GPUs のうちの特定のものを処理に使いたい時は、プログラムの起動前にその「装置番号」を環境変数 CUDA_GPU に設定しておく必要があります（この設定がない場合、装置番号 0 の CUDA GPU を使って処理を行います)。

UNIX で C-shell 系 shell（csh や tcsh）を使っている場合
　setenv　CUDA_GPU　装置番号
B-shell（sh や bash）
　CUDA_GPU=装置番号 &&　export　CUDA_GPU
Windows のコマンドプロンプト
　set　CUDA_GPU=装置番号

CUDA GPU の装置番号は CUDA SDK のサンプルプログラム deviceQuery で調べることができます。また、ぼくが書いたプログラム cuda_gpu でそれを標準出力に書き出すこともできます。そのためには CUDA toolkit をインストールしてある環境で以下を行って下さい。

以下の 2 個の書庫ファイル（内容は同じ）のいずれかをダウンロードする。
　http://www-bl20.spring8.or.jp/~sp8ct/tmp/cuda_gpu.taz

http://www-bl20.spring8.or.jp/~sp8ct/tmp/cuda_gpu.zip

これを解凍・展開したディレクトリで以下を入力する（コンパイルエラーが発生したら、テキストファイル make.cuda_gpu を適当に修正して下さい）。

```
sh   make.cuda_gpu
```

プログラム cuda_gpu を正常に作成できたら、それを以下のように起動する。

```
cuda_gpu
→
    0    装置番号０の CUDA GPU の機種名
    1    装置番号１の ...
    ...
```

さて、CUDA GPU 用のプログラムのコンパイル・インストール法ですが、これは書庫ファイル gravity.zip を解凍・展開して得たディレクトリ gravity/cuda/ に移動した後、以下のように入力するだけです。

```
make              ← とりあえずプログラムをコンパイルしてみる。
make  install     ← それに成功したらプログラムをインストールする。
make  clean       ← 不要になったファイルを消去
```

これにより、以前に紹介した CPU 用のものと同様に、CUDA GPU 用の４個のプログラムがディレクトリ gravity/cuda/bin/ にコピーされます。

以上のようなコンパイル・インストールの処理に失敗した時や、CUDA GPU 用のプログラムをチューニングしたい場合は、ファイル gravity/cuda/Makefile の設定内容を修正する必要があります。具体的には、以下の４個の「make 変数」CUDA、LIBS、NVCC および DEFS の内容を再設定すれば良いだけです。

コンパイルに失敗した場合に設定内容を修正すべき変数：

CUDA（現在の設定内容は「/usr/local/cuda」）

CUDA toolkit をインストールしたディレクトリの名前を設定する。

LIBS（「-L$(CUDA)/lib64 ...」）

現在は 64 ビット CPU 用 Linux に対応した CUDA runtime library を使うように設定してあるので、32 ビット CPU 用 Linux の環境の場合には「lib64」を「lib」に書き換える必要がある。

プログラムのチューニングのための変数：

NVCC（「nvcc　-O3　-arch=sm_13」）

この設定の前半の CUDA コンパイラの名前「nvcc」と実行コードの最適化レベル「-O3」は修正不要。それらに続く「-arch=sm_*」の「*（２桁の整数値）」は CUDA GPU の機種ごとに異なる compute capability で、NVIDIA のページ https://developer.nvidia.com/cuda-gpus を参考にして使用する CUDA GPU の機種に応じた値を設定する必要がある。なお、CUDA GPU は上位互換なので、それに適したものよりも低い compute capablity の値を設定しても問題ない

（色々と試した訳ではないですが、そうした方が高速に動作する実行コードになることもあるようです）。

DEFS（現在は何も設定していない）

下記の CUDA GPU 用プログラムの３個の「チューニング・パラメータ」を組み合わせた合計 2^3 = ８通りのいずれかを設定する。

－use_fast_math

これは CUDA コンパイラ nvcc のオプション指定。精度はやや落ちるが高速な数値計算用のコードを生成する。

－DDOUBLE

これはぼくが書いた CUDA コードに固有のマクロ定数の宣言。これを指定すると数値計算を倍精度の浮動小数点演算で行う。なお、この指定を省略すると、数値計算を単精度で実行する。

－DKEPLER

これもぼくが書いたコードに固有のマクロ定数の宣言。詳細は後で説明しますが、最新の Kepler アーキテクチャーの CUDA GPU を使う場合に指定すると良い（Fermi アーキテクチャなどのものを使う場合に指定しても問題ないですが、実行コードの処理速度は遅くなります）。

なお、これは後になって気づいたことですが、「-DDOUBLE」で指定した倍精度浮動小数点演算には「-use_fast_math」は効果がないようです。つまり、「-DDOUBLE　-use_fast_math」の組み合わせを指定しても nvcc は「-DDOUBLE」だけを指定した時と同じ実行コードを生成します。

上記の「-DKEPLER」の指定の有無によって、CUDA GPU 用のプログラムは重力の計算に使用する CUDA コードを切り替えます。これら２種類のコードはどちらも gravity/cuda/si.cu に書き込まれており、その内容の違いを一言で言うと、以前に紹介した CPU 用の新旧のプログラムで使っている C 言語コード（旧版の gravity/src/si.c と新版の gravity/src/si.h）の違いに概ね対応しています。

On Tue, 09 Apr 2013 19:03:09 +0900 Tsukasa NAKANO wrote:

> (4) 関数 SurfaceIntegral() について
>
> 単純な形状の立方体や GLOBE に対するものも含めて、ここで紹介している STL データと
> して与えられた物体像による重力の値を計算するプログラムはすべて、gravity/src/si.c に
> 記した C 言語の関数 SurfaceIntegral () を使っています。この関数は STL の facet それぞれ
> に対して論文
>　　gravity/doc/Werner_1996.pdf
>　　http://adsabs.harvard.edu/full/1997CeMDA..65..313W
> に載っていた（Gauss の発散定理と Stokes の公式を用いて体積積分から変換した）面積分

> の式をそのまま使っているため、処理速度に難があります。個々の facet の形状だけに
> 依存する量（現在の si.c のコードで大文字で記してある auto 変数 D??、N? や O??? の
> 値）を最初にまとめて計算してメモリに格納し、それらを面積分で使うようにすれば、
> 現状の倍近い処理速度を達成できると思われます。

On Thu, 13 Jun 2013 13:35:38 +0900 Tsukasa NAKANO wrote:

> (2) 面積分の計算アルゴリズムの改良による高速化
> ...
> 改良したコードはインクルードファイル gravity/src/si.h に書き込みました。

