Assignment 2 Technical Report on Monte Carlo Framework

Introduction

This document is comprised of two interwoven parts. The first is a technical report detailing a Monte Carlo particle transport framework built by the author for the purpose of completing the second assignment of physics 539 at the University of British Columbia in 2012, but which has been designed for wider more general-purpose uses. The second part enumerates solutions to questions posed on the assignment. As these solutions are relevant for characterizing the framework, and a discussion of the solutions without accompanying discussion about the framework would be bland, it was decided to combine the two as "fluidly" as possible but to clearly indicate which parts are mostly relevant for the tired, over-worked marker.

The framework described herein is the result of many hours of careful design and painstaking experimentation. Upon learning that we would have a chance to work on such a beast, the author immediately set out to produce a quality product that would (hopefully) result in the establishment of the work as an open source, libre transport code. In particular, the author wishes to express that he does *not* have delusions of grandeur - namely, he *is* aware of many clinical-grade codes (including EGSnrc, GEANT4, and many other high-quality, highly-technical frameworks such as GPUMCD) which would be an excellent choice for someone looking to perform Monte Carlo transport simulations.

What is the worth of a new project, then? For one, well-established codes like EGSnrc carry an enormous baggage that represent years of experimentation, testing, and platform support. The age of the underlying code is enough to have survived many updates to the Fortran language. Additionally, the types of simulations which can be easily performed are dependent on the various authors of the current source; the time scale of such frameworks mean that not only have the particle energies, materials of interest, and computer hardware changed considerably since inception - even the *types* of hardware have changed. In other words, we're playing an entirely different game these days.

Rationale

Having still not addressed *why* my framework is of any worth, let me first enumerate my design goals. The framework should...

1. ...impose no constraints on the user which are physically consistent,
2. ...be configured directly with *code* - not with ghastly configuration files,
3. ...be entirely modular with all components of a simulation being loaded into the simulation kernel on-the-fly,
4. ...remain as simple as possible, offloading post-simulation analysis to helper programs where appropriate,
5. ...be able to have modules written in the language of choice for the user,
6. ...be fast enough to justify the choice of the simulation kernel being written in C++, but no faster.

These goals have been met with the exception being the first (due to time constraints.) Note that although speed is not a true design goal, the efficiency has been found to be 'suitable' for the types of problems examined by the author. Benchmarks are given where appropriate.

Utility

Thus, the utility of this framework is that, first and foremost, it is designed to be used however one would like it to be used. There are very little limitations for the user who is able to comfortably work with source code. In particular, there is no facility for the non-programmer: the code is *designed* to change over time, and the modular design means that modules which contain so-called "code-rot" can be pruned easily. To the author's knowledge, there are no particle transport codes which cater to the programmer and not the user. In the following report, we will see what limitations and fortitude's this choice of paradigm provides us.

Getting the code

An open source project would be worthless if the code was not easily accessible. A working copy of the code is parked at Github and can be obtained at `https://github.com/hdclark/Transport-Framework`.

As there are approximately 5000 lines of non-comment code at the time of writing, the author will *gladly* give a guided tour, in-depth explanations, and example computations on request. Note that ample (perhaps, excessive) commenting is provided for all files.

Compilation is as simple as

```
make -j 4
```

which will generate the simulation kernel "transport" and the various shared libraries included.

Basic premise

At the most basic level, this framework takes a cache of particles which originate from a source, cycles through the cache sequentially, determines how far each particle should move based on an arbitrary set of parameters (typically energy and geometry), and performs one or more interactions. Each interaction might result in energy being deposited or additional particles being created. Such energy is recorded in the format desired (voxels, vector-based formats) and such particles are added to the cache. Simulation continues until the cache empties.

Some interesting capabilities have emerged from the code. For example

1. The "particles" need not be particles. They could also be sources (brachytherapy seeds, MRI nucleii, etc.) or sinks (black holes, detectors, etc..)
2. The "sources" need not be beams. They can be made to vary spatially and over a set of keyframes, such as a rotating gantry head.
3. All interactions and particles are treated using polymorphism, so that very little additional work is required to simulate exotic particles.
4. All interactions are treated in a fully-relativistic manor[4], so that the spacetime the simulation is embedded in could be changed to something more "exotic."

Basic usage

This framework will only work on Unix-like systems. Parts of the code rely on the GNU Compiler Collection (GCC,) and unless otherwise stated, the code has been written in C++11. Compiler support for C++11 is currently limited on other systems.

The basic modules one must provide for a typical particle transport simulation are geometry, source energies, material properties, a concept of the type of particles which exist in the simulation environment (photons, electrons, etc.,) implementation of the interactions one requires, and a list of the quantities one wishes to measure. In practice, common particles, materials, beams, and logging routines carry over from simulation to simulation. Thus, differentiation typically requires creation (or modification) of three files: Constants.cc, Geometry.cc, and Beam.cc. The creation of such files is outside the scope of this document, but several examples are provided for each.

Once written and compiled, each module is loaded (as a shared library) into the simulation kernel. Mixing between modules is performed only where required (such as the interaction modules.) Thus, the simulation kernel (contained within the file Transport.cc) has basic common code that looks like

```
1     ...
2     libraries.push_back("./lib_photons.so");
3     libraries.push_back("./lib_electrons.so");
4     libraries.push_back("./lib_positrons.so");
5     libraries.push_back("./lib_random_MT.so");
6     libraries.push_back("./lib_memory.so");
7     libraries.push_back("./lib_coherent.so");
8     libraries.push_back("./lib_photoelectric.so");
9     libraries.push_back("./lib_compton.so");
10    libraries.push_back("./lib_pair.so");
11    libraries.push_back("./lib_no_interaction.so");
12    libraries.push_back("./lib_localdump.so");
13    libraries.push_back("./lib_slowdown.so");
14    libraries.push_back("./lib_water_csplines.so");
15    libraries.push_back("./lib_logging.so");
16    libraries.push_back("./lib_detect.so");
```

[4]Note that the modules provided implement particle interactions to first-order only. This statement implies that the framework is designed to handle higher-order diagrams given the appropriate modules.

```
    libraries.push_back("./lib_voxel_mapping.so");
    ...
```

to which one simply needs to add the following code to perform a simulation in an infinite water tank with simplified 6 MV source.

```
    ...
    //------------ Infinite Water tank setup ---------------
    libraries.push_back("./lib_beam_6MV.so");
    libraries.push_back("./lib_geometry_inf_water.so");
    ...
```

To switch to simulation of the water tank setup specified in the final question of the assignment, we simply switch this code with the following.

```
    ....
    //--------------- Water Tank setup ---------------------
    libraries.push_back("./lib_beam_6MV.so");
    libraries.push_back("./lib_geometry_water_tank.so");
    ...
```

Further, to emphasize the concision of the setup, the following switch will simulate a CT scanner complete with a unique xray tube and a small water sphere, perform the imaging procedure, and output data which can be[5] reconstructed to give a completely virtual image of a virtual object.

```
    ...
    //----------------- CT geometry ------------------------
    libraries.push_back("./lib_beam_xray_N7599.so");
    libraries.push_back("./lib_geometry_CT_imager.so");
    ...
```

There are additional customization's one would typically like to make. These include the number of particles to simulate, pseudo-random number generator seed (specified as arguments to the simulation kernel) and minutiae allowing one to toggle the various approximations and techniques used in the simulation[6].

A typical invocation would look like

```
./transport -s 123456_hashed_into_some_seed_value_789 -p 1000000
```

which is equivalent to

```
./transport --seed 123456_hashed_into_some_seed_value_789 --particles 1000000
```

Finally, we are in a position to begin discussion of the actual simulations.

Problem 1

Question: Using an infinite water continuum geometry, an "isotropically-oriented" source, and a 6 MV spectrum, produce a plot of the relative probability of interactions a photon could undergo in the medium versus photon energy.

Already, there is a large amount of variability possible due to the particular interactions consider, the data available for the medium, and, perhaps, even due to the quality of random numbers generated.

We will begin with the mass attenuation coefficients used. The data provided in the assignment was in the form of a lookup table. To avoid discretization, it was deemed necessary to interpolate the data given. Three techniques were performed and are included with the source. The first is a simple linear interpolation. The results were found to be adequate, but, somehow, left something to be desired. The second approach was to attempt to fit the data with functions. Several types of functions were considered, but since the coefficients are extremely logarithmic focus was on fast computation. High accuracy fits were found using various trial functions. Initially, exponential functions and natural logarithms were used extensively but were found to have the wrong curvature to robustly describe all coefficients provided. Figure 1.4 shows such a fit.

In general, however, a delightfully simple fitting kernel was found that could accurately faithfully reproduce the curvature demanded by more troubling data like the Compton mass-attenuation coefficients. The procedure was found to work well on a

[5] And, in fact, *has been* reconstructed. A complete report of this simulation has been written and is available upon request.

[6] In principle, the links to the shared libraries can be passed in as parameters, but the author found it slightly cumbersome to do so for the sake of this assignment.

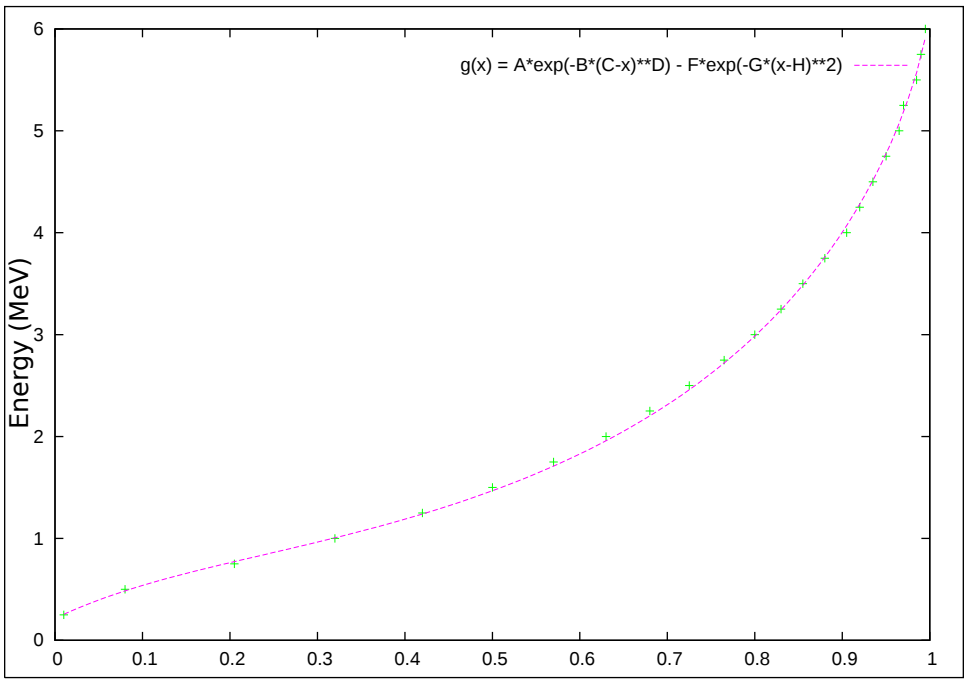


Figure 1.4: An example of a fit performed on the 6 MV spectrum data using strictly exponentials. The fit is fairly good, considering that a lookup table would otherwise be used. The data itself is distribution suitable for converting a clamped $[0, 1]$, uniformly-distributed randomly-generated number into a photon energy.

wide variety of "log-log" data. The basic premise is to fit a function of the form

$$F(x) = \left(\frac{1}{f_1(x)} + \frac{1}{f_2(x)} + \ldots + \frac{1}{f_n(x)} \right)^{-1} \tag{1.1}$$

where n is a parameter which describes the amount of curvature in the data (typically on the order of three or four for the mass attenuation coefficients used herein) and the functions $f_i(x)$ are taken to be $f_i(x) \equiv \alpha_i x^{\beta_i}$ for computational efficiency.

The results of an example fit are shown in figure 1.5. A much higher quality fit results compared with direct exponential fitting, and although division is typically costly for a floating-point unit (FPU) to perform, modern compilers can make use of vectorization techniques to perform many simultaneous divisions. Such operations are no more costly than exponential functions[7].

After such trouble, however, it was decided that an even more generic technique should be sought. This search led to the use of so-called *c-splines* which will result in a twice-differentiable fit. In physics, it is always a nice bonus to deal with functions which have continuous derivatives, so this approach is used for the final approximation of the data. In general, the fits are no worse than the linear interpolation, but there is a well-known issue due to overspecification of data when fitting c-splines that can result in the introduction of slight oscillation into the interpolation. Additionally, extrapolation using c-splines could adequately be described as a 'total nightmare' and should be avoided. We refer the reader to [2] for further discussion of the fortitude in and caveats of using c-splines.

