RADIOBIOLOGICAL MODELING USING TRACK STRUCTURE ANALYSIS

A Thesis
Presented to
The Academic Faculty

by

Matthew T. Coghill

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Medical Physics

School of Mechanical Engineering
Georgia Institute of Technology
August 2012

RADIOBIOLOGICAL MODELING USING TRACK STRUCTURE ANALYSIS

Approved by:

Professor Chris Wang, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Professor Sang Cho
School of Mechanical Engineering
Georgia Institute of Technology

Professo Nolan Hertel
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: 8 May 2012

To my parents,

Tim and Karen Coghilll,

who raised me and have provided

for me throughout.

PREFACE

The modern availability and usability of a comprehensive nano-scale track-structure code such as GEANT4-DNA allows scientists, engineers and researchers to begin to examine the fundamental nature of radiobiology from a new perspective. This thesis is meant to help form the basis for future efforts in making the connection between effects of radiation on biological material and its physical action on a nanodosimetric scale through computer simulation. Perhaps its purpose is best described by the Latin phrase:

in specialibus generalia quaerimus

The study of the effects of radiation on biological material has traditionally been an outside-in field of research. Cells were irradiated and effects observed. The phrase, meaning "to seek the general in the specifics", describes the potential of nanoscale track-structure analysis to provide foundational reasoning to back these experimental observations and eventually form a more accurate and comprehensive predictive model.

ACKNOWLEDGEMENTS

I want to thank my advisor Dr. Chris Wang as well as members of my research group Brian Lee, Corey Ginetz and Spenser Lewis.

I would also like to thank my roommate, Uriah M. Clemmer IV, who put up with my crunch-time shenanigans and odd working hours.

I want to acknowledge the GEANT4 collaboration as well as the GEANT4-DNA working group. Without their work this thesis would not be possible.

Contents

List of Tables

List of Figures

SUMMARY

The promise of comprehensive, nanoscale, characterization of radiation track-structure is a basic, physical level understanding of the relationship between the characteristics of a given radiation and the damage caused. The effectiveness of radiation in achieving biological endpoints is known to differ with linear eneryg transfer (LET) as well as particle type. This thesis examines this multi-dimensional relationship using track-structure analysis and document the differences in complexity of damage produced by each type of radiation at various lineal energies by using cluster analysis of certain energy depositions. An effort is also made to determine if the implementation of a basic chromatin fiber geometry offers any benefits over previously suggested methods for this type of work.

The author presents, in the following chapters: motivations for the construction of this thesis; background for the adoption of track-structure analysis; methods for the production of results; the ultimate results of simulation and cluster analysis; conclusions of these results; and a brief summary of recent developments and suggestions for future work.

Chapter I

OBJECTIVE AND MOTIVATION

Few developments have happened faster than the application of radiation to medicine. The mysterious ability of Roentgen's x-ray was observed. Followed soon by crude radiographs, it was less than a year before the presence of rashes as a result of exposure had led to the dermatological application of x-rays. Marie Curie's discovery of the first radioactive isotopes predicated the introduction of radiation in an oncological setting. While many of its effects were apparent, their extent and the underlying mechanisms were hardly understood. Examination of the biological effects of radiation has increasingly moved to a smaller scale. Investigation of general effects progressed to cellular experiments concerning the different effects of various types of radiation on cell cultures.[7][8] The scale continued to shrink as Kellerer and Rossi proposed the dual radiation action theory [49] which posited a precise connection between the physical action of radiation on a sub-cellular scale and resultant biological effects giving rise to the field of microdosimetry.[35] Much in opposition to the investigation of average, macroscopic quanties in dosimetry, microdosimetry focuses on an investigation of stochastic of microscopic interaction.[37] The field of nanodosimetry is yet another progression as investigators focus on the action of radiation on the nanometer scale in an attempt to better understand radiation as a whole.

Radiation is defined as a process by which energy is carried, by waves or particles, through a medium. It is typically classified in in two ways: ionizing and non-ionizing. Ionizing radiation, due to its sufficient energy, can liberate electrons from atoms. This ability to induce modifications to its medium has significant biological consequence. Each ionization event is the result of energy transfer from the incident particle. The

rate at which a particle will induce ionization events is dependent on its mass and effective charge. Linear Energy Transfer (LET) is a measure developed to quantify the energy transferred to the medium, by secondary electrons, from the incident particle.

The general effects of radiation on organisms have been well studied. It has been established that radiation can affect cells: their function, their reproductive characteristics, and ultimately their life-cycle. Experiments have also determined that the primary target, by which radiation exerts its effects, is DNA.[63] Ionization events in the nucleus can result in direct and indirect damage to the genetic material of the cell. The connection between the absorption of energy from radiation and biological effect is evident. However, experiments and studies have shown that the effects of radiation can also vary dependent upon the type of incident particle. Relative biological effectiveness (RBE) is a measure that has been established to qualify the different effectiveness of damage caused by various types of ionizing radiation. It is defined as the ratio of the dose of a standard photon beam (250 kVp x-ray beam or Co-60 gamma-ray beam) to the dose of the test beam that is necessary to produce the same level of biological effect (it is an iso-effecitve dose ratio)[40].

The connection between RBE and LET is important. The biological effects of radiation are intrinsically linked to the ionization of cell nuclear material. The ionization of nuclear material is, in turn, derivative of the energy transfer from an incident particle to the medium.[10][9] The difference between RBE of various particles with similar LETs has been established.[67] While a connection with ionization density and secondary particle production has been proposed, the underlying interactions that form the basis of this connection are not necessarily understood.[11]

The modern development of comprehensive particle track-structure simulations in water allows the nanoscale examination of the behavior of various radiations and their associated secondary particles (most importantly electrons) and reactive chemical species.[38] Through the establishment of various parameters attempts can be made

to, in part, explain the characteristic effectiveness of different types of radiation.

The comprehensive, nanoscale, characterization of radiation track-structure has been an objective of researchers for more than three decades.[35] Monte Carlo codes, of varying levels of detail, have been available since the 1970s.[62] Their development has followed, not coincidentally, the historical increase in computing power available to the average investigator. It is due to this computational limit that many of these codes did not examine low energy electromagnetic processes involving charged particles. A small number of research groups developed proprietary, in-house codes. However, few of these have propagated due to their usability, narrow focus, and the prior stated limit of computational power. The development of GEANT4[2] and its release in the late 1990s represented a significant improvement in the capability of detailed track-structure code. Its open-source, object-oriented technology provided substantial versatility and opportunity for expansion. A desire to more accurately model radiation at the nanoscale level led to the creation of a GEANT4-DNA toolkit.[44] This toolkit implements processes for the simulation of low energy electromagnetic processes. Integrated into the primary GEANT4 tookit it has provided a significant new avenue for the comprehensive, nanoscale investigation of radiation track-structure.

In this thesis the particle transport simulation toolkit GEANT4 has been used by the author to simulate the incidence of various radiation particles upon a hypothetical cell nucleus in an effort to connect the observed facts of the RBE-LET relationship with data derived from Monte Carlo track-structure simulation. Chapter two of this thesis provides a general background of the rationale behind this work as well as details of the programs used for the execution of the methods. Chapter 3 describes the methods used for the production and processing of data. Chapter 4 and 5, respectively, detail the results of this work and the conclusions of the author. The final chapter describes the author's ideas and suggestions for optimization, continuation and further development of this work. Included in the appendix is the author's code

as run in GEANT4 as well as the R code used for processing of the data outputted by GEANT4.

Chapter II

BACKGROUND

2.1 Radiobiology and Particle Track Structure

It is certain that ionizing radiation can lead to biological damage. While the effects of radiation are readily evident, the underlying mechanisms are not. In consideration of biological effects, it has been determined that the principle target for radiation is the deoxyribonucleic acid (DNA) inside the cell nucleus.[63] The blueprint of the cell, DNA is collected in much larger units called chromosomes. DNA exists as two complementary strands entwined in a double-helix formation. The specific arrangement of four DNA bases (adenosine, cytosine, guanine and thymine) constitutes a collection of discrete genes which, together, provide instruction for the functional processes of a living cell. During interphase, chromosomes are compartmentalized inside a cell nucleus with each one having its own territory. The structure subunits of a chromosome include: nucleosomes, chromatin fibers, and chromatin fiber. Each of these structures, illustrated in figure 1, loops facilitates the packing of a large amount of DNA into the relatively small cell nucleus. Various structure proteins, predominantly the family of histones, coordinate strands of DNA into a dense arrangement.

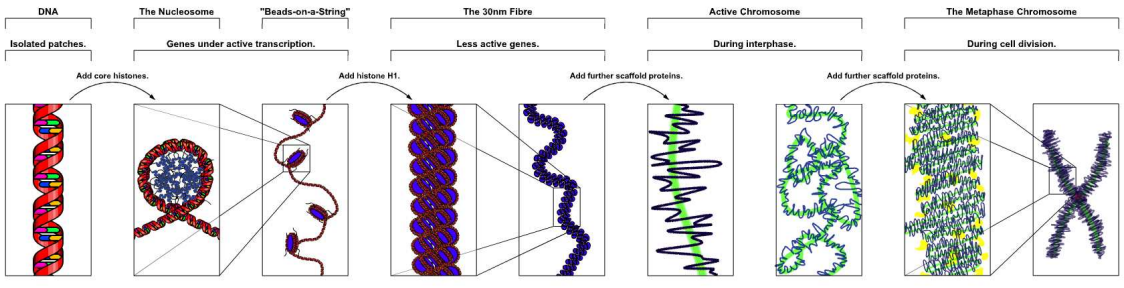


Figure 1: The structures of genetic material in the nucleus.[77]

The action by which radiation causes damage in cells is divided into two classifications. When the radiation interacts directly with the cell target (DNA) it can cause ionization or excitation events that lead to biologic consequence. This direct action is the dominant process for high-LET radiations. Radiation may also act indirectly by affecting other molecules in the nucleus. Water, the primary constituent material of the cell medium, is of particular concern. The radiolysis of water can result in the formation of several different free radicals such as $^{\bullet}OH$, e^{-}_{aq}, $^{\bullet}H$, and H_3O^{+}, which may diffuse over short distances (a few nanometers)[79] and can chemically attack and cause damage to nearby DNA. Their interactions with DNA can result in biologically important changes in DNA structure. The comparative importance of indirect action and direct action are dependent on the type of incident radiation, the LET of the incident radiation, and the structure and organization of the chromosome.

Ionizing radiation, through direct and indirect effects, can cause damage to the structure of chromatin. Incident radiation and/or free radical (primarily OH*) interactions can cause base aberrations (base deletions, base modification and sugar cross-linking) as well as single-strand breaks. These damages have varying degrees of severity but share one common trait: their coincident occurrence has significant impact upon the ability of the cell to successfully repair the damage. Single-strand breaks (SSB) are of little biological consequence as they are easily repaired by the cell using the complementary strand as a template.

Significantly more important when considering biological effect are double-strand breaks (DSB). A DSB is defined as the coincident occurrence of two SSBs on opposite strands less than 10 base pairs apart. DSBs are significantly more difficult for the cell to repair. Furthermore, the occurrence of multiple lesions of base damages, SSBs and/or DSBs in a general area increases the severity of the damage and decreases the effectiveness of the cell's repair mechanisms. These clustered DNA damage sites

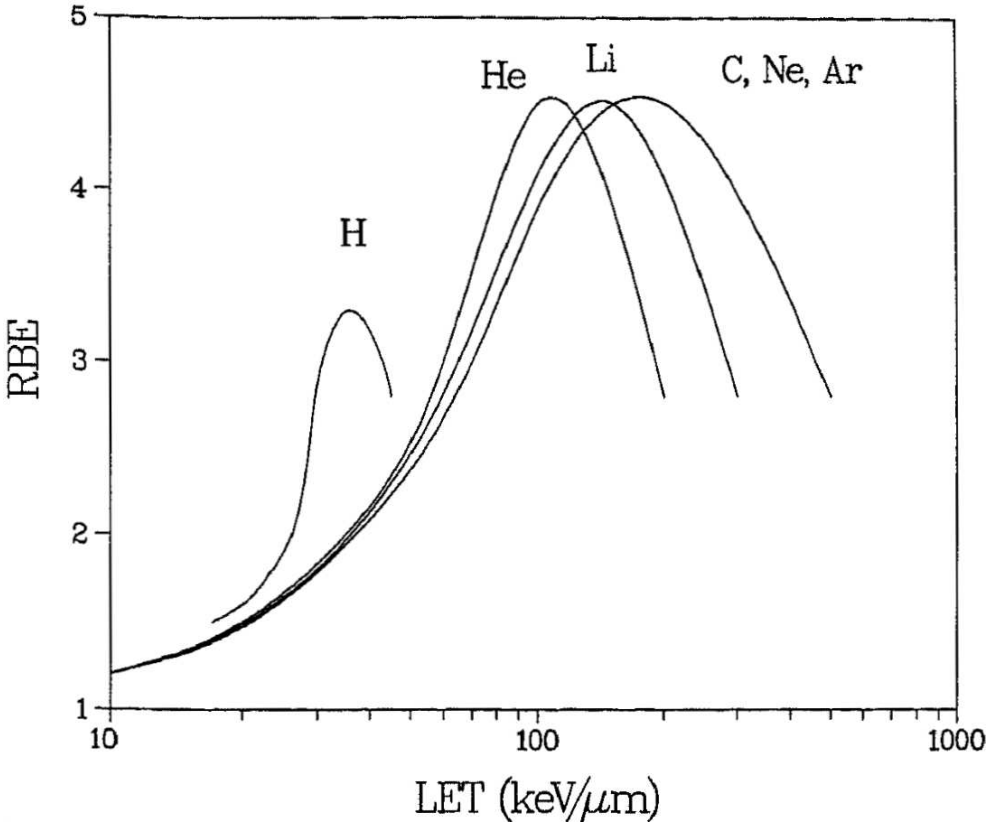


Figure 2: Relationship between RBE and LET based on experimental data. From Raju.[67]

are termed "locally multiply damaged sites (LMDS)" and are preferentially associated with high-LET radiations.[36] All of these damage sites, if left un-repaired or mis-repaired by the cell, may have significant biological consequences including chromosomal aberrations, cell transformation and, ultimately, cell death. It has been established that RBE is coordinated to a radiation particle's LET.

The variance of RBE is not, however, simple a function of only LET. As shown in figure 2, RBE can vary significantly for different particles of similar LET. RBE is actually dependent on ionization density[11] which, while intrinsically related to LET, is also dependent upon particle mass and effective charge. Nanoscale track structure analysis allows the definition of the physical difference in ionization density for various particles with similar LET and will possibly allow these effects to be coordinated to chromatin organized to uniquely determine an RBE value for an incident radiation.

2.2 Monte Carlo Simulation of Particle Transport and Interactions

The Monte Carlo Method is a "statistical approach to the study of differential equations." [58] It makes use of repeated sampling of pseudo-random number in an attempt to determine the behavior of a probabilistic event. The stochastic nature of radiation transport lends itself well to simulation in this manner. Monte Carlo simulation are computationally expensive. The small scale of DNA target has limited the use of track structure modeling for nanodosimetric purposes and it is only with time that the feasibility of simulation of track-structures at low-energy has become statistically feasible on a large scale.

The early progression of track structure codes for biophysical modeling is well documented by Nikjoo, et al.[62] Their focus is on modeling, in detail, the track-structure of radiation in an aqueous environment (like the cellular medium). The differences between track-structure codes lies primarily in the theoretical models and cross-sections that each codes uses.(Nikjoo, *et al.* summarizes this well)[60] The agreement of most track structure codes is good at high-energy; however, at lower energies the consensus becomes less clear.

The uncertainty of cross-section data for low-energy electrons in water is problematic. Their influence upon the ultimate spatial distribution of energy deposition is certainly significant. Existing cross-sections for low-energy interaction in liquid water are derived from water vapor experiments. The effect of phase difference on yield is significant The theoretical assumptions made by the authors of each code in deriving these cross-sections have appreciable impact upon the nanoscale action of the codes.[73]

The production and subsequent diffusion of reactive chemical species during the energetic degradation of secondary electrons has a significant nanoscale effect on the

energy deposition distribution of radiation. The coupling of the actions of these reactive chemical species to radiation particles is an ongoing effort. As early as the late 1980s the authors of the track-structure code OREC[72][71] made this coupling. The coupling of the physical and chemical actions of radiation involves a significant amount of uncertainty. The simulation of chemical actions, specifically diffusion of radicals and their cross-interactions, requires a temporal component not otherwise accommodated in other detailed track-structure codes.[16] Originally developed in the late 1990s, the detailed track-structure code PARTRAC[27] models the time-dependent interaction of reactive chemical species with DNA material.[28] A benchmark comparison between NOREC (a revised code based on OREC)[70] and PARTRAC shows only small difference possibly attributable to the developments in the understanding of input parameters for interaction in liquid water.[17]

The development of the GEANT4-DNA toolkit[44] represents the evolution of low-energy Monte Carlo simulation. GEANT4's design and versatility brings these simulations into the mainstream. The recent integration of a radiation chemistry module, developed for the PARTRAC code, into GEANT4 will provide users with an even greater capability to simulating particle interaction on a nanometer scale.

2.3 GEANT4 in General

GEANT4 is a toolkit for the simulation of particles through matter.[2] Its origin, as a development, can be traced back to studies done at CERN and KEK in the early 1990s.[4] These studies proposed ways to improve upon the existing GEANT3 Monte Carlo code as well as other existing Monte Carlo code by taking advantage of modern computing techniques. The Original development at CERN, known as RD44, quickly adapted object oriented technology and the C++ programming language. Its first release, in December 1998, as GEANT4, led to the formation of the GEANT4 Collaboration in January 1999. It was created to address the demand for comprehensive

simulation of particle detectors used in large and complex nuclear physics experiments. (involving significantly higher energies than often concern the radiobiologist). Its development was made feasible by the increase in the user base of comprehensive nuclear track-structure simulation software facilitated by the proliferation of low-cost computational power. The design of GEANT4 was meant to address the shortcoming of many previously available codes. Its use of object-oriented technology implemented in the modern C++ programming language allowed modularity and versatility not present in previously available codes. GEANT4 is also distinguished by the large scale of the backing developmental collaboration and the ready availability of its source to all users.

Since its development GEANT4 has continued to evolve and expand. The original toolkit implemented methods for handling the fundamental aspects of physical simulation: geometry, materials, particle interactions, particle generation, particle tracking, generation and storing of event and tracking data, and detector and tracking visualization. Form these building blocks the code has expanded wildly.[3] Its object-oriented nature allows the end-user to make use of the existing framework by instantiating useful class systems while maintaining a unique and personal application framework

The GEANT4 Collaboration has fostered further development by facilitating the formation of multiple working groups. These working groups, made up of relevant experts, are tasked with individually managing releasable components. The framework of the Collaboration allows new working groups to be created and existing ones maintained in am manner that allows GEANT4 to organically develop and progress. Since GEANT4's introduction these working groups have developed new, additive, toolkits which have expanded the capabilities of the original program. These expanded capabilities are targeted to address shortcomings and new experimental applications and include: new methods for modeling geometries, new underlying physical models,

physics for addressing new particles, and physics for additional energy ranges.

The Collaboration also actively supports a web portal which facilitates the dissemination of new versions of the GEANT4 toolkit, relevant documentation, reference material, and answers to frequently asked questions. In addition the Stanford Linear Accelerator Center (SLAC) provides an internet-based forum for the congregation and interaction of the GEANT4 user base with each other as well as with members of the GEANT4 Collaboration.

At its core GEANT4 is a repository of much of the world's knowledge of particle interactions. The ease with which new physical models are added to GEANT4 has allowed the toolkit to develop and evolve with new research. GEANT4 is capable of accommodating multiple physical models. While default setups are given in abstract base classes, the end-user is left with an unlimited array of options to customize model usage. The user can choose models, define the ranges of their usage, and even modify their behavior. These are simply the capabilities provided to the application developer. An advanced user can modify the basic function of processes and incorporate personal models. This modular versatility is rooted in GEANT4's adoption of object-oriented technology. Many previous codes are complex, unwieldy and opaque and further development, manipulation, and replacement of physical models can be difficult and sometimes impossible without significant redesign.

Each process invoked in GEANT4 is typically well documented by published work of the GEANT4 Collaboration. These documents make liberal justifications for model usage and provide the original publications demonstrating the foundation for each model. It is possible for a user, relatively nave in the intricacies of object-oriented coding, to review and understand the basic operations performed by the classes associated with each physical model by investigating the associated source file.

2.3.1 For the End-User

When designing an application, as an end-user, there are three user action classes whose instantiation and registration is mandatory. GEANT4 provides an abstract interface for these user action classes.. The necessary classes are: *G4VUserDetector-Construction*, *G4VUserPrimaryGeneratorAction* and *G4VUserPhysics*. The DetectorConstruction and PrimaryGeneratorAction classes provide for the registration of the system geometry as well as the geometry of the associated particle source. While the registration of geometry will not be foreign to any Monte Carlo veteran, the necessity of the UserPhysicsList might be. GEANT4 has no default physics processes. Instead the user must register the physics process(es) that he/she wishes to use. Without the registration of these processes GEANT4 cannot transport particles of any type or energy. This requirement for verbosity is derivative of the facility for easy manipulation of transportation process and physics process handling. GEANT4 does provide abstract base classes which invoke a standard set of model with default energy limits. Examples included with the GEANT4 distribution also provide a number of PhysicsLists with widely varying purposes and constructions.

There are five further optional user action classes. It should be noted that these are simply abstract interfaces through which the user-designed application can communicate with the core GEANT4 program. The three mandatory and five optional user action base classes are listed below with their descriptions. They are listed in similar order to the description provided in the top-level diagram shown in figure 3.

- *G4VUserDetectorConstruction* for defining the material and geometrical setup of the detector. Several other properties, such as detector sensitivities and visualization attributes, are also defined in this class.

- *G4VUserPhysicsList* for defining all the particles, physics process and cut-off parameters.

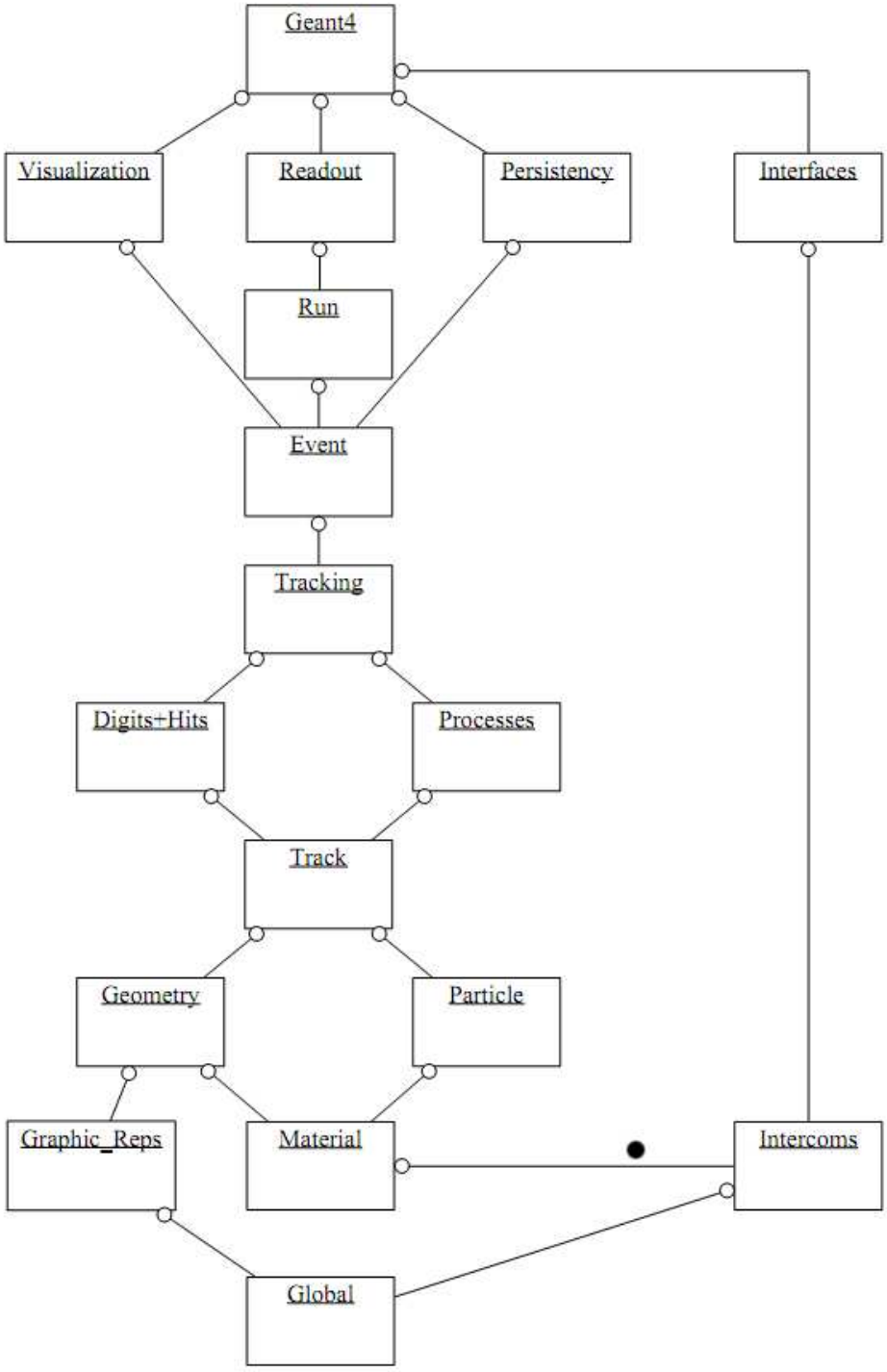


Figure 3: GEANT4 toolkit Top Level Diagram (TLD). Adapted from Agostinelli, *et al.*[2] The dependencies move in a single direction and each addresses certain aspects of simulation.

- *G4UserPrimaryGneeratorAction* for generating the primary verifices and particles.
- *G4UserRunAction* for actions at the beginning and end of every run.
- *G4UserEventAction* for actions at the beginning and end of every event.
- *G4UserStackingAction* for customizing access to the track stacks.
- *G4UserTrackingAction* for actions at the creation and completion of every track.
- *G4UserSteppingAction* for customizing behavior at every step.

When establishing the physics list for a given application the user must define, in one manner or another: the physical processes for each particle type and all energies to be addressed. A corollary for this requirement is that the user is not required to define processes to address particle types and energies that will not be handled by the application. The user is also given the ability to manipulate various aspects of the physical models invoked such as energy cutoffs and the volumes where models apply. The user may also choose to instantiate models derived from user constructed classes within the application itself. (or user constructed classes integrated into the toolkit itself). This allows a customizable physical treatment of each particle at the user's discretion.

