Chapter 5

Computational Methods for Photon-Counting and Photon-Processing Detectors

Luca Caucci, Yijun Ding and Harrison H. Barrett

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72151

Abstract

We present computational methods for attribute estimation of photon-counting and photon-processing detectors. We define a photon-processing detector as any imaging device that uses maximum-likelihood methods to estimate photon attributes, such as position, direction of propagation and energy. Estimated attributes are then stored at full precision in the memory of a computer. Accurate estimation of a large number of attributes for each collected photon does require considerable computational power. We show how mass-produced graphics processing units (GPUs) are viable parallel computing solutions capable of meeting the required computing needs of photon-counting and photon-processing detectors, while keeping overall costs affordable.

Keywords: photon-processing detectors, maximum-likelihood estimation, GPU, parallel processing, gamma-ray photons, charged particles

1. Introduction

In broad terms, detectors used in imaging can be classified into a small number of categories depending on their working principles. These categories include integrating detectors, pixelated photon-counting detectors as well as a new class of detectors, which we refer to as photon-processing detectors.

An integrating detector measures charges accumulated at each pixel location. These charges are induced by light impinging on the detector and are proportional to the average number of photons incident on each pixel. Dedicated circuitry reads out these changes and converts them to numbers roughly proportional to the charge accumulated at each pixel.

A photon-counting detector works by counting the number of photoelectric interactions observed during the exposure time. Count registers associated with each pixel are read at the

end of the exposure time, thus making the output of photon-counting detectors a collection of pixel counts.

A photon-processing detector may use any existing detector technology to measure several "attributes" for each photon entering the detector. Attributes include the photon position, its direction of propagation and the amount of energy it deposited in the detector. This is accomplished by reengineering the detector design so that additional information can be extracted from raw unprocessed data. Important aspects of any photon-processing detector include the algorithm used to estimate photon attributes from raw data as well as how these attributes are represented and stored in the memory of a computer.

Particle-processing detectors are a variation on photon-processing detectors and are designed to detect charged particles, such as alpha and beta particles. Particle-processing detectors enable a new imaging technique—called charged-particle emission tomography (CPET)—which attains high-resolution 3D imaging in living organisms so long as accurate estimation of parameters for each charged particle is available.

This chapter is organized as follows. Section 2 provides an overview of detectors suitable for photon counting and photon processing. Maximum-likelihood estimation (MLE) and its properties are discussed in some detail in Section 3. The next section—Section 4—introduces graphics processing units (GPUs) and the compute unified device architecture (CUDA) programming model. Section 5 presents algorithms for photon-counting detectors, while Section 6 discusses photon- and particle-processing detectors and presents fast GPU-based algorithms for maximum-likelihood estimation of photon parameters. Finally, Section 7 summarizes this chapter and discusses possible applications of photon-processing detectors.

A portion of this chapter has been adapted from Y. Ding, *"Charged-Particle Emission Tomography"* [1].

2. Detectors for photon counting and photon processing

2.1. Gamma-ray cameras

Gamma-ray cameras are used in nuclear medicine to image gamma-ray photons emitted by radioactive elements. The first gamma-ray camera was developed by Hal Oscar Anger in 1957 [2]. Anger's original design, often referred to as an "Anger camera," is still widely used today. A diagram of an Anger camera is provided in **Figure 1**.

An Anger camera includes a scintillation crystal, a light guide and an array of photomultiplier tubes (PMTs). When a gamma-ray photon interacts with the scintillation crystal, a burst of visible-light photons is produced. Some of these photons propagate through the crystal and the light guide and enter one or more PMTs. When a photon enters a PMT and interacts with it, a measurable electrical signal in the form of a narrow current pulse is produced. This pulse is transmitted to amplifying electronics, so that it can be analyzed. A transimpedance amplifier converts the current pulse to voltage. A shaping amplifier further amplifies the signal and reshapes it, making it broader and smoother. A broad signal is easier to sample via an analog-to-digital converter. The output of the analog-to-digital converter can be scaled to obtain an

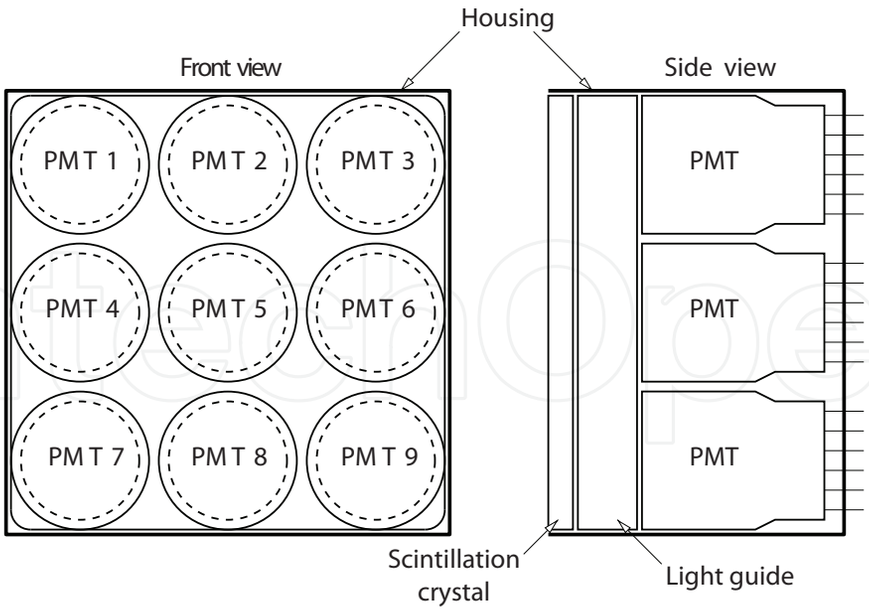

Figure 1. Diagram of a typical gamma-ray camera (adapted from [3]).

integer number representing the number of photons entering the PMT. Digitized samples collected from each of the K PMTs are then scanned for events. Detected events are stored in the memory of a computer in the form of scaled PMT samples $g_1, \ldots, g_K$.