Coming back to the problem at hand, we are able to generate a plot of the relative probability of each interaction type by simply examining the mass attenuation coefficients which we have nicely fit with c-splines. Such a plot is shown in figure 1.6 in absolute terms and alternatively in figure 1.7 with each interaction displayed as a fraction of the total. The latter is perhaps clearer intuitively, but the former provides more information regarding the quality of fit.

Problem 2

Question: Using the same setup as for the previous question, plot the average fraction of energy transferred to recoil electrons in Compton scattering versus incident photon energy.

The data for this plot are collected during simulation. This is a slow way to perform any simulation, but we use a low number of photons to keep total execution time low.

[7]Note this statement is nearly meaningless without specifying the exact CPU and FPU in question. The point is meant to be illustrative and not exact. Testing should be performed whenever such benchmarking is an issue.

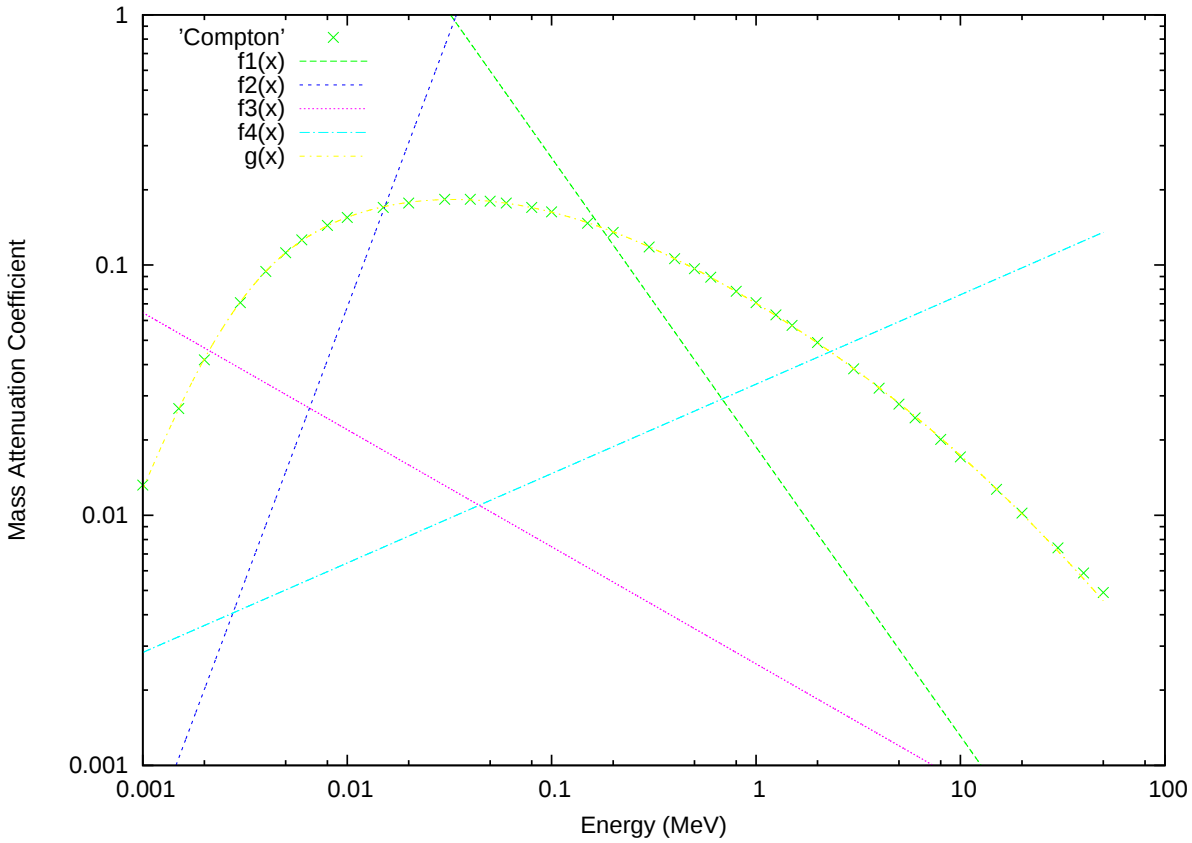


Figure 1.5: An example of a fit performed on the Compton mass-attenuation coefficients (in water) versus energy. The fit is the curve punctuated with data points. The other functions are the $f_i(x)$ functions used in the fit. Both fit quality and ease of fit were preferable to using exponentials, natural logarithms, or Gaussians.

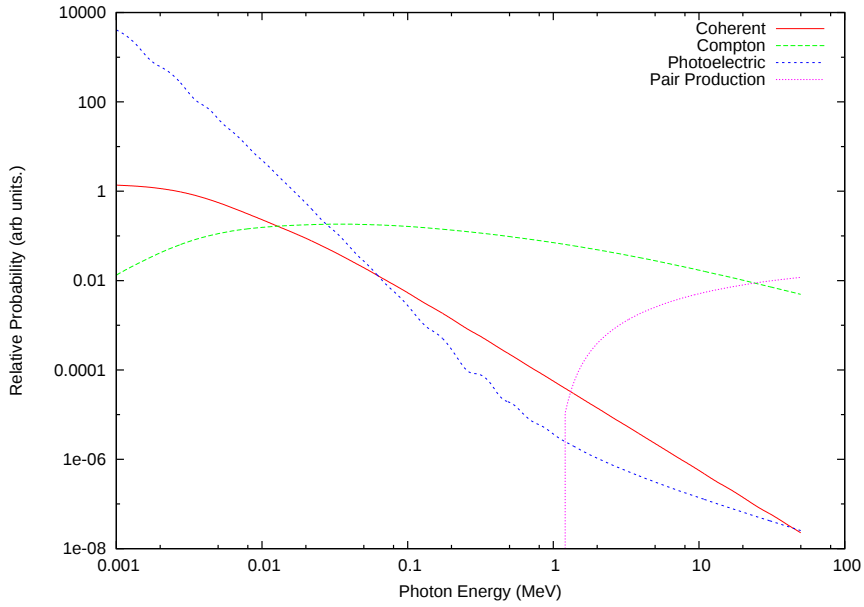


Figure 1.6: Relative probability of photons interactions in water in arbitrary units. In actuality, these coefficients are simply the linear attenuation coefficients. Note the slight oscillation of the photoelectric effect interpolation due to the use of c-splines. In general, this is of no noticeable concern for simulations.

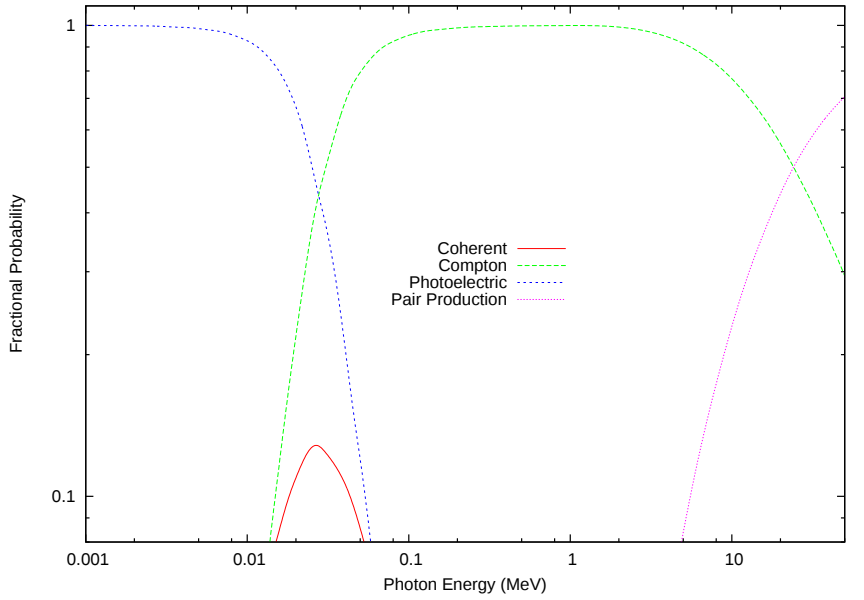


Figure 1.7: Fractional probability for each individual interaction type considered in the assignment. Note that the coherent interaction is marginally important over a small region, and fairly unimportant everywhere else.

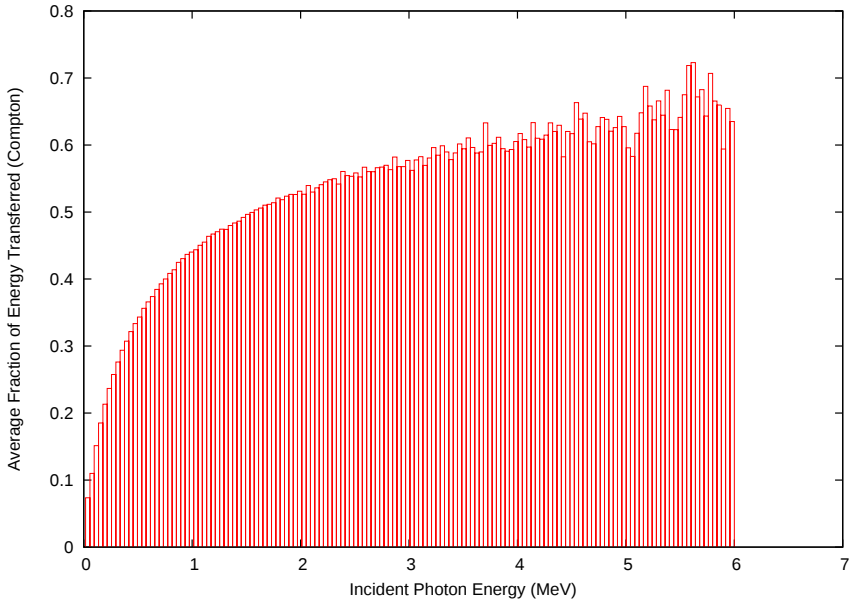


Figure 1.8: Average fraction of energy transferred to recoil electrons in Compton interactions versus incident photon energy. Data was collected *in silico* for one million photons.

```
$ time ./transport -p 1000000
--(I) In function: main: Proceeding with random seed 12345677.
--(I) In function: main: Proceeding with 1000000 particles (using 10000 per loop.).

real 12m32.936s
user 9m20.487s
sys 2m57.278s
```

Note that this program was run using a single $1.33MHz$ CPU. The program produces a file /tmp/Transport_Fraction_Transferred_Compt which contains two columns of data: the incident photon energy and the fraction of energy transferred. In order to be able to display the data in a useful way, a Lua script is provided in the /Helpers folder. It is not terribly efficient, but works well. In particular, one needs to execute

```
$ ./Fractional_E_Transfer_Bin_Averager.lua
```

which will produce an additional file /tmp/Transport_Fraction_Transferred_Compton.plot which contains a directly plottable histogram, as shown in figure 1.8.

Problem 3

Question: Using the same setup as for the previous question, plot the average number of interactions a photon will undergo before being absorbed into the water medium versus initial photon energy.

The data for this plot is easy enough to collect. As it turns out, the previous simulation has already produced the data needed for processing. The file /tmp/Transport_Interaction_Count.process contains two columns: the initial photon energy and the number of interactions undergone in the medium. Running the accompanying processing script is equally as easy

```
$ ./Interaction_Count_Bin_Averager.lua
```

which produces the file /tmp/Transport_Interaction_Count.plot which can be directly plotted as in figure 1.9. The average number of interactions appears to remain somewhat steadily around 15. This data was also collected for one million photons.

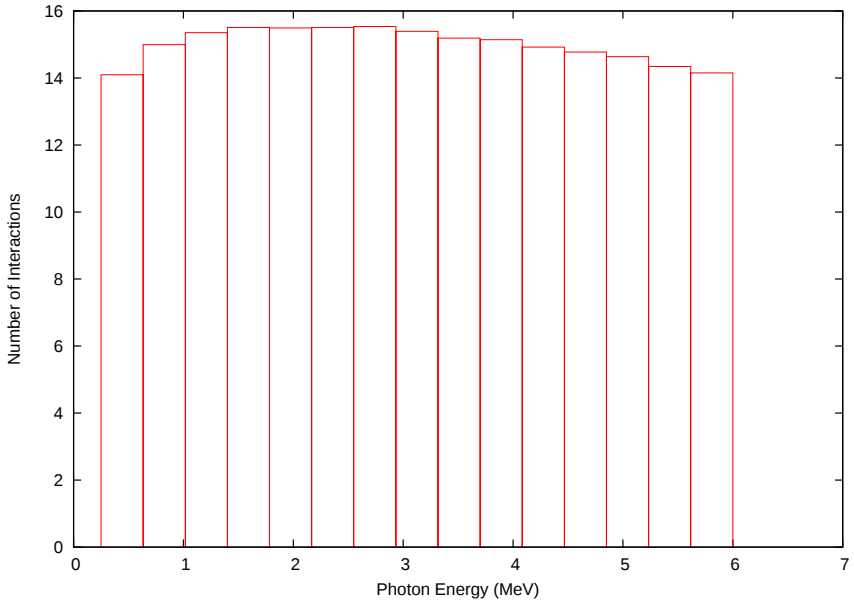


Figure 1.9: Average number of interactions performed within a medium by photons of a given initial energy. Note that the step-spectrum given in the assignment was used for this question, so a finer resolution is not possible.