Physics may also be linked to various regions. These regions are typically defined in the DetectorConstruction class (where geometry is set) and may instantiate multiple (or a single) physical volumes. This allows for substantial variance reduction as computationally intensive physical models can be instantiated only where necessary.[1]

Range cuts are used in GEANT4 to allow the tracking and production of particle to be optimized for geometry and performance. A user-defined range cut instructs the application to not produce any secondary particles whose range is below the

[1] Advantageous for variance reduction in less-sensitive regions.

threshold. The ranges of particle of various energies are calculated using range-energy and absorption length-energy tables derived from the corresponding processes. The use of range cuts provides the user an easy method for altering the scale of the track-structure simulation.[2]

2.3.2 Physics Processes in General

GEANT4 incorporates a large library of models for handling physics of a variety of particles at a large number of energies. The GEANT4 Collaboration has published on most models incorporated into the toolkit.[15] The instantiation of the models included with GEANT4-DNA toolkit often supersede the less defined models of the original toolkit. These lower energy processes were not originally simulated in detail as their actions are unnecessary for the simulation of the high-energy physics experiments for which GEANT4 was originally built.

2.3.3 Physics Process in GEANT4-DNA

GEANT4-DNA introduces a new set of physical models in an effort to increase the definition of physical modeling at lower particle energies. Certain processes, handled vaguely by standard physics, are treated by models with increased detail and very low energy electromagnetic processes are introduced.[3] Even a basic outline of physics models implemented in GEANT4-DNA is impractical in the scope of this work. Ultimately large reference works could be written describing these models in detail. Currently GEANT4-DNA handles the transport of electrons, protons, hydrogen nuclei, alpha nuclei (with various charges) and four heavier ions (carbon, nitrogen, oxygen and iron ions). For review of the physical models implmented in GEANT4-DNA the author recommends the publication of Incerti, *et al.*, Villagrasa, *et al.*, Chauvie, *et al.*, Incerti, *et al.* and Incerti, *et al.*[45][75][13][44][14] Each of these publications is

[2]Range cuts allow scale to be changed quickly without the need to translate the desired scale to energy cutoffs for each particle individually.

[3]These very low energies were not previously treated.

authored by members of the GEANT4-DNA working group and include a review of the physics processes implemented by GEANT4-DNA with extended references to the experimental, theoretical and analytical works which provide the basis for the various physical models.

2.3.4 Primary Generator in General (particle generator)

Primary particle generation in GEANT4 is instantiated by the mandatory user action class *G4VUserPrimaryGeneratorAction*. This class requires the definition of two instances, *G4PrimaryVertex* and *G4PrimaryParticle*. The former states the particles starting point in space and time. The latter states the primary particle's type, initial momentum and other characteristics.

2.3.4.1 Particle Gun

The original method for generating particles in GEANT4, *G4ParticleGun*, is designed to simulate a beam of particles. The particle gun allows customization of particle type, energy, polarization, charge and number of particles shot per event. It is possible to affect the particle gun differently with random (or pre-defined) characteristics dependent on event number and other customizations.

2.3.4.2 General Particle Source

The *G4GeneralParticleSource* is a more advanced implementation of the particle generator.[22] It is included in the standard GEANT4 toolkit provides built-in support for spectral, spatial and angular distribution of generated primary particles. The General Particle Source is easily controlled within existing applications using a built-in command tree in the user interface.

2.3.5 Geometry in General

The geometry of the application is described in the user's application's DetectorConstruction class. In this class can articulate a wide variety of geometries. GEANT4 is a

STEP compliant code. STEP (Standard for the Exchange of Product model data) is a standard protocol (ISO 10303) for the exchange of geometrical data. GEANT4 makes use of the EXPRESS description method defined in STEP. The implementation of the STEP standard allows interchange with most CAD software.

As a STEP compliant code GEANT4 supports multiple methods for representing solid objects. The immediately relevant methods are Constructive Solid Geometry (CGS) and Boundary Represented Solids (BREPs). CSG, a method involving the buildup of objects from multiple constituent solids, is normally easier to implement and provides for superior computational performance. BREP allows for the production of significantly more complex and exotic solids.[5]

GEANT4 provides a series of abstract classes for the building of solids using the CSG representation. These abstract classes define various three-dimensional primitives. The CSG primitive solids are: boxes, tubes, cones, spheres, wedges and toruses. GEANT also provides for more advanced solids using CSG representation. These solids are well documented in the EXPRESS description in the STEP manual. These classes facilitate (amongst many solids) the CSG representation of trapezoids, tetrahedra, tubes with hyperbolic profiles, paraboloids, ellipsoids, cones and tubes with elliptical cross-sections, twisted tubes and trapezoids, and extruded solids.

Chapter III

METHODS

This work makes use of two applications with slightly different methods for defining sensitive sites and data processing as shown in figure 4. The first application is similar to the work of Francis, *et al.*[21] and uses a simpler method making use of a rejection sampling technique to define which points have fallen on theoretical sensitive sites; heretofore this application, based on its use of a rejection algorithm, is referred to as the Rejection Application. A second application makes use of a predefined sensitive region for scoring damages and, based on its implementation of a basic chromatin fiber model, is referred to as the Chromatin-Inclusive Application. Each application makes use of cluster analysis to expose the density of significant energy depositions and the differences between particles of different types and energies. The two applications use slightly different input values in the cluster analysis algorithm (DBSCAN) in an effort to coordinate results. The two applications are run together in an effort to determine if a pre-defined sensitive region is beneficial when performing relatively simple statistical analysis of energy deposition interactions.[1]

3.1 Programs Used

Two programs were used for the primary production of results in this work. Geant4, a particle simulation toolkit, was used to simulate the transport of radiation particles through a model cell and output a comprehensively detailed track structure of primary and secondary particle as well as associated energy deposition events. This data

[1]This is a matter of open debate. Until available software allows more detailed simulation of damage to genetic material (and not statistical approximations) are more detailed model of genetic material may not offer a significant advantage in predictive value over a simplified statistical model.

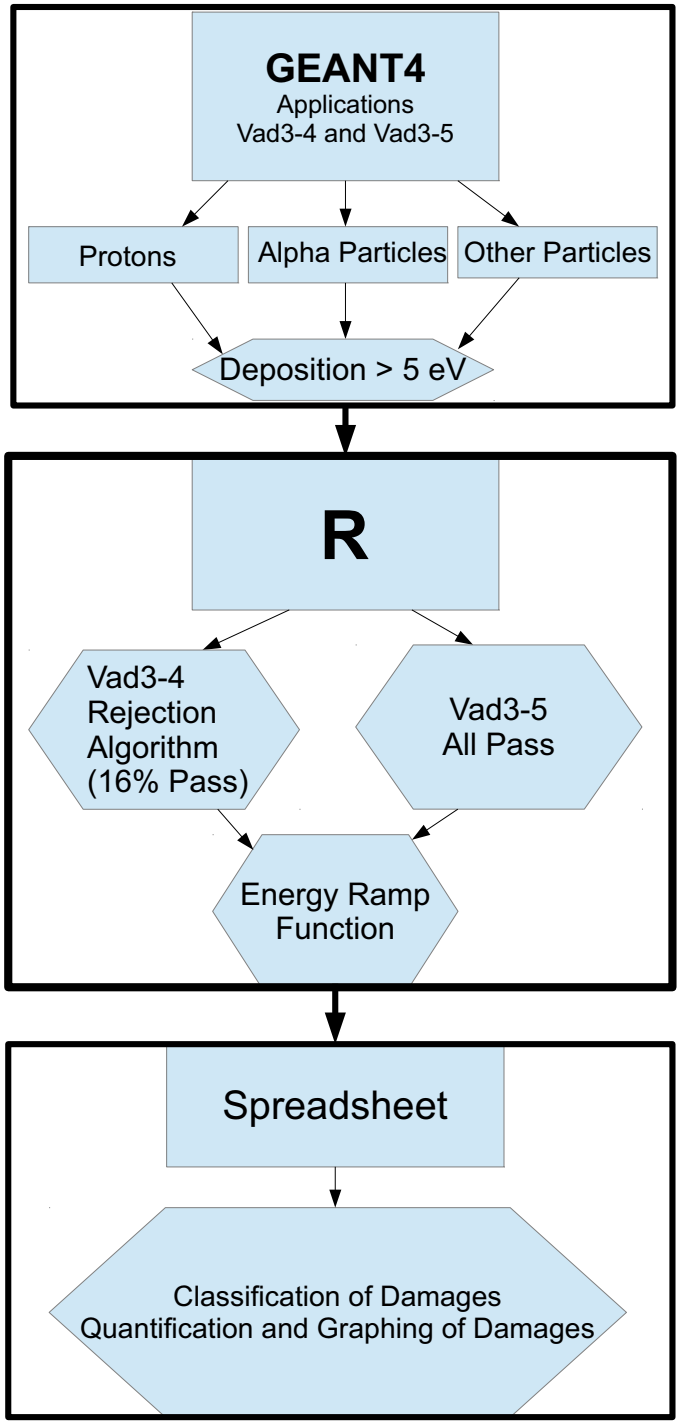


Figure 4: The general design of the data creation and processing underlying the production of this thesis.

was processed using the R language environment which is designed to "statistically explore data sets" and provides a simple user interface and large library of data mining algorithms.

3.1.1 GEANT4

The GEANT4 versions used by the author in the development of this work were GEANT4.9.4p1, GEANT4.9.4p2 and GEANT4.9.5. The production of final results was done using GEANT4.9.5. The author made substantial use of the examples provided with the distribution. These examples are invaluable assisting the novice user over the steep learning curve. Prior knowledge of object-oriented programming (OOP) (and C++ specifically) is certainly beneficial before introduction to the GEANT4 toolkit.[2]

GEANT4 is designed to be compiled using G++ in a Linux environment.[3] The ease of install install in LINUX varies with the user's Linux distribution. Most common desktop distributions will provide the potential user with some difficulty in finding and acquiring the plethora of packages needed to run GEANT4 at its full capability. GEANT4 can also be installed in Windows using Cygwin though the author has no experience with this type of installation.[33]

3.1.1.1 Geant4 VMWARE

The author settled on the use of image provided by Geant4@IN2P.[34] This images, easily implemented in the free VMware viewer, is kept up to date with current GEANT4 releases. It includes all major packages along with multiple ancillary programs recommended for use with GEANT4. This method of implementing GEANT4 has advantages and disadvantages. It allows the user to bypass the installation phase and updates and simply download the corresponding image to run the new distribution. The limitations of the VMware software possibly impose a slight performance

[2]However, it should be noted that this was the author's first experience with OOP.
[3]SciLinux5.5 is currently recommended by the GEANT4 Collaboration

penalty and limit the application of multiple CPUs. However, for the casual user the author wholly recommends the image distributed by Geant4@IN2P3.

3.1.2 R

R, a "language and environment for statistical computing and graphics" was used by the author to process the immediate output from GEANT4. A GNU project, R is based on the S language developed at Bell laboratories and is available from the website of the R Project.[1] Its primary utility to the author was for statistical processing of the GEANT4 output. The R environment provides a system for the handling and processing of large data sets along with a plethora of third party extensions implementing a variety of algorithms and function.

3.2 GEANT4 Model

The author has developed an application using the GEANT4-DNA toolkit. A single geometry setup was used for all trials presented in this work.[4] The materials of the geometry are homogenous thus, in a sense, the geometry of volumes is inconsequential relative to the definition of sensitive detector regions when considering the outputted results. The difference in the definition of sensitive regions also required differences in the methods for generating incident radiation particles.

3.2.1 Description of Geometry

This experiment uses an original physical geometry.[5,6] The geometry attempts to model a cellular nucleus with its associated genetic material. The genetic material is modeled with the assumption of uniform distribution throughout the nucleus. The

[4]It is quite important to note, especially in this work, that the instantiation of geometry does not necessarily reflect the scoring mechanisms or sensitive regions which are defined in a separate, though dependent, manner

[5]This original geometry was originally proposed by Dr. Chris Wang based upon prior knowledge of basic nuclear geometry and characteristics.

[6]The author places an emphasis on understanding the importance of separate definition of sensitive detector regions relative to physical geometry.

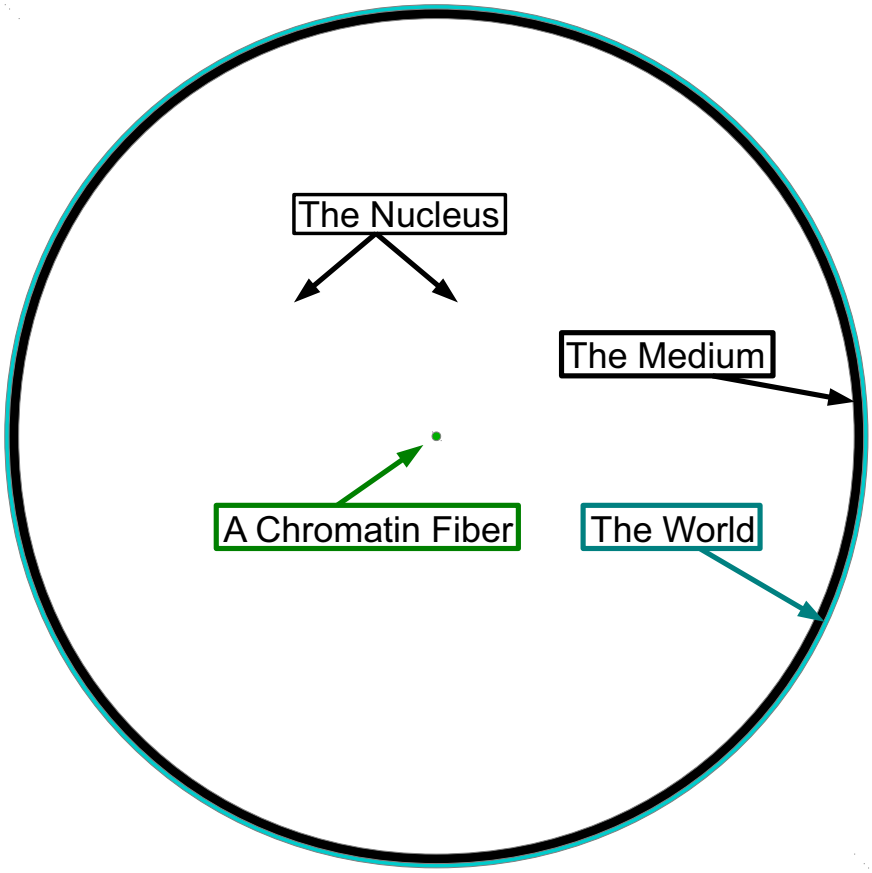


Figure 5: A top-down illustration of the arrangement of volumes in the author's applications.

geometric model makes use of identically sized cylinder representing strands of chromatin sized according to the basic characteristics of chromatin.

The basic model geometry is shown in figures 5 and 6.[7] There are 4 volumes defined in this work. The material defined in each volume are identical and each volume exists only to allow for flexibility in definition of sensitive volumes (scoring regions) and customizable instantiation of physics models.

[7]Figures 5 and 6 show only a single fiber along the center-line. This is done to demonstrate scale. The actual geometries used for the author's applications incorporates a number of chromatin fibers distributed throughout the nuceleus volume.

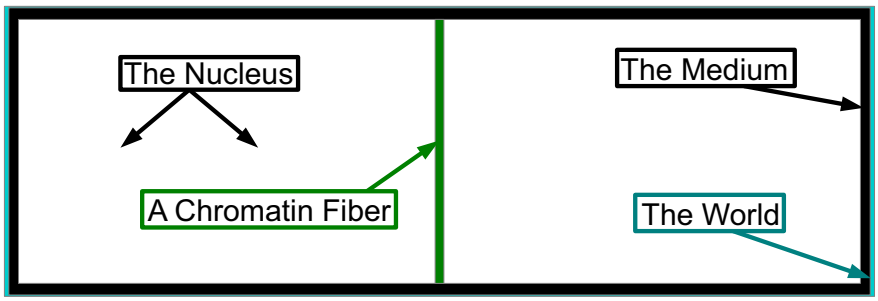


Figure 6: A cross-sectional illustration of the arrangement of volumes in the author's applications.

3.2.1.1 The Nucleus

The nucleus volume was modeled as a simple cylinder.[8] The nucleus volume was constructed on a 1:5 scale as a 1 micron height and 3 micron diameter simple cylinder.[9] The scaled size of the nucleus was implemented in an effort to reduce variance.

The nucleus volume was modeled using the G4Tubs function built into GEANT4. This function facilitates the creation of tube-like solids. The nucleus was thus represented as a tube with zero inner radius (a cylinder). It was oriented lengthwise along the z-axis. The material of the nucleus was set to "waterMaterial" which was defined in the user's class as the GEANT4 "NIST Water Equivalent"[10] with a density of 1.0 $\frac{gm}{cm^3}$.

3.2.1.2 The Chromatin Fibers

Chromatin fibers were defined in the geometry used in both applications; however, the chromatin volumes are only used independently of the nucleus volume in the CHROMATIN-INCLUSIVE APPLICATION. The implementation of the chromatin fibers is done based on an assumption of uniform distribution of genetic material throughout

[8] Based on the suggestion of Dr. Chris Wang (personal communication)

[9] The original dimensions were 7.5 micron height and 15 micron diameter which were arrived at independently but are similar to the dimensions used by Friedland *et al.*[24]

[10] A material definition provided by the GEANT4 toolkit.

the nucleus. They allow a simple method for the restriction of registered events to a defined sensitive region.

The simplest implemementation of chromatin fiber, numerous simple cylinders with 30nm diameter are placed in a uniform manner throughout the nucleus volume. The diameter corresponds to the experimentally observed diameter of chromatin fiber. The number of chromatin fiber volumes placed in the nucleus volume was calculated so that the volume of chromatin fiber volumes equated to 16% of the nucleus volume.[11]

A simple method making use of concentric shells of cylinders (chromatin fiber volumes) was chosen to approximate a uniform distribution. The Parameterisation feature included in the GEANT4 geometry system was used to implement repeating cylinders (chromatin fibers) throughout the nucleus volume. The *G4PVParameterised* class allows the user to defined volumes according to equations with position (and/or size) varying with each implementation.

The dimensions of each chromatin fiber volume are identical and are not modified by the parameterisation class. The transformation is calculated according to two variables: the number of shells and the nubmer of chromatin on the first shell.

$$nCro_i = nCro_1 * i \tag{1}$$

$$nCro_T = 1 + \sum_{i=1}^{nSh} nCro_1 * i \tag{2}$$

This method, described by equations 1 and 3.2.1.2, results in discrete possible values for total chromatin fibers. These two variables can be altered to best approximate the target number of total chromatin fibers (to achieve the 16% value). There is a single chromatin placed on the zeroeth shell.[12] The distance between each shell is constant and is calculated based on the number of shells and the radius of the parent,

[11] This method distinguishes itself as the registration of energy depositions is restricted to this pre-defined region.

[12] The zeroeth shell is the centerline of the nucleus.

nucleus, volume. The number of chromatin fibers on each shells increases, relative to the number of chromatin fibers on the first shell, proportionally to the radius of the first shell. The algorithm implemented in the ChromatinParameterisation class results in the population of shells moving outward from the center. Each individual shell is populated in a counter-clockwise motion (when observing in the direction of the negative z-axis) starting at 0* relative to the positive x-axis.

3.2.1.3 The World and Medium

In addition two other volumes were created in these applications. The world volume is a cylinder encompassing the nucleus and a small buffer. Just inside of this world volume is an identically shaped Medium volume. This setup was derived, in part, from the Microdosimetry example and done to accommodate the implementation of special physics settings by the user.

3.2.2 Description of Physics

The physics list used for both applications is identical. It was derived, in large part, from the microdosimetry example included with the GEANT4 toolkit. This physics lists activates a full suite of GEANT4-DNA specific physics included in the GEANT4 release used. (GEANT4.9.4p2). For the author's applications: the processes, the models and the energies which each model handles are shown in table **??** for each particle.[13]

Physics processes and models in GEANT4 are enabled by region. For these applications a single region was created. This regions, known in-program as a *G4Region*, was defined by the medium volume. In this *G4Region* GEANT4-DNA physics were enabled. It is inclusive of the nucleus and chromatin volumes. This implementation

[13]Reviews of the models used in GEANT4-DNA are referenced in the background section of this work

Table 1: GEANT4-DNA processes and models used in the author's applications. Each model can be examined in more detail by consulting the GEANT4 source code and refernces cited in the text.

Particles					
Process	e^-	p^+	H^0	α, He^+, He^0	C, N, O & Fe
Elastic Scattering	4 eV - 1 MeV Champion				
Excitation	9 eV - 1 MeV Born	10 eV - 500 keV Miller-Green 500 keV - 100 MeV Born	10 eV - 500 keV Miller-Green	1 keV - 400 MeV Miller-Green	
Ionization	11 eV - 1 MeV Born	100 eV - 500 keV Rudd 500 keV - 100 MeV Born	100 eV - 100 MeV Rudd	1 keV - 400 MeV Rudd	.5 MeV/u - 10e6 MeV/u Rudd
Charge Increase/Decrease		10 eV - 10 MeV Dingfelder	100 eV - 10 MeV Dingfelder	1 keV - 400 MeV Dingfelder	
Vibrational Excitation	2 - 100 eV Sanche				
Attachment	4 - 13 eV Melton				

of physics in separately defined regions instead of specific volumes allows some flexibility for the end-user. The region was defined as the medium to allow the author to expand and contract the region outside of the nucleus where DNA physics were active without manipulating the world volume. It may be desirable, for equilibrium purposes, to have a region of active DNA physics larger than the scoring region. In these applications the medium is only slightly larger than the nucleus volume and dose not currently make a consideration for equilibrium. Only regular GEANT4 physics were active in the world volume outside of the medium.

3.2.3 Description of Primary Generator

The implementation of the General Physical Source (GPS) allowed the user to bypass the cumbersome use of algorithms in the PrimaryGenerator class. The built-in accommodations for source location distributions, energy distributions, angular distributions, and other features made GPS attractive to the author. GPS is typically activated using the UIMessenger interface allowing simple, in application, manipulation of particle type and energy.

The REJECTION APPLICATION simulates the passage of protons, alpha particles, carbon ions, nitrogen ions, oxygen ions and iron ions of various energies as shown in table 2. What distinguishes this example is the stationary source. In the REJECTION APPLICATION the source fires from a static position in the center of the top of the nucleus cylinder. The result is a pencil beam directed in the negative z-direction through the height of the cylinder (lengthwise). The particles and energies simulated by the REJECTION APPLICATION are outlined in table 2.

The CHROMATIN-INCLUSIVE APPLICATION differs in how the source is implemented. The location of the source is a randomly chosen point on the nucleus surface that is re-sampled with each incident particle generated. It fires from the interior

Table 2: Particles and energies simulated by author using the REJECTION APPLICATION.

Particle Type	Energies Simulated MeV
Protons	0.06, 0.08, 0.10, 0.20, 0.40, 0.60, 0.80, 1.00, 2.00, 5.00, 7.00, 10.0, 20.0
Alphas	0.20, 0.60, 1.00, 2.00, 7.00, 10.0, 16.0, 20.0, 40.0, 70.0, 100
Carbons	7.00, 8.00, 10.0, 50.0, 100, 500, 1000, 5000
Nitrogens	8.00, 10.0, 40.0, 70.0, 100, 400, 700, 1000
Oxygens	40.0, 70.0, 100, 400, 1000
Irons	50.0 100, 200, 500, 1000, 3000, 5000, 11000, 25000, 50000, 100000, 500000, 1000000

Table 3: Particles and energies simulated by author using the CHROMATIN-INCLUSIVE APPLICATION.

Particle Type	Energies Simulated MeV
Protons	0.10, 0.20, 0.30, 0.60, 1.00, 2.00, 5.00, 10.0, 20.0
Alphas	0.30, 0.60, 1.00, 5.00, 10.0, 40.0, 100

surface of the nucleus volume with an angular cosine distribution.[14] Due to variance issues and computational requirements, it presently addresses only protons and alpha particles.[15] The particles and energies simulated by the CHROMATIN INCLUSIVE-APPLICATION are outlined in table 3.

For both applications the number of particles simulated was determined according to resultant variance and availability of computational resources. All simulations used mono-energetic sources. The moving source simulations required a larger number of simulated particles due to the introduced variance of the varying location and angular distributions.

[14] A lambertian distribution.

[15] There is no physical (GEANT4-related) reason precluding simulation of heavier particles.

3.2.4 Description of Tracking and Scoring

When running simulations using the applications there are three distinct phases that are important to tracking and scoring: runs, events and interactions. The run is initiated from the GUI and, in this case, uses mono-energetic primary particles. Each run proscribes a certain number of events. These events, in these applications, represent a single incident primary particle. Each particle and its secondaries can produce interactions.

The basic tracking of particles does not differ between the Rejection Application and the Chromatin-Inclusive Application. All particles are tracked throughout the volumes and these results are outputted to the ROOT file. The total energy deposition of all interaction in the nucleus is summed and used to calculate dose delivered to the nucleus volume by each primary particle and these results are outputted to the dose file.

The storage of interaction for processing is significantly different for the two applications. Whereas the Rejection Application registers all interactions (above 5 eV) in the nucleus volume (inclusive of chromatin fibers) the Chromatin-Inclusive Application only stores interaction (above 5 eV) which occur in the defined chromatin fibers. These data are outputted to the SB file.

Each run of of the author's application results in the output of three files. Variable numbers of primary particles are simulated during each run. Each primary particle is simulated (along with secondary particles) before the subsequent primary particle is initiated. The three types of files outputted on each run are:

- Root File: This file is outputted at the end of each run and consists of trees of data cataloging locations of interaction, process types, energy deposited, and other physical parameters.

- Dose File: This csv (comma-separated value) file outputted at the end of each

run. It provides the dose deposited by each event along with the dose of the total run. It also includes the number of potential strand breaks (interactions depositing greater than 5 eV).

- SB File:[16] This is a csv file outputted at the end of each event. This file contains the location of each interaction with greater than 5 eV energy deposition. The location (x,y,z) and energy are stored for each qualifying interaction.

The number of events simulated in each run is limited by the maximum file sizes of under two gigabytes. Simulation times are significant but are well coordinated to output file size. Simulation times were less than twelve minutes using Geant4 in a virtual environment running on a single core of an Intel Q6600 processor.

3.3 Processing in R

R was used for the processing and mining of the results ouputted from Geant4.[17] It performs the rejection sampling,[18] the energy ramp function and instantiates DBSCAN algorithm. It also includes various features for aggregating the results of multiple particle energies. It facilitates the processing of events individually and in combined groups. It outputs all data in a format easily viewed and subsequently processed in any common spreadsheet application.

3.3.1 Energy Ramp Function

A ramp function is used (in both applications) to determine if registered possible strand breaks (events with deposition >5 eV) are to be considered as strand breaks. This function is adapted from the work of Friedland, *et al.*[30] and of Francis, *et al.*.[21] The function scores interactions as damage with zero probability for interactions with

[16] SB File refers to its storage of locations of what are considered potential strand breaks.
[17] The author's R code is included in Appendix B.
[18] As documented the Russian-roulette rejection sampling techniques are only applied to the Rejection Application

energy deposition of 5 eV. The probability of an interaction being scored as a damage increases linearly from 5 eV until it reaches unity at 37.5 eV. Equation 3 is used along with a pseudo-random number generator to pass and reject potential damages.

$$mod = \frac{E - E_{low}}{E_{high} - E_{low}} \tag{3}$$

The reasoning for this ramp function is based on the understanding that excitations and ionizations can both cause local damage under a variety of circumstances. Ionizations with high energy deposition have a large probability of causing damage. Excitations have a lower, but still significant, probability of causing damage through the formation of reactive chemical species.

