Date: Sat, 01 Jun 2013 12:47:03 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Satoshi Okumura,
Cc: Kentaro Uesugi, Tooru Matsumoto, Akira Shimada, MATSUNO Junya, Kadono Kadono, Takashi Matsushima, Michihiko Nakamura
Subject: ovoid_fitting+BoundingBox_again

つちやまさま、

GSJ/AIST のなかのです。連合大会の時に東北大学の奥村くんからぼくが書いた楕円体近似の計算機プログラムで使っている楕円体の軸方向の「角度」について尋ねられました。この楕円体近似では当初使っていた計算式に間違いがあったり、計算機プログラムを大幅に改造したりしたので、それらの詳細やそもそもの手法の弱点などについてきちんと説明しておく必要性を感じました。さらに、連合大会の直前につちやまさんの要望で開発した粒径算出法（楕円体近似とは別のつちやまさんが「ノギス法」と呼んでいる手法）を奥村くんにも紹介した方が良いと考えました。以下ではこれらについて少し詳しく書いてみます。

(1) 楕円体近似の手法について
(2) 楕円体近似用の計算機プログラム
(3) 「ノギス法」による物体像の直交する３方向の「径」の算出法
(4) 「ノギス法」用の計算機プログラム
(5) 「ノギス法」の実行例

(1) 楕円体近似の手法について

後で詳しく説明するように、ぼくが書いた「画像上の物体像の楕円体近似」用の計算機プログラムには新旧２種類のもの（sliceOF と si_of）があります。これらを使う場合のノミナルな引用文献は以下の池田くんの論文（の付録）です：

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

ただし、この付録に記されている計算式は間違っています。その修正版が以下の文書です（池田くんは以下の文書の旧版を英訳して論文に載せた）：

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

なお、ぼくが書いた新旧２種類の楕円体近似用の計算機プログラムはとちらも、この文書に記した修正後の計算式を使っています。

以上の計算式の間違いに関することや、ぼくが考案した楕円体近似の手法の弱点については以下の文書（E-mail）で詳しく論じていますので、ご覧下さい：

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

これに書いたように、ぼくの楕円体近似はそもそもは画像上の物体像を楕円体で近似する手法ではなく、物体像を構成している画素群の配列の方向（～ 物体像の軸方向）を推定する手法です。物体像の「２次モーメント」が物理的（？）な意味を持つと仮定して、それを使って楕円体の軸半径を算出しています。しかし、以前に京都大学の山路 敦さんからこの仮定に文句をつけられました。確かに、物体像の２次モーメントは特別扱いするほど重要な物理量ではありません。数学的には２次モーメントは１次モーメント（～ 平均値）の次に単純な「統計量」なので、その理由で楕円体近似に２次モーメントを導入したと言う方が合理的に聞こえるかもしれません。

物体像の領域 V における２次モーメント ∫ r^2 dV を使っているので、ぼくの楕円体近似で推定した軸半径は物体像の重心からの距離 r が大きい位置にある画素群（～ 物体像の表面にある画素群）によって概ね決まります。また、物体像内部の画素群も２次モーメントに寄与しますが、それは物体像の表面の推定に伴う誤差（「雑音」）を緩和してくれます。そして、多分、物体像内部の画素群による寄与のため、楕円体とはまったく異なる形状の像（特に、外形が非常に凸凹な像）に対しては近似楕円体の軸半径は外形に対応していない値になります。それでは、どのようにすれば物体像の外形だけを反映した軸半径（もしくは粒径）を得ることができるか？ それには２種類の方法が考えられます：

[1] ３次以上の高次モーメントを使った楕円体近似
これは結局、使用した高次モーメントの「意味」について２次の場合と同様な文句をつけられる可能性が高いのでダメな感じがします。

[2] 地球の形状の解析と同様に物体像の表面のデータだけを使う手法
画像から抽出した物体像の表面のデータは前述の「雑音」に弱いので、この手法では安定した結果を得られないような気がします。なお、ぼくとしては、物体像の表面のデータは STL 形式のデータに変換した後に、地球などの形状データに対するものと同じプログラム（例えば、stl_*）を使って解析したいと考えています。

後で説明する「ノギス法」による粒径の算出は後者の手法です。

(2) 楕円体近似用の計算機プログラム

数年前から SLICE シリーズの計算機プログラム群の保守を行っていないので、ぼくとしては楕円体近似に sliceOF を使ってほしくありません。

SLICE シリーズを使う場合のノミナルな引用文献（？）
http://www-bl20.spring8.or.jp/slice/

SLICE シリーズの最新（？）のソースコードが入っている書庫ファイル
http://www-bl20.spring8.or.jp/~sp8ct/tmp/slice.tar.gz

新しい楕円体近似用の計算機プログラム si_of のソースコードや Windows 用の実行ファイルなどの一式は以下の書庫ファイルに入っています。

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

si_of のインストール法や起動法の詳細は前記の E-mail をご覧下さい。

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

si_of の起動法やテキスト出力は sliceOF のものと少しだけ違います。

コマンドラインからの起動法

sliceOF　directory　nameFile　{Ux　Uy　Uz}

si_of　directory　nameFile　Ux　Uy　Uz

sliceOF で省略できた画素の辺長 Ux、Uy、Uz を si_of では常に指定する必要がある。ただし、sliceOF の省略時の既定値は Ux=Uy=Uz=1。

標準出力へのテキスト出力

sliceOF（各行に以下の 15 個の値が並んでいる）

[1] 画像上の物体像の画素値 PV = 0 ～

[2] 物体像を構成する画素の個数 N

[3,4,5] 物体像の重心の画像上での座標値 Ix0, Iy0, Iz0

[6,7,8] 物体像の重心の座標値 x0=Ux*Ix0, y0=Uy*Iy0, z0=Uz*Iz0

[9,10,11] 近似楕円体の軸方向を表す度単位の角度 L, P, T

[12,13,14] Ux、Uy、Uz を考慮した楕円体の軸半径 A, B, C

[15] 近似楕円体の体積 V=A*B*C*π*4/3

si_of（各行に以下の 11 個の値が並んでいる）

[1] PV

[2] N

[3,4,5] x0, y0, z0

[6,7,8] L, P, T

[9,10,11] A, B, C

sliceOF や si_of が出力した角度の値 L、P、T を以下の式に代入すれば、近似楕円体の直交した 3 軸それぞれに平行な単位ベクトルを計算できます。

a 軸方向の単位ベクトルの成分値

ax = sin(T) * sin(P) * cos(L) + cos(T) * sin(L)

ay = sin(T) * sin(P) * sin(L) - cos(T) * cos(L)

az = - sin(T) * cos(P)

b 軸方向

bx = cos(T) * sin(P) * cos(L) - sin(T) * sin(L)

by = cos(T) * sin(P) * sin(L) + sin(T) * cos(L)

bz = - cos(T) * cos(P)

c 軸方向

cx = cos(P) * cos(L)
cy = cos(P) * sin(L)
cz = sin(P)

また、Ux=Uy=Uz=1 の場合、近似楕円体の中心と３軸の方向に対応した（a,b,c）座標系における座標値と、画像に付随した座標系の原点の位置を物体像の重心に平行移動した（x,y,z）座標系における座標値の相互の変換式は以下の通りです。

(a,b,c) → (x,y,z)

x = a * ax + b * bx + c * cx
y = a * ay + b * by + c * cy
z = a * az + b * bz + c * cz

(x,y,z) → (a,b,c)

a = ax * x + ay * y + az * z
b = bx * x + by * y + bz * z
c = cx * x + cy * y + cz * z