Detailed analysis of the physical processes that take place inside the scintillation crystal and each PMT allows us to derive a statistical model for the scaled PMT samples $g_1, \ldots, g_K$ produced by a gamma-ray camera with K PMTs. Because of noise, PMT samples $g_1, \ldots, g_K$ can be thought of random variables. If we normalize each PMT signal by the gain of the PMTs and we ignore the noise in the gain, random variables $g_1, \ldots, g_K$ can be shown to be conditionally independent and to follow Poisson statistics with means, respectively, $\overline{g}_1(\boldsymbol{R}, E), \ldots, \overline{g}_K(\boldsymbol{R}, E)$ [4]. Thus, we can write:

$$\mathrm{pr}(g_1, \ldots, g_K \mid \boldsymbol{R}, E) = \prod_{k=1}^{K} \frac{\left[\overline{g}_k(\boldsymbol{R}, E)\right]^{g_k} \exp\left[-\overline{g}_k(\boldsymbol{R}, E)\right]}{g_k!}. \tag{1}$$

Functions $\overline{g}_1(\boldsymbol{R}, E), \ldots, \overline{g}_K(\boldsymbol{R}, E)$ are called mean detector response functions (MDRFs), and they describe the mean detector response upon detection of a gamma-ray photon with energy E at location $\boldsymbol{R}$.

2.2. Semiconductor detectors for charged particles

Semiconductor pixelated detectors can be used to measure position and energy of charged particles, including alpha and beta particles. One possible detector configuration consists of a layer of semiconductor material (which we refer to as the "active volume"), a set of anodes placed on one side of the detector's active volume, and some data-processing circuitry (such as application-specific integrated circuits or ASICs) that measures the anode signals and converts them into digital signals.

When a charged particle enters the detector's active volume and deposits some of its energy, electron-hole pairs are produced along the particle's track. The electrons and holes drift in opposite directions under an electric field applied throughout the detector's active volume. This process is accompanied by the production of electrical charges, which are collected by electrodes on one side of the detector's active volume. These charges are then converted to digital signals (e.g., number of electron-hole pairs produced) and are either sent to a computer or accumulated in count registers to form an image.

An example of a semiconductor pixelated detector for alpha and beta particle is the Medipix2 sensor (**Figure 2**) developed at CERN [5]. The Medipix2 sensor features an array of 256 × 256 square pixels of size 55 μm. The counter in each pixel of a Medipix2 sensor can record the duration of an event that is above a threshold, from which the energy collected at each pixel and the particle's residual energy can be measured.

A statistical model for the data produced by a semiconductor detector for charged particles (such as the Medipix2 sensor) must take into account the so-called charge sharing effect [6] as well as many variables, including particle's position $\boldsymbol{R}$ and energy E, its angle of incidence (denoted as the unit vector $\vec{\boldsymbol{s}}$) and bias voltage V_{bias} applied across the semiconductor. Some recent results for the Medipix2 sensor have been reported in [1, 7]. When a highly energetic particle enters the detector, a large number of charges will be collected at its electrodes. In such a case, the statistics of pixel outputs $g_1, \ldots, g_M$ (where M denotes the number of detector pixels) conditioned on $\boldsymbol{R}$, E, $\vec{\boldsymbol{s}}$ and V_{bias} approach Gaussian statistics, and we can write:

$$\text{pr}\left(g_1, \ldots, g_M \middle| \boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right) = \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi\sigma_m^2\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right)}} \exp\left[-\frac{\left(g_m - \overline{g}_m\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right)\right)^2}{2\sigma_m^2\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right)}\right], \tag{2}$$

in which $\overline{g}_m\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right)$ is the mean of the m^{th} pixel and $\sigma_m\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, V_{bias}\right)$ is the standard deviation of g_m.

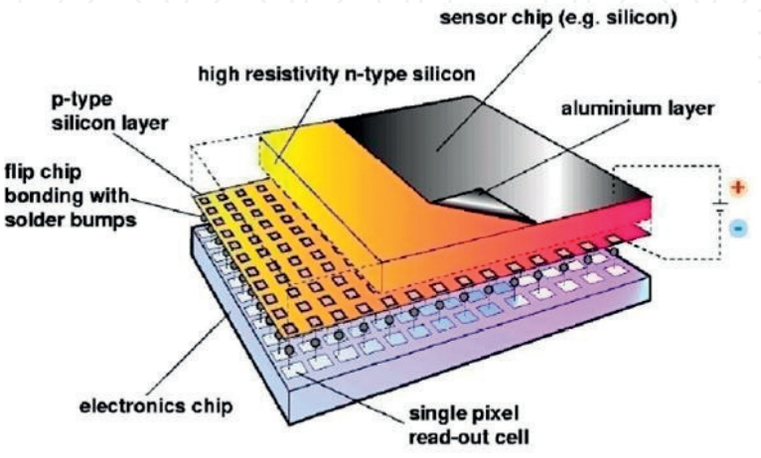

Figure 2. Diagram of the Medipix2 chip sensor (https://medipix.web.cern.ch).

2.3. Intensified charge-coupled detectors

A charge-coupled detector (CCD) is a semiconductor device that produces a pixelated image by converting incoming photons into electric charges, which are then stored at each pixel location. These charges are induced by photons with energy exceeding the semiconductor bandgap. The most general form for the mean output $\overline{g}_m$ (calculated by imaging the same object over and over again) is [8]:

$$\overline{g}_m = \int_{det} d^2R \int_0^\infty dE \int_{hemi} d\Omega \int_0^T dt\, \eta_m\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}\right) L\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, t\right), \tag{3}$$

in which m varies from 1 to the total number of pixels M, $\eta_m\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}\right)$ is the quantum efficiency at pixel m, point $\boldsymbol{R}$ on detector face, photon energy E and along direction $\vec{\boldsymbol{s}}$. The function $L\left(\boldsymbol{R}, E, \vec{\boldsymbol{s}}, t\right)$ is the spectral photon radiance at point $\boldsymbol{R}$ for photon energy$E$, time t and along direction $\vec{\boldsymbol{s}}$ [8, 9]. In Eq. (3), the spatial extent of the detector was denoted as "det" and "$\int_{\text{hemi}} d\Omega$" means integration over all the possible directions $\vec{\boldsymbol{s}}$ incident on the detector. Finally, integration over the time variable t starts at time $t = 0$ and ends at time $t = T$.

