# 引き継ぎ

山田 祐也

2020 年 3 月 23 日

## 1　目的

山田の研究の引き継ぎ。次のことを引き継いで、今後の研究がスムーズに進むようにする。

- 修論後研究集会までに検討することになっていたもの
- モンテカルロシミュレーションに使用したソフトウェアの利用法
- その他参考になろうもの

## 2　修論後の検討 (しようと思っていたものも含めて)

### 2.1　定常状態への到達

修論時点では tMCimg による定常状態への到達は定性的に判断していたので、これを定量的に評価し、より正確なシミュレーション結果の比較を行う。Fluence(内部分布) の各 Voxel の値に対して、2 時刻間の最大変化率を見ることで定常状態に達しているかを判定できると考える。

$$\text{Max Difference}(\%) = \max_{\text{Voxel}} \frac{\text{Fluence at } t_2 - \text{Fluence at } t_1}{\text{Fluence at } t_1} \times 100$$

上記を 200ps から 2000ps まで、200ps 刻みで計算した結果は次の表と図の通り。どの程度の Max Difference までを許容するかは別途議論の必要がある。

### 2.2　領域の形状による影響

修論では直方体領域に対するシミュレーションを考えていたが、MCML では円筒座標を用いているので実際にはこの直方体に内接する円柱領域でのシミュレーションになっていた。元々MCML では無限平面を考えることになっていて、とはいえ現実にその領域でシミュレーションを行うことは不可能なので適当に切ってシミュレーションすることになる。そこで、設定の簡便さと、適当な大きさの直方体領域を考えれば問題なくシミュレーションできるものと考えていたこととで、直方体領域でのシミュレーションをすることにしていたのだが、観測したい領域に対してどれほど十分な大きさの直方体領域を取れば問題ないと言えるのかという点については考察が不足していた。そのため、修論時にやったものより大きい領域でシミュレーションする (観測する領域は変えずに)、$y = x$ の平面で切った結果を見るなどしてこの点をさらに議論する必要がある。

Table 1: Max Difference

| Time(ps) | Max Difference(%) |
|---|---|
| 400 | 100 |
| 600 | 29.36 |
| 800 | 11.49 |
| 1000 | 10.23 |
| 1200 | 12.61 |
| 1400 | 6.96 |
| 1600 | 4.84 |
| 1800 | 4.24 |
| 2000 | 2.87 |

# 3 ソフトウェア手引き

## 3.1 MCML

### 3.1.1 Download

`https://omlc.org/software/mc/`
上記サイトから MCML と CONV をそれぞれダウンロードし、適当に展開する。

### 3.1.2 Installation

上記を展開したディレクトリでそれぞれ以下を実行する。
(for MCML installation): cc -o mcml mcmlio.c mcmlnr. mcmlgo.c mcmlmain.c
(for CONV installation): cc -o conv convi.c convo.c convnr.c conviso.c convconv.c convmain.c

### 3.1.3 Usage

- MCML
  インプットファイル '*.mci' を作成する。サンプルがあるので基本はそれに倣って設定すれば良い。
- CONV
  MCML のアウトプット '*.mco' を処理してグラフ描画用のデータを出力する。infinitely narrow beam に対するシミュレーション結果 (コマンド 'oo') と Gaussian Beam などを入力としたシミュレーション結果 (コマンド 'oc' ) が出力できる。

## 3.2 tMCimg

### 3.2.1 Download

`http://www.nmr.mgh.harvard.edu/PMI/resources/tmcimg/index.htm`
計算機環境に合ったものをダウンロードする。

