From: Tsukasa NAKANO
To: Kentaro UESUGI, Masayuki Uesugi
Date: Fri, 18 Jun 2010 21:36:41 +0900
Subject: SP-uCT-simulator
~~Attached: tg2raw.csh, jc.taz, 020607a.taz, install.csh~~

うえすぎさま、

GSJ/AIST のなかのです。指定した（任意の）スライス画像群（tomograms）に対して、それらを再構成（再々構成？）することが可能な、SP-μCT で得られるものと同等な測定データ（raw data）のファイル一式（output.log、dark.img および q*.img）を合成する（SP-μCT シミュレータとでも呼ぶべき処理用の） C-shell script "tg2raw.csh" を書きました。また、それをそちらの Linux 機上で実行する際に必要なプログラム群のインストールや、こちらで実行したその動作テストに用いた３種類の画像群 +C-shell scripts の再配置を自動で行うインストーラの C-shell script "install.csh" も作成しました。以下のようにすれば "tg2raw.csh"（のテスト）用の環境を構築できるはずです。

~~[1] この E-mail に添付した４個のファイル(上記の２個の "*.csh" と２個の書庫ファイル "*.taz") をそちらの Linux 機の HDD の空きディレクトリにコピーする。~~

~~[2] そのディレクトリに移動して "csh install.csh" と入力する。~~

上記のものは旧版の “tg2raw.csh” などのインストール法です。後の方に記したインストール法に従って新版の “tg2raw.csh” などをインストールして下さい。なお、下記のインストールに関する注意書きは新版に対しても有効です。

ただし、

[a] "install.csh" は "tg2raw.csh" が使うプログラム群のソースファイルやテストに使う画像が入った書庫ファイルを www-bl20.spring8.or.jp から wget するので、使用する Linux 機はインターネット（SPring-8 の中では LAN ？)に接続していないといけません。

[b] これらのプログラム群を gcc でコンパイルする設定にしてあります。"tg2raw.csh" の処理時間の大部分を費やすことになるスライスごとの sinogram を

作るプログラム tg2sg（正確には、そのマルチスレッド版の tg2sg_t）や、その後で合成した測定データから画像再構成を行う際に用いる hp2tg（hp2tg_tint；整数演算で逆投影する版）のコンパイルには icc を使う方が良いです。それをしたい場合はまず、"install.csh" が展開した "sp8ct/src/Makefile.thread" の先頭付近の "CC=" の設定を書き換えます（２個目の "CC=" の行をコメントアウトすれば icc を使うようになっています）。その後、"install.csh" の ~~38～39~~ 38 行目のようにして、プログラムのコンパイルとファイル移動を行えば OK です。なお、tg2sg と hp2tg には CUDA 版もあります。ぼくはそれらの使用も試しました。それらを使う場合は、"install.csh" の ~~46～47~~ 45～46 行目のようにして、CUDA 版を（シンボリック）リンクすれば良いだけです。

このようにして準備が整ったら、以下の３個のテストを試してみて下さい。

(1) ２値画像を使ったテスト

(1a) 測定データの合成法（C-shell script "run.jc_raw" で実行できる）。

```
    csh  run.jc_raw  >  jc/raw.log
or
    (  time  csh  run.jc_raw  )  |&  tee  jc/raw.log
```

ただし、後者を行えば処理の実行時間を計測でき、それは最終行として端末に表示される。なお、次に紹介する画像再々構成の処理は上の例の両方で "jc/raw.log"に書き込んだパラメータの値を使うようになっている。そのため、ここでの例で使っているファイル名（"jc/raw.log"）を別のものにしてはいけない。これらの（"tg2raw.csh" が出力する）パラメータの値の意味については後で説明する。

(1b) 画像再々構成法（"run.jc_tg"）

```
    csh  run.jc_tg  >  jc/tg.log
or
    (  time  csh  run.jc_tg  )  |&  tee  jc/tg.log  |  tail -1
```

