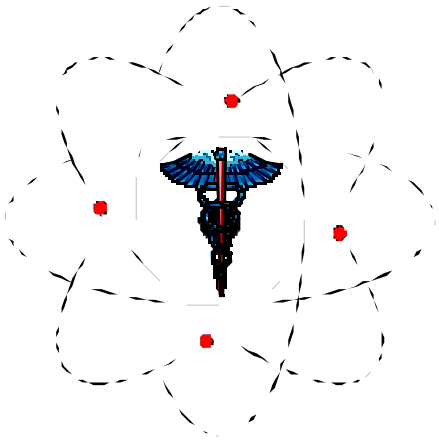
PARAPET



Eidolon: A Monte Carlo Simulator for Multi-ring 3D PET Scanners

Authors: *Habib Zaidi* * *and Christian Morel*

Geneva University Hospital (HUG)
Division of Nuclear Medicine
CH-1211 Geneva 4, Switzerland
email: hzaidi@dmnu-pet5.hcuge.ch
WWW: http://dmnu-pet5.hcuge.ch

September 1999

* Corresponding author

1. Introduction

This report documents the development and implementation aspects of the *Eidolon* Monte Carlo package for simulating cylindrical multi-ring PET tomographs. It is worth to note that most of this document is taken from the deliverable for Task 2.1 of the PARAPET project. The aim is to use the simulator to test different reconstruction algorithms. Although many potential application in scatter modelling and correction have been investigated.

Monte Carlo simulation of three-dimensional (3D) PET data is a very powerful tool to understand performances of 3D positron tomographs [1,2,3] as well as to assess 3D image reconstruction algorithms and their implementations [4]. Since it allows to obtain separate images of unscattered and scattered coincidences, it may help developing and evaluating 3D attenuation and scatter correction techniques [5,6]. In this report, we present an object-oriented, extensible design for a Monte Carlo simulator for 3D positron tomography. The principles of the simulator and mathematical modelling are described in detail. Results from phantom simulation studies including absorption and scattering of the photons in the field-of-view are also presented. Finally, some guidelines regarding installation, compilation and exploitation of the package are provided.

2. Computational Methods

2.1 The random number generator

The use of random numbers is central in all Monte Carlo simulations and the generator is an important part in a Monte Carlo code. The random numbers are used in sampling values from appropriate frequency functions. The Marsaglia algorithm [7] was used in this work for generating uniformly distributed pseudo-random numbers. The sequence of 24 bit pseudo-random numbers has a period of about 2^{144}, and has passed stringent statistical tests for randomness and independence.

2.2 Simulation of PET imaging

Development of a system to simulate realistic PET imaging began with the assumption of a reasonable model. The model assumes a cylindrical array of multicrystal detector modules separated by infinitesimal air gaps and a known spatial distribution of source and scatter phantoms. Annihilation events (pairs of 511 keV photons travelling in opposite directions) are generated uniformly within the source. Actually, photons propagate along nearly collinear paths. Photon acollinearity depends on the momentum of the positronium atom formed by the electron and the positron shortly before the annihilation. Photon acollinearity (about 0.5° FWHM) and positron range effect are not simulated, but could be readily incorporated as quadratic contribution to the resulting spatial resolution. The resulting annihilation photons are transported until they are absorbed in the detector ring or escape without hitting the detector ring. Depending on the choice of the user, annihilation photons hitting a detector can either be detected with 100% efficiency while assuming they deposit all their energy in the detector crystal, or interact with the detector crystal like with any other scattering media. Annihilation photons are paired if the ray connecting the two detectors lies within the field-of-view (FOV) (Figure 1). If two photons originate from the same annihilation and do not undergo scattering within the source, the coincidence is considered as a true coincidence, whereas if one or both photons arising from the same annihilation undergo scattering, the coincidence is considered as a scattered coincidence (Figure 2).

The transport of light produced during scintillation in the crystals and the block-associated electronic chain (photomultipliers, pre-amplifiers and amplifiers) were not simulated. Random coincidences are not simulated but could be analytically modelled after simulation using the singles rate. Interaction within scatter or detector objects can be switched on and off. In case interaction within detector objects is switched off, any photon impinging on a detector is assumed to deposit all its energy in the detector crystal. When it is switched on, photon pairs are recorded once two photons resulting from one annihilation event have passed the energy window set for discrimination. Radial samples are assumed to be equidistant, although ring curvature can be taken into account for sampling.

2.3 Photon pathlength

The survival probability, i.e. the probability of a photon to not interact along a pathlength d is given by:

$$p(d) = \exp\left[-\int_0^d \mu(x) \cdot dx\right] \tag{1}$$

where $\mu(x)$ is the attenuation coefficient at position x, a function of the photon energy and of the propagating medium. Typical minimum survival probabilities at 511 keV are 0.15 for head scans and 0.003 for body scans. The survival probability given by equation 1 is also referred to as the *narrow-beam attenuation*. For photons, the range (maximum distance travelled before interaction) is infinite. To force interaction along a given distance $d_{\max}$, photon pathlength d can be sampled from:

$$d = \frac{1}{\mu} \cdot \ln\left[1 - \mathrm{R} \cdot \left(1 - e^{-\mu d_{\max}}\right)\right] \tag{2}$$

Where R is a random number uniformly distributed between 0 and 1.

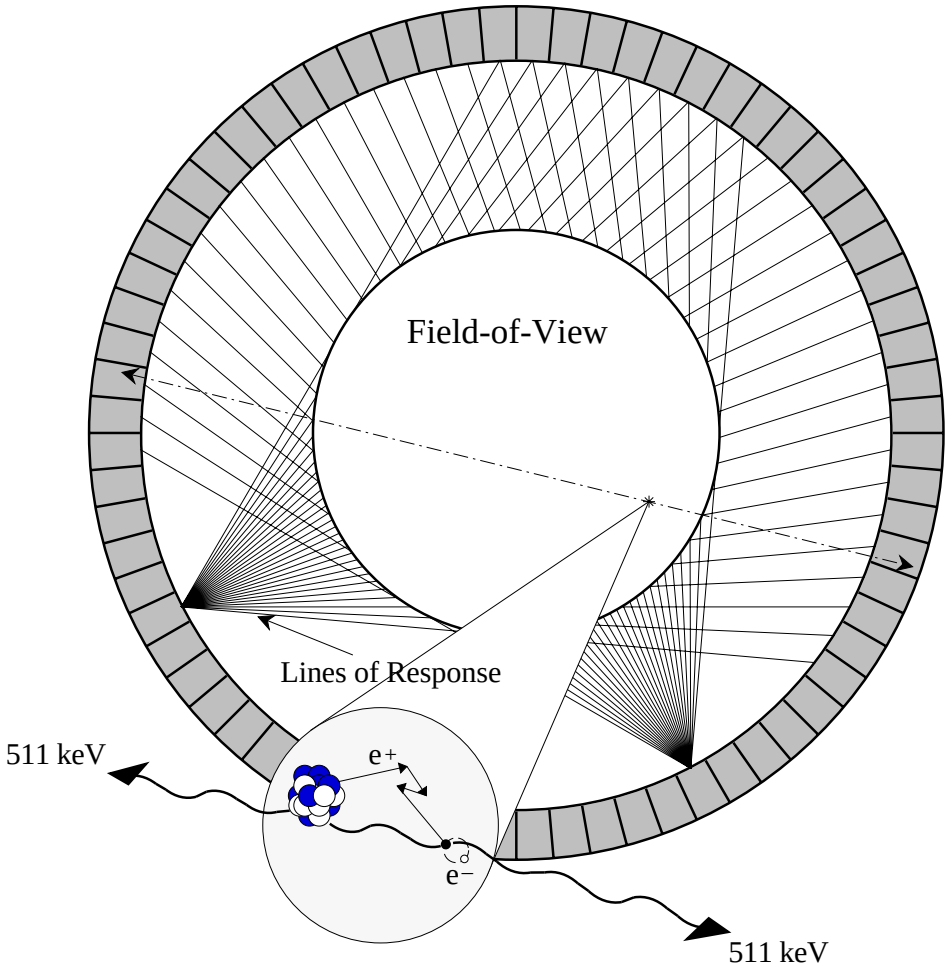


Figure 1. Born from the decay of the radio-isotope, a positron travels a few millimetres only to be annihilated by a nearby atomic electron, producing two 511 keV photons emitted in opposite directions. A positron emission tomograph contains a set of detectors usually arranged in adjacent rings surrounding the field-of-view (FOV). Pairs of annihilation photons are detected in coincidence. The size of the FOV is defined by the number of opposite detectors in coincidence.

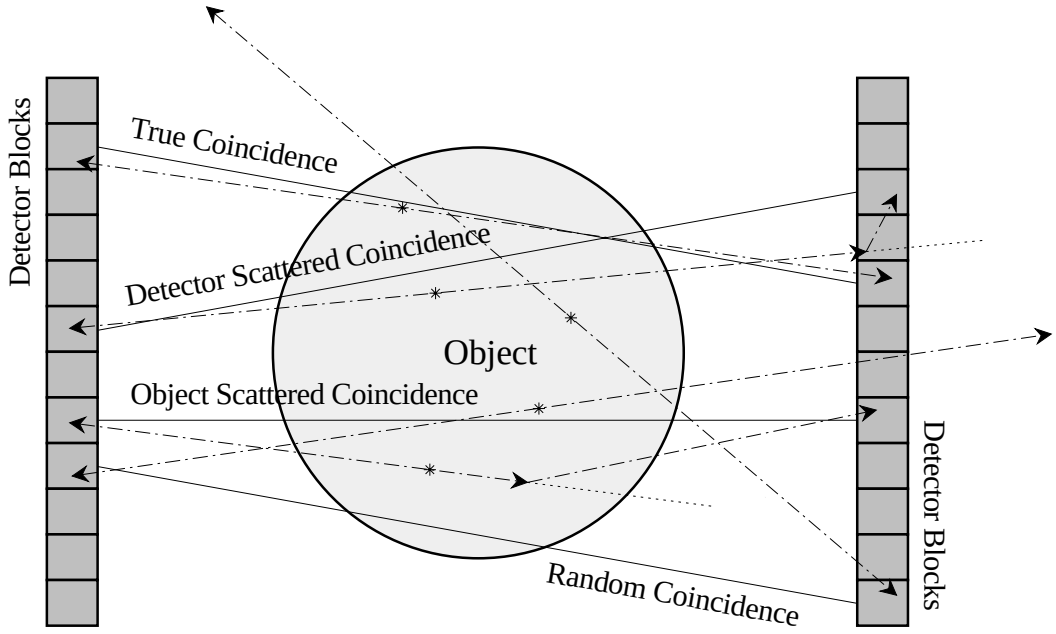


Figure 2. Schematic of a simulation model for 3D positron tomography. Detection of true, random, object scattered and detector scattered coincidences are indicated along the corresponding LORs. Dash lines indicate photon paths.