An intensified charge-coupled detector (ICCD) uses an image intensifier (such as a microchannel plate (MCP)) to amplify scintillation light before imaging it onto a CCD sensor. The image intensifier provides optical gain (in the range from 10^5 to 10^6 or more) so that low-energy optical photons (emitted, e.g., upon interaction of a charged particle with a scintillator) can be imaged with practically any CCD sensor. Lenses, usually placed between the image intensifier and the CCD sensor, reimage the image intensifier's output window on the CCD sensor. Examples of intensified charge-coupled detectors include the iQID sensor developed at the University of Arizona by Brian W. Miller [10].

A proper statistical model for an intensified charge-coupled detector must consider both the statistics of the output produced by the image intensifier and the statistics of the data produced by the CCD sensor. To find a model for the image intensifier, we begin by noticing that each point in the CCD sensor can be propagated back through the lenses all the way to the entrance face of the image intensifier. Therefore, we can consider the number of photons p_m impinging on the image intensifier at locations that fall within pixel m on the CCD sensor. Under broad conditions, we can show that p_m obeys Poisson statistics and we denote the mean of p_m as $\overline{p}_m$. For large enough $\overline{p}_m$, the statistics of p_m are approximatively Gaussian.

A general expression that relates $\overline{p}_m$ to the sensor output $\overline{g}_m$ takes the form:

$$\overline{g}_m = \overline{A}\overline{p}_m, \tag{4}$$

in which $\overline{A}$ denotes the mean of the image intensifier amplification (gain) A . The variance, σ_m^2, of $\overline{g}_m$ is related to $\overline{p}_m$ and the statistics of A as follows [7]:

$$\sigma_m^2 = \overline{p}_m\left(\sigma_A^2 + \overline{A}^2\right) + \sigma_{\text{read}}^2, \tag{5}$$

in which σ_A^2 is the variance of the amplification A and σ_{read}^2 denotes the variance of the sensor's readout noise. If the blur introduced by the image intensifier and optics located between the image intensifier and the CCD sensor is smaller than the size of a sensor pixel, then output g_m is independent on $g_{m'}$ for any $m' \neq m$. If we further assume that the amplification A and the readout noise also obey Gaussian statistics, we can write [1, 7]:

$$\text{pr}(g_1, \ldots, g_M) = \prod_{m=1}^{M} \frac{1}{\sqrt{2\,\pi\,\sigma_m^2}} \exp\left[-\frac{(g_m - \overline{g}_m)^2}{2\,\sigma_m^2}\right]. \tag{6}$$

3. Maximum-likelihood estimation

Maximum-likelihood estimation (MLE) is a statistical method that uses observed noisy data to estimate model parameters. For a good historical treatment of the concept of maximum-likelihood estimation, the interested reader can consult [11]. Given a set of observed data and an underlying model (which depends on some unknown parameters), MLE calculates the values of the parameters that better explain the observed data. The observed data that are used for maximum-likelihood estimation are realizations of random variables. Thus, parameters we estimate from these data are realizations of random variables as well.

Maximum-likelihood estimation can, in principle, be used with all the detectors discussed above. For example, we show how maximum-likelihood estimation is used to estimate position of interaction from PMT data, and we discuss an efficient parallel algorithm for it. Moreover and as we argue in Section 6, maximum-likelihood estimation is the estimation method of choice for photon-processing detectors.

3.1. Mathematical description

Let us denote the parameters we want to estimate as the vector $\boldsymbol{\theta}$. The model itself is characterized by a probability density function (PDF), denoted as $\text{pr}(\boldsymbol{x}|\boldsymbol{\theta})$. We use the vector $\boldsymbol{x}$ to refer to the *complete data*, while we denote the *incomplete data* as $\boldsymbol{y}$ [3, 8]. We stress that we do not directly observe $\boldsymbol{x}$, but only indirectly and through the vector $\boldsymbol{y}$. Vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are statistically related via the PDF $\text{pr}(\boldsymbol{y}|\boldsymbol{x})$. Probability density functions $\text{pr}(\boldsymbol{x}|\boldsymbol{\theta})$ and $\text{pr}(\boldsymbol{y}|\boldsymbol{x})$ allow us to write

$$\text{pr}(\boldsymbol{y}|\boldsymbol{\theta}) = \int pr(\boldsymbol{y}|\boldsymbol{x})\; pr(\boldsymbol{x}|\boldsymbol{\theta})\; d\boldsymbol{x}, \tag{7}$$

in which $\text{pr}(\boldsymbol{y}|\boldsymbol{\theta})$ is the PDF of the observed data $\boldsymbol{y}$ given the parameter $\boldsymbol{\theta}$. Eq. (7) above takes into account two separate "mechanisms" that, when concatenated, produce a sample $\boldsymbol{y}$ from the value of $\boldsymbol{\theta}$. The first mechanism produces the complete data $\boldsymbol{x}$ according to $\text{pr}(\boldsymbol{x}|\boldsymbol{\theta})$, while the second mechanism samples $\text{pr}(\boldsymbol{y}|\boldsymbol{x})$ to produce the incomplete data $\boldsymbol{y}$.

