Date: Thu, 09 Mar 2017 16:47:29 +0900
From: Tsukasa NAKANO
To: Kentaro Uesugi
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: ray_sum

うえすぎさま、

GSJ/AIST のなかのです。

(0)
画像（物体像）を透過した X 線の投影値を単純な「ray sum」で近似する X-ray CT simulators を書いてみました。これらは X 線光路を幅が 0 の直線と仮定し、それが横切る画素ごとの「半直線の長さ（光路長）と画素値（X 線 LAC）の積」の総和（ray sum）を X 線投影値とする X-ray CT simulators で、古の Herman の教科書（Image reconstrucion from projections）にも解説が載っています。

注

書庫ファイル

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

などに入っている以前に紹介した parallel beam（PB）CT simulators プログラム "tg2sg*" や "tg2xp*" は個々の X 線検出器の幅について積算した投影値（つまり、X 線光束の投影値）を計算します。また、書庫ファイル radon.taz に入っている PB、fan beam（FB）と cone beam（CB）CT simulators プログラム "pb_*"、"fb_*" と "cb_*" はいずれも Radon 変換の高精度な近似式（？）を用いて 2 もしくは 3 次元画像を透過した X 線の投影値を算出します。

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

(1)
computer graphics の技法として有名な Cohen and Sutherland（C & S）と Digital Differential Analyzer（DDA）の 2 つのアルゴリズムを応用した 2 次元画像に対する ray sum の汎用計算コード（rs_2d.[h,c]）を書き、それを使った 2 および 3 次元画像を撮影対象とする PB-CT と FB-CT simulators の合計 4 個のプログラムを作成しました。

pb_2d と fb_2d

PB-CT と FB-CT のそれぞれで撮影した 2 次元（断面）画像の投影値の sinogram を計算するシングルスレッドプログラム。

pb_3d と fb_3d

PB-CT と FB-CT のそれぞれで撮影した 3 次元画像の X 線投影画像群を計算するマルチスレッドプログラム。

(2)
C＆S と DDA のアルゴリズムはどちらも 3 次元に拡張できるので（以前に紹介した鳥瞰図描画用プログラム群はそれを行ったコードを使っています)、それらを応用した 3 次元画像用 ray sum の計算コードやそれを使った CB-CT simulator プログラムも作成可能です。しかしながら、その完成にはちと時間がかかりそうなので、とりあえずは上記のプログラム（＋α）だけをここで紹介します。

(3)
新しいプログラムのソースコードの類を以下の書庫ファイルに入れておきました。
http://www-bl20.spring8.or.jp/~sp8ct/tmp/raysum.taz

(4)
この書庫ファイルには以前に紹介した PB および FB-CT simulators で撮影した sinogram や投影画像群からもとの画像を（再）再構成するために必要なプログラムのコードも入れてあります（以下にそれらの説明文を引用しました；ただし、プログラム sg_p2f はオマケです)。

注
書庫ファイル raysum.taz には新しいプログラム pb_2d で撮影した 2 次元 PB-CT の sinogram から画像再構成を行うために必要なプログラム "sg2tg*" のコードを入れてありません。そのことに関する以前の注意書きも以下に引用しました（そのインストール法は後で説明します)。

On Mon, 05 Sep 2016 17:13:33 +0900 Tsukasa NAKANO wrote:
> 2D PB-CT simulators で撮影した sinogram からの画像再構成には既存の（書庫ファイル
> radon.taz には入っていない）プログラム "sg2tg*"をお使い下さい。それら（CPU 用と CUDA
> GPU 用のものを合わせて 8 個あります）のインストール法は書庫ファイル radon.taz に入って
> いるファイル install.txt に記した通りです。なお、2D FB-CT で得た sinogram をそのまま処理
> できる画像再構成プログラムはありません。radon.taz に入っているプログラム sg_f2p を使って
> そのバイナリデータを PB-CT 用のものに変換し、それを "sg2tg*" に食わせて画像再構成を
> 行って下さい。

On Tue, 13 Sep 2016 13:17:36 +0900 Tsukasa NAKANO wrote:
>　　sg_p2f　　PB CT の sinogram を FB CT のものに変換するプログラム
>　　sg_f2p　　FB CT の sinogram を PB CT のものに変換するプログラム

On Mon, 05 Sep 2016 17:13:33 +0900 Tsukasa NAKANO wrote:
> 以前に紹介した mouse.txt などで使っているように、3D CT simulators で撮影した XP 画像群から
> 以下のプログラムを使って画像再構成を行うことができます。
>　　pb_cbp
>　　SIXM 用のコードを書き換えた CBP 法を用いた PB-CT 用のプログラム。

> fb_cbp
> 前述の FB-CT の画像再構成法に関するメモ
> http://www-bl20.spring8.or.jp/~sp8ct/tmp/fbct.pdf
> の後半に記されている CBP 法を用いた画像再構成プログラム

On Mon, 15 Aug 2016 18:25:00 +0900 Tsukasa NAKANO wrote:
> trim_float 再構成画像の各スライスのトリミングを行うためのプログラム。
> t2t_float float-TIFF 画像を integer-TIFF 画像に変換するプログラム
> t2g_float float-TIFF 画像を animation-GIF に変換するプログラム

(5)
新しい PB および FB-CT simulators プログラム "[p,f]b_[2,3]d" の使用法はいずれも書庫ファイル radon.taz に入っている同名もしくは類似の名前のプログラムのものとまったく同じです。それらと先に説明文を引用した画像再構成用などのプログラムそれぞれの使用法は前記の書庫ファイル radon.taz の中のテキストファイル radon/usage.txt に記されています。新しい書庫ファイル raysum.taz に入れたプログラム専用の説明書をまだ作っていないので、以下のようにして展開した radon/usage.txt で代用して下さい（すみません）。

```
wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/radon.taz
tar xzf radon.taz
```

(6)
書庫ファイル raysum.taz には新しい PB および FB-CT simulators プログラムを UNIX 端末から実行するデモ用の C-shell scripts（その実態はこちらでデモを実行した時の端末入力などを羅列したテキストファイル）を入れてあります。それらをこの E-mail にも添付しておきました。以下のファイルです。

raysum/lenna.txt
512×512 画素・8 ビット画素値の「Lenna」の画像（lenna.tif）を新しい 2 次元 PB および FB-CT simulators プログラムのそれぞれで撮影した後、それらの sinograms からもとの 2 次元画像を再再構成する。

raysum/040711j.txt
1000×1000 画素×720 スライス・8 ビット画素値の測定 040711j の再構成画像（040711j/byte/*.tif）を新しい 3 次元 PB-CT と FB-CT simulators プログラムで撮影した後、それらの投影画像群からもとの 3 次元画像を再再構成する。ただし、スレッド数（THREADS）が 8 のマルチスレッドで画像の撮影と再構成の両方の処理を行う設定にしてある。

(7)
これらのスクリプトを実行するためには UNIX 端末からの

```
# 書庫ファイル raysum.taz の中のプログラムのインストール
# lenna.txt で使う画像再構成プログラム sg2tg のインストール
# lenna.txt で撮影する画像 lenna.tif のインストール
# 040711j.txt で撮影する画像 “040711j/byte/*.tif” のインストール
```

用の入力が必要です。この E-mail に添付したテキストファイル raysum.txt にその入力行を書き込んでおきました。UNIX 端末から以下を入力するだけでデモ用のスクリプトの実行の準備が整うハズです。

```
csh   raysum.txt
```

注

デモのスクリプトを実行するためには、カレントディレクトリ "." が実行パスにが設定されている必要があります。

(8)

UNIX 端末から以下を入力すれば lenna.txt と 040711j.txt を実行できます。

```
cd   raysum
csh   lenna.txt          #  1 分以内に終了するはず
csh   040711j.txt        # 計算機にもよるが 10 分以上かかるはず
```

(9)

これらのスクリプトが処理した以下の画像を比較してみて下さい。

lenna.txt が処理した画像

lenna.tif と pb.tif および fb.tif

もとの画像と PB および FB CT simulators で撮影した画像

040711j.txt が処理した画像

pb_xp/*.tif と fb_xp/*.tif（それぞれ 360 画像）

PB および FB CT simulators で撮影した投影画像

040711j/byte/*.tif、pb_tg/*.tif と fb_tg/*.tif（720 画像）

もとの画像と simulators で撮った投影画像を再構成した画像

(10)

デモ用のスクリプトは実行した X-ray CT simulators や画像再構成プログラムの処理時間(単位は秒)を UNIX コマンド /usr/bin/time を使って計測します。lenna.txt や 040711j.txt の左端に 2 個の "#" が付いている行（コメント行）に記されている値がこちらの主力計算機（gsjgix）で実行した時の処理時間です。それを含む 4 台の計算機による処理時間は以下のようになりました。

lenna.txt で実行したプログラムの処理時間（秒）

host	pb_2d	sg2tg	fb_2d	sg2tg
gsjgix	0.59	0.29	1.30	0.29
gsjvix	0.74	0.71	1.56	0.44
vrm	1.22	2.68	1.81	1.73
DBR73	0.92	0.85	1.67	0.75

040711j.txt で実行したプログラムの処理時間（秒）

host	pb_3d	pb_cbp	trim	t2t	fb_3d	fb_cbp	trim	t2t
gsjgix	168.68	152.29	64.70	39.16	172.04	153.12	64.54	40.07
gsjvix	248.54	290.15	183.41	83.36	260.11	274.36	157.31	88.79
vrm	255.99	338.06	114.88	60.34	266.57	279.11	119.42	61.55
DBR73	376.16	300.05	198.03	110.63	381.70	302.71	198.10	112.89