Logging

The code used to collect information from the simulation is delightfully simple to use. A generic logging module was written such that the method one instantiates a log file and the method one writes to the same log file are identical - the file is silently created by the framework if it does not already exist, and the file is never created if it is never written to. The interface works using the C++ STL-type fstream. A function is called which has access to a list of the currently open fstreams. If the key which is sent into the function is already taken, the function returns a reference to the file stream. Otherwise, the function attempts to open a new file, puts the new key into the list, and recursively calls itself so that it can return a proper reference to the file.

Thus, all logging in the framework has the following signature

```
log_function("Fraction_Transferred_Compton") << incoming_photon_E << " " <<  ((electron_E-electron_mass),
```

which will produce the file /tmp/Transport_Fraction_Transferred_Compton.process if it does not already exist. With some additional tricks, we can have the logging routine silently load the log file with some useful header information pre-inserted. The benefit of this arrangement is twofold. First, the user never needs to worry about creating or managing a log file. Second, the interface is homogeneous so that there is no need to treat the first or seventieth writes differently.

An added benefit of this design is that the output of files can be controlled directly with a simple conditional. For instance, if we have a line that looks like

```
if(false) log_function("Fraction_Transferred_Compton") << incoming_photon_E << " " <<  ((electron_E-electr
```

then no file is ever created. This saves us from having many empty files in the output directory.

Of course, keeping in line with the design goals of the project, the logging routine is loaded as a module. This module can be changed, enhanced, or removed as per the users wishes.

Problem 4

Question: Using the same setup as for the previous question, plot the angular distribution of Compton-scattered photons.

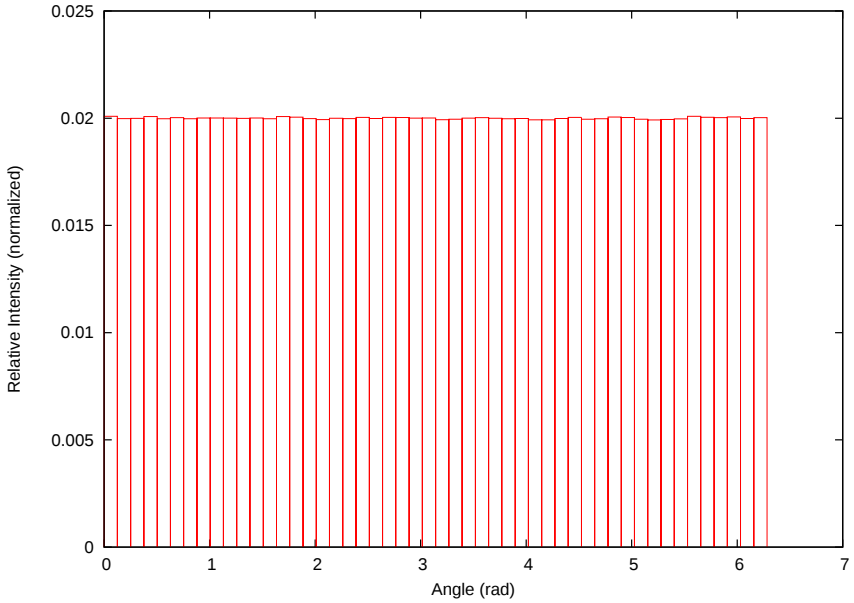


Figure 1.10: Sampled angular distribution for plane-orientation angle, which is randomly distributed for two-body interactions. This angular distribution is for energies from 0 to 0.5 MeV where the beam spectrum provides good statistics.

This question requires some elaboration on the technique used to orient the photons. First, let us perform the analysis using the helper functions

```
$ ./Compton_Angular_Distribution_Bin_Averager_phi.lua
$ ./Compton_Angular_Distribution_Bin_Averager_theta.lua
```

which produces the files /tmp/Transport_Photon_Angular_Distribution_phi.plot and /tmp/Transport_Photon_Angular_Distribution_thet These files are shown in figures 1.11 and 1.10. Note that the Compton scatter happens in a plane, and the plane is randomly oriented about the incoming photon trajectory. This implies that one of the angles should be uniformly distributed for each photon energy. Indeed, figure 1.10 shows this behavior for some low energy. The plot for the other angle is much more interesting. It is plotted by binning both energy and angle and separate curves are produced for the energy bins. Comparison of the data with the theoretical distribution (shown in figure 1.3 of the previous assignment) is very good.

This data was collected for one million photons but each photon has most likely undergone multiple Compton interactions, producing many more samples. Due to the energy spectrum being bottom heavy, better statistics are found for lower energies.

Technique for orienting particles after two-body scatter

Given that rotation about an arbitrary specified plane by a specific angle can result in a cumbersome hodge-podge of rotation matrices and coordinate singularities, a novel technique is provided here. It appears to be numerically stable, but is not particularly fast though it is believed to be approximately as fast as using a rotation matrix approach.

The basic premise is to take a unit vector $\hat{u} = (u_x, u_y, u_z)$ representing the incoming photon's three-momentum, specify a plane which is aligned such that it intersects the unit vector at both head and tail, rotate the plane about the vector by a randomly-chosen angle ϕ, and then rotate the unit vector in the plane by the angle θ. This new unit vector defines the direction of travel of the outgoing photon.

To avoid the use of rotation matrices, we deal directly with the plane equation. We specify a unit vector transverse to $\hat{u}$ which we call $\hat{v}$. The direction of $\hat{v}$ is fixed by an angle R which we provide and is subject to the constraint that it remain perpendicular to $\hat{u}$. Thus, given $\hat{u}$, we must have that

$$\hat{u} \cdot \hat{v} = 0 \Rightarrow u_x v_x + u_y v_y + u_z v_z = 0. \tag{1.2}$$

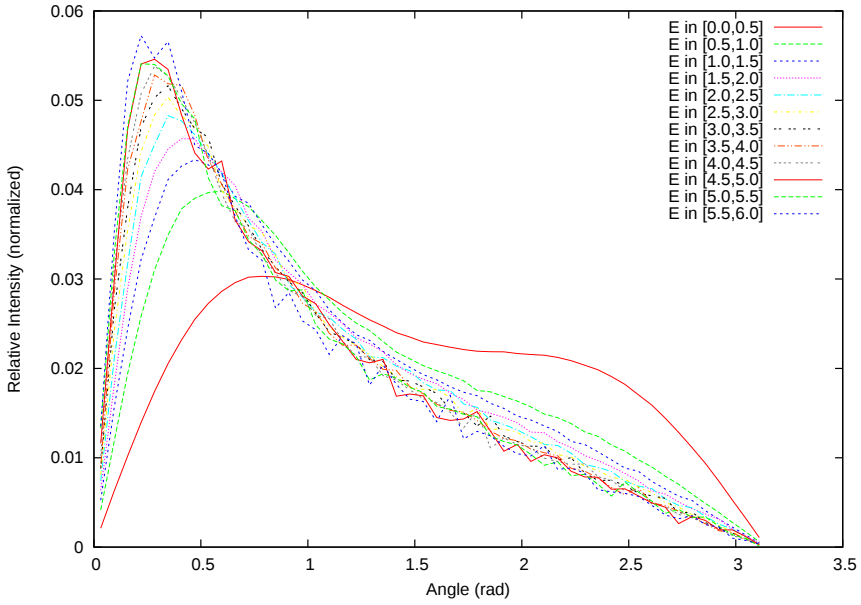


Figure 1.11: Sampled angular distribution for in-plane photon deflection angle. The lowest energy distribution is the curve with the lowest peak on the left side of $\pi/2$. Agreement with theoretical distribution is strong.

Since $\hat{v}$ is of unit length, we transform to an angular coordinate system

$$\hat{v} = (\cos(s)\sin(t), \sin(s)\sin(t), \cos(t)) \tag{1.3}$$

and then fix either s or t with the angle R so that we can solve for the remaining angle. The solution is not pretty, and there are two principle solutions in either case, but it is computable, and gives us a certain amount of freedom. In general, we *do* pick up coordinate singularities for a general angle, but we fortunately notice that coordinate singularities are confined to specific regions of either solution. Thus, if the value of R is within a certain range, we simply switch from setting, say, $s = R$ to $t = R$ and then solve for the corresponding s. As an illustration, one such solution looks like

$$s = \log\left(\frac{u_z\left(e^{2it}+1\right) + \sqrt{{u_z}^2e^{4it} + {u_y}^2e^{4it} + {u_x}^2e^{4it} + 2{u_z}^2e^{2it} - 2{u_y}^2e^{2it} - 2{u_x}^2e^{2it} + {u_z}^2 + {u_y}^2 + {u_x}^2}}{u_ye^{2it} + iu_xe^{2it} - u_y - iu_x}\right) \tag{1.4}$$

The reader will notice that the expression is in complex notation and involves a logarithm, a square-root, many (complex) exponentials, and division - and thus we expect performance to be poor. In contrast, however, rotation matrices can involve the computation of many trigonometric functions and several matrix computations. In general, performance has been found to be sufficient. For instance, as we will see later, simulations with *billions* of particles can be achieved in hours. For a more complete description of the method described here, we refer the reader to the implementation in the source file MyMath.cc.

Shown in figure 1.12 are some iso-curves corresponding to either fixed R or fixed θ.

Problem 5

Question: Using a finite water tank of $50\,cm$ depth, plot the percentage-depth kerma for $1\,MeV$ photons along the central axis.

This simulation can be carried out fairly quickly.

```
$ time ./transport -p 1000000
--(I) In function: main: Proceeding with random seed 12345677.
--(I) In function: main: Proceeding with 1000000 particles (using 10000 per loop.).
```

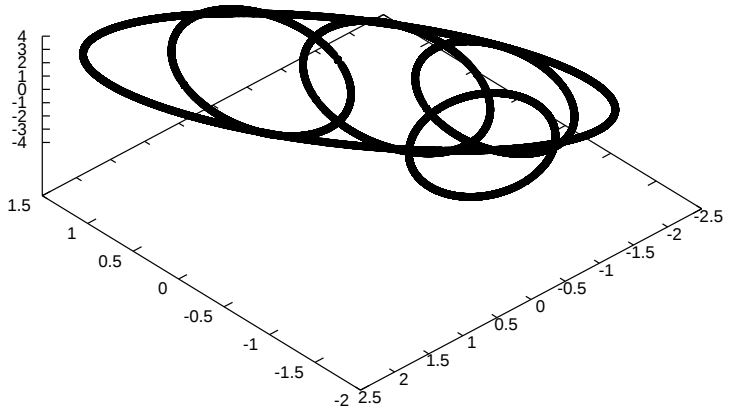


Figure 1.12: Iso-curves corresponding to either fixed R or fixed θ to illustrate the utility of the technique described for rotation of angle θ about an arbitrary plane specified with a unit vector and a random number R. Of particular interest is the fact that the curves are seamless.

```
real 1m43.145s
user 1m39.024s
sys 0m4.066s
```

Since the beam is monoenergetic, narrow, and we only consider photons along the central axis, the percentage-depth kerma curve can be generated simply by collecting the distance each photon travels into the medium and normalizing. As such, we expect a simple exponential decay. This is confirmed in figure 1.13.

The helper scripts used to generate this (and the next question's plot) are run like

```
$ ./PD_Kerma_1MeV_Binner.lua
```

which processes the file /tmp/Transport_PD_Kerma_1MeV.process into a plottable file /tmp/Transport_PD_Kerma_1MeV.plot.

Problem 6

Question: Similar to the previous question, plot the percentage-depth kerma for 10 MeV photons along the central axis.

This question is nearly identical to the previous question, and we expect only the decay rate to be different. The plot is shown in figure 1.14. For reference, the exponential decay expected for a narrow, monoenergetic 10 MeV beam is shown to perfectly coincide with the result of the simulation.

Problem 7

Question: Similar to the previous question, plot the percentage-depth *dose* for a 6 MeV beam of photons along the central axis.

This question presents a bit of a challenge compared with the monoenergetic case. Since each photon energy has a corresponding mass attenuation coefficient and average energy absorbed, we have to take into account the energy of each photon.