### 3.2.2 Installation

基本的には上記 URL に従ってインストールする。山田の環境 (Apple LLVM version 9.1.0 (clang-902.0.39.2)) では下記のようにはじめ make が通らなかったが修正して通るようになった。

```
=>make
gcc -I.. -g -O2 -c doconfig.c
gcc -I.. -g -O2 -c doscale.c
gcc -I.. -g -O2 -c fpusetup.c
gcc -I.. -g -O2 -c hisfile.c
gcc -I.. -g -O2 -c moveSD.c
gcc -I.. -g -O2 -c ran.c
gcc -I.. -g -O2 -c segfile.c
segfile.c:79:15: warning: format specifies type 'int' but the argument has type 'unsigned long' [-Wformat]
p->nxstep * p->nystep * p->nzstep * sizeof(TISSUE)); 1 warning generated.
gcc -I.. -g -O2 -c storage.c
storage.c:45:11: error: assignment to cast is illegal, lvalue casts are not supported
return ((void *)p = alloc_vector(n1, nb));
1 error generated.
make[1]: *** [storage.o] Error 1
```

そこで該当箇所 libmccore/storage.c:45 を次のように修正した。

```
    return p = alloc_vector(n1, nb);
```

これで make が通った。

### 3.2.3 Usage

基本は ‘sample.cfg‘ に倣って設定する。infinitely narrow beam を用いた定常状態のシミュレーションの場合には、‘freq‘ と ‘source[:rad]‘ は 0 にする。シミュレーションの実行時間は ‘start_time‘, ‘gate_width‘, ‘ngate‘ で指定する。‘gate_width * ngate‘ がシミュレーション時間になる。それぞれの値がもつ意味は下記の内部シミュレーション結果の解釈を参照。別途各 voxel に属性を割り振る binaly file を用意する必要があるが、これも doc に従ってやれば良い。

### 3.2.4 アウトプットの解釈

tMCimg によるアウトプットはそのままでは使えないので、別途変換する必要がある。内部のシミュレーション結果 (‘*.2pt‘) と境界のシミュレーション結果 (‘*.his‘) では若干処理方法が異なるので注意を要する。アウトプットの形式は doc を参照されたいが以下に要約する。

- 境界シミュレーション結果
  境界相当の Voxel に入った photon について、その座標、到達時刻、内部 Voxel の各属性のものをどれほど通ったかを 32bit floating point で記録する。例えば、2 層からなるモデル (layer1, larye2) があったとしたら、境界シミュレーションのアウトプットは

  x, y, z, total time traveled by a photon, total length traveled by the photon in layer 1, total length traveled by the photon in layer 2

  が並んでいることになる.

- 内部シミュレーション結果
  内部シミュレーションの結果は、Voxel 毎、‘gate_width‘ 毎に 64bit floating point で記録する。例えば、定常状態のシミュレーションにおいて総 Voxel 数が 10 で、‘gate_width = 10ps, ngate = 2‘ とすると、シミュレーションは 20ps まで行われ、そのアウトプットは 20 個の数値

  Fluence in Voxel 0 at 10ps, Fluence in Voxel 1 at 10ps, ..., Fluence in Voxel 9 at 10ps, Fluence in Voxel 0 at 20ps, Fluence in Voxel 1 at 20ps, ..., Fluence in Voxel 9 at 20ps

  が並ぶ。なので、特に事情がなければ ngate = 1 としておくのがよいと思われる。これらは全部バイナリで書き出されるので、 ‘hexdump‘するなどして適切に変換する。

#### 3.2.5 Mersenne Twister の使い方

tMCimg は GSL(Gnu Scientific Library) 経由で Mersenne Twister を利用できる。これを利用するには事前に GSL を使えるようにしておく必要がある。導入方法と動作確認は例えば以下を参照。

`http://hooktail.sub.jp/computPhys/gsl/install.html`

続いて tMCimg ディレクトリ内の config.h:86 に以下を記載する。

#define HAVE_GSL 1

これで tMCimg は libmccore/ran.c の中で GSL 経由の MT を利用しようとする。

最後に再び make する。但し、実行前に tMCimg/Makefile を以下の通り修正する。

```
#before
30: GSLLIBS =
49: >-gcc $(CFLAGS) -o tMCimg tMCimg.lo $(LDFLAGS) $(LIBS)


    #after
30: GSLLIBS = -lgsl -lgslcblas -lm
49: >-gcc $(CFLAGS) -o tMCimg tMCimg.lo $(LDFLAGS) $(LIBS) $(GSLLIBS)
```