ただし、後者の例では計測した処理の実行時間の行だけを端末に表示するようにしている。こちらのログのファイル名 "jc/tg.log" は好きなようにして良い。

元の画像に関する情報

スライス画像が格納されているディレクトリ：jc/tiff/

画素数：1147（横）×984（縦）×96（スライス数）

画素値の値域：０～１（０が白、１が黒で表示される属性付ですが）。

合成した測定データ

測定データー式を格納するディレクトリ：jc/raw/

画素値のビット数：12（これは次の画素の辺長を決める；説明省略）

画素（正方形）の辺長：...（これにも説明が必要ですが、後回しにします）

測定画像の画素数：~~1623（≒1147×√２）~~ 1512×96

dark と IO 画像の枚数：１と２（"tg2raw.csh" では固定）

投影数：900

再々構成した画像

スライス画像を格納するディレクトリ：jc/tg/

画素数：~~1623×1623~~ 1512×1512×96

画素値と CT 値の対応関係：スライスごとに CT 値の値域で正規化。

(2) 多値（３値）画像を使ったテスト

(2a) 測定データの合成法（"run.u+s_raw"）

```
csh  run.u+s_raw  >  020607a/u+s_raw.log
```

or

```
(  time  csh  run.u+s_raw  )  |&  tee  020607a/u+s_raw.log
```

ただし、先のテストの場合と同じ理由で上記のログのファイルの名前 "020607a/u+s_raw.log" を変えてはいけない。

(2b) 画像再々構成法（"run.u+s_tg"）

```
csh  run.u+s_tg  >  020607a/u+s_tg.log
```

or

```
( time csh run.u+s_tg  ) |& tee 020607a/u+s_tg.log | tail -1
```

元の画像の情報

ディレクトリ：020607a/u+s/

画素数：332×348×509

画素値の値域：0～２（"run.u+s_raw" が最初に置換する；後述）

合成した測定データ

ディレクトリ：020607a/u+s_raw/

画素値のビット数：12

画素（正方形）の辺長：...

測定画像の画素数：~~493（～348×√２）~~ 481×509

dark と IO 画像の枚数：(1) と同様

投影数：450

再々構成した画像

ディレクトリ：020607a/u+s_tg/

画素数：~~493×493~~ 481×481×509

画素値と CT 値の対応関係：(1) と同様

(3) 多値（256 値）画像を使ったテスト

(3a) 測定データの合成法（"run.xct_raw"）

```
csh  run.xct_raw  >  020607a/xct_raw.log
```

or

```
(  time  csh  run.xct_raw  )  |&  tee  020607a/xct_raw.log
```

ただし、先のテストの場合と同じ理由で上記のログのファイルの名前 "020607a/xct_raw.log" を変えてはいけない。

(3b) 画像再々構成法（"run.xct_tg"）

```
csh  run.xct_tg  >  020607a/xct_tg.log
```

or

```
( time csh run.xct_tg ) |& tee 020607a/xct_tg.log | tail -1
```

元の画像の情報

ディレクトリ：020607a/xct/

画素数：(2) のものと同じ

画素値の値域：0～255

合成した測定データ

ディレクトリ：020607a/xct_raw/

ディレクトリ以外の（3）の合成した測定データの諸元は（2）と同様。

再々構成した画像

ディレクトリ：020607a/xct_tg/

ディレクトリ以外の（3）の再々構成した画像の諸元は（2）と同様。

上のテスト（2）と（3）で使った画像はいずれも、金沢大学の森下知晃くんが地質学雑誌の口絵（vol.109、III-IV、2003）に載せた北海道・幌満岩体の symplectite（UZ1-2-1）を撮影したものです。BL47XU で行った測定 020607a（15 keV、1.5 sec.、0.5 μm、1000×1018 画素、750 投影）のデータから再構成した SP-μCT 画像をトリミングしてから縮小した（２×２×２画素を１画素にまとめた）ものが（3）の元画像で、それに各種の画像処理を加えた結果の（2）の元画像には symplectite のマトリックス（画素値１）とクラスタラベリングで「色分け」した spinel 像（２～）が含まれています。ただし、前記のように、(2a) の処理では "run.u+s_raw" が spinel 像の画素値（クラスタ番号）すべてを２に置換した後に "tg2raw.csh" を使った測定データの合成を行いました。