3.3.2 Sampling Rejection

This section applies only to the Rejection Applicatoin which lacks the more restriction definition of sensitive volume implemented in the Chromatin-Inclusive Application. The former application defines sensitive sites using a Russian Roulette technique. To simualte the 16% genetic material in the nucleus volume, it rejects 84

3.3.3 DBSCAN Algorithm

To perform cluster analysis the author chose the DBSCAN algorithm. The choice of the DBSCAN algorithm was based on author's desire for a density-based clustering algorithm. DBSCAN (density-based spatial clustering of applications with noise) is a commonly used clustering algorithm proposed by Ester, *et al.*[18].[19]

DBSCAN relies on clusters designate by reach-ability. The user identifies two parameters:

- *minPts*: the minimum number of points required to form a cluster.

[19]The specific implementation of the DBSCAN algorithm used by the author was obtained from the *'fpc'* (flexible procedures for clustering) library developed by Hennig[39] which is included in the repositories built into the current release of the R software.

- ϵ/*eps*: the maximum distance searched to find a neighboring point.

The DBSCAN algorithm first visits an unvisited point[20]. It then proceeds to search in the ϵ neighborhood for further points. If a sufficient number of points are found to form a cluster then these points are added and their ϵ neighborhoods are searched. This is done until a point in the cluster is reached which does not have minpts in is ϵ neighborhood. The DBSCAN algorithm, barring pre-processing optimization, operates with $O(n^2)$[21] complexity.[22]

The *minpts* value used in this work was two; it was derived from the inherent requirements for the formation of a theoretical double strand break. The second input parameter, ϵ, determined the distance to be searched for neighboring damages. The two applications documented in this work make use of separate values for ϵ.

The value of ϵ for the Rejection Application is 3.2nm and was adopted from the work of Francis, *et al.*[21]. This value is roughly the lineal distance occupied by 10 base pairs on a DNA strand. It has been experimentally established that this 10 base pair difference is correlated well to the formation of biologically consequential damage. Brenner and Ward[12] have established an experimental correlation between the yield of double-strand breaks and the presence of clusters of multiple ionizations 2-3 nm in siz.e

The Chromatin-Inclusive Application took a different approach to the value of ϵ. The reason for this separate approach was the poor fit of the results of this application when compared to those of the prior application. This action is justified by the empirical nature of the many of the values adopted for the Rejection Application led the author of this work to manipulate the ϵ value in an effort to match the results

[20]In this work a point corresponds to the location of a damage as determined by the applied functions from the energy depositions stored in the SB file ouputted by Geant4

[21]"Big O notation" is used to denote the limiting complexity of a given operation. In this case the time complexity of the operation scales with the square of the size of the analyzed dataset.

[22]This is important as there is great variation in performance between clustering algorithms and techniques for determining damage density

Table 4: Examples of probabilities of damage classifications for cetain cluster sizes.

# of Damages (n)	1	2	3	4	5	6	7
P(SSSB)	1.00	0.00	0.00	0.00	0.00	0.00	0.00
P(CSSB)	0.00	0.50	0.25	0.13	0.06	0.03	0.02
P(SDSB)	0.00	0.50	0.00	0.00	0.00	0.00	0.00
P(CDSB)	0.00	0.00	0.75	0.88	0.94	0.97	0.98

of the two application. It is also proposed by the author that the 3.2 nm search may be, *per se*, inappropriate as the value of ϵ. With the defined chromatin fibers the 3.2 nm value may result in a *de facto* overestimation of clusters. The reason for this overestimation is due to the three dimensional search when the DNA strand only extends in two directions and may be oriented in a non-linear fashion. The ϵ value for the Chromatin-Inclusive Application was ultimately set to .8 nm. This value, as is hown in the results, produces similar results to the Rejection Application.

3.3.4 Categorization of Damages

The type of damage caused is based entirely on the size of the clusters. Simple strand breaks are associated with singular energy depositions and thus these types of singular damage are classified as simple strand breaks. Multiply damaged sites are associated with clusters with two or more energy depositions. These sites can be classified as complex single strand breaks, simple double strand breaks or complex double strand breaks depending upon the cluster size (number of energy depositions). The probability of the occurence of each type of strand break at a for a certain multiply damaged site (cluster) is defined by equations 4 and 5.

$$P_{SSB} = 0.5^{n-1} \tag{4}$$

$$P_{DSB} = 1 - 0.5^{n-1} \tag{5}$$

The variable n is representative of the number of damages in a certain cluster.

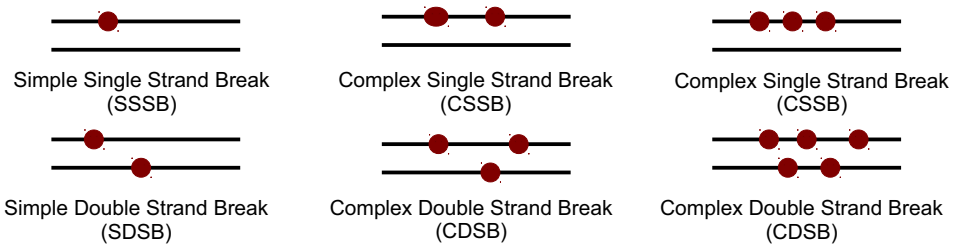


Figure 7: Examples of strand break classifications.

Singular, non-clustered, damages are classified as simple single strand breaks. Multiply damaged sites are classified by the number of damages in a certain cluster. Simple double stranded breaks are defined as sites where two lesions in a cluster are determined to have occurred on opposite strands. Complex strand breaks are determined to be all single and double strand breaks which do not fit the requirements of their corresponding "simple" designation. Probabilities for the classification of clusters is shown in table 4. Figure 7 shows six examples of the strand break classification.

3.3.5 Lineal Energy

Linear energy transfer (LET) is defined as the energy transferred to material by an ionizing radiation.[6] It is desirable to coordinate the results of this work to LET. However, the restrictive definition of LET[47] has led the author to adopt a similar, but easier to derive, measure; lineal energy,[68] y, is defined as energy imparted to a medium by a radiation divided by the mean chord length of the volume cross by the incident particle.[41][42]

$$y = \frac{\epsilon_s}{\bar{l}} \tag{6}$$

Equation 6 is used for the calculation of lineal energy. The chord lengths used for each application are defined using two different methods according to the positioning of the source. The Rejection Application assumes a chord length of exactly one unit height. The Chromatin-Inclusive Application uses the work of Langworthy

[55] to derive the mean chord length of a right cylinder. The ultimate derivation and resultant mean chord length for the geometry used is shown in equation 7.

$$\bar{s} = \frac{2hd}{(d + 2h)} \tag{7}$$

These assumptions result in chord lengths ($\bar{l}$) for the Rejection Application and Chromatin-Inclusive Application of 1.0 micron and 1.2 micron respectively.

Chapter IV

RESULTS AND DISCUSSION

4.1 Visualization of Geometry and Track Structure

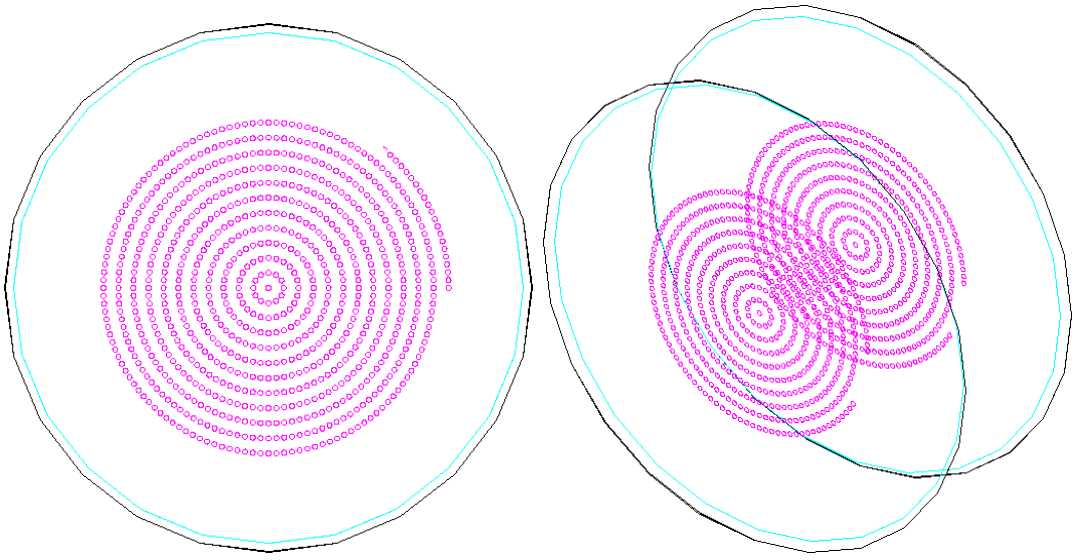
The OpenGL (OGL) visualization driver included in the standard GEANT4 toolkit was used to visualize the applications to verify the accuracy of the geometry and the primary particle generator (location distribution and angular distribution). Figure 8: (a) and (b) shows the geometry during the drawing period. The action of the methods used for parametrization of the chromatin volumes can be seen as the copies are drawn in a counter-clockwise direction filling each shell from the center to the boundary resulting in the complete geometry shown in figure 8: (c) and (d).

The OGL module is also capable of drawing particle track structures. Figures 9, 10 and 11 show track structures for three different type of incident radiation particles. Each of these particles, as is shown in this work, has similar lineal energy deposition through the volume. Together these figures of particle track structure provides a basic illustration of ionization density for each of these particles.[1]

4.2 A Comparison of Applications

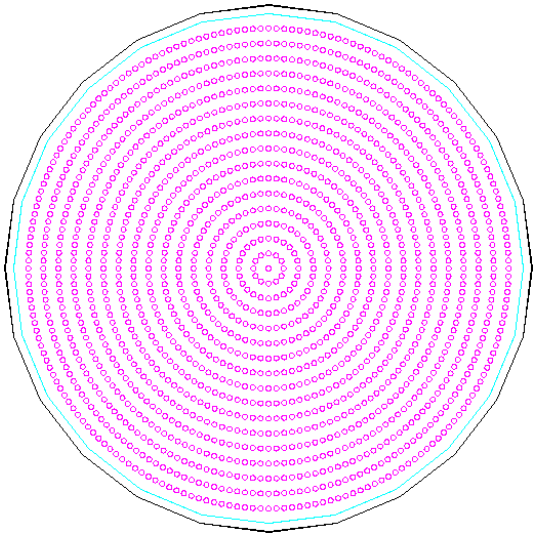
With the necessity of a pre-defined sensitive region in question a comparison of the Rejection Application and the Chromatin-Inclusive Application is presented. The differences between these two applications are described in detail in the Methods chapter. It is important to note that, in addition to the change in definition of sensitive sites, the Chromatin-Inclusive Application presents a smaller

[1]While a difference can be seen between track-structures the small sample size and different physical model coverage for each particle (all electrons are shown, not just those of ionizations) results in some obfuscation of the expected apparent "connective-ness" between interactions of each primary particle type.

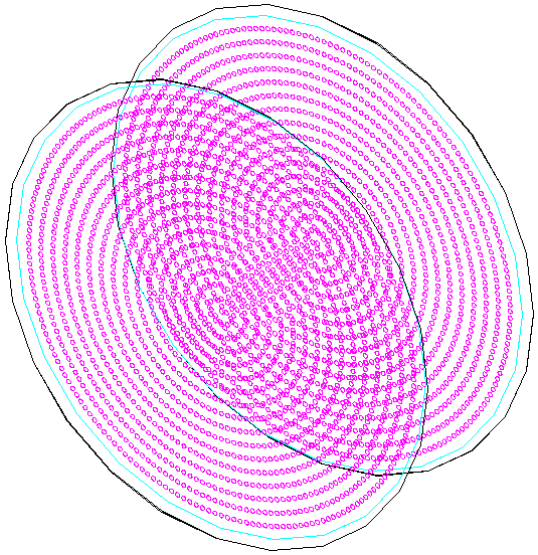


(a) Mid Draw (0*)

(b) Mid Draw (45*)



(c) Full Draw (0*)



(d) Full Draw (45*)

Figure 8: The application geometry during, (a) and (b), and after completion, (c) and (d), of the drawing period as rendered in the OpenGL Event Display.

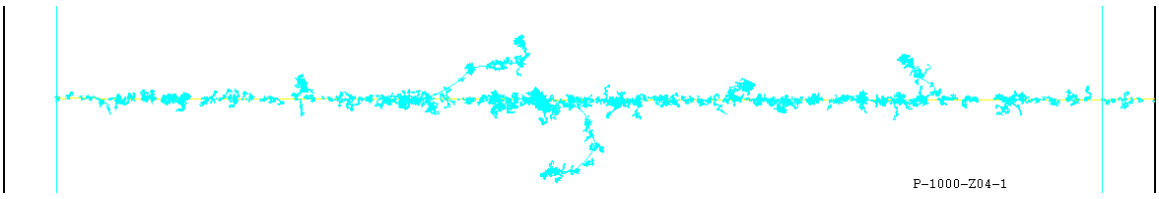


Figure 9: Visualization of single 1 MeV proton track structure. Ions are illustrated in blue; electrons are illustrated in red.

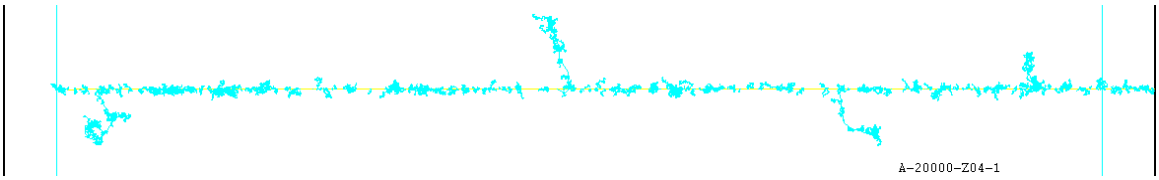


Figure 10: Visualization of single 20 MeV alpha particle track structure. Ions are illustrated in blue; electrons are illustrated in red.

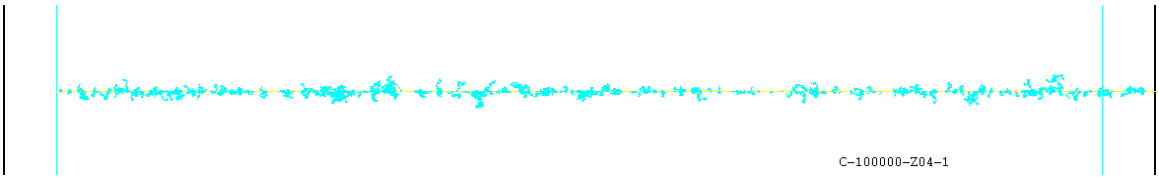


Figure 11: Visualization of single 1000 MeV carbon ion track structure. Ions are illustrated in blue; electrons are illustrated in red.

ϵ value[2] in comparison to the REJECTION APPLICATION for processing with the DBSCAN cluster analysis algorithm. The derivation of this smaller value is empirical based on the matching of results from the two applications.[3]

4.2.1 Lineal Energy

The lineal lnergy has been derived for both applications.[4,5] The lineal energy derived from each application, for protons and alpha particles, is compared in figure to each particle's total stopping power and $\frac{(ParticleEnergy)}{(ProjectedRange)}$ adopted from NIST's PSTAR and ASTAR databases.[59] The apparent validity of these values has led to their use throughout the Results chapter to coordinate the results of the cluster analysis technqiques and data processing.[6]

[2] ϵ is defined and its value described in the Methods chapter.

[3] Justification in slightly greater detail is provided in the Methods chapter.

[4] The adoption of Lineal Energy as a stand-in for LET is described in the Methods chapter.

[5] The slight deviation of the method for calculating lineal energy in this work from the true method proscribed by the ICRU is described in the Methods chapter.

[6] Here it stands in for LET in an effort to validate one of the objectives of this work: to analyze the relationship between LET, particle type and damage complexity.

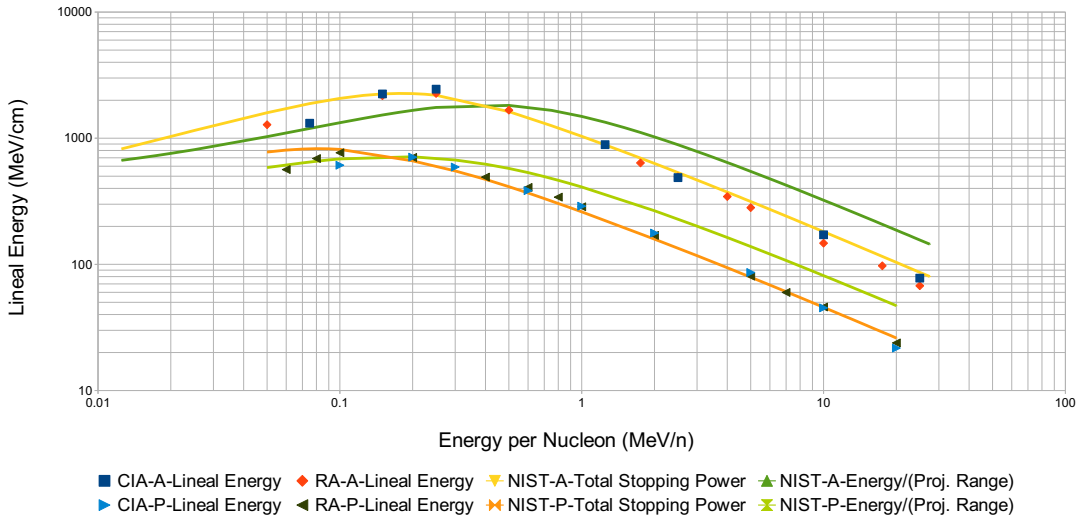


Figure 12: The derived lineal energy for protons and alpha particles for the Rejection Application (prefix *RA*) and the Chromatin-Inclusive Application (prefix *CIA*).

4.2.2 Strand Break Production

The production of damages is proportional to dose and this proportionality holds roughly constant all lineal energies and similar between both protons and alpha particles. Table 5 shows the number of strand breaks registered by both applications.[7,8] The small deviations, shown in the figures, appear to be due to the influence of noise noise. The application of the energy ramp function might account for a small difference between the two applications due to different energy distributions of potential strand breaks (damages).[9]

Lineal energy is thus established as a relative non-factor in the production of damages at sensitive sites. Cluster analysis and further classification of singly and

[7]this value is the result of applying the rejection algorithm (if applicable) and the energy ramp function to the number of potential damages stored by Geant4

[8]These strand breaks are also referred to as "damages." They are energy depositions greater than 5 eV that have been passed by the rejection algorithm (if applicable) and energy ramp function for cluster analysis by the DBSCAN algorithm

[9]As detailed in the Methods chapter these strand breaks are simply a record of all interaction recorded by Geant4 for each application which deposit energy greater than 5 eV at sensitive sites.

Table 5: The average number of strand breaks for all energies run of protons and alphas for the Rejection Application and the Chromatin-Inclusive Application

Application	Particle Type	Average	σ	$\frac{\sigma}{\bar{\mu}}$
Rejection Application	Proton	95.72	5.54	5.78%
	Alpha Particle	96.88	3.37	3.48%
Chromatin-Inclusive Application	Proton	94.87	5.90	6.22%
	Alpha Particle	101.87	6.27	6.16%

multiply damage sites[10] allows the qualification of damages by complexity. Figure 13 and 14 show the strand break production for alpha particles and protons using the Rejection Application and the Chromatin-Inclusive Application normalized per unit dose to the nucleus volume. The increased complexity of damages for higher lineal energy particles can be readily seen. The most severe damage according to the classification scheme, complex double stand breaks, exhibit the greater variation with lineal energy.

4.2.3 Damage Complexity

A commonly used parameter to qualifying the complexity of damage produced by incident radiation particles is the ratio of double strand breaks to single strand breaks. Figure 15 shows this ratio for the Rejection Application and the Chromatin-Inclusive Application for both protons and alpha particles. The influence of lineal energy upon this measure of complexity is immediately obvious becoming especially pronounced at higher lineal energies.

4.3 REJECTION APPLICATION in Detail

The superior convergence of the Rejection Application application, due to its scoring simplicity, allowed a significantly larger amount of data to be collected from

[10] According to the classification scheme detailed in the Methods chapter of this work

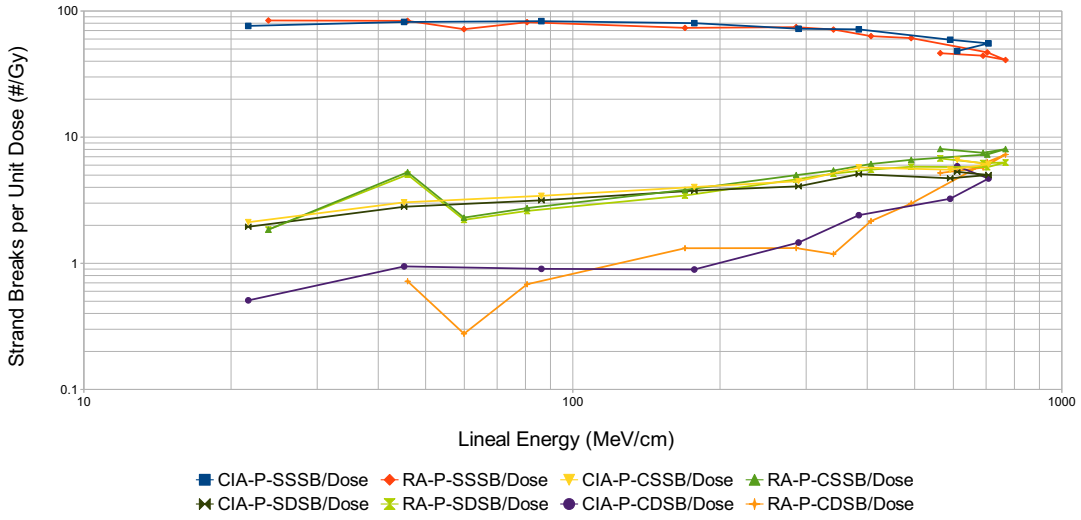


Figure 13: For protons of various energies a comparison of the number of six types of strand breaks: simple single strand breaks, complex single strand breaks, total single strand breaks, simple double strand breaks, complex double strand breaks and total double strand breaks. Values derived from the Rejection Application is denoted by the prefix *RA* and values derived from the Chromatin-Inclusive Application are denoted by the prefix *CIA*.

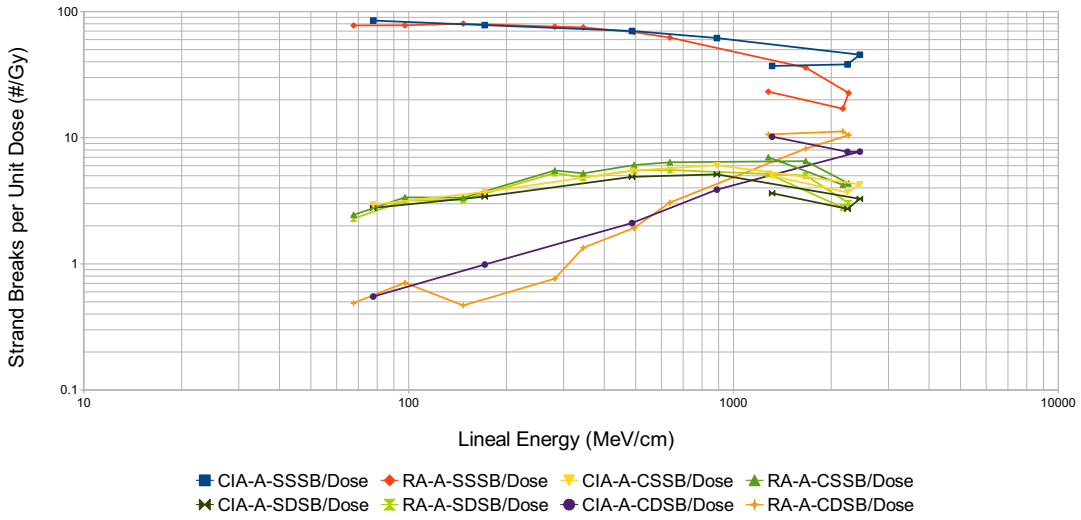


Figure 14: For alpha particles of various energies a comparison of the number of six types of strand breaks: simple single strand breaks, complex single strand breaks, total single strand breaks, simple double strand breaks, complex double strand breaks and total double strand breaks. Values derived from the Rejection Application is denoted by the prefix *RA* and values derived from the Chromatin-Inclusive Application are denoted by the prefix *CIA*.

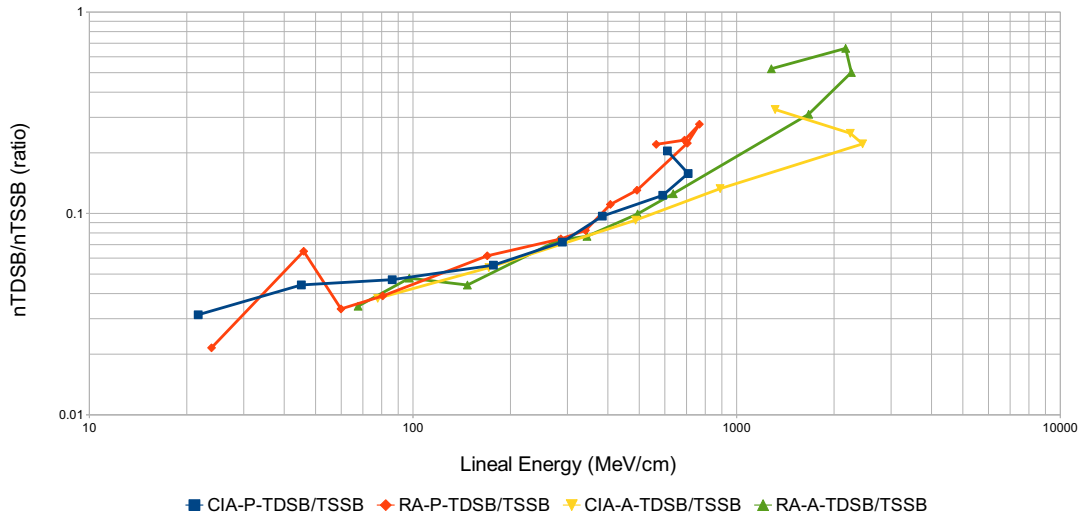


Figure 15: Comparison of the ratio of total double strand breaks (simple+complex) to total single strand breaks (simple+complex) for models the Rejection Application (prefix *RA*) and the Chromatin-Inclusive Application (prefix *CIA*) for protons and alpha particles. These values are coordinated to lineal energy.

it. The data was processed with a variety of techniques in an attempt to quantify and graphically illustrate the characteristics of different incident radiations.