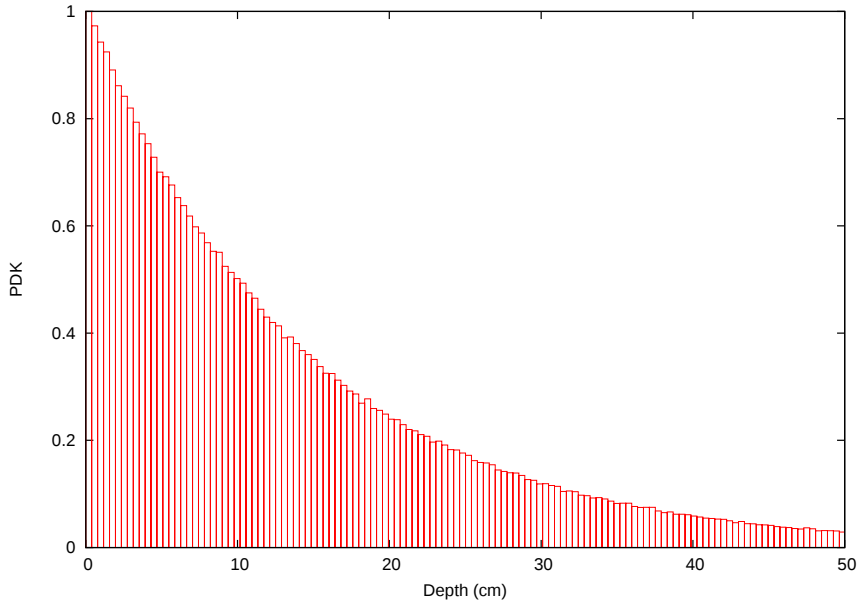


Figure 1.13: Percentage-depth kerma curve for 1 MeV photons in water.

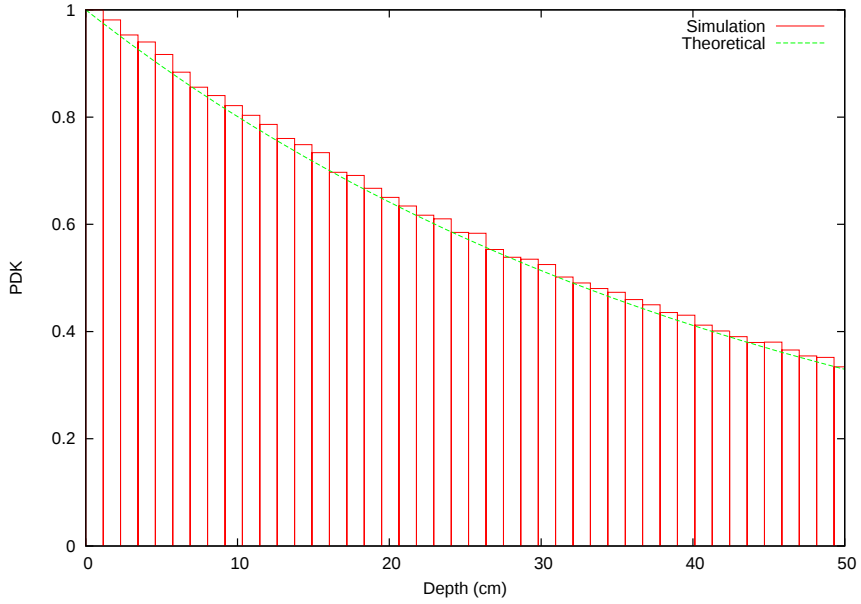


Figure 1.14: Percentage-depth kerma curve for 10 MeV photons in water.

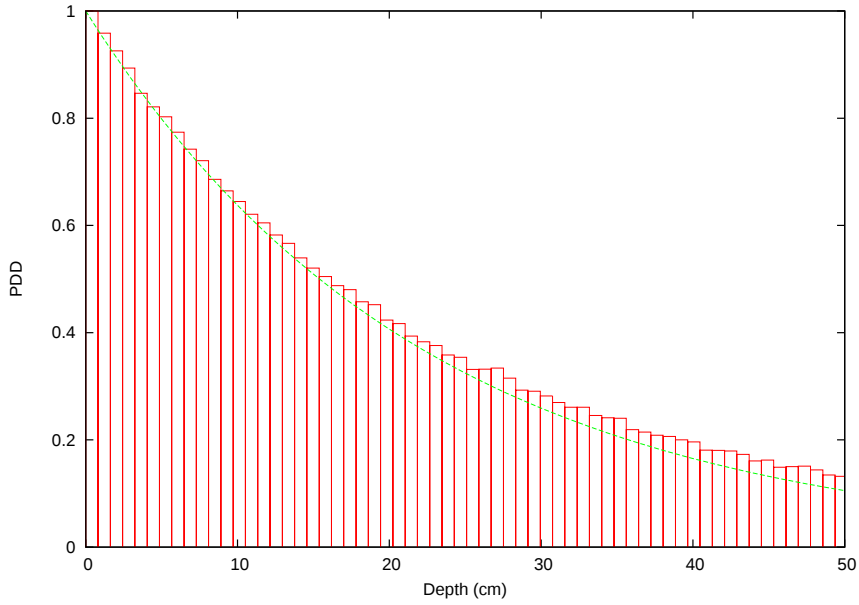


Figure 1.15: Percentage-depth dose curve for $6\,MV$ photons in water. Also shown is an exponential decay curve which we would expect to see for monoenergetic photons of energy $3\,MeV$ which fits the data well at low-depth, but clearly does not describe the entire PDD. There is no build-up region because we do not allow electrons to travel through the medium. Electrons simply dump their energy wherever they interact with a photon.

A helper script was written to collect and bin this data, and then numerically integrates over bins to give the percentage-depth dose curve shown in figure 1.15. It is run like

```
$ ./PD_Dose_6MV_Binner_Averager.lua
```

and processes the file /tmp/Transport_PD_Dose_6MV.process into a directly plotable file /tmp/Transport_PD_Dose_6MV.plot.

One may wonder why the PDD lacks a characteristic build-up curve. The conspicuous absence is due our imprecise handling of electrons (as specified in the instructions of this question.) Namely, we do not allow electrons to travel through a material. When a photon interacts with an electron, we make the electron immediately dump its energy into the medium on the spot. Therefore, there is no hope of seeing a build-up region because we never have an electronic disequilibrium. This treatment is an unrealistic one.

Problem 8

Question: Consider a water tank of width and breadth $30\,cm$ and depth $60\,cm$. Simulate a $6\,MV$ parallel photon beam with a field size of $10 \times 10\,cm^2$ incident on the surface. Treat electrons with the continuously-slowing-down approximation (CSDA) and assume that the stopping power is constant at $2.0\,MeV/cm$. Collect dose and kerma in a voxel geometry with cubic voxels of linear dimension $5\,mm$. Implement an electron cutoff energy at $100\,keV$ such that electrons do not undergo a CSDA slowdown, but rather locally dump all their kinetic energy. Perform the simulation for one million photons.

This question is where the heart of the assignment lies, since this is the first time we are asked to simulate electron transport. In general, one million photons was found to produce very noisy data. Furthermore, the runtime was found to be very short. Considering only those photons which undergo a single interaction before producing an electron, we have an incredibly fast runtime.

```
$ time ./transport -p 1000000
--(I) In function: main: Proceeding with random seed 12345677.
--(I) In function: main: Proceeding with 1000000 particles (using 10000 per loop.).
```

```
real 0m21.597s
user 0m20.972s
sys 0m0.163s
```

Simulations with more than *two billion* photons were performed in less than eight hours!

Considering all photons the simulation time is somewhat longer, but it is still quite manageable.

```
$ time ./transport -p 1000000
--(I) In function: main: Proceeding with random seed 12345677.
--(I) In function: main: Proceeding with 1000000 particles (using 10000 per loop.).

real 1m33.937s
user 1m33.441s
sys 0m0.440s
```

The difference between the two cases are the number of events which we are able to register. In other words, if we limit ourselves to photons which undergo a single interaction, then we are (literally) removing photons from the simulation by culling photons which undergo more. Judging from the simulation time, we can estimate that we are effectively simulating about $2/9^{ths}$ the photons we specify. Still, $2/9^{ths}$ of two billion is nearly 450 million photons, and the statistics of such computations lead us to believe that a higher fraction of photons are effectively simulated. However, actual depth-dose curves are computed without culling photons based on the number of interactions undergone.

Some dose profiles are shown for various numbers of particles in figures 1.16, 1.17, and 1.22 and some beam cross-sectional slices are shown in figures 1.23 and 1.24. Note that these distributions give the dose in absolute value so that the reader might be able to somehow judge *how much* of a real beam is being simulated. As can be seen by the doses of the order of μGy, we are (very) far from simulating every photon for a typical treatment. If the reader would rather a PDD, normalization is a trivial operation.

The analysis of these computations could be made highly complex and one could use such an analysis to both troubleshoot and tune the framework. However, in the interest of brevity, let us examine the presence of some key factors which indicate a general sort of 'health' about the code. Three such features are observable 'from ten thousand feet', as the saying goes.

The first is the presence of a build-up region. This indicates immediately that the framework is able to handle charged particle transport. The build-up region is due to the lack of electronic equilibrium characteristic of a beam crossing the boundary of a media, and so the presence of such a region is a fortuitous characteristic of proper charged particle interaction handling. The build-up region is shown more closely in figure 1.19. The depth to D_{max} is approximately $1.5\,cm$, as it should be for a (real life) $6\,MV$ beam.

The second is the general curvature of the curve. In simulations without electron transport, the point of electron interaction is the point where dose is delivered. Thus the delivery becomes highly dependent on the number of photons which interact at each point. Therefore, since the number of photons at depth in a media falls off like an exponential, we get (roughly) a dose profile that looks like an exponential. When we consider electron transport, though, we no longer deliver all dose at the point of interaction. This works to broaden out the dose curve at depth, and shifts it toward a more linear-like curve. Therefore, if the curvature looks very much like an exponential (like for the PDD computed in 1.15), then we probably are not handling charged particle dose delivery correctly. Instead, if the curvature seems 'flat,' then we can be satisfied that the charged particle dose delivery is being smeared out, as it should be.

The final easily-indentifiable criteria in these simulations is actually an artifact of the finite depth of the water tank. At the far edge of the water tank is a drop-off region which is caused by photons and electrons being culled from the simulation when they are beyond $50\,cm$ depth. This results in a loss of equilibrium because the electrons that would normally spray backward from beyond $50\,cm$ depth delivering dose are simply removed. In practice, this is not an issue - we simply increase the depth of the tank beyond the depth that we care about, or we simulate a small extra buffer zone behind the $50\,cm$ mark. The drop-off zone is a useful feature of our computations, though, because it shows us that charged particles are indeed being deflected backward, and that our voxel geometry matches the geometry of the water tank.

We can also make use of the computed depth-kerma curves for comparisons. Since kerma is accumulated in a voxel at a point, then we do not have the smooth smearing out that we do for dose due to the CSDA treatment we assumed. Thus, to smooth out the data, nearest-neighbor averaging has been used on one hundred pixels at each depth. To compare the smoothness achieved with that of the dose (using the same averaging technique for both,) refer to figure 1.25 which gives both accumulated dose and kerma for 100 million photons.

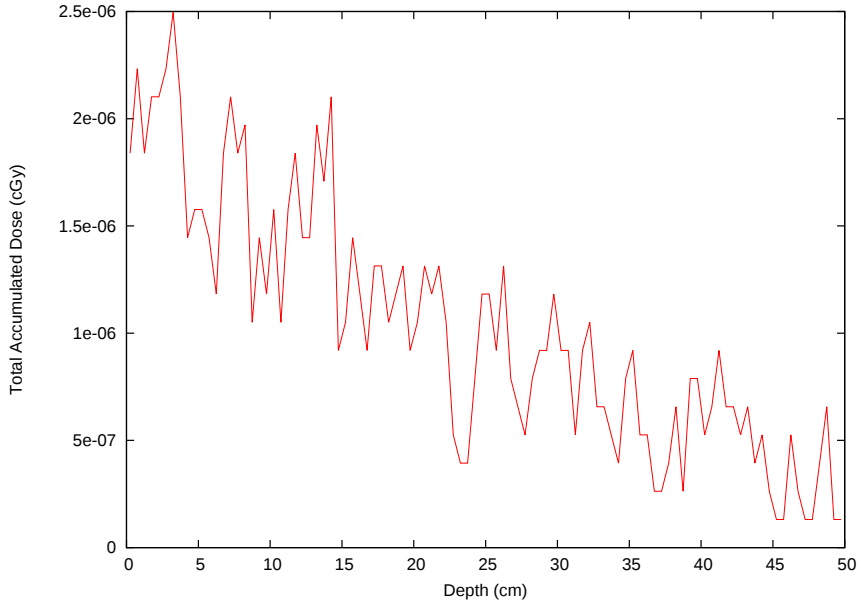


Figure 1.16: Dose profile for one million photons. This result is very noisy, but one can note that curvature seems to be flat enough to indicate the presence of charged particle transport. The data is almost entirely unusable.