MLE solves the estimation problem by finding the vector $\boldsymbol{\theta}$ that maximized the *likelihood* $\text{L}(\boldsymbol{\theta}; \boldsymbol{y})$ for observed data $\boldsymbol{y}$. Mathematically, this concept is formalized as:

$$\widehat{\boldsymbol{\theta}}_{ML} = \text{argmax}_{\boldsymbol{\theta}}\; \text{pr}(\boldsymbol{y}|\boldsymbol{\theta}) = \text{argmax}_{\boldsymbol{\theta}}\; \text{L}(\boldsymbol{\theta}; \boldsymbol{y}) \tag{8}$$

in which the "hat" symbol denotes an estimated quantity, and we have defined the likelihood as:

$$\mathrm{L}(\boldsymbol{\theta};\boldsymbol{y}) = \mathrm{pr}(\boldsymbol{y}|\boldsymbol{\theta}). \tag{9}$$

Likelihood $\mathrm{L}(\boldsymbol{\theta};\boldsymbol{y})$ has to be interpreted as a function of $\boldsymbol{\theta}$ for fixed (measured) $\boldsymbol{y}$. In Eq. (8), we used "$\mathrm{argmax}_{\boldsymbol{\theta}}\,\mathrm{L}(\boldsymbol{\theta};\boldsymbol{y})$" to denote the value of $\boldsymbol{\theta}$ that maximizes the likelihood. Because $\boldsymbol{y}$ is the result of a noisy measurement, the actual value of $\boldsymbol{y}$ in Eq. (8) will change if the measurement is repeated. In other words, $\boldsymbol{y}$ is a random quantity, and this implies that the ML estimate $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ is random as well.

Alternatively, $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ can be calculated by rewriting Eq. (8) as

$$\widehat{\boldsymbol{\theta}}_{ML} = \mathrm{argmax}_{\boldsymbol{\theta}}\ln\left[\mathrm{pr}(\boldsymbol{y}|\boldsymbol{\theta})\right] = \mathrm{argmax}_{\boldsymbol{\theta}}\,\ell(\boldsymbol{\theta};\boldsymbol{y}), \tag{10}$$

in which we have introduced the log-likelihood [8]

$$\ell(\boldsymbol{\theta};\boldsymbol{y}) = \ln\left[\mathrm{L}(\boldsymbol{\theta};\boldsymbol{y})\right]. \tag{11}$$

Because the logarithm is a strictly monotonic function, the expression in Eq. (10) is equivalent to the one in Eq. (8). Often, the log-likelihood $\ell(\boldsymbol{\theta};\boldsymbol{y})$ is numerically easier to calculate with a computer than the likelihood $\mathrm{L}(\boldsymbol{\theta};\boldsymbol{y})$.

3.2. Properties of ML estimates

Maximum-likelihood estimates have many desirable properties. Some of these properties are summarized below.

- **Asymptotic efficiency.** If $\boldsymbol{y}$ represents a set of repeated independent and identically distributed samples $y_1, \ldots y_M$, asymptotic efficiency of MLE implies that, as M increases, the variance of each component of $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ converges to the smallest possible value, which is given by the *Cramér-Rao lower bound* [12, 13].
- **Functional invariance.** Assume the ML estimate of a parameter vector $\boldsymbol{\theta}$ is $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ and consider a function $u(\boldsymbol{\theta})$ of $\boldsymbol{\theta}$. We can identify $u(\boldsymbol{\theta})$ with the parameter vector $\boldsymbol{\mu}$, and we can consider a maximum-likelihood estimate $\widehat{\boldsymbol{\mu}}_{\mathrm{ML}}$ of $\boldsymbol{\mu}$. Then [14]

  $$\widehat{\boldsymbol{\mu}}_{\mathrm{ML}} = u\left(\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}\right). \tag{12}$$

  This equation shows that the property of being a maximum-likelihood estimate is preserved if we consider a function of the maximum-likelihood estimate itself.
- **Sufficiency.** Any quantity $T(y_1, \ldots, y_M)$ calculated from samples $y_1, \ldots, y_M$ and used to estimate an unknown parameter vector $\boldsymbol{\theta}$ is said to be a *sufficient statistic* for $y_1, \ldots, y_M$ if no other quantity that can be calculated from the same samples would provide additional information regarding the value of the parameter vector $\boldsymbol{\theta}$. In simple terms, a sufficient statistic is a function of the samples $y_1, \ldots, y_M$ that "compresses" them without losing any

information about $\boldsymbol{\theta}$. Sufficiency for a maximum-likelihood estimate $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ can be stated by saying that $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ is a function of a sufficient statistic for $\boldsymbol{\theta}$ [15].

- **Consistency.** Consistency of an estimator regards the behavior of the estimator as the sample size M increases. Consider the case in which $\boldsymbol{y}$ is a set of repeated independent and identically distributed samples $y_1, \ldots, y_M$. It is possible to show that, when the range of the elements of $\boldsymbol{y} = (y_1, \ldots, y_M)$ does not dependent on the parameter vector $\boldsymbol{\theta}$, there exists a maximum-likelihood estimate $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ that, as M increases, converges in probability to the true value of the parameter vector. A consistent maximum-likelihood estimate is unique [16].
- **Asymptotic normality.** Because the ML estimate $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ of $\boldsymbol{\theta}$ is a random variable, it makes sense to consider its probability density function. As the sample size M increases, the probability density function of $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ converges to the probability density function of a normally distributed random variable with mean equal to the true value of the parameter we want to estimate and covariance matrix equal to the inverse of the *Fisher information matrix* [17].

4. Graphics processing units and CUDA

Driven by the insatiable demand for real-time rendering in gaming and entertainment, graphics processing units (GPUs) have become highly parallel devices capable of running general-purpose code. Newer products that offer an ever-increasing amount of computational power are constantly introduced in the market at very competitive prices.

Programming languages have been developed to harness the parallel capabilities of GPU devices. The most widespread language for GPU programming is called compute unified device architecture (CUDA), which was introduced in 2006 by NVIDIA. Due to its similarity to C, CUDA has rapidly become the *de facto* programming language for GPUs.

4.1. The CUDA programming model

In CUDA, the GPU is usually referred to as the *device* and the computer that hosts it is referred to as the *host*. Many GPU devices can be installed in the same host, and it is not uncommon to have systems with more than one GPU device. Each GPU device has its own memory, which we refer to as *device memory*. In CUDA, it is also common to refer to the memory installed in the host as *host memory*. CUDA provides library functions to allocate blocks of memory in device memory and to transfer blocks of data from host memory to device memory and vice versa. As shown in **Figure 3**, a typical GPU device includes some GPU cores (ranging in number from a few hundreds to a few thousands) and some control logic.