注

このベンチマークテストに使った計算機（host）の諸元は以下の通りです。

gsjgix	Intel Xeon E5-2687W CPU and Samsung 840 PRO Series SSD
	CentOS 6.8 and Intel C++ compiler 16.0.4
gsjvix	Intel Xeon X5570 CPU and Intel 320 Series SSD
	Red Hat Enterprise Linux 5.11 and Intel C++ compiler 16.0.4
vrm	Intel Xeon E5-2609 v2 CPU and Toshiba MQ01ABD100H HDD
	Scientific Linux 6.7 and GNU C++ compiler 4.4.7
DBR73	Intel Core i7-4710MQ CPU and Samsung 840 EVO mSATA SSD
	Microsoft Windows 8.1, MSYS2, MinGW64 and GNU C++ complier 6.3.0

(11)

計算機 gsjgix の上で新しい X-ray CT simulators と「プログラム名に "_g" を付けた以前の書庫ファイル radon.taz の中の CUDA GPU 版の simulators」の実行結果の比較も行いました。lenna.txt や 040711j.txt の場合と同様の処理によって得た画像を並べた PDF ファイル lenna.pdf と 040711j.pdf をこの E-mail に添付します。

lenna.pdf（sinogram の画像は載せていない）

画像上部のラベル

各プログラムの処理時間（単位は秒）と sinogram の投影数

上段の画像

sinogram から再再構成した画像。撮影した画像領域の外部のノイズの分布を見るため、ヒストグラム平滑化で画素値を強調してある。

中段の画像

もとの画像と同じ画素数になるようにトリミングした再構成画像

下段の画像

再構成画像ともとの画像の差分画像。差分画素値の範囲は -64～64。

040711j.pdf

最下段のもの以外の画像上部のラベル

各プログラムの処理時間（単位は秒）と処理した画像の枚数

最上段の画像

以前の simulators で撮影した投影画像。１投影だけを撮影した。

上から２番目の画像

新しい simulators で撮影した投影画像のうち最初の投影番号のもの。

上から 3 番目の画像

新しい simulators の投影画像から再構成したスライス 360 の画像

最下段の画像

再構成画像ともとの画像の差分画像。差分画素値の範囲は -64〜64。

これらより、新しい X-ray CT simulators プログラムはべらぼうに高速であり、かつ、撮影した投影画像の画質も以前にものと概ね同じであることがわかります。

長い E-mail になりました。とりあえず以上です。

P.S.

float-TIFF の画像として格納した再構成画像のトリミングやその integer-TIFF の画像への変換には予想以上に長い処理時間を要するようです。

添付ファイル lenna.txt

```
# PB-CT

    pb_2d
#       usage : pb_2d image.tif base step interval {views} SG.bin

    /usr/bin/time -f %e pb_2d lenna.tif 0 1 1 pb.bin
#       725     1138    1       -362    0.000000        105449.434538
##      0.62

    sg2tg
#       usage : sg2tg sinogram dr r0/dr t0 {f0 df} BPS tomogram

    /usr/bin/time -f %e sg2tg pb.bin 1 -362 0 0 1 8 pb.tif
#       -6.024368       244.333786
##      0.28

    trim_float
#       usage : trim_float org.tif x1 y1 x2 y2 new.tif
#               trim_float org/ nameFile x1 y1 x2 y2 new/

#   (725-512)/2 = 106.5
#   106+512-1 = 617

    trim_float pb.tif 106 106 617 617 pb.tif
#       pb.tif : using integer PV as is floating-point PV (warning).

    t2t_float
#       usage : t2t_float org.tif {base step} BPS new.tif
```

```
#              t2t_float org/ nameFile {base step} BPS new/

    t2t_float pb.tif 0 1 8 pb.tif > /dev/null

# FB-CT

    fb_2d
#       usage : fb_2d image.tif base step interval distance {views} SG.bin

    /usr/bin/time -f %e fb_2d lenna.tif 0 1 1 1024 fb.bin
#       775     2275    1       1024    387     0.000000        105473.473567
##      1.31

    sg_f2p
#       usage : sg_f2p SG_f SS SD interval center SG_p

    sg_f2p fb.bin 1024 1024 1 387 fb.bin
#       1.000000        362.009517

    /usr/bin/time -f %e sg2tg fb.bin 1 -362.009517 0 0 1 8 fb.tif
#       -1.784088       243.570816
##      0.32

    head -1 fb.bin
#       725     1139

    trim_float fb.tif 106 106 617 617 fb.tif
#       fb.tif : using integer PV as is floating-point PV (warning).

    t2t_float fb.tif 0 1 8 fb.tif > /dev/null
```

添付ファイル 040711j.txt

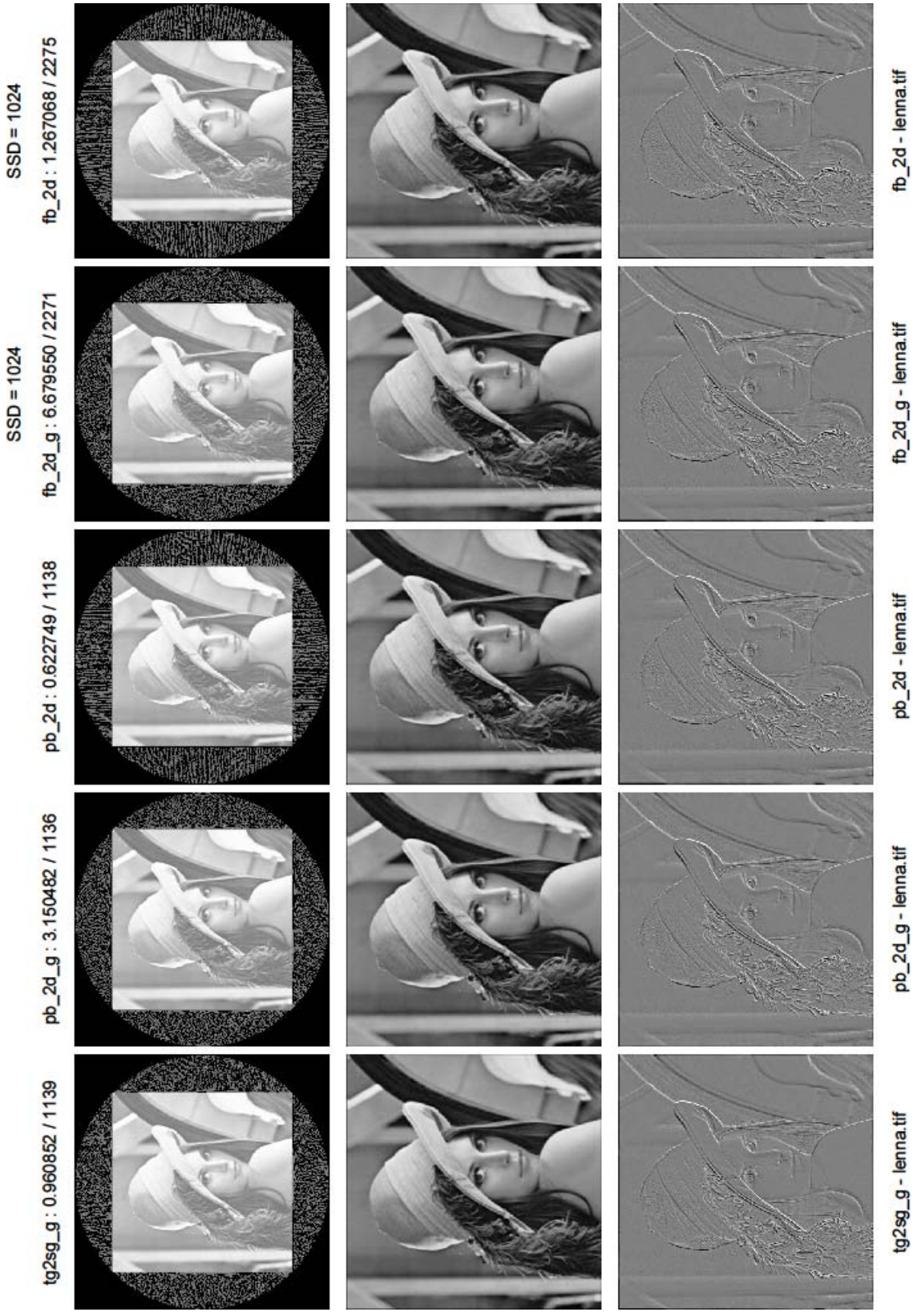
```
setenv THREADS 8

# PB-CT

    pb_3d
#       usage : pb_3d image/ NBS_file interval {views {start end}} XP_format

    mkdir pb_xp

    /usr/bin/time -f %e pb_3d 040711j/byte - 1 360 pb_xp/%03d.tif > pb_xp.log
#       1415    720     360     1       707
##      164.84

    pb_cbp
#       usage : pb_cbp XP/ nameFile Dr Or {layer1 layer2} RA0 TG_format
```

```
    mkdir pb_tg

    /usr/bin/time -f %e pb_cbp pb_xp - 1 707 0 pb_tg/%03d.tif > pb_tg.log
#       1415    720     1
##      151.98

#   (1415-1000)/2 = 207.5
#   207+1000-1 = 1206

    /usr/bin/time -f %e trim_float pb_tg - 207 207 1206 1206 pb_tg
##      64.55

    /usr/bin/time -f %e t2t_float pb_tg - 0 1 8 pb_tg > /dev/null
##      39.42

# FB-CT

    fb_3d
#       usage : fb_3d image/ NBS_file interval distance {views {start end}} XP_format

    mkdir fb_xp

    /usr/bin/time -f %e fb_3d 040711j/byte - 1 2000 360 fb_xp/%03d.tif > fb_xp.log
#       1512    720     360     2000    1       755.5
##      172.65

    fb_cbp
#       usage : fb_cbp XP/ nameFile A B Du Ou {layer1 layer2} RA0 TG_format

    mkdir fb_tg

    /usr/bin/time -f %e fb_cbp fb_xp - 2000 2000 1 755.5 0 fb_tg/%03d.tif > fb_tg.log
#       1413    720     1.000000e+00
##      153.32

#   (1413-1000)/2 = 206.5
#   206+1000-1 = 1205

    /usr/bin/time -f %e trim_float fb_tg - 206 206 1205 1205 fb_tg
##      64.09

    /usr/bin/time -f %e t2t_float fb_tg - 0 1 8 fb_tg > /dev/null
##      40.80
```