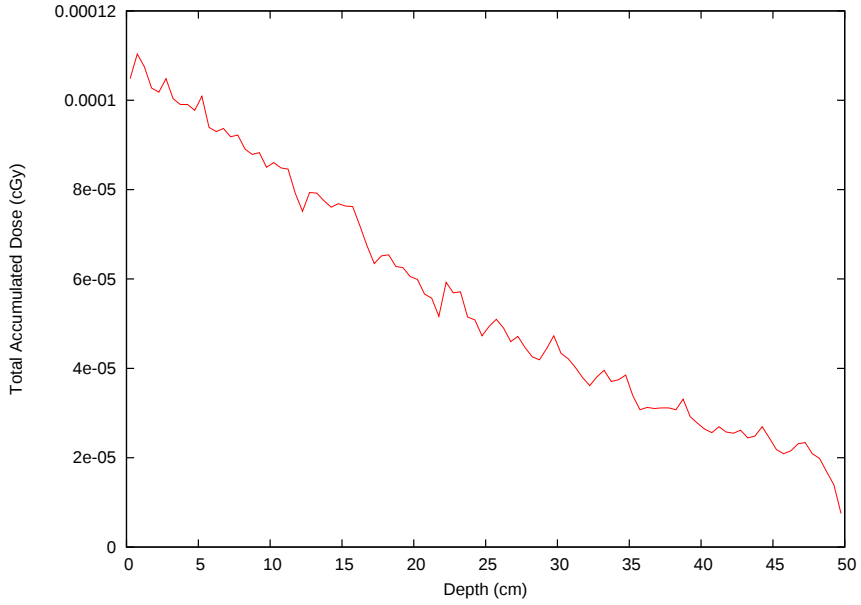


Figure 1.17: Dose profile for 50 million photons. The curve remains somewhat noisy. To reduce noise, one could average neighbouring voxels. Such averaging is performed in latter comparisons.

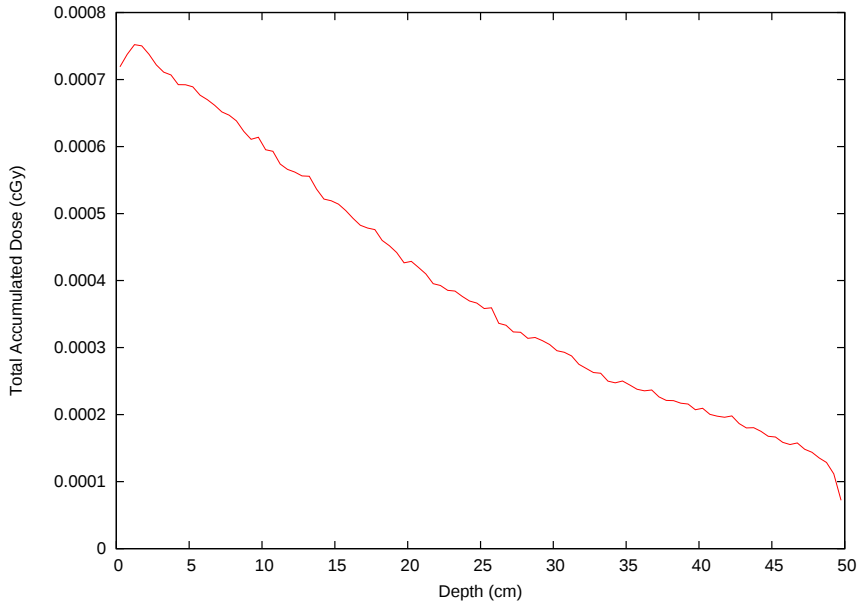


Figure 1.18: Depth dose profile for 350 million photons. Note the presence of both a build-up region and a drop-off region. The build-up region is shown more closely in figure 1.19. The drop-off region is due to our treatment of the far edge of the water tank.

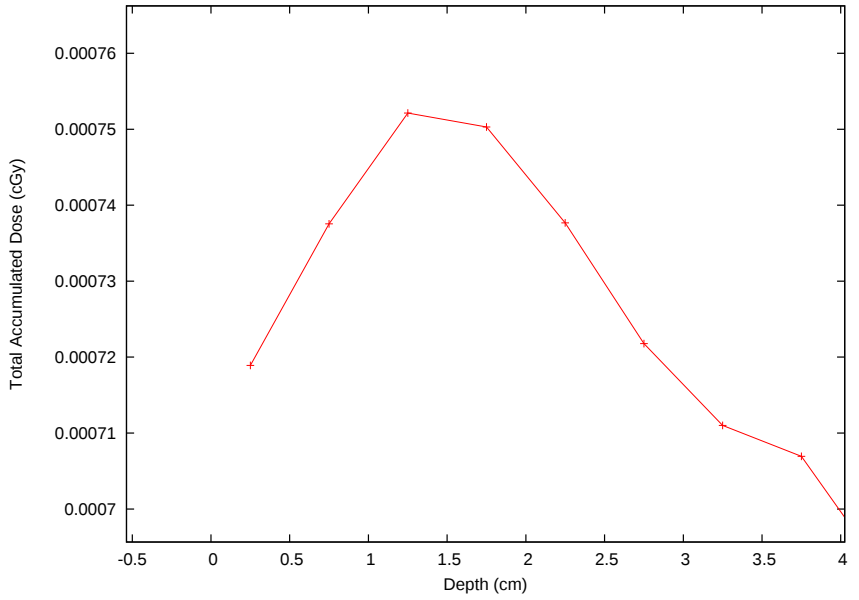


Figure 1.19: Depth dose profile for 350 million photons. Note the depth to D_{max} is approximately 1.5 cm, as it should be for a (real life) 6 MV beam.

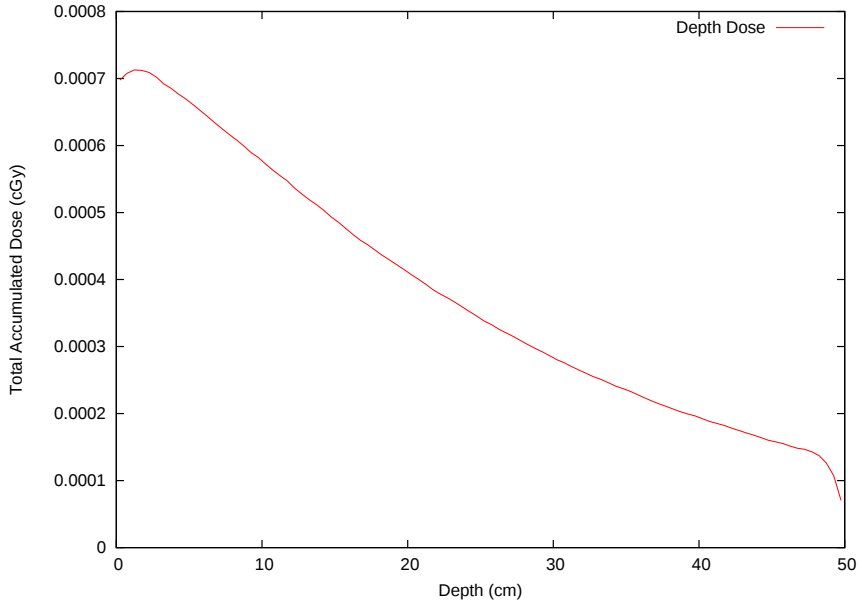


Figure 1.20: Depth dose profile for 350 million photons using one hundred-nearest-neighbor averaging. The build-up and a drop-off regions are clearly shown.

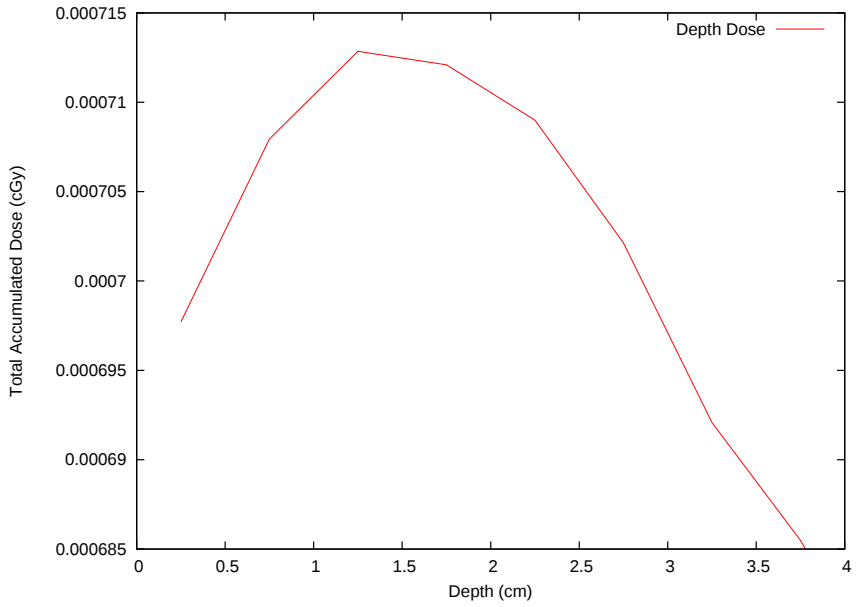


Figure 1.21: Depth dose profile for 350 million photons using one hundred-nearest-neighbor averaging. The depth to D_{max} is still approximately 1.5 cm, but we are limited in accuracy by voxel size.

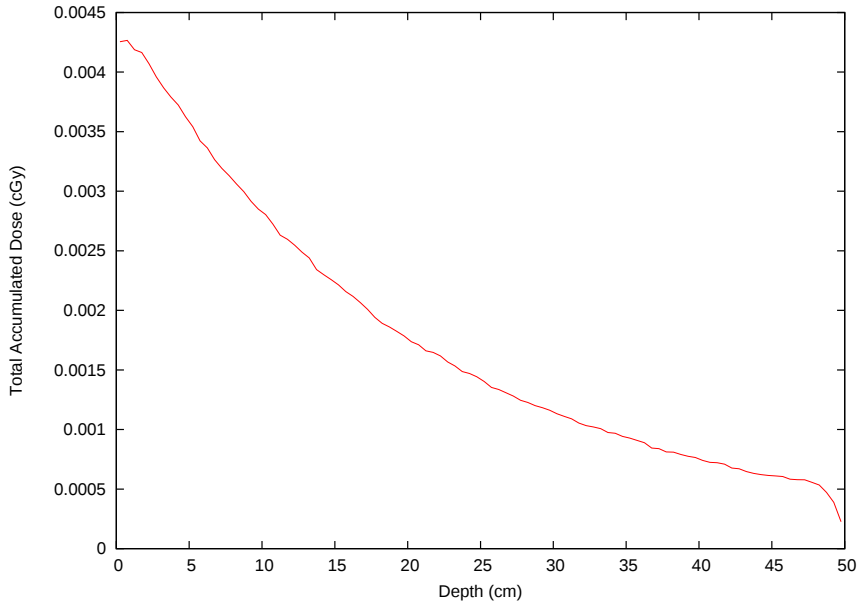


Figure 1.22: Dose profile for approximately 1.9 billion (single interaction) photons. Compared with the 50 million photon PDD in figure 1.17, the curvature of this PDD is wrong, due to considering single interactions only. The build-up region is also mis-shapen due to the culling of photons which would have otherwise bounced backward toward the peak and helped round out the region near D_{max}.

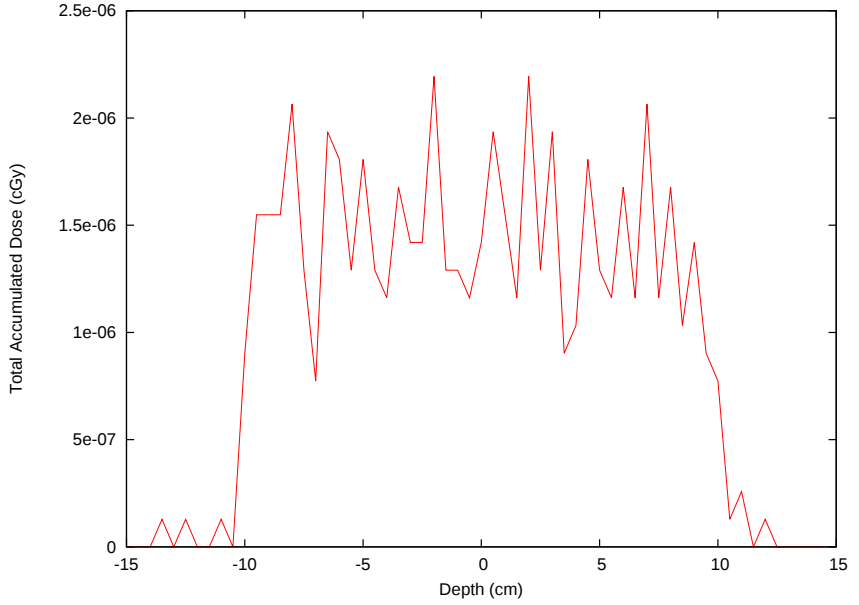


Figure 1.23: Dose profile cross-section for one million photons.