ついでながら、近似楕円体の表面に分布する点の（a,b,c）座標系における座標値は経度λ（= -180～180 度）と緯度φ（= -90～90 度）を使って以下の式で表現できます：

a = A * cos(φ) * cos(λ)
b = B * cos(φ) * sin(λ)
c = C * sin(φ)

(3) 「ノギス法」による物体像の直交する３方向の「径」の算出法

つちやまさんの要望に応じて、実際のモノに対してノギスを使って「径」を測定する方法（「ノギス法」）を模した計算機プログラム群を書きました。

２次元の物体の場合、ノギス法ではノギスをあてる方向を様々に変えた多数の測定を行うことにより物体の最小（もしくは最大）の幅をまず決め、その後、それと垂直な方向の幅を物体の最大（もしくは最小）の幅とします。

これは「物体像がちょうどおさまる長方形（bounding box)」のうちで幅が最小（もしくは最大）のものを探すことに相当します。

つまり、２次元画像上の物体像の径を決めるノギス法には

ノギス法_sl：短径→長径の順で径を決める（short axis first）
ノギス法_ls：長径→短径の順で径を決める（long axis fist）

の２種類があり、これらで得られる短径もしくは長径は三角形ではない２次元の像に対して同じ値になりません。

２次元画像上の物体像の径を短径→長径の順に決めるノギス法_slの具体的な計算の手続きは以下の通りです。

ノギス法_sl の手続き：

[1] 計算の手間を軽減するため、物体像の径に関与する物体像表面（２次元の場合は輪郭線）を構成している頂点の座標値だけを抽出する。これら N 個の頂点 i = 1～N の画像上の座標値をそれぞれ (xi, yi) とする。

[2] これらの頂点を画像原点を中心として回転させる。指定した角度 θ の回転に対して、頂点 i の回転後の座標値 (ai, bi) をまず計算する。その後、これらの座標値それぞれの最小値 aS と bS、最大値 aL と bL、および座標値が分布している値域の幅 A と B を調べる：

$$ai(\theta) = \cos(\theta) * xi + \sin(\theta) * yi$$
$$bi(\theta) = -\sin(\theta) * xi + \cos(\theta) * yi$$

$$aS(\theta) = \min_{i=1\sim N}\{ai(\theta)\}$$
$$bS(\theta) = \min_{i=1\sim N}\{bi(\theta)\}$$
$$aL(\theta) = \max_{i=1\sim N}\{ai(\theta)\}$$
$$bL(\theta) = \max_{i=1\sim N}\{bi(\theta)\}$$

$$A(\theta) = aL(\theta) - aS(\theta)$$
$$B(\theta) = bL(\theta) - bS(\theta)$$

[3] θ=0～180 度における $A(\theta)$のうちの最小値を物体像の短径と見なし、それと同じ角度 θ における $B(\theta)$を物体像の長径と見なす。

また、２次元物体像の径を長径→短径の順に決定するノギス法_ls は上記の[3]だけを以下のように変更した手続きで実行できます。

ノギス法_ls の手続き：

[1,2] ノギス法_sl のものと同じ

[3] θ =0～180 度における B(θ)のうちの最大値を物体像の長径と見なし、それと同じ角度 θ における A(θ)を物体像の短径と見なす。

それから、その証明は省略しますが、上の[2]と[3]を以下のように変更した手続きを使えば、長径を先に決める_ls と等価な処理を角度 θ の値を変えた検索を行わずに実行できます（これをノギス法_ls_exact と呼びます)。

ノギス法 _ls_exact の手続き：

[1] ノギス法_sl や_ls のものと同じ。

[2] ２点間の距離が最大の頂点のペア（xj,yj）と（xk,yk）を探し出す。

[3] その距離を物体像の長径 B とする。また、以下の式で計算した値 ai の値域の幅を物体像の短径 A とする：

$$ai = \{ (yj - yk) * xi + (xj - xk) * yi \} / B$$

$$A = \max_{i=1\sim N} \{ ai \} - \min_{i=1\sim N} \{ ai \}$$

２次元物体像の長径の値を高精度に決定したい場合、ノギス法_ ls では角度 θ の「刻み幅」を「小さな値」に設定してやる必要があります。一方、ノギス法_ls_exact にはその制約はなく、物体像の径の値を（数値計算の精度の範囲内で）正確に決めることができます。なお、その証明は省略しますが、短→長の順で物体像の径を正確に決めるノギス法_sl_exact は原理的に実現不可能です。

以上のような２次元画像上の物体像に対する３種類のノギス法による処理結果をこの E-mail に添付した PDF ファイル 060410g_235.pdf に示しました。黄色で塗った物体像に対する３種類のノギス法のそれぞれで得た bounding box を赤線で描きました。青線は前述の楕円体近似と同様な手法で得た物体像の近似楕円を示しており、ノギス法および楕円近似のそれぞれで得た物体像の長径 A と短径 B および短軸半径 a と長軸半径 b の値も線と同じ色で記しておきました。なお、物体像を取り囲んでいる緑色の線はその凸法です。以下の E-mail をご覧下さい。

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

060410g_235.pdf の物体像の場合、その輪郭線を構成している頂点の個数は 2119 個ですが、凸包の頂点は 33 個だけです。ノギス法の手続き[1]で物体像の凸法の頂点だけを抽出すれば、それ以降の手続き[2]や[3]の処理時間を大幅に短縮できます。そして、３次元画像上の物体像に対する処理では凸包の頂点だけを使わないとノギス法は実用にならないと思われます。

３次元画像上の物体像に対するノギス法も２次元像の場合と同様の手続きで実行できます。s、i、l を短、中（intermediate)、長径として、３次元では以下の５種類のノギス法が考えられます（中径を最初に決めることはできません)。

_lis_exact：長→中→短の順で物体像の径を正確に決める。
_sil：短→中→長の順で物体像の径を決める。
_sli：短→長→中
_lis：長→中→短
_lsi：長→短→中

また、これらを実行する具体的な手続きは以下のようになります。

ノギス法 _lis_exact の手続き：

[1] ３次元画像上の物体像の凸法の頂点を抽出して以下の計算で使う。
[2] ２点間の距離が最大の頂点のペアを探し、その距離を長径 C とする。
[3] それらの２頂点を結ぶ方向と直交している面内に投影した頂点の座標値を計算し、２次元の場合と同様な手法でその面内における長径と短径をこの順で決める。これらを３次元像の中径 B と短径 A と見なす。

ノギス法 _sil の手続き：

[1] _lis_exact のものと同じ
[2] ２個の角度（経度と緯度）を変えながら、それらで決まる軸方向に投影した頂点の座標値を計算する。半球面をカバーする多数の軸方向に投影した座標値の値域の幅のうちで最小の値を短径 A とする。
[3] その軸方向（短径 A を得た軸方向）と直交する面内に投影した頂点の座標値を計算し、２次元の場合と同様の手法でその面内における短径と長径をこの順で決める。これらを３次元像の中径 B と長径 C と見なす。

これら以外のノギス法の手続きも同様ですが、ここではその説明を省略します。

(4) 「ノギス法」用の計算機プログラム

先にも書いたようにノギス法は画像上の物体像の表面のデータだけを使うので、以下の３段階の計算機処理によってそれを実行することにしました。

[1] 画像上の物体像の凸包の STL 形式データを抽出する。
[2] それに含まれる凸包の頂点の座標値をテキスト形式に変換する。
[3] 複数のノギス法のいずれかでこれらの座標値から物体像の径を計算する。