添付ファイル raysum.txt

後に示す 3/23 の E-mail の添付ファイル raysum.csh.txt を御覧下さい。

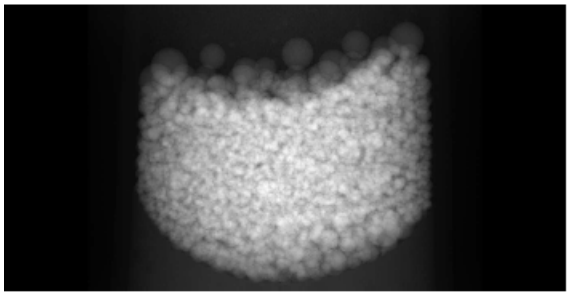
添付ファイル lenna.pdf



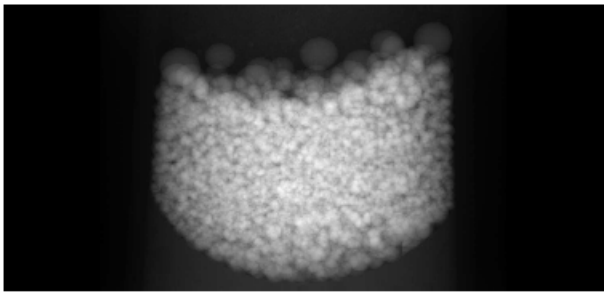
添付ファイル 040711j.pdf

SSD = 2000

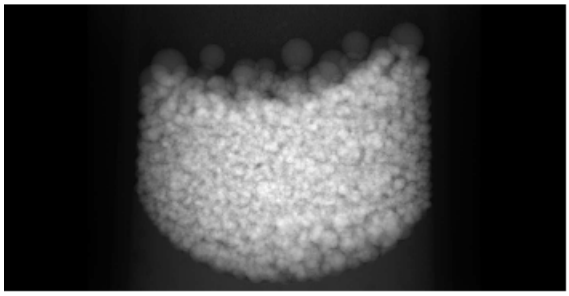
pb_3d_g : 40.703891 sec. / 1 view



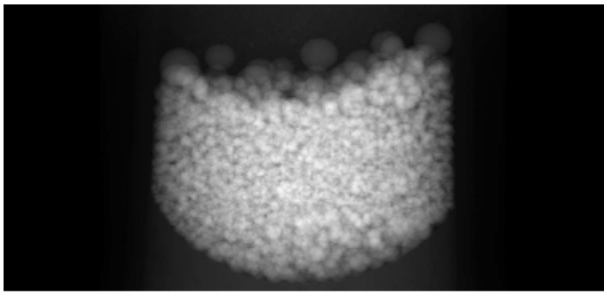
fb_3d_g : 41.763283 sec. / 1 view



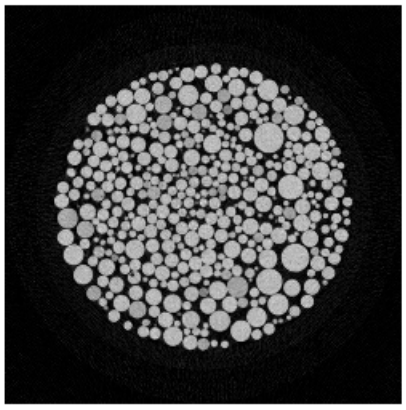
pb_3d : 161.344119 sec. / 360 views



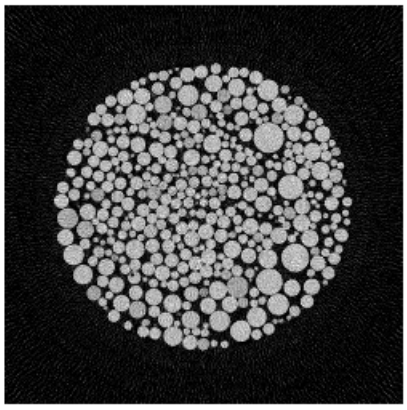
fb_3d : 166.756954 sec. / 360 views



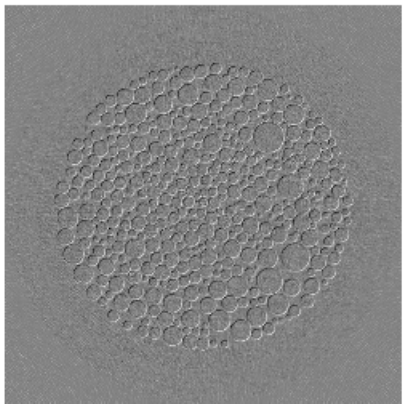
pb_cbp : 150.188299 sec. / 720 slices



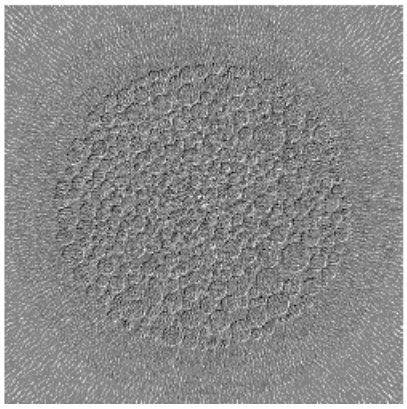
fb_cbp : 152.129367 sec. / 720 slices



pb_cbp - 040711j/byte/360.tif



fb_cbp - 040711j/byte/360.tif



Date: Thu, 09 Mar 2017 17:39:05 +0900
From: Tsukasa NAKANO
To: Kentaro UESUGI
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: Re: ray_sum

うえすぎさま、
なかのです。E-mail ありがとうございます。
(1)
実は、もとのスライス画像の各辺沿いの画素数が偶数の場合、その画像と「それを撮影した投影画像を再再構成した画像」は回転中心の設定法のために 0.5 画素幅づつ位置ズレします。つまり、X-ray CT simulators ではもとの画像の中央の点を回転中心にしているため、偶数画素数の画像ではその点が画素の辺の上に位置します。再構成画像では回転中心を画素の中央に置くようにしているので、それともとの画像とは 0.5 画素幅だけ位置ズレが生じます。もとの画像として各辺沿いの画素数が奇数のものを使えば良かった。
(2)
X-ray CT simulators に使っている float-TIFF の書き込み用関数のコードは画像圧縮のような余計なことをまったくせずにデータをシーケンシャルに書き込む「高速版」であり、それを読み込む関数も時間がかかるディスクのランダムアクセスをしていないハズなので、float-TIFF の処理が遅いのは上杉君が言う通りディスク I/O の性能のためだと思います。問題は最も最近に導入した DBR73 の SSD が遅いことです（やはり、Samsung 製はだめか？）。
とり急ぎ、

> On Thu, 9 Mar 2017 16:50:08 +0900 Kentaro UESUGI wrote:
> > 中野さん
> > 上杉です
> > ありがとうございます。
> >
> > 040711j.pdf の一番下の残渣の画像ですけども、
> > 左上から右下に向かってきれいにずれているように見えます。
> > これはどこかの丸め誤差でも引きずっているのでしょうか？
> >
> > > P.S.
> > > float-TIFF の画像として格納した再構成画像のトリミングやその integer-TIFF の
> > > 画像への変換には予想以上に長い処理時間を要するようです。
> >
> > これは読み書きの問題だけではない。ということでしょうか。
> > （gsjgix と vrm の t2t が結構速いのは、disk アクセス速度が聞いている気がしますが）

Date: Fri, 10 Mar 2017 16:37:18 +0900
From: Tsukasa NAKANO
To: Kentaro UESUGI
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: Re: ray_sum

うえすぎさま、
なかのです。

(1)
> 偶数と奇数ですね。理解しました。
> （似たような話がずいぶん前にあったような・・・、なんだっけ？）

もとの画像の（スライス面内の）各辺の画素数を奇数にして処理してみました。

lenna 画像：trim_gray で (0,0)-(510,510) の領域を切り出し
もとの画像は 512×512 画素 → 511×511 画素にした。
040711j/byte：si_trim で (0,0,0)-(998,998,719) を切り出し
1000×1000×720 → 999×999×720（スライス数はもとのまま）

その結果の lenna_odd.pdf と 040711j_odd.pdf をこの E-mail に添付しました。pb_2d_g で処理したもの以外は予想通りです（pb_2d_g にはバグがあるようです)。

(2)
> SSD は当たり外れと経年変化量が大きいので、がっつり読み書きするときは、
> 実は大容量のバッファを積んだ HDD の RAID システムの方が
> 安定して色々できるような気がします。vrm は 3TB HDDx8 RAID6 と、
> 一応 /media/ssd に SSDx2 RAID0 もあります。

X-ray CT simulators で撮影した画像とそれらから再構成した画像のファイルやディレクトリを RAM disk（/dev/shm）に置くようにしてデモを実行しました。

```
sed -e s@pb.bin@/dev/shm/pb.bin@g -e s@pb.tif@/dev/shm/pb.tif@g ¥
    -e s@fb.bin@/dev/shm/fb.bin@g -e s@fb.tif@/dev/shm/fb.tif@g lenna.txt | csh
sed -e s@pb_xp@/dev/shm/pb_xp@g -e s@pb_tg@/dev/shm/pb_tg@g ¥
    -e s@fb_xp@/dev/shm/fb_xp@g -e s@fb_tg@/dev/shm/fb_tg@g 040711j.txt | csh
```

処理時間は以下の通りです（"_shm" 付きのホスト名のものが RAM disk を使用)。

lenna.txt

host	pb_2d	sg2tg	fb_2d	sg2tg
gsjgix	0.59	0.29	1.30	0.29
gix_shm	0.54	0.32	1.30	0.29
gsjvix	0.74	0.71	1.56	0.44
vix_shm	0.78	0.49	1.62	0.52
vrm	1.22	2.68	1.81	1.73
vrm_shm	0.83	1.36	1.78	1.37
DBR73	0.92	0.85	1.67	0.75