2.4 Interaction processes

An interaction occurs if the photon has not escaped the current volume (phantom or detectors) after travelling a pathlength d sampled according to an attenuation distribution function $\mu exp[-\mu d]$. Cross-section data for photoelectric interactions, incoherent scattering (Compton scattering) and coherent scattering (Rayleigh scattering) are calculated for the volume material specifications from parametrizations. The type of interaction at a given photon energy is selected according to the relative probabilities of each type of processes which are proportional to the cross-sections. Thus, the probability to have an interaction is obtained from an attenuation distribution function with

$$\mu = \mu_{photo} + \mu_{incoh} + \mu_{coh} \tag{3}$$

where μ is the total linear attenuation coefficient and μ_{photo}, μ_{incoh} and μ_{coh} are the photoelectric, Compton and Rayleigh linear attenuation coefficients, respectively.

Let R be a random number uniformly sampled between 0 and 1. Then, the type of interaction will be: (1) photoelectric absorption if $R \leq \mu_{photo}/\mu$; (2) incoherent scattering if $\mu_{photo}/\mu < R \leq (\mu_{photo}+\mu_{incoh})/\mu$; (3) else coherent scattering. In the case of photoelectric absorption, the total photon energy is transferred to an atomic electron and the random walk of the photon is terminated. In an incoherent scattering, a fraction of the photon energy is transferred to a free electron. Energy of the Compton scattered photons are sampled according to the Klein-Nishina distribution function. The direction of the scattered photon is then updated so as to keep momentum of the system. In coherent scattering, photon momentum is transferred to an atom, thus resulting only in a change of the direction of the scattered photon with a negligible energy loss for the scattered photon. Coherent scattering of a photon is generated using the random number composition and rejection techniques to sample the momentum of the scattered photon and the scattering angle according to the form factor distribution [8].

It is common to neglect coherent scattering in Monte Carlo simulations of photon transport because of its low contribution to the total cross-section at 511 keV. For example, in water, the dominant effect is Compton scattering. The coherent contribution to the total cross-section is less than 1% for energies above 250 keV. However, this contribution is in the order of 7% for high-Z materials like BGO. In our code, efforts have been made to treat the coherent scattering process adequately.

2.5. Detector block energy and spatial resolutions

Stochastic variations are introduced when the photon deposits its energy in the crystal. A finite energy resolution $\Delta E/E$ is simulated by convolving the detected energy with a Gaussian function. A knowledge of the energy resolution of the detector block is required for at least one specific energy. The energy resolution is then proportional to $1/\sqrt{E}$. The energy resolution of the detector blocks depends on the PET tomograph

being simulated. For the detector blocks of the ECAT 953B positron tomograph, it is 23% at 511 keV [9]. Photon pairs are recorded in the sinogram once two photons resulting from one annihilation event have passed the energy window set for discrimination. To take into account the intrinsic spatial resolution of the detectors, a convolution of the apparent coordinates of lines of responses can also be performed if required. Like energy discrimination thresholds, other parameters of the scanner such as energy and spatial resolutions of the detector blocks can be tailored through a graphical user interface or text files.

The position blurring step then calculates the mean detection coordinates (*X*,*Y*,*Z*) of each incident photon. This is done by computing the centroid of all interaction vertices, each weighted by the ratio of its individual energy to the total energy. A lower cut-off energy is set to model the energy acceptance window used in PET scanners to reject scattered radiation. When a photon falls below this user selectable energy value, its history is terminated. The mean *X* and *Y* coordinates of each photon are smeared to account for spatial resolution and position miscoding in the detector block. For each detected coincidence, a convolution with a Gaussian having an appropriate width (σ_{gauss}) is performed:

$$\sigma_{gauss} = \sqrt{\sigma_{range}^{\ 2} + \sigma_{non-col}^{\ 2} + \sigma_{block}^{\ 2}} \qquad (2)$$

where:

σ_{range}: accounts for the loss of resolution due to the positron range;
$\sigma_{non\text{-}col}$: accounts for the loss of resolution due to the annihilation photon non-colinearity effect. This value is estimated in the centre, where this effect is maximum, and assumed constant over the whole FOV.
σ_{block}: accounts for the loss of resolution due to the block blurring effect.

2.6 Software description

The Monte Carlo simulator, *Eidolon*, was written in Objective-C and runs on any platform using appropriate GNU compiler and objective-c associated libraries. Objective-C is an object oriented computer programming language. It is a superset of ANSI C and provides classes and message passing similar to Smalltalk. The problem was modelled by a set of general classes permitting future extension. The simulator is structured to be a convenient test bench to develop C modules that can be integrated into conventional C programs. In order to ease the job of incrementally adding capabilities, a modular design featuring dynamically loadable program elements was adopted. The basic building block is a *model element class* which allows elements to be browsed, inspected, adjusted, created and destroyed through a graphical inspector. A controller object oversees the simulation process. *Eidolon*, like any object-oriented programs, uses objects built from a variety of classes. In Objective-C, objects are defined by their class. Another characteristic of Objective-C is that many decisions are postponed from compile time to run time. Therefore, the Objective-C language depends on a run time system to execute compiled code.

The application is organised in a convenient way by dividing the program into different compartments called *subprojects*. Each subproject could be compiled and debugged separately. The software is structured as follows: the graphical interface (not used on Parsytec-CC system), objective-C classes, categories and protocols, and procedural programming C routines. The graphical interface as well as the link between graphical objects and objective-C code was designed using the NEXSTEP Interface Builder (nib) development tool.

Class definitions are additive; each new class is based on another class through which it inherits methods and instance variables. The new class simply adds to or modifies what it inherits. Inheritance links all classes in a hierarchical tree with a single class, the Object class, as its root. *Eidolon* takes advantage of this property by organising a hierarchical tree such that methods used by different objects need not be implemented twice. Basic classes are used to handle gamma pair creation, photon interaction and detection methods as well as other matrix handling routines, 3D shape definitions and geometrical calculations. Categories are extensively used to add methods to an existing class. This simplifies the management of large classes and allows to beneficiate of incremental compilation. Protocols are also used to capture similarities among classes that are not hierarchically related. A protocol is a list of method declarations, not associated with a particular class definition. Thus, any class or many classes can implement them by adopting the protocol.

In the version implemented under NEXSTEP environment, *Eidolon* exploits the dynamic loading property of objective-C, thus classes and categories are loaded while the program is running. The new code is incorporated into the program and treated identically to classes and categories loaded at the beginning. For example, objects adjustable by the user through the graphical interface for a particular simulation study, such as source, scatterer or scanner parameters are dynamically loaded into the kernel at run-time. These are

referred to as dynamically loadable modules or *bundles*. Unfortunately, dynamic loading is not yet supported on gcc/powerpc-aix4.1.4. Thus bundles are handled as other classes in the version running on the Parsytec system.

FORTRAN routines from the CERN GEANT code [8] calculating cross-section data and simulating different interaction processes were translated to C. Other C routines for generating random numbers using Marsaglia and Sobol algorithms were also included. The reference image as well as sinogram data sets are saved in CTI Matrix 6 format using C routines. This comprises matrix file utilities including headers used for reconstruction as well as sinogram and image handling tools. These data could also be retrieved from the disk and inspected through the graphical inspector. All the above described modules were integrated into the application with minor modifications.

Besides the fidelity of the simulated PET data and the ease with which software models can be prepared, the remaining important consideration in performing simulation of PET imaging systems is the computational time required to generate adequate data sets. The time needed to perform a simulation study depends on the complexity of the chosen sets of source, scatter and detector objects, and on selected interactions. Timing of a single MPC 604 processor was carried out against the HP9000/712 station. The same simulation benchmark was used in the timing in which a line source was simulated in the centre of a water-filled cylindrical phantom for the ECAT-953B PET scanner (CTI PET Systems Inc., Knoxville, TN 37933) operated in 3D mode (16 rings of 384 detectors each with a ring radius of 38 cm) in scatter-free imaging, and when considering attenuation and scattering within the phantom and detector blocks. The resulting computing time in minutes and the ratios between 4, 8, 12 and one single node to track 10 million annihilation pair photon histories are given in Table 1. Time required by slave processors to write data sets on disk is not included in the time quoted in table 1. The time required to perform the I/O operations and summation of data sets is negligible compared to that required for large simulation studies. A linear scaling of the computing time with the number of processors has been achieved.

Table 1. Comparison of the computing times for different simulation studies. Computing times required to track 10 million annihilation pairs on both the HP 9000 712/60 workstation and the Parsytec CC system having 1, 4, 8 and 12 computing nodes are given in minutes. The speed-up achieved is also reported.

	HP 9000 712/60	Parsytec CC1	Parsytec CC4	Parsytec CC8	Parsytec CC12
Scatter-free imaging	72.33	25.88	6.38 **(4)**	3.7 **(7)**	2.2 **(11.2)**
Detector-scatter imaging	1425.65	436.55	110.26 **(4)**	54.6 **(8.2)**	36.7 **(11.9)**
Object-scatter imaging	2256.2	493.35	122.1 **(4)**	63.97 **(7.7)**	42.5 **(11.6)**
Full-scatter imaging	2540.45	657	170.5 **(3.9)**	83.66 **(7.8)**	55.7 **(11.8)**

There is no theoretical limit on the number of processors to be used. Upgrade of the system is reasonable to obtain better performance. The techniques developed in this work are also suitable for the implementation of the Monte Carlo code on other parallel computer systems.

2.7 User Interface

The program is controlled by a dedicated graphical user interface. The user must select scanner dimensions, choose a set of simple geometric shapes for the source and for the scattering media, and define the composition of the scatterer. Source shapes may be multiples and disjoints and may comprise cold inserts, whereas the scattering shape must be unique and convex and must surround all the source shapes.

The reference image and the sinograms may be viewed as they are generated. All simulation parameters can be browsed, inspected, adjusted, created and destroyed through the graphical inspector and can be entered directly on the keyboard or by using the appropriate sliders in the graphical inspector. In the version implemented on the Parsytec-CC system, the graphical interface is no longer used and simulation parameters are introduced via dedicated files.

3. Monte Carlo Code Features

The simulation program could be used to obtain information on the different processes occurring within the phantom and the detectors. For example, energy pulse-height distributions, point-spread functions and the scatter fractions could be obtained. The scattered coincidences in the energy-pulse-height distributions could be distinguished according to the number of successive scatterings (coherent and incoherent) in the phantom. The scatter fraction is defined as the ratio between the number of events in the image which comes from photons scattered in the phantom and the number of events coming from primary (unscattered) photons. The scatter fraction is of great importance for quantitative estimation of the scattering contribution. The scatter-to-total fraction is defined as the ratio between the number of scattered photons and the total number of photons (scattered and unscattered).

The simulator is an efficient tool that can be used to generate data sets in a controllable manner in order to assess different reconstruction algorithms. As the 'best' algorithm can only be selected with respect to a certain task, different ‘basic’ performance measures can be used. Image degrading effects are illustrated using simulated projections of the digitised 3D Hoffman brain phantom (Hoffman *et al* 1990). A slice of this phantom is shown in Figure 3. The ratio between the activity in white, grey matter and ventricles was chosen as 1:4:0, respectively.