4.3.1 Lineal Energy

The verification of lineal energy derived for the Rejection Application against total stopping power shown in figure 12 introduces the presentation of figure which shows the lineal energy for the Rejection Application for protons, alpha particle and an assortment of heavier ions.

4.3.2 Damage Complexity

Figure 17 shows the ratio of the number of clusters classified as DSBs to the number of clusters and singular damages classified as SSBs for particles of various types at various energies,[11]. The data can be compared to the results, shown in figure ref

[11]In figure 17 energy is shown scaled per nucleon not only to scale the data to be easily displayed on a single graph but also because the implementation of physics models scales, in part, with particle mass.

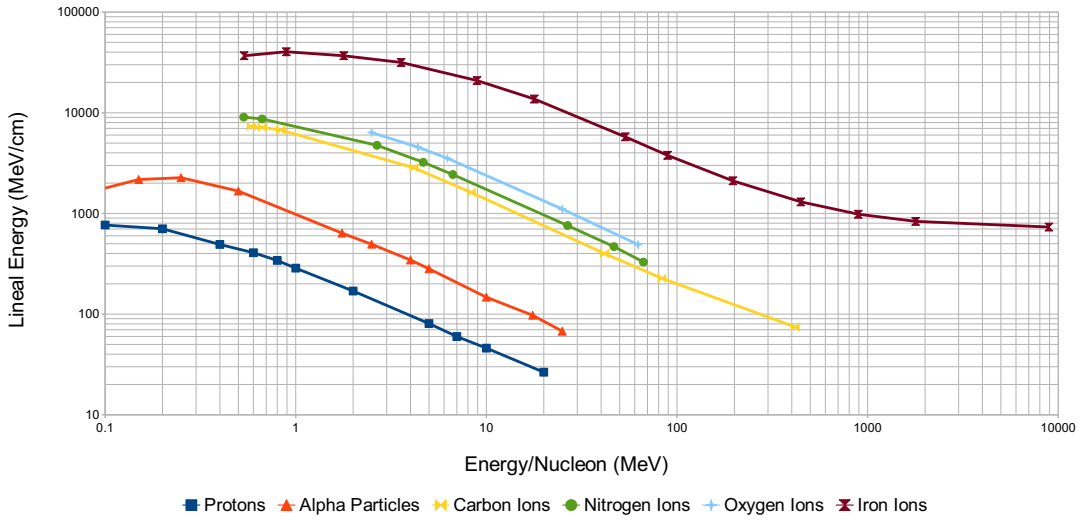


Figure 16: The user-derived average lineal energy for the protons, alpha particle, carbon ions, nitrogen ions, oxygen ions and iron ions as calculated using the REJECTION APPLICATION. These values is coordinated to Energy/Nucleon.

obtained by Francis, *et al.*[21] who used a substantially similar method though limited the scope of their analysis to protons.

The differences in damage complexity and their relationship to lineal energy for each particle are illustrated in figure 18. It is important to note that, due to the limits of Born theory, GEANT4 does not properly address heavier ions with energies below 0.5-1.0 MeV/u. Figure 19 shows the full range of particles energies simulated (coordinated by lineal energy).[12]

The most important damage produced by radiation particles according to the classification scheme adopted by the author is the complex double strand break. These damages represent those damages which, biologically, are the most difficult for the cell to repair effectively. Figure shows the production of CDSBs coordinated to lineal energy for protons, alpha particles, carbon ions, nitrogen ions, oxygen ions and iron ions.

[12]Figure 19 is inclusive of the previously inaccurately addressed lower energies. The effects of the inaccuracy can be seen for heavier ions with higher lineal energies.

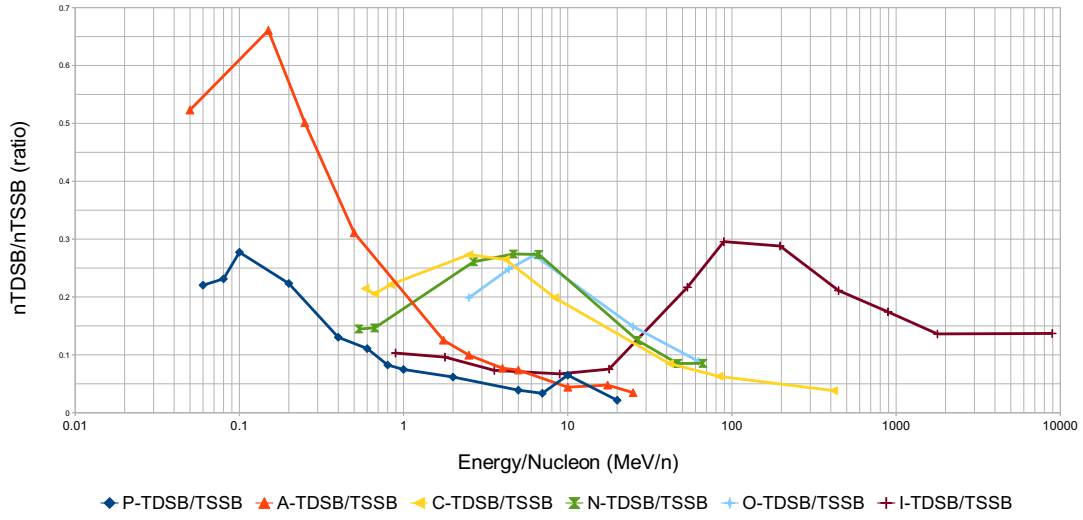


Figure 17: The ratio of the number of calculated double strand breaks to the number of calculated single strand breaks coordinated to energy per nucleon.

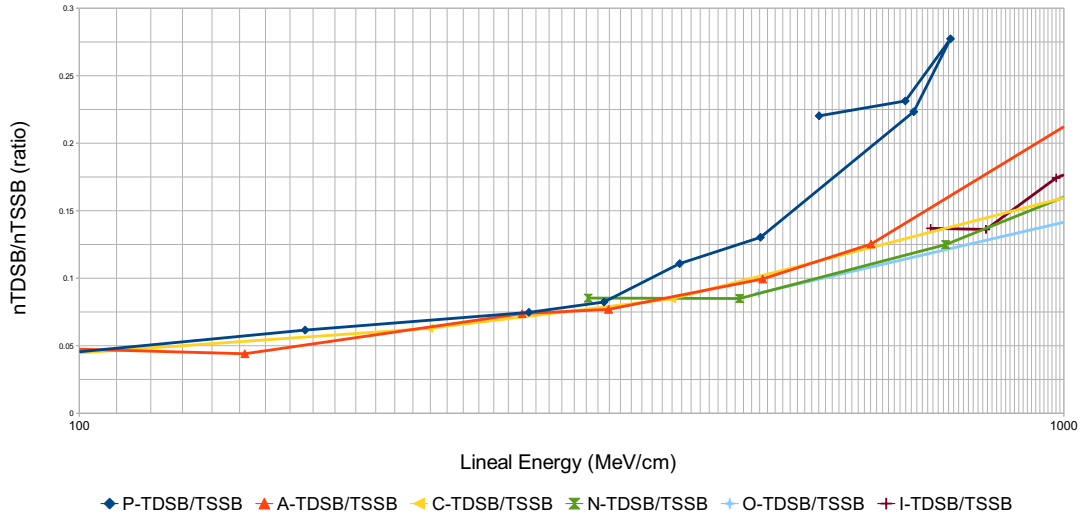


Figure 18: The ratio of the number of calculated double strand breaks to the number of calculated single strand breaks coordinated to derived quantity lineal energy.

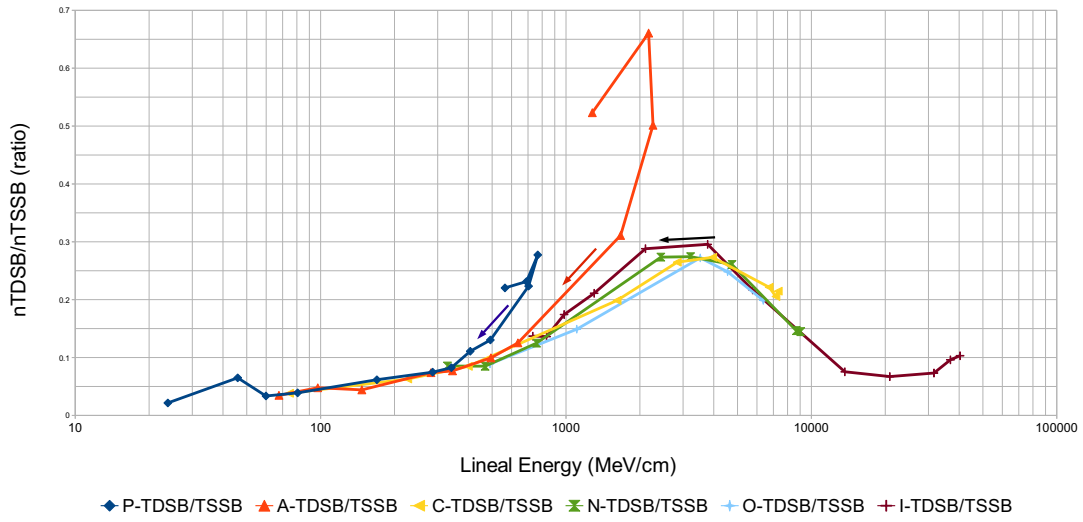


Figure 19: The ratio of the number of calculated double strand breaks to the number of calculated single strand breaks coordinated to derived quantity lineal energy. Points are connected in accordance with particle energy. The progression of particle energy is shown by arrows.

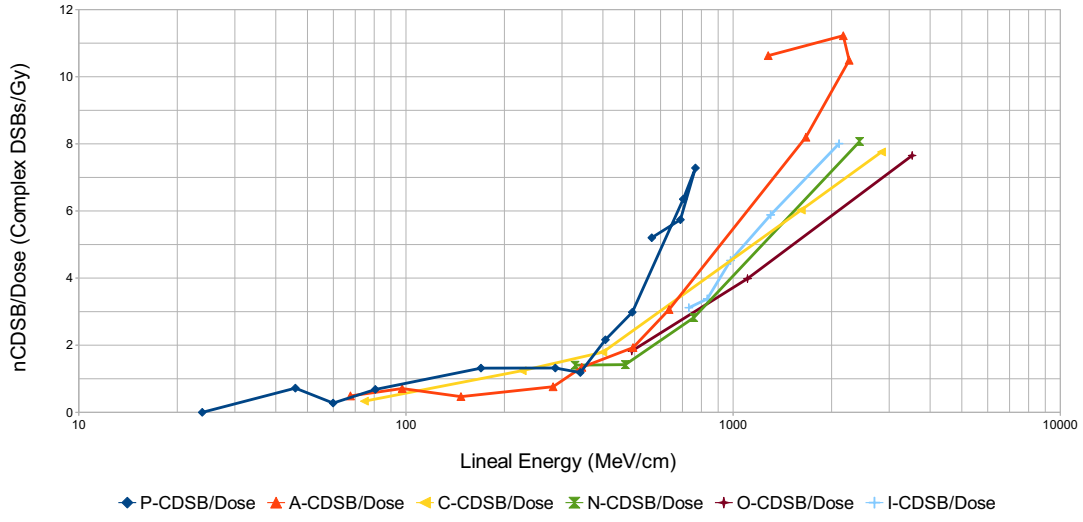
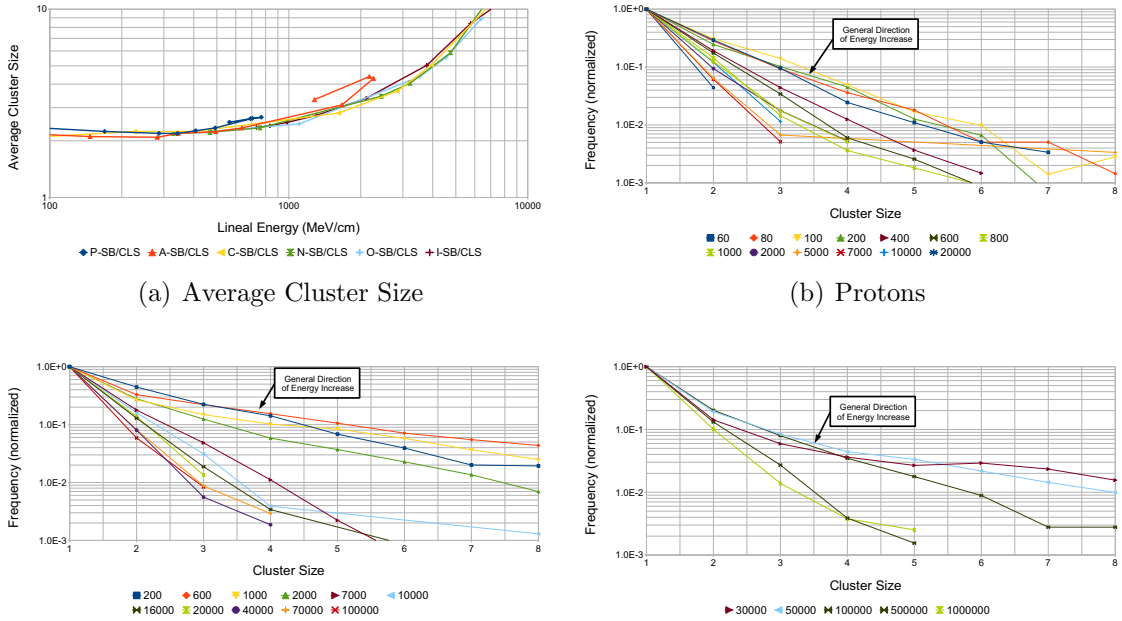


Figure 20: The number of complex double strand breaks coordinated to lineal energy and normalized per unit dose. Higher lineal energy data points for heavier ions (lower energies) are not shown due to inaccuracies noted in text.



(a) Average Cluster Size

(b) Protons

(c) Alpha Particles

(d) Carbon Ions

Figure 21: The average cluster size (a) for particle simulated using the REJECTION APPLICATION. The frequency of occurrence of clusters (of certain numerical size) is detailed for protons (b), alpha particles (c) and carbon ions (d).

The complexity of damage can also be characterized in greater detail by looking at the specific size of clusters produced by different radiations. The average cluster size, as shown in figure 21, is relatively indifferent to particle type at a certain lineal energy (with deviation near peak lineal energy). The distributions of cluster sizes for simulated proton, alpha particle and carbon ion energies simulated using the REJECTION APPLICATION are also shown in figure 21. It can be seen that, in general, as energy increases the overall complexity of damage decreases for both particle. At each particle energy the frequency of clusters of various sizes is shown relative to the number of simple single strand breaks (singular damages).

Figure 22 shows a more detailed comparison of particle type for radiations of a similar lineal energy.[13] The frequency of occurrence of clusters with greater than

[13]Because particles were initiated at certain energies with lineal energy a derived quantity, these particle have, only, approximately the same lineal energy.

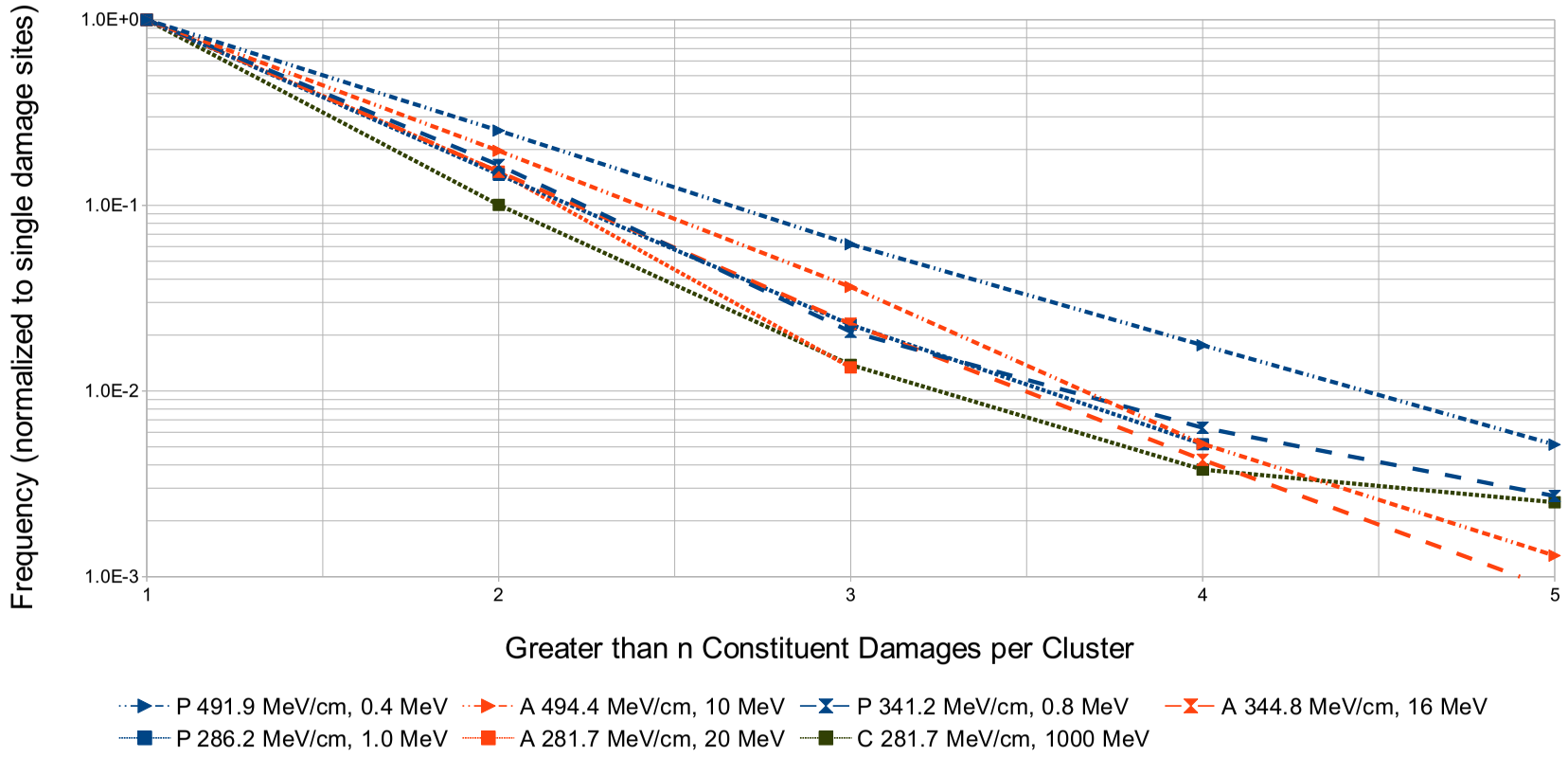


Figure 22: The frequency of occurence of clusters with greater than a certain number of damages relative to the occurence of simple single strand breaks (singular damages) for protons, alpha particles, and carbon ions with similar lineal energy.

a certain number of damages is shown in figure 22.[14] The display of frequency of occurrence as "greater than n damages" instead of "n damages" better displays the difference in complexity between each type of particle which is the central purpose of figure 22.

[14]This value is different than the frequency of occurrence shown in (a), (b) and (c) of figure 21

Chapter V

CONCLUSIONS

This thesis establishes a methodology for using the Monte Carlo track structure code Geant4 to computationally quantify damage done by simulated incident radiation particles to cell nuclear material. The results presented in this thesis shows, using cluster analysis, that the complexity of damage varies not only with lineal energy but also with particle type. It can also be concluded, from the work shown, that a pre-defined sensitive site, in the form of a basic chromatin fiber, does not provide significant benefit to results.

The Rejection Application is based on the techniques adopted by Francis, *et al.*[21] and makes use of a rejection algorithm that designate 16% of energy depositions greater than 5eV as sensitive sites. Its results compare favorably to those obtained by Francis, *et al.*. The Chromatin-Inclusive Application is an evolution of the Rejection Application that designates sensitive sites as occurring in a predefined region. Modifications had to be made to the data processing techniques[1] to match the data produced by the two applications. No significant advantage can be distinguished for the Chromatin-Inclusive Application. The statistical nature of the cluster analysis technique for analyzing damage complexity caused by various radiations is judged, by the author, to be effectively handled by the simpler Rejection Application.

The data shows a definite increase in damage complexity, as quantified by the metrics introduced by the author, for protons versus alphas and heavier ions at any given lineal energy. This difference is greater near the peak lineal energy of a given

[1]The modification to the input variables of the DBSCAN algorithm are detailed in the Methods chapter

particle type. The greater production of complex double strand breaks is especially important given the cell's decreased effectiveness in repairing these types of damages. A smaller sample of data also suggests that alpha particles have a superiority to carbon ions of similar lineal energy. This trend appears to carry for relationships between ions of different mass.[2] The similarity between particle lineal energy and linear energy transfer (LET) leads the author to propose that these inferences hold true when the coordinating measure used is LET instead of the less restrictive lineal energy.

[2]Suggesting some correllation to mass which is, in turn, coordinated with effective charge. The effect of these paramters together or indepedently is not directly investigated by this work

Chapter VI

FUTURE DEVELOPMENTS

6.1 Integration of Advanced Geometry and Systems Biology

The ultimate goal of successful prediction of RBE using track structure analysis is currently far away. One of the relatively simple, yet enormously complex, steps towards this goal is the implementation of a realistic geometry. Various past simulation experiments have implemented geometries of varying complexity. The simplest of these are no more complex than the geometry implemented by the author in the applications examined in this thesis while other, more complex solutions to modeling the geometry of the nucleus (and ultimately the cell) are offered, in brief, below.

The author implemented a non-dynamic static version of chromatin. This implementation was based on a uniform and random distribution of chromatin. A more complex method based is the use of a polymer chain model. The polymer chain method is based on the knowledge that, during G_0/G_1 interphase, DNA condenses into folded, loping structures of several megabase pair (Mbp) size.[80]

One method for the implementation of a polymer chain model is the use of a random-walk algorithm for the creation of the backbone which forms the higher-order structure of chromatin.[69][53] The random walk method assigns segments of geometry with randomly sampled twists, turns and/or bends. This method can make use of Brownian dynamics and Monte Carlo simulation.[57] Experimental data regarding various parameters such as bending rigidity, torsional rigidity, stretching rigidity and intrachain interaction can be used to bias the progress of the random walk and allow the final, combined structure to resemble that of chromatin *in situ*.[54]

The use of random walk generated geometry and Monte Carlo particle simulation to predict DNA breakage has been studied by Ponomarev, *et al.*. Of specific interest to the author is work examining fragment size distribution and dose response[64] as well as the connection of higher-order DNA structure to the relation between DNA breakage and incident radiation LET.[65]

Advances in structural biology have lead to increasingly complex and detailed three dimensional chromatin models.[50] The integration of increasingly complex geometries with Monte Carlo particle track structure simulation and analysis is an ongoing area of research.[28][23] Currently research has been published using the PARTRAC Monte Carlo track structure code to begin form a ful, detailed scale simulation of "systems radiation biology."[30] This work has progressed to include the investigation of the spatial and/or temporal aspects of ion track interaction[51], diffusion of reactive chemical species[52], strand break repair[25][26], and bystander effects[29]. The ultimate goal is a four-dimensional simulation that will, ideally, model, in detail, all spatial and temporal aspects and results of particle interaction with cell material.

6.2 GEANT4

The GEANT4 project is ongoing. New developments from the GEANT4-DNA and Electromagnetic Physics working groups are included with each release. Currently the development of GEANT4-DNA includes work towards a multi-scale approach to physics (combinations of condense and explicitly detailed treatments), implementation of radiation chemistry, and the provision of advanced cellular geometries.[43]

6.2.1 Radiation Chemistry

The current release ofGEANT4-DNA (4.9.5) is the first to include capability for modeling radiation chemistry. The processes and models for simulating radiatiation chemistry are detailed by Karamitros, *et al.*[46] and are derived from the Monte Carlo code, PARTRAC devleoped by Friedland, *et al.*.[52][23] The present capabilities of

the GEANT4 software include the ability to simulate the chemical stage in the temporal aspect up to 1 microsecond after irradiation. Currently the implementation of radiation chemistry has been integrated into the working release of GEANT4 but has not been fully documented or demonstrated by its developers.[1]

6.2.2 Cloud Computing

Much has been made of the advancements in Monte Carlo simulation made possible by the increased availibility of cheap computational power. The next step in this evolutionary process is the application of commercial cloud computing infrastrucure. The on-demand availibility and the unique cost-structure of commercial cloud computing makes vast computational resources available to a variety of researchers when before it was limited to a handful of elite research universities, institutes and national laboratories with much greater funding resources. The author did not make use of cloud computing for this work but, for the production of final results, the utility of the commercial cloud computing infrastructure seems undeniable.[76] Furthermore, the implementation of GEANT4 in this environment has been made easier by the provision of easy installation methods for certain cloud computing services by GEANT4's supportive user base.[66]

[1] The most common and useful way of demonstration is through inclusion of turorials and examples in the working release of GEANT4.

Appendix A

GEANT4 USER APPLICATION CODE

This application owes much to the examples provided by the GEANT4 Collaboration. The author would like to express his sincere gratitude to the GEANT4 Collaboration and especially to the members of the GEANT4-DNA working group without whom this work would not have been possible.

A.1 REJECTION APPLICATION

This work is, at its base, an evolution of the classes provided in the examples included with the GEANT4 toolkit. Its basic structure is adapted directly (and openly) from the microdosimetry example. Some user classes are unaltered from there existance in that example (such as G4ElectronCapture). Others have been altered significantly to fit the needs of the author. User classes not instantiated and GEANT4 capabilities not used in the examples have also be introduced based upon the needs to the author.