こちらではこれら３個のテストを高速 CPU（Intel Core i7 965）を搭載した Linux 機と NVIDIA Tesla C1060 GPU 搭載機のそれぞれで実行しました。ただし、GPU で実行したのは前述のプログラム tg2sg と hp2tg だけです。また、CPU ではこれらのマルチスレッド版プログラムを４スレッドで実行させました。こちらで行った３（×２）個のテストに要した処理時間（秒）は以下の通りでした。

テスト	Intel Core i7 搭載機	NVIDIA Tesla C1060 搭載機
測定データの合成		
(1a)	147.26	189.89
(2a)	116.56	195.68
(3a)	1360.07	1116.26
画像再々構成		
(1b)	177.14	29.33
(2b)	58.87	22.17
(3b)	59.83	22.41

長くなったので今日はここまでにします。

From: Tsukasa NAKANO

To: Kentaro UESUGI, Masayuki Uesugi

Date: Sun, 20 Jun 2010 16:42:05 +0900
Subject: SP-uCT-simulator-2
~~Attached: tg2raw.csh~~

うえすぎさま、

GSJ/AIST のなかのです。SP-μCT シミュレータの話の続きです。まず、金曜日の E-mail の補足と訂正です。

[1] 金曜日の E-mail の最初の方に書いたインストーラ "install.csh" によるインストール後は（金曜日の E-mail に添付した）書庫ファイル "jc.taz" と "020607a.taz" は不要なので消去してもかまいません。

[2] 金曜日の E-mail に添付した SP-μCT シミュレータの C-shell script "tg2raw.csh" に軽微な間違いを見つけました。CT スキャンする画像のスライスの周囲に付加する「不完全 CT」防止用の「余白」が広すぎました。適切な広さの余白を付けるよう修正した "tg2raw.csh" をこの E-mail に添付しましたので、今後はこちらをお使い下さい。

上の [2] の修正で "tg2raw.csh" が合成する測定データのサイズが若干小さくなります。そのため、プログラム hp2tg によるその再々構成の処理時間は少しだけ短くなるはずです。しかし、同一画素値で埋めた余白の「計測」時間は微々たるものなので、プログラム tg2sg によるスライスごとの sinogram 作成に要する処理時間は以前の値とさほど変わらないはずです（つまり、"tg2raw.csh" による測定データ合成の処理時間は短縮されないはず）。これらのことの確認のため、先の E-mail に記した３×２個のテストを再度行ってみました。修正版の "tg2raw.csh" で新たに合成した測定画像のサイズ（画素数）は以下の値になりました（新たに再々構成した画像の画素数もこれらに応じた値になりましたが、それらについては記載を省略します；わかりますよね？）。

テスト (1a) で合成した測定画像の画素数：1623×96 → 1512×96
(2a) および (3a) の画素数：493×509 → 481×509

また、これらのテストの処理時間は以下の値（秒）になりました。

テスト　Intel Core i7 搭載機　　NVIDIA Tesla C1060 搭載機

測定データの合成（以前とさほど変わらず）

(1a)	147.26 → 146.82	189.89 → 184.92
(2a)	116.56 → 116.41	195.68 → 189.76
(3a)	1360.07 → 1334.83	1116.26 → 1114.9

画像再々構成（以前より短くなった？）

(1b)	177.14 → 155.93	29.33 → 26.90
(2b)	58.87 → 59.35	22.17 → 21.74
(3b)	59.83 → 57.71	22.41 → 22.22

テスト用の C-shell scripts "run.*_raw" に記してあるように、"tg2raw.csh" の起動法は以下の通りです。

```
csh  tg2raw.csh  TG  views  BPS  RAW
```