ただし、これが CUDA GPU のプログラミングで最も面白く、通常 CPU のものの常識に反することですが、「-DKEPLER」を指定した場合の最新の Kepler アーキテクチャ用のコードは CPU 用の新版のプログラムで使っている si.h ではなく、旧版で使っていた si.c に近い処理内容になっています。このようになっている理由の詳細は省略しますが、簡単に言うとそれは以下の通りです。

> CUDA GPU は超多数の GPU コアが個別に行うメモリ・アクセスが苦手で、それを補うために複数のコアが協調して動作する必要がある。しかし、その際に共有して使用できるメモリの量にも制限があるため、si.h のように事前の計算が可能な値をメモリに置いておいても、さほど高速化できない。特に、個々のコアの処理能力が高い Kepler アーキテクチャの CUDA GPU では、事前に計算した値をメモリから持って来るよりも、si.c のようにそれらをその都度計算し直した方が高速になる。

これ（「-DKEPLER」）以外の make 変数 DEFS に設定可能な「-use_fast_math」と「-DDOUBLE」は、CUDA GPU 用プログラムの実行速度と計算精度のトレード･オフに関係します。これら（の組み合わせ）を指定したそれぞれの場合に実際の計算結果がどうなるのかを小惑星イトカワの形状モデル f0196608 を使って調べてみました。添付した PDF ファイル gpu.pdf の表をご覧下さい。ただし、その各欄の項目の意味は以下の通りです。

GPU
: 使用した CUDA GPU の機種名。ただし、括弧で囲んだものは計算精度の基準とした CPU 用のプログラムを実行したホスト計算機の名前。

object
: 実行した CUDA GPU 用のプログラムの名前 stl_*_g。ただし、CPU 用のものは８スレッドを並列稼動したプログラム stl_*_m。

NVCC
: make 変数 NVCC の CUDA GPU の compute capibilty の設定

DEFS
: make 変数 DEFS に設定した内容

sec.
: プログラムの秒単位の実行時間（起動から終了までの経過時間）

dP min.、dP max.、
dX min.、dX max.、
dY min.、dY max.、
dZ min.、dZ max.

これらはそれぞれ計算結果の重力ポテンシャル（P）と重力加速度ベクトルの３成分（X、Y、Z）に関する値の最小値と最大値。ただし、括弧で囲んだ CPU の値は物体像全体の P、X、Y、Z そのものの値の最小値と最大値。また、CUDA GPU の行のものはそれぞれの観測点における「CUDA GPU で 計算した値」と「CPU で計算した値」の「差」の最小値と最大値。

ぼくは gpu.pdf の表に太字で記した「ほどほどの性能」を達成できそうなチューニング・パラメータを指定した CUDA GPU 用プログラムを実戦で使用するつもりでした。しかし、それらよりも２倍以上も高速に動作する「-use_fast_math」を指定したプログラムでも計算精度は概ね同じです。CUDA GPU 用のプログラムのコンパイルの際にはこれを指定すべきでしょうね。なお、倍精度浮動小数点数演算で得た CPU の計算結果との比較から、「-DDOUBLE」を指定しない単精度演算でも計算精度に問題はない感じです。

またしても長い E-mail になってしまいました。とりあえず、ここまで。

添付ファイル "ns.pdf "（1 / 4）

host	model	run	threads	sec.	nano sec.
gsjgix	f0049152	pxyz	-	336.893807	139.447470
gsjgix	f0049152	pxyz_s	-	124.210253	51.413250
gsjgix	f0049152	pxyz_m	1	123.667104	51.188429
gsjgix	f0049152	pxyz_m	2	65.461036	27.095707
gsjgix	f0049152	pxyz_m	4	33.861472	14.015979
gsjgix	f0049152	pxyz_m	8	17.614100	7.290848
gsjgix	f0049152	pxyz_g	K20	2.205713	0.912991
gsjgix	f0196608	pxyz_g	K20	20.482203	0.529876
gsjgix	f0049152	pg	-	318.973911	132.030046
gsjgix	f0049152	pg_s	-	104.414618	43.219418
gsjgix	f0049152	pg_m	1	104.467494	43.241305
gsjgix	f0049152	pg_m	2	55.297099	22.888638
gsjgix	f0049152	pg_m	4	28.696931	11.878267
gsjgix	f0049152	pg_m	8	14.962336	6.193227
gsjgix	f0049152	pg_g	K20	2.336241	0.967020
gsjgix	f0196608	pg_g	K20	20.253148	0.523950
gsjgix	f0049152	np	-	318.111434	131.673049
gsjgix	f0049152	np_s	-	102.229369	42.314897
gsjgix	f0049152	np_m	1	102.191257	42.299122
gsjgix	f0049152	np_m	2	54.092022	22.389832
gsjgix	f0049152	np_m	4	27.947149	11.567916
gsjgix	f0049152	np_m	8	14.498661	6.001302
gsjgix	f0049152	np_g	K20	2.144160	0.887513
gsjgix	f0196608	np_g	K20	20.395434	0.527631
gsjgix	f0049152	dv	-	318.100209	131.668403
gsjgix	f0049152	dv_s	-	104.606342	43.298777
gsjgix	f0049152	dv_m	1	104.480619	43.246737
gsjgix	f0049152	dv_m	2	55.264998	22.875351
gsjgix	f0049152	dv_m	4	28.517266	11.803899
gsjgix	f0049152	dv_m	8	14.806436	6.128697
gsjgix	f0049152	dv_g	K20	2.164367	0.895877
gsjgix	f0196608	dv_g	K20	19.483657	0.504044

GPU	object	NVCC	DEFS	sec.	nano sec.
(gsjgix)	pxyz_m_8			288.427092	7.461629
K20	pxyz	-arch=sm_35		27.296900	0.706173
K20	pxyz	-arch=sm_35	-use_fast_math	9.477799	0.245191
K20	pxyz	-arch=sm_35	-DDOUBLE	90.487250	2.340912
K20	pxyz	-arch=sm_35	-DDOUBLE -use_fast_math	90.322671	2.336654
K20	**pxyz**	**-arch=sm_35**	**-DKEPLER**	**20.543892**	**0.531472**
K20	pxyz	-arch=sm_35	-DKEPLER -use_fast_math	7.814951	0.202173
K20	pxyz	-arch=sm_35	-DKEPLER -DDOUBLE	53.798292	1.391766
K20	pxyz	-arch=sm_35	-DKEPLER -DDOUBLE -use_fast_math	53.757530	1.390711
(gsjgix)	pg_m_8			245.675027	6.355631
K20	pg	-arch=sm_35		26.653264	0.689522
K20	pg	-arch=sm_35	-use_fast_math	9.187943	0.237693
K20	pg	-arch=sm_35	-DDOUBLE	88.880779	2.299352
K20	pg	-arch=sm_35	-DDOUBLE -use_fast_math	88.801726	2.297307
K20	**pg**	**-arch=sm_35**	**-DKEPLER**	**20.259451**	**0.524113**
K20	pg	-arch=sm_35	-DKEPLER -use_fast_math	7.529857	0.194798
K20	pg	-arch=sm_35	-DKEPLER -DDOUBLE	53.163159	1.375335
K20	pg	-arch=sm_35	-DKEPLER -DDOUBLE -use_fast_math	53.229599	1.377054