A.1.0.1 GEANT4 License

```
1  // ********************************************************************
2  // * License and Disclaimer                                           *
3  // *                                                                  *
4  // * The  Geant4 software  is  copyright of the Copyright Holders  of *
5  // * the Geant4 Collaboration.  It is provided  under  the terms  and *
6  // * conditions of the Geant4 Software License,  included in the file *
7  // * LICENSE and available at  http://cern.ch/geant4/license .  These *
8  // * include a list of copyright holders.                             *
9  // *                                                                  *
10 // * Neither the authors of this software system, nor their employing *
11 // * institutes,nor the agencies providing financial support for this *
12 // * work  make  any representation or  warranty, express or implied, *
13 // * regarding  this  software system or assume any liability for its *
```

```
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ******************************************************************
```

A.1.1 Base Class and Make File

A.1.1.1 GNUmakefile

```
#----------------------------------------------------------------
# $Id: GNUmakefile,v 1.2 2010-01-11 16:13:32 gcosmo Exp $
#----------------------------------------------------------------
# GNUmakefile for examples module.  Gabriele Cosmo, 06/04/98.
#----------------------------------------------------------------

name := Vad3-4

G4TARGET := $(name)
G4EXLIB := true

ifndef G4INSTALL
  G4INSTALL = ../..
endif

.PHONY: all
all: lib bin

include $(G4INSTALL)/config/binmake.gmk
```

A.1.1.2 Vad3-4.cc

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"

#ifdef G4VIS_USE
  #include "G4VisExecutive.hh"
#endif

#ifdef G4UI_USE
  #include "G4UIExecutive.hh"
```

```
10 #endif
11
12 #include "Randomize.hh"          // header for randomization. needed for initiation of
       RNG
13
14 #include "DetectorConstruction.hh"
15 #include "PhysicsList.hh"
16 #include "PrimaryGeneratorAction.hh"
17 #include "RunAction.hh"
18 #include "SteppingAction.hh"
19 #include "SteppingVerbose.hh"
20 #include "HistoManager.hh"
21 #include "EventAction.hh"
22
23 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
24
25 int main(int argc,char** argv)
26 {
27   // This line sets the RNG engine. There are various engines available.
28   CLHEP::HepRandom::setTheEngine(new CLHEP::RanecuEngine);
29
30   // Construct the default run manager
31   G4RunManager * runManager = new G4RunManager;
32
33   // Set mandatory user initialization classes
34   DetectorConstruction* detector = new DetectorConstruction;
35   runManager->SetUserInitialization(detector);
36   runManager->SetUserInitialization(new PhysicsList);
37
38   // Set mandatory user action classes
39   runManager->SetUserAction(new PrimaryGeneratorAction(detector));
40   PrimaryGeneratorAction* primary = new PrimaryGeneratorAction(detector);
41
42   HistoManager*  histo = new HistoManager();
43
44   // Set optional user action classes
45   RunAction* RunAct = new RunAction(detector,histo);
46   runManager->SetUserAction(RunAct);
47
48   // Need to set up EventAction class
49   runManager->SetUserAction(new EventAction(histo,RunAct));
```

```
50
51    runManager->SetUserAction(new SteppingAction(RunAct,detector,primary,histo));
52
53 #ifdef G4VIS_USE
54    G4VisManager* visManager = new G4VisExecutive;
55    visManager->Initialize();
56 #endif
57
58    // Initialize G4 kernel
59    runManager->Initialize();
60
61    remove ("Vad3-4.root");
62
63    // Get the pointer to the User Interface manager
64    G4UImanager* UImanager = G4UImanager::GetUIpointer();
65
66    if (argc==1)   // Define UI session for interactive mode.
67    {
68      #ifdef G4UI_USE
69        G4UIExecutive* ui = new G4UIExecutive(argc, argv);
70        UImanager->ApplyCommand("/control/execute microdosimetry.mac");
71        ui->SessionStart();
72        delete ui;
73      #endif
74    }
75    else           // Batch mode
76    {
77      G4String command = "/control/execute ";
78      G4String fileName = argv[1];
79      UImanager->ApplyCommand(command+fileName);
80    }
81
82    #ifdef G4VIS_USE
83      delete visManager;
84    #endif
85
86    delete runManager;
87
88    return 0;
89    }
```

A.1.2 include

A.1.2.1 DetectorConstruction.hh

```
1  #ifndef DetectorConstruction_h
2  #define DetectorConstruction_h 1
3
4  #include "G4VUserDetectorConstruction.hh"
5  #include "G4VPhysicalVolume.hh"
6  #include "G4LogicalVolume.hh"
7  #include "G4Box.hh"
8  #include "G4Tubs.hh"
9  #include "G4Sphere.hh"
10 #include "G4Material.hh"
11 #include "G4NistManager.hh"
12 #include "G4PVPlacement.hh"
13 #include "G4UserLimits.hh"
14 #include "G4VisAttributes.hh"
15
16 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
17
18 class G4Region;
19
20 class DetectorConstruction : public G4VUserDetectorConstruction
21 {
22 public:
23
24   DetectorConstruction();
25
26   ~DetectorConstruction();
27
28   G4VPhysicalVolume* Construct();
29
30   G4double GetWorldDiameter()    {return WorldDiameter;};     // function for getting
         WorldDiameter
31   G4double GetNucleusHeight()    {return NucleusHeight;};     // function for getting
         NucleusHeight
32   G4double GetNucleusDiameter() {return NucleusDiameter;};     // function for getting
          NucleusDiameter
33   G4double GetNucleusMass()    {return NucleusMass;};       // function for getting
         NucleusMass
```

```
34    G4double GetChromatinMass()    {return ChromatinMass;};     // function for getting
          ChromatinMass
35
36  private:
37
38    G4double          WorldDiameter;
39    G4double          WorldHeight;
40    G4double          MediumDiameter;
41    G4double          MediumHeight;
42    G4double          NucleusDiameter;
43    G4double          NucleusHeight;
44
45    G4float           NucleusMass;
46    G4float           ChromatinMass;
47
48    G4VPhysicalVolume*      physiWorld;
49    G4LogicalVolume*      logicWorld;
50    G4Tubs*           solidWorld;
51
52    G4Material*         waterMaterial;
53    G4Region*         mediumRegion;
54    G4Region*         nucleusRegion;
55
56    void DefineMaterials();
57
58    G4VPhysicalVolume* ConstructDetector();
59  };
60  #endif
```

A.1.2.2 ChromatinParameterisation.hh

```
1  #ifndef ChromatinParameterisation_H
2  #define ChromatinParameterisation_H 1
3
4  #include "globals.hh"
5  #include "G4VPVParameterisation.hh"
6
7  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
8
9  class G4VPhysicalVolume;
10 class G4Tubs;
```

```
11
12 // Dummy declarations to get rid of warnings ...
13 class G4Trd;
14 class G4Trap;
15 class G4Cons;
16 class G4Orb;
17 class G4Sphere;
18 class G4Torus;
19 class G4Para;
20 class G4Hype;
21 class G4Box;
22 class G4Polycone;
23 class G4Polyhedra;
24
25 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
26
27 class ChromatinParameterisation : public G4VPVParameterisation
28 {
29   public:
30
31     ChromatinParameterisation(G4int NbOfShells,
32                               G4int NbFirstShell,
33                               G4double NucleusHeight,
34                               G4double NucleusDiameter,
35                               G4double ChromatinDiameter);
36
37   virtual
38     ~ChromatinParameterisation();
39
40     void ComputeTransformation(const G4int CopyNo, G4VPhysicalVolume* physVol) const;
41
42     void ComputeDimensions(G4Tubs& Chromatin, const G4int copyNo, const
           G4VPhysicalVolume* physVol) const;
43
44
45
46   private:
47
48     void ComputeDimensions (G4Trd&,const G4int,const G4VPhysicalVolume*) const {}
49     void ComputeDimensions (G4Trap&,const G4int,const G4VPhysicalVolume*) const {}
50     void ComputeDimensions (G4Cons&,const G4int,const G4VPhysicalVolume*) const {}
```

```
51      void ComputeDimensions (G4Sphere&,const G4int,const G4VPhysicalVolume*) const {}
52      void ComputeDimensions (G4Orb&,const G4int,const G4VPhysicalVolume*) const {}
53      void ComputeDimensions (G4Torus&,const G4int,const G4VPhysicalVolume*) const {}
54      void ComputeDimensions (G4Para&,const G4int,const G4VPhysicalVolume*) const {}
55      void ComputeDimensions (G4Hype&,const G4int,const G4VPhysicalVolume*) const {}
56      void ComputeDimensions (G4Box&,const G4int,const G4VPhysicalVolume*) const {}
57      void ComputeDimensions (G4Polycone&,const G4int,const G4VPhysicalVolume*) const
            {}
58      void ComputeDimensions (G4Polyhedra&,const G4int,const G4VPhysicalVolume*) const
            {}
59
60
61      // intializing variables used
62
63      G4int fNb;
64      G4int fFS;
65      G4double fND;
66      G4double fNH;
67      G4double fCD;
68      G4double gap;
69
70      G4double* NbEach;
71      G4double* RaEach;
72      G4double* AngEach;
73
74      G4int thisNb;          // this is integer for counter
75      G4double thisRa;
76      G4double thisAng;
77
78      G4int NbChromatin;       // stores total number of chromatin
79
80      G4double* xC;
81      G4double* yC;
82      G4double* zC;
83
84  };
85
86  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
87
88  #endif
```

A.1.2.3 PrimaryGeneatorAction.hh

```
#ifndef PrimaryGeneratorAction_h
#define PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"
//#include "G4ParticleGun.hh"
#include "G4GeneralParticleSource.hh"    // using the General Particle Source instead
    of ParticleGun
#include "DetectorConstruction.hh"
#include "G4Event.hh"
#include "G4ParticleTable.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
public:

  PrimaryGeneratorAction(DetectorConstruction*);
  ~PrimaryGeneratorAction();

  void GeneratePrimaries(G4Event*);

  //void GetEventNum()  (return eventNum;);

private:

  //G4ParticleGun*          particleGun;
  G4GeneralParticleSource*    GPS;          // using GPS instead
  DetectorConstruction*      Detector;
};
#endif
```

A.1.2.4 PhysicsList.hh

```
#ifndef PhysicsList_h
#define PhysicsList_h 1

#include "G4VUserPhysicsList.hh"
#include "G4ProcessManager.hh"
```

```
6  #include "G4ParticleTypes.hh"
7
8  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
9
10 class PhysicsList: public G4VUserPhysicsList
11 {
12   public:
13
14     PhysicsList();
15     virtual ~PhysicsList();
16
17     void SetGammaCut(G4double);
18     void SetElectronCut(G4double);
19     void SetPositronCut(G4double);
20     void SetProtonCut(G4double);
21
22   protected:
23
24     // these methods construct particles
25     void ConstructBosons();
26     void ConstructLeptons();
27     void ConstructBarions();
28
29     // these methods construct physics processes and register them
30     void ConstructGeneral();
31     void ConstructEM();
32
33     // Construct particle and physics
34     void ConstructParticle();
35     void ConstructProcess();
36
37     // set cuts
38     void SetCuts();
39
40   private:
41     G4double cutForGamma;
42     G4double cutForElectron;
43     G4double cutForPositron;
44     G4double cutForProton;
45
46 };
```

```
47 #endif
```

A.1.2.5 G4ElectronCapture.hh

```
1  #ifndef ElectronCapture_h
2  #define ElectronCapture_h 1
3
4  #include "G4VDiscreteProcess.hh"
5  #include "globals.hh"
6
7  class G4Region;
8
9  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
10
11 class G4ElectronCapture : public G4VDiscreteProcess
12 {
13   public:
14
15     G4ElectronCapture(const G4String& regName, G4double ekinlimit);
16
17     virtual ~G4ElectronCapture();
18
19     void SetKinEnergyLimit(G4double);
20
21     virtual void BuildPhysicsTable(const G4ParticleDefinition&);
22
23     virtual G4bool IsApplicable(const G4ParticleDefinition&);
24
25     virtual G4double PostStepGetPhysicalInteractionLength( const G4Track& track,
26                             G4double previousStepSize,
27                             G4ForceCondition* condition);
28
29     virtual G4VParticleChange* PostStepDoIt(const G4Track&, const G4Step&);
30
31   protected:
32
33     virtual G4double GetMeanFreePath(const G4Track&, G4double,G4ForceCondition*);
34
35   private:
36
37     // hide assignment operator as private
```

```
38      G4ElectronCapture(const G4ElectronCapture&);
39      G4ElectronCapture& operator = (const G4ElectronCapture &right);
40
41      G4double kinEnergyThreshold;
42      G4String regionName;
43      G4Region* region;
44  };
45
46  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
47
48  #endif
```

A.1.2.6 RunAction.hh

```
1  #ifndef RunAction_h
2  #define RunAction_h 1
3
4  #include "DetectorConstruction.hh"
5  #include "HistoManager.hh"
6
7  #include "G4UserRunAction.hh"
8  #include "globals.hh"
9  #include <iostream>
10
11 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
12
13 class G4Run;
14 class G4Timer;
15
16 class RunAction : public G4UserRunAction
17 {
18 public:
19
20   RunAction(DetectorConstruction*, HistoManager *);
21   ~RunAction();
22
23   void BeginOfRunAction(const G4Run*);
24   void EndOfRunAction(const G4Run*);
25
26   //the equation below add the dose for the run
27   void AddDoseN(G4double energy){doseN +=energy;}
```

```
28    G4double GetDoseN(){return doseN;}
29
30    void AddDoseNC(G4double energy){doseNC +=energy;}
31    G4double GetDoseNC(){return doseNC;}
32
33    void AddDoseC(G4double energy){doseC +=energy;}
34    G4double GetDoseC(){return doseC;}
35
36    //void SetDoseCBeginEvt(G4double energy){DoseBegin = energy;}
37    //G4double GetDoseCBeginEvt(){return DoseBegin;}
38
39
40 private:
41
42    DetectorConstruction* Detector;
43    HistoManager* Histo;          //
44    G4double doseN;              //
45    G4double doseNC;
46    G4double doseC;
47    G4double DoseBegin;
48    G4double doseDelta;           // intializing doseDelta
49    G4Timer* timer;              // pointer for timer.
50
51 };
52 #endif
```

A.1.2.7 EventAction.hh

```
1 #ifndef EventAction_h
2 #define EventAction_h 1
3
4 #include "G4UserEventAction.hh"
5
6 class G4Event;
7 class HistoManager;
8 class RunAction;
9
10 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
11
12 class EventAction : public G4UserEventAction
13 {
```

```
14    public:
15      EventAction(HistoManager*, RunAction*);
16      ~EventAction();
17
18    public:
19      void BeginOfEventAction(const G4Event*);
20      void EndOfEventAction(const G4Event*);
21
22    private:
23      HistoManager*       Histo;
24      RunAction*        Run;
25
26  };
27
28  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
29
30  #endif
```

A.1.2.8 SteppingAction.hh

```
1  #ifndef SteppingAction_h
2  #define SteppingAction_h 1
3
4  #include "G4UserSteppingAction.hh"
5
6  class RunAction;
7  class DetectorConstruction;
8  class PrimaryGeneratorAction;
9  class HistoManager;
10
11 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
12
13 class SteppingAction : public G4UserSteppingAction
14 {
15 public:
16
17   SteppingAction(RunAction*, DetectorConstruction*, PrimaryGeneratorAction*,
         HistoManager*);
18   ~SteppingAction();
19
20   void UserSteppingAction(const G4Step*);
```

```
21
22 private:
23
24   RunAction*           Run;       // not sure what the point of these pointers are. Is
         this not done in the class file with run, det, pri, histo?
25   DetectorConstruction*   Detector;
26   PrimaryGeneratorAction*   Primary;
27   HistoManager*        Histo;
28
29 };
30 #endif
```

A.1.2.9 SteppingVerbose.hh

```
1 class SteppingVerbose;
2
3 #ifndef SteppingVerbose_h
4 #define SteppingVerbose_h 1
5
6 #include "G4SteppingVerbose.hh"
7 #include "G4UnitsTable.hh"
8
9 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
10
11 class SteppingVerbose : public G4SteppingVerbose
12 {
13   public:
14
15     SteppingVerbose();
16     ~SteppingVerbose();
17
18     void StepInfo();
19     void TrackingStarted();
20 };
21 #endif
```

A.1.2.10 HistoManager.hh

```
1 #ifndef HistoManager_h
2 #define HistoManager_h 1
```

```
3
4  #include <fstream>
5  #include <iostream>
6
7  #include "globals.hh"
8
9  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
10
11   // AIDA is Abstract Interfaces for Data Analysis
12   // http://aida.freehep.org/
13 namespace AIDA
14 {
15   class IAnalysisFactory;
16   class ITree;
17   class ITuple;
18 }
19
20 const G4int MaxNtupl = 3;        // this set the max ntupl. this has to change for new
       ntuple
21
22 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
23
24 class HistoManager
25 {
26 public:
27
28   HistoManager();
29   ~HistoManager();
30
31   void book();       // book function.  has no return and thus no type. void function
32   void save();       // save function. has no return and thus no type. just performs
         an action.
33   void process(std::ostream& a_out, G4int con);
34   void stream(G4int con, G4int event_id);
35   void FillNtuple(G4int id, G4int column, G4double value);
36   void AddRowNtuple(G4int id);
37   void ResetNtuple(G4int id);
38
39 private:
40
41   G4String           fileName[2];       // fileName is a string [2]
```

```
42    G4String           fileType;        //
43    G4String           fileOption;
44    AIDA::IAnalysisFactory*    af;
45    AIDA::ITree*         tree;
46    AIDA::ITuple*        ntupl0;        // pointer for ntupl0. additional line for
          another ntuple
47    AIDA::ITuple*        ntupl1;
48    AIDA::ITuple*        ntupl2;
49    std::string          csvfilename;
50
51    G4bool             factoryOn;
52
53 };
54
55 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
56
57 #endif
```

A.1.3 src

A.1.3.1 DetectorConstruction.cc

```
1  #include "DetectorConstruction.hh"
2  #include "G4Region.hh"
3  #include "G4ProductionCuts.hh"
4  #include "RunAction.hh"
5  #include "G4PVParameterised.hh"
6  #include "ChromatinParameterisation.hh"
7
8
9  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
10
11 DetectorConstruction::DetectorConstruction()     //constructor for this class
12 :physiWorld(NULL), logicWorld(NULL), solidWorld(NULL)
13 {}
14
15 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
16
17 DetectorConstruction::~DetectorConstruction()    //destructor for this class
18 {}
19
```

```
20 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
21
22 G4VPhysicalVolume* DetectorConstruction::Construct()
23
24 {
25   DefineMaterials();
26   return ConstructDetector();
27 }
28
29 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
30
31 void DetectorConstruction::DefineMaterials()
32 {
33
34   // Water is defined from NIST database
35   G4NistManager * man = G4NistManager::Instance();
36   G4Material * H2O = man->FindOrBuildMaterial("G4_WATER");
37
38   // Default materials in setup
39   waterMaterial = H2O;
40
41 }
42
43 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
44
45 G4VPhysicalVolume* DetectorConstruction::ConstructDetector()
46 {
47
48 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
49
50 // World Volume // The Purpose of this volume is to surround everything. Physics are
       off here.
51
52   WorldDiameter = .0031*mm;          // diameter of world. Change here.
53   WorldHeight = .0011*mm;
54
55   solidWorld = new G4Tubs("World",       // name of this solid volume (in this case
         the World volume)
56               0,          // inside radius of world?
57               WorldDiameter/2,  // outside radius of world?
58               WorldHeight/2,
```

```
59                0,           // angular extent minimum
60                twopi);        // angular extent maximum
61
62    logicWorld = new G4LogicalVolume(solidWorld,  // the parent solid of this logical
          volume
63                waterMaterial,     // the material of this logical volume
64                "World");      // the name of this logical volume
65
66    physiWorld = new G4PVPlacement(0,      // governs placement of physical volume.
          There is no rotation.
67                G4ThreeVector(),  // placement of the volume
68                "World",       // name of the volume
69                logicWorld,      // the logical volume associated with this physical
                      placement
70                0,           // the mother volume. In this case 0 because this is the "
                      world"
71                false,         // no boolean operation
72                0);          // copy number (what?)
73
74
75  // Medium Volume // The purpose of this volume is to provide a medium around the
        target. Physics can be activated in this region.
76
77    G4double MediumDiameter = .999*WorldDiameter; // setting the medium just inside the
          world. (for now)
78    G4double MediumHeight = .999*WorldHeight;
79
80    G4Tubs* solidMedium = new G4Tubs("Medium",       // name of solid
81                        0,           // inside radius?
82                        MediumDiameter/2, // outside radius
83                        MediumHeight/2,
84                        0,           //
85                        twopi);
86
87    G4LogicalVolume* logicMedium = new G4LogicalVolume(solidMedium,    // parent solid
          of this logical volume
88                            waterMaterial,       // its material
89                            "Medium");         // its name
90
91    G4VPhysicalVolume* physiMedium = new G4PVPlacement(0,        // no rotation of this
          volume
```

```
92                          G4ThreeVector(),     // default placement by a three vector
                                (0,0,0)
93                          "Medium",         // its name
94                          logicMedium,        // its associated logical volume
95                          physiWorld,         // its mother volume (physical)
96                          false,            // boolean
97                          0);             // copy number
98
99
100 // The nucleus - this volume is the nucleus. The nucleus is represented in this
        example as a cylinder.
101
102   NucleusHeight = .001*mm;        // height of nucleus
103   NucleusDiameter = .003*mm;        // diameter of nucleus
104
105   G4Tubs* solidNucleus = new G4Tubs("Nucleus",       // its name
106                   0,              // inner radius (a cylinder thus zero)
107                   NucleusDiameter/2,     // outer radius. tube is created from
                            centroid
108                   NucleusHeight/2,     // half length. The tube is created from the
                            centroid so it extends -2.5 to 2.5 micron
109                   0,              // phi
110                   twopi);           // phi (it is a full tube 0-2pi)
111
112   G4LogicalVolume* logicNucleus = new G4LogicalVolume(solidNucleus,    // its
          associated solid
113                           waterMaterial,         // its assigned material
114                           "Nucleus");          // its name
115
116   G4VPhysicalVolume* physiNucleus = new G4PVPlacement(0,           // there is no
          rotation of this volume respectively
117                           G4ThreeVector(),       // default placement by a three vector
                                (0,0,0)
118                           "Nucleus",           // its name
119                           logicNucleus,        // associated logical volume
120                           physiMedium,         // parent physical volume
121                           false,             // boolean
122                           0);              // copy number
123
124 // The chromatin fibers.
125
```

```
126  // here are variables setting up parameterisation. will be tweaked
127  G4double ChromatinDiameter = .00003*mm;      // 30nm diameter
128  G4double ChromatinHeight = NucleusHeight;
129  G4int NbOfShells = 9;              // inclusive of zeroeth shell
130  G4int NbFirstShell = 8;
131  G4int NbOfChromatin = 1633;            // should eventually either get rid of this (
         go by shells only) or rectify to match in parameterisation class
132
133  // The number of chromatin was designed to meet the 16% threshold of DBSCAN. Here
         it is 16.33%.
134
135
136  G4Tubs* solidChromatin = new G4Tubs("Chromatin",
137                  0,
138                  ChromatinDiameter/2,
139                  ChromatinHeight/2,
140                  0,
141                  twopi);
142
143  G4LogicalVolume* logicChromatin = new G4LogicalVolume(solidChromatin,
144                          waterMaterial,
145                          "Chromatin");
146
147  G4VPVParameterisation* ChromatinParam = new ChromatinParameterisation(NbOfShells,
148                                NbFirstShell,
149                                ChromatinHeight,
150                                NucleusDiameter,
151                                ChromatinDiameter);
152
153  G4VPhysicalVolume* physiChromatin = new G4PVParameterised("Chromatin",
154                              logicChromatin,
155                              logicNucleus,
156                              kUndefined,
157                              NbOfChromatin,
158                              ChromatinParam);
159
160
161  // below the density of the target is set. (this might should be called from
         material settings... using this for dose calculations)
162
163  G4double NucleusVolume = NucleusHeight*pi*NucleusDiameter*NucleusDiameter/4;
```

```
164   G4double density = 1.0*g/cm3;
165   NucleusMass = density*NucleusVolume;
166   G4cout << "\n\n\n\nNucleusVolume" << NucleusVolume << "\nNucleusMass = " <<
          NucleusMass << "\n";
167   G4double ChromatinVolume = NbOfChromatin*ChromatinHeight*pi*ChromatinDiameter*
          ChromatinDiameter/4;
168   ChromatinMass = density*ChromatinVolume;
169
170
171
172 // Visualization attributes
173
174   G4VisAttributes* worldVisAtt = new G4VisAttributes(G4Colour(1.0,1.0,1.0));     //
          white
175   worldVisAtt->SetVisibility(true);                          // visibility is true
176   logicWorld->SetVisAttributes(worldVisAtt);                      // logicWorld (
          logical volume) set to worldVisAtt (white and visible)
177
178   G4VisAttributes* nucleusVisAtt = new G4VisAttributes(G4Colour(1.0,0.0,0.0));  //
          red
179   nucleusVisAtt->SetVisibility(true);                        // visible
180   logicNucleus->SetVisAttributes(nucleusVisAtt);                  // logicNucleus set
           to nucleusVisAtt (red and visible)
181
182   G4VisAttributes* chromatinVisAtt = new G4VisAttributes(G4Colour(0.0,1.0,0.0));  //
          blue????
183   chromatinVisAtt->SetVisibility(true);                      // visible
184   logicChromatin->SetVisAttributes(chromatinVisAtt);              // logicNucleus
          set to nucleusVisAtt (red and visible)
185
186
187
188
189 // Create G4Region (Defines a region or a group of regions in the
190 // detector geometry setup, sharing properties associated to materials
191 // or production cuts which may affect or bias specific physics processes.)
192 // I am setting up two regions (for now) for the nucleus and the medium.
193
194   mediumRegion = new G4Region("Medium");          // setting up region for medium (
          here we specify physical volume, I THINK???, or just names the Region???)
```

```
195    //nucleusRegion = new G4Region("Nucleus");         // setting up region for nucleus
           (here we specify physical volume, I THINK???)
196
197    G4ProductionCuts* cuts = new G4ProductionCuts();     // cutoffs for particle
           production
198
199    G4double defCut = 1*nanometer;                  // setting a default cut of 1nanometer.
            If particle won't travel further it is terminated.
200    cuts->SetProductionCut(defCut,"gamma");           // setting production cut (cuts)
           for gammas to defCut
201    cuts->SetProductionCut(defCut,"e-");            // setting production cut (cuts) for
           electrons to defCut
202    cuts->SetProductionCut(defCut,"e+");            // setting production cut (cuts) for
           positrons to defCut
203    cuts->SetProductionCut(defCut,"proton");          // setting production cut (cuts)
           for protons to defCut
204
205
206    mediumRegion->SetProductionCuts(cuts);            // setting production cuts in
           medium (to cuts)
207    mediumRegion->AddRootLogicalVolume(logicMedium);     // adding the logical volume
           for medium
208    //nucleusRegion->SetProductionCuts(cuts);          // setting production cuts in
           nucleus (to cuts)
209    //nucleusRegion->AddRootLogicalVolume(logicNucleus);     // adding the logical
           volume for nucleus (could add others to this region or others)
210
211    return physiWorld;
212 }
```

A.1.3.2 ChromatinParameterisation.cc

```
1 #include "ChromatinParameterisation.hh"
2 #include "G4VPhysicalVolume.hh"
3 #include "G4ThreeVector.hh"
4 #include "G4Tubs.hh"
5
6 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
7
8 ChromatinParameterisation::ChromatinParameterisation(
9                     G4int NbOfShells,
```

```
10                  G4int NbFirstShell,
11                  G4double NucleusHeight,
12                  G4double NucleusDiameter,
13                  G4double ChromatinDiameter)
14
15 {
16   fNb = NbOfShells;
17   fFS = NbFirstShell;
18   fND = NucleusDiameter;
19   fNH = NucleusHeight;
20   fCD = ChromatinDiameter;
21
22   //temporary
23
24   //temporary
25
26
27   // Below are calculations for setting out cylinders
28
29   gap = fND/2/fNb;
30
31   // This section will count how many members (and future correction can be done here
         )
32
33
34   NbEach = new G4double[fNb];    // array is created. It must be pointer because it
         has variable size therefore needs to use dynamic memory allocation.
35   RaEach = new G4double[fNb];
36   AngEach = new G4double[fNb];
37
38   thisNb = 1;          // stores for each shell, here it is 1 for zeroeth shell
39   thisRa = 0;
40   thisAng = 0;
41
42   NbEach[0] = thisNb;              // sets number for zeroeth
43   RaEach[0] = thisRa;
44   AngEach[0] = thisAng;
45
46   NbChromatin = thisNb;        // stores total number of chromatin
47
48
```