The projections of this phantom at different levels of fidelity are generated. The strengths of the image degrading factors are characteristic of a ^{18}F-FDG brain study and an energy window of [380-850] keV. Figure 4.E-H shows the effects of different aspects of image degradation on FBP reconstructions. The loss of resolution caused by detector blurring (FWHM = 4 mm) on projection data and FBP reconstructions is shown in Figure. 4.B and 4.F.

Figure 3. Transaxial slice of the digital 3D Hoffman brain phantom.

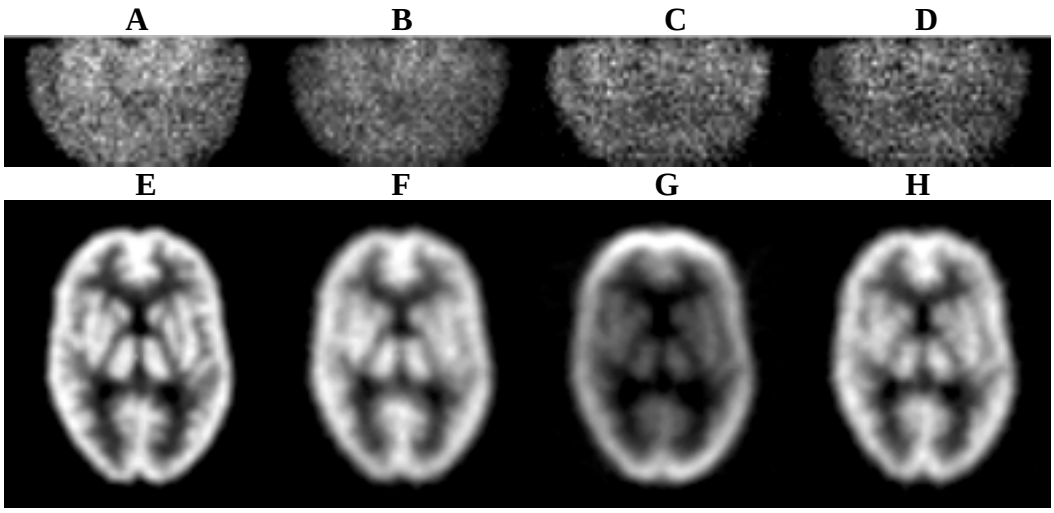


Figure 4. Simulation of the Hoffman brain phantom in different imaging situations showing 2D projections (top row) and their filtered backprojection reconstructions with the reprojection algorithm (bottom row). In case **A** and **E** a 2D projection and reconstructed image neglecting attenuation, scatter and detector blurring. In **B** and **F** effects of detector blurring are included, while in **C** and **G**, effects of detector blurring, attenuation and scatter are included in the simulation. Finally, in **D** and **H** effects of detector blurring, attenuation and scatter are included and appropriate corrections for attenuation and scatter applied.

In a real measurement, scatter components in the detectors and in the FOV are indistinguishable from other secondary effects. Monte Carlo simulation of energy distributions provide insight into the scattering processes arising in 3D positron tomography. The detected events can thus be distinguished and separated in an efficient way into scattered and unscattered coincidences. The sinograms resulting from the simulation of the Hoffman phantom have been separated into unscattered and scattered coincidences and are illustrated in Figure 5.

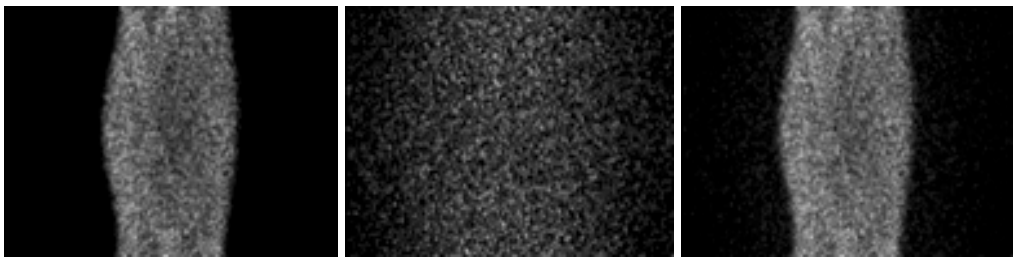


Figure 5. Representative simulated sinograms of the Hoffman brain phantom taking into account the effects of detector blurring, attenuation and scatter showing only unscattered events (left), scattered events (middle) and total detected events (right).

Eidolon was used to obtain unscattered and scattered energy distributions of coincident detections (Figure 6), as well as to study line-spread functions and scatter fractions for the ECAT-953B PET scanner (Figure 7). Figure 6 shows an energy-pulse height distribution obtained by simulation of a line source in the centre of a 20 cm diameter water-filled cylindrical phantom. An energy resolution of 23% FWHM was assumed since this is typical value for BGO block detectors. Ten million annihilations were tracked. The scattered coincidences in the energy-pulse height distribution were itemised according to the number of successive scatterings in the phantom. It is clear from Figure 6 that some scattered photons were not rejected by the 250-750 keV energy window discrimination following from the limited energy resolution of the detectors.

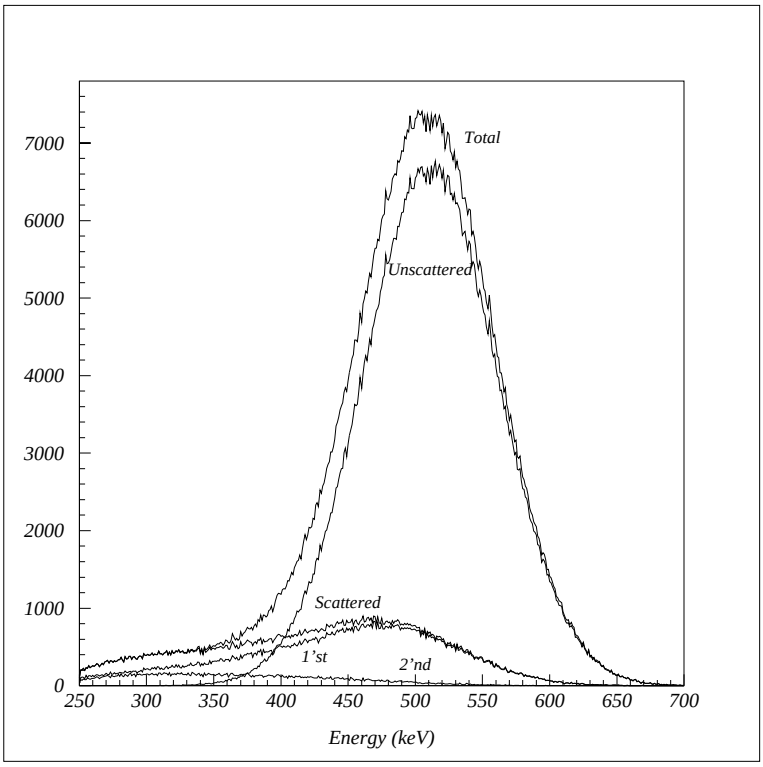


Figure 6. Energy distributions of coincident detections resulting from the simulation of a line source placed in the centre of a 20 cm diameter cylinder filled with water. Photons impinging on a detector are assumed to deposit all their energy in the detector crystal. Energy resolution is proportional to the inverse square root of the deposited energy and is simulated by convolving the deposited energy with a Gaussian function whose FWHM is 23% for 511 keV photons. Both photons of a coincident detection have to pass an energy window set between 250 and 850 keV. Distributions of photons resulting from exactly one or two successive Compton scatterings in the FOV are shown.

The lower energy threshold (LET) can be easily changed and its effect on the scatter components studied in an effective way. Figure 7 shows transaxial profiles in the projection space for a simulated line source in a 20 cm diameter water-filled cylinder as a function of the LET. In each case, the cylinder is centred on the origin of the graph. It is clear that the boundary (at -10 cm, +10 cm) of the object has no influence on the profile. However, the LET set-up greatly influences the amount of scatter in the projection data.

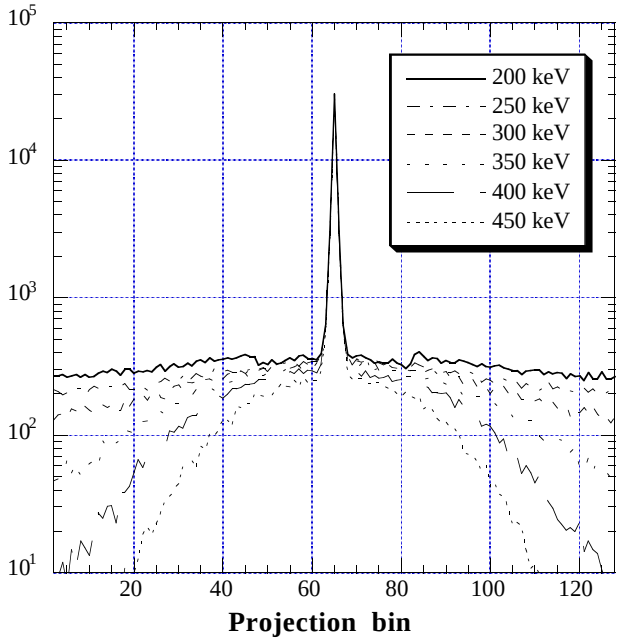


Figure 7. Sum of one-dimensional transaxial projections resulting from the simulation of a line source placed in a 20 cm diameter cylinder filled with water as a function of the lower energy discrimination threshold. This illustrates the compromise that should be attained between increasing the lower energy threshold to reduce scattered events and the variance in the reconstructed images resulting from limited statistics.

Table 2 shows scatter fractions obtained with *Eidolon* for three different radial positions of a line source placed in a 20 cm diameter cylinder filled with water. Ten million annihilation events were generated for each radial position of the line source. To allow comparison with measurements, detector scatters were not counted, since they are usually not differentiated from the trues in experimental data. The scatter fraction determined with the line source in the centre of the phantom is 0.37. In the same detection conditions, a real measurement of this scatter fraction gave 0.42 [9]; it was estimated to be 0.46 using a different Monte Carlo simulator [3]. The discrepancy between our value and the one given by Michel *et al* [3] results from the fact that we did not consider those photons scattered only within the detector crystals as part of the scatter fraction. Consistent with the real measurements, our results show that the scatter fraction decreases when the source moves off-axis.

Table 2. Comparison between Monte Carlo estimations and real measurements of the scatter fraction for different radial positions of a line source placed in a 20 cm diameter cylinder filled with water. Same detection conditions as in Figure 6 apply, except that interaction within detector objects was switched on and energy window was set between 380 and 850 keV for comparison with published data.