040711j.txt

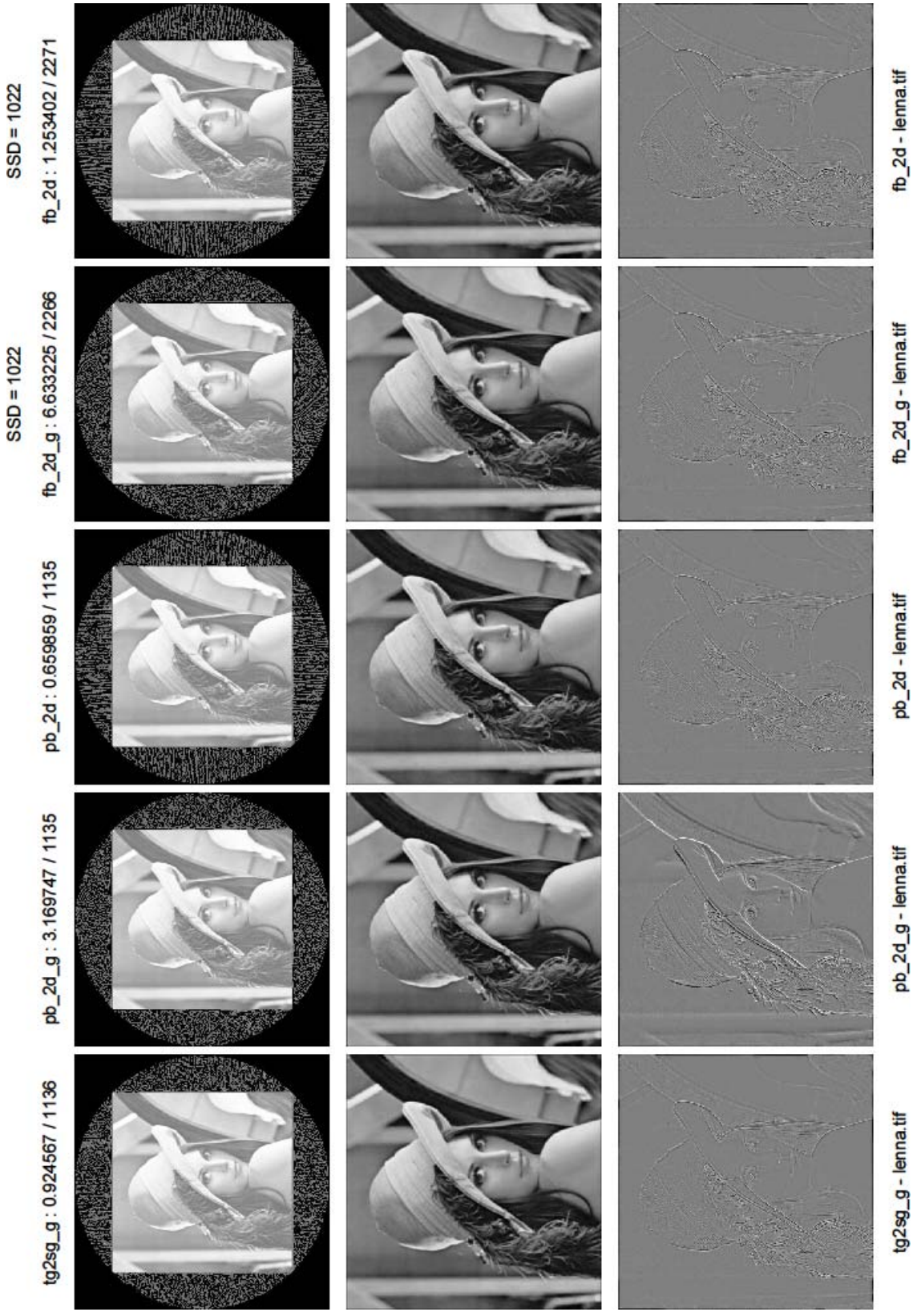
host	pb_3d	pb_cbp	trim	t2t	fb_3d	fb_cbp	trim	t2t
gsjgix	168.68	152.29	64.70	39.16	172.04	153.12	64.54	40.07
gix_shm	162.45	151.54	60.81	38.61	169.82	151.86	61.19	43.23
gsjvix	248.54	290.15	183.41	83.36	260.11	274.36	157.31	88.79
vix_shm	248.35	252.57	152.59	88.16	258.45	265.66	175.01	95.80
vrm	255.99	338.06	114.88	60.34	266.57	279.11	119.42	61.55
vrm_shm	259.66	257.46	91.15	59.99	263.97	250.49	89.30	64.71
DBR73	376.16	300.05	198.03	110.63	381.70	302.71	198.10	112.89

Float-TIFF 画像の I/O の速度が効くプログラム trim（本当は trim_float）や t2t（t2t_float）の処理速度は RAM disk を使っても劇的に速くならないので、問題は HDD や SSD の装置の性能ではないようですね（RAM disk を使うと遅くなる場合もあります；why ?）。

とり急ぎ、

On Thu, 9 Mar 2017 17:48:01 +0900 Kentaro UESUGI wrote:
> 中野さん
> 上杉です
>
> 偶数と奇数ですね。理解しました。
> （似たような話がずいぶん前にあったような・・・、なんだっけ？）
>
> SSD は当たり外れと経年変化量が大きいので、がっつり読み書きするときは、
> 実は大容量のバッファを積んだ HDD の RAID システムの方が
> 安定して色々できるような気がします。vrm は 3TB HDDx8 RAID6 と、
> 一応 /media/ssd に SSDx2 RAID0 もあります。

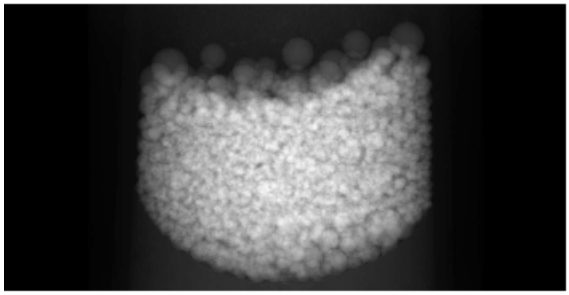
添付ファイル lenna_odd.pdf



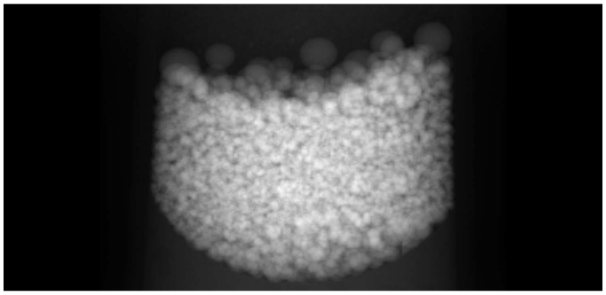
添付ファイル 040711j_odd.pdf

SSD = 1998

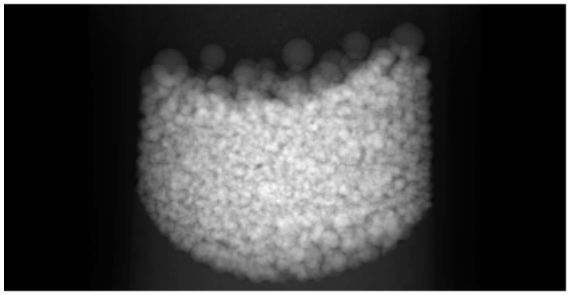
pb_3d_g : 40.442195 sec. / 1 view



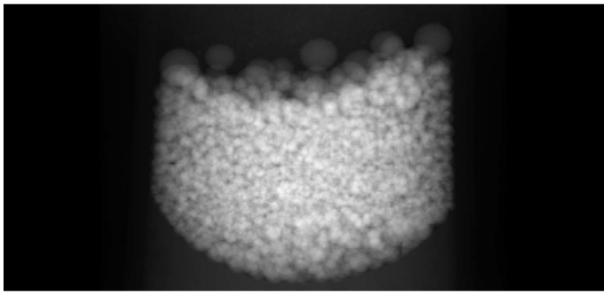
fb_3d_g : 41.646979 sec. / 1 view



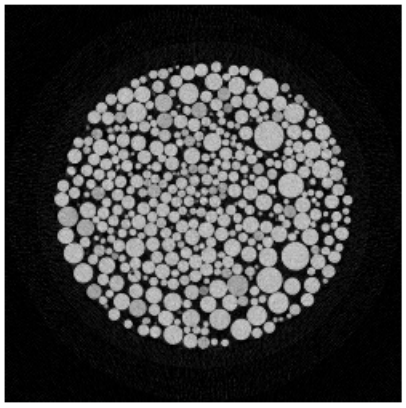
pb_3d : 157.744553 sec. / 360 views



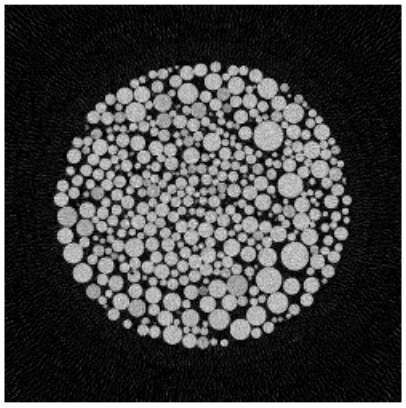
fb_3d : 163.011601 sec. / 360 views



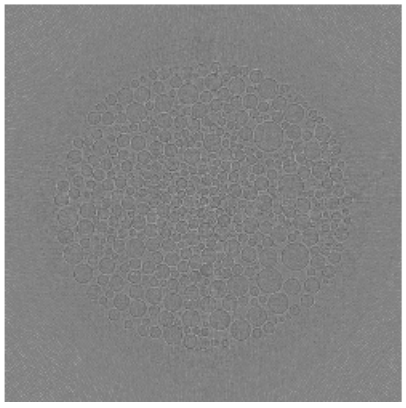
pb_cbp : 150.261818 sec. / 720 slices



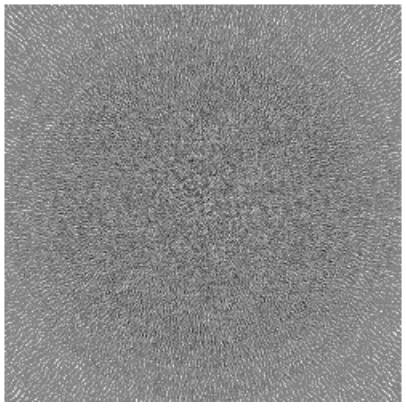
fb_cbp : 151.287141 sec. / 720 slices



pb_cbp - 040711j/byte/360.tif



fb_cbp - 040711j/byte/360.tif



Date: Tue, 21 Mar 2017 18:45:50 +0900
From: Tsukasa NAKANO
To: Kentaro Uesugi
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: ray_sum_CB-CT_simulator