これらのうち、[1]と[2]の処理には既存のプログラムを流用します。

２次元画像上の物体像の処理
[1,2] t_ch_line（凸包の頂点の座標値のテキストデータを出力）

３次元
[1] si_ch_zcp（PLY/ZCP 形式ファイルに凸包のデータを出力）
[2] zcp_dmp（PLY/ZCP 形式のデータをテキストデータに変換）

３次元画像上の物体像の処理では STL 形式データファイルではなく、 それよりも凸包の頂点の座標値の抽出処理に適している PLY/ZCP 形式データファイルを出力・変換するプログラムを使います。

これらのインストールや起動法などの説明を以下の PDFs に記しました（ただし、si_ch_zcp のコンパイルに関しては後に記した注意書きをご覧下さい)。

t_ch_line と si_ch_zcp
http://www-bl20.spring8.or.jp/~sp8ct/tmp/ch.pdf
zcp_dmp
http://www-bl20.spring8.or.jp/~sp8ct/tmp/stl.pdf

また、以下の２個の zip 形式書庫ファイルのそれぞれにこれらのプログラムのソースファイルなど（Windows 用の実行ファイルを含む）が入っています。

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

さて、前記[3]の複数のノギス法用の計算機プログラムなどは書庫ファイル

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

の中に入っています（これら２個の書庫ファイルの内容は同じです)。

t_of.exe、si_of.exe
楕円もしくは楕円体近似用のプログラムの Windows 用実行ファイル

t_ch_line.exe、si_ch_zcp.exe、zcp_dmp.exe
前記の[1]と[2]のプログラムの Windows 用実行ファイル

gawk.exe
後述する awk スクリプトを Windows 上で実行するためのインタプリタ GNU-awk（gawk）の Windows 用実行ファイル

Makefile
後述する C 言語ソースコードのコンパイル用の「Makefile」。GNU-C Compiler（gcc）の開発パッケージがインストールされている環境なら、端末から "make" と入力するだけで以下に記した８個の Windows 用の実行ファイル*.exe に相当する８個の実行コードをコンパイルできる。

line2ab.awk、line2ab.c、line2ab.exe
２次元画像上の物体像の頂点のデータを読み込んでノギス法_ls_exact を実行する awk スクリプト、それを翻訳した C 言語のソースコード、および、それをコンパイルした Windows 用の実行ファイル

line2ab_ls.awk、line2ab_ls.c、line2ab_ls.exe
line2ab_sl.awk、line2ab_sl.c、line2ab_sl.exe
　ノギス法_ls もしくは_sl 用のスクリプト、ソース、実行ファイル

zcp2abc.awk、zcp2abc.c、zcp2abc.exe
　３次元画像上の物体像の頂点のデータを読み込んでノギス法_lis_exact を実行する awk スクリプト、それを翻訳した C 言語のソースコード、および、それをコンパイルした Windows 用の実行ファイル
zcp2abc_sil.awk、zcp2abc_sil.c、zcp2abc_sil.exe
zcp2abc_sli.awk、zcp2abc_sli.c、zcp2abc_sli.exe
zcp2abc_lis.awk、zcp2abc_lis.c、zcp2abc_lis.exe
zcp2abc_lsi.awk、zcp2abc_lsi.c、zcp2abc_lsi.exe
　_sil、_sli、_lis、_lsi 用のスクリプト、ソース、実行ファイル

bb_awk.bat、bb_exe.bat
bb_awk.csh、bb_exe.csh
　起動パラメータとして指定した３次元２値画像上の物体像（画素値１の領域）の楕円体近似、凸包の抽出と３次元像に対する５種類のノギス法_lis_exact、_sli、_sil、_lis、_lsi をこの順で次々と実行し、これらで得た近似楕円体の短、中、長軸半径と５セットの短、中、長径の値を標準出力に書き出す Windows 用バッチファイル（*.bat）と C-shell scripts（*.csh）。ただし、"bb_awk.*" は前記の awk スクリプトで、また、"bb_exe.*" は C 言語コードをコンパイルした得た実行コードを用いてノギス法の処理を行う。

060410g/
　ぼくがノギス法のテストに使った３次元２値画像が入ったディレクトリ。SPring-8 で撮影した Stardust サンプル C2054.035.4 #10 の X 線 CT 画像を加工したもの。その詳細は以下の E-mail の p.13 をご覧下さい。

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

060410g_awk.txt、060410g_exe.txt
　起動パラメータとして「060410g」を指定して起動した bb_awk.bat と bb_exe.bat のそれぞれが標準出力に書き出したテキストデータ。

このように、書庫ファイル bb.taz や bb.zip の中にはノギス法の一連の処理を Windows のコマンドプロンプトで実行するために必要なファイルすべてが入っています。それゆえ、Windows で実行する場合は bb.taz や bb.zip を解凍・展開すればインストールの作業は終了です。

Linux や MacOS の場合は bb.taz と前述の書庫ファイル ch.zip や stl.zip を解凍・展開した後、プログラムの実行コードをコンパイルする必要があります。

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

```
unzip  -d  stl  stl.zip
cd  stl/src  &&  make install  &&  cd  ../..

wget  http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/ch.zip
unzip  ch.zip
cd  ch  &&  make  &&  cd  ..
```

ただし、「f77 がない」と言われたら以下を入力してみて下さい。

```
make  FC="gfortran  -O"  &&  cd ..
```

さらに、「gfortran がない」と言われたら GNU-FORTRAN（gfortran）をインストールし、その後「make FC="..."」を再度入力して下さい。

```
# この E-mail に添付した bb.taz をコピーする。
tar  xzf  bb.taz
cd  bb  &&  make  &&  cd ..

ln  stl/bin/zcp_dmp  ch/t_ch_line  ch/si_ch_zcp  bb
```

← 凸包に関連した３個の実行コードをディレクトリ bb にリンク

(5) 「ノギス法」の実行例

ノギス法の実行には物体像を識別済みの画像が必要です。画像から物体像の凸包を抽出する t_ch_line や si_ch_zcp は起動パラメータ rangeList による指定で多値画像上の物体像識別（画素値の２値化）を実行時に行えますが、ここでは物体像の画素に画素値１が格納されているディレクトリ bb/060410g/ の画像に対するノギス法の実行例を示します。

楕円（体）近似用の計算機プログラムとは異なり、現時点では画素値によって区別した複数の物体像に対するノギス法の一括処理はできません。これは凸包抽出プログラム t_ch_line や si_ch_zcp が複数の物体像を含む画像に対応していないためです。今後、複数の物体像それぞれの凸包を一度に抽出する計算機プログラムを書くつもりでいます。

まず、バッチファイル "bb_*.bat" や C-shell scripts "bb_*.csh" により３次元２値画像 060410g に対して楕円体近似と５種類のノギス法すべて（合計６種類の処理）を実行し、それらで得た３次元像の３個の径の値が並んでいる以下の６行のテキストデータを標準出力に書き出してみます。

１行目：近似楕円体の短、中、長軸半径
２：ノギス法_lis_exact で得た物体像の短、中、長径
３：_sli で得た物体像の短、中、長径
４：_sil で得た物体像の短、中、長径
５：_lis で得た物体像の短、中、長径
６：_lsi で得た物体像の短、中、長径

ディレクトリ bb に移動して以下を入力して下さい。

Windows

```
bb_awk.bat  060410g
bb_exe.bat  060410g  {step}
```

Linux や MacOS

```
csh  bb_awk.csh  060410g
csh  bb_exe.csh  060410g  {step}
```

ここで、bb_exe.* の２番目の起動パラメータの値 step は省略可能です。これは_lis_exact 以外のノギス法が径の検索に使う度単位の角度の「刻み幅」で、省略時の規定値は１度です。