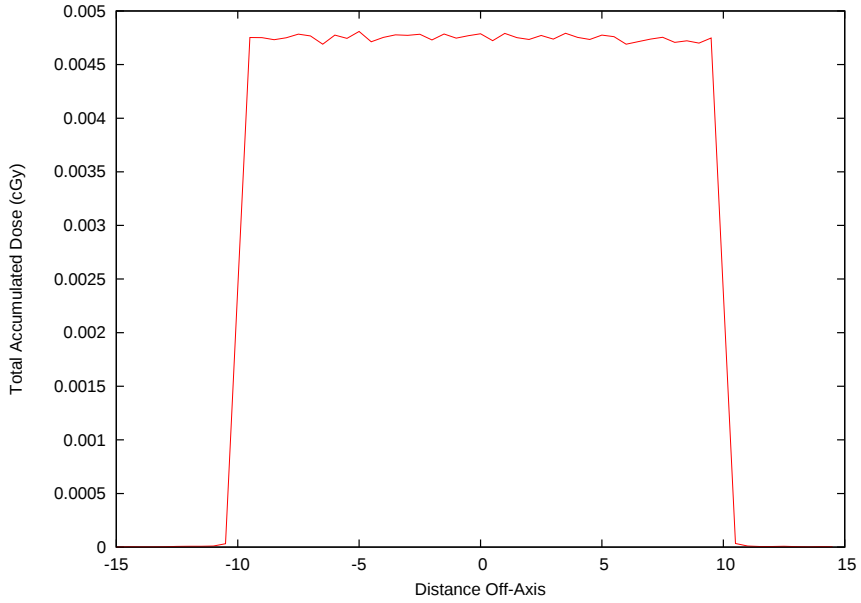


Figure 1.24: Dose profile cross-section for approximately 1.9 billion (single interaction) photons.

Examining the accumulated dose and kerma curves, we can immediately see that the kerma curve lacks a drop-off region. The build-up region is harder to decisively claim is non-existent due to noise, but it appears not to.

As per the second part of question 8, which asks to score the number of primary interactions undergone in each voxel and then to approximate the energy transferred in each voxel by sampling the photon spectrum, the (average) fraction of energy transferred at each photon's energy, and then multiply by the number of photons of that energy, we can instead simply accumulate the (average) fraction of energy transferred by each photon multiplied with the photon's energy *for each interaction* which obsolves us of having to keep track of the actual number of primary events. In other words, instead of multiplying by the number of photons which underwent a primary interaction, we can simply accumulate the quantity of interest *directly* by each photons as it undergos an interaction[8].

The result of this accumulation is shown in figure 1.26.

To get a better feel for what the data actually looks like, the data is shown as an image montage in figure 1.27 and a single frame at depth is shown in figure 1.28. In single frame, one can directly see the variation over the size of the field for 100 million photons (which is very much so able to be seen with the naked eye.) For comparison, the same frame is taken from a simulation with only one million photons for figure 1.29. The image shows a defined field size, but the variation is visibly enormous.

Output voxel format

The module for voxel storage used here (and provided in the source) outputs the scored data into a series of two-dimensional images. The file format is a atypical format, though. It is described after the rationale is given.

In general, it would be most convenient if one would be able to use standard Unix text-processing commands like *awk*, *sed*, *cut*, and *paste* to prepare output data for various purposes. In particular, one would like to be able to

1. View the data as images, for rapid varification.
2. Process the data, usually by examining the central axis or some region of the data.
3. Plot the data along (a) certain direction(s.)

[8]For completeness, the number of primary interactions was still scored for each dataset, it was never needed though.

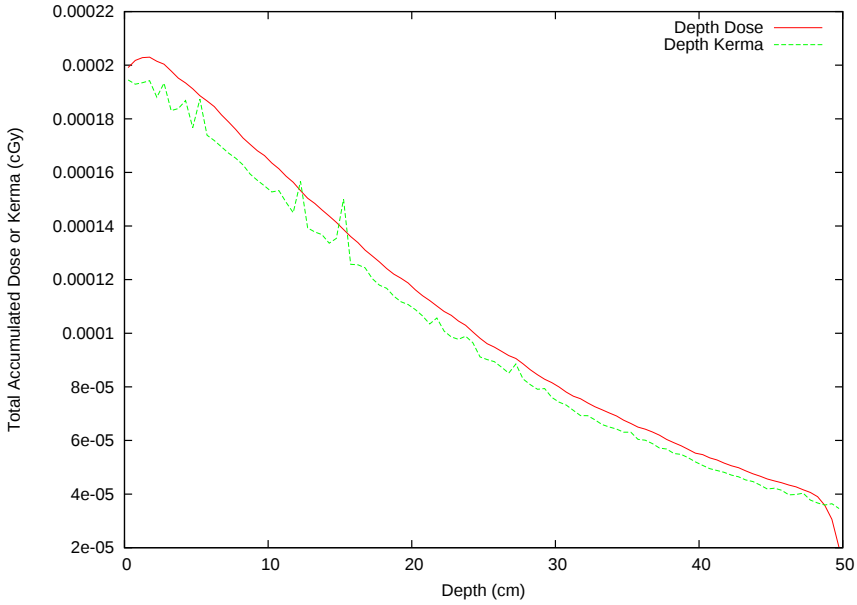


Figure 1.25: Averaged dose and kerma depth profiles for 100 million photons. The kerma curve is much noisier because the accumulation occurs at points and is not smeared out over several voxels as it is for dose. Note the absence of the drop-off region for the kerma curve, which we would expect. It appears there is no build-up region for the kerma curve, but it is difficult to say with certainty, given the noise in the data near the region.

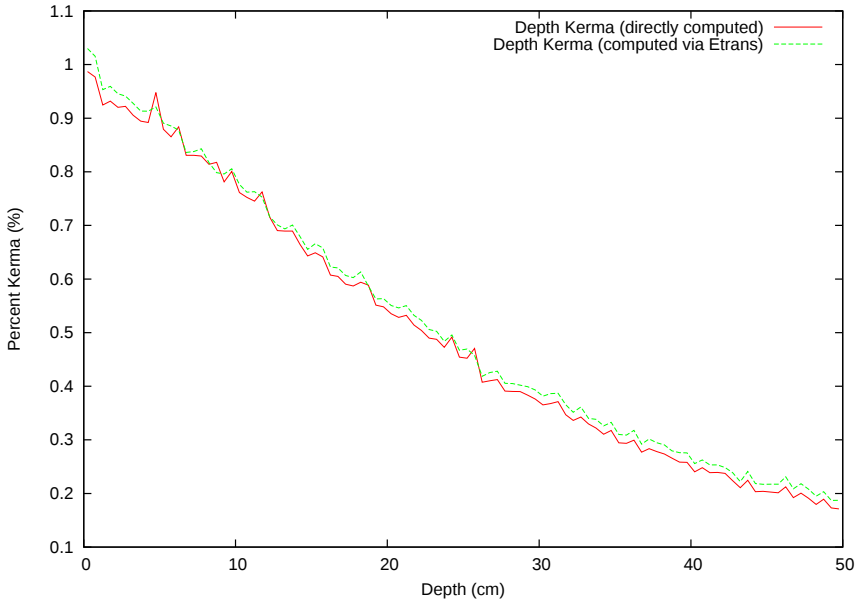


Figure 1.26: Percent-depth kerma curve for ten million photons averaged over 100 nearest neighbour voxels - as directly accumulated by recording the energy transferred (by the electron) and as reconstructed using the scored $h\nu \cdot E_{trans}$ quantity. Agreement is slightly exxaggerated due to normalization, but features in one curve are distinct in the other.

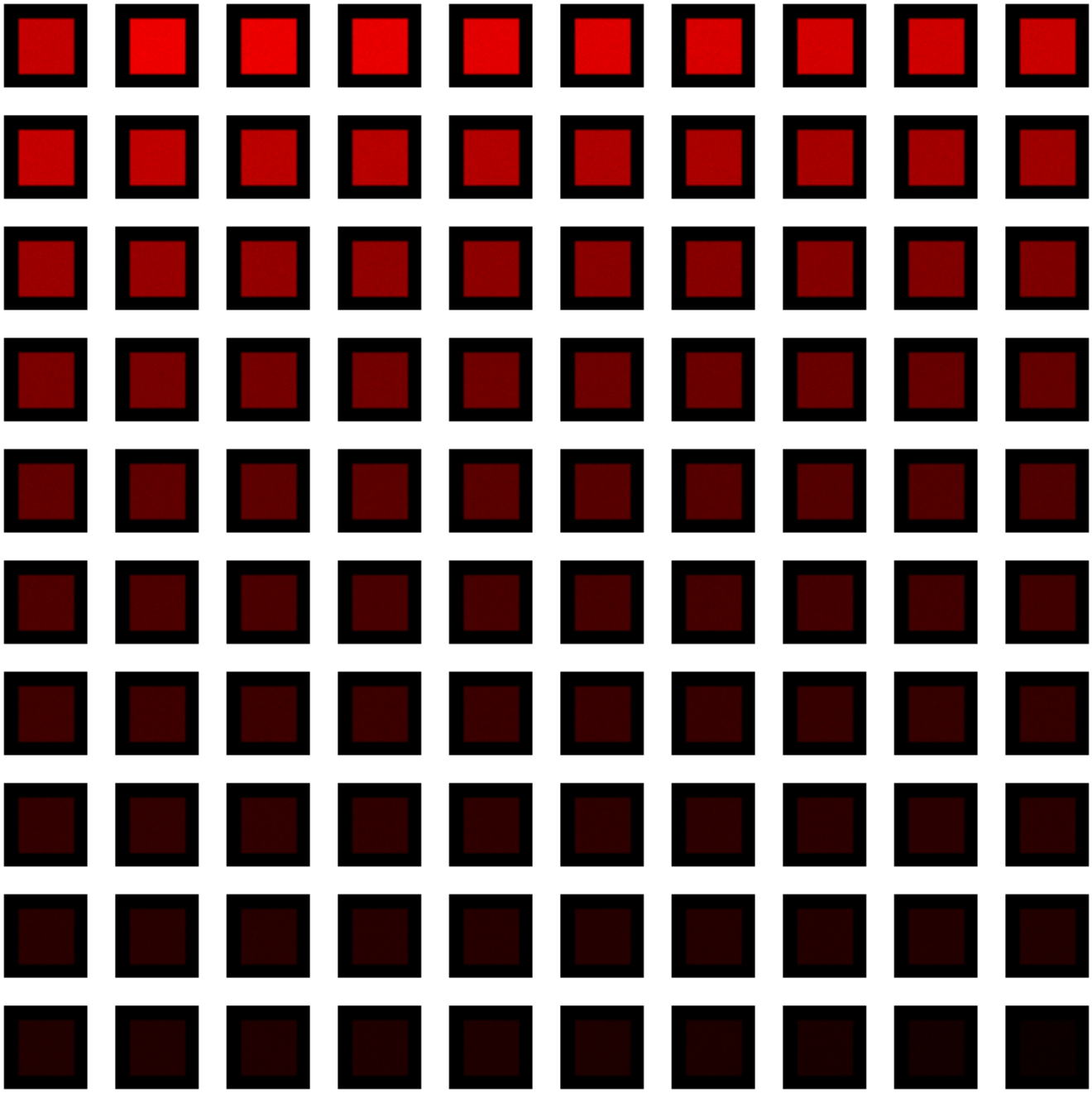


Figure 1.27: Dose montage for 100 million photons. Starting at top left and proceeding left to right, top to bottom are the horizontal slices in the x-y plane starting at depth 0 cm. Each slice is 0.5 cm thick.