```
49
50
51    for (G4int i=1; i < (fNb); i++)        // storing these so that they can be
          manipulated more easily with a rectifying function
52    {
53      thisNb = fFS*i*i;              // number on this shell
54      NbEach[i] = thisNb;              // number on the shell in question
55
56      thisRa = i*gap;
57      RaEach[i] = thisRa;
58
59      thisAng = 2*pi/thisNb;
60      AngEach[i] = thisAng;
61
62      NbChromatin = thisNb + NbChromatin;    // adds to NbChromatin
63    }
64
65    xC = new G4double[NbChromatin];
66    yC = new G4double[NbChromatin];
67    zC = new G4double[NbChromatin];
68    G4int index = 0;
69    G4double angle;
70
71    for (G4int i = 0; i < (fNb); i++)
72    {
73      thisAng = AngEach[i];              // the spacing for this row.
74      thisRa = RaEach[i];
75
76      for (G4int ii = 0; ii < NbEach[i]; ii++)
77      {
78        angle = thisAng*ii;            // the angle for each cylinder
79        xC[index] = cos(angle)*thisRa;
80        yC[index] = sin(angle)*thisRa;
81        zC[index] = 0;
82        index++;
83      }
84
85    }
86  }
87
88  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

```
89
90 ChromatinParameterisation::~ChromatinParameterisation()
91 {}
92
93 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
94
95 void ChromatinParameterisation::ComputeTransformation(const G4int copyNo,
       G4VPhysicalVolume* physVol) const
96 {
97   G4ThreeVector centers(xC[copyNo],yC[copyNo],zC[copyNo]);       // beware will need
         to hand calc this until there is a rectifying function in place
98   physVol->SetTranslation(centers);
99 }
100
101 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
102
103 void ChromatinParameterisation::ComputeDimensions(G4Tubs& Chromatin, const G4int
       copyNo, const G4VPhysicalVolume*) const
104 {
105   Chromatin.SetInnerRadius(0);
106   Chromatin.SetOuterRadius(fCD/2);
107   Chromatin.SetZHalfLength(fNH/2);
108   Chromatin.SetStartPhiAngle(0);
109   Chromatin.SetDeltaPhiAngle(2*pi);
110 }
```

A.1.3.3 PrimaryGeneratorAction.cc

```
1 #include "PrimaryGeneratorAction.hh"     // include the primarygeneratoraction header
2 #include "Randomize.hh"            // include the randomize header for the random
      number generator
3 #include "DetectorConstruction.hh"       // include the detectorconstruction header to
       access those values stored
4
5 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
6
7 PrimaryGeneratorAction::PrimaryGeneratorAction(DetectorConstruction* det)
8 :Detector(det)
9 {
10   G4int n_particle = 1;
11   //GPS = new G4GeneralParticleSource(n_particle);
```

```
12
13    // NOrmally you can replace G4ParticleGun with G4GeneralParticleSource
14    // straight-up. Howeve,r ParticleGun can take number of particles
15    // GPS can't. Need to use SetNumberOfParticle function inside GPS
16
17    GPS = new G4GeneralParticleSource();
18    GPS->SetNumberOfParticles(n_particle);
19
20
21    // below are the default gun parameters
22
23    //  these have been commented out because they are not used for GPS. Use macro
          instead?
24    //GPS->SetParticleEnergy(10.*MeV);
25    //GPS->SetParticleMomentumDirection(G4Threevector(0.,0.1.));
26    //GPS->SetParticlePosition(G4ThreeVector(0.,0.,0.*mm));
27  }
28
29  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
30
31  PrimaryGeneratorAction::~PrimaryGeneratorAction()
32  {
33    delete GPS;
34  }
35
36  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
37
38  void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
39  {
40    GPS->GeneratePrimaryVertex(anEvent);
41    // may be able to hardcode some default particle settings in here such as cosine
          law surface distribution.
42    // Look at /source/event/include/G4GeneralParticleSource then ../
          G4SingleParticleSource then ../G4SPSAngDistribution.hh
43    // might work...might not. Would allow the things that don't normally vary to be
          hard coded (though the difference between this
44    // and just always including it in the macro inputted in the terminal is minimal...
45    // minimal meaning no difference (I don't think) except that if I am not going to
          allow it to change why require it to be
46    // inputted.
47  }
```

A.1.3.4 PhysicsList.cc

```
#include "PhysicsList.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

#include "PhysicsList.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

PhysicsList::PhysicsList(): G4VUserPhysicsList()
{
  defaultCutValue   = 1*micrometer;      // setting cuts
  cutForGamma      = defaultCutValue;
  cutForElectron    = defaultCutValue;
  cutForPositron    = defaultCutValue;
  cutForProton     = defaultCutValue;

  SetVerboseLevel(1);
}


//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

PhysicsList::~PhysicsList()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructParticle()
{
  // Construct Particles
  ConstructBosons();
  ConstructLeptons();
  ConstructBarions();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructBosons()
{
```

```
40    // Gamma
41    G4Gamma::GammaDefinition();
42  }
43
44  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
45
46  void PhysicsList::ConstructLeptons()
47  {
48    // Leptons
49    G4Electron::ElectronDefinition();
50    G4Positron::PositronDefinition();
51  }
52
53  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
54
55  // Including DNA
56  #include "G4DNAGenericIonsManager.hh"
57  // End of DNA
58
59  void PhysicsList::ConstructBarions()
60  {
61    // Barions
62    G4Proton::ProtonDefinition();
63    G4GenericIon::GenericIonDefinition();
64
65    // Geant4-DNA new particles
66    G4DNAGenericIonsManager* genericIonsManager;      // G4DNAGenericIonManager *
          genericIonsManager
67    genericIonsManager=G4DNAGenericIonsManager::Instance();
68    genericIonsManager->GetIon("alpha++");
69    genericIonsManager->GetIon("alpha+");
70    genericIonsManager->GetIon("helium");
71    genericIonsManager->GetIon("hydrogen");
72    // Invoking additional ions added in 4.9.5
73    genericIonsManager->GetIon("carbon");
74    genericIonsManager->GetIon("nitrogen");
75    genericIonsManager->GetIon("oxygen");
76    genericIonsManager->GetIon("iron");
77  }
78
79  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
```

```
80
81 void PhysicsList::ConstructProcess()
82 {
83   AddTransportation();
84   ConstructEM();
85   ConstructGeneral();
86 }
87
88 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
89
90 // Geant4-DNA MODELS
91
92 #include "G4DNAElastic.hh"
93 #include "G4DNAChampionElasticModel.hh"
94 #include "G4DNAScreenedRutherfordElasticModel.hh"
95
96 #include "G4DNAExcitation.hh"
97 #include "G4DNAMillerGreenExcitationModel.hh"
98 #include "G4DNABornExcitationModel.hh"
99
100 #include "G4DNAIonisation.hh"
101 #include "G4DNABornIonisationModel.hh"
102 #include "G4DNARuddIonisationModel.hh"
103
104 #include "G4DNAChargeDecrease.hh"
105 #include "G4DNADingfelderChargeDecreaseModel.hh"
106
107 #include "G4DNAChargeIncrease.hh"
108 #include "G4DNADingfelderChargeIncreaseModel.hh"
109
110 #include "G4DNAAttachment.hh"
111 #include "G4DNAMeltonAttachmentModel.hh"
112
113 #include "G4DNAVibExcitation.hh"
114 #include "G4DNASancheExcitationModel.hh"
115
116 //
117
118 #include "G4LossTableManager.hh"
119 #include "G4EmConfigurator.hh"
120 #include "G4VEmModel.hh"
```

```
121 #include "G4DummyModel.hh"
122 #include "G4eIonisation.hh"
123 #include "G4hIonisation.hh"
124 #include "G4ionIonisation.hh"
125 #include "G4eMultipleScattering.hh"      // source/processes/electromagnetic/standard/
126 #include "G4hMultipleScattering.hh"      // source/processes/electromagnetic/standard/
127 #include "G4BraggIonGasModel.hh"
128 #include "G4BetheBlochIonGasModel.hh"
129 #include "G4UrbanMscModel93.hh"
130 #include "G4MollerBhabhaModel.hh"
131 #include "G4IonFluctuations.hh"
132 #include "G4UniversalFluctuation.hh"
133
134 #include "G4ElectronCapture.hh"
135
136 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
137
138 void PhysicsList::ConstructEM()
139 {
140
141   theParticleIterator->reset();      // reset condition of iterator for while loop
142
143   while( (*theParticleIterator)() )
144   {
145
146     G4ParticleDefinition* particle = theParticleIterator->value(); // set value for
          iterator
147     G4ProcessManager* pmanager = particle->GetProcessManager();   // source/processes
          /management/src/G4ProcessManager.cc
148     G4String particleName = particle->GetParticleName();
149
150     // ******************************
151     // 1) Processes for the World region
152     // ******************************
153
154     // Process for Electrons
155     if (particleName == "e-")
156     {
157
158       // STANDARD msc is active in the world
159       //
```

```
160         G4eMultipleScattering* msc = new G4eMultipleScattering();
161         pmanager->AddProcess(msc, -1, 1, 1);                    // Process, AtRest, AlongStep
                , PostStep. -1 is inactive. otherwise it is order of DoItmethod. Inverse
                order of GetInteractionLength
162
163         // STANDARD ionisation is active in the world
164         G4eIonisation* eion = new G4eIonisation();
165         eion->SetEmModel(new G4MollerBhabhaModel(), 1);
166           // first number sets model and second number gives size of a vector (and
                index).
167           // I think when two models are used that you have to give 1, 2, etc as it
                needs to store them all????
168           // see:
169           // \source\processes\electromagnetic\standard\src\G4eIonization.cc
170           // \source\processes\electromagnetic\standard\src\G4MollerBhabhaModel.cc
171           // \source\processes\electromagnetic\utils\src\G4VEnergyLossProcess.cc
172         pmanager->AddProcess(eion, -1, 2, 2);
173           // Process, AtRest, AlongStep, PostStep. -1 is inactive. otherwise it is
                order of DoItmethod. Inverse order of GetInteractionLength
174
175         // DNA elastic is not active in the world
176         G4DNAElastic* theDNAElasticProcess = new G4DNAElastic("e-_G4DNAElastic");
177         theDNAElasticProcess->SetModel(new G4DummyModel(),1);
178         pmanager->AddDiscreteProcess(theDNAElasticProcess);
179
180         // DNA excitation is not active in the world
181         G4DNAExcitation* dnaex = new G4DNAExcitation("e-_G4DNAExcitation");
182         dnaex->SetModel(new G4DummyModel(),1);
183         pmanager->AddDiscreteProcess(dnaex);
184
185         // DNA ionisation is not active in the world
186         G4DNAIonisation* dnaioni = new G4DNAIonisation("e-_G4DNAIonisation");
187         dnaioni->SetModel(new G4DummyModel(),1);
188         pmanager->AddDiscreteProcess(dnaioni);
189
190         // DNA attachment is not active in the world
191         G4DNAAttachment* dnaatt = new G4DNAAttachment("e-_G4DNAAttachment");
192         dnaatt->SetModel(new G4DummyModel(),1);
193         pmanager->AddDiscreteProcess(dnaatt);
194
195         // DNA vib. excitation is not active in the world
```

```
196         G4DNAVibExcitation* dnavib = new G4DNAVibExcitation("e-_G4DNAVibExcitation");
197         dnavib->SetModel(new G4DummyModel(),1);
198         pmanager->AddDiscreteProcess(dnavib);
199
200         // THE FOLLOWING PROCESS WILL KILL ALL ELECTRONS BELOW A SELECTED ENERY
                THRESHOLD
201         // Capture of low-energy e-G4ElectronCapture* ecap = new G4ElectronCapture("
                Target", 5.1*eV);
202         G4ElectronCapture* ecap = new G4ElectronCapture("Medium",5.1*eV);
203         pmanager->AddDiscreteProcess(ecap);
204       }
205
206       // Processes for Protons
207       else if( particleName == "proton" )
208       {
209         // STANDARD msc is active in the world
210         G4hMultipleScattering* msc = new G4hMultipleScattering();
211         pmanager->AddProcess(msc, -1, 1, 1);
212
213         // STANDARD ionisation is active in the world
214         G4hIonisation* hion = new G4hIonisation();
215         hion->SetEmModel(new G4BraggIonGasModel(), 1);
216         hion->SetEmModel(new G4BetheBlochIonGasModel(), 2);
217         pmanager->AddProcess(hion, -1, 2, 2);
218
219         // DNA excitation is not active in the world
220         G4DNAExcitation* dnaex = new G4DNAExcitation("proton_G4DNAExcitation");
221         dnaex->SetModel(new G4DummyModel(),1);
222         dnaex->SetModel(new G4DummyModel(),2);
223         pmanager->AddDiscreteProcess(dnaex);
224
225         // DNA ionisation is not active in the world
226         G4DNAIonisation* dnaioni = new G4DNAIonisation("proton_G4DNAIonisation");
227         dnaioni->SetModel(new G4DummyModel(),1);
228         dnaioni->SetModel(new G4DummyModel(),2);
229         pmanager->AddDiscreteProcess(dnaioni);
230
231         // DNA charge decrease is ACTIVE in the world since no corresponding STANDARD
                process exist
232         pmanager->AddDiscreteProcess(new G4DNAChargeDecrease("
                proton_G4DNAChargeDecrease"));
```

```
233     }
234
235     // Processes for Hydrogen (proton and electron, zero charge)
236     else if ( particleName == "hydrogen" )
237     {
238       // DNA processes are ACTIVE in the world since no corresponding STANDARD
              processes exist
239       pmanager->AddDiscreteProcess(new G4DNAIonisation("hydrogen_G4DNAIonisation"));
240       pmanager->AddDiscreteProcess(new G4DNAExcitation("hydrogen_G4DNAExcitation"));
241       pmanager->AddDiscreteProcess(new G4DNAChargeIncrease("
              hydrogen_G4DNAChargeIncrease"));
242     }
243
244     else if (particleName == "GenericIon")
245     {
246     // THIS IS NEEDED FOR STANDARD ALPHA G4ionIonisation PROCESS
247
248       // STANDARD msc is active in the world
249       pmanager->AddProcess(new G4hMultipleScattering, -1, 1, 1);
250
251       // STANDARD ionisation is active in the world
252       G4ionIonisation* hion = new G4ionIonisation();
253       hion->SetEmModel(new G4BraggIonGasModel(),1);
254       hion->SetEmModel(new G4BetheBlochIonGasModel(), 2);
255       pmanager->AddProcess(hion, -1, 2, 2);
256     }
257
258     // Alphas
259     else if ( particleName == "alpha" )
260     {
261       // STANDARD msc is active in the world
262       G4hMultipleScattering* msc = new G4hMultipleScattering();
263       pmanager->AddProcess(msc, -1, 1, 1);
264
265       // STANDARD ionisation is active in the world
266       G4ionIonisation* hion = new G4ionIonisation();
267       hion->SetEmModel(new G4BraggIonGasModel(),1);
268       hion->SetEmModel(new G4BetheBlochIonGasModel(), 2);
269       pmanager->AddProcess(hion, -1, 2, 2);
270
271       // DNA excitation is not active in the world
```

```
272         G4DNAExcitation* dnaex = new G4DNAExcitation("alpha_G4DNAExcitation");
273         dnaex->SetModel(new G4DummyModel(),1);
274         pmanager->AddDiscreteProcess(dnaex);
275 
276         // DNA ionisation is not active in the world
277         G4DNAIonisation* dnaioni = new G4DNAIonisation("alpha_G4DNAIonisation");
278         dnaioni->SetModel(new G4DummyModel(),1);
279         pmanager->AddDiscreteProcess(dnaioni);
280 
281         // DNA charge decrease is ACTIVE in the world since no corresponding STANDARD
                process exist
282         pmanager->AddDiscreteProcess(new G4DNAChargeDecrease("alpha_G4DNAChargeDecrease
                "));
283       }
284 
285       // Alpha+ (singularily charged alphas)
286       else if ( particleName == "alpha+" )
287       {
288 
289         // DNA processes are ACTIVE in the world since no corresponding STANDARD
                processes exist
290         pmanager->AddDiscreteProcess(new G4DNAExcitation("alpha+_G4DNAExcitation"));
291         pmanager->AddDiscreteProcess(new G4DNAIonisation("alpha+_G4DNAIonisation"));
292         pmanager->AddDiscreteProcess(new G4DNAChargeDecrease("alpha+
                _G4DNAChargeDecrease"));
293         pmanager->AddDiscreteProcess(new G4DNAChargeIncrease("alpha+
                _G4DNAChargeIncrease"));
294 
295       }
296 
297       // Helium atoms (neutral alphas)
298       else if ( particleName == "helium" )
299       {
300         // DNA processes are ACTIVE in the world since no corresponding STANDARD
                processes exist
301         pmanager->AddDiscreteProcess(new G4DNAExcitation("helium_G4DNAExcitation"));
302         pmanager->AddDiscreteProcess(new G4DNAIonisation("helium_G4DNAIonisation"));
303         pmanager->AddDiscreteProcess(new G4DNAChargeIncrease("
                helium_G4DNAChargeIncrease"));
304       }
305 
```

```
306     // THESE ARE NEW TREATMENTS ADDED 2012/03/23 by mcoghill3
307     // Carbon atoms (neutral carbon)
308     else if ( particleName == "carbon" )
309     {
310       //G4cout << "\n\n carbon \n\n" << G4endl;
311       // DNA process are active in the world since no corresponding STANDARD
              processes exist??? (is this true??? mcoghill3 added this 2012-02-14
312       pmanager->AddDiscreteProcess(new G4DNAIonisation("carbon_G4DNAIonisation"));
313     }
314
315     else if ( particleName == "nitrogen" )
316     {
317       //G4cout << "\n\n nitrogen \n\n" << G4endl;
318       // DNA process are active in the world since no corresponding STANDARD
              processes exist??? (is this true??? mcoghill3 added this 2012-02-14
319       pmanager->AddDiscreteProcess(new G4DNAIonisation("nitrogen_G4DNAIonisation"));
320     }
321
322     else if ( particleName == "oxygen" )
323     {
324       //G4cout << "\n\n oxygen \n\n" << G4endl;
325       // DNA process are active in the world since no corresponding STANDARD
              processes exist??? (is this true??? mcoghill3 added this 2012-02-14
326       pmanager->AddDiscreteProcess(new G4DNAIonisation("oxygen_G4DNAIonisation"));
327     }
328
329     else if ( particleName == "iron" )
330     {
331       //G4cout << "\n\n iron \n\n" << G4endl;
332       // DNA process are active in the world since no corresponding STANDARD
              processes exist??? (is this true??? mcoghill3 added this 2012-02-14
333       pmanager->AddDiscreteProcess(new G4DNAIonisation("iron_G4DNAIonisation"));
334     }
335   }
336
337   // ************************************
338   // 2)   Define processes for Target Area
339   //    For these purposes we will activate
340   //    physics in the medium for now
341   //    the medium includes the nucleus and
342   //    other volumes included in the nucleus
```

```
343   // ************************************
344
345   // STANDARD EM processes should be inactivated when corresponding DNA processes are
          used
346   // - STANDARD EM e- processes are inactivated below 1 MeV
347   // - STANDARD EM proton & alpha processes are inactivated below standEnergyLimit
348
349   G4double standEnergyLimit = 99.9*MeV;
350
351   G4double massFactor = 1.0079/4.0026;
352   G4EmConfigurator* em_config = G4LossTableManager::Instance()->EmConfigurator();
353
354   G4VEmModel* mod;
355
356   // *** e-
357
358   // ---> STANDARD EM processes are inactivated below 1 MeV
359
360   mod =  new G4UrbanMscModel93();
361   mod->SetActivationLowEnergyLimit(1*MeV);
362   em_config->SetExtraEmModel("e-","msc",mod,"Medium");
363
364   mod = new G4MollerBhabhaModel();
365   mod->SetActivationLowEnergyLimit(0.99*MeV);
366   em_config->SetExtraEmModel("e-","eIoni",mod,"Medium",0.0,100*TeV, new
          G4UniversalFluctuation());
367
368   // ---> DNA processes activated
369
370   mod = new G4DNAChampionElasticModel();
371   em_config->SetExtraEmModel("e-","e-_G4DNAElastic",mod,"Medium",0.0,1*MeV);
372
373   mod = new G4DNABornIonisationModel();
374   em_config->SetExtraEmModel("e-","e-_G4DNAIonisation",mod,"Medium",11*eV,1*MeV);
375
376   mod = new G4DNABornExcitationModel();
377   em_config->SetExtraEmModel("e-","e-_G4DNAExcitation",mod,"Medium",9*eV,1*MeV);
378
379   mod = new G4DNAMeltonAttachmentModel();
380   em_config->SetExtraEmModel("e-","e-_G4DNAAttachment",mod,"Medium",4*eV,13*eV);
381
```

```
382     mod = new G4DNASancheExcitationModel();
383     em_config->SetExtraEmModel("e-","e-_G4DNAVibExcitation",mod,"Medium",2*eV,100*eV);
384
385     // *** proton
386
387     // ---> STANDARD EM processes inactivated below standEnergyLimit
388
389     // STANDARD msc is still active
390     // Inactivate following STANDARD processes
391
392     mod = new G4BraggIonGasModel();
393     mod->SetActivationLowEnergyLimit(standEnergyLimit);
394     em_config->SetExtraEmModel("proton","hIoni",mod,"Medium",0.0,2*MeV, new
            G4IonFluctuations());
395
396     mod = new G4BetheBlochIonGasModel();
397     mod->SetActivationLowEnergyLimit(standEnergyLimit);
398     em_config->SetExtraEmModel("proton","hIoni",mod,"Medium",2*MeV,100*TeV, new
            G4UniversalFluctuation());
399
400     // ---> DNA processes activated
401
402     mod = new G4DNARuddIonisationModel();
403     em_config->SetExtraEmModel("proton","proton_G4DNAIonisation",mod,"Medium",0.0,0.5*
            MeV);
404
405     mod = new G4DNABornIonisationModel();
406     em_config->SetExtraEmModel("proton","proton_G4DNAIonisation",mod,"Medium",0.5*MeV
            ,100*MeV);
407
408     mod = new G4DNAMillerGreenExcitationModel();
409     em_config->SetExtraEmModel("proton","proton_G4DNAExcitation",mod,"Medium",10*eV
            ,0.5*MeV);
410
411     mod = new G4DNABornExcitationModel();
412     em_config->SetExtraEmModel("proton","proton_G4DNAExcitation",mod,"Medium",0.5*MeV
            ,100*MeV);
413
414     // *** alpha
415
416     // ---> STANDARD EM processes inactivated below standEnergyLimit
```

```
417
418     // STANDARD msc is still active
419     // Inactivate following STANDARD processes
420
421     mod = new G4BraggIonGasModel();
422     mod->SetActivationLowEnergyLimit(standEnergyLimit);
423     em_config->SetExtraEmModel("alpha","ionIoni",mod,"Medium",0.0,2*MeV/massFactor, new
            G4IonFluctuations());
424
425     mod = new G4BetheBlochIonGasModel();
426     mod->SetActivationLowEnergyLimit(standEnergyLimit);
427     em_config->SetExtraEmModel("alpha","ionIoni",mod,"Medium",2*MeV/massFactor,100*TeV,
            new G4UniversalFluctuation());
428
429     // ---> DNA processes activated
430
431     mod = new G4DNARuddIonisationModel();
432     em_config->SetExtraEmModel("alpha","alpha_G4DNAIonisation",mod,"Medium",0.0,100*MeV
            );
433
434     mod = new G4DNAMillerGreenExcitationModel();
435     em_config->SetExtraEmModel("alpha","alpha_G4DNAExcitation",mod,"Medium",1*keV,100*
            MeV);
436 }
437
438 //....oooOO0OOooo.........oooOO0OOooo.........oooOO0OOooo.........oooOO0OOooo....
439
440 void PhysicsList::ConstructGeneral()
441 {}
442
443 //....oooOO0OOooo.........oooOO0OOooo.........oooOO0OOooo.........oooOO0OOooo....
444
445 void PhysicsList::SetCuts()
446 {
447   if (verboseLevel > 0 )
448   {
449     G4cout << "PhysicsList::SetCuts:" << "CutLength : " << G4BestUnit(defaultCutValue
            ,"Length") << G4endl;
450   }
451
452   // set cut values for gamma first, electrons second, positrons third
```

```
453    // this is because some processes for electrons and positrons need cut values for
           gamma (duh)
454    // appears these are all overridden in the Medium by the G4Region section of
           DetectorConstruction.
455    SetCutValue(cutForGamma,"gamma");
456    SetCutValue(cutForElectron,"e-");
457    SetCutValue(cutForPositron,"e+");
458    SetCutValue(cutForProton,"proton");
459
460    if (verboseLevel>0)
461    {
462      DumpCutValuesTable();
463    }
464 }
```

A.1.3.5 G4ElectronCapture.cc

```
1  #include "G4ElectronCapture.hh"
2  #include "G4ParticleDefinition.hh"
3  #include "G4Step.hh"
4  #include "G4Track.hh"
5  #include "G4Region.hh"
6  #include "G4RegionStore.hh"
7  #include "G4Electron.hh"
8
9  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
10
11 G4ElectronCapture::G4ElectronCapture(const G4String& regName, G4double ekinlim)
12   : G4VDiscreteProcess("eCapture", fElectromagnetic), kinEnergyThreshold(ekinlim),
13     regionName(regName), region(0)
14 {
15   if(regName == "" || regName == "world") {
16     regionName = "DefaultRegionForTheWorld";
17   }
18 }
19
20 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
21
22 G4ElectronCapture::~G4ElectronCapture()
23 {}
24
```

```
25 //....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....
26
27 void G4ElectronCapture::SetKinEnergyLimit(G4double val)
28 {
29   kinEnergyThreshold = val;
30   if(verboseLevel > 0) {
31     G4cout << "### G4ElectronCapture: Tracking cut E(MeV) = "
32      << kinEnergyThreshold/MeV << G4endl;
33   }
34 }
35 //....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....
36
37 void G4ElectronCapture::BuildPhysicsTable(const G4ParticleDefinition&)
38 {
39   region = (G4RegionStore::GetInstance())->GetRegion(regionName);
40   if(region && verboseLevel > 0) {
41     G4cout << "### G4ElectronCapture: Tracking cut E(MeV) = "
42      << kinEnergyThreshold/MeV << " is assigned to " << regionName
43      << G4endl;
44   }
45 }
46
47 //....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....
48
49 G4bool G4ElectronCapture::IsApplicable(const G4ParticleDefinition&)
50 {
51   return true;
52 }
53
54 G4double
55 G4ElectronCapture::PostStepGetPhysicalInteractionLength(const G4Track& aTrack,
56               G4double,
57               G4ForceCondition* condition)
58 {
59   // condition is set to "Not Forced"
60   *condition = NotForced;
61
62   G4double limit = DBL_MAX;
63   if(region) {
64     if(aTrack.GetVolume()->GetLogicalVolume()->GetRegion() == region &&
65    aTrack.GetKineticEnergy() < kinEnergyThreshold) { limit = 0.0; }
```

```
66    }
67    return limit;
68 }
69
70 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
71
72 G4VParticleChange* G4ElectronCapture::PostStepDoIt(const G4Track& aTrack,
73                  const G4Step&)
74 {
75    pParticleChange->Initialize(aTrack);
76    pParticleChange->ProposeTrackStatus(fStopAndKill);
77    pParticleChange->ProposeLocalEnergyDeposit(aTrack.GetKineticEnergy());
78    return pParticleChange;
79 }
80
81 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
82
83 G4double G4ElectronCapture::GetMeanFreePath(const G4Track&,G4double,
84                 G4ForceCondition*)
85 {
86    return DBL_MAX;
87 }
```

