Joint ICTP-IAEA Workshop on Monte Carlo Radiation Transport and Associated Data Needs for Medical Applications
28 October – 8 November 2024
ICTP, Trieste, Italy

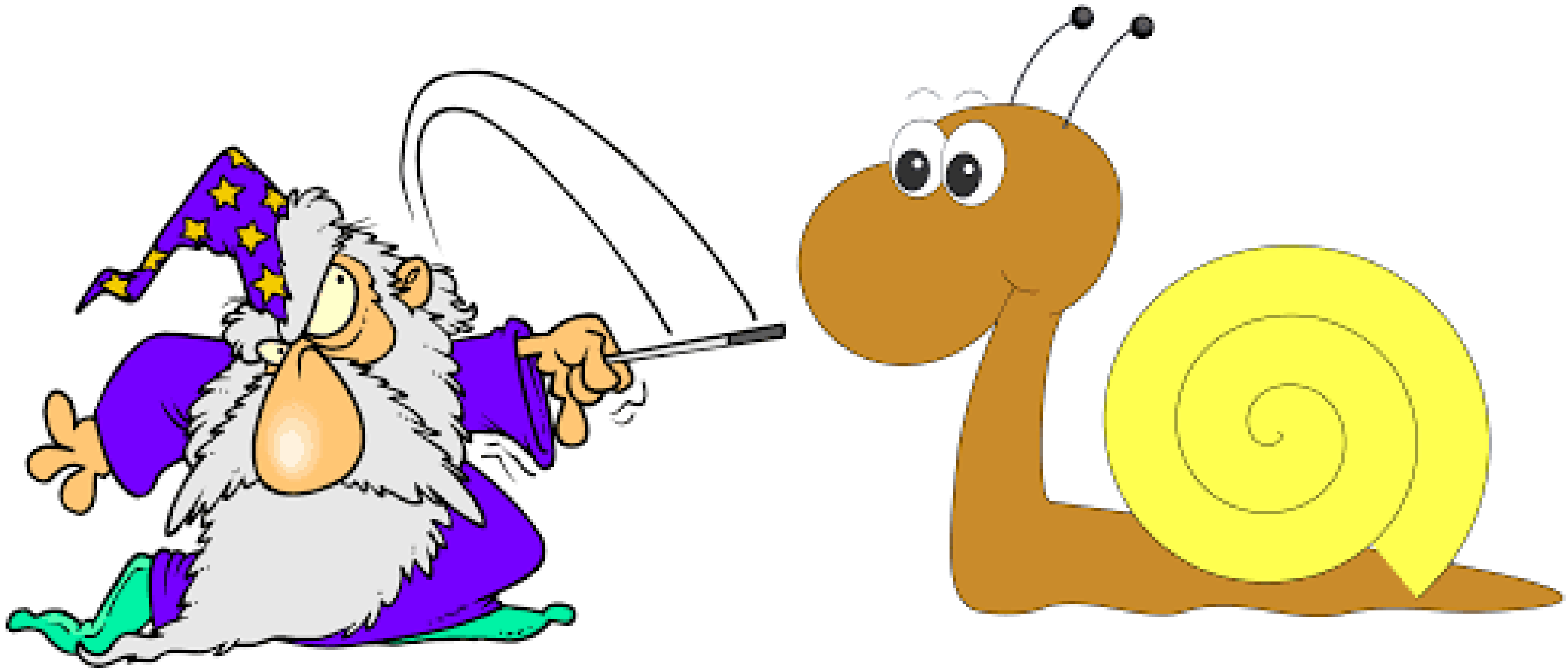
Lecture 13

Variance reduction techniques

Iwan Kawrakow

Senior Director of Science
ViewRay Inc.

The black art of fast Monte Carlo simulations



The black art of fast Monte Carlo simulations

Hello (Monte Carlo) 12d world!

More dimensions = more efficient

$$r^2 = \sum_{i=1}^{d} x_i^2$$

Beyond 4 dimensions, Monte Carlo wins !

Monte Carlo

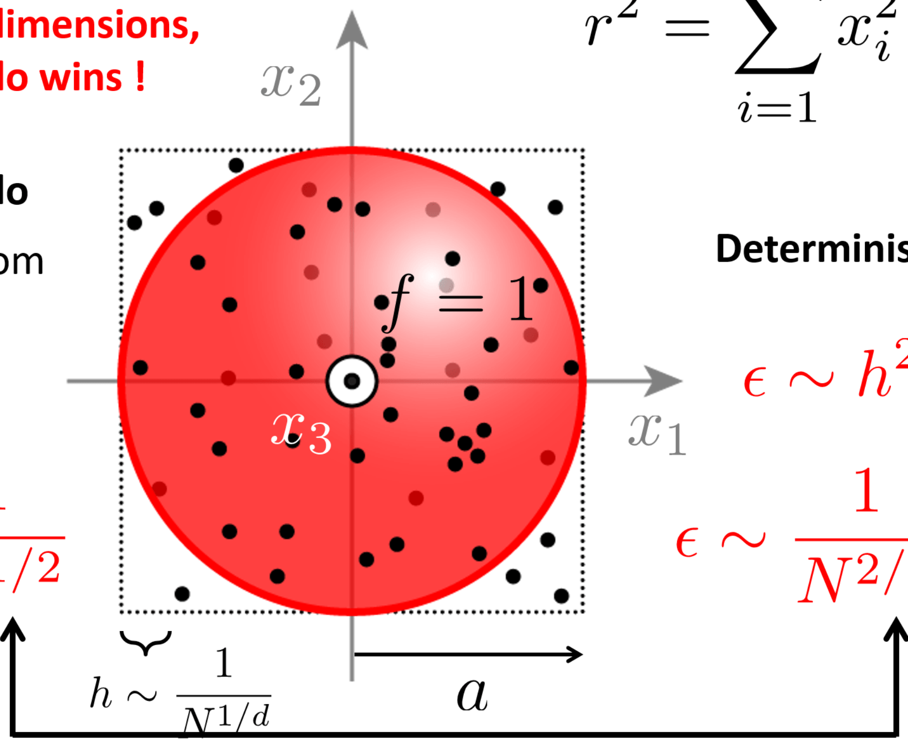
It is a random sampling process, so we have