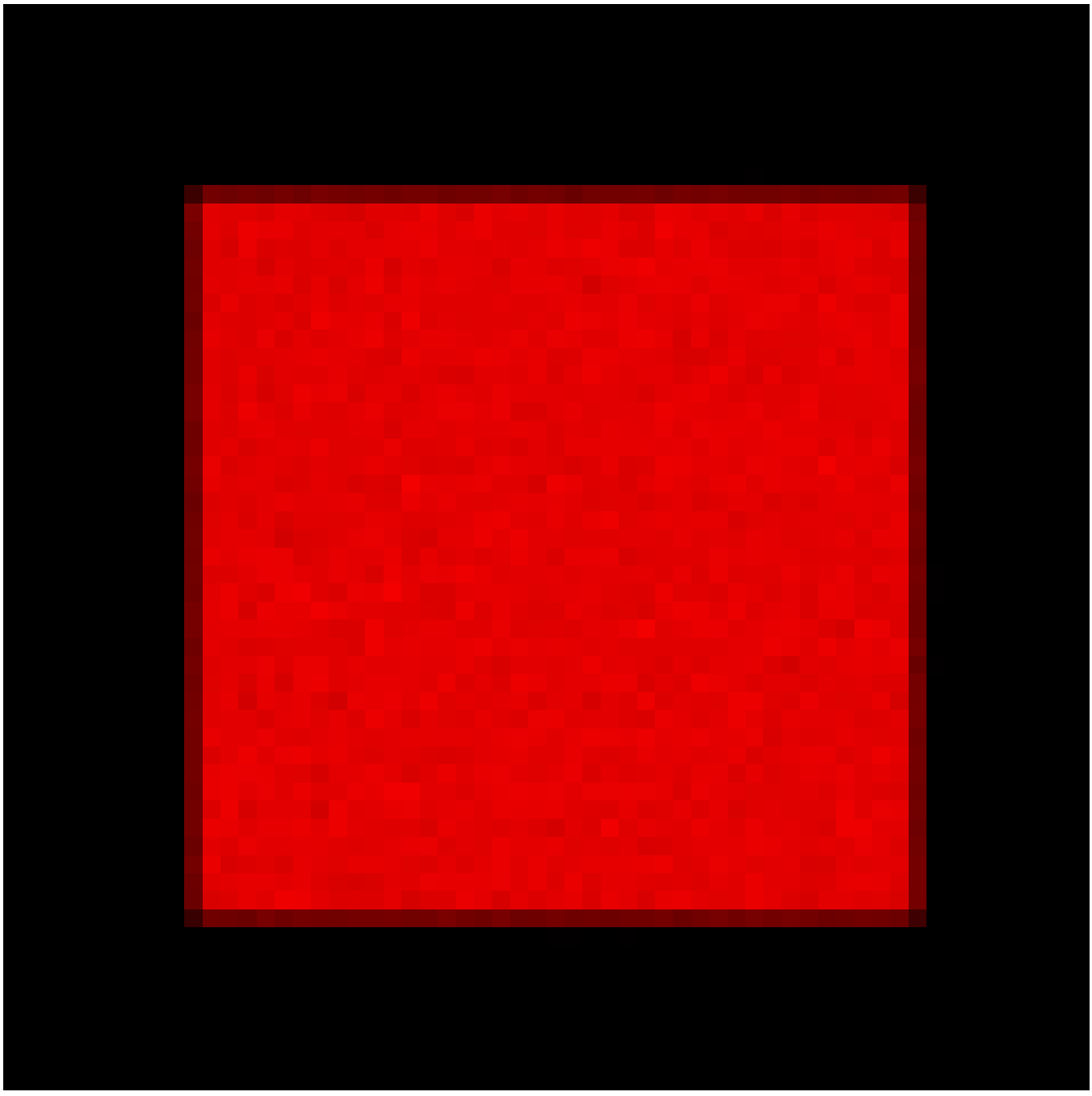


Figure 1.28: Dose in the x-y plane for 100 million photons at depth 2.5 cm. Note that there is a small amount of fringing due to charged particle transport around the edges.

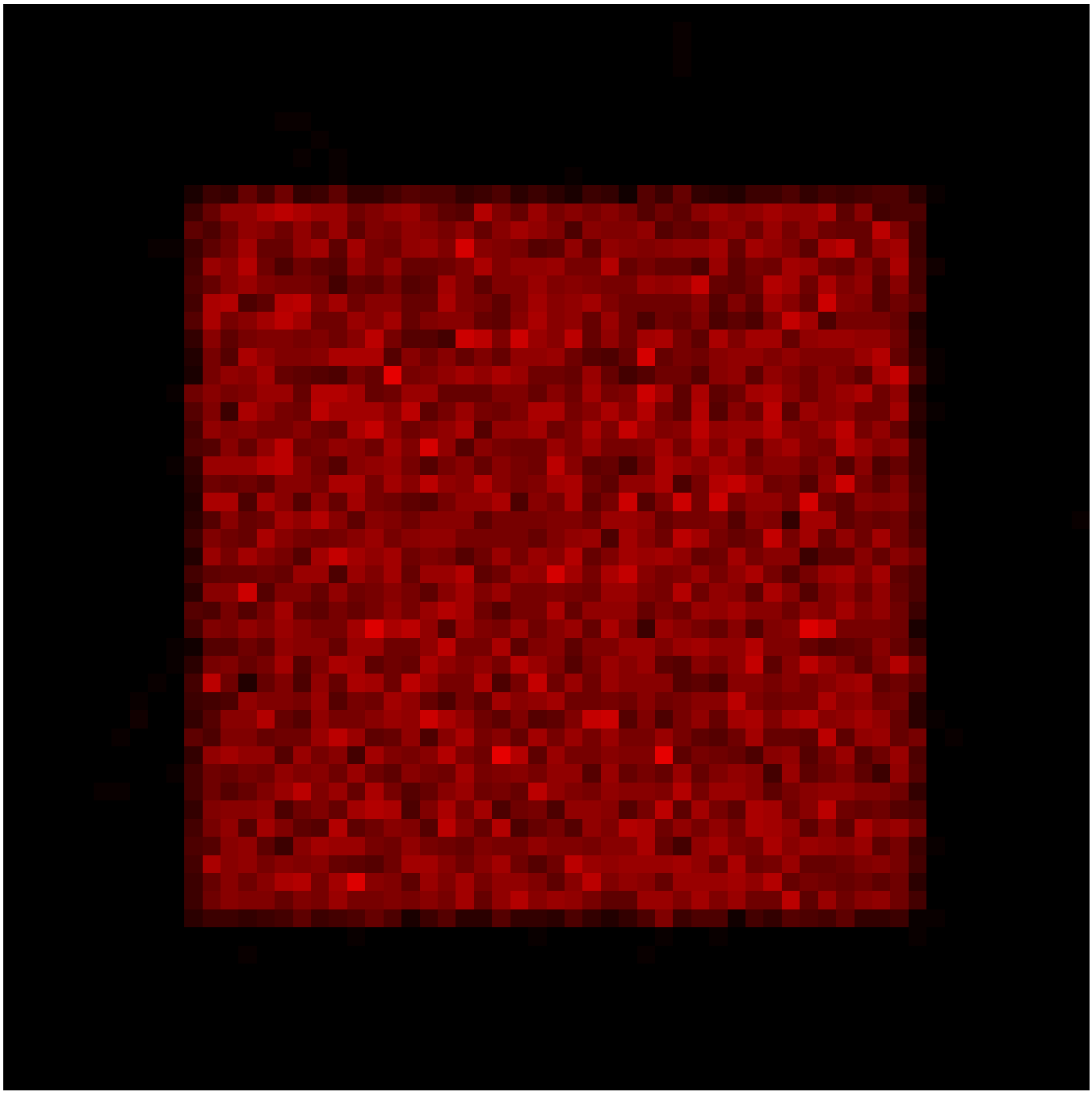


Figure 1.29: Dose in the x-y plane for one million photons at depth 2.5 cm. One can clearly make out the field size, but the variation is *enormous*.

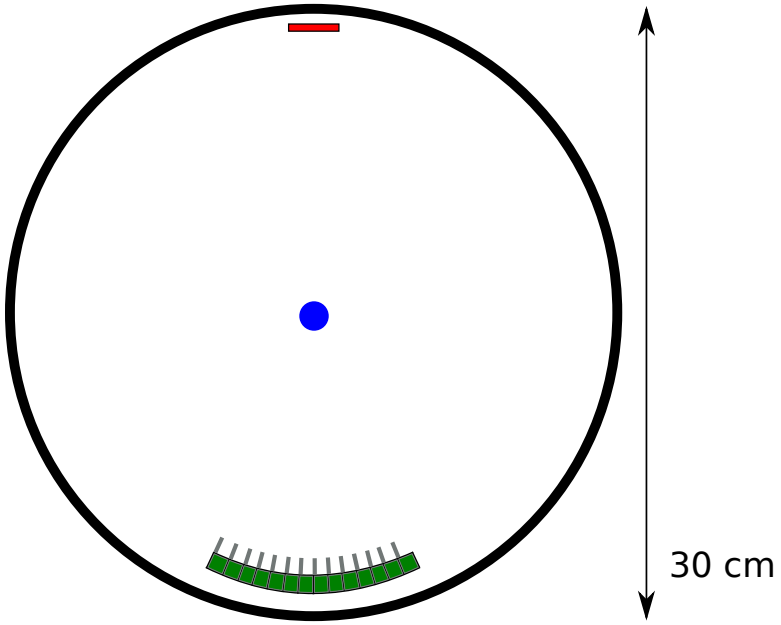


Figure 1.30: Simple geometry schematic diagram. The source is red, the detectors green, and the water sphere is blue. Note that the clearance is large compared with the detector length. Since we can rotate the detector, we can deal with objects larger than the detector width. .

In practice, it is difficult to achieve all three goals natively with the same format. For example, if we decide to output data in image format then we can rapidly view the data, but cannot easily process or plot the data without involving another program - often tying us to that program. If instead we output a series of plottable data, then we have a difficult time viewing or post-processing.

A solution to this issue is in the form of a text-based image format. One such group of formats are known as the Netpgm formats, which are extremely easy to understand. Essentially, instead of a binary image, the data is written in row-column RGB format. It can be read by humans, parsed by machines, and viewed as an image in most image viewers. Indeed, because the data is in row-column format, we can even directly plot the image data directly. Most importantly, we are able to use standard Unix text-processing commands to simplify data manipulation and processing.

This convenience comes with a price, though: the files can be quite large because they are plain text and have no means of compression. In practice this is not an issue because the voxels sizes are usually sufficiently small to produce data less than one megabyte in size. Further, excellent compression can be obtained using the standard DEFLATE algorithm (zip,7z,gz.)

Keeping with the mantra, of course, the module may be swapped out at the user's convenience.

Other applications

Many applications of this framework are possible. One such example, a complete, realistic, totally virtual CT scanner is introduced briefly here. This simulation was the result of a project in a concurrent class and used strictly this framework.

A 2 cm diameter water sphere was imaged using a simulated Hamamatsu $N7599$ x-ray tube source. A fan-beam detector was simulated, but was approximated as being *perfectly* efficient. Usable images were found with only five million photons.

Figure 1.30 shows the simulated geometry and a rough scale, and figure 1.31 shows a sampling of the detector geometry, including collimators and nine detector cells. The CT scanner is treated as being very thin (compared with the object being imaged,) but is not treated as being two-dimensional.

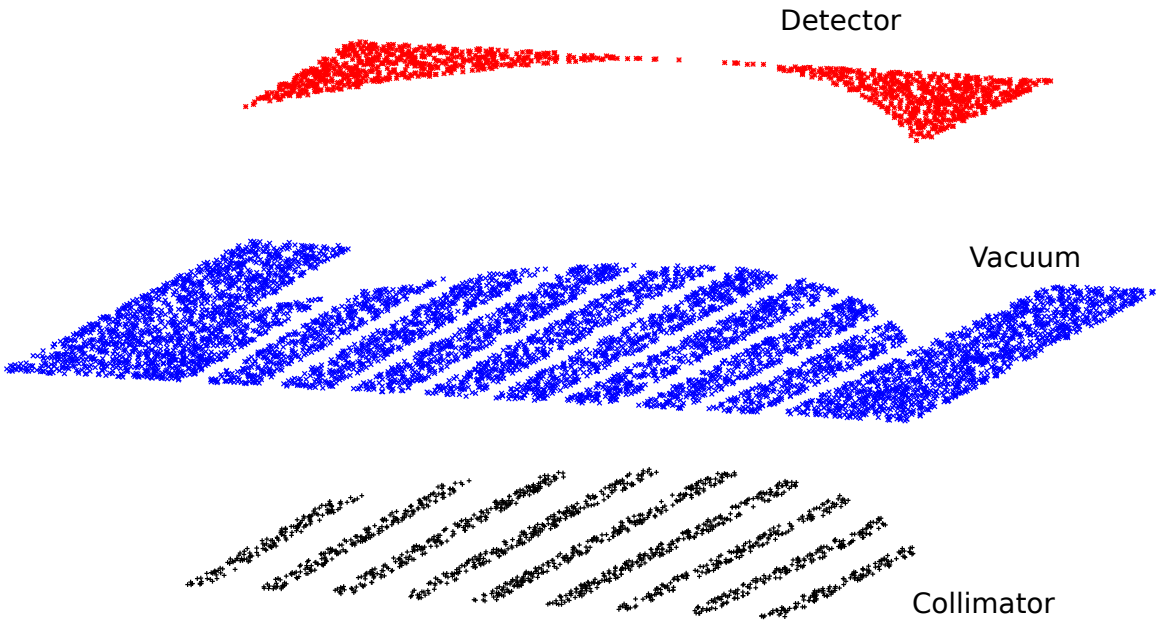


Figure 1.31: Simulated detector geometry. The collimators can be clearly seen. Note that both the collimators and the detector extend beyond the rectangular region sampled. Both regions have a thickness of 1 cm.