A.1.3.6 RunAction.cc

```
1 #include "RunAction.hh"
2 #include "G4Run.hh"
3 #include "G4Timer.hh"       // include for timer
4
5 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
6
7 RunAction::RunAction(DetectorConstruction* det, HistoManager* his)     // using
      variable in namespace RunAction
8 :Detector(det),Histo(his)
9 {
10    timer = new G4Timer;                          // timer constructor
11 }
12
13 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
14
15 RunAction::~RunAction()                          // destructor
```

```
16 {
17   delete timer;                              // timer destructor
18 }
19
20 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
21
22 void RunAction::BeginOfRunAction(const G4Run* run)
23 {
24   // Histograms
25   Histo->book();
26
27   // setting up for dose calculations
28   doseN = 0;
29   doseNC = 0;
30   doseC = 0;
31   doseDelta = 0;
32
33   G4cout << "### Run" << run->GetRunID() << " start." << G4endl;          // get run ID
34   timer->Start();                                  // start the timer
35 }
36
37 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
38
39 void RunAction::EndOfRunAction(const G4Run* run)
40 {
41
42   //Histo->stream();
43
44   // save the histogram
45
46   Histo->stream(2,0); //should create another function to overload???
47
48   Histo->save();
49
50   G4cout << "\n \n Dose in Nucleus: " << GetDoseN() << " Gy \n \n" << G4endl;
51   // ouput does to the terminal
52   G4cout << "\n \n Dose in Nucleus || Chromatin: " << GetDoseNC() << " Gy \n \n" <<
         G4endl;
53   // ouput does to the terminal
54   G4cout << "\n \n Dose in Chromatin: " << GetDoseC() << " Gy \n \n" << G4endl;
55   // ouput does to the terminal
```

```
56
57    timer->Stop();                                    // stop the timer
58    G4cout << "number of events: " << run->GetNumberOfEvent() << " " << *timer <<
          G4endl;    // output number of events
59 }
```

A.1.3.7 EventAction.cc

```
1  #include "EventAction.hh"
2  #include "G4Event.hh"
3  #include "G4EventManager.hh"
4  #include "HistoManager.hh"
5  #include "RunAction.hh"
6
7  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
8
9  EventAction::EventAction(HistoManager* his, RunAction* run)
10 :Histo(his),Run(run)
11 {}
12
13 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
14
15 EventAction::~EventAction()
16 {}
17
18 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
19
20 void EventAction::BeginOfEventAction(const G4Event*)
21 {
22   //Run->SetDoseCBeginEvt(Run->GetDoseC());
23 }
24
25 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
26
27 void EventAction::EndOfEventAction(const G4Event* evt)
28 {
29   G4int event_id = evt->GetEventID();
30   G4int event_num = event_id+1;
31   G4cout << "\n (EAC) Event Number: " << event_id << "\n";
32   G4double DoseNEvt = Run->GetDoseN();
33   G4double DoseNCEvt = Run->GetDoseNC();
```

```
34    G4double DoseCEvt = Run->GetDoseC();
35    G4double DoseEvtNorm = DoseNCEvt/event_num;
36    G4cout << "\n DoseNCEvtNorm: " << DoseEvtNorm << "\n";
37
38    Histo->FillNtuple(2, 0, event_num);
39    Histo->FillNtuple(2, 1, DoseNEvt);
40    Histo->FillNtuple(2, 2, DoseNCEvt);
41    Histo->FillNtuple(2, 3, DoseCEvt);
42    Histo->AddRowNtuple(2);
43
44    Histo->stream(1,event_id);
45    Histo->ResetNtuple(1);
46 }
```

A.1.3.8 SteppingAction.cc

```
1  #include "SteppingAction.hh"
2  #include "RunAction.hh"
3  #include "DetectorConstruction.hh"
4  #include "PrimaryGeneratorAction.hh"
5  #include "HistoManager.hh"
6
7  #include "G4SteppingManager.hh"
8  #include "G4VTouchable.hh"
9  #include "G4VPhysicalVolume.hh"
10
11 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
12
13 SteppingAction::SteppingAction(RunAction* run, DetectorConstruction* det,
       PrimaryGeneratorAction* pri, HistoManager* his)
14 :Run(run),Detector(det),Primary(pri),Histo(his)
15 {}
16
17 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
18
19 SteppingAction::~SteppingAction()
20 {}
21
22 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
23
```

```
24 void SteppingAction::UserSteppingAction(const G4Step* step)   // what is this void
       function? task5a/src/SteppingAction.cc this is called at each G4Step
25 {
26   G4double flagParticle=0.;
27   G4double flagProcess=0.;
28   G4double x,y,z,xp,yp,zp,e;
29   G4double mod;
30   G4double DoseC,DoseNC,DoseN; //DoseBegin,DoseInc;
31
32   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "e-")        flagParticle = 1;
33   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "proton")    flagParticle = 2;
34   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "hydrogen") flagParticle = 3;
35   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "alpha")     flagParticle = 4;
36   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "alpha+")    flagParticle = 5;
37   if (step->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() ==
         "helium")    flagParticle = 6;
38
39   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="msc")
           flagProcess =10;
40   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="e-
         _G4DNAElastic")    flagProcess =11;
41   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="e-
         _G4DNAExcitation")     flagProcess =12;
42   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="e-
         _G4DNAIonisation")     flagProcess =13;
43   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="e-
         _G4DNAAttachment")     flagProcess =14;
44   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="e-
         _G4DNAVibExcitation")    flagProcess =15;
45   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="eCapture"
         )       flagProcess =16;
46
47   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
         proton_G4DNAExcitation")  flagProcess =17;
48   if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
         proton_G4DNAIonisation")  flagProcess =18;
```

```
49  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        proton_G4DNAChargeDecrease")  flagProcess =19;
50
51  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        hydrogen_G4DNAExcitation")  flagProcess =20;
52  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        hydrogen_G4DNAIonisation")  flagProcess =21;
53  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        hydrogen_G4DNAChargeIncrease")  flagProcess =22;
54
55  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        alpha_G4DNAExcitation")  flagProcess =23;
56  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        alpha_G4DNAIonisation")  flagProcess =24;
57  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        alpha_G4DNAChargeDecrease") flagProcess =25;
58
59  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="alpha+
        _G4DNAExcitation")  flagProcess =26;
60  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="alpha+
        _G4DNAIonisation")  flagProcess =27;
61  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="alpha+
        _G4DNAChargeDecrease")  flagProcess =28;
62  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="alpha+
        _G4DNAChargeIncrease")  flagProcess =29;
63
64  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        helium_G4DNAExcitation")  flagProcess =30;
65  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        helium_G4DNAIonisation")  flagProcess =31;
66  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="
        helium_G4DNAChargeIncrease")  flagProcess =32;
67
68  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="hIoni")
            flagProcess =33;
69  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()=="eIoni")
            flagProcess =34;
70
71
```

```
72  if (step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName()!="
        Transportation")      // if statement defined below for examples that are not
        transport
73  {
74    x=step->GetPreStepPoint()->GetPosition().x()/nanometer;
75    y=step->GetPreStepPoint()->GetPosition().y()/nanometer;
76    z=step->GetPreStepPoint()->GetPosition().z()/nanometer;
77    xp=step->GetPostStepPoint()->GetPosition().x()/nanometer;
78    yp=step->GetPostStepPoint()->GetPosition().y()/nanometer;
79    zp=step->GetPostStepPoint()->GetPosition().z()/nanometer;
80    e=step->GetTotalEnergyDeposit()/eV;
81
82    Histo->FillNtuple(0, 0, flagParticle);
83    Histo->FillNtuple(0, 1, flagProcess);
84    Histo->FillNtuple(0, 2, x);
85    Histo->FillNtuple(0, 3, y);
86    Histo->FillNtuple(0, 4, z);
87    Histo->FillNtuple(0, 5, e);
88    Histo->FillNtuple(0, 6, sqrt((xp-x)*(xp-x)+(yp-y)*(yp-y)+(zp-z)*(zp-z)));
89    Histo->AddRowNtuple(0);
90
91    // below I am setting up the dose counter. I limit it to only the nucleus. Later
          I might limit it to the chromatin when that is added.
92
93    if(step->GetPreStepPoint()->GetPhysicalVolume()->GetName() == "Nucleus")
94    {
95      DoseN = (step->GetTotalEnergyDeposit()/joule)/(Detector->GetNucleusMass()/kg);
              // if this is just nucleus and not chromatin need some subraction for
          mass
96      Run->AddDoseN(DoseN);
97
98    }
99
100   if((step->GetPreStepPoint()->GetPhysicalVolume()->GetName() == "Nucleus") || (
          step->GetPreStepPoint()->GetPhysicalVolume()->GetName() == "Chromatin"))
101   {
102     DoseNC = (step->GetTotalEnergyDeposit()/joule)/(Detector->GetNucleusMass()/kg);
103     Run->AddDoseNC(DoseNC);
104     if(e >= 5)
105     {
106       //DoseBegin = Run->GetDoseCBeginEvt();
```

```
107           //DoseInc = DoseC - DoseBegin;
108           //G4cout << "\n DoseC: " << DoseC << "\n DoseBegin: " << DoseBegin << "\n
                  DoseInc: " << DoseInc << G4endl;
109           Histo->FillNtuple(1, 0, x);
110           Histo->FillNtuple(1, 1, y);
111           Histo->FillNtuple(1, 2, z);
112           Histo->FillNtuple(1, 3, e);
113           Histo->AddRowNtuple(1);
114         }
115       }
116
117       if(step->GetPreStepPoint()->GetPhysicalVolume()->GetName() == "Chromatin")  //
              might get some speed by tucking chromatin and nucleus checks into the check
              for both. Not sure... but maybe.
118       {
119         DoseC = (step->GetTotalEnergyDeposit()/joule)/(Detector->GetChromatinMass()/kg)
                ;
120         //could be
121         //DoseC = (step->GetTotalEnergyDeposit())/(Detector->GetChromatinMass())*Gy;
122
123         Run->AddDoseC(DoseC);
124
125       }
126     }
127 }
```

A.1.3.9 SteppingVerbose.cc

```
1  #include "SteppingVerbose.hh"
2
3  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
4
5  SteppingVerbose::SteppingVerbose()
6  {}
7
8  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
9
10 SteppingVerbose::~SteppingVerbose()
11 {}
12
13 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
```

```
14
15 void SteppingVerbose::StepInfo()
16 {
17   CopyState();
18
19   G4int prec = G4cout.precision(3);
20
21   if( verboseLevel >= 1 )
22   {
23     if( verboseLevel >= 4 ) VerboseTrack();
24     if( verboseLevel >= 3 )
25     {
26       G4cout  << G4endl;
27       G4cout  << std::setw( 5) << "#Step#"      << " "
28           << std::setw( 6) << "X"           << "      "
29           << std::setw( 6) << "Y"           << "      "
30           << std::setw( 6) << "Z"           << "      "
31           << std::setw( 9) << "KineE"       << " "
32           << std::setw( 9) << "dEStep"      << " "
33           << std::setw(10) << "StepLeng"
34           << std::setw(10) << "TrakLeng"
35           << std::setw(10) << "NextVolu"
36           << std::setw(10) << "Process"    << G4endl;
37     }
38
39     G4cout  << std::setw( 5) << fTrack->GetCurrentStepNumber() << " "
40         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
41         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
42         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
43         << std::setw( 6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
44         << std::setw( 6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy")
45         << std::setw( 6) << G4BestUnit(fStep->GetStepLength(),"Length")
46         << std::setw( 6) << G4BestUnit(fTrack->GetTrackLength(),"Length");
47
48     // if( fStepStatus != fWorldBoundary){
49     if( fTrack->GetNextVolume() != 0 )
50     {
51       G4cout << std::setw(10) << fTrack->GetNextVolume()->GetName();
52     }
53
54     else
```

```
55      {
56        G4cout << std::setw(10) << "OutOfWorld";
57      }
58
59      if(fStep->GetPostStepPoint()->GetProcessDefinedStep() != NULL)
60      {
61        G4cout << std::setw(10) << fStep->GetPostStepPoint()->GetProcessDefinedStep()->
              GetProcessName();
62      }
63
64      else
65      {
66        G4cout << "User Limit";
67      }
68
69      G4cout << G4endl;
70
71      if( verboseLevel == 2 )
72      {
73        G4int tN2ndariesTot =   fN2ndariesAtRestDoIt + fN2ndariesAlongStepDoIt +
              fN2ndariesPostStepDoIt;
74
75        if(tN2ndariesTot>0)
76        {
77          G4cout  << "     :----- List of 2ndaries - "
78              << "#SpawnInStep=" << std::setw(3) << tN2ndariesTot
79              << "(Rest="  << std::setw(2) << fN2ndariesAtRestDoIt
80              << ",Along=" << std::setw(2) << fN2ndariesAlongStepDoIt
81              << ",Post="  << std::setw(2) << fN2ndariesPostStepDoIt
82              << "), "
83              << "#SpawnTotal=" << std::setw(3) << (*fSecondary).size()
84              << " ----------------"
85              << G4endl;
86          for(size_t lp1=(*fSecondary).size()-tN2ndariesTot;
87          lp1<(*fSecondary).size(); lp1++)
88          {
89            G4cout  << "     : "
90                << std::setw(6)
91                << G4BestUnit((*fSecondary)[lp1]->GetPosition().x(),"Length")
92                << std::setw(6)
93                << G4BestUnit((*fSecondary)[lp1]->GetPosition().y(),"Length")
```

```
94                 << std::setw(6)
95                 << G4BestUnit((*fSecondary)[lp1]->GetPosition().z(),"Length")
96                 << std::setw(6)
97                 << G4BestUnit((*fSecondary)[lp1]->GetKineticEnergy(),"Energy")
98                 << std::setw(10)
99                 << (*fSecondary)[lp1]->GetDefinition()->GetParticleName();
100              G4cout << G4endl;
101            }
102
103            G4cout  << "     :----------------------------------"
104                << "----------------------------------------"
105                << "-- EndOf2ndaries Info ---------------"
106                << G4endl;
107          }
108        }
109
110    }
111 G4cout.precision(prec);
112 //G4cout<< "exit StepppingVerbose::StepInfo    " <<G4endl;
113 }
114
115 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
116
117 void SteppingVerbose::TrackingStarted()
118 {
119   CopyState();
120   G4int prec = G4cout.precision(3);
121   if( verboseLevel > 0 )
122   {
123     G4cout  << std::setw( 5) << "Step#" << " "
124         << std::setw( 6) << "X" << "     "
125         << std::setw( 6) << "Y" << "     "
126         << std::setw( 6) << "Z" << "     "
127         << std::setw( 9) << "KineE" << " "
128         << std::setw( 9) << "dEStep" << " "
129         << std::setw(10) << "StepLeng"
130         << std::setw(10) << "TrakLeng"
131         << std::setw(10) << "NextVolu"
132         << std::setw(10) << "Process" << G4endl;
133     G4cout  << std::setw( 5) << fTrack->GetCurrentStepNumber() << " "
134         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
```

```
135         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
136         << std::setw( 6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
137         << std::setw( 6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
138         << std::setw( 6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy")
139         << std::setw( 6) << G4BestUnit(fStep->GetStepLength(),"Length")
140         << std::setw( 6) << G4BestUnit(fTrack->GetTrackLength(),"Length");
141     if(fTrack->GetNextVolume())
142     {
143       G4cout  << std::setw(10) << fTrack->GetNextVolume()->GetName() << " ";
144     }
145     else
146     {
147       G4cout  << std::setw(10) << "OutOfWorld" << " ";
148     }
149     G4cout  << std::setw(10) << "initStep" << G4endl;
150   }
151   G4cout.precision(prec);
152   G4cout  << "exit SteppingVerbose::TrackingStarted()    " <<G4endl;
153 }
```

A.1.3.10 HistoManager.cc

```
1 #include "HistoManager.hh"
2 #include "G4UnitsTable.hh"
3
4
5
6 #ifdef G4ANALYSIS_USE
7 #include "AIDA/AIDA.h"
8 #endif
9
10 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
11
12 HistoManager::HistoManager()
13 :af(0),tree(0),factoryOn(false)
14 {
15 #ifdef G4ANALYSIS_USE
16   // Creating the analysis factory
17   af = AIDA_createAnalysisFactory();
18   if(!af)
19   {
```

```
20      G4cout << " HistoManager::HistoManager() :" << " problem creating the AIDA
            analysis factory." << G4endl;
21    }
22 #endif
23
24    fileName[0] = "vad3-4";
25    fileType = "root";
26    fileOption = "export=root";
27
28    ntupl0=0;             // initialize ntupl0
29    ntupl1=0;             // initialize ntupl1
30    ntupl2=0;             // initialize ntupl2
31 }
32
33 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
34
35 HistoManager::~HistoManager()
36 {
37 #ifdef G4ANALYSIS_USE
38    delete af;
39 #endif
40 }
41
42 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
43
44 void HistoManager::book()         // void function book
45 {
46 #ifdef G4ANALYSIS_USE
47    if(!af) return;
48
49    // Creating a tree mapped to an hbook file
50    fileName[1] = fileName[0] + "." + fileType;    // add file type to file name (and
          next)
51    G4bool readOnly = false;
52    G4bool createNew = true;
53    AIDA::ITreeFactory* tf = af->createTreeFactory();      // create tree factory
54    tree = tf->create(fileName[1], fileType, readOnly, createNew, fileOption);     //
          options for file creation. all PRESET in variables.
55    delete tf;       // delete pointer as it is no longer needed
56    if(!tree)
57    {
```

```
58      G4cout  << "HistoManager::book() :"
59          << "problem creating the AIDA tree with "
60          << " storeName = "  << fileName[1]
61          << " storeType = "  << fileType
62          << " readOnly = " << readOnly
63          << " createNew = "  << createNew
64          << " options = "  << fileOption
65          << G4endl;
66      return;
67    }
68
69    // Create a histogram and nutpl factory
70    AIDA::IHistogramFactory* hf = af->createHistogramFactory(*tree);       // hf points
          to
71    AIDA::ITupleFactory* ntf = af->createTupleFactory(*tree);
72
73    ntupl0 = ntf->create("ntuple0", "Beam Profile", "double flagParticle, flagProcess,
          x, y, z, e, d");   //naming fields for nutple10
74    ntupl1 = ntf->create("ntuple1", "SSB" , "double x, y, z, e");
75    ntupl2 = ntf->create("ntuple2", "Dose" , "double EventNum, EvtDoseN, EvtDoseNC,
          EvtDoseC");
76    factoryOn = true;
77
78    delete hf;     // delete the pointer
79    delete ntf;    // delete the pointer
80
81    if (factoryOn)
82      G4cout << "\n----> Histogram Tree is opened in " << fileName[1] << G4endl;
83  #endif
84  }
85
86  //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
87
88  void HistoManager::save()
89  {
90    #ifdef G4ANALYSIS_USE
91      if (factoryOn)
92      {
93        tree->commit();        // Writing the histograms to the file
94        tree->close();         // and closing the tree (and the file)
95        G4cout << "\n----> Histogram Tree is saved in " << fileName[1] << G4endl;
```

```
96        delete tree;
97        tree = 0;
98        factoryOn = false;
99      }
100   #endif
101 }
102
103 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
104
105 void HistoManager::process(std::ostream& a_out, G4int con)
106 {
107   #ifdef G4ANALYSIS_USE
108
109     //AIDA::ITupleFactory* tuple;
110
111     AIDA::ITuple* ntuplP;
112
113     if (con == 1)
114     {
115       ntuplP = ntupl1;
116       a_out << "\"x\",\"y\",\"z\",\"e\"" << G4endl;
117     }
118     else if (con == 2)
119     {
120       ntuplP = ntupl2;
121       a_out << "\"EventId\",\"DoseN\",\"DoseNC\",\"DoseC\"" << G4endl;
122     }
123     G4int coln = ntuplP->columns();
124     for (int i=0; i<coln; i++)
125     {
126       G4cout  << "process_file :"
127           << "icol = " << i
128           << ", label = " << ntuplP->columnName(i)
129           << ", type = " << ntuplP->columnType(i)
130           << G4endl;
131     }
132     G4cout << "process_file: rows = " << ntuplP->rows() << G4endl;
133
134     std::vector<std::string> types = ntuplP->columnTypes();
135
136     char del = ',';
```

```
137
138
139
140     ntuplP->start();
141     while(ntuplP->next())
142     {
143       for(int col=0;col<coln;col++)
144       {
145         if(col)
146         {
147           a_out << del;
148         }
149         if(types[col]=="float")
150         {
151           float v = ntuplP->getFloat(col);
152           a_out << v;
153         }
154         else if(types[col]=="double")
155         {
156           double v = ntuplP->getDouble(col);
157           a_out << v;
158         }
159         else if(types[col]=="char")
160         {
161           char v = ntuplP->getChar(col);
162           a_out << v;
163         }
164         else if(types[col]=="short")
165         {
166           short v = ntuplP->getShort(col);
167           a_out << v;
168         }
169         else if(types[col]=="int")
170         {
171           int v = ntuplP->getInt(col);
172           a_out << v;
173         }
174         else if(types[col]=="long")
175         {
176           long v = ntuplP->getLong(col);
177           a_out << v;
```

```
178         }
179         else if(types[col]=="boolean")
180         {
181           bool v = ntuplP->getBoolean(col);
182           a_out << v;
183         }
184         else if(types[col]=="string")
185         {
186           std::string v = ntuplP->getString(col);
187           a_out << v;
188         }
189         else
190         {
191           G4cout << "process_file : unknown type." << types[col] << G4endl;
192         }
193       }
194       a_out << G4endl;
195     }
196   #endif
197 }
198
199 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
200
201 void HistoManager::stream(G4int con, G4int event_id)
202 {
203
204   std::stringstream str;
205   if (con == 1)
206   {
207     str << "pSSB" << event_id+1 << ".csv";
208   }
209   else if (con == 2)
210   {
211     str << "Dose" << ".csv";
212   }
213   csvfilename = str.str();
214
215
216   std::ofstream out(csvfilename.c_str());
217   if(out.fail())
218   {
```

```
219     G4cout<< "can't open out.csv" << G4endl;
220   }
221
222   process(out, con);
223
224   out.close();
225 }
226
227
228 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
229
230 void HistoManager::FillNtuple(G4int nt, G4int column, G4double value)
231 {
232   if (nt >= MaxNtupl)
233   {
234     G4cout << "---> warning from HistoManager::FillNtuple() : Ntuple " << nt << "
          dose not exist " << column << value << G4endl;
235     return;
236   }
237 #ifdef G4ANALYSIS_USE
238   if(nt==0) ntupl0->fill(column, value);
239   if(nt==1) ntupl1->fill(column, value);
240   if(nt==2) ntupl2->fill(column, value);
241 #endif
242 }
243
244 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
245
246 void HistoManager::AddRowNtuple(G4int nt)
247 {
248   if (nt >= MaxNtupl)
249   {
250     G4cout  << "---> warning from HistoManager::AddRowNtuple() : Ntuple " << nt << "
          do not exist" << G4endl;
251     return;
252   }
253 #ifdef G4ANALYSIS_USE
254   if(nt==0) ntupl0->addRow();
255   if(nt==1) ntupl1->addRow();
256   if(nt==2) ntupl2->addRow();
257 #endif
```

```
258 }
259
260 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
261
262 void HistoManager::ResetNtuple(G4int nt)
263 {
264   if (nt >= MaxNtupl)
265   {
266     G4cout << "---> warning from HistoManager::ResetNtuple() : Ntuple " << nt << "
          do not exist" << G4endl;
267     return;
268   }
269 #ifdef G4ANALYSIS_USE
270   if(nt==0) ntupl0->reset();
271   if(nt==1) ntupl1->reset();
272   if(nt==2) ntupl2->reset();
273 #endif
274 }
```

Appendix B

R STATISTICAL PROGRAM CODE

This final version of this code for data processing, DSB-21s.R, can perform all of the data processing needed to produce the results in this thesis. This code requires instantiation of the 'fpc' library described in the Methods chapter. The command to instantiate this library is: *library('fpc')*.