ノギス法のどれか一つだけを実行したい時は以下のようにします。まず、２次元２値画像上の物体像（画素値１の領域）に対する入力は以下の通りです（以下の例でテキストファイル"ab*.txt"にリダイレクトしている処理結果のテキストデータについては後で説明します)。

ノギス法 _ls_exact を実行：

```
t_ch_line  060410g/235.tif  1  |  gawk  -f  line2ab.awk  > ab.txt
t_ch_line  060410g/235.tif  1  |  line2ab  -  > ab.txt
```

ノギス法 _sl を実行（_ls も同様）：

```
t_ch_line  060410g/235.tif  1  |  gawk  -f  line2ab_sl.awk > ab_sil.txt
t_ch_line  060410g/235.tif  1  |  line2ab_sl  -  {step}  > ab_sil.txt
```

ここで、line2ab_sl（および line2ab_ls）の起動パラメータ step は物体像の径の検索に使う度単位の度の「刻み幅」で、省略時の規定値は１度です。

次に３次元画像に対する場合ですが、凸包抽出に時間がかかるかもしれないので、それと凸包の頂点のデータのテキストデータへの変換をまず実行しておきます。

```
si_ch_zcp  060410g  -  1  ch.zcp
zcp_dmp  ch.zcp  >  ch.txt
```

ただし、これ以降の処理ではバイナリファイル ch.zcp は不要です。Linux や MacOS で実行する場合には以下の入力を行うことにより ch.zcp を作ることなくテキストファイル ch.txt を得ることができます。

```
si_ch_zcp  060410g  -  1  -  |  zcp_dmp  -  >  ch.txt
```

その後以下のように入力すれば、それぞれのノギス法で３次元像の径を計算することができます（"abc*.txt" に書き込んだ処理結果ついては後で説明します)。

ノギス法_lis_exact：

```
gawk  -f  zcp2abc.awk  ch.txt  > abc.txt
zcp2abc  ch.txt  > abc.txt
```

_sil（_sli、_lis、_lis も同様）:

gawk -f zcp2abc_sil.awk ch.txt > abc_sil.txt

zcp2abc_sil ch.txt {step} > abc_sil.txt

以上のような実行例で "ab*.txt" や "abc*.txt" にリダイレクトした line2ab や zcp2abc などの処理結果のテキストデータの内容は以下の通りです。

"ab*.txt" にリダイレクトした２次元像に対する処理結果

１行目：物体像の短径 A と長径 B

２：bounding box（BB）の頂点の座標値 x(aS,bS) と y(aS,bS)

３：BB の頂点の座標値 x(aL,bS) と y(aL,bS)

４：BB の頂点の座標値 x(aL,bL) と y(aL,bL)

５：BB の頂点の座標値 x(aS,bL) と y(aS,bL)

"abc*.txt" にリダイレクトした３次元像に対する処理結果

１行目：物体像の短径 A、中径 B、長径 C

２：BB の頂点の座標値 x(aS,bS,cS)、y(aS,bS,cS)、z(aS,bS,cS)

３：BB の頂点の座標値 x(aL,bS,cS)、y(aL,bS,cS)、z(aL,bS,cS)

４：BB の頂点の座標値 x(aL,bL,cS)、y(aL,bL,cS)、z(aL,bL,cS)

５：BB の頂点の座標値 x(aS,bL,cS)、y(aS,bL,cS)、z(aL,bL,cS)

６：BB の頂点の座標値 x(aS,bS,cL)、y(aS,bS,cL)、z(aS,bS,cL)

７：BB の頂点の座標値 x(aL,bS,cL)、y(aL,bS,cL)、z(aL,bS,cL)

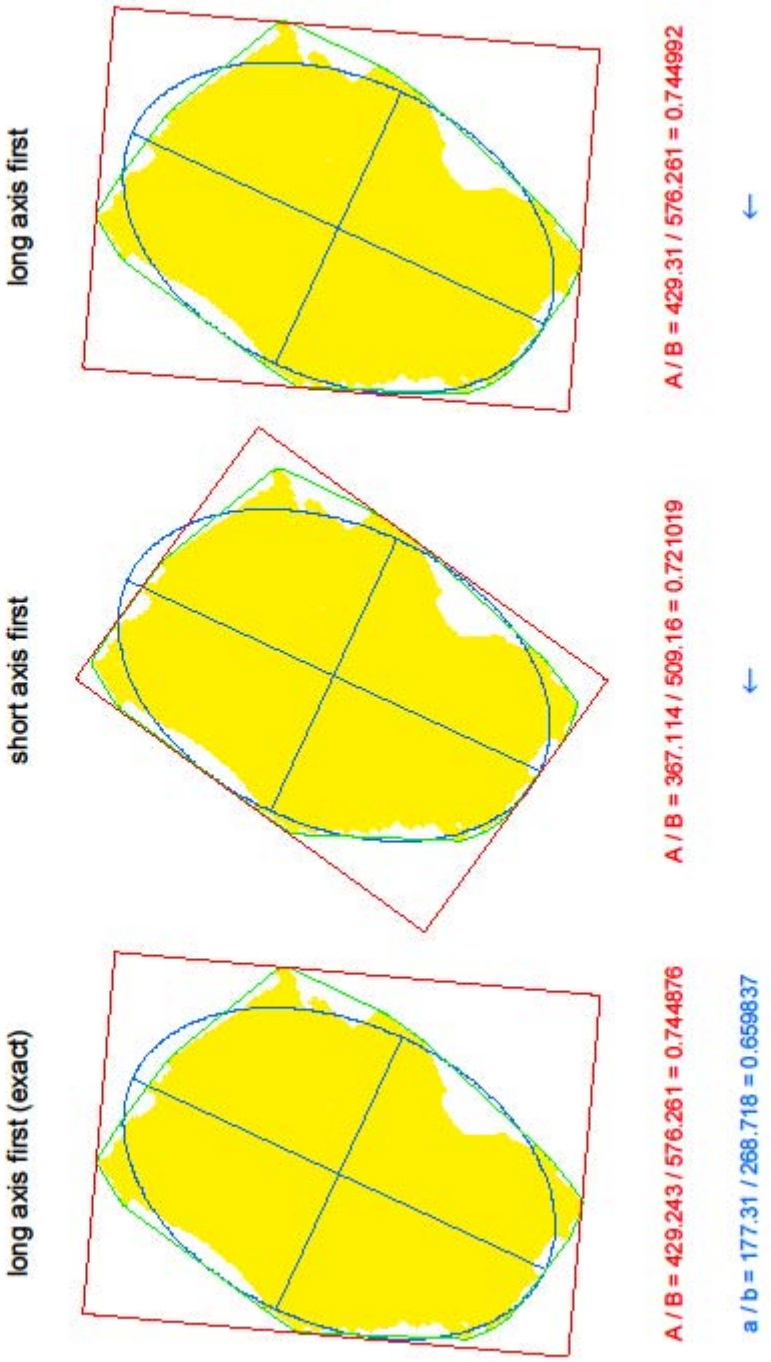
８：BB の頂点の座標値 x(aL,bL,cL)、y(aL,bL,cL)、z(aL,bL,cL)

９：BB の頂点の座標値 x(aS,bL,cL)、y(aS,bL,cL)、z(aL,bL,cL)