Programmers access the parallel capabilities of a CUDA-enabled device by writing *kernels*, which are pieces of code that look very similar to regular C functions. In CUDA, a kernel is run in parallel on many different GPU cores. A kernel in execution is referred to as a *thread*. Threads are grouped into blocks, which can be 1D, 2D or 3D, and blocks are grouped into a grid. Grids can be 1D, 2D or 3D. The size and dimensionality of blocks and grids are decided

http://dx.doi.org/10.5772/intechopen.72151

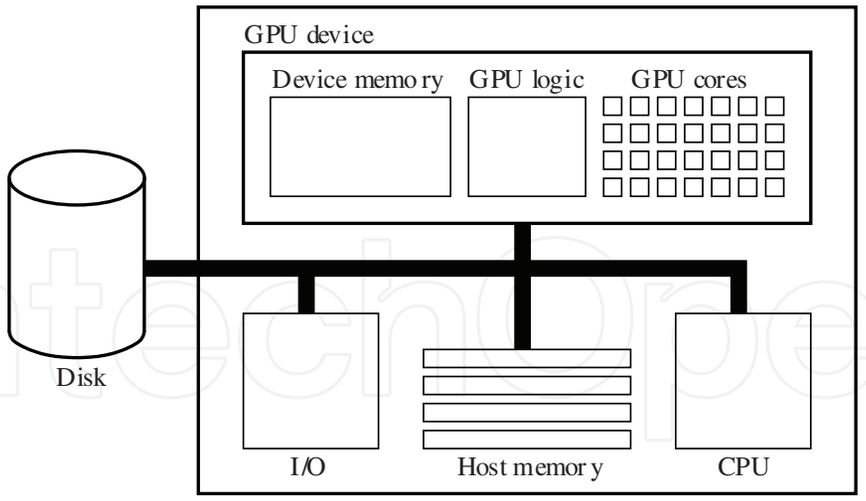

Figure 3. Diagram of a computer equipped with a GPU device (adapted from [3]).

by the programmer via an *execution configuration*, which is also used to call a kernel and instruct the GPU hardware to execute threads.

In a GPU device, thread scheduling is extremely efficient and it is performed by the hardware and without the intervention of the programmer. To improve performance, the hardware also suspends execution for threads that are waiting for completion of memory transfers between device memory and GPU registers. When that happens, the hardware selects for execution threads that already have data to be processed. The programmer is typically unaware of what threads are running at any given time, nor does he know what kernel instruction is being executed by a specific thread. In other words, the programmer cannot rely on any particular scheduling order of GPU threads. There are, however, situations in which it is necessary to ensure that a block of threads has reached a certain instruction in a kernel before all the threads in the block can continue. In CUDA, this is accomplished via *synchronization barriers*.

Synchronization barriers are often used when threads have to exchange data with each other via shared variables. Without any synchronization mechanism, a thread will not be able to know if the content of a shared variable has already been written by a cooperating thread. Synchronization barriers solve this problem by suspending thread execution until *all* the threads in the same block have reached a synchronization barrier. In CUDA, synchronization barriers are allowed only among the threads in the same block.

GPU devices are equipped with different memory spaces. This includes *global memory* (which is used to share data between the host and the device) as well as *shared memory*. While global memory is rather slow and physically separated from the GPU cores, shared memory is much faster and it is built on the same chip as the GPU cores. Threads use shared memory to efficiently share data among them.

Another type of memory space available in a GPU device is *texture memory*. As the name suggests, texture memory has been designed to speed up and facilitate 3D rendering in computer games and computer-generated scenes. This is the reason why texture memory supports unique features including hardware-based on-the-fly interpolation of texture data.

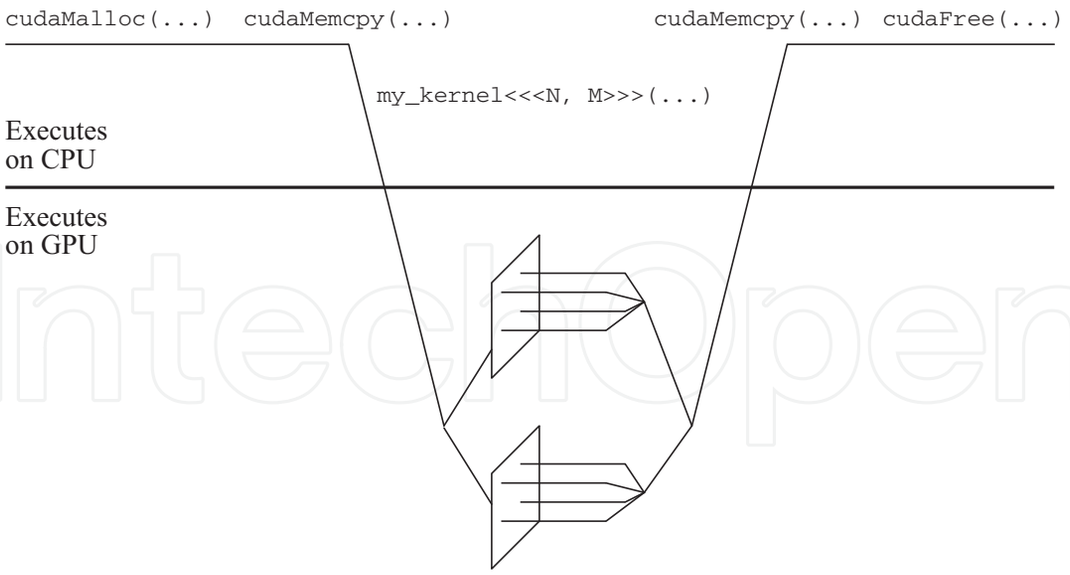

Figure 4. Workflow of a CUDA application (adapted from [3]).

4.2. Workflow of a CUDA application

The basic steps that are needed to execute a kernel are summarized in **Figure 4**.