Radial position [mm]	*Eidolon*	Spinks *et al* [9]	Michel *et al* [3]
0	0.37	0.42	0.46
40	0.36	0.40	
80	0.29	0.30	

4. Phantom Simulations

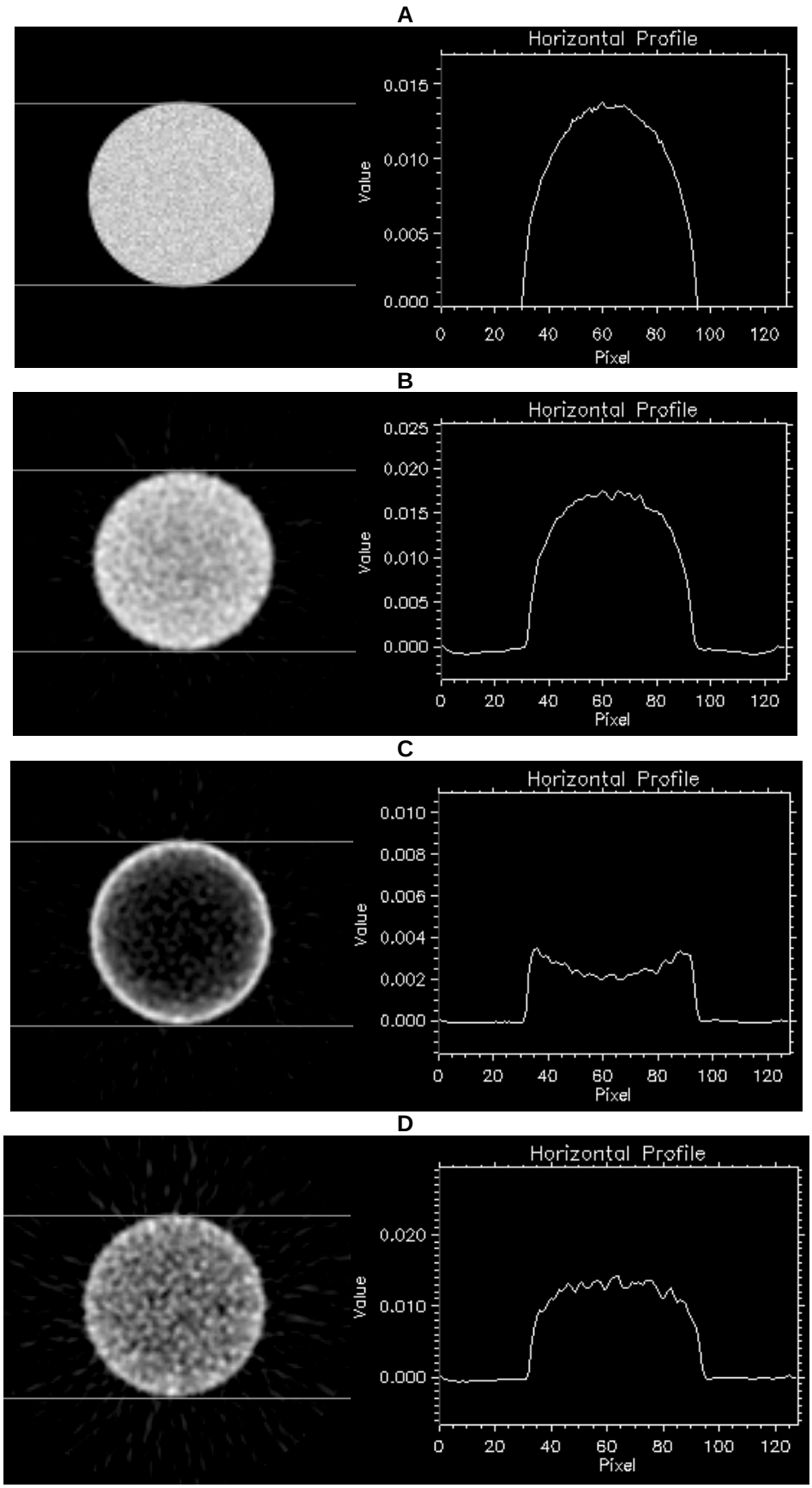


Figure 8. Central slice and projections of a uniform cylindrical phantom. **A.** Reference image. **B.** Image reconstructed from a data set generated without scatter simulation. **C.** Image reconstructed before attenuation corrections from a data set generated with scatter simulation. **D.** Image reconstructed after applying attenuation corrections to the same data set as in **C**.

A phantom collection was generated using simple source geometries and distributions for an ideal scanner (without noise sources such as scattered coincidences). The first step in the present investigation, was the accurate simulation of photon transport in both the phantom and the detector blocks. In order to validate the model, simple phantoms (line sources, cylindrical sources) and more elaborated phantoms specifically designed to study the contribution of scattered coincidences (Utah phantom) are used for the simulation study. Scans were generated for the 16 ring scanner ECAT 953B with 384 detectors/ring, and operated in 3D mode. A 3D scan consists of 256 sinograms, each of which is a matrix of 128 radial samples (sampling distance 3.1 mm) by 96 angular samples. In this study, the radial samples were assumed to be equidistant, although ring curvature in the sample spacing could be included.

Eidolon was used to simulate a uniform water-filled cylindrical phantom (Figure 8) and the Utah phantom which was designed with a high degree of inhomogeneity both transaxially and axially in order to compare and test scatter correction techniques in 3D PET [10]. Phantom data sets for the ECAT-953B PET scanner were generated both with and without scatter simulation. The outer compartment of the phantom which is generally used to provide activity from outside the FOV, was left empty. Before reconstruction, attenuation correction was applied to those data sets generated with scatter simulation; attenuation correction files were created by forward projecting the 3D density map estimated with a constant linear attenuation coefficient of 0.096 cm^{-1}. No scatter correction was applied to the sets generated with scatter simulation. Generated data sets were reconstructed using two different exact 3D reconstruction algorithms implemented on a high-performance parallel platform [11].

In Figure 8, projections of the central slice of a uniform phantom reconstructed before and after applying attenuation corrections to the data set generated with scatter simulation are shown.

Scatter fractions determined using simulation of the Utah phantom when varying the lower energy discrimination threshold are shown in Figure 9. As the threshold is lowered, the scatter fraction increases steadily.

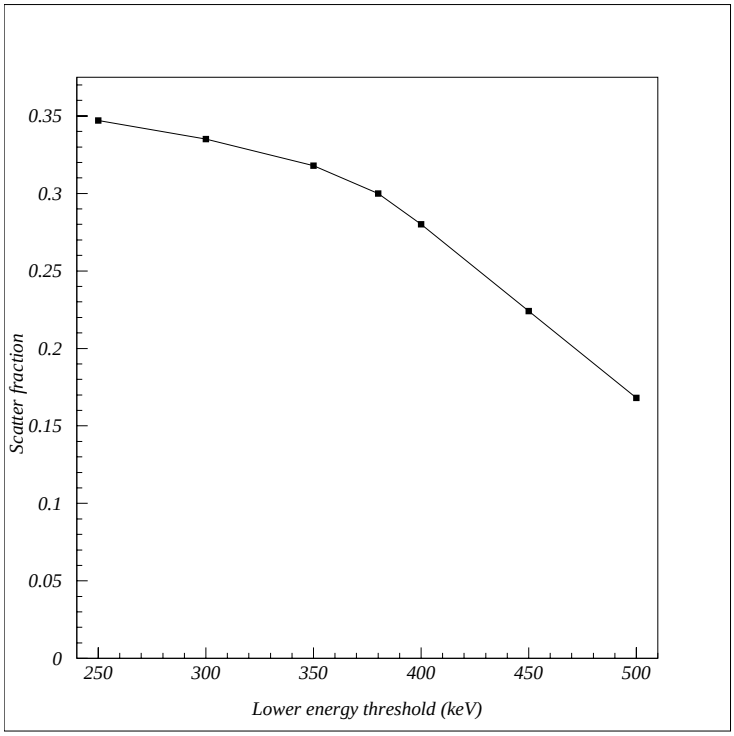


Figure 9. Variation of scatter fraction determined for the Utah phantom as a function of the lower energy discrimination threshold.

5. Shape-based versus Voxel-based phantoms

Object modelling is fundamental for performing photon and electron transport efficiently by means of a Monte Carlo method. It consists of a description of the geometry and material characteristics for an object.[1] The material characteristics of interest include density and energy-dependent cross-sections. The modelling includes simple geometry (SG), shape-based (SB), and voxel-based (VB) approaches. The three approaches use a piece wise uniform distribution of object characteristics to model an object. With the SG

model, an object is composed of a simple combination of primitives such as cylinders and spheres. The SB approach represents the boundaries of shapes by mathematical equations. Regular shapes such as spheres, cylinders, parallelepipeds, etc. have been used to approximate irregularly-shaped regions. The VB approach discretizes an object into tiny cubes (voxels) with uniform characteristics. An object is thus represented by a union of voxels of the same size. As an improvement to the mathematical anthropomorphic phantoms, a new family of phantoms was constructed from CT and MRI data. The human phantoms present advantages towards the location and shape of the organs, in particular the hard bone and bone marrow.

Computerised anthropomorphic phantoms can either be defined by mathematical (analytical) functions, or digital (voxel-based) volume arrays. Since high statistics are necessary to model imaging simulations, the computing time needed to track photons becomes of paramount importance. The software phantoms modelled in these imaging simulations have sometimes been limited to simple point, rod and slab shapes of sources and attenuating media. Such simple geometries are useful in studying basic issues of scatter and attenuation, but clinically realistic distributions cannot be adequately evaluated by such simple geometries. The intricate protuberances and convolutions of human internal structures are important in evaluating imaging techniques. As the resolution of imaging equipment improves, it is essential to enhance our computer models. In order to make 3-dimensional anatomical data suitable for use as input to our simulation studies, we must be able to delineate the surfaces and internal volumes which define the various structures of the body. These segmented volumes can then be indexed to activity distributions or other physical characteristics (density or elemental composition).

A physical brain phantom has also been developed to simulate the activity distributions found in the human brain in the cerebral blood flow and metabolism studies currently employed in PET [16]. The phantom utilises thin layers of Lucite to provide apparent relative concentrations of 4, 1 and 0 for gray matter, white matter and ventricles, respectively, in the brain. A clinically realistic source distribution simulating brain imaging was created in digital format. Zubal [15] developed a typical anthropomorphic VB adult phantom by manual segmentation of CT transverse slices of a living human male performed by medical experts. A computerised 3D volume array modelling all major internal structures of the body was then created. Each voxel of the volume contains an index number designating it as belonging to a given organ or internal structure. These indexes can then be used to assign a value, corresponding to, e.g. density or activity. The Zubal phantom should be accessed as a 128*x*128*x*243 one-byte array. There are a total of 243 slices in this volume with each slice having a dimension of 128*x*128 bytes. In the original color slices the organ numbers are offset from 0 by 63 so the values for skin etc. start at 64 through 126. The 0 for outside of phantom remains fixed. The organ numbers have all been shifted down for the composite phantom, and expanded to the list actually shown in table 3 [15].

The current version of the program is capable of modelling photon transport through media in which the material properties vary in three dimensions. Both SB and VB models are supported within the simulator. A detailed description of the way to specify the source and scattering medium for the SB model is given in section 8 (**How to configure the simulator**). The advantages of the VB model is to allow some calculations requiring the use of more accurate representations of individuals based on volumetric scans, such as CT, MRI to be made at the expense of the increased computing time. Ray-tracing approaches based on an implementation of Siddon's algorithm are used to compute the length of intersection of each ray with every voxel. By considering pixel boundaries as the intersection of orthogonal sets of equally spaced planes, rather than independently, the computational complexity is greatly reduced (figure 10). The points of intersection of a line with a set of parallel planes are particularly simple to determine. Once one such point has been found, all the others may be found by recursion. A detailed description of the algorithm may be found elsewhere [17].