$$\epsilon \sim \frac{1}{N^{1/2}}$$

Deterministic

$$\epsilon \sim h^2$$

$$\epsilon \sim \frac{1}{N^{2/d}}$$

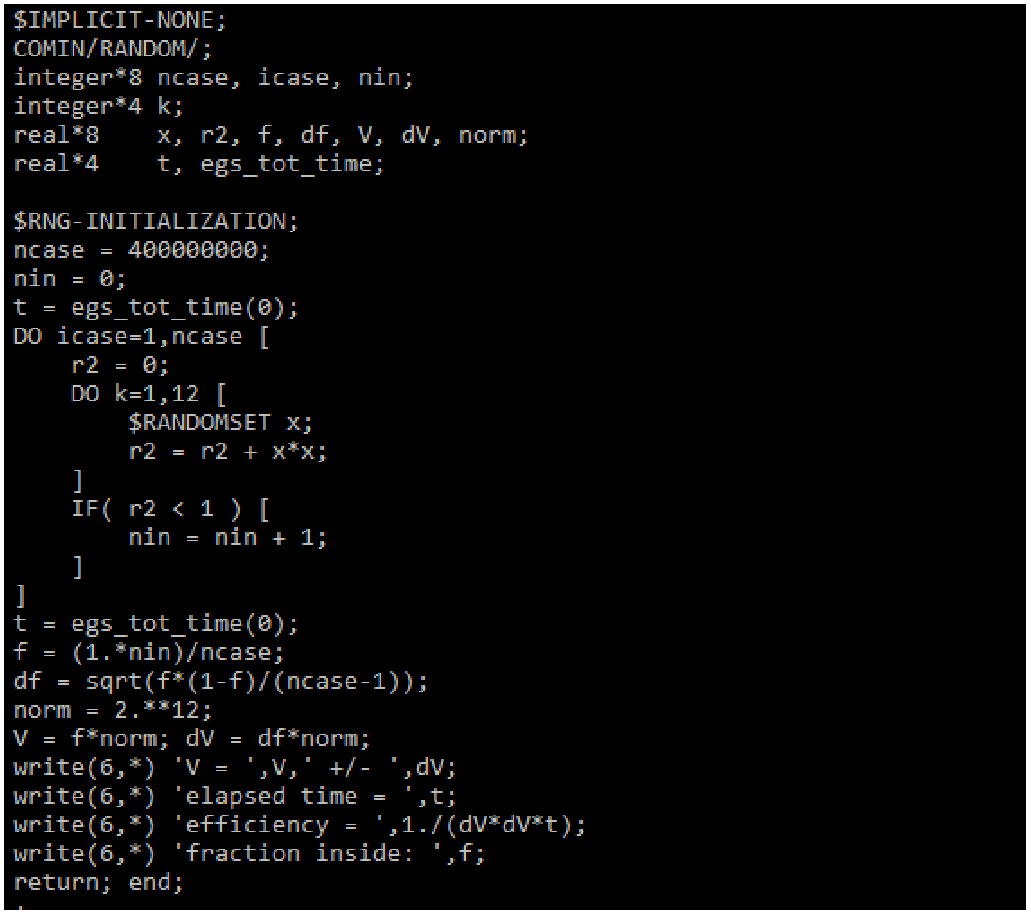


Definition of efficiency

$$\varepsilon = \frac{1}{T\sigma^2}$$

- σ is an estimate of the uncertainty of the quantity of interest, *e.g.*
  - The uncertanty of the 12d sphere volume
  - The uncertainty of the photon fluence in a region of interest
  - The uncertainty of the dose at $d_{\max}$
  - ...
- T is the total CPU time for the calculation.
- As $\sigma^2 \propto 1/N$ and $T \propto N$, the efficiency is independent of the number of samples N used to determine it
- Efficiency comparisons only make sense for the same situation on the same hardware

Version 0

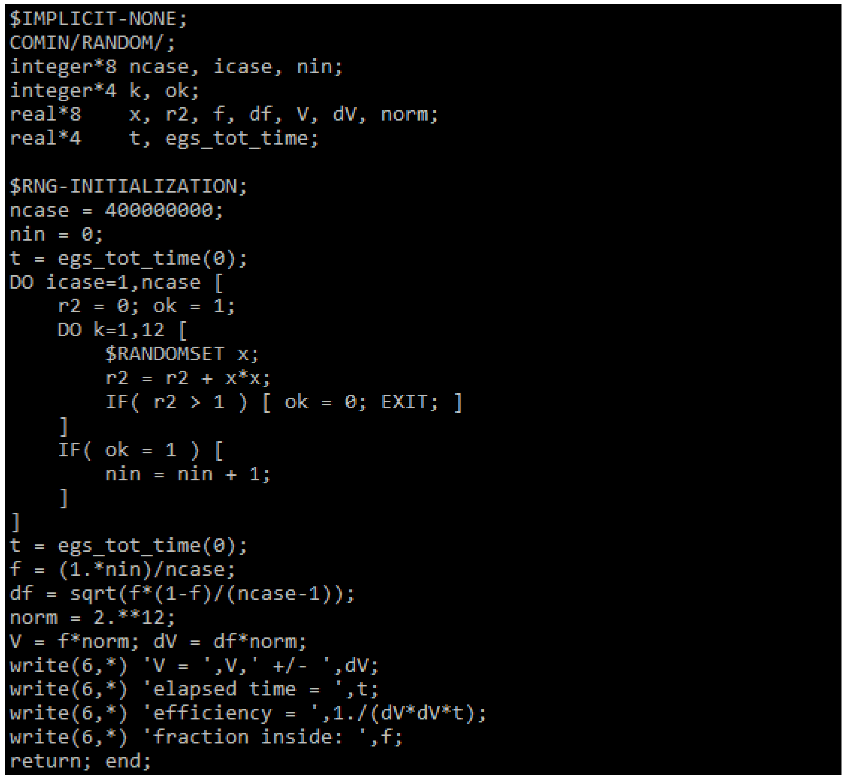


```
$IMPLICIT-NONE;
COMIN/RANDOM/;
integer*8 ncase, icase, nin;
integer*4 k;
real*8    x, r2, f, df, V, dV, norm;
real*4    t, egs_tot_time;

$RNG-INITIALIZATION;
ncase = 400000000;
nin = 0;
t = egs_tot_time(0);
DO icase=1,ncase [
    r2 = 0;
    DO k=1,12 [
        $RANDOMSET x;
        r2 = r2 + x*x;
    ]
    IF( r2 < 1 ) [
        nin = nin + 1;
    ]
]
t = egs_tot_time(0);
f = (1.*nin)/ncase;
df = sqrt(f*(1-f)/(ncase-1));
norm = 2.**12;
V = f*norm; dV = df*norm;
write(6,*) 'V = ',V,' +/- ',dV;
write(6,*) 'elapsed time = ',t;
write(6,*) 'efficiency = ',1./(dV*dV*t);
write(6,*) 'fraction inside: ',f;
return; end;
```

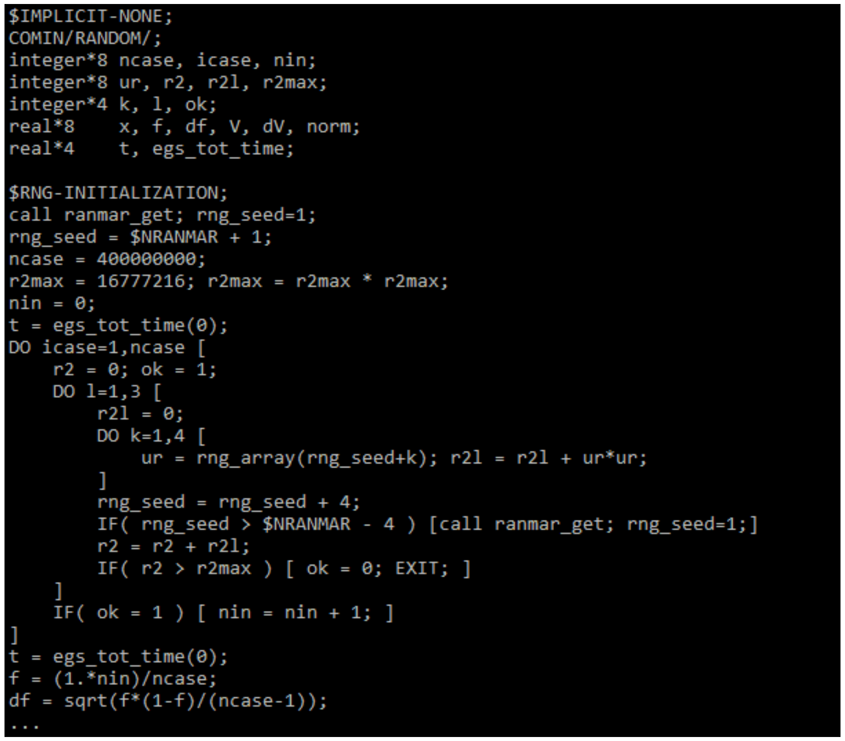
$$\varepsilon = 2.5 \times 10^3\,\mathsf{s}^{-1} \equiv \varepsilon_0$$