添付ファイル "ns.pdf "（2 / 4）

host	model	run	threads	sec.	nano sec.
sp8ct	f0049152	pxyz	-	582.084603	240.937125
sp8ct	f0049152	pxyz_s	-	324.629597	134.371054
sp8ct	f0049152	pxyz_m	1	340.396439	140.897284
sp8ct	f0049152	pxyz_m	2	167.157398	69.189981
sp8ct	f0049152	pxyz_m	4	94.311451	39.037504
sp8ct	f0049152	pxyz_m	8	59.337903	24.561213
sp8ct	f0049152	pxyz_g	GTX_680	2.266897	0.938317
sp8ct	f0196608	pxyz_g	GTX_680	25.183020	0.651487
sp8ct	f0049152	pg	-	513.739871	212.647795
sp8ct	f0049152	pg_s	-	268.320953	111.063716
sp8ct	f0049152	pg_m	1	260.783039	107.943614
sp8ct	f0049152	pg_m	2	135.279980	55.995244
sp8ct	f0049152	pg_m	4	79.327020	32.835131
sp8ct	f0049152	pg_m	8	51.497196	21.315778
sp8ct	f0049152	pg_g	GTX_680	2.434625	1.007743
sp8ct	f0196608	pg_g	GTX_680	25.236561	0.652872
sp8ct	f0049152	np	-	520.571663	215.475618
sp8ct	f0049152	np_s	-	264.006507	109.277875
sp8ct	f0049152	np_m	1	265.213113	109.777315
sp8ct	f0049152	np_m	2	134.495656	55.670596
sp8ct	f0049152	np_m	4	78.705161	32.577730
sp8ct	f0049152	np_m	8	50.626403	20.955339
sp8ct	f0049152	np_g	GTX_680	2.252894	0.932520
sp8ct	f0196608	np_g	GTX_680	24.484158	0.633407
sp8ct	f0049152	dv	-	517.498430	214.203542
sp8ct	f0049152	dv_s	-	266.480450	110.301893
sp8ct	f0049152	dv_m	1	269.697182	111.633366
sp8ct	f0049152	dv_m	2	134.091549	55.503327
sp8ct	f0049152	dv_m	4	73.865344	30.574428
sp8ct	f0049152	dv_m	8	44.867669	18.571677
sp8ct	f0049152	dv_g	GTX_680	2.051323	0.849086
sp8ct	f0196608	dv_g	GTX_680	23.664590	0.612205

GPU	object	NVCC	DEFS	sec.	nano sec.
(sp8ct)	pxyz_m_8			1217.761176	31.503569
GTX_680	pxyz	-arch=sm_30		33.618992	0.869726
GTX_680	pxyz	-arch=sm_30	-use_fast_math	10.275388	0.265825
GTX_680	pxyz	-arch=sm_30	-DDOUBLE	206.491671	5.341954
GTX_680	pxyz	-arch=sm_30	-DDOUBLE -use_fast_math	207.038048	5.356089
GTX_680	**pxyz**	**-arch=sm_30**	**-DKEPLER**	**26.039547**	**0.673645**
GTX_680	pxyz	-arch=sm_30	-DKEPLER -use_fast_math	9.364018	0.242248
GTX_680	pxyz	-arch=sm_30	-DKEPLER -DDOUBLE	231.582772	5.991063
GTX_680	pxyz	-arch=sm_30	-DKEPLER -DDOUBLE -use_fast_math	231.560142	5.990477
(sp8ct)	pg_m_8			1169.738297	30.261213
GTX_680	pg	-arch=sm_30		32.383118	0.837754
GTX_680	pg	-arch=sm_30	-use_fast_math	10.004385	0.258814
GTX_680	pg	-arch=sm_30	-DDOUBLE	202.896135	5.248938
GTX_680	pg	-arch=sm_30	-DDOUBLE -use_fast_math	203.362629	5.261006
GTX_680	**pg**	**-arch=sm_30**	**-DKEPLER**	**26.075013**	**0.674562**
GTX_680	pg	-arch=sm_30	-DKEPLER -use_fast_math	10.302571	0.266528
GTX_680	pg	-arch=sm_30	-DKEPLER -DDOUBLE	225.427023	5.831813
GTX_680	pg	-arch=sm_30	-DKEPLER -DDOUBLE -use_fast_math	225.455733	5.832556

添付ファイル "ns.pdf "（3 / 4）

host	model	run	threads	sec.	nano sec.
gsjvix	f0049152	pxyz	-	469.589264	194.372925
gsjvix	f0049152	pxyz_s	-	219.924151	91.031256
gsjvix	f0049152	pxyz_m	1	219.308322	90.776352
gsjvix	f0049152	pxyz_m	2	111.395830	46.109089
gsjvix	f0049152	pxyz_m	4	58.129607	24.061073
gsjvix	f0049152	pxyz_m	8	28.680794	11.871587
gsjvix	f0049152	pxyz_g	C2070	2.840807	1.175870
gsjvix	f0196608	pxyz_g	C2070	37.386007	0.967179
gsjvix	f0049152	pg	-	462.374084	191.386410
gsjvix	f0049152	pg_s	-	185.719775	76.873342
gsjvix	f0049152	pg_m	1	185.226964	76.669357
gsjvix	f0049152	pg_m	2	92.540368	38.304415
gsjvix	f0049152	pg_m	4	49.266831	20.392583
gsjvix	f0049152	pg_m	8	24.254286	10.039362
gsjvix	f0049152	pg_g	C2070	2.694715	1.115400
gsjvix	f0196608	pg_g	C2070	34.265602	0.886454
gsjvix	f0049152	np	-	461.377910	190.974072
gsjvix	f0049152	np_s	-	182.100676	75.375320
gsjvix	f0049152	np_m	1	198.760964	82.271366
gsjvix	f0049152	np_m	2	91.038664	37.682828
gsjvix	f0049152	np_m	4	45.425167	18.802437
gsjvix	f0049152	np_m	8	23.735029	9.824430
gsjvix	f0049152	np_g	C2070	2.505152	1.036935
gsjvix	f0196608	np_g	C2070	34.232158	0.885588
gsjvix	f0049152	dv	-	461.037828	190.833305
gsjvix	f0049152	dv_s	-	184.738263	76.467073
gsjvix	f0049152	dv_m	1	184.657898	76.433808
gsjvix	f0049152	dv_m	2	93.101940	38.536862
gsjvix	f0049152	dv_m	4	47.249561	19.557592
gsjvix	f0049152	dv_m	8	24.254719	10.039541
gsjvix	f0049152	dv_g	C2070	2.407201	0.996391
gsjvix	f0196608	dv_g	C2070	32.648322	0.844614