ここで、例えば aS と aL は物体像の領域の a 座標値（BB の短径方向に沿った座標値）の最小値と最大値です。また、例えば x(aL,bL,cL) は BB に付随した座標系における座標値が（aL,bL,cL）の点に対応している３次元画像に付随した座標系における x 座標値を意味しています。これらの頂点の座標値から BB の辺の向き（～物体像の軸方向）を計算することができます。

非常に長い E-mail になりました。とりあえず以上です。

添付ファイル “060410g_235.pdf”



Date: Wed, 03 Jul 2013 14:56:12 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Satoshi Okumura
Cc: Kentaro Uesugi, Tooru Matsumoto, Akira Shimada, MATSUNO Junya, Kadono Kadono, Takashi Matsushima, Michihiko Nakamura
Subject: new_programs_for_BBs

つちやまさま、

GSJ/AIST のなかのです。画像上の「つながっている画素群（クラスタもしくは物体像)」を識別する「ラベル」を画素値として格納した「クラスタ画像」から、指定した複数個の物体像の凸包（convex hull、CH）や、それらの物体像の複数種類の Bounding Box（BB）のすべて（計測の手順を変えた「ノギス法」で得られる複数の BBs のすべて）を計算する以下の４個のプログラムを書きました。

t_chs　TIFF　rangeList
si_chs　directory　nameFile　rangeList
　２（t_chs）もしくは３次元（si_chs）クラスタ画像上の指定したクラスタそれぞれの凸包を抽出する。これらを使えば従来の t_ch_line や si_ch_stl もしくは si_ch_zcp なら複数回の起動が必要になる複数個の凸包の抽出の処理を一度の起動だけで実行できる。

t_bbs　TIFF　rangeList
si_bbs　directory　nameFile　rangeList
　２（t_bbs）もしくは３次元（si_bbs）クラスタ画像上の指定した物体像の凸包を抽出し、それを用いてそれぞれの物体像の以下の２（t_bbs）もしくは４種類（si_bbs）の BBs のパラメータを一度に計算する。

t_bbs が個々の物体像ごとに算出する２種類の BBs
　SL（short first で径を決めた BB）
　　物体像の短径を最初に決め、その後調べた短径と直交している方向の物体像の幅を長径とした長方形
　LS（long first で径を決めた BB）
　　物体像の長径を最初に決め、その後調べた長径と直交している方向の物体像の幅を短径とした長方形

si_bbs が個々の物体像ごとに算出する４種類の BBs
　SIL（short → intermediate → long の順に径を決めた BB）
　　物体像の短径を最初に決め、それと直交した面内における物体の投影像の最小の幅を中径とする。その後調べた、それら２個の径に直交している方向の像の幅を長径とした直方体。
　SLI（short → long → intermediate の順に径を決めた BB）
　　物体像の短径を最初に決め、それと直交した面内における物体の投影像の最大の幅を長径とする。その後調べた、それら２個の径に直交している方向の像の幅を中径とした直方体。

LSI（long → short → intermediate の順に径を決めた BB）
物体像の長径を最初に決め、それと直交した面内における物体の投影像の最小の幅を短径とする。その後調べた、それら２個の径に直交している方向の像の幅を中径とした直方体。
LIS（long → intermediate → short の順に径を決めた BB）
物体像の長径を最初に決め、それと直交した面内における物体の投影像の最大の幅を中径とする。その後調べた、それら２個の径に直交している方向の像の幅を短径とした直方体。

これらのプログラムのソースファイルや Windows 用の実行ファイルは以前にも紹介した以下の書庫ファイルに入れてあります。

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

また、プログラムのインストール法も以前のものと同じです（gfortran をインストールすれば MacOS の上でも新しいプログラムをコンパイルできます)。

On Sat, 01 Jun 2013 12:47:03 +0900, Tsukasa NAKANO wrote:

> wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/ch.zip
> unzip ch.zip
> cd ch && make && cd ..
>
> ただし、「f77 がない」と言われたら以下を入力してみて下さい。
> make FC="gfortran -O" && cd ..
> さらに、「gfortran がない」と言われたら GNU-FORTRAN（gfortran）をインストールし、
> その後「make FC="..."」を再度入力して下さい。

以下にクラスタ画像の上の複数個の物体像の CHs や BBs を一度に抽出することが可能な新しいプログラムに関することを書きます。

(0) ３次元凸包抽出用 FORTRAN サブルーチン CHULL3 のバグについて
(1) プログラム t_chs と si_chs の使い方
(2) プログラム t_bbs と si_bbs の詳細

(0) ３次元凸包抽出用 FORTRAN サブルーチン CHULL3 のバグについて

以前の E-mail（http://www-bl20.spring8.or.jp/~sp8ct/tmp/ch.pdf ）にも書いたように、ぼくは３次元画像上の物体像（および STL データが表現する物体像）の凸包の抽出に杉原厚吉さんが書いた FORTRAN のサブルーチン CHULL3 を流用しています。このサブルーチンは非常に良くできていますが、最近になってその出力に以下の２種類の問題が生じていることがあることに気づきました。

[a] 致命的な問題
凸包を構成している三角形として線分に退化したもの（３頂点が１本の直線上に並んでいる三

角形）が混じっていることがある。

[b] STL データに変換した場合に問題になること
これを文章で説明するのは難しいので実例の図を用意しました。この E-mail に添付した stl.gif をご覧下さい。CHULL3 で抽出した凸包を構成している三角形には「STL NG」と記した図のような、STL データとして許されていない配置のものが含まれていることがありました。

これらのうちの[2]は鳥瞰図の描画のような凸包の STL データの利用に際して実用上の問題はないと思われます。また、CHULL3 のソースコードは難解なので、この問題を修正するのは非常に難しいです。そのため、今のところは問題点[2]に対して何も対処していません。

一方、[1]は凸包のデータの利用に際して大問題です。特に、後で紹介するプログラム si_bbs では凸包を構成している三角形のデータを使って SIL や SLI の BBs の短径を物体像の回転なしで厳密に計算するようになっています。そこで、以前に書いたものを含めた CHULL3 を利用しているプログラムすべてに問題点[1]を回避するようなコードを追加しました。

と言うわけで、前記の書庫ファイル ch.taz や ch.zip のものはもちろんのこと、以前に紹介した以下の書庫ファイルに入れたソースコードを修正しましたので、今後はこちらをお使い下さい。

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

(1) プログラム t_chs と si_chs の使い方

これらは後で説明する t_bbs と si_bbs のプロトタイプの実行のために書きました。それ以外に使うことはないと思うので、使用法だけを簡単に説明します。

プログラム t_chs

起動法

t_chs　TIFF　rangeList

機能

クラスタ画像 TIFF の上の rangeList で指定した画素値のそれぞれが表す物体像の 2 次元凸包を抽出し、それらの輪郭線のデータ（複数行のテキストデータ）を画素値ごとにその昇順で標準出力に書き出す。

出力（すべての行に 2 個の整数値が並んでいる）

出力されたそれぞれの凸包の輪郭線のデータは物体像を表す画素値ごとに以下の構成になっている。ただし、以下で用いている行の順番は画素値ごとのデータそれぞれに対する値である。

1 行目

[1] 画素値 i

[2] その画素値の物体像の凸包の輪郭線を構成する点の個数 Ni

2 ～ 1 + Ni 行

[1,2] 凸包の輪郭線を構成する点の x および y 座標値

si_chs

起動法

si_chs　directory　nameFile　rangeList

機能

directory と nameFile で指定したクラスタ画像の上の rangeList で指定した画素値のそれぞれが表す物体像の 3 次元凸包を抽出し、それらの表面を構成する三角形のデータ（複数行のテキストデータ）を画素値ごとにその昇順で標準出力に書き出す。