Versions 1 & 2



```
$IMPLICIT-NONE;
COMIN/RANDOM/;
integer*8 ncase, icase, nin;
integer*4 k, ok;
real*8    x, r2, f, df, V, dV, norm;
real*4    t, egs_tot_time;

$RNG-INITIALIZATION;
ncase = 400000000;
nin = 0;
t = egs_tot_time(0);
DO icase=1,ncase [
    r2 = 0; ok = 1;
    DO k=1,12 [
        $RANDOMSET x;
        r2 = r2 + x*x;
        IF( r2 > 1 ) [ ok = 0; EXIT; ]
    ]
    IF( ok = 1 ) [
        nin = nin + 1;
    ]
]
t = egs_tot_time(0);
f = (1.*nin)/ncase;
df = sqrt(f*(1-f)/(ncase-1));
norm = 2.**12;
V = f*norm; dV = df*norm;
write(6,*) 'V = ',V,' +/- ',dV;
write(6,*) 'elapsed time = ',t;
write(6,*) 'efficiency = ',1./(dV*dV*t);
write(6,*) 'fraction inside: ',f;
return; end;
```

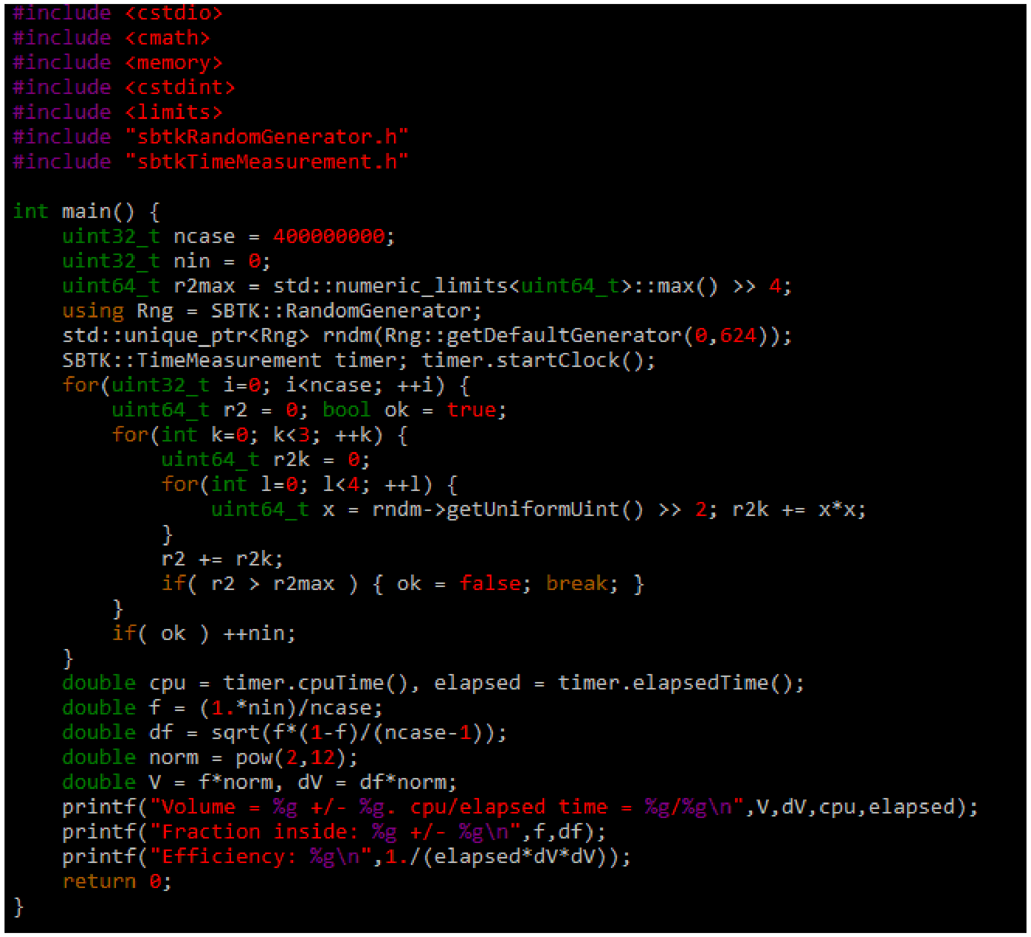


```
$IMPLICIT-NONE;
COMIN/RANDOM/;
integer*8 ncase, icase, nin;
integer*8 ur, r2, r2l, r2max;
integer*4 k, l, ok;
real*8    x, f, df, V, dV, norm;
real*4    t, egs_tot_time;

$RNG-INITIALIZATION;
call ranmar_get; rng_seed=1;
rng_seed = $NRANMAR + 1;
ncase = 400000000;
r2max = 16777216; r2max = r2max * r2max;
nin = 0;
t = egs_tot_time(0);
DO icase=1,ncase [
    r2 = 0; ok = 1;
    DO l=1,3 [
        r2l = 0;
        DO k=1,4 [
            ur = rng_array(rng_seed+k); r2l = r2l + ur*ur;
        ]
        rng_seed = rng_seed + 4;
        IF( rng_seed > $NRANMAR - 4 ) [call ranmar_get; rng_seed=1;]
        r2 = r2 + r2l;
        IF( r2 > r2max ) [ ok = 0; EXIT; ]
    ]
    IF( ok = 1 ) [ nin = nin + 1; ]
]
t = egs_tot_time(0);
f = (1.*nin)/ncase;
df = sqrt(f*(1-f)/(ncase-1));
...
```

$$\varepsilon = 3.8 \times 10^3\,\mathsf{s}^{-1} \approx 1.5 \times \varepsilon_0 \qquad \varepsilon = 8.2 \times 10^3\,\mathsf{s}^{-1} \approx 3.3 \times \varepsilon_0$$

Version 3 (C++, single-threaded)

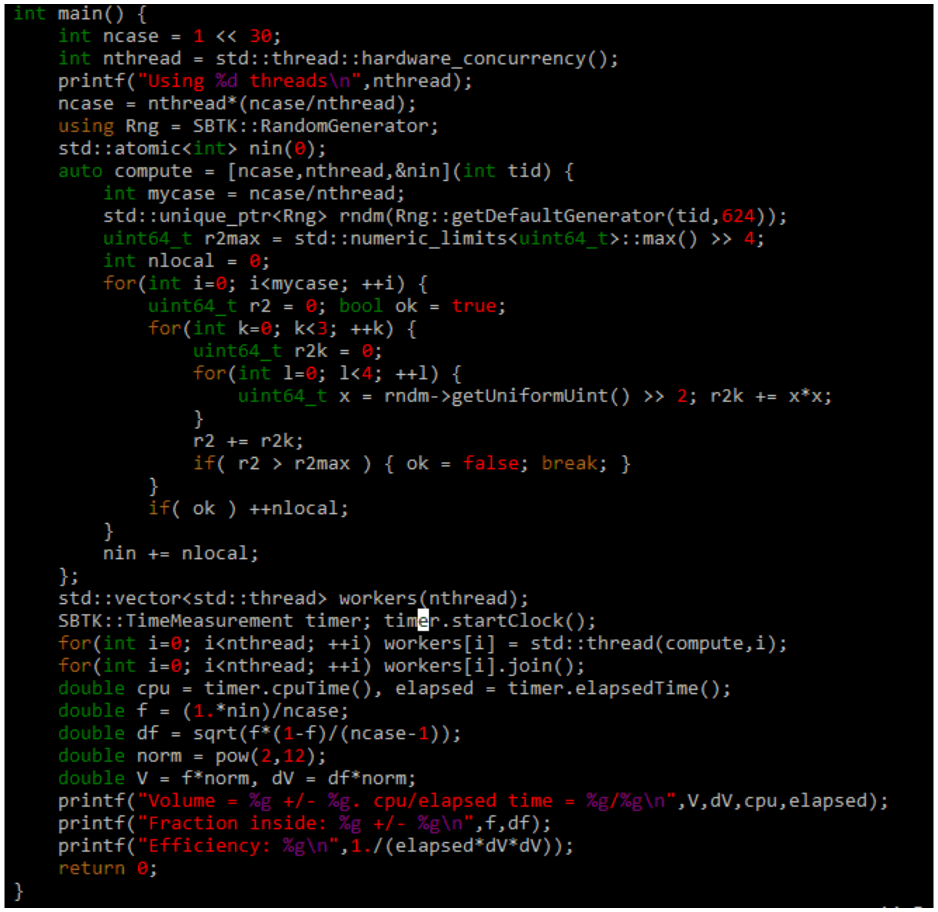


```
#include <cstdio>
#include <cmath>
#include <memory>
#include <cstdint>
#include <limits>
#include "sbtkRandomGenerator.h"
#include "sbtkTimeMeasurement.h"

int main() {
    uint32_t ncase = 400000000;
    uint32_t nin = 0;
    uint64_t r2max = std::numeric_limits<uint64_t>::max() >> 4;
    using Rng = SBTK::RandomGenerator;
    std::unique_ptr<Rng> rndm(Rng::getDefaultGenerator(0,624));
    SBTK::TimeMeasurement timer; timer.startClock();
    for(uint32_t i=0; i<ncase; ++i) {
        uint64_t r2 = 0; bool ok = true;
        for(int k=0; k<3; ++k) {
            uint64_t r2k = 0;
            for(int l=0; l<4; ++l) {
                uint64_t x = rndm->getUniformUint() >> 2; r2k += x*x;
            }
            r2 += r2k;
            if( r2 > r2max ) { ok = false; break; }
        }
        if( ok ) ++nin;
    }
    double cpu = timer.cpuTime(), elapsed = timer.elapsedTime();
    double f = (1.*nin)/ncase;
    double df = sqrt(f*(1-f)/(ncase-1));
    double norm = pow(2,12);
    double V = f*norm, dV = df*norm;
    printf("Volume = %g +/- %g. cpu/elapsed time = %g/%g\n",V,dV,cpu,elapsed);
    printf("Fraction inside: %g +/- %g\n",f,df);
    printf("Efficiency: %g\n",1./(elapsed*dV*dV));
    return 0;
}
```