Propagating photons through the CT geometry, through the water sphere, we can collect photons on the opposing side to determine how many have been attenuated within the object. Performing a virtual calibration and normalization, we see that the event counts form a nearly perfect (if not, noisy) half circle. This is *precisely* what we would expect for a circular object being imaged by a thin scanner since the attenuation will follow the thickness of water "seen" by each photon. This data is shown in figure 1.32.

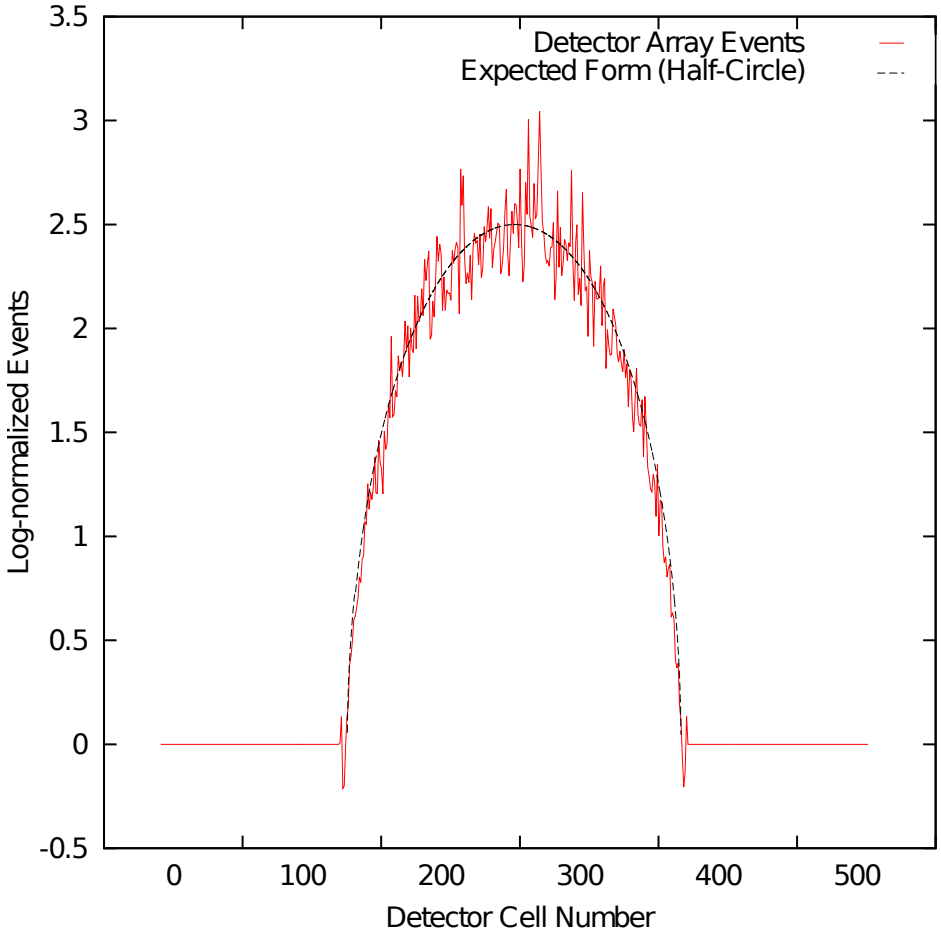
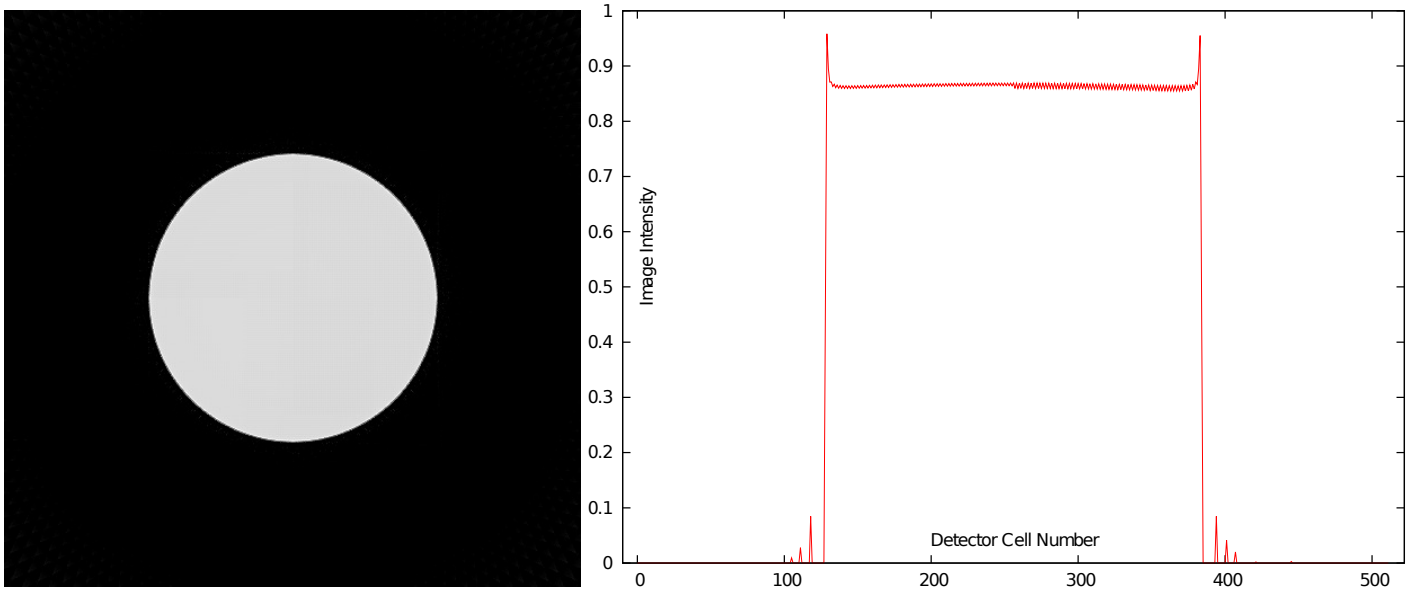
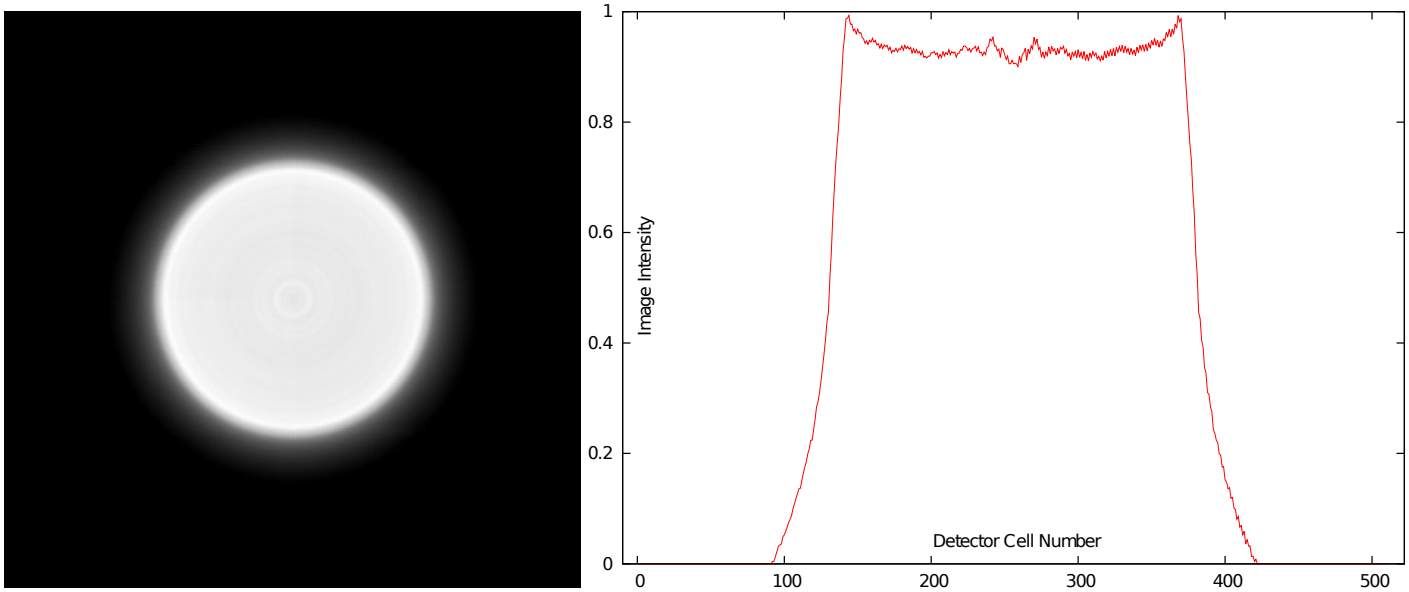


Figure 1.32: Log-normalized detector event counts for the fictitious xray spectrum (this computation only used $500k$ photons.)

Having collected events in succession around the object (in 2° increments,) we can use a filtered-backprojection reconstruction to produce an image. The image produced is shown in figure 1.33b along with the *perfect* image we should see. As can be seen, the reproduction is faithful, and we managed it by using only five million photons. The detectors, CT rotation, the geometry of the object to be imaged, and the xray source all required a single module each. The modules are not dissimilar to those of the modules required to simulate the $6\,MV$ beam in a water tank, and inspection of the source files should convince the reader of the ease in which very different simulations can be performed with minimal changes to the code.



(a) Filtered back-projection for perfect data. This is the signal which we would hope to see for an infinite number of simulated photons.



(b) Filtered back-projection with a deviated ramp filter and additional 3-neighbor nearest-neighbor averaging performed post-ramp filter. The image is clearly recognizable, and the largest discrepancy is broadening about the edge of the sphere. Some of this broadening is attributable to the backprojection technique.

Figure 1.33

Afterword

Hopefully the utility of the framework introduced here has been demonstrated, if not for quality of computation or speed, then for the ease in which computations can be performed.

There are several places the author plans to continue work on this framework. Some specific aspects which will be investigated by the author are listed below, but obvious items like optimization and parallelization are also planned.

1. Handling of non-static stopping powers. Some parts of the code rely on a completely static stopping power for the CSDA interaction type. In general, this should be handled using stopping power at the present energy instead.

2. The use of $PEGS*$ data. Several high-quality sources of data which could be used in these types of simulations are freely available and well-studied. Further, approximation techniques with this data are well studied. The author intends to write a module for interfacing this data and the framework.

3. Higher-order interactions. The computation of higher-order diagrams would lead to more interesting interactions between particles. Since low-loop tree-level interactions are straightforward to compute, and even easier to add to the framework, there would be little reason not to add them eventually.

4. Implementing the PRESTA algorithms for boundary crossings would produce more accurate computations in the vicinity of media changes.

5. A module for the handling of CT data would allow for real-world applicability - even if the computations are not clinically acceptable (which this framework will never strive to be.)

6. The creation of a brachytherapy module would be a wonderful third example modality to simulate. Having this, the framework would come with one (extremely basic) linac simulator, one basic imaging simulator, and one *in situ* source simulator.

As the author wishes to release the code under a libre license, it should be clear that suggestions, ideas, and criticisms are welcomed. The primary place to do so is at the website where the code is hosted, but email and catching him in person are perfectly fine.

References

[1] Peter J. Mohr, Barry N. Taylor, and David B. Newell, *Codata recommended values of the fundamental physical constants: 2006*, Rev. Mod. Phys. **80** (2008), 633–730.

[2] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery, *Numerical recipes in c++: the art of scientific computing*, 2nd ed., Cambridge University Press, New York, NY, USA, 2002.