出力（すべての行に 3 個の整数値が並んでいる）

出力されたそれぞれの凸包の表面のデータは物体像を表す画素値ごとに以下の構成になっている。ただし、以下で用いている行の順番と頂点の番号（j = 0～ ; 行の順番を k = 1～ として、j = k - 2）はいずれも、画素値ごとのデータそれぞれに対する値である。

1 行目

[1] 画素値 i

[2] その画素値の凸包の表面を構成する頂点の個数 Vi

[3] その画素値の凸包の表面を構成する三角形の個数 Fi

2 ～ 1 + Vi 行

[1,2,3] 凸包の表面を構成する頂点 j の x、y および z 座標値

2 + Vi ～ 1 + Vi + Fi 行

[1,2,3] 凸包の表面を構成する三角形の 3 個の頂点の番号

なお、凸包の表面を構成するそれぞれ三角形の 3 頂点は凸包を外部から見た時に反時計回りで並ぶ配置になっている。

(2) プログラム t_bbs と si_bbs の詳細

以前に紹介したものとは異なり、これらのプログラムは（クラスタ）画像の上の物体像の複数種類の BBs のすべてを（数値計算の精度の範囲内で）正確に決めることができます。ここではその原理について少し説明しておきます。

まず、以下に引用した以前の E-mail の言及「短→長の順で物体像の径を正確に決めるノギス法_sl_exact は原理的に実現不可能です」は間違いでした。

On Sat, 01 Jun 2013 12:47:03 +0900, Tsukasa NAKANO wrote:

> (3) 「ノギス法」による物体像の直交する 3 方向の「径」の算出法
> ...
> ノギス法_ls_exact の手続き：
>　　[1] ノギス法_sl や_ls のものと同じ。← 物体像の凸包を抽出しておく
>　　[2] 2 点間の距離が最大の頂点のペア（xj,yj）と（xk,yk）を探し出す。
>　　[3] その距離を物体像の長径 B とする。また、以下の式で計算した値 ai の値域の幅を
>　　物体像の短径 A とする：

>
>　　ai = { (yj - yk) * xi + (xj - xk) * yi } / B
>　　A = max i=1～ N { ai } - min i=1～N { ai }
>
> ２次元物体像の長径の値を高精度に決定したい場合、ノギス法_ ls では角度 θ の「刻み幅」を
> 「小さな値」に設定してやる必要があります。一方、ノギス法_ls_exact にはその制約はなく、
> 物体像の径の値を（数値計算の精度の範囲内で）正確に決めることができます。なお、その
> 証明は省略しますが、短→長の順で物体像の径を正確に決めるノギス法_sl_exact は原理的に
> 実現不可能です。

上記の処理[2]によって２次元物体像の長径を正確に決めることができますが、その短径を正確に決める手法はもう少し複雑です。そのためには物体像の凸包をあらかじめ抽出しておくことが必要不可欠です。それを行った後、以下の２段階の処理によって物体像の正確な短径の値を得ることができます。

[a] 物体像の２次元凸包の輪郭線を構成している辺 e のそれぞれに対して、その「垂線の足」との距離 D が最大となる凸包の頂点 v を探す。
[b] D(e) のうちで最小のものが物体像の短径（最小の幅）である。

ここで「垂線の足」とは「辺 e を含む直線 L と、L と直交している頂点 v を通る直線の交点」です。結局、処理[a]で探している距離 D(e) は凸包（多角形）の辺 e を「真横」に置いた場合の凸包の「高さ」に相当し、処理[b]では凸包を「１回転」することによって、それらの「高さ」の最小値を求めています。

３次元の物体像の場合の処理内容も同様で、以下のように読み替えれば、前記の処理[a]と[b]によって物体像の短径を正確に決めることができます。

処理 [b] で使っている語の読み替え
　２次元凸包の輪郭線を構成している辺 e → ３次元凸包の表面を構成している三角形 f
垂線の足の定義
　辺 e を含む直線 L と、L と直交している ... → 三角形 f を含む平面 P と、P と直交している ...

さらに、３次元物体像の場合には長径や短径を決めた後にそれと直交する面内に投影した物体像の長径や短径を決める必要がありますが、これも２次元物体像に対する手法で対処できます。このようにして、計測の手順を変えた「ノギス法」で得られる複数の BBs すべてを正確に計算できるようになりました。

さて、プログラム t_bbs と si_bbs の使用法は以下の通りです。

プログラム t_bbs
起動法
　t_bbs　TIFF　rangeList
機能
　クラスタ画像 TIFF の上の rangeList で指定した画素値のそれぞれが表す物体像の２種類の BBs のパラメータを算出し、それらの複数行のテキストデータを画素値ごとにその昇順で標準

出力に書き出す。

出力

物体像を表す画素値ごとに以下の５行が出力される。

１行目（１個の整数値）

物体像を表す画素値

２行目（２個の浮動小数点数の値が並んでいる）

SL（短 → 長径の順で決めた BB）の短径と長径

３行目（６個の浮動小数点数）

SL の３個の頂点の x および y 座標値（後述）

４行目（２個の浮動小数点数）

LS（長 → 短径の順で決めた BB）の短径と長径

５行目（６個の浮動小数点数）

LS の３個の頂点の x および y 座標値（後述）

ただし、上記の BBs の３頂点（O、A および B とする）はいずれも、以下の３条件を満たすものを選んだ（それぞれの BB の「原点」である頂点 O には２個の候補があるが、そのどちらが選ばれているかは不明）。

[1] 辺 OA は BB の短径の辺と一致。

[2] 辺 OB は BB の長径の辺と一致。

[3] ベクトル OA、OB は２次元・右手系・直交座標系を構成。

si_bbs

起動法

si_bbs　directory　nameFile　rangeList

機能

directory と nameFile で指定したクラスタ画像の上の rangeList で指定した画素値のそれぞれが表す物体像の４種類の BBs のパラメータを算出し、それらの複数行のテキストデータを画素値ごとにその昇順で標準出力に書き出す。

出力

物体像を表す画素値ごとに以下の９行が出力される。

１行目（１個の整数値）

物体像を表す画素値

２行目（３個の浮動小数点数の値が並んでいる）

SIL（短 → 中 → 長径の順に決めた BB）の短径、中径と長径

３行目（12 個の浮動小数点数）

SIL の４個の頂点の x、y および z 座標値（後述）

４行目（３個の浮動小数点数の値が並んでいる）

SLI（短 → 長 → 中径の順に決めた BB）の短径、中径と長径

５行目（12 個の浮動小数点数）

SLI の４個の頂点の x、y および z 座標値（後述）

６行目（３個の浮動小数点数の値が並んでいる）

LSI（長 → 短 → 中径の順に決めた BB）の短径、中径と長径

７行目（12 個の浮動小数点数）

LSI の４個の頂点の x、y および z 座標値（後述）

８行目（３個の浮動小数点数の値が並んでいる）

LIS（長 → 中 → 短径の順に決めた BB）の短径、中径と長径

９行目（12 個の浮動小数点数）

LIS の４個の頂点の x、y および z 座標値（後述）

ただし、上記の BBs の４頂点（O、A、B および C）はいずれも、以下の４条件を満たすものを選んだ（それぞれの BB の「原点」である頂点 O には２個の候補があるが、そのどちらが選ばれているかは不明）。