ただし、上記の起動パラメータの意味は以下の通りです。

TG

CT スキャンを行う３次元画像のスライス画像ファイル（TIFF の黒白、グレーもしくは疑似カラー形式のもの）だけが格納されているディレクトリのパス名。"tg2raw.csh" はそれらのファイル名の英数字順にスライスを積層した構成の３次元画像を取り扱う。これらのスライスの横・縦画素数は個別の値でかまわないが、不完全 CT の防止のために周囲に余白を付けて元のスライス画像をセンタリングしてから処理を行うので、それらの画素数のそれぞれがすべて同じ値になっていることが望ましい。また、"tg2raw.csh" は与えられた画像上の画素は立方体だと仮定し、かつ、それらが格納している画素値が LAC の値そのものを表していると見なして測定画像の X 線強度の値を計算する。そして、この強度の値が適当な範囲におさまるように、"tg2raw.csh" は与えられた３次元画像もしくは合成する測定画像の画素の辺長を決定する（後述）。

views

CT スキャンの投影数。適当な自然数を指定すれば良い。

BPS

合成する測定画像（X 線強度の画像）もしくは「検出器」のビット数。16 以下の適当な自然数を指定すれば良い。

RAW

合成する測定データのファイル一式（output.log、dark.img および q*.img）を

格納するディレクトリの名前。ただし、"tg2raw.csh" を起動する前にこのディレクトリを自分で作成しておく必要がある。

上記のものは “tg2raw.csh” の旧版の起動法ですが、新版でも有効です。後述のように、新版では指定を省略できる起動パラメータがひとつ増えました。

すべての処理が終了した後、"tg2raw.csh" は以下の７個の数値をタブコード区切りで一行にまとめて標準出力に書き出します。

[1] 測定画像の横画素数（もしくは、CT スキャナの検出器の個数；N）
[2] 投影数（普通は記号 M と書く起動パラメータ views の値）
[3] 測定画像の縦画素数（もしくは、元の３次元画像のスライス数；Z）
[4] 測定した投影値の最大値（P；後述）
[5] 測定画像もしくは検出器のビット数（起動パラメータ BPS の値）
[6] 画素の辺長（dr = ln(2^BPS-1)/P；後述）
[7] CT スキャンの回転軸の位置（r0/dr = (1-N)/2；この値はオマケ）

上記のものは旧版の “tg2raw.csh” の端末出力です。後述のように、新版では旧版よりもひとつ増えた８個の値を２行にまとめて端末に出力します。

これらのパラメータ値を使えば、ディレクトリ RAW の中に格納されている合成した測定データセットに対して普通に画像再構成（画像再々構成）を行えるはずです。ぼくが書いた SP-μCT 画像用のプログラム群を用いる場合は、先に紹介した C-shell scripts "run.*_tg" に記したようにすれば OK です。

```
mkdir  tg        ← 再々構成する画像を格納するディレクトリ
dit_tbl  RAW  |  hp2tg  0  Z-1  dr  r0/dr  0  0  8  tg  >  tg.log
```

ただし、上で用いた RAW、Z-1、dr、r0/dr はそれぞれ前述の "tg2raw.csh" で使っていたかそれで得たパラメータ値です（ただし、スライス番号の上限値 Z-1 に関しては引き算が必要ですが）。また、ここではスライスごとにそこに出現した最小値と最大値で正規化して CT 値を８ビットの画素値として格納するように指定しました。それから、これは余談ですが、ぼくが書いたプログラム tg2tg（先に紹介したディレクトリ sp8ct/ にソースファイルなどが入っています）を使えば、このようなスライスごとにまちまちな画素値と CT 値の対応関係をスライスすべてで同じものに揃えることができます。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/sp8ct_cuda.pdf

さて、"tg2raw.csh" が行っている具体的な処理内容は以下の通りです。

(1) プログラム pig を使って与えられたスライス画像それぞれの横および縦画素数の最大値 Nx と Ny を調べる。そして、√(Nx^2+Ny^2) 以上の最小の自然数 N を CT スキャナの検出器数とする。

(2) 与えられたスライス画像の全領域が CT スキャナの視野に入る（つまり、不完全 CT にならない）ように、プログラム paste_gray を用いてその上下・左右に画素値０の余白を付加してセンタリングした N×N 画素のスライス画像を作る。

(3) tg2sg を用いてそれぞれのスライス画像の投影の sinogram を作成する。ただし、その際に画素の辺長としてとりあえずの値１を指定している。また、これらの sinograms に出現した投影の最大値 P を調べておく。