In a typically CUDA application, one or more blocks of device memory are allocated by the host via the **`cudaMalloc(...)`** CUDA library function. The host then copies input data from host memory via one or more **`cudaMemcpy(...)`** function calls. Kernel execution is started with a call of the form **`my_kernel < <<N, M>> > (...)`**, in which **`my_kernel`** is the name of the kernel, **`N`** is the grid size and **`M`** is the block size. Parameters, such as pointers to device memory, are passed to the kernel as parameters enclosed in parentheses. Once *all* the threads have finished executing, the control returns to the CPU. Results can be copied from device memory to host memory via one or more calls to **`cudaMemcpy(...)`**. Finally, device memory that was previously allocated is released via the **`cudaFree(...)`** call.

The CUDA environment automatically defines read-only built-in variables that can only be used in a kernel. These variables include **`blockIdx`**, **`blockDim`** and **`threadIdx`**. The variable **`threadIdx`** enumerates threads within each block in ascending order and starting from 0. Similarly, **`blockIdx`** enumerates blocks within the grid. The size of each block is contained in the variable **`blockDim`**. Built-in variables are used by the programmer to calculate which element(s) of the input array(s) a thread has to work on, or where in device memory a result has to be stored.

5. Algorithms for photon-counting detectors

To make our discussion more concrete, we begin this section by considering a GPU algorithm for maximum-likelihood estimation (MLE) of position of interaction of gamma-ray photons in an Anger camera. We then comment on ways to adapt our algorithm to other cases, including photon-counting and photon-processing detectors (Section 6).

We showed in Section 2.1 that digitized PMT signals $g_1, \ldots, g_K$ obey Poisson statistics and we denoted the means of $g_1, \ldots, g_K$ as $\overline{g}_1(\boldsymbol{R}, E), \ldots, \overline{g}_K(\boldsymbol{R}, E)$, respectively. If photon energy E is known, the likelihood for the estimation of position $\boldsymbol{R}$ under the assumption of Poisson noise is written as:

$$\mathrm{L}\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right) = \prod_{k=1}^{K} \frac{\left[\overline{g}_k(\boldsymbol{R}, E)\right]^{g_k} \exp\left[-\overline{g}_k(\boldsymbol{R}, E)\right]}{g_k!}. \tag{13}$$

Functions $\overline{g}_1(\boldsymbol{R}, E), \ldots, \overline{g}_K(\boldsymbol{R}, E)$ are called mean detector response functions (MDRFs) and they can be either measured, derived analytically or estimated via simulation codes. Using Eq. (13), an ML estimate $\widehat{\boldsymbol{R}}_{\mathrm{ML}} = \left(\widehat{x}_{\mathrm{ML}}, \widehat{y}_{\mathrm{ML}}\right)$ of $\boldsymbol{R} = (x, y)$ can be found as:

$$\widehat{\boldsymbol{R}}_{\mathrm{ML}} = \mathrm{argmax}_{\boldsymbol{R}} \mathrm{L}\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right). \tag{14}$$

Equivalently, we can consider the logarithm of $\mathrm{L}\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right)$ in the maximization step and write:

$$\widehat{\boldsymbol{R}}_{\mathrm{ML}} = \mathrm{argmax}_{\boldsymbol{R}} \sum_{k=1}^{K} \left\{ g_k \log\left[\overline{g}_k(\boldsymbol{R}, E)\right] - \overline{g}_k(\boldsymbol{R}, E) \right\}, \tag{15}$$

in which we omitted the $\log\left(g_k!\right)$ term as it does not depend on $\boldsymbol{R}$ and, therefore, it will not affect the estimate $\widehat{\boldsymbol{R}}_{\mathrm{ML}}$.

The algorithm we present here uses the fact that, for fixed $g_1, \ldots, g_K$ and E, the log-likelihood $\ell\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right) = \log \mathrm{L}\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right)$ is a smooth function of $\boldsymbol{R}$. Hence, maximum-likelihood estimate $\widehat{\boldsymbol{R}}_{\mathrm{ML}}$ can be searched for in an iterative fashion by first evaluating $\ell\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right)$ over a coarse grid of S_x-by-S_y points that uniformly spans the whole detector space. The point of the grid that maximizes $\ell\left(\boldsymbol{R}; g_1, \ldots, g_K, E\right)$ is used in the next iteration as the center of a new grid smaller than the previous one by a factor $\alpha > 1$. This process is repeated M times. We refer this algorithm as the *contracting grid algorithm* [18, 19].

Figure 5 shows pseudocode for a possible GPU implementation of the contracting grid algorithm. We used superscripts to make it clear on which memory space a given variable is stored. Variables with no superscript will denote either numerical constants (such as the contracting factor α) or local variables, typically stored in GPU registers.

The algorithm of **Figure 5** assumes that an array of R PMT sample vectors $\boldsymbol{g}_0, \ldots \boldsymbol{g}_{R-1}$ is available. These data are stored in device memory and we decided to use in our GPU implementation a grid of size $R \times 1 \times 1$ with 2D blocks of size $S_x \times S_y$. This thread hierarchy follows naturally from the data we have to process and how we process them. In fact, the block index is used to index one of the $\boldsymbol{g}_0, \ldots \boldsymbol{g}_{R-1}$ vectors, while the 2D thread index is used to identify a point of the contracting grid (of size $S_x \times S_y$).

```
function 2D-M L( g_0^[global], ..., g_{R-1}^[global])
    i ← threadIdx.x
    j ← threadIdx.y
    if (i = 0) ∧ (j = 0) then
        r ← blockIdx.x
        g^[shared] ← g_r^[global]
        x_⋆^[shared] ← (a + b)/2
        y_⋆^[shared] ← (c + d)/2
        Δ_x^[shared] ← (b − a)/S_x
        Δ_y^[shared] ← (d − c)/S_y
    end if
    __syncthreads
    for m = 0, ..., M − 1 do
        x ← x_⋆^[shared] + [i − (S_x − 1)/2] · Δ_x^[shared]
        y ← y_⋆^[shared] + [j − (S_y − 1)/2] · Δ_y^[shared]
        ℓ_{i,j}^[shared] ← 0
        for k = 0, ..., K − 1 do
            ḡ_k ← tex2DLayered(ḡ^[texture], x, y, k)
            if (g_k^[shared] ≠ 0) ∨ (ḡ_k ≠ 0) then
                ℓ_{i,j}^[shared] ← ℓ_{i,j}^[shared] + g_k^[shared] · log(ḡ_k) − ḡ_k
            end if
        end for
        __syncthreads
        if (i = 0) ∧ (j = 0) then
            ℓ_max ← −∞
            for i_test = 0, ..., S_x − 1 do
                for j_test = 0, ..., S_y − 1 do
                    if ℓ_max < ℓ_{i_test, j_test}^[shared] then
                        ℓ_max ← ℓ_{i_test, j_test}^[shared]
                        i_max^[shared] ← i_test
                        j_max^[shared] ← j_test
                    end if
                end for
            end for
            Δ_x^[shared] ← Δ_x^[shared] / a
            Δ_y^[shared] ← Δ_y^[shared] / a
        end if
        __syncthreads
        if (i = i_max^[shared]) ∧ (j = j_max^[shared]) then
            x_⋆^[shared] ← x
            y_⋆^[shared] ← y
        end if
        __syncthreads
    end for
    if (i = 0) ∧ (j = 0) then
        r ← blockIdx.x
        x̂_r^[global] ← x_⋆^[shared]
        ŷ_r^[global] ← y_⋆^[shared]
    end if
end function
```