うえすぎさま、

GSJ/AIST のなかのです。画像（物体像）を透過した X 線の投影値を単純な ray sum で近似する cone beam（CB）CT simulator のマルチスレッドプログラム cb_ss を書きました。

> On Thu, 09 Mar 2017 16:47:29 +0900 Tsukasa NAKANO wrote:
> > C & S と DDA のアルゴリズムはどちらも 3 次元に拡張できるので（以前に紹介した鳥瞰図
> > 描画用プログラム群はそれを行ったコードを使っています)、それらを応用した 3 次元画像
> > 用 ray sum の計算コードやそれを使った CB-CT simulator プログラムも作成可能です。

新しいプログラム cb_ss には
　3 次元用に拡張した Cohen and Sutherland（C & S）のアルゴリズム
と
　ray とスライス面が直交しないことを仮定した
　1 枚のスライス画像（single slice）だけを処理対象とする
　準 3 次元用の Digital Differential Analyzer（DDA）のアルゴリズム
を用いた ray sum の計算コード "rs_ss.[h,c]" を組み込みました。

> "rs_ss.[h,c]" は positron emission tomography（PET）の simulator には使えないので、その作成のために完全な 3 次元用の ray sum の計算コードを今後書きたいと思っています。

cb_ss のソースコードの類を以下の書庫ファイルに追加しました。
　http://www-bl20.spring8.or.jp/~sp8ct/tmp/raysum.taz
また、この書庫ファイルには以前に紹介した書庫ファイル
　http://www-bl20.spring8.or.jp/~sp8ct/tmp/radon.taz
に入っている FDK（Feldkamp、Davis and Kress）法による CB-CT 用画像再構成プログラム fdk のソースコードの類も入れておきました。

> On Mon, 05 Sep 2016 17:13:33 +0900 Tsukasa NAKANO wrote:
> > fdk
> > 以下のメモに記されている FDK 法を用いた CB-CT 用のプログラム。
> > http://www-bl20.spring8.or.jp/~sp8ct/tmp/fdk.pdf
> > ただし、これは理論的な根拠が薄弱な近似的な画像再構成法である。

新しいプログラム cb_ss の使用法は以前に紹介した書庫ファイル radon.taz の中の CB-CT simulator のプログラム群 "cb_[0-8]" のものとまったく同じです。書庫ファイル raysum.taz に入れたプログラムの説明書をまだ書いていないので、それについては書庫ファイル radon.taz の中の説明書 radon/usage.txt を御覧下さい。それは以下のようにすればダウンロード・展開できます。

```
On Thu, 09 Mar 2017 16:47:29 +0900 Tsukasa NAKANO wrote:
>       wget   http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/radon.taz
>       tar   xzf   radon.taz
```

書庫ファイル raysum.taz には CB-CT simulator のデモ用の C-shell script "cb_test.txt" も入れてあります（念のため、それをこの E-mail にも添付しました；その実態はこちらでデモを実行した時の入出力などを書き込んだテキストファイルです)。また、cb_test.txt の実行の準備を行う（3/9 の E-mail に添付した "raysum.txt" を少しだけ書き換えた）C-shell script "raysum.csh" をこの E-mail に添付しておきました。UNIX 端末から以下の 3 行を入力するだけで cb_ss や fdk による CB-CT simulator のデモを実行できるはずです。

```
csh   raysum.csh
cd   raysum
csh   cb_test.txt
```

cb_test.txt によるデモでは以下の 2 種類の画像群を作成します。

```
cb_xp/*.tif：CB-CT simulator で撮影した 360 枚の投影画像
cb_tg/*.tif：その投影画像から（再）再構成した 721 枚のスライス画像
```

ただし、計算誤差のため、CB-CT simulator の投影画像から再構成したスライス画像の枚数はもとの数(720)と同じになりません。しかしながら、最後の番号のもの以外のスライス画像（今の場合はスライス 0～719 の画像）はほぼ完全に再構成（再現？）できているハズなので、それを御確認下さい。

cb_test.txt は UNIX コマンド /usr/bin/time によって実行したプログラムの処理時間を計測します。ただし、cb_test.txt の 2 個の「#」で始まるコメント行に埋め込まれている cb_ss の処理時間（901.89 秒）はその初版の値です。cb_test.txt を 4 台の計算機で実行して得た（最新版の cb_ss を含む）4 個のプログラムの秒単位の処理時間は以下の通りでした。

Host	cb_ss	fdk	trim_float	t2t_float
gsjgix	500.15	227.90	65.27	39.55
gsjvix	683.04	356.32	147.91	87.29
vrm	862.97	492.73	117.80	60.61
DBR73	1158.70	923.27	517.07	292.44

On Thu, 09 Mar 2017 16:47:29 +0900 Tsukasa NAKANO wrote:

> 注
> このベンチマークテストに使った計算機（host）の諸元は以下の通りです。
> gsjgix
> Intel Xeon E5-2687W CPU and Samsung 840 PRO Series SSD
> CentOS 6.8 and Intel C++ compiler 16.0.4
> gsjvix
> Intel Xeon X5570 CPU and Intel 320 Series SSD
> Red Hat Enterprise Linux 5.11 and Intel C++ compiler 16.0.4
> vrm
> Intel Xeon E5-2609 v2 CPU and Toshiba MQ01ABD100H HDD
> Scientific Linux 6.7 and GNU C++ compiler 4.4.7
> DBR73
> Intel Core i7-4710MQ CPU and Samsung 840 EVO mSATA SSD
> Microsoft Windows 8.1, MSYS2, MinGW64 and GNU C++ complier 6.3.0

計算機 gsjgix の上で新しい CB-CT simulator と「"_g" が付いた名の、書庫ファイル radon.taz の中の CUDA GPU 版の CB-CT simulator」の実行結果の比較を行いました。この E-mail に添付した 040711j.pdf を御覧下さい。その左側には 3/10 の E-mail で紹介した PB および FB-CT simulators で撮影した「スライス面内の各辺の画素数を奇数にした 3 次元画像」の投影画像やそれらの再構成画像および差分画像（のスライス画像）を並べました。また、右側に示した CB-CT simulator の画像は「3 個の辺の画素数のそれぞれを奇数にした 3 次元画像」に対する処理結果です。そして、FDK 法の不完全な画像再構成の様相を見るため、CB-CT simulator の再構成画像および差分画像としてスライスの位置が異なる 3 枚のスライス画像を示しました。

040711j.pdf の右側の表示から以下のことがわかります。

[1] 単純な ray sum 計算を行っている cb_ss で撮影した投影画像は Radon 変換の高精度な近似（？）を行っている cb_8 の画像と見分けがつかない。
[2] cb_8 に比べると cb_ss はべらぼうに高速（当社比で 40 倍近く高速）。
[3] この拡大率（後注参照）なら FDK 法の不完全な画像再構成に問題なし。
[4] CB-CT で撮影した FB-CT のものと原理的に等価な中央（番号 360）のスライスの再構成画像の画質が悪い。これは何故？

注

ここで撮影した 3 次元 CT 画像 040711j/byte の実寸は 5.83^2（スライス面内）×4.1986 mm^3 であり、また、040711j.pdf の右側の上部に記されているように、CB-CT simulator に指定した SSD（source-sample distance；光源とサンプル回転軸の距離）の実寸は 11.66 mm です。つまり、CB-CT の光源から画像を見込む角度は 73.74（スライス面の方向）と 53.982 度（サンプル回転軸の方向）になります。これらの角度を cone beam による物体像の拡大率に換算すると、...。

という訳で、単純な ray sum 計算による X-ray CT simulators は「使いモノ」になりそうです。とりあえず以上です。

P.S.