GPU	object	NVCC	DEFS	sec.	nano sec.
(gsjvix)	pxyz_m_8			472.361030	12.220014
C2070	**pxyz**	**-arch=sm_20**		**37.803861**	**0.977989**
C2070	pxyz	-arch=sm_20	-use_fast_math	14.768863	0.382072
C2070	pxyz	-arch=sm_20	-DDOUBLE	102.313178	2.646849
C2070	pxyz	-arch=sm_20	-DDOUBLE -use_fast_math	102.383259	2.648662
C2070	pxyz	-arch=sm_20	-DKEPLER	47.285873	1.223289
C2070	pxyz	-arch=sm_20	-DKEPLER -use_fast_math	19.111344	0.494412
C2070	pxyz	-arch=sm_20	-DKEPLER -DDOUBLE	103.400341	2.674974
C2070	pxyz	-arch=sm_20	-DKEPLER -DDOUBLE -use_fast_math	103.389512	2.674694
(gsjvix)	pg_m_8			404.254103	10.458083
C2070	**pg**	**-arch=sm_20**		**34.587521**	**0.894782**
C2070	pg	-arch=sm_20	-use_fast_math	14.395168	0.372404
C2070	pg	-arch=sm_20	-DDOUBLE	99.174134	2.565642
C2070	pg	-arch=sm_20	-DDOUBLE -use_fast_math	99.160657	2.565293
C2070	pg	-arch=sm_20	-DKEPLER	43.889780	1.135432
C2070	pg	-arch=sm_20	-DKEPLER -use_fast_math	18.808437	0.486576
C2070	pg	-arch=sm_20	-DKEPLER -DDOUBLE	101.231639	2.618870
C2070	pg	-arch=sm_20	-DKEPLER -DDOUBLE -use_fast_math	101.210191	2.618315

添付ファイル "ns.pdf "（4 / 4）

host	model	run	threads	sec.	nano sec.
gsjkic	f0049152	pxyz	-	863.324678	357.348339
gsjkic	f0049152	pxyz_s	-	557.441794	230.736945
gsjkic	f0049152	pxyz_m	1	562.337855	232.763528
gsjkic	f0049152	pxyz_m	2	281.375571	116.467298
gsjkic	f0049152	pxyz_m	4	281.316934	116.443027
gsjkic	f0049152	pxyz_m	8	281.229684	116.406913
gsjkic	f0049152	pxyz_g	C1060	3.100790	1.283483
gsjkic	f0196608	pxyz_g	C1060	36.916614	0.955035
gsjkic	f0049152	pg	-	853.333844	353.212921
gsjkic	f0049152	pg_s	-	523.539742	216.704169
gsjkic	f0049152	pg_m	1	523.890872	216.849509
gsjkic	f0049152	pg_m	2	262.342495	108.589106
gsjkic	f0049152	pg_m	4	262.337049	108.586852
gsjkic	f0049152	pg_m	8	262.305610	108.573838
gsjkic	f0049152	pg_g	C1060	2.906979	1.203260
gsjkic	f0196608	pg_g	C1060	34.168412	0.883939
gsjkic	f0049152	np	-	852.884612	353.026975
gsjkic	f0049152	np_s	-	519.194607	214.905626
gsjkic	f0049152	np_m	1	519.788449	215.151430
gsjkic	f0049152	np_m	2	260.221720	107.711272
gsjkic	f0049152	np_m	4	260.328830	107.755607
gsjkic	f0049152	np_m	8	260.147047	107.680363
gsjkic	f0049152	np_g	C1060	2.579625	1.067761
gsjkic	f0196608	np_g	C1060	32.634672	0.844261
gsjkic	f0049152	dv	-	877.174596	363.081113
gsjkic	f0049152	dv_s	-	518.669955	214.688461
gsjkic	f0049152	dv_m	1	518.836269	214.757302
gsjkic	f0049152	dv_m	2	259.648293	107.473919
gsjkic	f0049152	dv_m	4	259.559340	107.437099
gsjkic	f0049152	dv_m	8	259.497179	107.411369
gsjkic	f0049152	dv_g	C1060	2.628079	1.087817
gsjkic	f0196608	dv_g	C1060	32.977599	0.853133

GPU	object	NVCC	DEFS	sec.	nano sec.
(gsjkic)	pxyz_m_8			4496.502284	116.324836
C1060	**pxyz**	**-arch=sm_13**		**37.036232**	**0.958130**
C1060	pxyz	-arch=sm_13	-use_fast_math	24.450492	0.632536
C1060	pxyz	-arch=sm_13	-DDOUBLE	479.267073	12.398673
C1060	pxyz	-arch=sm_13	-DDOUBLE -use_fast_math	479.194047	12.396784
C1060	pxyz	-arch=sm_13	-DKEPLER	33.880647	0.876495
C1060	pxyz	-arch=sm_13	-DKEPLER -use_fast_math	24.986384	0.646400
C1060	pxyz	-arch=sm_13	-DKEPLER -DDOUBLE	512.005396	13.245616
C1060	pxyz	-arch=sm_13	-DKEPLER -DDOUBLE -use_fast_math	511.942157	13.243980
(gsjkic)	pg_m_8			4190.128045	108.398912
C1060	**pg**	**-arch=sm_13**		**34.709064**	**0.897926**
C1060	pg	-arch=sm_13	-use_fast_math	22.344690	0.578059
C1060	pg	-arch=sm_13	-DDOUBLE	469.953271	12.157725
C1060	pg	-arch=sm_13	-DDOUBLE -use_fast_math	469.953302	12.157726
C1060	pg	-arch=sm_13	-DKEPLER	32.627559	0.844077
C1060	pg	-arch=sm_13	-DKEPLER -use_fast_math	23.653859	0.611927
C1060	pg	-arch=sm_13	-DKEPLER -DDOUBLE	503.466376	13.024711
C1060	pg	-arch=sm_13	-DKEPLER -DDOUBLE -use_fast_math	503.539273	13.026597