Figure 5. GPU pseudocode for ML estimation via a contracting-grid search algorithm.

Our GPU implementation uses shared memory to either store data that are used multiple times during thread execution (this would be the case, e.g., of PMT data vector g_r) or to share common variables among all the threads in the same block. Each thread in a block calculates

the likelihood $\ell\left(x, y; \boldsymbol{g}^{[\text{shared}]}\right)$ for one of the points in the contacting grid and shares the value of the likelihood among all the threads in the same block.

MDRF data (previously estimated via simulation codes [4]) are stored in a 2D *layered* texture and used during the calculation of the log-likelihood $\ell\left(x, y; \boldsymbol{g}^{[\text{shared}]}\right)$ (denoted as $\ell_{i,j}^{[\text{shared}]}$ in the pseudocode). MDRF data are transparently interpolated by the hardware during texture fetching. Moreover, we set texture boundary conditions so that, should the point (x, y) fall outside the detector's entrance face, $\overline{g}_k(x, y)$ would evaluate to 0. Physically, this can be interpreted as no PMT signals being produced for a gamma-ray "interaction" outside the detector's entrance face.

Besides code speed and clarity, a layered texture makes our code extremely flexible. By changing S_x, S_y and/or α, it is possible to change the size of the contracting grid or its contracting factor to find the desired trade-off between speed and estimation accuracy.

5.1. Comments and applications to photon counting

The algorithm we discussed above was specifically designed for gamma-ray data and it uses calibration data in the form of mean detector response functions (MDRFs). The output of the algorithm is a list of positions in the form $\left\{\left(\widehat{x}_0, \widehat{y}_0\right), \ldots, \left(\widehat{x}_{R-1}, \widehat{y}_{R-1}\right)\right\}$. This list can directly be fed to an algorithm for list-mode image reconstruction. Implementation details and results are reported in [20]. Common practice, however, is to *bin* the list-mode data and count the number of points $\left(\widehat{x}_r, \widehat{y}_r\right)$ that fall within each bin. As we argue in [21], one drawback of this step is that it introduces some error, as all the points within each bin are represented with a single point location.

The algorithm we presented in **Figure 5** is one example of a contracting grid algorithm for maximum-likelihood estimation. The main assumption we made was that the likelihood $\mathrm{L}(\boldsymbol{g}\,|\boldsymbol{\theta})$ (or its logarithm) is a smooth function of $\boldsymbol{\theta}$, the vector of parameters we want to estimate. This is true for many estimation problems. Therefore, the algorithm of **Figure 5** provides a general pattern for the implementation of maximum-likelihood estimation on a GPU device.

6. Photon-processing detectors and algorithms

For each photon-absorption event in a detector, there are many parameters we can consider. These parameters include photon position $\boldsymbol{R}$ with respect to a plane or a reference point, direction of propagation $\vec{s}$ and energy E the photon deposited in the detector. Depending on the application (e.g., single-photon emission computed tomography for 4D angiography or coincidence detection in positron emission tomography), we might also need to consider the time t the photon impinged on the detector. Finally, some imaging techniques (such as two-photon quantum imaging) do require measurements of quantum mechanical parameters, one example being quantum spin.

6.1. Mathematical description

We refer to a set of photon parameters as an *attribute vector,* and we denote it as $\boldsymbol{A}$. Hence, depending on the application, an attribute vector might have five or more components. We denote the number of components of $\boldsymbol{A}$ as N. Because of noise, it is not possible to estimate exactly the components of $\boldsymbol{A}$ and we use the notation $\widehat{\boldsymbol{A}}$ to denote an estimated attribute vector.

We define a *photon-processing* detector as any imaging device that [9]:

- uses a gain mechanism (such as an image intensifier) to obtain multiple measurements (e.g., multiple pixel values) for each absorbed photon;
- uses these measurements to perform maximum-likelihood estimation of photon attribute vector $\widehat{\boldsymbol{A}}_j$, for$j = 1, \ldots, J$;
- stores the estimated attributes at full precision as a list $\widehat{\boldsymbol{\mathcal{A}}} = \left\{\widehat{\boldsymbol{A}}_1, \ldots, \widehat{\boldsymbol{A}}_J\right\}$ and without performing any binning.

Photon-processing detectors are fundamentally different than photon-counting detectors. While photon-counting detectors only consider photon position and record the number of photons (or charged particles) that fall within each bin over a predetermined amount of time, photon-processing detectors use maximum-likelihood to estimate a wide range of attributes and retain all the estimated information at full precision as the list $\widehat{\boldsymbol{\mathcal{A}}} = \left\{\widehat{\boldsymbol{A}}_1, \ldots, \widehat{\boldsymbol{A}}_J\right\}$.