以前にも書いたように float TIFF の画像（ファイル）の処理に時間がかかり過ぎなので、何とかしないとダメですね。ただし、Windows 機 DBR73 で float TIFF の処理がべらぼうに遅いのは AIST の管理部門が強制的に更新したウィルスチェッカ（McAfee）のせいかもしれません。

添付ファイル cb_test.txt

```
setenv THREADS 8

# CB-CT

    cb_ss
#       usage : cb_ss image/ NBS_file Ixy Iz SSD {views {start end}} XP_format

    mkdir cb_xp

    /usr/bin/time -f %e cb_ss 040711j/byte - 1 1 2000 360 cb_xp/%03d.tif > cb_xp.log
#       1512    1114    360     2000    1       755.5   1       556.5
##      901.89

    fdk
#       usage : fdk XP/ nameFile A B Du Ou Dw Ow {layer1 layer2} RA0 TG_format

    mkdir cb_tg

    /usr/bin/time -f %e fdk cb_xp - 2000 2000 1 755.5 1 556.5 0 cb_tg/%03d.tif > cb_tg.log
#       1413    721     1.000000e+00    1.000000e+00
##      225.54

#   (1413-1000)/2 = 206.5
#   206+1000-1 = 1205

    /usr/bin/time -f %e trim_float cb_tg - 206 206 1205 1205 cb_tg
##      63.48

    /usr/bin/time -f %e t2t_float cb_tg - 0 1 8 cb_tg > /dev/null
##      40.32
```

添付ファイル raysum.txt

後に示す 3/23 の E-mail の添付ファイル raysum.csh.txt を御覧下さい。

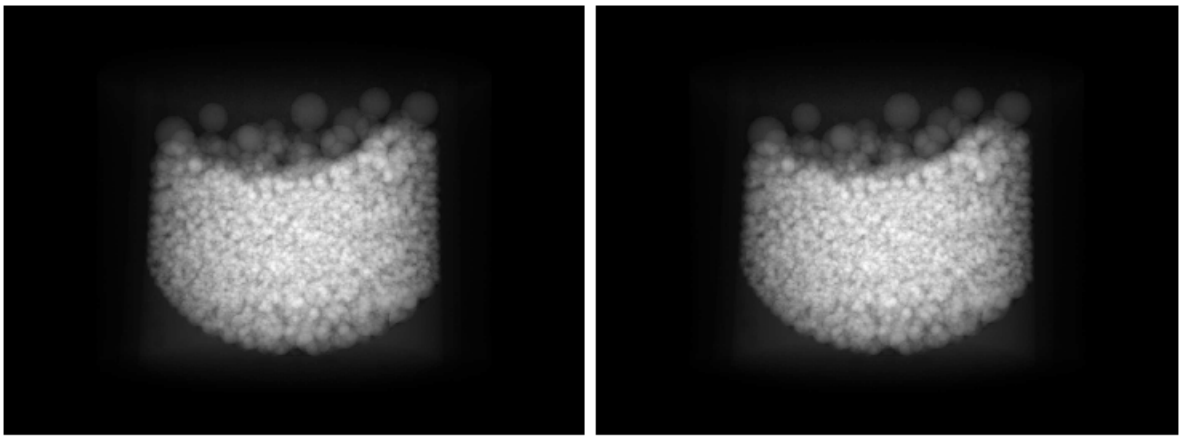
添付ファイル 040711j.pdf

ページの右側の CB-CT に関する内容だけを掲載します。

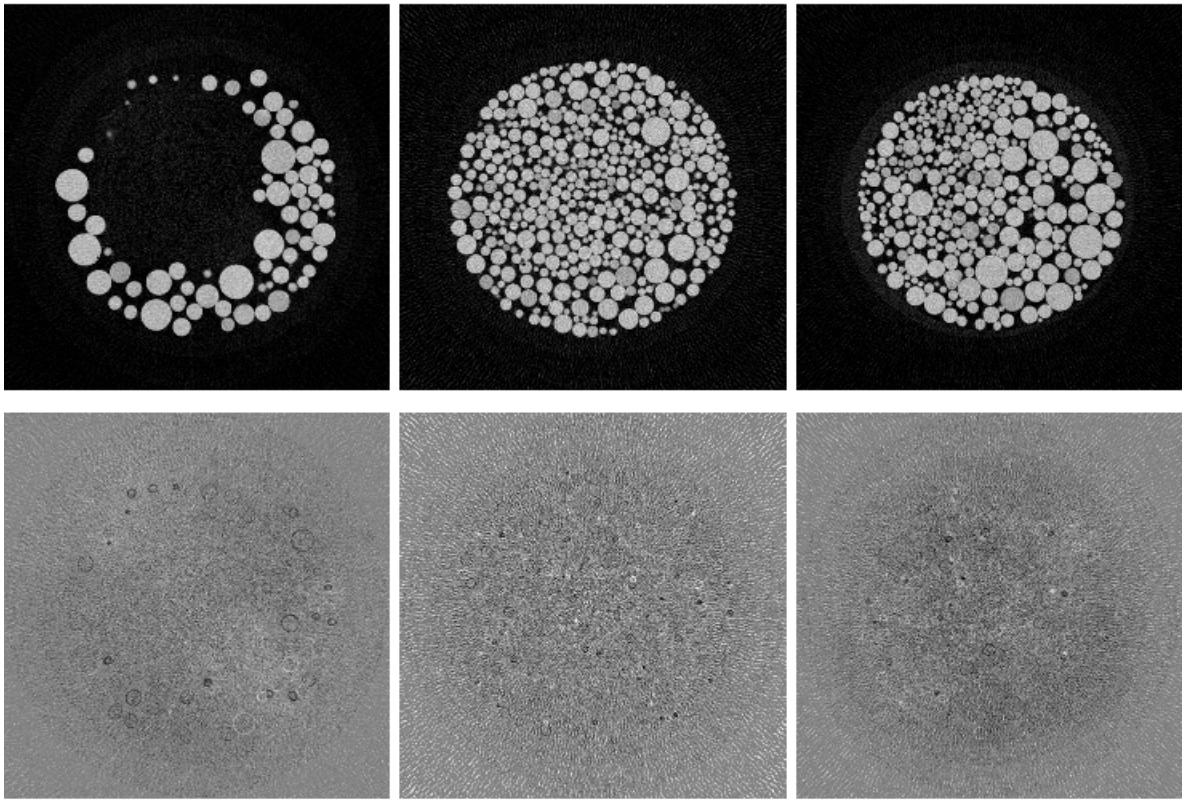
SSD = 2000 (11.66 mm)

cb_8 : 19049.314339 sec. / 1 view

cb_ss : 503.320563 sec. / 360 views



fdk : 222.743216 sec. / 722 slices



fdk - 040711j/byte/180.tif

fdk - 040711j/byte/360.tif

fdk - 040711j/byte/540.tif

Date: Tue, 21 Mar 2017 19:11:43 +0900
From: Tsukasa NAKANO
To: Tsukasa NAKANO
Cc: Kentaro Uesugi, Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: Re: ray_sum_CB-CT_simulator

うえすぎさま、
なかのです。すみません。大ボケでした。cb_ss の処理時間（500 秒程度）は 360 投影の値なので、以前の cb_8 による 1 投影の処理時間（約 19050 秒）に比べると 19050/(500/360)=13716 → cb_ss は cb_8 より 1 万倍以上高速です。

Date: Thu, 23 Mar 2017 19:26:04 +0900
From: Tsukasa NAKANO
To: Kentaro Uesugi
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: PB_and_CB-CT_HiPic_images

うえすぎさま、

GSJ/AIST のなかのです。先日紹介した単純な ray sum 計算による PB- および CB-CT simulators のプログラム（pb_3d と cb_ss）で撮影した画像（物体像）の X 線投影画像群を SPring-8 での通常の X 線 CT 実験の測定データに準じたものに変換する C-shell scripts "pb_hp.csh" と "cb_hp.csh" を書きました。

起動法

```
csh   pb_hp.csh   parameters.csh
csh   cb_hp.csh   parameters.csh
```

これらのスクリプトは起動時に指定されたテキストファイル（parameter.csh）を C-shell script として内部的に実行し、その中で「変数（後述）」に設定されているパラメータ値を用いて X-ray CT simulators による画像の撮影処理などを行います。以下が parameter.csh の具体的な内容です。

pb_hp.csh 用の parameter.csh（後述するファイル 040711j_pb.txt の内容）

```
@ threads = 8
set image=040711j/byte
set nbs=-
set i_xy=`echo 0.000583 ¥* 3.662520 / 125 | bc -l`
@ views = 360
```

```
@ pv_d = ( 1 << 8 ) - 1
@ pv_i = ( 1 << 14 ) - 1
set hp=040711j_pb
```

cb_hp.csh（040711j_cb.txt）

```
@ threads = 8
set image=040711j/byte
set nbs=-
set i_xy=`echo 0.000583 ¥* 3.662520 / 125 | bc -l`
set i_z=$i_xy
set ssd=`echo $i_xy ¥* 2000 | bc -l`
set sdd=$ssd
@ views = 360
@ pv_d = ( 1 << 8 ) - 1
@ pv_i = ( 1 << 14 ) - 1
set hp=040711j_cb
```

上記のように、parameter.csh の変数の設定法（と言うよりも C-shell script の変数の設定法）には設定値に応じた 2 通りの形式のものがあります。

```
@   変数名 = 設定値              # 設定値が整数の場合のみ
set   変数名 = 設定値
```

このような変数への設定値として
他の変数の設定値（$変数名)、
逆引用符「`」で囲んだコマンドの実行結果（それが標準出力に書き出す値)、
C-shell で実行可能な整数演算（「<<」は左シフト演算子です）の結果の値
を引用することができます。また、parameter.csh の記号「#」から行末まではコメントと見なされるので、そこに何を書いてもかまいません。いずれにせよ、parameter.csh は C-shell script なので、pb_hp.csh や cb_hp.csh の実行に必要なパラメータの設定法にはかなりの自由度があります。

さて、pb_hp.csh と cb_hp.csh 用のパラメータの値を設定しなければならない C-shell の変数の名前とそれらの設定内容は以下の通りです。

threads（整数値）
　X-ray CT simulators のマルチスレッド処理で使用するスレッド数
image
　撮影する 3 次元画像のスライス画像が入っているディレクトリの名前
nbs
　変数 image に設定したディレクトリの下の「スライス画像のファイルの名前（name)」と「その