$$\varepsilon = 12.5 \times 10^3 \approx 5 \times \varepsilon_0$$

Version 4 (C++, multi-threaded)



```
int main() {
    int ncase = 1 << 30;
    int nthread = std::thread::hardware_concurrency();
    printf("Using %d threads\n",nthread);
    ncase = nthread*(ncase/nthread);
    using Rng = SBTK::RandomGenerator;
    std::atomic<int> nin(0);
    auto compute = [ncase,nthread,&nin](int tid) {
        int mycase = ncase/nthread;
        std::unique_ptr<Rng> rndm(Rng::getDefaultGenerator(tid,624));
        uint64_t r2max = std::numeric_limits<uint64_t>::max() >> 4;
        int nlocal = 0;
        for(int i=0; i<mycase; ++i) {
            uint64_t r2 = 0; bool ok = true;
            for(int k=0; k<3; ++k) {
                uint64_t r2k = 0;
                for(int l=0; l<4; ++l) {
                    uint64_t x = rndm->getUniformUint() >> 2; r2k += x*x;
                }
                r2 += r2k;
                if( r2 > r2max ) { ok = false; break; }
            }
            if( ok ) ++nlocal;
        }
        nin += nlocal;
    };
    std::vector<std::thread> workers(nthread);
    SBTK::TimeMeasurement timer; timer.startClock();
    for(int i=0; i<nthread; ++i) workers[i] = std::thread(compute,i);
    for(int i=0; i<nthread; ++i) workers[i].join();
    double cpu = timer.cpuTime(), elapsed = timer.elapsedTime();
    double f = (1.*nin)/ncase;
    double df = sqrt(f*(1-f)/(ncase-1));
    double norm = pow(2,12);
    double V = f*norm, dV = df*norm;
    printf("Volume = %g +/- %g. cpu/elapsed time = %g/%g\n",V,dV,cpu,elapsed);
    printf("Fraction inside: %g +/- %g\n",f,df);
    printf("Efficiency: %g\n",1./(elapsed*dV*dV));
    return 0;
}
```

$\varepsilon = 84.2 \times 10^3\,\text{s}^{-1} \approx 33.7 \times \varepsilon_0$ (8 threads on a 4-core laptop)

Can we do better?

$$V = \int\limits_{-1}^{1} \mathrm{d}x_1 \cdots \mathrm{d}x_{12} \Theta\left(1 - x_1^2 - \cdots - x_{12}^2\right) \tag{1.1}$$

$$= 2^{12} \int\limits_{0}^{1} \mathrm{d}x_1 \cdots \mathrm{d}x_{12} \Theta\left(1 - x_1^2 - \cdots - x_{12}^2\right) \tag{1.2}$$

$$= 2^{24} \int\limits_{0}^{1} \frac{\mathrm{d}x_1}{2\sqrt{x_1}} \cdots \frac{\mathrm{d}x_{12}}{2\sqrt{x_{12}}} \Theta\left(1 - x_1^2 - \cdots - x_{12}^2\right) \left(\prod_{i=1}^{12} x_i\right)^{1/2} \tag{1.3}$$

$$= 2^{24} \int\limits_{0}^{1} \mathrm{d}\eta_1 \cdots \mathrm{d}\eta_{12} \Theta\left(1 - \eta_1^4 - \cdots - \eta_{12}^2\right) \left(\prod_{i=1}^{12} \eta_i\right) \tag{1.4}$$

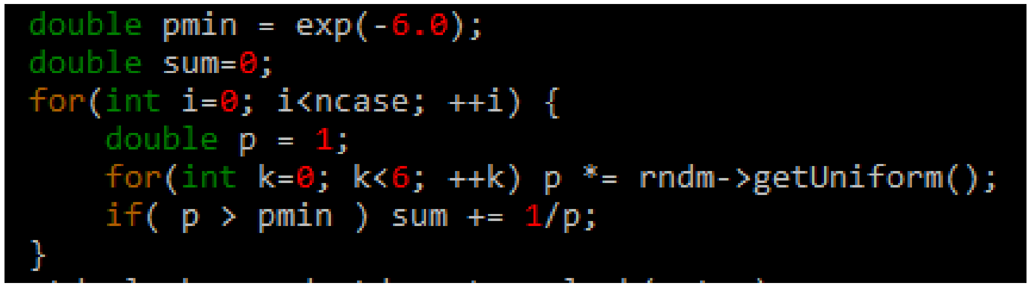
$$\varepsilon = 362.1 \times 10^3\,\mathsf{s}^{-1} \approx 145 \times \varepsilon_0$$

Can we do even better?

$$V = 2^{12}\int_0^1 \mathrm{d}x_1\cdots\mathrm{d}x_{12}\Theta\left(1-x_1^2-\cdots-x_{12}^2\right) \tag{1.5}$$

$$= 2^{12}\int_0^\infty\int_0^{2\pi} r_1\mathrm{d}r_1\mathrm{d}\phi_1 e^{-6r_1^2}\cdots r_6\mathrm{d}r_6\mathrm{d}\phi_6 e^{-6r_6^2}\Theta\left(1-\sum_{i=1}^6 r_i^2\right)\exp\left(6\sum_{i=1}^6 r_i^2\right) \tag{1.6}$$

$$= \left(\frac{\pi}{6}\right)^6\int_0^1 \mathrm{d}\eta_1\cdots\mathrm{d}\eta_6\Theta\left(\prod_{i=1}^6\eta_i-e^{-6}\right)\left(\prod_{i=1}^6\eta_i\right)^{-1} \tag{1.7}$$