GPU	object	NVCC	DEFS	sec.	dP min.	dP max.	dX min.	dX max.	dY min.	dY max.	dZ min.	dZ max.
(gsjgix)	pxyz_m_8			288.427092	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
K20	pxyz	-arch=sm_35		27.296900	-3.30e-06	3.20e-06	-2.31e-05	2.20e-05	-2.57e-05	2.38e-05	-2.94e-05	3.03e-05
K20	pxyz	-arch=sm_35	-use_fast_math	9.477799	-2.50e-06	3.80e-06	-1.91e-05	2.32e-05	-2.50e-05	2.91e-05	-2.85e-05	2.77e-05
K20	pxyz	-arch=sm_35	-DDOUBLE	90.487250	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	pxyz	-arch=sm_35	-DDOUBLE -use_fast_math	90.322671	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	**pxyz**	**-arch=sm_35**	**-DKEPLER**	**20.543892**	**-3.04e-06**	**3.10e-06**	**-2.23e-05**	**2.22e-05**	**-2.48e-05**	**2.41e-05**	**-3.30e-05**	**3.11e-05**
K20	pxyz	-arch=sm_35	-DKEPLER -use_fast_math	7.814951	-2.53e-06	3.70e-06	-1.83e-05	2.46e-05	-2.59e-05	2.83e-05	-2.92e-05	2.97e-05
K20	pxyz	-arch=sm_35	-DKEPLER -DDOUBLE	53.798292	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	pxyz	-arch=sm_35	-DKEPLER -DDOUBLE -use_fast_math	53.757530	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
(gsjgix)	pg_m_8			245.675027	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
K20	pg	-arch=sm_35		26.653264	-3.50e-06	3.30e-06	-2.29e-05	2.52e-05	-2.84e-05	2.36e-05	-2.94e-05	3.06e-05
K20	pg	-arch=sm_35	-use_fast_math	9.187943	-4.30e-06	2.20e-06	-2.15e-05	2.19e-05	-2.62e-05	2.63e-05	-3.01e-05	3.19e-05
K20	pg	-arch=sm_35	-DDOUBLE	88.880779	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	pg	-arch=sm_35	-DDOUBLE -use_fast_math	88.801726	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	**pg**	**-arch=sm_35**	**-DKEPLER**	**20.259451**	**-3.10e-06**	**2.98e-06**	**-2.08e-05**	**2.38e-05**	**-2.44e-05**	**2.60e-05**	**-2.88e-05**	**2.76e-05**
K20	pg	-arch=sm_35	-DKEPLER -use_fast_math	7.529857	-4.30e-06	2.50e-06	-2.14e-05	2.26e-05	-2.59e-05	2.95e-05	-3.07e-05	3.12e-05
K20	pg	-arch=sm_35	-DKEPLER -DDOUBLE	53.163159	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
K20	pg	-arch=sm_35	-DKEPLER -DDOUBLE -use_fast_math	53.229599	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07

GPU	object	NVCC	DEFS	sec.	dP min.	dP max.	dX min.	dX max.	dY min.	dY max.	dZ min.	dZ max.
(sp8ct)	pxyz_m_8			1217.761176	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
GTX_680	pxyz	-arch=sm_30		33.618992	-3.30e-06	3.20e-06	-2.31e-05	2.20e-05	-2.57e-05	2.38e-05	-2.94e-05	3.03e-05
GTX_680	pxyz	-arch=sm_30	-use_fast_math	10.275388	-2.50e-06	3.80e-06	-1.91e-05	2.32e-05	-2.50e-05	2.91e-05	-2.85e-05	2.77e-05
GTX_680	pxyz	-arch=sm_30	-DDOUBLE	206.491671	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	pxyz	-arch=sm_30	-DDOUBLE -use_fast_math	207.038048	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	**pxyz**	**-arch=sm_30**	**-DKEPLER**	**26.039547**	**-3.04e-06**	**3.10e-06**	**-2.23e-05**	**2.22e-05**	**-2.48e-05**	**2.41e-05**	**-3.30e-05**	**3.11e-05**
GTX_680	pxyz	-arch=sm_30	-DKEPLER -use_fast_math	9.364018	-2.53e-06	3.70e-06	-1.83e-05	2.46e-05	-2.59e-05	2.83e-05	-2.92e-05	2.97e-05
GTX_680	pxyz	-arch=sm_30	-DKEPLER -DDOUBLE	231.582772	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	pxyz	-arch=sm_30	-DKEPLER -DDOUBLE -use_fast_math	231.560142	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
(sp8ct)	pg_m_8			1169.738297	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
GTX_680	pg	-arch=sm_30		32.383118	-3.50e-06	3.30e-06	-2.29e-05	2.52e-05	-2.84e-05	2.36e-05	-2.94e-05	3.06e-05
GTX_680	pg	-arch=sm_30	-use_fast_math	10.004385	-4.30e-06	2.20e-06	-2.15e-05	2.19e-05	-2.62e-05	2.63e-05	-3.01e-05	3.19e-05
GTX_680	pg	-arch=sm_30	-DDOUBLE	202.896135	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	pg	-arch=sm_30	-DDOUBLE -use_fast_math	203.362629	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	**pg**	**-arch=sm_30**	**-DKEPLER**	**26.075013**	**-3.10e-06**	**2.98e-06**	**-2.08e-05**	**2.38e-05**	**-2.44e-05**	**2.60e-05**	**-2.88e-05**	**2.76e-05**
GTX_680	pg	-arch=sm_30	-DKEPLER -use_fast_math	10.302571	-4.30e-06	2.50e-06	-2.14e-05	2.26e-05	-2.59e-05	2.95e-05	-3.07e-05	3.12e-05
GTX_680	pg	-arch=sm_30	-DKEPLER -DDOUBLE	225.427023	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
GTX_680	pg	-arch=sm_30	-DKEPLER -DDOUBLE -use_fast_math	225.455733	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07