上の画素値（整数値）と CT 値の対応関係を表す 1 次式

CT 値 = base + step × 画素値

の係数値 base および step」の合計 3 個の値を空白もしくはタブコード区切りで各行に並べたテキストファイル（NBS_file）の名前。これらの行の順に応じた位置にスライス画像を配置した 3 次元画像を撮影することになる。なお、nbs に「-（負の符号)」で始まる文字列（注 1）を設定すると、base = 0 かつ step = 1 だと見なされる（注 2）。

i_xy と i_z（ただし、i_z は cb_hp.csh でのみ設定が必要）

撮影する 3 次元画像のスライス面内の画素（正方形と仮定）の辺長とスライス間隔。

ssd および sdd（これらはどちらも cb_hp.csh に対してのみ設定が必要）

光源（点と仮定）とサンプル回転軸の距離（source-sample distance)、および、光源と検出器面の距離（source-detector distance）

views（整数値）

撮影する X 線投影画像の枚数（投影数)。PB-CT ではサンプル回転角の刻みは 180 / views 度に、また、CB-CT では 360 / views 度になる。

pv_d および pv_i（どちらも整数値）

作成する HiPic 形式の暗電流および入射 X 線強度（I0）画像の画素値。

hp

作成する HiPic 形式の画像（dakr.img と "q*.img"）および測定ログのファイル（output.log）を格納するディレクトリの名前。それが存在しなければ、新しいディレクトリを自動的に作成するようにしてある。

注 1

NBS_file の名前（変数 nbs の設定値）として以下の簡略記法による指定が可能です（ただし、最後のものを除いて、変数 image に設定したディレクトリの下にスライス画像以外のファイルがあるとエラーになります)。

-（1 個の負の符号）

変数 image のディレクトリの下のファイルを名前の英数字順に並べた NBS_file を指定したことにする。

--r（負の符号は 2 個）

ファイル名をその英数字の逆順に並べた NBS_file を指定。

--n（負の符号は 2 個）

ファイル名をそれに含まれる数値順に並べた NBS_file を指定。

--nr もしくは --rn（負の符号は 2 個）

ファイル名をそれに含まれる数値の逆順に並べた NBS_file。

-ファイル名（1 個の負の符号とファイル名）

そのディレクトリの下にある 1 個のファイルだけを選択。

注 2

すべてのスライス画像で base = 0 かつ step = 同じ値（df とします）なら、df を変数 i_xy や i_z に設定する画素の辺長の値に繰り込んで指定することができます。つまり、CT 値は画素の辺長の逆数の単位の値なので、画素の辺長の真の値を dxy と dz として、

i_xy の設定値 ＝ dxy × df
i_z の設定値 ＝ dz × df
とすればよろしい。先の例の 040711j_pb.txt と 040711j_cb.txt ではそれを行っています（画像 040711j/byte では df = 3.6652520 / 125）。

pb_hp.csh と cb_hp.csh は「作成した測定データからの画像再構成」に必要な以下の値をタブコード区切りで 1 行に並べて標準出力に書き出します。

pb_hp.csh の出力（5 個の数値が 1 行に並んでいる）
[1-3] 撮影した X 線投影画像の横・縦画素数と投影数
[4] スライス面内の正方形画素の辺長 Dr
[5] サンプル回転軸の位置 Or
cb_hp.csh の出力（9 個の数値）
[1-3] 撮影した X 線投影画像の横・縦画素数と投影数
[4] 光源とサンプル回転軸の距離（SSD）
[5] 光源と検出器面の距離（SDD）
[6] スライス面と平行な方向の検出器の間隔 Du
[7] サンプル回転軸に対応するスライス面と平行な方向の検出器の位置 Ou
[8] サンプル回転軸と平行な方向の検出器の間隔 Dw
[9] 検出器上で光源からの X 線が直交する点のサンプル回転軸の方向の位置 Ow

なお、これらのスクリプトが作る暗電流（dark.img）や pb_hp.csh の入射 X 線強度（I0）の HiPic 形式画像の画素値は変数 pv_d や pv_i の設定通りの一様な空間分布ですが、cb_hp.csh の I0 および透過 X 線強度（I）画像（q*.img）では検出器面への cone beam の「斜め入射」を考慮した空間分布に従う X 線強度値（画素値）になっています（ただし、画像の再再構成の処理では I / I0 を行うので、この空間的な X 線強度値の変動は打ち消されますが）。

以上のような pb_hp.csh と cb_hp.csh のコードを書庫ファイル
http://www-bl20.spring8.or.jp/~sp8ct/tmp/raysum.taz
に追加しておきました。また、その中には先に例として示したパラメータ値を設定した C-shell script のファイル 040711j_pb.txt と 040711j_cb.txt も入れてあります。そして、この E-mail に添付した C-shell script "raysum.csh.txt" を UNIX 端末から
csh raysum.csh.txt
と入力して実行すれば、今回のものを含む、書庫ファイル raysum.taz に入っているスクリプトによるすべてのデモを実際に試すことができます。

こちらで実行した pb_hp.csh と cb_hp.csh のデモ（or テスト）の出力内容をコメントとして埋め込んだ C-shell script "hp.txt" をこの E-mail に添付しました（書庫ファイル raysum.taz にも入れてあります）。UNIX 端末から
cd raysum

csh　hp.txt

と入力するだけで以下の一連の処理を実行します。

[1] 以下の入力により PB-CT simulator で撮影した測定データを作成
　csh　pb_hp.csh　040711j_pb.txt
[2] プログラム hp_cbp を使ってその測定データからもとの画像を再再構成
[3] その画像をもとのものと同じ画素数・画素値のビット数になるように加工

[4] 以下の入力により CB-CT simulator で撮影した測定データを作成
　csh　cb_hp.csh　040711j_cb.txt
[5] プログラム hp_fdk を使ってその測定データからもとの画像を再再構成
[6] その画像をもとのものと同じ画素数・画素値のビット数になるように加工

ただし、上の処理で実行する hp_cbp と hp_fdk は以前に cbp と fdk として紹介したものを改名したプログラムです。その詳しい話は以下の文書の最後の部分を御覧下さい。

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

hp.txt によるデモでは UNIX コマンド /usr/bin/time を使って主要な処理の実行に要した秒単位の時間を計測するようにしてあります。左端に２個の「#」が付いている hp.txt の行に記されている数値がこちらで実行した時の処理時間です。また、rhp.pdf にも書いたように、SPring-8 の X 線 CT 実験で得た測定データを用いた画像再構成プログラム hp_cbp と hp_fdk を従来の浮動小数点数画素値の TIFF 画像（float-TIFF）だけではなく通常の整数画素値の TIFF 画像（integer-TIFF）を出力できるように改造したので、その機能を試すべく hp.txt を改造したスクリプトを作成し実行してみました。オリジナルの hp.txt に埋め込まれている処理時間とその改造版のスクリプトの処理時間は以下のようになりました。

PB-CT の処理時間

	pb_hp	hp_cbp	trim	byte	cbp+trim+byte
f	186.14	154.27	66.02	39.66	259.95
i	191.51	207.84	34.58	80.17	322.59

CB-CT の処理時間

	cb_hp	hp_fdk	trim	byte	fdk+trim+byte
f	547.87	200.10	66.27	41.06	307.43
i	547.18	252.23	35.42	81.79	369.44

ここで、左端が "f" の行の値がオリジナルの hp.txt に埋め込まれている処理時間、また、"i" が改造版の値です。ラベル pb_hp と cb_hp の欄の値は pb_hp.csh と cb_hp.csh の処理時間で、これらは "f" と "i" でまったく同じ処理内容ですが、参考のためにそれぞれの処理時間を載せました。その後に実行した hp_cbp と hp_fdk は "f" では float-TIFF、"i" では integer-TIFF のファイルを出力しています。また、trim と byte の欄の値はそれぞれ別のプログラムで実行した float-TIFF と integer-TIFF に対する画像領域のトリミングと

８ビット画素値の画像への変換に要した処理時間です。そして、各行の最後に "f" と "i" で異なる形式の TIFF 画像を対象とした３つの処理の合計の処理時間の値を示しました。

驚くべきことに "f" と "i" の行の最後の合計の処理時間は "i" の方が長いです。つまり、再構成画像を float-TIFF に書き込んだ後のトリミングや画素値の変換に長い処理時間がかかるのは float-TIFF のファイル I/O が遅いためではないようです。

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 raysum.csh.txt

```
# installation of programs in raysum.taz

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/raysum.taz
tar xzf raysum.taz
cd raysum
make
cd ..
rm raysum.taz

# installation of program "sg2tg" for lenna.txt

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/sp8ct.taz
tar xzf sp8ct.taz
cd sp8ct/src
make sg2tg
cd ../..
mv sp8ct/src/sg2tg raysum
rm -r sp8ct sp8ct.taz

# installation of programs "hp_cbp" and "hp_fdk" for pb_hp.csh and cb_hp.csh

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/rhp.taz
tar xzf rhp.taz
cd rhp
make hp_cbp hp_fdk
cd ..
mv rhp/hp_cbp rhp/hp_fdk raysum
rm -r rhp rhp.taz

# installation of image "lenna.tif" for lenna.txt

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/lenna.taz
tar xzf lenna.taz lenna/lenna.tif
```

```
mv lenna/lenna.tif raysum
rm -r lenna lenna.taz

# installation of image "040711j/byte/*.tif" for 040711j.txt and cb_test.txt

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/040711j_byte.taz
tar xzf 040711j_byte.taz
mkdir raysum/040711j
mv 040711j_byte raysum/040711j/byte
rm 040711j_byte.taz
```