B.1 DSB-21s.R

```
ion = "A";
concat = 1;
set.seed(2)
pct = 1;
emin = 5;
emax = 37.5;
eps = .8;
minpts = 2;
dirs = list.files(pattern=paste(ion,"-*",sep=""));
ifelse(length(dirs)==1,resultT<-matrix(0,length(dirs)+1,9),resultT<-matrix(0,length(
    dirs),9));
rownames(resultT)<-rownames(resultT,do.NULL=FALSE,prefix="xx");
ifelse(length(dirs)==1,clusterNbT<-matrix(0,length(dirs)+1,9),clusterNbT<-matrix(0,
    length(dirs),9));
rownames(clusterNbT)<-rownames(clusterNbT,do.NULL=FALSE,prefix="xx");
for(iii in 1:length(dirs))
{
  doseloc = paste(dirs[iii],"Dose.csv",sep="/");
  dose.m <- read.csv(doseloc,sep=",");
  names = list.files(path=dirs[iii],pattern="SSB*");
  result = matrix(0,length(names)+1,9);
  colnames(result) <- c("pSB","nSB","nSSB","nCLS","nDSB","Events","DoseN","DoseNC","
      DoseC");
  rownames(result)<-rownames(result,do.NULL=FALSE,prefix="Run");
  rownames(result)[length(names)+1]<-"Total";
```

```
23  clusterNb = matrix(0,length(names)+1,9);
24  colnames(clusterNb) <- c("1","2","3","4","5","6","7","8","More");
25  rownames(clusterNb)<-rownames(result,do.NULL=FALSE,prefix="Run");
26  rownames(clusterNb)[length(names)+1]<-"Total";
27  for(i in 1:(length(names)/concat))
28  {
29    name = paste("run",formatC(i,digits=3,wid=3,flag="0"),".raw", sep="");
30    place = matrix(0,0,4);
31    colnames(place) <- c("x","y","z","e");
32    for(iiii in 1:concat)
33    {
34      loc = paste(dirs[iii],names[concat*(i-1)+iiii],sep="/");
35      run.raw <- read.csv(loc,sep=",");
36      place = rbind(place,run.raw);
37    }
38    nCSV = nrow(place);
39    mod1 = matrix(runif(nCSV,0,1),nrow=nCSV);
40    run.hit = place[mod1<pct,1:4];
41    mod2 = matrix(runif(nrow(run.hit),0,1),nrow=nrow(run.hit));
42    mod3 = (run.hit[,4]-emin)/(emax-emin)>mod2;
43    run.SSB = run.hit[mod3,1:4];
44    result[i,6]=1;
45    result[i,7]=dose.m[i,2];
46    result[i,8]=dose.m[i,3];
47    result[i,9]=dose.m[i,4];
48    if(nrow(run.SSB)==0)
49    {
50      result[i,1:9]=0;
51    }
52    else
53    {
54      result[i,1] = nCSV;
55      run.SSB = run.SSB[,-4];
56      run.ds <-dbscan(run.SSB,eps,minpts);
57      result[i,2]=nrow(run.SSB);
58      result[i,3]=sum(run.ds$cluster==0);
59      result[i,4]=max(run.ds$cluster);
60      clusterNb[i,1]=sum(run.ds$cluster==0);
61      for(ii in 1:ifelse(result[i,4]==0,1,result[i,4]))
62      {
```

```
63            result[i,5]=result[i,5]+(1-.5^(ifelse(sum(run.ds$cluster==ii)==0,0,sum(run.ds
                  $cluster==ii)-1)));
64            number = ifelse(sum(run.ds$cluster==ii)<9,sum(run.ds$cluster==ii),9);
65            clusterNb[i,number]=clusterNb[i,number]+1;
66          }
67        }
68    }
69    result[length(names)+1,1]=sum(result[,1]);
70    result[length(names)+1,2]=sum(result[,2]);
71    result[length(names)+1,3]=sum(result[,3]);
72    result[length(names)+1,4]=sum(result[,4]);
73    result[length(names)+1,5]=sum(result[,5]);
74    result[length(names)+1,6]=sum(result[,6]);
75    result[length(names)+1,7]=max(result[,7]);
76    result[length(names)+1,8]=max(result[,8]);
77    result[length(names)+1,9]=max(result[,9]);
78    resultT[iii,] = result[length(names)+1,];
79    rownames(resultT)[iii] <- sub(paste(ion,"-",sep=""),'',dirs[iii]);
80    resultName <- paste("Results/",dirs[iii],".csv",sep="");
81    write.csv(result,resultName);
82    clusterNb[length(names)+1,1]=sum(clusterNb[,1]);
83    clusterNb[length(names)+1,2]=sum(clusterNb[,2]);
84    clusterNb[length(names)+1,3]=sum(clusterNb[,3]);
85    clusterNb[length(names)+1,4]=sum(clusterNb[,4]);
86    clusterNb[length(names)+1,5]=sum(clusterNb[,5]);
87    clusterNb[length(names)+1,6]=sum(clusterNb[,6]);
88    clusterNb[length(names)+1,7]=sum(clusterNb[,7]);
89    clusterNb[length(names)+1,8]=sum(clusterNb[,8]);
90    clusterNb[length(names)+1,9]=sum(clusterNb[,9]);
91    clusterNbT[iii,] = clusterNb[length(names)+1,];
92    rownames(clusterNbT)[iii] <- sub(paste(ion,"-",sep=""),'',dirs[iii]);
93    clusterNbName <- paste("Results/",dirs[iii],"cls.csv",sep="");
94    write.csv(clusterNb,clusterNbName);
95 }
96 resultT <- resultT[sort(rownames(resultT)),]
97 colnames(resultT) <- c("pSB","nSB","nSSB","nCLS","nDSB","Events","DoseN","DoseNC","
       DoseC");
98 folder = tail(strsplit(getwd(), "/")[[1]], 1);
99 resultFile = paste("Results/",folder,"-Results.csv",sep="");
100 write.csv(resultT,resultFile);
101 resultT
```

```
clusterNbT <- clusterNbT[sort(rownames(clusterNbT)),]
colnames(clusterNbT) <- c("1","2","3","4","5","6","7","8","More");
folder = tail(strsplit(getwd(), "/")[[1]], 1);
clusterNbFile = paste("Results/",folder,"-clusterNb.csv",sep="");
write.csv(clusterNbT,clusterNbFile);
clusterNbT
resultC=resultT;
clusterNbC=clusterNbT;
```

Appendix C

OTHER NOTES

C.1 Doxygen

GEANT4 is a large and complex toolkit. Its object-oriented nature, while being highly modular and manipulable, does not lend its code to being easily understood by the casual, inexperienced, user. The use of a documentation generator allows the simplified interpretation of GEANT4 code. The author recommends the use of the Doxygen software. The GEANT4 collaboration offers a Doxgyen version of documentation of recent release on its website. However, the author finds the personal use of Doxygen to be especially beneficial in understanding the interaction of user applications with GEANT4 base classes.[74]

REFERENCES

[1] "The R Project for Statistical Computing," March 2012. http://www.r-project.org/index.html.

[2] Agostinelli, S., Allison, J., Amako, K., Apostolakis, J., Araujo, H., Arce, P., Asai, M., Axen, D., Banerjee, S., Barrand, G., Behner, F., Bellagamba, L., Boudreau, J., Broglia, L., Brunengo, A., Burkhardt, H., Chauvie, S., Chuma, J., Chytracek, R., Cooperman, G., Cosmo, G., Degtyarenko, P., Dell'Acqua, A., Depaola, G., Dietrich, D., Enami, R., Fliciello, A., Ferguson, C., Fesefeldt, H., Folger, G., Foppiano, F., Forti, A., Garelli, S., Giani, S., Giannitrapani, R., Gibin, D., Cadenas, J. J. G., Gonzalez, I., Abril, G. G., Greeniaus, G., Greiner, W., Grichine, V., Grossheim, A., Guatelli, S., Gumplinger, P., Hamatsu, R., Hashimoto, K., Hasui, H., Heikkinen, A., Howard, A., Ivanchenko, V., Johnson, A., Jones, F. W., Kallenbach, J., Kanaya, N., Kawabata, M., Kawabata, Y., Kawaguti, M., Kelner, S., Kent, P., Kimura, A., Kodama, T., Kossov, M., Kurashige, H., Lamanna, E., Lara, T. L. V., Lefebure, V., Lei, F., Liendl, M., Lockman, W., Longo, F., Magni, S., Maire, M., Medernach, E., Minamimoto, K., de Freitas, P. M., Morita, Y., Murakami, K., Nagamatu, M., Nartallo, R., Nieminen, P., Nishimura, T., Ohtsubo, K., Okamura, M., O'Neale, S., Oohata, Y., Paech, K., Perl, J., Pfeiffer, A., Pia, M. G., Ranjard, F., Rybin, A., Sadilov, S., Salvo, E. D., Santin, G., Sasaki, T., Savvas, N., Sawada, Y., Scherer, S., Sei, S., Sirotenko, V., Smith, D., Starkov, N., Stoecker, H., Sulkimo, J., Takahata, M., Tanaka, S., Tcherniaev, E., Safai, E., Tehrani, Tropeano, M., Truscott, P., Uno, H., Urban, L., Urban, P., Verderi, M., Walkden, A., Wander, W., Weber, H., Wellisch, J. P., Wenaus, T., Williams, D. C., Wright, D., Yamada, T., Yoshida, H., and Zschiesche, D., "Geant4 - a simulation toolkit," *Nuclear Instruments and Methods in Physics Research A*, vol. 506, no. 3, pp. 250–303, 2003.

[3] Allison, J., Amako, K., Apostolakis, J., Araujo, H., Dubois, P. A., Asai, M., Barrand, G., Capra, R., Chauvie, S., Chytracek, Cirrone, G. A. P., Cooperman, G., Cosmo, G., Cuttone, G., Daquino, G. G., Donszelmann, M., Dressel, M., Folger, G., Foppiano, F., Generowicz, J., Grichine, V., Guatelli, S., Gumplinger, P., Heikkinen, A., Hrivnacova, I., Howard, A., Incerti, S., Ivanchenko, V., Johnson, T., Jones, F., Koi, T., Kokoulin, R., Kossov, M., Kurashige, H., Lara, V., Larsson, S., Lei, F., Link, O., Longo, F., Maire, M.,

Mantero, A., Mascialino, B., McLaren, I., Lorenzo, P. M., .Minamimoto, K.Murakami, Nieminen, P., Pandola, L., Parlati, S., Peralta, L., Perl, J., Pfeiffer, A., Pia, M. G., Ribon, A., Rodrigues, P., Russo, G., Sadilov, S., Santin, G., Sasaki, T., Smith, D., Starkov, N., Tanaka, S., Tcherniaev, E., Tom, B., Trindade, Truscott, P., Urban, L., Verderi, M., A.Walkden, P.Wellisch, J., C.Williams, D., D.Wright, and Yoshida, H., "Geant4 developments and applications," *IEEE Transactiosn on Nuclear Science*, vol. 53, pp. 270–278, February 2006.

[4] Amako, K., ed., *Proceedings of CHEP94*, no. LBL-35822 CONF-940492, (San Fransisco, CA, USA), 1994.

[5] Apostolakis, J., Asai, M., Bogdanov, A. G., Burkhardt, H., Cosmo, G., Elles, S., Folger, G., Grichine, V. M., Gumplinger, P., Heikkinen, A., Hrivnacova, I., Ivanchenko, V. N., Jacquemier, J., Koi, T., Kokoulin, R. P., Kossov, M., Kurashige, H., McLaren, I., Link, O., Maire, M., Pokorski, W., Sasaki, T., Starkov, N., Urban, L., and Wright, D. H., "Geometry and physics of the geant4 toolkit for high and medium energy applications," *Radiation Physics and Chemistry*, vol. 78, pp. 859–873, 2009.

[6] Attix, F. H., *Introduction to Radiological Physics and Radiation Dosimetry*. Wiley-VCH, 1986.

[7] Barendsen, G. W. and Beusker, T. L. J., "Effects of different ionizing radiations on human cells in tissue culture: I. irradiation techniques and dosimetry," *Radiation Research*, vol. 13, no. 6, pp. 832–840.

[8] Barendsen, G. W., Beusker, T. L. J., Vergroesen, A. J., and Budke, L., "Effects of different ionizing radiations on human cells in tissue culture: Ii. biological experiments," *Radiation Research*, vol. 13, no. 6, pp. 841–849.

[9] Barendsen, G. W. and Walter, H. M. D., "Effects of different ionizing radiations on human cells in tissue culture: Ii. modification of radiation damage," *Radiation Research*, vol. 21, no. 2, pp. 314–329.

[10] Barendsen, G. W., Walter, H. M. D., Fowler, J. F., and Bewley, D. K., "Effects of different ionizing radiations on human cells in tissue culture: Ii. experiments with cyclotron-accerlerated alpha-particles and deuterons," *Radiation Research*, vol. 18, no. 1, pp. 106–119.

[11] Bewley, D. K., "A comparison of the response of mammalian cells to fast neutrons and charged particle beams," *Radiation Research*, vol. 34, no. 2, pp. 446–458, 1968.

[12] Brenner, D. J. and Ward, J. F., "Constraints on energy deposition and target size of multiply damaged sites associated with dna double-strand breaks," *International Journal of Radiation Biology*, vol. 61, no. 6, pp. 637–748, 1992.

[13] Chauvie, S., Francis, Z., Guatelli, S., Incerti, S., Mascialino, B., Moretto, P., Nieminen, P., and Pia, M. G., "Models of biological effects of radiation in the geant4 toolkit," *IEEE Nuclear Science Symposium*, vol. 2, pp. 803–805, 2006.

[14] Chauvie, S., Francis, Z., Guatelli, S., Incerti, S., Mascialino, B., Moretto, P., Nieminen, P., and Pia, M. G., "Geant4 physics processes for microdosimetry simulation: Design foundation and iplementation of the first set of models," *IEEE Transactions on Nuclear Science*, vol. 54, pp. 2619–2628, December 2007.

[15] Cirrone, G. A. P., Cuttone, G., Donadio, S., Grichine, V., Guatclli, S., Gumplinger, P., Ivanchenko, V., Maire, M., Mantero, A., and Mascilino, B., "Precision validation of geant4 electromagnetic physics," vol. 1, pp. 482–485.

[16] Dingfelder, M., "Track structure: time evolution from physics to chemistry," *Radiation Protection Dosimetry*, vol. 122, no. 1-4, pp. 16–21, 2006.

[17] Dingfelder, M., Ritchie, R. H., Turner, J. E., Friedland, W., Paretzke, H. G., and Hamm, R. N., "Comparisons of calculations with PARTRAC and NOREC: transport of electrons in liquid water," *Radiation Research*, vol. 169, no. 5, pp. 584–594, 2008.

[18] Ester, M., Kriegel, H. P., Sander, J., and Xu, X., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Simoudis, E., Han, J., and Fayyad, U., eds.), KDD-96, pp. 226–231, Association for the Advancement of Artificial Intelligence, AAAI Press, 1996.

[19] Francis, Z., Incerti, S., Capra, R., Mascialino, B., Montarou, G., Stepan, V., and Villagrasa, C., "Molecular scale track structure simulations in liquid water using the geant4-dna monte-carlo processes," *Applied Radiation and Isotopes*, vol. 69, pp. 220–226, 2011.

[20] Francis, Z., Incerti, S., Ivanchenko, V., Champion, C., Karamitros, M., Bernal, M. A., and Bitar, Z. E., "Monte carlo simulation of energy-deposit clustering for ions of the same let in liquid water," *Physics in Medicine and Biology*, vol. 57, pp. 209–224, 2012.

[21] Francis, Z., Villagrasa, C., and Clairand, I., "Simulation of dna damage clustering after proton irradiation using an adapted dbscan algorithm," *Computer Methods and Programs in Biomedicine*, vol. 101, pp. 265–270, 2011.

[22] Freguson, C., *Geant4 General Particle Source Users Manual*. Uos-GSPM-Tech, 1.0 ed.

[23] Friedland, W., Dingfelder, M., Kundrat, P., and Jacob, P., "Track structures, DNA targets and radiation effects in the biophysical monte carlo simulation code PARTRAC," *Mutation Research*, vol. 711, pp. 28–40, 2011.

[24] Friedland, W., Jacob, P., Bernhardt, P., Paretzke, H. G., and Dingfelder, M., "Simulation of DNA damage after proton irradiation," *Radiation Research*, vol. 159, no. 3, pp. 401–410, 2003.

[25] Friedland, W., Jacob, P., and Kundrat, P., "Stochastic simulation of DNA double-strand break repair by non-homologous end joining based on track structure calculations," *Radiation Research*, vol. 173, no. 5, pp. 677–688, 2010.

[26] Friedland, W., Jacob, P., and Kundrat, P., "Mechanistic simulation of radiation damage to DNA and its repair: on the track towards systems radiation biology modelling," *Radiation Protection Dosimetry*, vol. 143, no. 2-4, pp. 542–548, 2011.

[27] Friedland, W., Jacob, P., Paretzke, H. G., Perzl, M., and Stork, T., "Simulation of strand breaks and short dna fragments in the biophysical model PARTRAC," vol. 204, pp. 43–46, 1997.

[28] Friedland, W., Jacob, P., Paretzke, H. G., and Stork, T., "Monte Carlo simulation of the production of short dna fragments by low-linear energy transfer radiation using higher-order dna models," *Radiation Research*, vol. 150, no. 2, pp. 170–182, 1998.

[29] Friedland, W., Kundrat, P., and Jacob, P., "Track structure calculations on hypothetical subcellular targets for the release of cell-killing signals in bystander experiments with medium transfer," *Radiation Protection Dosimetry*, vol. 143, no. 2-4, pp. 325–329, 2011.

[30] Friedland, W., Paretzke, H. G., Ballarini, F., Ottolenghi, A., Kreth, G., and Cremer, C., "First steps towards systems radiation biology studies concerned with dna and chromosome structure within living cells," *Radiation and Environmental Biophysics*, vol. 47, no. 1, pp. 49–61, 2008.

[31] Garty, G., Schulte, R., Shchemelinin, S., Leloup, C., Assaf, G., Breskin, A., Chechik, R., Bashkirov, V., Milligan, J., and Grosswendt, B., "A nanodosimetric model of radiation-induced clustered dna damage yields," *Physics in Medicine and Biology*, vol. 55, pp. 761–781, 2010.

[32] Geant4 Collaboration, *Geant4 user's guide for application developer's*, 2009. http://cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/fo/BookForAppliDev.pdf.

[33] Geant4 Collaboration, "Geant4," March 2012. http://geant4.cern.ch//.

[34] Geant4IN2P3, "Geant4 for VMware & VirtualBox," March 2012. http://geant4.in2p3.fr/spip.php?rubrique8.

[35] Goodhead, D. T., "An assessment of the role of microdosimetry in radiobiology," *Radiation Research*, vol. 91, no. 1, pp. 45–76, 1982.

[36] Goodhead, D. T., "Mechanisms for the biological effectiveness of high-let radiations," *Journal of Radiation Research*, vol. 40, no. SUPPL, pp. 1–13, 1999.

[37] Goodhead, D. T., "Energy deposition stochastic and track structure: what about the target," *Radiation Protection Dosimetry*, vol. 122, no. 1-4, pp. 3–15, 2006.

[38] Goodhead, D. T. and Nikjoo, H., "Current status of ultrasoft x rays and track structure analysis as tools for testing and developing biophysical models of radiation action," *Radiation Protection and Dosimetry*, vol. 31, no. 1-4, pp. 343–350, 1990.

[39] Hennig, C., "*fpc*: flexible procedures for clustering," February 2012. http://www.homepages.ucl.ac.uk/ ucakche/.

[40] ICRP, "Relative biological effectiveness (rbe), quality factor (q), and radiation weighting factor (w_r)," *Annals of the ICRP*, vol. 33, no. 4, 2003.

[41] ICRU, *ICRU Report 36: Microdosimetry*. International Commission on Radiation Units and Measurements, Bethesda, MD, USA, 1983.

[42] ICRU, *ICRU Report 60: Fundamental quantities and units for ionizing radiation*. International Commission on Radiation Units and Measurements, Bethesda, MD, USA, 1998.

[43] Incerti, S., "The geant4-dna project: Overview and status," October 2011. lecture slides.

[44] Incerti, S., Baldacchino, G., Bernal, M., Capra, R., Champion, C., Francis, Z., Guatelli, S., Gueye, P., Mantero, A., Mascialino, B., Moretto, P., Nieminen, P., Rosenfeld, A., Villagrasa, C., and Zacharatoul, C., "The geant4-dna project," tech. rep., The Geant4-DNA Collaboration, 2009.

[45] Incerti, S., Ivanchenko, A., Karamitros, M., Mantero, A., Moretto, P., Tran, H. N., Mascialino, B., Champion, C., Ivanchenko, V. N., Bernal, M. A., Francis, Z., Villagrasa, C., Baldacchino, G., Gueye, P., Capra, R., Nieminen, P., and Zacharatou, C., "Comparison of geant4 very low energy cross section models with experimental data in water," *Medical Physics*, vol. 37, September 2010.

[46] Karamitros, M., Mantero, A., Incerti, S., Friedland, W., Baldacchino, G., Barberet, P., Bernal, M., Capra, R., Champion, C., Bitar, Z. E., Francis, Z., Gueye, P., Ivanchenko, A., Ivanchenko, V., Kurashige, H., Mascialino, B., Moretto, P., Nieminen, P., Santin,

G., Seznec, H., Tran, H. N., Villagrasa, C., and Zacharatou, C., "Modeling radiation chemistry in the geant4 toolkit," *Progress in Nuclear Science and Technology*, vol. 2, pp. 503–508, 2011.

[47] Kellerer, A. M. and Chmelevsky, D., "Criteria for the applicability of let," *Radiation Research*, vol. 63, pp. 226–234, 1975.

[48] Kellerer, A. M. and Rossi, H. H., "Summary of quantities and functions employed in microdosimetry,"

[49] Kellerer, A. M. and Rossi, H. H., "Rbe and the primary mechanism of radiation action," *Radiation Research*, vol. 47, no. 1, pp. 15–34, 1971.

[50] Korolev, N., Fan, Y., Lyobartsev, A. P., and Nordenskiöld, L., "Modelling chromatin structure and dynamics: status and prospects," *Current Opinion in Structural Biology*, vol. 22, pp. 151–159, 2012.

[51] Kreipl, M. S., Friedland, W., and Paretzke, H. G., "Interaction of ion tracks in spatial and temporal proximity," *Radiation and Environmental Biophysics*, vol. 48, no. 4, pp. 349–359, 2009.

[52] Kreipl, M. S., Friedland, W., and Paretzke, H. G., "Time-and space-resolved Monte Carlo study of water radiolysis for photon, electron and ion irradiation," *Radiation and Environmental Biophysics*, vol. 48, no. 1, pp. 11–20, 2009.

[53] Kreth, G., Munkel, C., Langowski, J., Cremer, T., and Cremer, C., "Chromatin structure and chromosome aberrations: modeling of damage induced by isotropic and localized irradiation," *Mutation Research*, vol. 404, no. 1-2, pp. 77–88, 1998.

[54] Langowski, J., "Polymer chain models of DNA and chromatin," *The European Physical Journal E: Soft Matter and Biological Physics*, vol. 19, no. 3, pp. 241–249, 2006.

[55] Langworthy, J. B., "The chord distribution for a right circular cylinder," tech. rep., Naval Research Laboratory, 1988.

[56] Lindborg, L. and Nikjoo, H., "Microdosimetry and radiation quality determinations in radiation protection and radiation therapy," *Radiation Protection Dosimetry*, vol. 143, no. 2-4, pp. 402–408, 2011.

[57] Münkel, C. and Langowski, J., "Chromosome structure predicted by a polymer model," *Physical Review E*, vol. 57, no. 5, pp. 5888–5896, 1998.

[58] Meteopolis, N. and Ulam, S., "The Monte Carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[59] National Iinstitute of Standards and Technology, "Pstar and astar databases for protons and helium ions," March 2012. http://physics.nist.gov/PhysRefData/Star/Text/programs.html.

[60] Nikjoo, H., Charlton, D. E., and Goodhead, D. T., "Monte carlo track structure studies of energy deposition and calculation of initial dsb and rbe," *Advances in Space Research*, vol. 14, no. 10, pp. 161–180, 1994.

[61] Nikjoo, H., Goodhead, D. T., Charlton, D. E., and Paretzke, H. G., "Energy deposition in small cylindrical targets by ultrasoft x-rays," *Physics in Medicine and Biology*, vol. 34, no. 6, pp. 691–705, 1989.

[62] Nikjoo, H., Uehara, S., Wilson, W. E., Hoshi, M., and Goodhead, D. T., "Track structure in radiation biology: theory and applications," *International Journal of Radiation Biology*, vol. 73, no. 4, pp. 355–364, 1998.

[63] Olive, P. L., "The role of dna single and double-strand breaks in cell killing by ionizing radiation," *Radiation Research*, vol. 150, no. 5S, pp. 42–51, 1998.

[64] Ponomarev, A. L., Brenner, D., Hlatky, L. R., and Sachs, R. K., "A polymer, random walk model for the size-distribution of large dna fragments after high linear energy transfer radiation," *Radiation and Environmental Biophysics*, vol. 39, no. 2, pp. 111–120, 2000.

[65] Ponomarev, A. L. and Cucinotta, F. A., "Chromatin loops are responsible for higher counts of small dna fragments induced by high-let radiation, while chromosomal domains do not affect the fragment sizes," *International Journal of Radiation Biology*, vol. 82, no. 2, pp. 293–305, 2006.

[66] Poole, C. M., Cornelius, I., Trapp, J. V., and Langton, C. M., "Technical note: Radiotherapy dose calcualtions using GEANT4 and the Amazon Elastic Compute Cloud," 2011.

[67] Raju, M. R., "Proton radiobiology, radiosurgery and radiotherapy," *Radiation Research*, vol. 67, no. 3, pp. 237–259, 1995.

[68] Rossi, H. H., "Radiation quality," *Radiation Research*, vol. 107, no. 1, pp. 1–10, 1986.

[69] Sachs, R. K., van den Engh, G., Trask, B., Yokota, H., and Hearst, J. E., "A random-walk/giant-loop model for itnerphase chromosomes," *Proceedings of the National Academy of Sciences*, vol. 92, no. 7, pp. 2710–2714, 1995.

[70] Semenenko, V. A., Turner, J. E., and Borak, T. B., "NOREC, a Monte Carlo code for simulating electron tracks in liquid water," *Radiation and Environmental Biophysics*, vol. 42, no. 3, pp. 213–217, 2003.

[71] Turner, J. E., Hamm, R. N., Wright, H. A., Ritchie, R. H., Magee, J. L., Chatterjee, A., and Bolch, W. E., "Studies to link the basic radiation physics and chemistry of liquid water," *International Journal of Radiation Applications and Instrumentation. Part C. Radiation Physics and Chemistry*, vol. 32, no. 3, pp. 503–510, 1988.

[72] Turner, J. E., Magee, J. L., Wright, H. A., Chatterjee, A., Hamm, R. N., and Ritchie, R. H., "Physical and chemical development of electron tracks in liquid water," *Radiation Research*, vol. 96, no. 3, pp. 437–449, 1983.

[73] Uehara, S., Nikjoo, H., and Goodhead, D. T., "Comparison and assessment of electron cross sections for monte carlo track structure codes," *Radiation Research*, no. 2, pp. 202–213, 1999.

[74] van Heesch, D., "Doxygen," February 2012. http://www.stack.nl/ dimitri/-doxygen/.

[75] Villagrasa, C., Francis, Z., and Incerti, S., "Physical models implemented in the geant4-dna extension of the geant-4 toolkit for calculating initial radiation damage at the molecular level," *Radiation Protection Dosimetry*, vol. 143, no. 2-4, pp. 214–218, 2011.

[76] Wang, H., Ma, Y., Pratx, G., and Xing, L., "Toward real-time Monte Carlo simulation using a commercial cloud computing infrastructure," *Physics in Medicine and Biology*, vol. 56, pp. N175–N181, 2011.

[77] Wheeler, R., "Chromatin structure," April 2012. http://en.wikipedia.org/wiki/File:Chromatin_Structures.png/.

[78] Wilson, W. E. and Paretzke, H. G., "Calculation of distributions for energy imparted and ionization by fast protons in nanometer sites," *Radiation Research*, vol. 87, no. 3, pp. 521–537, 1981.

[79] Wright, H. A., Magee, J. L., Hamm, R. N., Chatterjee, A., Turner, J. E., and Klots, C. E., "Calculations of physical and chemical reactions produced in irradiated water containing DNA," tech. rep., Oak Ridge National Laboratory, Lawrence Berkeley Laboratory, January 1985. Symposium on Microdosimetry, Tolouse, France.

[80] Yokota, H., van den Engh, G., Hearst, J. E., Sachs, R. K., and Trask, B. J., "Evidence for the organization of chromatin in megabase pair-sized loops arranged along a random walk path in the human G0/G1 interphase nucleus," *The Journal of Cell Biology*, vol. 130, no. 6, pp. 1239–1249, 1995.