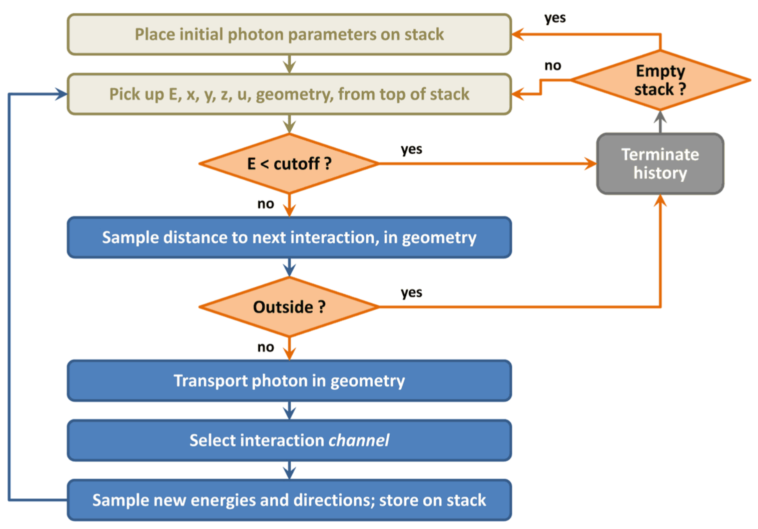

```
double pmin = exp(-6.0);
double sum=0;
for(int i=0; i<ncase; ++i) {
    double p = 1;
    for(int k=0; k<6; ++k) p *= rndm->getUniform();
    if( p > pmin ) sum += 1/p;
}
```

$$\varepsilon = 6.8 \times 10^7\,\mathsf{s}^{-1} \approx 27000 \times \varepsilon_0$$

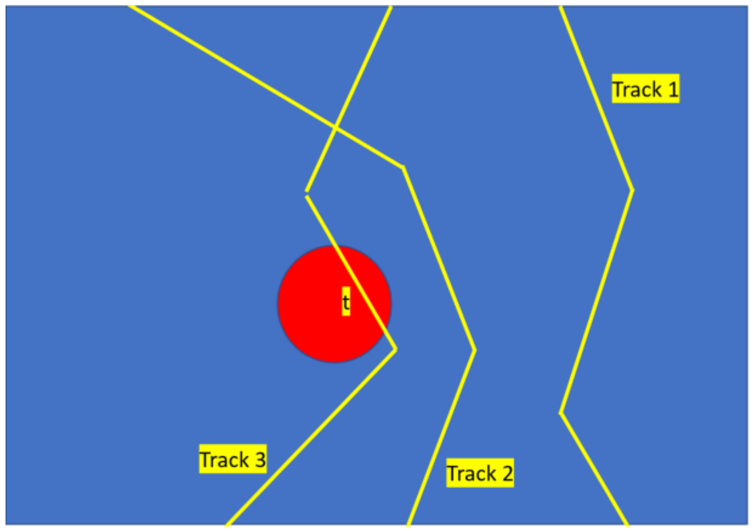
Hello (Monte Carlo) 12d world - Summary

- Speed-up compared to naive implementation: 27,000!
- Better coding: factor of ~ 5
- Parallel implementation: factor of ~ 6 (on 4-core laptop)
- Clever algorithm: factor of ~ 900

What do points in a sphere have in common with radiation transport?


Place initial photon parameters on stack
Pick up E, x, y, z, u, geometry, from top of stack
E < cutoff ?
yes
no
Sample distance to next interaction, in geometry
Outside ?
yes
no
Transport photon in geometry
Select interaction *channel*
Sample new energies and directions; store on stack
Empty stack ?
yes
no
Terminate history

Consider tracks with 2 interactions



Track 1: $(\lambda_1, \theta_1, \phi_1, \lambda_2, \theta_2, \phi_2) \rightarrow 0$
Track 2: $(\lambda_1, \theta_1, \phi_1, \lambda_2, \theta_2, \phi_2) \rightarrow 0$
Track 3: $(\lambda_1, \theta_1, \phi_1, \lambda_2, \theta_2, \phi_2) \rightarrow t$

$\Rightarrow$ Points in 6d space

There are many known VRTs

- Proper selection of available/development of new VRTs is the bread and butter of MC simulations
- Many papers since the veray early days of MC radiation transport
- Some commonly used tecniques are
  - Particle splitting
  - Russian Roulette
  - Interaction forcing
  - Exponential transforms
  - Correlated sampling
  - Importance and/or stratified sampling
  - Cross section enhancement
  - ...

VRT in the EGSnrc package

- Forcing photon interactions (BEAMnrc, CAVRZnrc)
- Bremsstrahlung splitting (BEAMnrc)
- Russian Roulette (BEAMnrc, cavity, egs_chamber)
- Bremsstrahlung cross section enhancement (BEAMnrc)
- Photon splitting (DOSXYZnrc, CAVRZnrc, cavity)
- Photon cross section enhancement (egs_chamber)
- Correlated sampling (egs_chamber)

Explorer

Explorer

Particle splitting

- In a MC simulation, one can split a particle into N identical particles at any time
- Each of the daughter particles gets $1/N$ of the statistical weight of the original particle
- Each daughter particle can then be transported separately thus improving the information gain
- Typical application: particles arriving in a region that is only rarely visited during a simulation or in combination with Russian Roulette
- Particle splitting is a true VRT, it does not modify the physics in any way

Russian Roulette (RR)

- RR is the reverse of particle splitting: at any time one can terminate a particle trajectory with a given probability p (*i.e.* play a RR game with the particle where the survival probability is p)
- If the particle survives, its statistical weight is increased by $1/p$.
- A particle surviving a RR game represents all other particles killed in the game
- Typical application: avoid transporting particles that contribute nothing or very little to the quantity of interest
- RR is a true VRT, it does not modify the physics in any way

Bremsstrahlung splitting

figures/brem_split-eps-converted-to.pdf

Approach A: $E_{after} = E_{in} - E_{\gamma 1}$ (or $E_{\gamma 2}$ or $E_{\gamma 3}$)

Approach B: $E_{after} = E_{in} - \overline{E}_{\gamma}$

Which is correct?

Bremsstrahlung splitting is simply Particle splitting + RR

Step 1 Split electron into N electrons, each having a weight of $1/N$

Step 2 Sample 1 bremsstrahulung photon for each of the N electrons $\Rightarrow N\ \gamma$'s

Step 3 Play RR with the electrons with $p = 1/N \Rightarrow$ 1 electron with weight 1 survives on average

$\Rightarrow$ Approach A is correct

Note: Energy conservation is only fulfilled *on average* and not on event-by-event basis

$\Rightarrow$ Not suitable if event-by-event energy conservation is important

Example: bremsstrahlung splitting in kV beams

figures/eff_rel_ubs_vs_dbs-eps-converted-to.pdf

From [Mainegra-Hing and Kawrakow, Med. Phys. 33, (2006) 2683]

Photon forcing

- Consider a photons passing through a geometry (or region) with a thickness of X mfp
- Fraction of photons interacting in the geometry will be $1 - e^{-X}$
- Fraction of photons leaving without interaction will be e^{-X}

$\Rightarrow$ Split photon into an interacting portion (weight $1 - e^{-X}$) and a non-interacting portion (weight e^{-X})

$\Rightarrow$ Transport the non-interacting portion to end of geometry (or region)

$\Rightarrow$ Force mfp to interaction to be between 0 and X for interacting portion

$$\gamma\text{-mfp} = -\ln(1 - \text{RN}(1 - e^{-X}))$$

Used *e.g.* in CAVRZnrc for simulations related to the primary air kerma standard.