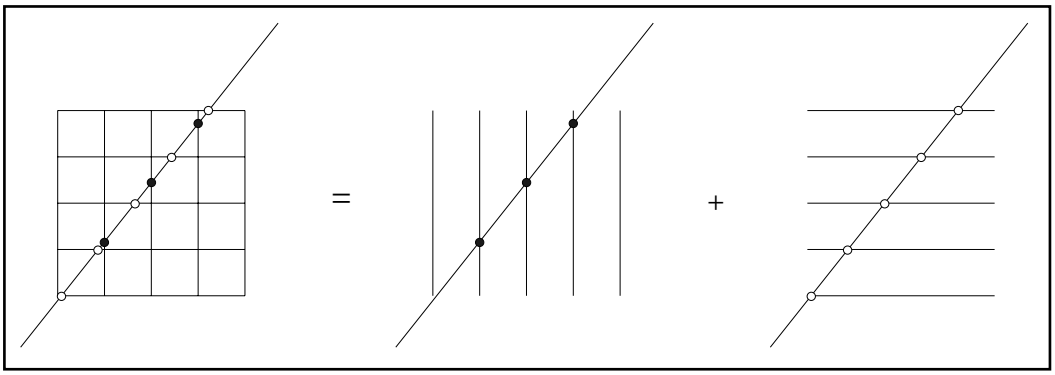


Figure 10. The lengths of intersection of a line with an array of pixels may be computed independently for every pixel (*left*), or, more efficiently, by considering pixel boundaries as the intersection of orthogonal sets of equally spaced planes (*middle* and *right*).

Table 3. Classification or organ codes in the voxel-based simulator.

organ id	organ code
0	outside phantom
1	skin
2	brain
3	spinal cord
4	skull
5	spine
6	rib cage & sternum
7	pelvis
8	long bones
9	skeletal muscle
10	lungs
11	heart
12	liver
13	gall bladder
14	kidney
15	pharynx
16	oesophagus
17	stomach
18	small bowel
19	colon
20	pancreas
21	adrenals
22	fat
23	blood pool
24	gas (bowel)
25	fluid (bowel)
26	bone marrow
27	lymph nodes
28	thyroid
29	trachea
30	cartilage
31	spleen
32	urine
33	feces
34	testes
35	prostate
37	rectum
39	diaphragm
40	bladder
63	lesion
70	dens of axis
71	jaw bone
74	lacrimal glands
75	spinal canal
76	hard palate
77	cerebellum
78	tongue
85	medulla oblongota
91	pons
99	uncus(ear bones)
104	sinuses/mouth cavity
106	optic nerve
113	cerebral falx
119	eye
121	lens
122	cerebral aquaduct
125	teeth

6. General Considerations Regarding Portability and Parallel Monte Carlo

Although variance reduction techniques have been developed to reduce computation time, the main drawback of the Monte Carlo method is that it is extremely time-consuming. To obtain good statistics requires to track hundreds of millions of particles. Consequently, a large amount of CPU time (weeks or even months) may be required to obtain useful simulated data sets. With the development of parallel-processing computers, researchers have turned their efforts towards the parallelisation of Monte Carlo codes and among all simulation techniques of physical processes, the Monte Carlo method is probably the most suitable one for parallel computing.

GNU CC is available from the Free Software Foundation and comes with an Objective-C compiler since version 2.7.1. The current distribution of GNU CC (version 2.8.1) includes an Objective-C compiler and runtime library. It also includes the *Object class* (super class in the hierarchical tree) and relevant libraries. Porting the simulator on any architecture/operating system is thus possible. The simulator was successfully ported on HP, Sun SPARC stations including Fujitsu parallel system, as well as the Parsytec-CC system (AIX.4.1) for use on this parallel architecture.

Most serial Monte Carlo codes are readily adaptable to a parallel environment. Variance reduction techniques must still be tailored to the specific problem. Care must also be taken to ensure reproducible results. The developer must assume that calculations on different processors are independent. Event parallelism is the most straightforward approach and is the most suitable here. It involves running independent simulations concurrently. The independence between simulations is utilised for parallelisation. The basic idea in implementing Monte Carlo in parallel is to generate data sets on multiple processors, so that the same simulation is performed with different seeds on different nodes of the parallel platform at once. Each individual processor runs its own random walks with different initial configurations. One designated processor, say processor 0 (entry node on the Parsytec-CC system), operates as a controller which supplies the other processors with the parameters that govern the simulation processes. Besides running its own sequence, the controller gathers and adds data from the other processors at the end of the whole simulation.

7. Future Prospects

The simulator is a powerful tool to investigate the quantitative accuracy of different reconstruction algorithms and scatter correction methods in 3D PET. A phantom collection has already been generated using the simulator described in this report. Our partners within the PARAPET project are pursuing the evaluation of reconstruction algorithms. The simulator has also been used by one of the authors (HZ) to implement a Monte Carlo-based scatter correction technique and to assess the accuracy of published techniques and new methods under development.

If you do end up using our Monte Carlo simulator and/or our phantoms, we would appreciate referencing one of the publications that describe its development and implementation:

• H. Zaidi, AK. Hermann Scheurer, and C. Morel, "An Object-Oriented Monte Carlo Simulator for 3D Positron Tomographs", Comput. Meth. Prog Biomed., **57:** 133-145 (1999)

• H. Zaidi, C. Labbé, and C. Morel, "Implementation of a Monte Carlo simulation environment for fully 3D PET on a high-performance parallel platform", Parallel Computing, **24**: 1523-1536 (1998)

• H. Zaidi, AK. Hermann Scheurer, and C. Morel. "Development of an Object-Oriented Monte Carlo simulator for 3D Positron Tomography". Proceedings of the International Meeting on Fully Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine (3D'97), Pittsburgh, USA 25-28 June 1997, pp 176-179.

• H. Zaidi, C. Labbé, and C. Morel. "Improvement of the performance and accuracy of PET Monte Carlo simulations". SPIE's International Symposium on Medical imaging 1999, San Diego, USA, Vol. **3659**, edited by J. M. Boone and J. T. Dobbins (1999), pp 582-593.

if you have any questions, please contact me.

8. Software Implementation

This section provides a brief description of the *Eidolon* source code. It does not explain Objective-C, NEXTSTEP/GNUSTEP, the class libraries, or how to use the development tools. Please refer to the NS documentation for excellent explanations about objects, classes, methods, categories, protocols, *Application*, *View*, *List*, *Browser*, *Interface Builder*, *Project Manager*, *.nib* files, subprojects, dynamically loadable modules, the debugger, etc. (*http://www.apple.com/enterprise/*, h*ttp://www.NMR.EMBL-Heidelberg.DE/GNUstep/*). For information on Objective-C, a tutorial is available at (*http://developer.apple.com/techpubs/macosx/ObjectiveC/index.html*).

In the description below, **Class** names will be indicated in **bold** style, *Method* names in *italic* style and ***Object*** names in ***bold-italic*** style. `Filenames` and more generally file content citations will be written in `Courier` fonts.

Functional diagram

The functional diagram of the simulator is illustrated below (figure 11).

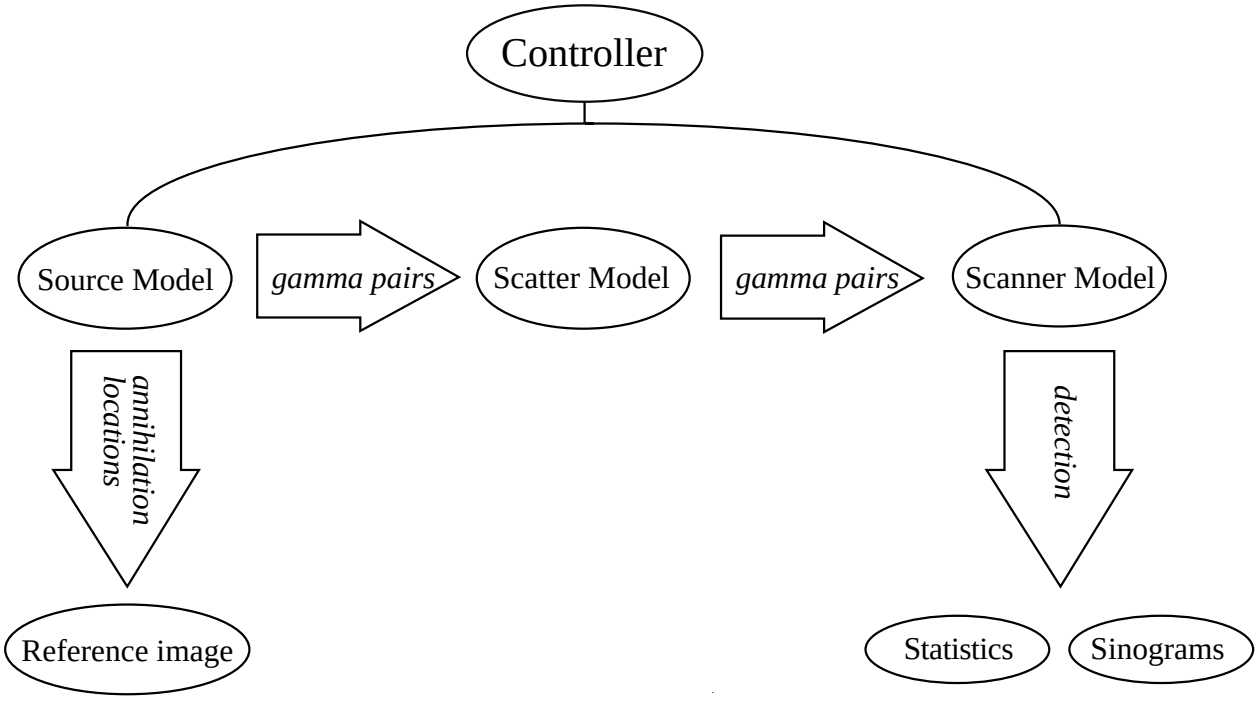


Figure 11. Functional diagram of the Monte Carlo simulator.

Class structure

In the class structure diagram below, class names in bold style indicate classes defined in *Eidolon*. The rest are part of the class library of NEXTSTEP/GNU C compiler. The topmost level, **Object**, is on the left. Indentation denotes a subclass relationship. For example, **SourceInspector** and **ScattererInspector** are both subclasses of **ShapeInspector**, which is a subclass of **PETInspector**, a subclass of **Object**. (In GNUSTEP, the **Object** class is a subclass of **NSObject**).

Object
- **Controller**
- Responder
  - View
    - **ColorTwoDView**
      - **ClickTwoDView**
- **Shape3D**
  - **Source**

Scatterer
Point3D
Direction3D
Matrix4
Photon
GammaPair
PRNGenerator
Sinogram
Image
PixelVolume
ScannerModel
ShapeBasedModel
SourceModel
ScatterModel
PETUnitsManager
PETInspector
ImageInspector
PixelVolumeInspector
SinogramInspector
ScannerInspector
ShapeInspector
SourceInspector
ScattererInspector
Timer

.nib files

`Eidolon.nib` ~ the main application nib file, with the Model Browser window, an Inspector window, the Status window, and the instantiation of the **Controller**.