添付ファイル hp.txt

```
# number of threads for hp_cbp and hp_fdk

    setenv THREADS 8

# PB-CT

    cat 040711j_pb.txt
#       #
#       @ threads = 8
#       set image=040711j/byte
#       set nbs=-
#       set i_xy=`echo 0.000583 ¥* 3.662520 / 125 | bc -l`
#       @ views = 360
#       @ pv_d = ( 1 << 8 ) - 1
#       @ pv_i = ( 1 << 14 ) - 1
#       set hp=040711j_pb

    csh pb_hp.csh
#       usage : pb_hp.csh parameter.csh

    /usr/bin/time -f %e ¥
    csh pb_hp.csh 040711j_pb.txt
#       1415    720     360     .00001708199328000000  707
##      186.14

### xv 040711j_pb/*.img

    hp_cbp
#       usage : hp_cbp HiPic/ Dr Or {layer1 layer2} RA0 {BPS} TG_format

    mkdir pb_tg

    /usr/bin/time -f %e ¥
```

```
    hp_cbp 040711j_pb .00001708199328000000 707 0 pb_tg/%03d.tif >pb_tg.log
#       1415    720     .00001708199328000000
##      154.27

#    (1415-1000)/2=207.5
#    207+1000-1=1206

    /usr/bin/time -f %e ¥
    trim_float pb_tg - 207 207 1206 1206 pb_tg
##      66.02

    /usr/bin/time -f %e ¥
    t2t_float pb_tg - 0 1 8 pb_tg >/dev/null
##      39.66

### eog 040711j/byte/360.tif pb_tg/360.tif

# CB-CT

    cat 040711j_cb.txt
#       #
#       @ threads = 8
#       set image=040711j/byte
#       set nbs=-
#       set i_xy=`echo 0.000583 ¥* 3.662520 / 125 | bc -l`
#       set i_z=$i_xy
#       set ssd=`echo $i_xy ¥* 2000 | bc -l`
#       set sdd=$ssd
#       @ views = 360
#       @ pv_d = ( 1 << 8 ) - 1
#       @ pv_i = ( 1 << 14 ) - 1
#       set hp=040711j_cb

    csh cb_hp.csh
#       usage : cb_hp.csh parameter.csh

    /usr/bin/time -f %e ¥
    csh cb_hp.csh 040711j_cb.txt
#       1512    1114    360     ¥
#       .03416398656000000000  .03416398656000000000  ¥
#       .00001708199327999999  755.5   ¥
#       .00001708199327999999  556.5
##      547.87
```

```
### xv 040711j_cb/*.img

    hp_fdk
#       usage : hp_fdk HiPic/ A B Du Ou Dw Ow {layer1 layer2} RA0 {BPS} TG_format

    mkdir cb_tg

    /usr/bin/time -f %e ¥
    hp_fdk 040711j_cb .03416398656000000000 .03416398656000000000 ¥
                      .00001708199327999999 755.5 ¥
                      .00001708199327999999 556.5 ¥
                      0 cb_tg/%03d.tif >cb_tg.log
#       1413    721     1.708199e-05    1.708199e-05
##      200.10

#   (1413-1000)/2=206.5
#   206+1000-1=1205

    /usr/bin/time -f %e ¥
    trim_float cb_tg - 206 206 1205 1205 cb_tg
##      66.27

    /usr/bin/time -f %e ¥
    t2t_float cb_tg - 0 1 8 cb_tg >/dev/null
##      41.06

### eog 040711j/byte/360.tif cb_tg/360.tif
```

Date: Mon, 27 Mar 2017 16:03:50 +0900
From: Tsukasa NAKANO
To: Kentaro Uesugi
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: CB-CT_照明中心

うえすぎさま、

GSJ/AIST のなかのです。3/23 の E-mail で紹介した「CB-CT simulator で撮影した投影画像群を SPring-8 での通常の X 線 CT 実験の測定データファイルに準じたものに変換する C-shell script "cb_hp.csh"」を少しだけ改造しました。それが出力する HiPic 形式の入射および透過 X 線強度画像の値に cone beam の「斜め入射」だけでなく X 線 flux の「幾何学拡散」の寄与を加味しました。

On Thu, 23 Mar 2017 19:26:04 +0900 Tsukasa NAKANO wrote:
> なお、これらのスクリプトが作る暗電流（dark.img）や pb_hp.csh の入射 X 線
> 強度（I0）の HiPic 形式画像の画素値は変数 pv_d や pv_i の設定通りの一様な
> 空間分布ですが、cb_hp.csh の I0 および透過 X 線強度（I）画像（q*.img）では
> 検出器面への cone beam の「斜め入射」を考慮した空間分布に従う X 線強度の
> 値（画素値）になっています（ただし、画像の再再構成の処理では I / I0 を行う
> ので、この空間的な X 線強度値の変動は打ち消されますが）。

今、cone beam の光源（点光源と仮定）から出た X 線が垂直に入射する検出器面の上の点を「照明中心」と呼びます。cb_hp.csh のパラメータ SDD（source-detector distance）はこのような点光源と照明中心の間の距離で、また、検出器の画素の辺長（Du と Dw）を単位とする照明中心の座標値が Ou と Ow です。ただし、ここでは検出器面内の点の位置を「照明中心を原点とする実寸の座標値（u, w)」 で表します。cb_hp.csh は「中央の座標値が（u,w）の I0 画像の画素」の入射 X 線強度の測定値 I0(u,w）を以下の式で計算しています。

I0(u,w) = (SDD / L)^2 × cos(Θu) × cos(Θw)

ただし、

L = √(SDD^2 + u^2 + w^2)
　光源と「検出器の画素の中央の点（u,w)」の距離。光源から距離 L の位置での「光源を見込む単位立体角あたりの X 線 flux」は幾何学拡散のために L^2 に反比例した値になる。
cos(Θu) = SDD / √(SDD^2 + u^2) および
cos(Θw) = SDD / √(SDD^2 + w^2)
　中央の位置が（u,w）の検出器の画素に斜め入射する X 線の方向余弦の u および w 成分値。斜め入射する X 線の受光に関与する画素の実質的な断面積はもとの値 Du×Dw にこれら 2 つの方向余弦を乗じた値になる。

また、同様に、cb_hp.csh は「CB-CT simulator で得た画素の中央の点における投影値 p(u,w)」から以下の式で「その画素の透過 X 線強度の測定値 I(u,w）」を計算しています。

I(u,w) = I0(u,w) × exp{ - p(u,w) }

cb_hp.csh を含む X-ray CT-simulators に関する一連の話は以上で終わりで、以下は余談です。実際の CB-CT 装置で撮影した時、現状では撮影した画像からその照明中心の位置を推定できません。ただ、上にも書いたように、理論的には CB-CT の入射 X 線強度は照明中心を中心とする同心円状の空間分布になるはずなので、それを使って照明中心の位置を推定できる可能性があります。そこで、2015 年 3 月に SPring-8 の医療棟で測定した CB-CT のデータを調べてみました。結論だけ言うと "1503*" の測定画像はノイズレベルが高すぎて照明中心の位置の推定に使えそうにありません。測定 "1503*" では暗電流（dark）と I0 画像として 30 枚づつの測定画像を積算・平均していますが、それらの画素値の標準偏差は非常に大きく、平均の画像も雑音だらけでした。測定 "1503*" の積算・平均した I0 と dark の差分画像の画素値の値域を参考のため以下

に示します。下限の負の値が上限の正の値に匹敵する絶対値になっている測定も多々あります。

測定番号	最初に撮影した I0 - dark の画素値	最後の I0 – dark の画素値
150303a	-438.733337 ~ 4273.666992	-2029.599976 ~ 2803.232910
150303b	-472.399902 ~ 3528.233398	-1513.799805 ~ 2115.733215
150303d	-1547.300293 ~ 1524.000046	-2255.499512 ~ 2219.099609
150303e	135.533340 ~ 4598.699219	-879.766663 ~ 2711.000000
150303f	-5298.766602 ~ 2150.233398	-5111.366211 ~ 2987.033203
150312a	-19.433350 ~ 2349.866699	-2307.433594 ~ 2802.233246
150312b	-2108.599609 ~ 1585.800049	-1747.000000 ~ 2077.133423
150312c	-1991.266602 ~ 1756.666504	-902.033203 ~ 3789.066895
150312d	-4043.033203 ~ 1969.000000	-1645.833496 ~ 2656.599960
150312e	-219.533325 ~ 1412.833374	-1546.800293 ~ 782.266678

とりあえず以上です。

Date: Tue, 28 Mar 2017 14:54:08 +0900
From: Tsukasa NAKANO
To: Kentaro Uesugi
Cc: Masayuki Uesugi, Akihisa TAKEUCHI, Masato Hoshino, "TSUCHIYAMA, Akira", MATSUNO Junya, ogawa.motohiro.44a
Subject: Ray_sum_E-mails

うえすぎさま、

GSJ/AIST のなかのです。単純な ray sum 計算による X-ray CT simulators のプログラム群に関する E-mails を加筆・修正して 1 個の PDF にまとめました。

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

また、それらの実行を準備する 3/21 の E-mail に添付した boot strap 用スクリプト raysum.csh.txt もアップロードしておきました。

http://www-bl20.spring8.or.jp/~sp8ct/tmp/raysum.csh.txt

これらをご自由にお使い下さい。とり急ぎ、

Date: Tue, 11 Apr 2017 23:52:25 +0900
From: Tsukasa NAKANO
To: MATSUNO Junya, 中村隆太
Cc: Kentaro UESUGI
Subject: MOUSE

まつのくん、
なかむらくん、

なかのです。計算機 MOUSE の上で

wget http://www-bl20.spring8.or.jp/¥~sp8ct/tmp/raysum.csh.txt
csh raysum.csh.txt
cd raysum
csh cb_test.txt
cd ..

と入力して

http://www-bl20.spring8.or.jp/sp8ct/tmp/raysum.pdf#page=16

で紹介している、

[1] cone beam（CB）CT simulator、cb_ss
[2] FDK 法による CB CT 画像の再構成プログラム、fdk
[3] float-TIFF の画像のトリミングプログラム、trim_float
[4] float-TIFF から普通の integer-TIFF への変換プログラム、t2t_float

を順次実行するデモ用の C-shell script "cb_test.txt" を走らせてみました。それらの秒単位の処理時間は以下の通りです（比較用に他の計算機の処理時間も載せました）：

host	cb_ss	fdk	trim_float	t2t_float
gsjgix	500.15	227.90	65.27	39.55
gsjvix	683.04	356.32	147.91	87.29
vrm	862.97	492.73	117.80	60.61
DBR73	1158.70	923.27	517.07	292.44
pr47	2363.90	1360.47	158.80	99.02
MOUSE	532.37	410.84	49.27	34.53

MOUSE の CPU の処理速度と SSD の I/O 速度はどちらもぼくの主力計算機 gsjgix 並みですね。とり急ぎ、