[1] 辺 OA は BB の短径の辺と一致。

[2] 辺 OB は BB の中径の辺と一致。

[3] 辺 OC は BB の長径の辺と一致。

[4] ベクトル OA、OB、OC は３次元・右手系・直交座標系を構成。

最後に、t_bbs と si_bbs の実行例を示します。ただし、以下で用いた２値画像 060410g の上の画素値１の物体像の BBs は以前にも計算したことがあります。

```
t_bbs   060410g/235.tif   1
→
    1
    366.731 507.435          ← SL の短・長径
    213.447 -15.2935508.327 202.739 -88.2368392.722
    429.243 576.261          ← LS の短・長径
    49.5785 -14.1193477.137 23.8693 -1.42145559.881

si_bbs   060410g   -   1
→
    1
    367.587 436.118 569.222          ← SIL の短・中・長径
    332.78   617.893 52.8146 （改行していない）
    92.6944 404.963 -126.462（改行していない）
    53.3371 652.582 385.841 （改行していない）
    562.485 156.111 293.661
    367.587 443.788 570.619          ← SLI の短・中・長径
    356.331 605.432 36.0756 （改行していない）
    116.245 392.503 -143.201 （改行していない）
    60.149   665.823 360.998 （改行していない）
    560.466 146.825 307.394
```

407.197 442.861 590.349　　← LSI の短・中・長径
161.497 673.61　446.847 （改行していない）
-139.173541.508 206.108 （改行していない）
452.067 610.277 118.695 （改行していない）
253.497 121.61　634.847
460.691 467.097 590.349　　← LIS の短・中・長径
-57.2715475.47　-27.8706（改行していない）
396.533 532.206 -83.3574（改行していない）
-22.987 630.763 411.319 （改行していない）
34.7285 -76.5302160.129

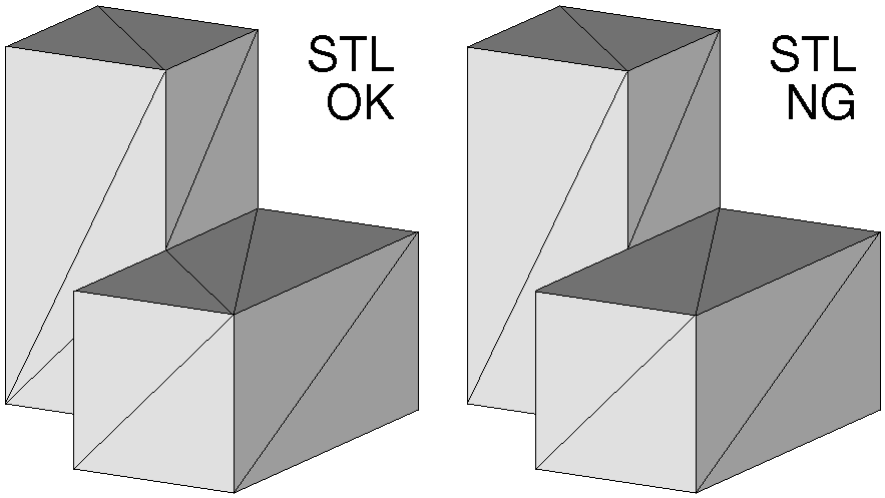
これらの物体像と BBs を一緒に描いた図も作ってみました。この E-mail に添付したファイル 060410g_235_bb.pdf と 060410g_bb.tif をご覧下さい。ただし、前者は以前の E-mail に添付した 060410g_235.pdf と同様の２次元像の図です。また、後者に並べた 12 個の図はそれぞれ、左上に示した BB（SIL、SLI、LSI もしくは LIS）の短（A)、中（B）もしくは長径（C）の方向から眺めた鳥瞰図です。これらの鳥瞰図には視線方向に関わらず BB の短、中および長径の辺を赤、緑もしくは青色で描いていますが、例えば短径の方向 A から眺めた鳥瞰図では赤色の短径の辺が点に縮退しているために見えなくなっています。

長い E-mail になりました。とりあえず以上です。

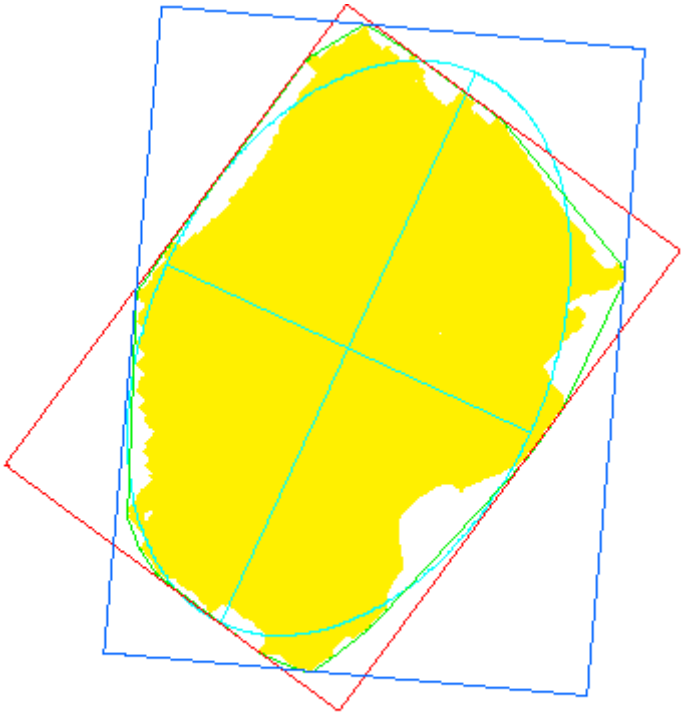
P.S.

計算機の上での物体像（もしくは点群）の BB の計算に対応している「ノギス（nonis、これはドイツ語）法」は、計算幾何学の分野では「キャリパー（caliper、カリパス、測径両脚器）法」と呼ばれているようです（どちらも意味は同じですが、論文に書く時は注意が必要)。

添付ファイル “stl.gif”