Photon splitting

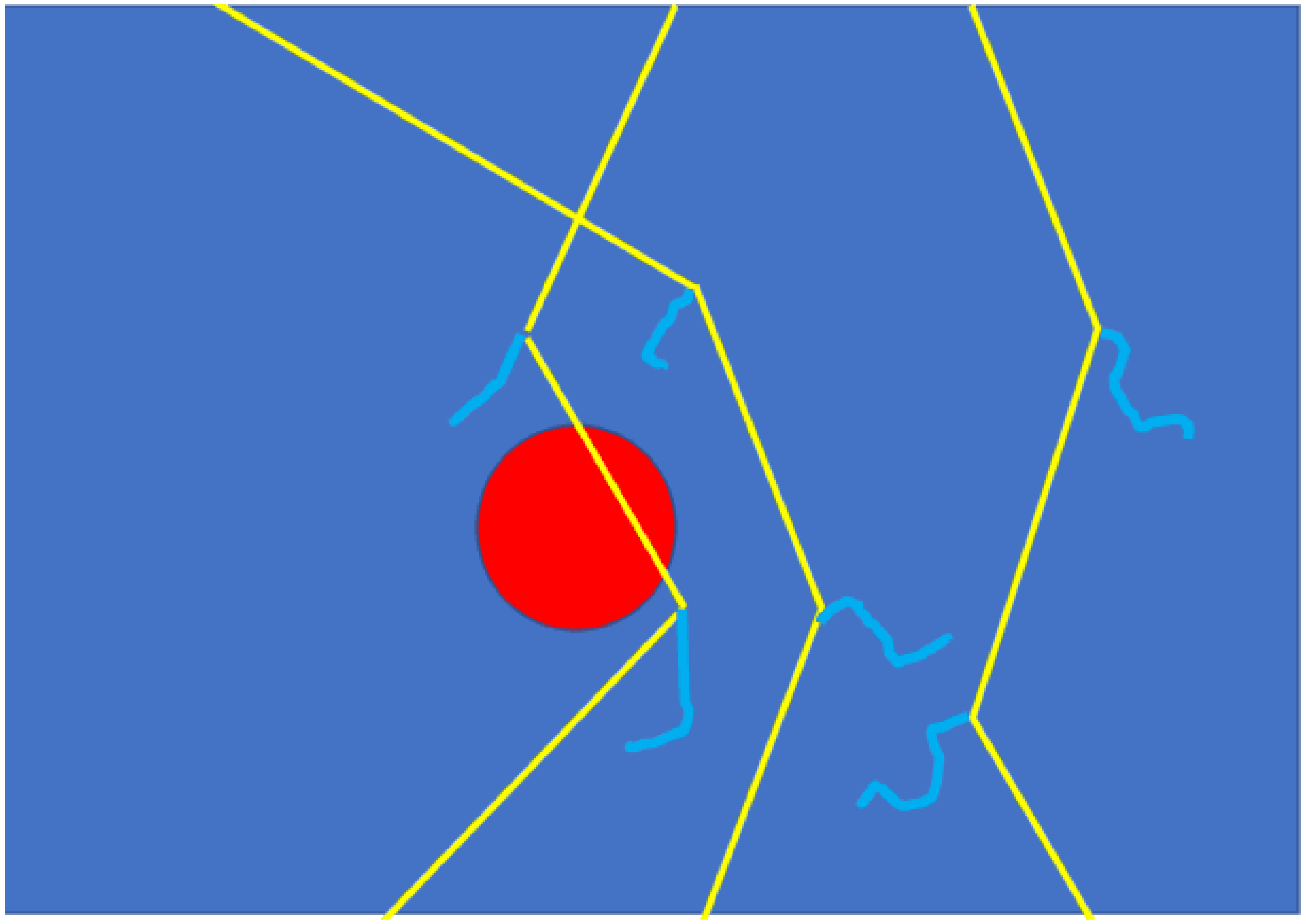
- λ is γ-mfp to an interaction. Normally $\lambda = -\ln(1-\eta)$
- With splitting on (splitting number N_s)

$$\lambda_i = -\ln\left(1 - \frac{\eta + i}{N_\mathrm{s}}\right) \ , \quad i = 0 \ldots N_\mathrm{s}$$

  i.e. we have interactions uniformly spread through the phantom and a single photon sets several electrons in motion

- Electrons have weight w/N_s (w = initial photon's weight)
- RR with scattered photons $\Rightarrow$ weight w if they survive
- Surviving scattered photons are split again
- Introduced for xVMC in Phys.Med.Biol **45** (2000) 2163
- Used in DOSXYZnrc, CAVRZnrc, cavity

Ion chamber simulations



Photon cross section enhancement (XCSE)

- Increase photon cross section in a (set of) region(s) by a factor of C
- When a photon interacts, split it into an interacting portion (weight $1/C$) and a non-interacting portion (weight $1 - 1/C$)
- Keep electrons set in motion (they have weight of w_0/C)
- Play RR with scattered photon(s) and non-interacting portion of initial photon, *i.e.*, if $\eta < 1/C$ keep scattered photon(s), else keep initial photon. Surviving photons have again weight w_0
- Used in egs_chamber
- Introduced in Med.Phys. 35 (2008) 1328

Photon cross section enhancement (XCSE)

figures/from_iwan_egs_chamber/tracks-eps-converted-to.pdf

Correlated sampling

figures/correlated-eps-converted-to.pdf

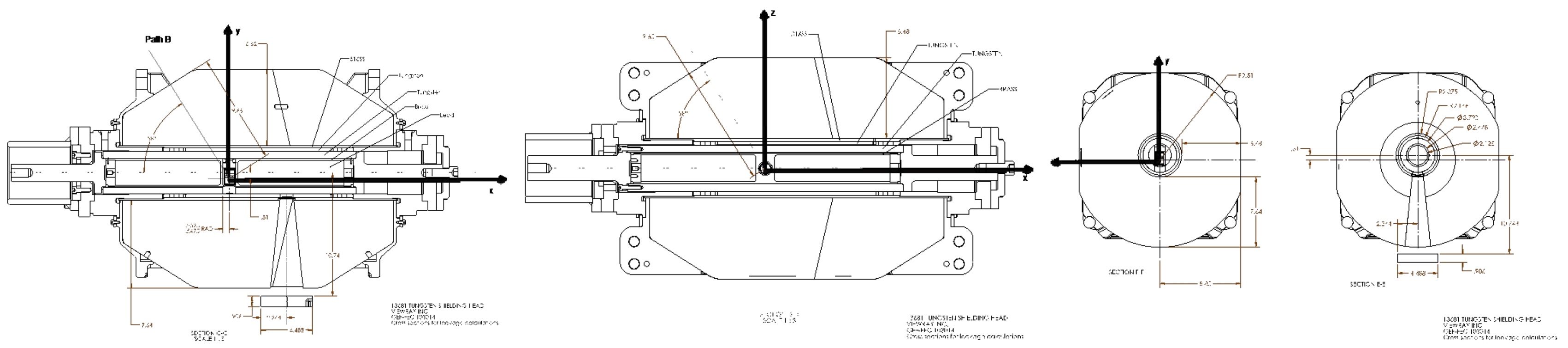
Efficiency gains due to

- Tracks not entering regions of interest only simulated in one geometry
- Correlation between tracks $\Rightarrow$ uncertainty on dose ratio can be significantly lower than individual dose uncertainties
- Available in egs_chamber

Approximate techniques

- Range rejection
- Use of high transport cutoff energies (ECUT, PCUT)
- Use of high secondary particle profuction thresholds (AE, AP)
- Use of simplified geometries
- The condensed history technique
- Use of approximate cross sections (*e.g.*, Klein-Nishina instead of bound Compton scattering, turn off spin effects in elastic scattering, etc.)
- Need to understand impact on accuracy very well!