`InfoPanel.nib` ~ the info panel**.**

`ShapeInspector.nib` ~ the panel for a **ShapeInspector**. Contains:
- shape selection radio buttons;
- data fields of source and scatter object sizes and center locations, and sliders;
- tilt knobs and data fields.

Other `.nib` files associated to dynamically loadable modules (bundles) are listed below.

Protocols

A protocol declares methods not associated with a class, but which any class, and if desired many classes, might implement. Any class that want to respond to a specific method should adopt the protocol and implement it.

`PETModel.h` ~ defines the **PETModel** protocol, an interface for objects of the **Controller** implementing parts of the physical model: **SourceModel**, **ScannerModel**, and **ScatterModel**.

`PETInspectable.h` ~ defines the **PETInspectable** protocol, an interface for objects that may be inspected in a **PETInspector**.

`PETBrowsable.h` ~ defines the **PETBrowsable** protocol, an interface for objects that may be browsed in the controller's modelBrowser.

`PETUnit.h` ~ defines the **ManagedUnits** protocol.

Categories

Categories are used throughout the application, mainly to add methods to classes by declaring them in an interface file under a category name and defining them in an implementation file. The category name indicates that the methods are additions to a class declared elsewhere, not a new class. This simplifies the management of large classes and allows to take advantage of incremental compilation.

Basic Classes

Controller

Description

As in the well-known Model-View-Controller paradigm, the ***Controller*** is the central object which manages the views and models that compose the application.
It instantiates and initializes the other principal objects (models and views).

Files

`Controller.h`	~	interface.
`Controller.m`	~	*PrivateMethods* and (public) *ModelMethods*.
`ControllerPETBrowsable.m`	~	interface to the browsable ***modelList***.

TwoDViews

Description

These two classes provide a very simple ***View*** for viewing planar image data provided as an array of floats via the *setData*: method. Only an extremely simple color map is implemented.

Classes

ColorTwoDView
ClickTwoDView ~ subclass of **ColorTwoDView**, implementing a *mouseDown* method to allow the value of the clicked-on pixel to be displayed in the *mouseLocXView* and *mouseLocYView* (set up in the `.nib` file).

Files

`ColorTwoDView.m, .h`	~	the **ColorTwoDView** class and its interface.
`ClickTwoDView.m, .h`	~	the **ClickTwoDView** class and its interface.

Photon

Description

Represents a photon as a point particle with a location, a direction, an energy, and the number of times it has scattered, the ability to move (*propagate:*), and the ability to *interact* with a **ShapeBasedModel** (*interactWithModelList:*). The **ShapeBasedModel** is either a scattering medium or the scanner's block detectors.

Files

`Photon.m, .h`

GammaPair

Description

Represents a pair of Gamma particles. Contains two Photons.

Files

`GammaPair.m, .h`

Timer

Description

May be started, stopped, and reset. Returns elapsed time as an integer or as a string. The **Timer** is used mainly to evaluate the time required to perform a particular simulation study.

Files

`Timer.m, .h`

PETUnitsManager

Description

Keeps track of whether the user wants to view dimensions in pixel units or mm. There are still some unresolved interface questions, such as updating the slider limits in the status view to reflect the current units.

Files

`PETUnitsManager.m, .h`

ShapeBasedModel

Description

A framework for browsable models build from 3D shapes, such as **SourceModel** and **ScatterModel**. For instance, ***parallelepipeds***, ***ellipsoids*** and and ***cylindroids*** are implemented. Provides basic create/destroy and browse functions. Used with **ShapeInspector**.

Files

`ShapeBasedModel.m, .h`

PETInspector

Description

A generalized inspector class for objects with parameters that may be set with sliders. Several subclasses are derived from **PETInspector**. Has hooks for a **PETUnitsManager**.

Files

`PETInspector.m, .h`

ShapeInspector

Description

A subclass of **PETInspector**, this is a generalized inspector class for objects based on the parameters of a 3D shape. Provides controls for center location, size, rotation, and shape selection.

Files

`ShapeInspector.m, .h`

Subprojects

PRNGenerator.subproj

This subproject encapsulates code for various random number generators. It contains one class, **PRNGenerator** and C routines.

PRNGenerator

Description

Provides methods to set different generator types (defined type PRNGeneratorType in `PRNGenerator.h`), and to compute the next deviate either Gaussian or uniform using the selected generator.

Files

`PRNGenerator.m, .h`

Other Sources

`random_number.c`	(Marsaglia algorithm)
`frand1.c`	(Park and Miller 's algorithm)
`sobol.c`	(Sobol algorithm)

Primitives3D.subproj

This subproject encapsulates four primitive 3D classes. To use it, an outside object must `#include Primitives3D.h`.

_____
Point3D

Description

Provides basic methods for points in 3D space: setting, copying, distance calculation, etc.

Files

`Point3D.m, .h`

_____
Direction3D

Description

A subclass of **point3D**. Provides basic methods for orientations in 3D space.

Files

`Direction3D.m, .h`

_____
Matrix4

Description

Provides methods for 4-dimensional matrices such as determinant, inversion, adjoint, and additional methods for performing 3D operations such as rotation/scaling/translation. **Matrix4** are used in *Eidolon* to represent transformations applied to **Point3D**.

Files

`Matrix4.m, .h`

_____
Shape3D

Description

Provides methods for simple 3D shapes: ***cylindroids***, ***ellipsoids***, and ***parallelepipeds***. Shapes are inspected using the companion class **ShapeInspector**. Functions to define a shape, apply a transform defined in a **Matrix4**, calculate intersection points of a photon with the chosen shape, and compute the distance to a **Point3D**.

Files

`Shape3D.m, .h`

_____
Other Sources

`shape3d_types.h` ~ defines ShapeTypes for the known shapes, and defines the default shape parameters (center location and 3-dimensional size).

Dynamically loaded modules

Sinogram.bproj

This subproject encapsulates the **Sinogram** class and its inspector.

_____
Sinogram

Description

Provides basic methods for a sinogram: sizing, and file I/O including CTI format. An important method is the element increment method, used whenever a gamma pair is detected by the **ScannerModel**. Hooks for use with **SinogramInspector**. Methods to compute the spectra of detected photons as well as parameters like scatter fraction are also implemented.

Protocols

PETInspectable

Files

`Sinogram.m, .h`

SinogramInspector

Description

Subclass of **PETInspector** that can display data from a ***Sinogram*** object in a ***ColorTwoDView***.

Files

`SinogramInspector.m, .h, .nib`

Image.bproj

This subproject encapsulates the **Image** class and its inspector.

Image

Description

Provides basic methods for a 3D volume image: sizing, and initialisation of image parameters, such as pixel size, file name for saving the reference image.

Protocols

PETInspectable

Files

`Image.m, .h`
`ImageCTI.c, .h`

ImageInspector

Description

Subclass of **PETInspector** that can display data from an ***Image*** object in a ***ColorTwoDView***.

Files

`ImageInspector.m, .h, .nib`

PixelVolume.bproj

This subproject encapsulates the **PixelVolume** class and its inspector. **PixelVolume** and an **Image** are somewhat similar but they perform different tasks. Nevertheless, **Image** class could be removed and its methods added to the **PixelVolume** class as categories.

PixelVolume

Description

Provides basic methods for a 3D volume image: sizing, file I/O including CTI format. An important method is the *pixelIncrement* method, used whenever a gamma pair is generated by the **SourceModel**. Hooks for use with **ImageInspector**.

Protocols

PETInspectable

Files

`PixelVolume.m, .h`
`reconstruction.c, .h`
`saveImageCTI.c`

PixelVolumeInspector

Description

Subclass of **PETInspector** that can display data from a ***PixelVolume*** object in a ***ColorTwoDView***.

Files

`PixelVolumeInspector.m, .h, .nib`

ScannerModel.bproj

This subproject encapsulates the **ScannerModel** class and its inspector.

ScannerModel

Description

A simple representation of a cylindrical scanner with adjustable number of rings and detectors per ring, with the capability to detect coincident pairs of ***Photon*** objects (through the *interact:* method) within an adjustable energy window and record this in a ***Sinogram*** object.

Protocols

PETModel
PETBrowsable
PETInspectable

Files

`ScannerModel.m, .h`

ScannerInspector

Description

Subclass of **PETInspector** for adjusting scanner parameters and dimensions.

Files

`ScannerInspector.m, .h, .nib`
`ScannerModelPETInspectable.m`

SourceModel.bproj

This subproject encapsulates the **SourceModel** class and its inspector.

Source

Description

Source is a subclass of **Shape3D** that can produce **GammaPair** of a specified energy initialized to lie within the volume of the **Shape3D**. A **Source** uses a ***PRNGenerator*** object to compute the location and direction of each ***GammaPair***. For efficiency purposes, the polar angle of photons is set in the aperture of the scanner through the *randomDirectionInAperture:* method.

Protocols

PETInspectable

Files

`Source.m, .h`

SourceModel

Description

A **SourceModel** is a subclass of **ShapeBasedModel**. It contains a ***List*** of ***Source*** objects. To generate a number of ***GammaPair***, use the *pairList:* method which produces a ***List***. The initial directions of the ***GammaPair*** produced by the enclosed ***Source*** may be constrained to lie within a given aperture with the *setMaxAperture:* method.

Protocols

PETModel
PETBrowsable
PETInspectable

Files

`SourceModel.m, .h`

SourceInspector

Description

Subclass of **ShapeInspector** (and thus **PETInspector**) for setting individual **Source** parameters and dimensions.

Files

`SourceInspector.m, .h, .nib`

ScatterModel.bproj

This subproject encapsulates the **ScatterModel** class and its inspector.

_____
Scatterer

Description

Scatterer is a subclass of **Shape3D** that can *interactWith:* ***GammaPair***. The interaction cross-sections and interaction simulation are computed using GEANT routines, `compton.c` and `rayleigh.c`, which were translated from FORTRAN to C using f2c.

Protocols

PETInspectable

Files

`Scatterer.m, .h`
`compton.c, .h`
`rayleigh.c, .h`

_____
ScatterModel

Description

A **ScatterModel** is a subclass of **ShapeBasedModel**. It contains a ***List*** of ***Scatterer*** objects.

Protocols

PETModel
PETBrowsable
PETInspectable

Files

`ScatterModel.m, .h`

_____
ScattererInspector

Description

Subclass of **ShapeInspector** (and thus **PETInspector**) for setting individual **Scatterer** parameters and dimensions. Setting of the chemical properties of **Scatterer** is also implemented.

Files

`ScattererInspector.m, .h, .nib`

Other Sources

`pet_utils.subproj`	~	contains `cti_utils.c` (interface file: `pet_utils.h`)
`PETDefaults.h`	~	defines defaults for scanner dimensions, image size, sinogram padding, source and scatterer dimensions, scatterer and detector composition, iteration parameters (pairs per step), etc.

How to install the Simulator

The latest version of *Eidolon* we have is in `http://dmnu-pet5.hcuge.ch/Eidolon.tar.gz.`

All *Eidolon* documentation is presented in this report. More details are presented in published papers.