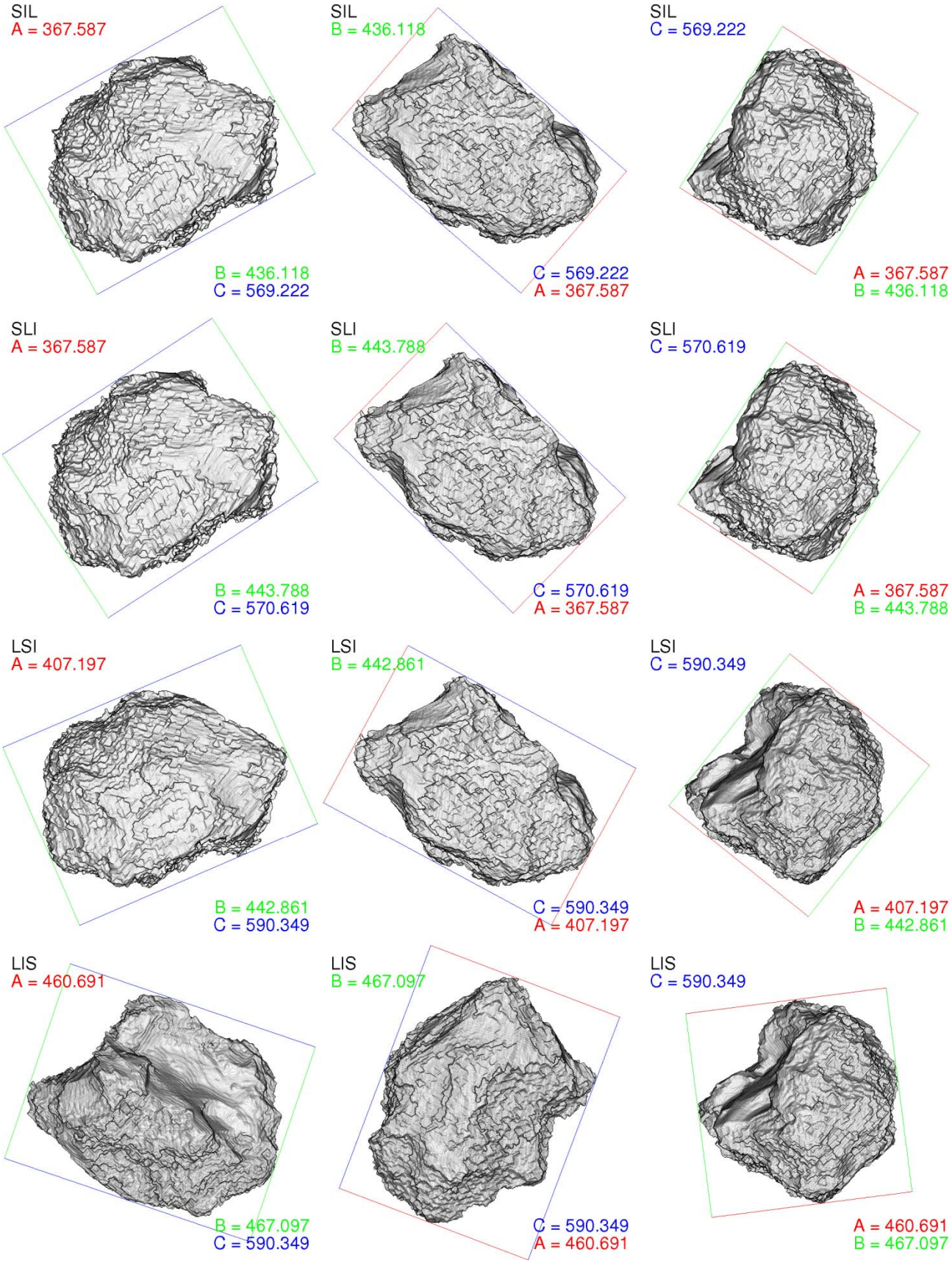


添付ファイル “060410g_235_bb.pdf”



177.31 / 268.718 = 0.659837
366.731 / 507.435 = 0.722715
429.243 / 576.261 = 0.744876

添付ファイル “060410g_bb.tif”



Date: Tue, 16 Jul 2013 12:26:26 +0900
From: Tsukasa NAKANO
To: "TSUCHIYAMA, Akira", Satoshi Okumura, Akira Shimada
Cc: Kentaro Uesugi, Tooru Matsumoto, MATSUNO Junya, Kadono Kadono, Takashi Matsushima, Michihiko Nakamura
Subject: C-shell_scripts_for_BBs

つちやまさま、
しまださま、

GSJ/AIST のなかのです。先週の SPring-8 実験の時に紹介した３次元２値もしくはクラスタ画像上の物体像の４種類の BBs（Bounding Boxes）を計算する C-shell script(s) を複数の画像を一度に処理できるようなものに改造しました。

diameters.csh　directories　rangeList

directories で指定したディレクトリのそれぞれの下に格納されている２値もしくはクラスタ画像の上の、rangeList で範囲を指定した画素値によって区別される物体像それぞれの４種類の BBs を計算し、それらの径の値をディレクトリ・画素値ごとに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。その各行には以下の 14 個の値が並んでいる。

[1] ３次元２値もしくはクラスタ画像のディレクトリ名
[2] 物体像を区別する画素値
[3,4,5] その物体像の SIL の短、中、長径の値
[6,7,8] SLI の短、中、長径
[9,10,11] LSI の短、中、長径
[12,13,14] LIS の短、中、長径

r+d.csh　directories

directories で指定したディレクトリのそれぞれの下に格納されている２値画像上の（画素値１の）物体像それぞれに対して、その近似楕円体と４種類の BBs を計算し、それらの径の値をディレクトリごとに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。その各行には以下の 16 個の値が並んでいる。

[1] ３次元２値画像のディレクトリ名
[2,3,4] その上の物体像を近似した楕円体の短、中、長軸半径
[5,6,7] その物体像の SIL の短、中、長径
[8,9,10] SLI の短、中、長径
[11,12,13] LSI の短、中、長径
[14,15,16] LIS の短、中、長径

---- 以上の２個は以前のものの改造版だが、以下は新しい C-shell script

d+r.csh　directories　rangeList

directories で指定したディレクトリそれぞれの下に格納されている２値もしくはクラスタ画像上の、rangeList で範囲を指定した画素値によって区別される物体像それぞれに対して、４種類の BBs と物体像の近似楕円体を計算し、それらの径の値をディレクトリ・画素値ごとに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。その各行には以下の 17 個の値が並んでいる。

[1] ３次元２値もしくはクラスタ画像のディレクトリ名
[2] 物体像を区別する画素値
[3,4,5] その物体像の SIL の短、中、長径
[6,7,8] SLI の短、中、長径
[9,10,11] LSI の短、中、長径
[12,13,14] LIS の短、中、長径
[15,16,17] 物体像を近似した楕円体の短、中、長軸半径

r+d.csh と d+r.csh では BBs と近似楕円体の径の出力順が異なっていることに注意して下さい。また、４種類の BBs を表す SIL、SLI、LSI と LIS の説明は以下の通りです（以前の E-mail にあった間違いを修正しました)。

On Wed, 03 Jul 2013 14:56:12 +0900, Tsukasa NAKANO wrote:

> si_bbs が個々の物体像ごとに算出する４種類の BBs
>　　SIL（short → intermediate → long の順に径を決めた BB）
>　　物体像の短径を最初に決め、それと直交した面内における物体の投影像の最小の幅を中径と
>　　する。その後調べたそれら２個の径に直交している方向の物体像の幅を長径とした直方体。
>　　SLI（short → long → intermediate の順に径を決めた BB）
>　　物体像の短径を最初に決め、それと直交した面内における物体の投影像の最大の幅を長径と
>　　する。その後調べたそれら２個の径に直交している方向の物体像の幅を中径とした直方体。
>　　LSI（long → short → intermediate の順に径を決めた BB）
>　　物体像の長径を最初に決め、それと直交した面内における物体の投影像の最小の幅を短径と
>　　する。その後調べたそれら２個の径に直交している方向の物体像の幅を中径とした直方体。
>　　LIS（long → intermediate → short の順に径を決めた BB）
>　　物体像の長径を最初に決め、それと直交した面内における物体の投影像の最大の幅を中径と
>　　する。その後調べたそれら２個の径に直交している方向の物体像の幅を短径とした直方体。

ここで紹介した３個の C-shell scripts は以前に紹介した以下の書庫ファイルの両方に入れてあります。

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

これらを解凍・展開し、その中のファイル ”*.csh” を実行パスに登録されているディレクトリにコピーすれば C-shell scripts を起動できるはずです。ただし、つちやまさんの Mac では問題ないハズですが、今回紹介した C-shell scripts の正常な実行には以下の３種類のプログラムのインストールが必要です。

[1] csh と awk

これらはそれぞれ tcsh と gawk でも代用可能ですが、シンボリック・リンクなどによって csh と awk として実行できるように設定しておく必要があります（MacOS や Linux では何もしなくても大丈夫？）。

[2] si_bbs

前記の書庫ファイルに入っている物体像の BBs の計算プログラム。

[3] si_of

物体像の楕円体近似プログラム。

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

とりあえず、以上です。