Example: shielding calculations

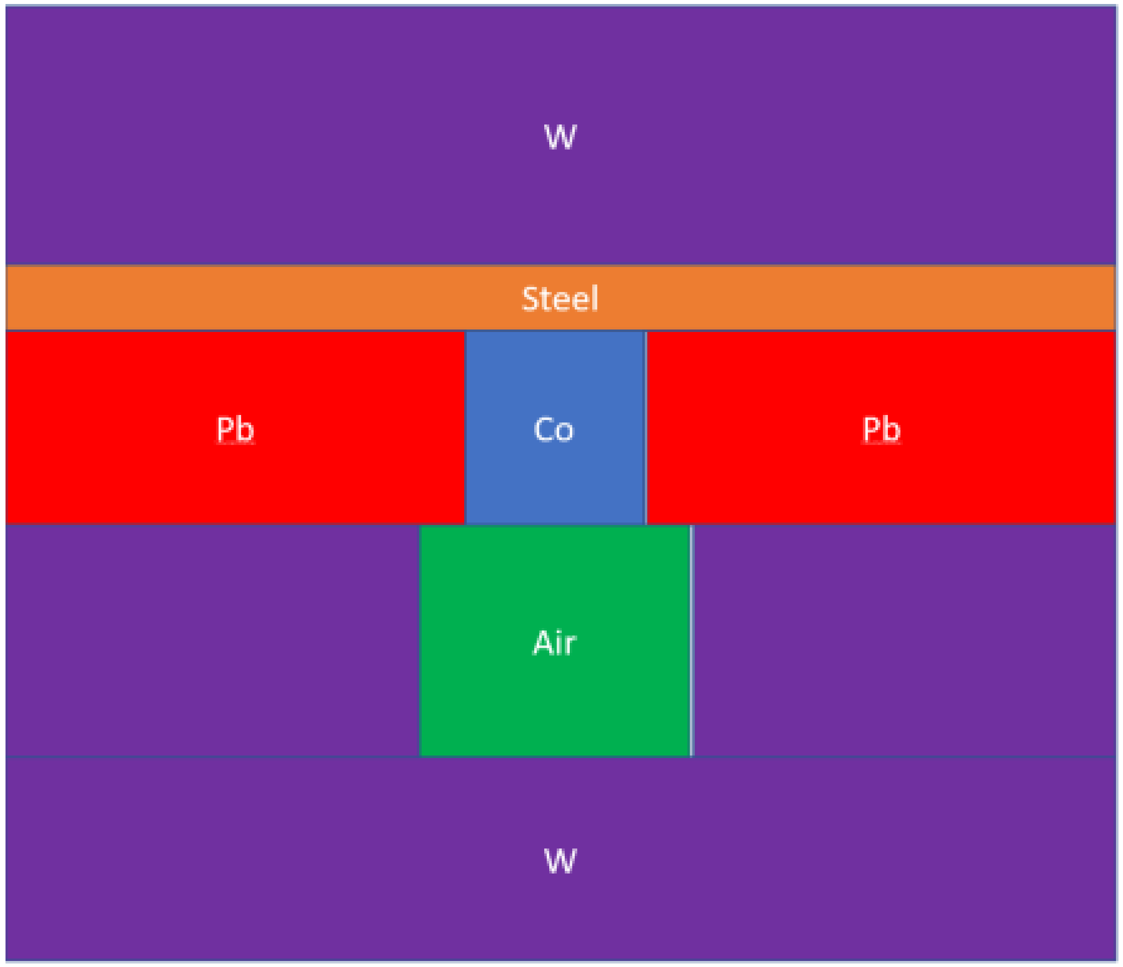


- 15,000 Ci source
- Original design: depleated uranium (DU)
- Design goal: replace DU with tungsten
- Goal: ≤ 1 mRad/h at 1 m from source to satisfy ICRP and IEC standards
- Constraint: new head must fit in existing gantry

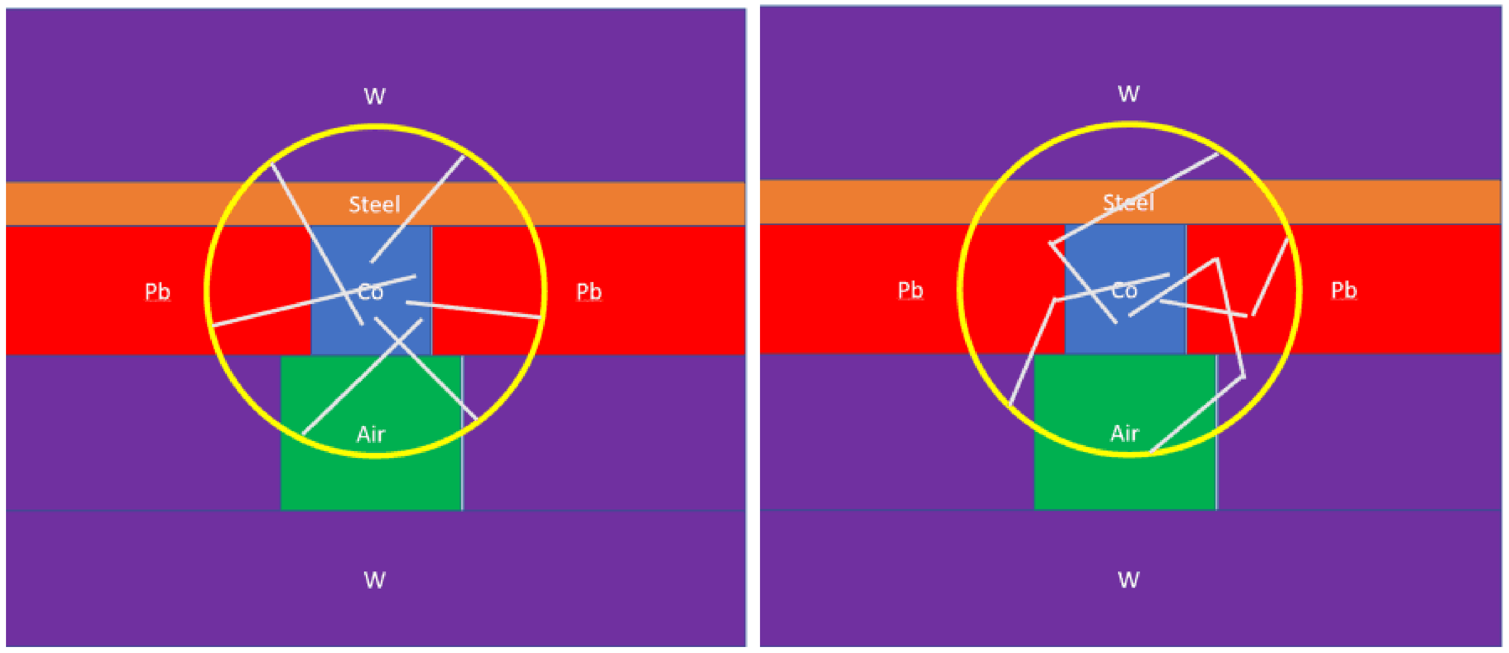
Seems Hopeless!

- ~ 1 out of 10^9 photons escape
- Need to simulate $\sim 10^{15}$ decays to get ~ 5% statistics in 25 cm^2 scoring regions at 1 m without using VRT
- All is scatter
- Highly heterogeneous geometry $\Rightarrow$ standard VRT's did not really work (at most a 10-fold acceleration)
- $\sim$ 2 year of computation on available computers
- Buy $500k+ cluster?

Divide et impera (Divide and Conquer)!



Divide et impera (Divide and Conquer)!



- Select N photons from source
- For each, place photon with weight $w_0 e^{-\lambda_i}$ at intersection with sphere
- Transport each photon. From each interaction, place scattered photon with weigth $w_0 e^{-\lambda_i}$ at intersection with sphere
- Discard transported photons when they reach sphere

Divide et impera (Divide and Conquer)!