NOTE: Only the files included in `Makefile` are actually used ! There may be files in a project directory that are not listed in the the main or other makefiles. These are **not used**.

To install *Eidolon*, you simply need to decompress and unarchive the package. The latest version of gcc (version 2.8.1) is highly recommended in order to compile the code. The simulator was successfully ported on HP, Sun SPARC stations including Fujitsu parallel system, as well as the Parsytec-CC system (AIX.4.1) for use on this parallel architecture. No PC running Linunx was available to us, but we don't think there will be a problem to run the simulator on this platform.

An example of parallel implementation of the simulator on the Parsytec-CC system is provided below. Users might implement the same methology on other parallel platforms. They also might use parallel programming environments like PVM or MPI.

How to parallelise the Simulator on Parsytec-CC

Please modify your `/epx/epx/nrm.conf` file to create a partition for each single processor. The simulator needs to read a file called `config.txt` to know the characteristics of your system (number of nodes, partition names). Our `nrm.conf` is configured as follows:

```
CC ParsytecCC FILE /epx/epx/nrm/system.rt 8,1,1 12544
@ - no init info -§
#
NODE parapet  0,0,0 master-epxd IO-Node
NODE cc002696  1,0,0 IO-Node
NODE cc002701  2,0,0 P-Node
NODE cc002702  3,0,0 P-Node
NODE cc002703  4,0,0 P-Node
NODE cc002704  5,0,0 P-Node
NODE cc002705  6,0,0 P-Node
NODE cc002706  7,0,0 P-Node
#
SERVER filesrv '/epx/bin/dserver -h'
#
PARTITION ccall: p8 0,0,0 7,0,0
#
PARTITION ccleft: p4 habib 0,0,0 3,0,0
PARTITION ccright: p4 claire 4,0,0 7,0,0
PARTITION twin1: p2 0,0,0 1,0,0
PARTITION twin2: p2 2,0,0 3,0,0
PARTITION compute1: p1 2,0,0 2,0,0
PARTITION compute2: p1 3,0,0 3,0,0
PARTITION compute3: p1 4,0,0 4,0,0
PARTITION compute4: p1 5,0,0 5,0,0
PARTITION compute5: p1 6,0,0 6,0,0
PARTITION compute6: p1 7,0,0 7,0,0
PARTITION entry: p1 0,0,0 0,0,0
PARTITION io: p1 1,0,0 1,0,0
#
```

An example of `config.txt` is shown below for our system consisting of 8 nodes (for 12 systems with 12 nodes, just add 4 compute nodes, compute7..compute10). Please modify according to your system.

`config.txt`

```
8               * Number of processors
```

```
entry           * name of partition 1 (single processor)
io              * name of partition 2 (single processor)
compute1        * name of partition 3 (single processor)
compute2        * name of partition 4 (single processor)
compute3        * name of partition 5 (single processor)
compute4        * name of partition 6 (single processor)
compute5        * name of partition 7 (single processor)
compute6        * name of partition 8 (single processor)
```

Two files handle the parallel features: `sumScans.c` (executable `sumScans`) which sums image and sinogram data sets and `para_main.c` (executable `paraMC`) which is in charge of running the simulator on the different nodes chosen in `config.txt`.

How to configure the Simulator

The physics of the simulation process (scatter, attenuation) may be selected from a file called `Interaction.txt (Eidolon/Parameters/Interaction.txt)`. An example of this file is:

`Interaction.txt`

```
0         * Object Scatter (0 -> no scatter; 1 -> with scatter)
0         * Detector Scatter (0 -> no scatter; 1 -> with scatter)
```

Thus 4 combinations are possible:

```
(0,0): Scatter-free imaging;
(0,1): Object scatter-imaging;
(1,0): Detector scatter-imaging;
(1,1): Full scatter-imaging.
```

`CRYSTAL_THICKNESS` in scanner parameters `(Eidolon/Parameters/Scanner.txt)` should be set to `0` in case of scatter-free imaging.

The sources' geometries, size, location and orientation are specified in a text file called `specifications.dat`. The user is free to build and save his own sources in other text files and call them in `Controller.m` class (method *run*) by replacing `specifications.dat` by the name of the new file. Examples of the Utah phantom are provided and `utah.dat`, respectively.

`Eidolon/Parameters/specifications.dat`

```
2              * 1--> VB digital phantom,  2-->SB geometrical shapes
2              * number of sources                                  (int)
Cylindroid     * src 1: shape                                       (string)
0 0            * src 1: tilt (azimuth,altitude rotation)            (double)
10 10 54       * src 1: size (x,y,z)                                (double)
0 0 54         * src 1: centre (x,y,z)                              (double)
1000000        * src 1: number of generated counts                  (long)
0              * src 1: number of shapes to exclude                 (int)
Parallelepiped * src 2: shape                                       (string)
0 0            * src 2: tilt (azimuth,altitude rotation)            (double)
10 10 54       * src 2: size (x,y,z)                                (double)
100 0 54       * src 2: centre (x,y,z)                              (double)
500000         * src 2: number of generated counts                  (long)
0              * src 2: number of shapes to exclude                 (int)
```

To create a cold source region (hole) within a simple geometry, you need to follow the same example provided in `Eidolon/Parameters/utah.dat` by simply excluding regions within that geometry. Better implementations will be provided with the next release including voxel-based implementation of phantoms which will be distributed to partners later.

utah.dat

```
2              * 1--> VB digital phantom,  2-->SB geometrical shapes
3              * number of sources                                  (int)
Cylindroid     * src 1: shape                                       (string)
0 0            * src 1: tilt (azimuth,altitude rotation)            (double)
100 100 75     * src 1: size (x,y,z)                                (double)
0 0 75         * src 1: centre (x,y,z)                              (double)
12289500       * src 1: number of generated counts                  (long)
1              * src 1: number of shapes to exclude                 (int)
Cylindroid     * src 1 exshape 1: shape                             (string)
0 0            * src 1 exshape 1: tilt (azimuth,altitude rotation) (double)
80 80 75       * src 1 exshape 1: size (x,y,z)                      (double)
0 0 75         * src 1 exshape 1: centre (x,y,z)                    (double)
Cylindroid     * src 2: shape                                       (string)
0 0            * src 2: tilt (azimuth,altitude rotation)            (double)
80 80 75       * src 2: size (x,y,z)                                (double)
0 0 75         * src 2: centre (x,y,z)                              (double)
10000000       * src 2: number of generated counts                  (long)
2              * src 2: number of shapes to exclude                 (int)
Cylindroid     * src 2 exshape 1: shape                             (string)
0 0            * src 2 exshape 1: tilt (azimuth,altitude rotation) (double)
22.5 22.5 52.5 * src 2 exshape 1: size (x,y,z)                      (double)
-47.5 0 52.5   * src 2 exshape 1: centre (x,y,z)                    (double)
Cylindroid     * src 2 exshape 2: shape                             (string)
0 0            * src 2 exshape 2: tilt (azimut,altitude rotation)  (double)
22.5 22.5 27.5 * src 2 exshape 2: size (x,y,z)                      (double)
47.5 0 27.5    * src 2 exshape 2: centre (x,y,z)                    (double)
Cylindroid     * src 3: shape                                       (string)
0 0            * src 3: tilt (azimut,altitude rotation)             (double)
22.5 22.5 27.5 * src 3: size (x,y,z)                                (double)
47.5 0 27.5    * src 3: center (x,y,z)                              (double)
634080         * src 3: number of generated counts                  (long)
0              * src 3: number of shapes to exclude                 (int)
```

The scatter parameters are specified in a text file called `Eidolon/Parameters/Scatterer.txt`. A sample of this file is:

Scatter.txt

```
Cylindroid     * shape                                              (string)
0 0            * tilt (azimuth,altitude rotation)                   (double)
10 10 54       * size (x,y,z)                                       (double)
0 0 54         * centre (x,y,z)                                     (double)
H2O            * SCATTERER COMPOUND //H2O, Air,Al,Pb,SoftTissue,Bone,Lung
```

In case the digital phantom used as input to the simulator is a voxel-based one, the specifications files should be replaced by an equivalent file which contains basically the name of the emission source distribution and segmented transmission/CT according the organ identities adopted by Zubal [14] and mentioned before.

An example of the Hoffman voxel-based phantom (`Eidolon/Parameters/hoff_mrc.dat`) is shown below:

hoff_mrc.dat

```
1                       * 1--> VB digital phantom, 2--> SB geometry
Phantoms/hoffman.cimg   * emission phantom name
2                       * number of bits
Phantoms/hoffman.tct    * Segmented CT phantom name
2                       * number of bits
128 128   19            * resolution  xyz (int)
2 2 6.86                * pixel size  xyz (double)
0                       * Axial (z) translation of the phantom in mm
```

```
1500000                      *  number of generated counts   (long)
```

`hoffman.cimg` corresponds to the distribution in the source volume, while `hoffman.tct` corresponds to the segmented distribution of attenuation coefficients indexed according to description by Zubal et al [15] and summarised in Table 3.

Different combinations of the SB and VB approach are possible. For example, one can choose to simulate a VB phantom (Hoffman Brain phantom) in a simple geometrical form containing uniform distribution of attenuation coefficients (cylinder) or a non-uniform VB phantom.

The equivalent scatterer SB geometry for such a phantom is again specified in a text file called `Eidolon/Parameters/scatter_hoffman.txt`

`scatter_hoffman.txt`

```
Cylindroid       * shape of Scatter (string)
0  0             * tilt of Scatter (Azimut and altitude Rotation) (double)
100 100 87.5     * Scatter size xyz   (double) 104   104
0 0 87.5         * Scatter centre xy (double)     Ctr_Z - translate_Z
H2O              * Chemical composition of scatterer
```

where `translate_Z` is the axial translation with respect to the Z-axis of the scanner.

The equivalent VB for the hoffman phantom (128*x*128*x*19 slices) should be a parallelepiped whose size is simply calculated by the resolution of the phantom and the pixel size in three-dimensions:

`Scatter_hoffman_vox.txt`

```
Parallelepiped      * shape of Scatter (string)
0  0             * tilt of Scatter (Azimut and altitude Rotation) (double)
128 128 87.5     * Scatter size xyz   (double)
0 0 87.5         * Scatter centre xy (double)     Ctr_Z - translate_Z
H2O                 * Chemical composition of scatterer
```

It is important to note that appropriate function call should be selected when choosing one of those two options. This should be done in the `Controller.m` class. That is, in the case that scatter is simulated within both scattered and detectors (full scatter simulation), the member function `detectPairsFullScatterVoxel` shoud be activated in the function `stepBufferedFullscatterVoxel` if the user want to simulate scattering in a VB phantom. The call to `detectPairsFullScatterVoxel` should be commented as shown below and vice versa. This is a simple way to handle all the options, but definitely needs some improvement.