(4) こうして得た最大の投影 P に対応する X 線透過率の最小値は exp(-P) である。ただし、上の投影の計算では画素の辺長を１としているので、それが dr なら投影の最大値は P×dr に、それに対応する透過率の最小値は T = exp(-P×dr) になる。ビット数が BPS の X 線透過率画像を考え、その上で透過率０が画素値０に、透過率１が画素値 2^BPS-1 に対応する場合、この最小値 T を画像上で０でない最小の透過率の値 1/(2^BPS-1) に対応させるには dr = ln(2^BPS-1)/P を測定画像の画素の辺長とすれば良い。

(5) 上のようにして決めた dr の値を補正してそれぞれのスライスの投影の sinogram を X 線透過率の sinogram の画像に変換する。ただし、この画像のビット数は BPS であり、そこでは画素値０が透過率０ に、画素値 2^BPS-1 が透過率１に対応する。なお、この処理には新たに書いたプログラム sg_p2t を用いた。

(6) プログラム si_rar を用いて X 線透過率の sinograms の画像から構成される３次元画像を直角回転して SP-μCT による測定で得られるものと同等な、サンプルを０～180 度のそれぞれの方向から透視した X 線透過率画像のシーケンスを作成する。

(7) プログラム tiff2hp を使って上で作った TIFF 形式の画像ファイルを HiPic （ITEX） 形式の透過 X 線強度の測定画像ファイルに変換する。また、それらに対応する画素値すべてが 2^BPS-1 の入射 X 線強度（I0）の画像と、画素値０の暗電流画

像も作っておく。ただし、これらの HiPic 形式の測定画像ファイルの名前は SP-μCT による測定の場合と同様(X 線透過・入射強度画像は q*.img、暗電流画像は dark.img) にしている。

(8) 上の X 線入射・透過強度の画像の測定時刻と撮影方向（０～180 度）を記録したファイル（output.log）もそれらしいものを作った。

以上の処理中に "tg2raw.csh" はディレクトリ /tmp/ を作業領域として使用します。具体的には、そこに sinogram のデータ（１画素当たり８バイトの倍精度浮動小数点数バイナリデータとそれを変換した TIFF 形式画像）と、それを直角回転した結果の TIFF 形式画像のファイル群を置きます。量的には８バイト長のバイナリデータが大きく、およそ８×N×M×Z バイトの HDD 領域を使用します（画像の容量はその 1/4 程度）。また、"tg2raw.csh" が行う処理のうちで最大のメモリを使うのはプログラム si_rar による sinogram 画像の直角回転です。このプログラムは処理する画像のビット数（今の場合は指定した BPS の値)に関わらず画素ごとに 16 ビットのメモリを使うので、"tg2raw.csh" の実行にはおよそ２×N×M×Z バイトの計算機メモリが必要です。

テスト（1a）の場合
N=1512、M=900、Z=96 → N×M×Z ～ 125 Mega
テスト（2a）および（3a）
N=481、M=450、Z=509 → N×M×Z ～ 105 Mega

長い E-mail になりました。とりあえず以上です。

From: Tsukasa NAKANO
To: Kentaro UESUGI, Masayuki Uesugi
Date: Mon, 21 Jun 2010 16:08:43 +0900
Subject: SP-uCT-simulator-3
~~Attached: tg2raw.csh、020607a_xct_000.gif~~

うえすぎさま、

GSJ/AIST のなかのです。何度もすみません。先日から紹介している C-shell script "tg2raw.csh"（SP-μCT シミュレータ）に２個のエラーを見つけました。どちらも昨日の

E-mail に書いた以下の部分に関係するエラーです。

On Sun, 20 Jun 2010 16:42:05 +0900 Tsukasa NAKANO wrote:
> また、"tg2raw.csh" は与えられた画像上の画素は立方体だと仮定し、
> かつ、それらが格納している画素値が LAC の値そのものを表している
> と見なして測定画像の X 線強度の値を計算する。そして、この強度の
> 値が適当な範囲におさまるように、"tg2raw.csh" は与えられた３次元
> 画像もしくは合成する測定画像の画素の辺長を決定する（後述）。