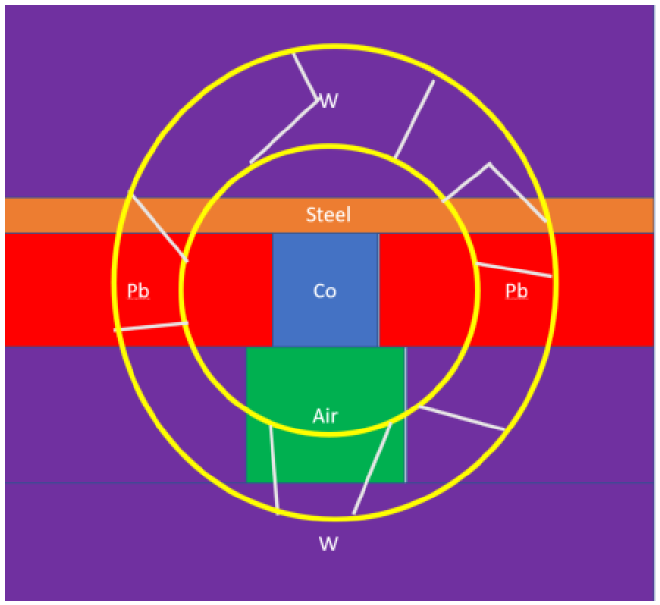
- Now have N_1 photons at first surface with weights w_i
- Compute average weight $\bar{w} = \sum w_i / N_1$
- Play RR with photons having $w_i < \bar{w}$ with survival probability $p = w_i / \bar{w}$
- Split photons with $w_i > \bar{w}$ into $w_i / \bar{w}$ photons

$\Rightarrow$ All photons now have weight $\bar{w}$

$\Rightarrow$ Photons “magically" concentrate along paths with less attenuation

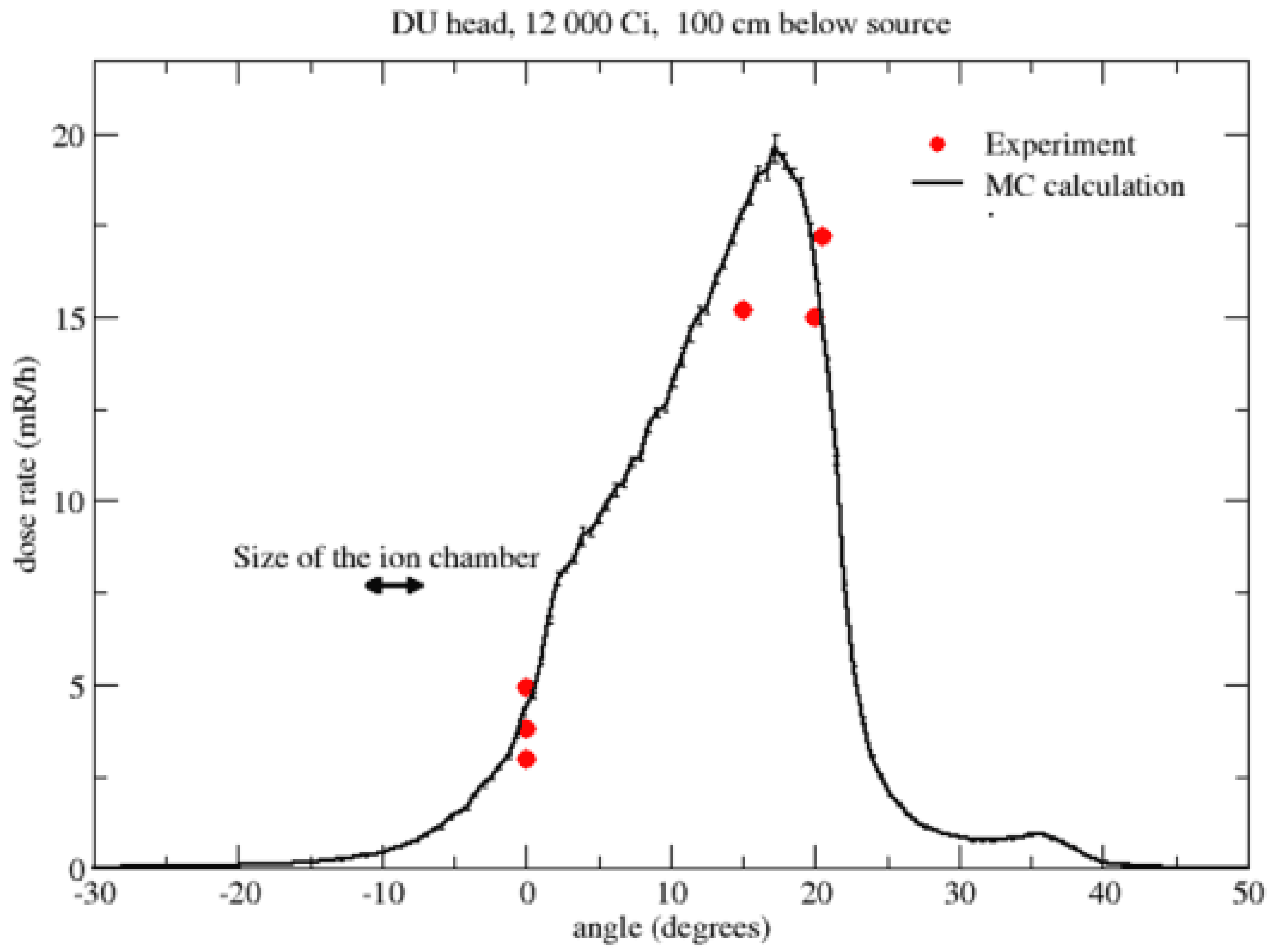
- Now add next surface
- Transport from first to second surface in the same way
- Repeat until exiting treatment head

Divide et impera (Divide and Conquer)!

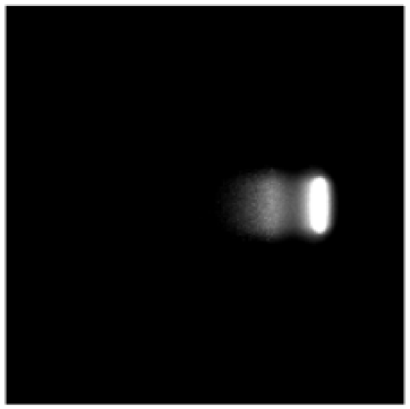


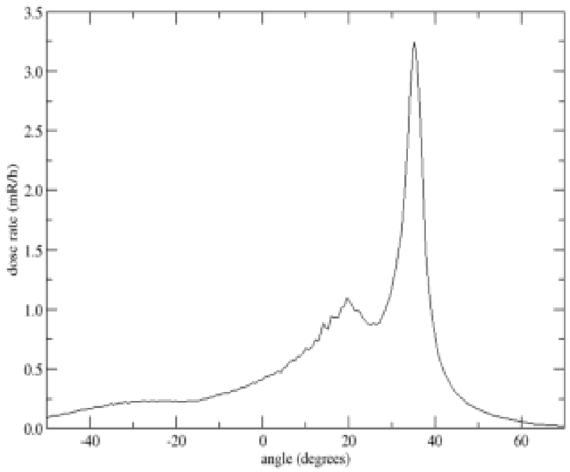
- 100 “bunches” of 10^6 photons are enough to get ~ 1% statistics!
- Note: each “bunch” is **one** statistically independent event
- 10-20 surfaces
- $\sim 10^6$ acceleration

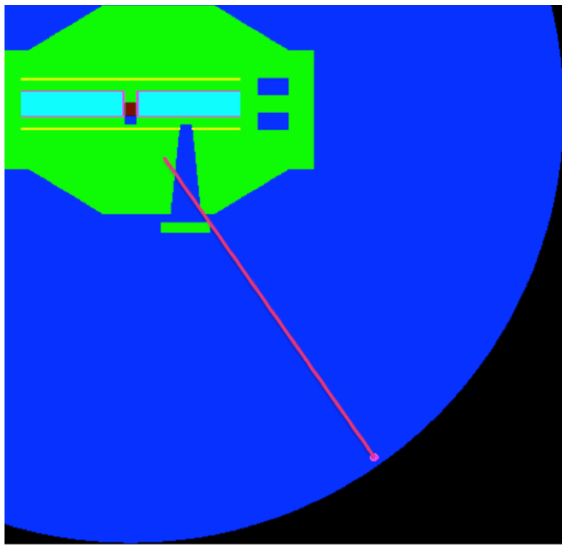
Comparison with measurements



Need to modify auxilary radiation shield







Summary

- VRT’s can increase the efficiency of MC simulations by orders of magnitude
- It is important to learn how to use the VRT’s available in the various EGSnrc user codes!