`Eidolon/Controller.m`

```
- (void) stepBufferedFullscatterVoxel: (int *) resolution: (Point3D *)
pixel_size: (float) translation_z
{
  int i;

   for (i = 0; i < DEFAULT_STEPS_PER_UPDATE; i++) {
if ([sources pairList: pairList count: stepSize]) {
[self detectPairsFullScatterVoxel: resolution: pixel_size: translation_z];
//[self detectPairsFullScatterPhantom: resolution: pixel_size:
translation_z];
     [self done];
     break;
 }
 else
 [self detectPairsFullScatterVoxel: resolution: pixel_size: translation_z];
//[self detectPairsFullScatterPhantom: resolution: pixel_size:
translation_z];
    }
}
```

The scanner parameters are specified in a text file called `Eidolon/Parameters/Scanner.txt`. A sample of this file for RPT-1 is:

`Scanner.txt`

```
16    * NUMBER OF RINGS                      //RPT
384   * NUMBER OF DETECTORS                  //RPT
380   * SCANNER RADIUS [mm]                  //RPT
6.75  * RINGSPACING [mm]                     //RPT
15    * MAXRINGDIFF (DEFAULT_RINGS-1)        //RPT
96    * NUMBER OF VIEWS                      //RPT
127   * NUMBER OF PROJS                      //RPT
0.23  * DETECTOR ENERGY RESOLUTION (FWHM)    //23% @ 511keV, ∝ 1/√E
250   * Lower ENERGY Threshold [keV]
850   * Higher ENERGY Threshold [keV]
30    * CRYSTAL_THICKNESS [mm]               //RPT, 0 if no detector scatter
BGO   * DETECTOR COMPOUND                    //BGO, NaI(Tl), BaF2,LuAP,LSO CsI
64    * FANBEAM_RANGE                        //RPT
4     * DETECTOR SPATIAL RESOLUTION FWHM [mm]
```

A collection of both shape-based and voxel-based phantoms and their scattering medium geometries as well as parameters of different scanners can be found in the directories `Parameters` (for text files) and `Phantoms` (for digital bitmap data). The resuling files will be saved in the directory `/DATA` by default, as fixed in `Simulation_main.m`. The file names are specified in a text file called `setup.txt` in the directory `Parameters` which contains the initial seed (first line) and the name of the file to be saved. An example is:

`Eidolon/Parameters/setup.txt`

```
1000
Cylinder_1
```

For parallelisation purposes, the same program should be run on all the nodes available and the resulting data sets collected and summed on the entry node, however, the seeds used for the initialisation of the random number generator should be different.

The seed is initialised in `PRNGenerator.m` (method *init*). For Marsaglia's algorithm, the function `start_random_number (int seed_a, int seed_b)` is used. This should be initialised as follows in the class `PRNGenerator.m` (e.g.):

`Eidolon/PRNGenerator.m`

```
- init

start_random_number (1000, 1000);  // example could be any n (int) (n,n)
```

This procedure initialises the state table u for a lagged Fibonacci sequence generator, filling it with random bits from a small multiplicative congruential sequence. The seeds are transformed into an initial state in such a way that identical results are guaranteed across a wide variety of machines.

A simple way of running the simulator in parallel is to initialise the seeds differently for the versions running on different processors, that is, in the `setup.txt` file. The approach adopted allows data to be saved in a temporary file on disk so that the random seed used is different on each processor.

How to compile the Simulator

To compile the software, you need to execute `make` or `make all` from the main directory. Please note that several `Makefiles` exist within the different subprojects (directories).

Ignore any warnings !

Although compiling is not necessary for users using SUN SPARC stations (binaries are supplied with this version), the user needs only to modify the text files. Two files handle the parallel features: `sumScans.c`

(executable `sumScans`) which sums image and sinogram data sets and `para_main.c` (executable `paraMC`) which is in charge of running the simulator on the different nodes chosen in `config.txt`. No modifications of the `Makefiles` are necessary since they do not contain platform-dependent commands (apart NEXT).

How to run the Simulator

If you are using the sequential version of the simulator, type `Simulator`. This will run the simulator on the a single processor.

To run the Simulator on the Parsytec-CC parallel platform, type `paraMC`. This will run the simulator on the different nodes selected in `config.txt`.

The following files will be saved in the directory /`DATA` by default (fixed in `Simulation_main.m`):

`timing`: time needed to perform the simulation;

`phantom.scn`: sinogram data sets;

`phantom.img`: reference image;

`phantom.dat`: statistics (number of events detected, scattered, etc.).

You can also save other parameters such as:

`Total_Spectra.dat`: Total energy spectra of detected photons;

`True__Spectra.dat`: energy spectra of unscattered detected photons;

`Scatter_Spectra.dat`: energy spectra of scattered detected photons.

You need to uncomment appropriate lines in `Simulation_main.m` if you want to save those parameters.

Example of the Output

```
Scanner: ECAT 953B
Source: line source (diameter =1mm) located at the axis of rotation of the
scanner
Scatterer: Water-filled cylinder (Radius =100 mm)
Energy Window:  250 - 750 keV
Energy resolution of the detector block: 23%
Statistics:
Total coincidences = 22851
True coincidences 3D= 14265
Scattered coincidences 3D= 8586
Single trues = 35723
Single scatters = 9979
Single scatters order 1 = 8242
Single scatters order 2 = 1510
Single scatters order 3 = 206
Single scatters order 4 = 21
Scatter fraction 3D= 0.602
Scatter-to-Total fraction 3D= 0.376
Total detector unscattered = 12088
Total detector scatters = 14874
Detector scatters order 1 = 9675
Detector scatters order 2 = 2686
Detector scatters order 3 = 1068
Detector scatters order 4 = 1445
Average distance traveled by photons in the crystal = 10.184 mm
Generated pairs = 1e7
```

References

[1] Thompson CJ, Cantu J-M, Picard Y. PETSIM: Monte Carlo program simulation of all sensitivity and resolution parameters of cylindrical positron imaging systems. Phys. Med. Biol. 37**:** 731-749 (1992).

[2] DelGuerra A, Nelson WR. Positron Emission Tomography Applications of EGS. in Monte Carlo transport of electrons and photons. Jenkins TM, Nelson WR, Rindi A. Plenum publishing corporation. 469-484 (1988).

[3] Michel C, Bol A, Spinks T, Townsend D, Bailey D, Grootoonk S, Jones T. Assessment of response function in two PET scanners with and without interplane septa. IEEE Trans. Med. Imag. 10: 240-248 (1991).

[4] Herrmann Scheurer AK, Egger ML, Joseph C, and Morel C. A Monte Carlo phantom simulator for positron emission tomoraphy. Proc. of the Workshop on Supercomputing in brain research: from tomography to neural networks, Jülich, 1994, eds Hermann HJ, Wolf DE and Pöppel E, pp. 205-209 (World Scientific Publishing, 1995).

Zaidi H, Hermann Scheurer AK, and Morel C. Development of an Object-Oriented Monte Carlo simulator for 3D Positron Tomography. Conf. Rec. of International Meeting on Fully Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine, Nemacolin Woodlands, 1997, eds Kinahan P and Townsend D, pp 175-179 (UPMC, Pittsburg, 1997).

Zaidi H, Hermann Scheurer AK, and Morel C. An Object-Oriented Monte Carlo Simulator for 3D Positron Tomographs", Comput. Meth. Prog Biomed., 57: 133-145 (1999)

Zaidi H, C. Labbé, and C. Morel. Implementation of a Monte Carlo simulation environment for fully 3D PET on a high-performance parallel platform. Parallel Computing, 24: 1523-1536 (1998)

Zaidi H, AK. Hermann Scheurer, and C. Morel. Development of an Object-Oriented Monte Carlo simulator for 3D Positron Tomography". Proceedings of the International Meeting on Fully Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine (3D'97), Pittsburgh, USA 25-28 June 1997, pp 176-179.

Zaidi H, Labbé C, and Morel C. Improvement of the performance and accuracy of PET Monte Carlo simulations". SPIE's International Symposium on Medical imaging 1999, San Diego, USA, Vol. 3659, edited by J. M. Boone and J. T. Dobbins (1999), pp 582-593.

[5] Harrison RL and Vannoy SD. Preliminary experience with the photon history generator module for a public-domain simulation system for positron emission tomography. Conf. Rec. Nucl. Sci. Symp. Vol 2, pp 1154-1158 (1993).

[6] Haynor DR, Harrison RL and Lewellen TK. Energy-based scatter correction for 3D PET: a Monte Carlo study of best possible results. Conf. Rec. of International Meeting in Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Nemacolin Woodlands, 1997, eds Kinahan P and Townsend D, pp 52-54 (UPMC, Pittsburgh, 1997).

[7] Marsaglia G, Zaman A. Monkey tests for random number generators. Computers & Mathematics with Applications. 23: 1-10 (1993).

[8] Brun R, Bruyant F, Maire M, McPherson AC, and Zanarini P. GEANT 3. CERN Data Handling Division DD/EE/84-1 (1986).

Brun R, Bruyant F, Maire M, McPherson AC, and Zanarini P. GEANT Detector Description and Simulation Tool, CERN Program Library, W5013 (1994).

[9] Spinks TJ, Jones T, Bailey DL, *et al*. Physical performance of a positron tomograph for brain imaging with retractable septa. Phys. Med. Biol. 37: 1637-1655 (1992).

[10] Townsend DW, Choi Y, Sashin D, Mintun MA. An investigation of practical scatter correction techniques for 3D PET. J. Nucl. Med. 50 (1994).

[11] Egger ML. Fast volume reconstruction in positron emission tomography: Implementation of four algorithms on a high-performance scalable parallel platform. Ph.D Thesis, University of Lausanne (1996).

Egger ML, Herrman Scheurer AK, Joseph C, and Morel C. Fast volume reconstruction in positron emission tomography: implementation of four algorithms on a high-performance scalable parallel platform, Conf. Rec. IEEE Med. Imag. Conf., Anaheim, 1996, pp. 1574-1578 (New York IEEE, 1997).

[12] Townsend DW, Geissbuhler A, Defrise M, Hoffman EJ, Spinks TJ, Bailey DL, Gilardi MC. Fully three-dimensional reconstruction for a PET camera with retractable septa. IEEE Trans. Med. Imag. 10: 499-594 (1991).

[13] Spinks TJ, Grootonk S, Bailey DL, Schnorr L. A comparison of 3D scatter correction methods in a neuroPET tomograph. Conf. Rec. Med. Imag. Conf. pp 1618-1622 (1995)

[14] Zubal IG, Harell CH. Voxel based Monte Carlo calculations of Nuclear Medicine images and applied variance reduction techniques. Image and Vision Computing. 10: 342-348 (1992).

[15] Zubal IG, Harrell CR, Smith EO, Rattner Z, Gindi G, Hoffer BP. Computerized 3-Dimensional Segmented Human Anatomy, Medical Physics, Vol. 21, No. 2, pp 299-302 (1994).

[16] Hoffman E J, Cutler P D, Digby W M, and Mazziotta J C. 3-D phantom to simulate cerebral blood flow and metabolic images for PET. IEEE Trans. Nucl. Sci. **37**, pp 616-620 (1990).

[17] Siddon R L, “Fast calculation of the exact radiological path for a three-dimensional CT array,” Med. Phys. **12**, pp 252-255 (1985).