まず、昨日の E-mail に添付した "tg2raw.csh" では「与えられた画像上の画素値が LAC の値そのものを表す」ように取り扱っていませんでした。画素値０を LAC の値０、最大の画素値（８ビット画像なら 255）を LAC の値１に対応づける設定になっていました。普通の画像ならこれでも問題なしですが、2 種類以上の異なったビット数のスライス画像を含む場合に問題が生じるので、上記の説明通りの動作を行うよう "tg2raw.csh" を修正しました。この E-mail に添付したものが修正版です。今後はこちらをお使い下さい

それから、これはテスト (3) で得られる再々構成画像を眺めるとわかりますが、"tg2raw.csh" で合成した測定画像には透過 X 線強度の値が不十分な画素があるようです。それから再々構成した画像上の高 CT 値の部分に streak noise（高 LAC 値の像の「影」）が生じています。これは「透過 X 線強度の値が適当な範囲にない」ために起こったようで、具体的には測定画像の画素の辺長（dr）の決め方が悪いみたいです。ただ、dr に直接関与する測定画像のビット数（BPS）を変えても状況は改善されないので、根本的な解決法を考えないとダメな感じです。

添付した画像 020607a_xct_000.gif をご覧ください。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/020607a_xct_000.gif

この上に並べた８枚はいずれも、テスト (3) の元画像（020607a/xct/）のうちの１枚のスライス（000.tif）だけを対象に合成した測定データから再々構成した画像です。"tg2raw.csh" で投影数（views）とビット数（BPS）を変えて合成した６組の測定データと、投影数を変えて tg2sg で作成した２組の８バイト浮動小数点数バイナリの投影 sinogram のデータを使いました。８枚の画像それぞれの左上隅に示した数値がこれらの投影数とビット数です（投影数だけのものがバイナリデータを使った画像）。また、それぞれの再々構成画像に出現した CT 値の値域を右下隅に表示しました。ただし、元画像の CT 値（= 画素値）の値域は０～255 です。また、８枚の画像上の画素値と表示輝度の対応関係はすべ

てそれ（CT 値 = 画素値 = 0～255）に揃えてあります。これらを眺めれば明らかなように、バイナリデータを使った場合には streak noise は発生していません。このエラーは X 線透過率の画像化の過程に原因があるようです。そして、ビット数を増やすと streak noise は徐々に少なくなりますが、16 ビットでもなくなりません。これはなぜだ？

と言うことで、「X 線 CT は原理は簡単だけれども測定は難しい」を結論としてこの問題の解決は今後の課題にしたいと思います。"tg2raw.csh" に関する話はこれでひとまず close にします。長々とすみませんでした。

From: Tsukasa NAKANO
To: Kentaro UESUGI, Masayuki Uesugi
Date: Mon, 28 Jun 2010 19:18:19 +0900
Subject: SP-uCT-simulator-final
~~Attached: 020607a_xct_000_wb.gif~~

うえすぎさま、

GSJ/AIST のなかのです。これで close にすると言った後に色々と試した結果、SP-μCT シミュレータ（"tg2raw.csh"）の以下の問題点を解決しました。

On Mon, 21 Jun 2010 16:08:43 +0900 Tsukasa NAKANO wrote:
> それから、これはテスト (3) で得られる再々構成画像を眺めるとわかりますが、
> "tg2raw.csh" で合成した測定画像には透過 X 線強度の値が不十分な画素がある
> ようです。それから再々構成した画像上の高 CT 値の部分に streak noise（高
> LAC 値の像の「影」）が生じています。これは「透過 X 線強度の値が適当な範
> 囲にない」ために起こったようで、具体的には測定画像の画素の辺長（dr）の決
> め方が悪いみたいです。ただ、dr に直接関与する測定画像のビット数（BPS）を
> 変えても状況は改善されないので、根本的な解決法を考えないとダメな感じです。