GPU	object	NVCC	DEFS	sec.	dP min.	dP max.	dX min.	dX max.	dY min.	dY max.	dZ min.	dZ max.
(gsjvix)	pxyz_m_8			472.361030	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
C2070	**pxyz**	**-arch=sm_20**		**37.803861**	**-3.30e-06**	**3.20e-06**	**-2.31e-05**	**2.20e-05**	**-2.57e-05**	**2.38e-05**	**-2.94e-05**	**3.03e-05**
C2070	pxyz	-arch=sm_20	-use_fast_math	14.768863	-2.50e-06	3.80e-06	-1.91e-05	2.32e-05	-2.50e-05	2.91e-05	-2.85e-05	2.77e-05
C2070	pxyz	-arch=sm_20	-DDOUBLE	102.313178	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pxyz	-arch=sm_20	-DDOUBLE -use_fast_math	102.383259	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pxyz	-arch=sm_20	-DKEPLER	47.285873	-3.04e-06	3.10e-06	-2.23e-05	2.22e-05	-2.48e-05	2.41e-05	-3.30e-05	3.11e-05
C2070	pxyz	-arch=sm_20	-DKEPLER -use_fast_math	19.111344	-2.53e-06	3.70e-06	-1.83e-05	2.46e-05	-2.59e-05	2.83e-05	-2.92e-05	2.97e-05
C2070	pxyz	-arch=sm_20	-DKEPLER -DDOUBLE	103.400341	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pxyz	-arch=sm_20	-DKEPLER -DDOUBLE -use_fast_math	103.389512	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
(gsjvix)	pg_m_8			404.254103	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
C2070	**pg**	**-arch=sm_20**		**34.587521**	**-3.50e-06**	**3.30e-06**	**-2.29e-05**	**2.52e-05**	**-2.84e-05**	**2.36e-05**	**-2.94e-05**	**3.06e-05**
C2070	pg	-arch=sm_20	-use_fast_math	14.395168	-4.30e-06	2.20e-06	-2.15e-05	2.19e-05	-2.62e-05	2.63e-05	-3.01e-05	3.19e-05
C2070	pg	-arch=sm_20	-DDOUBLE	99.174134	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pg	-arch=sm_20	-DDOUBLE -use_fast_math	99.160657	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pg	-arch=sm_20	-DKEPLER	43.889780	-3.10e-06	2.98e-06	-2.08e-05	2.38e-05	-2.44e-05	2.60e-05	-2.88e-05	2.76e-05
C2070	pg	-arch=sm_20	-DKEPLER -use_fast_math	18.808437	-4.30e-06	2.50e-06	-2.14e-05	2.26e-05	-2.59e-05	2.95e-05	-3.07e-05	3.12e-05
C2070	pg	-arch=sm_20	-DKEPLER -DDOUBLE	101.231639	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C2070	pg	-arch=sm_20	-DKEPLER -DDOUBLE -use_fast_math	101.210191	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07

GPU	object	NVCC	DEFS	sec.	dP min.	dP max.	dX min.	dX max.	dY min.	dY max.	dZ min.	dZ max.
(gsjkic)	pxyz_m_8			4496.502284	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
C1060	**pxyz**	**-arch=sm_13**		**37.036232**	**-7.40e-06**	**7.71e-06**	**-2.37e-05**	**2.47e-05**	**-4.11e-05**	**3.29e-05**	**-3.72e-05**	**4.02e-05**
C1060	pxyz	-arch=sm_13	-use_fast_math	24.450492	-6.60e-06	7.68e-06	-2.41e-05	2.27e-05	-3.95e-05	3.28e-05	-3.94e-05	4.08e-05
C1060	pxyz	-arch=sm_13	-DDOUBLE	479.267073	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pxyz	-arch=sm_13	-DDOUBLE -use_fast_math	479.194047	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pxyz	-arch=sm_13	-DKEPLER	33.880647	-6.70e-06	7.22e-06	-2.18e-05	2.14e-05	-3.59e-05	2.77e-05	-3.76e-05	3.99e-05
C1060	pxyz	-arch=sm_13	-DKEPLER -use_fast_math	24.986384	-6.00e-06	7.56e-06	-2.22e-05	2.02e-05	-3.86e-05	2.83e-05	-3.68e-05	4.08e-05
C1060	pxyz	-arch=sm_13	-DKEPLER -DDOUBLE	512.005396	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pxyz	-arch=sm_13	-DKEPLER -DDOUBLE -use_fast_math	511.942157	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
(gsjkic)	pg_m_8			4190.128045	(7.456264e-02)	(1.195008e-01)	(-5.137567e-01)	(5.959906e-01)	(-6.213155e-01)	(6.167402e-01)	(-6.425172e-01)	(6.339563e-01)
C1060	**pg**	**-arch=sm_13**		**34.709064**	**-6.70e-06**	**7.65e-06**	**-2.33e-05**	**2.20e-05**	**-3.29e-05**	**2.99e-05**	**-3.79e-05**	**4.47e-05**
C1060	pg	-arch=sm_13	-use_fast_math	22.344690	-7.49e-06	6.95e-06	-2.37e-05	2.12e-05	-3.28e-05	3.03e-05	-3.60e-05	4.24e-05
C1060	pg	-arch=sm_13	-DDOUBLE	469.953271	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pg	-arch=sm_13	-DDOUBLE -use_fast_math	469.953302	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pg	-arch=sm_13	-DKEPLER	32.627559	-6.70e-06	7.01e-06	-2.42e-05	2.97e-05	-3.34e-05	2.64e-05	-3.70e-05	4.57e-05
C1060	pg	-arch=sm_13	-DKEPLER -use_fast_math	23.653859	-7.32e-06	6.68e-06	-2.55e-05	2.55e-05	-3.63e-05	2.70e-05	-3.57e-05	4.53e-05
C1060	pg	-arch=sm_13	-DKEPLER -DDOUBLE	503.466376	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07
C1060	pg	-arch=sm_13	-DKEPLER -DDOUBLE -use_fast_math	503.539273	-1.00e-07	1.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07	-2.00e-07	2.00e-07

Date: Tue, 06 Aug 2013 15:33:27 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Kentaro Uesugi, "NAKASHIMA, Yoshito", Satoshi Okumura, Masayuki Uesugi, Michihiko Nakamura, Takashi Matsushima, Naru HIRATA, 道上達広
Subject: gravitation_STL_5

みなさま、

GSJ/AIST のなかのです。これまでにお送りした「STL データとして与えられた物体像に対してその内部の密度が一様と仮定した場合の重力ポテンシャルや重力加速度ベクトルの値を計算するプログラム」に関する一連の E-mails の最後を飾るものとして、火星の衛星 Phobos の表面の正規化した重力ポテンシャル（NP）と鉛直線偏差（DV）の空間分布を形状モデルを使って計算し、その結果を以下のようなカラーの鳥瞰アニメーションとして描画する具体的な手順を紹介します。

NP の値に応じて表面を色づけした Phobos の鳥瞰アニメーション
http://www-bl20.spring8.or.jp/~sp8ct/tmp/phobos/gaskell_pg_np.mpg