この問題は結局、投影 p から X 線透過率 t への変換式 t = exp(-p) の非線形性に由来したものでした。つまり、投影が大きい（== 透過率が小さい）ところでは投影に対する透過率の変化の割合 dt/dp = -exp(-p) = -t が小さいので、最大値に近い値の投影はどれもほぼ同じ透過率になってしまうということです。特に、以前の "tg2raw.csh" で作成していた投影の最大値に対応する透過率の最小値を画素値１とした透過率画像の場合はなおさ

らで、最大値に近い投影に対応する透過率の画素値はどれも同一の値（画素値１）になっていました。

そこで、新版の "tg2raw.csh" では投影の最大値に対応する最小の X 線透過率を指定した値β（「バイアス」と呼びます）に変換することにしました。そして、この値を画素の辺長 dr を介して変換式 t = exp(-p) に繰り込みます。つまり、以前と同様に画素の辺長に１を指定して計算した投影の最大値を P とすると、

$$\beta = \exp(-P \times dr) \rightarrow dr = -\ln(\beta) / P$$

βは透過率なので β＜１です。また、この値は透過率画像の１以上の画素値に対応していないとダメなので β≧ 1/(2^BPS-1) です。以前の "tg2raw.csh" ではこのβの下限の値に相当する画素の辺長 dr を使っていました。

$$\beta = 1/(2^{BPS} - 1) \rightarrow dr = \ln(2^{BPS} - 1) / P$$

さて、問題は新版の "tg2raw.csh" に指定すべき βの値です。これについては良くわからなかったので、以前と同様にテスト (3) のスライス画像を使って実験してみました。添付した画像 020607a_xct_000_wb.gif をご覧下さい。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/020607a_xct_000_wb.gif

この上に並んでいる８枚は投影数（views）に 450、ビット数（BPS）に 12 もしくは 16、β の値として 1/(2^BPS - 1)、0.001、0.01 もしくは 0.1 を指定して合成した測定データから再々構成した画像です。ただし、β = 1/(2^BPS -1) の画像には views と BPS の２個の値だけが、それ以外はβのものを加えた３個の値が左上隅に示されています。また、右下隅のものは各画像に出現した CT 値の値域です。これらから明らかなように、BPS＝12 なら 0.01、BPS＝16 なら 0.001 程度のβの値を指定すれば streak noise がほとんど生じないことがわかります。

注

大昔にも話したことですが、ぼくの CBP エンジンで推定した LAC（CT 値）には１％程度の計算誤差（使用している再構成フィルタに起因する不可避なエラー）が含まれます。テスト (3) の元画像の LAC の値域は０～255 ですから、その再々構成画像上に出現した CT 値の値域の下端（～０）と上端（～255）には 2.55 程度の CBP エンジンに起因する計算誤差が含まれるはずです。そして、β =

1/(2^BPS -1) の画像や BPS = 12 かつ β = 0.001 の画像の CT 値にはその値以上の、"tg2raw.csh" の処理に起因する計算誤差が含まれているようです。

このように合成する測定画像のビット数が 12 なら 0.01 程度のバイアスの値を指定すれば十分なことがわかりました。しかし、この値は元画像のビット数にも依存するはずです。"tg2raw.csh" のテスト用の３種類の画像に対しては以下のバイアスの値を指定すると streak noise は生じませんでした。

テスト (1) の ２値画像 : BPS = 12 かつ β = 1/(2^BPS -1)
テスト (2) の ３値画像 : BPS = 12 かつ β = 0.001
テスト (3) の 256 値画像 : BPS = 12 かつ β = 0.01

ところで、これは余談ですが、SP-μCT による実際の透過 X 線強度の測定では 10％(0.1) 程度の「バイアス」を推奨していますよね。これはシミュレータの場合よりもかなり大きい値ですが、それは多分、ノイズ（暗電流）レベルが高いためですね（ぼくの記憶では暗電流画像は８ビット程度のノイズを含む)。

と言うことで、上記のような透過 X 線強度のバイアスの値を指定できるように SP-μCT シミュレータの C-shell script "tg2raw.csh" を大幅に改造しました。新版の "tg2raw.csh" の起動法は以下の通りです。

```
csh  tg2raw.csh  TG  views  BPS  {bias}  RAW
```