DV の値に応じて表面を色付けした Phobos の鳥瞰アニメーション
http://www-bl20.spring8.or.jp/~sp8ct/tmp/phobos/gaskell_pg_dv.mpg

(0) 前準備

ここで紹介する処理は UNIX（Linux や MacOS X）と Windows のどちらでも実行できます。ただし、そのためには、下記のプログラムすべてを処理に使う計算機にあらかじめインストールしておく必要があります。

STL データとして与えられた物体像の重力値を計算するプログラム群
以前の説明に従って書庫ファイル
http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.taz or
http://www-bl20.spring8.or.jp/~sp8ct/tmp/gravity.zip
のいずれかを展開し、UNIX の場合はディレクトリ gravity/src/ や gravity/cuda/ の下のソースをコンパイルして得たプログラムをディレクトリ gravity/bin/ の下にインストールして下さい。また、Windows の場合はディレクトリ exe/ の下にある実行ファイルをご利用下さい。いずれの場合も bin/ や exe/ の絶対パス名を実行パスに登録するなどして、これらのプログラムを任意の場所から起動できるよう設定して下さい（これは以下に記したすべてのプログラムのインストールに関しても同様です）。

文字列処理用のインタプリタ awk（もしくは gawk）
これは UNIX の標準的なコマンドなので、Linux や MacOS X ならインストール済みのはずです。また、Windows の場合もインターネット上に gawk の実行ファイルがあるので、それを適宜インストールして下さい。

STL データ処理用のプログラム群（STL）

ファイル形式と同じ呼称で紛らわしいですが、ぼくはこのプログラム群を「STL（シリーズ）」と呼んでいます。これは公式には

https://www.gsj.jp/researches/openfile/openfile2006/openfile0448.html

で紹介されていますが、その説明書は以下のものの方が新しいです。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.pdf

この文書の p.4 にプログラム群のインストール法が記されています。ただし、それを行う際には下記の最新の書庫ファイルをお使い下さい。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.tar　or

http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.zip

STL データから物体像の鳥瞰図を描画する新版のプログラム群（HVD）

プログラム群「HVD」の説明書はまだ書いていません。その概略は

http://www-bl20.spring8.or.jp/~sp8ct/tmp/hvd_120301.pdf

の p.11～31 に記した通りです。HVD のプログラムのインストール法は STL シリーズのものと概ね同じです。まず、書庫ファイル

http://www-bl20.spring8.or.jp/~sp8ct/tmp/hvd.taz　or

http://www-bl20.spring8.or.jp/~sp8ct/tmp/hvd.zip

をダウンロードして下さい。Windows の場合、これらを展開したディレクトリ hvd/exe/ に実行ファイルが入っているので、それを実行パスに登録するなどして下さい。また、UNIX の場合はディレクトリ hvd/src/ に移動して端末から以下のように入力すれば OK です。

64 bit CPU 用の UNIX の場合、

```
make  -f  Makefile.64            ← プログラムを gcc でコンパイル
make  -f  Makefile.64  install
```

32 bit CPU 用の UNIX

```
make  -f  Makefile.32            ← プログラムを gcc でコンパイル
make  -f  Makefile.32  install
```

これにより hvd/bin/ にプログラム群がコピーされるはずなので、そのディレクトリの絶対パス名を実行パスに登録するなどして下さい。

なお、STL データから計算した NP や DV の値に応じた物体像表面の色づけには先の書庫ファイル gravity.zip などに入っている gravity/etc/cm_stl.txt に書き込まれている色のデータ（color map）を使います。また、後述する処理で使用する複数の awk スクリプトもそれと同じディレクトリ gravity/etc/ の下に入っています。これらのファイルの位置の指定を簡便にするため、ここで紹介する処理はすべてディレクトリ gravity/phobos/ の下で実行します。それゆえ、このディレクトリ gravity/phobos/ を各自で事前に作成しておいて下さい。

(1) Phobos の形状モデル

Phobos の形状モデルには複数のものがありますが、今回は JPL の Gaskell が Viking Orbiter 1 の光学画像をもとにして作ったデータファイルを使います。

Gaskell が作った Phobos の形状モデルの概要
http://nssdc.gsfc.nasa.gov/nmc/datasetDisplay.do?id=PSSB-00448

その複数種類のデータファイルが置いてある FTP サイト
http://sbn.psi.edu/pds/asteroid/VO1_SA_VISA_VISB_5_PHOBOSSHAPE_V1_0/data/

この FTP サイトに置いてあるものはすべてテキストファイルで、"*.tab" にはデータが、"*.lbl" にはその概略の説明が記されています。また、"*.tab" には２種類の形式のものがありますが、ここでは以下の４個の「VER 形式」のデータファイルのいずれかをディレクトリ gravity/phobos/ の下にダウンロードして下さい。

データファイルの名前	形状モデルの表面を構成する三角形の個数
phobos_ver64q.tab	49152 = 12 × 64^2
phobos_ver128q.tab	196608 = 12 × 128^2
phobos_ver256q.tab	786432 = 12 × 256^2
phobos_ver512q.tab	3145728 = 12 × 512^2

(2) VER 形式データファイルの STL 形式への変換

Phobos の形状モデルの VER 形式のデータファイルを STL データ処理用のプログラムで取り扱える VF 形式のものに変換する awk スクリプトを書きました。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/phobos/ver2vf.awk

これをディレクトリ gravity/phobos/ の下にダウンロードし、端末（Windows ならコマンドプロンプト）を開いてそこに移動します。その後、以下をキー入力すれば VER 形式のデータファイルを STL 形式のものに変換できます（以下に記したファイル名 "*.tab" と gaskell.stl は適宜書き換えて下さい）。

Linux や MacOS の場合

```
gawk -f ver2vf.awk phobos_ver64q.tab | vf2zcp - | zcp_stl - gaskell.stl
```

Windows の場合（作業用のファイル tmp.zcp を一時的に使っています）

```
gawk -f ver2vf.awk phobos_ver64q.tab | vf2zcp tmp.zcp
zcp_stl tmp.zcp gaskell.stl
rm tmp.zcp
```

(3) 生の重力ポテンシャルと重力加速度ベクトルの値（PG）の計算

さて、ここからが本番です。以前にも説明したように STL データから直接 NP と DV のそれぞれを個別に計算することも可能ですが、ここではそれらの両方を無駄なく取得するために STL データからまず PG を計算し、そのデータをテキストファイル pg.txt に書き込みます。そのためには以前に紹介した 4 個のプログラム "stl_pg*" のいずれかを以下のようにして実行します。