ただし、

TG（以前と変わらず）
　CT スキャンする３次元画像を入れたディレクトリの名前。
views（以前と変わらず）
　CT スキャンの投影数 (M)。
BPS（以前と同様だが、その下限の値をチェックするようにした）
　合成する測定画像のビット数で、２～ 16 の自然数。
bias（新設；省略可）
　X 線透過率の「バイアス」の値（β)。1/(2^BPS -1) $\leqq \beta < 1$ でなければならない。その指定を省略すると β = 1/(2^BPS - 1) と見なす。
RAW（以前と変わらず）

合成した測定データセット一式を格納するディレクトリの名前。

また、このバイアスの値の導入に伴い "tg2raw.csh" の端末出力も変えました。それぞれにタブコード区切りで値を並べた以下の２行を出力するようにしました。

１行目（６個の値が並んでいる）
- [1] CT スキャナの検出器数（N；以前とかわらず）
- [2] 投影数（views もしくは M；以前と変わらず）
- [3] スライス数（Z；以前と変わらず）
- [4] 測定した投影の最大値（Ｐ；以前と変わらず）
- [5] 測定画像もしくは検出器のビット数（BPS；以前と変わらず）
- [6] バイアスの値（β ；新設）

２行目（２個の値が並んでいる）
- [7] 画素の辺長（dr；以前とは異なる値の場合もある）
- [8] CT スキャンの回転軸の位置（r0/dr = (1-N)/2；以前と変わらず）

それから、"tg2raw.csh" が合成する測定データ一式ですが、旧版では回転軸の位置決めに使用する「180 度の方向からの透視画像」の合成を省略していました。これに関して、実際の SP-μCT の測定で得られるものと変わりないファイル群を合成するように修正しました。新版の "tg2raw.csh" が合成した測定データを使えば、SP-μCT の画像再構成のすべての処理を試すことができるはずです。

旧版のものに加えて、新版の "tg2raw.csh" の実行には余分なプログラムが１つ必要です（プログラム rar_gray)。これらのプログラム群の実行環境の設定と、旧版のものと同様な３個のテスト用の画像や C-shell scripts のインストール法も変えました（インストーラも改造しました)。それは以下のようにします。

[1] 以下の書庫ファイルをダウンロードし、空きディレクトリにコピーする。
http://www-bl20.spring8.or.jp/~sp8ct/tmp/sp8ct_demo.taz

[2] そのディレクトリで以下を入力し、この書庫ファイルを解凍・展開する。
```
tar   xzf  sp8ct_demo.taz
```

[3] 以下を入力してインストーラを実行する。
```
csh  install.csh
```

ただし、旧版のものと同様にインストーラはインターネット経由で "tg2raw.csh" の実行に必要なプログラムの書庫ファイル群を wget し、gcc を用いてそれらをコンパイルします（これらに関する注意点は旧版のものと同じです)。

新版の "tg2raw.csh" による３種類の画像を使ったテストの実行法は旧版のものと同じです。ただし、測定データの合成を行う "run.u+s_raw" と "run.xct_raw" には前記の streak noise が生じないことを確認したバイアスの値を埋め込んであります。また、"run.*_tg" には "tg2raw.csh" の端末出力の変更に応じた修正を加えました。そして、これらを用いた３×２個のテストを順に実行する C-shell script "run.all" も用意しておきました。それは以下のようにして起動します。

```
csh  run.all  |  tee  all.log
```

ただし、上記のログを入れるファイルの名前は "all.log" でなくても OK です。その各行には以下の値がタブコード区切りで書き込まれます。

奇数行（測定データの合成処理の概要）
- [1] テスト名（"jc_raw"、"u+s_raw"、"xct_raw"）
- [2-4] そのテストで合成する測定データの N、M、Z の値
- [5] そのテストに要した処理時間（秒）

偶数行（画像再々構成処理の概要）
- [1] テスト名（"jc_tg"、"u+s_tg"、"xct_tg"）
- [2] そのテストに要した処理時間（秒）

長い E-mail になりました。これで終わります。お騒がせしました。