旧版のプログラム stl_pg（低速）を使う場合

```
stl_pg   gaskell.stl   > pg.txt
```

新版のプログラム stl_pg_s（高速）を使う場合

```
stl_pg_s   gaskell.stl   > pg.txt
```

stl_pg_m を使ったマルチスレッド処理による PG の計算

スレッド数の設定（使用するスレッドの数は適宜書き換えて下さい）

```
setenv   THREADS   4                          ← csh や tcsh を使用している場合
THREADS=4   &&   export   THREADS             ← bsh や bash
set   THREADS=4                               ← Windows のコマンドプロンプト
```

stl_pg_m の実行

```
stl_pg_m   gaskell.stl   > pg.txt
```

stl_pg_g を使って CUDA GPU で PG を計算

CUDA GPU の選択（指定を省略すると装置番号 0 の GPU を使います）

```
setenv   CUDA_GPU   1                         ← csh or tcsh
CUDA_GPU=1   &&   export   CUDA_GPU           ← bsh or bash
set   CUDA_GPU=1                              ← Windows
```

stl_pg_g の実行

```
stl_pg_g   gaskell.stl   > pg.txt
```

以前にも紹介したように STL データの三角形の個数 M が多いと PG の計算には長い（M^2 に比例した）処理時間を要します。例えば、今回紹介した Phobos の形状モデルのうちで M が最多（M = 3145728）の phobos_ver512q.tab のデータに対しては、 ぼくが持っている最速の CUDA GPU（K20）でも PG の計算に 1 時間半近くかかります。また、CPU なら 8 スレッドで実行してもこれには丸 1 日以上かかかるはずです。それは大変なので、K20 で計算した phobos_ver512q.tab に対する PG のデータを書き込んだテキストファイルを以下のようにアップロードしておきましたので、PG の計算処理を省きたい場合にはこちらをお使い下さい。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/phobos/gaskell_pg.txt.gz

(4) NP と DV の値に応じて色付けした STL データファイルの作成

まず、作成した PG のデータから Phobos の形状モデルの表面に出現する NP や DV の最小値と最大値を算出し表示してみます。

形状モデルの重心における重力ポテンシャルと NP の最小値・最大値の表示

```
gawk  -f  ../etc/pg2np_0.awk  pg.txt
```

DV の最小値と最大値の表示

```
gawk  -f  ../etc/pg2dv_0.awk  pg.txt
```

これらの出力を参考に、色付けする NP や DV の下限と上限の値を決めて下さい。以下ではテキストファイル../etc/cm_stl.txt に記されている青、シアン、緑、黄、赤、マジェンタの順に変化する 32768 諧調の色データ（color map）を指定した値域の NP や DV の値に対応付け、そうして決めた色を物体像表面の三角形それぞれに塗布した新しい STL データファイル np.stl と dv.stl を作ります。

物体像の表面に出現する NP や DV の最小・最大値を色付けの下限・上限の値とした STL データファイルを作りたい場合は以下のように入力します。

NP で色付けした STL データファイル np.stl の作成

```
gawk  -f  ../etc/pg2np_1.awk  pg.txt  |  t2stl  ../etc/cm_stl.txt  np.stl
```

DV

```
gawk  -f  ../etc/pg2dv_1.awk  pg.txt  |  t2stl  ../etc/cm_stl.txt  dv.stl
```

また、NP や DV の値域を陽に指定する時は以下のように入力します（以下では、ぼくが指定した NP や DV の下限と上限の値を埋め込んであります)。

NP（= 0.580209 ～ 0.770075 の値域に color map を対応付ける）

```
gawk  -v  min=0.580209  （改行しない）
      -v  max=0.770075  （改行しない）
      -f  ../etc/pg2np_2.awk  pg.txt  |  t2stl  ../etc/cm_stl.txt  np.stl
```

DV（= 0.004621° ～ 56.378045°）

```
gawk  -v  min=0.004621  （改行しない）
      -v  max=56.378045  （改行しない）
      -f  ../etc/pg2dv_2.awk  pg.txt  |  t2stl  ../etc/cm_stl.txt  dv.stl
```

(5) 鳥瞰アニメーションの描画

物体像の鳥瞰図（鳥瞰アニメーション）を描く新版のプログラム群 HVD では、描画に使うパラメータのほとんどを環境変数に設定して指定します。以下で利用する STL データとして与えられた像の描画プログラム stl_bev_4 では、その実行に最小限必要なパラメータは鳥瞰図のフレーム（正方形）の横・縦画素

数だけです。そこで、その値（ここでは 800 としました）を stl_bev_4 の実行の直前に環境変数 BEV_SIZE に以下のようにして設定しておきます。

csh もしくは tcsh を使っている場合

```
setenv  BEV_SIZE  800
```

bsh もしくは bash

```
BEV_SIZE=800  && export  BEV_SIZE
```

Windows のコマンドプロンプト

```
set  BEV_SIZE=800
```

HVD のプログラム群は、以下の２通りの形式のいずれかで記述した鳥瞰図の視線方向（経度と緯度）を指定している行を標準入力（通常は端末入力）から順次読み取ります。

[1] 視線方向の経度λと緯度φ（単位は度）の２個の数値
[2] λの初期値、φの初期値、λの増分、φの増分と視線方向の総数の５個の数値

ここでは記法 [2] を用いて、φが常に 30°、λが 120°から－2°刻みで変化する（つまり、Phobos が地球と同方向に自転する）ような 180 通りの視線方向を指定してみました。

```
echo  120  30  -2  0  180  |  stl_bev_4  np.stl  np.gif
echo  120  30  -2  0  180  |  stl_bev_4  dv.stl  dv.gif
```

以上のようにして作成したアニメーション GIF のファイル np.gif と dv.gif の各フレームの周囲には広い「余白」の領域があります。これは STL シリーズのプログラム gif_trim や gif_area を使って削除することができます。その詳細は前記の文書 http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.pdf の p.21～22 をご覧下さい。ぼくは以下のような入力でそれを取り除きました（ここでは np.gif に対する入力例だけを示しましたが、dv.gif に対してもまったく同じです）。

UNIX の場合

```
gif_trim  np.gif  `gif_area  np.gif  0  0  0`  255  255  255  np_trimmed.gif
```

Windows の場合（エレガントな方法ではないです）

```
gif_area  np.gif  0  0  0
```

→ これで表示される４個の数値を x1、y1、x2、y2 とする。

```
gif_trim  np.gif  x1  y1  x2  y2  255  255  255  np_trimmed.gif
```

この後、各フレームの下部に「カラーバー」を貼り付け、GIF から MPEG 形式に変換したものがこの E-mail の最初に紹介した２個の動画ファイル “*.mpg” です。

またしても長い E-mail になりました。とりあえず以上です。