The full information from a photon-processing detector is retained if we simply store the N estimated attributes for each of J photons as the list $\widehat{\boldsymbol{\mathcal{A}}}$, but an equivalent construction as a random point process in attribute space offers new theoretical insights. From $\widehat{\boldsymbol{\mathcal{A}}}$, we introduce this point process as [3, 9]

$$u(\boldsymbol{A}) = \sum_{j=1}^{J} \delta\left(\boldsymbol{A} - \widehat{\boldsymbol{A}}_j\right), \tag{16}$$

where $\delta(\ldots)$ is the N-dimensional Dirac delta function. The mean of the point process $u(\boldsymbol{A})$ is obtained by averaging over the statistics of each of the attribute vectors $\widehat{\boldsymbol{A}}_1, \ldots, \widehat{\boldsymbol{A}}_J$ and then over J itself for a given object $\boldsymbol{f}$. This calculation gives a function $\overline{u}\left(\boldsymbol{A} \mid \boldsymbol{f}\right)$ of $\boldsymbol{A}$ for fixed $\boldsymbol{f}$. This function can be regarded as a vector $\overline{\boldsymbol{u}}\left(\boldsymbol{f}\right)$ in the Hilbert space $\mathbb{L}_2\left(\mathbb{R}^N\right)$, which is the vector space of square-integrable functions of N real variables. As shown in [22, 23], we can introduce the linear operator $\mathcal{L}$ that maps the object $\boldsymbol{f}$ (belonging to the infinite-dimensional Hilbert space $\mathbb{L}_2\left(\mathbb{R}^3\right)$) to $\overline{\boldsymbol{u}}\left(\boldsymbol{f}\right)$. In symbols,

$$\overline{\boldsymbol{u}}\left(\boldsymbol{f}\right) = \mathcal{L}\boldsymbol{f}. \tag{17}$$

A similar expression but for photon-counting detectors is:

$$\overline{\boldsymbol{g}}\left(\boldsymbol{f}\right) = \mathcal{H}\boldsymbol{f}, \tag{18}$$

in which the vector $\overline{\boldsymbol{g}}(\boldsymbol{f})$ is the mean over many realizations of the vector $\boldsymbol{g}$. Vector $\overline{\boldsymbol{g}}(\boldsymbol{f})$ belongs to the Euclidian vector space $\mathbb{E}^M$, which is the space of all M-dimensional vectors. We refer to $\mathcal{H}$ as a continuous-to-discrete operator [8] as it maps the function $f(\boldsymbol{r})$ of continuous variable $\boldsymbol{r}$ to a discrete vector $\overline{\boldsymbol{g}}(\boldsymbol{f})$ with M components. On the other hands, $\mathcal{L}$ is a continuous-to-continuous operator as it maps $f(\boldsymbol{r})$ to the function $\overline{u}(\boldsymbol{A} \mid \boldsymbol{f})$ of continuous variable $\boldsymbol{A}$ [8].

The key difference between $\mathcal{H}$ and $\mathcal{L}$ is that $\mathcal{H}$ must necessarily have a nontrivial null space, as it maps vectors in an infinite-dimensional Hilbert space to vectors in a finite-dimensional vector space. This means that for any imaging system that produces photon-counting data, there exist nonzero objects $\boldsymbol{f}_{\text{null}}$ that, on average, do not produce any data. Equivalently, we can say that there exist two objects $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ with $\boldsymbol{f}_1 \neq \boldsymbol{f}_2$ for which $\mathcal{H}\boldsymbol{f}_1 = \mathcal{H}\boldsymbol{f}_2$. A continuous-to-continuous operator—such as the operator $\mathcal{L}$ defined above—maps an object in an infinite-dimensional Hilbert space to another infinite-dimensional Hilbert space. Therefore, the same dimensionality analysis we considered for $\mathcal{H}$ does not apply to $\mathcal{L}$. In fact, $\mathcal{L}$ might allow a lower dimensional null space than $\mathcal{H}$ for the same imaging system [21].

6.2. Relationship to radiometry and the Boltzmann transport equation

The word "radiometry" refers to a set of techniques used in optics to describe and calculate the distribution of light. An important concept used in radiometry is that of radiance, denoted as $L\left(\boldsymbol{r}, \vec{\boldsymbol{s}}\right)$, which is a function that describes the radiant flux in an optical system as a function of three-dimensional spatial position $\boldsymbol{r}$ and direction $\vec{\boldsymbol{s}}$. Since the radiant flux is measured in Watts, the units of $L\left(\boldsymbol{r}, \vec{\boldsymbol{s}}\right)$ are Watts per square meter per steradian, or W/ (m^2· ster) [8]. From the radiance, other important radiometric quantities can be calculated. This includes the irradiance (power per unit area), radiant intensity (power per unit solid angle) and radiant flux (power).

Spectral dependence can be introduced in the basic definition of radiance by considering radiance per unit wavelength λ, which we denote as $L_\lambda\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, \lambda\right)$. The units of $L_\lambda\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, \lambda\right)$ are W/ (m^2· ster· nm), provided that the units of wavelength are nanometers (nm). Spectral radiance can also be measured in photon units by first expressing wavelength in terms of energy ($E = hc/\lambda$, in which h is Planck's constant and c is the speed of light) and then by dividing $L_\lambda\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, \lambda\right)$ by the energy of a photon. We denote this new quantity as $L_{p,E}\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, E\right)$, and its units are (photons/s)/(m^2· ster). Finally, we can consider a time-dependent spectral photon radiance, and we denote this function as $L_{p,E}\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, E, t\right)$.

In radiometry, the Boltzmann transport equation (BTE) allows to calculate $L_{p,E}\left(\boldsymbol{r}, \vec{\boldsymbol{s}}, E, t\right)$ at any point inside an optical system by taking into account absorption, emission, scattering and propagation of light. In its most general form, the BTE is written as [8]:

$$\frac{\partial L_{p,E}}{\partial t} = \left[\frac{\partial L_{p,E}}{\partial t}\right]_{\text{abs}} + \left[\frac{\partial L_{p,E}}{\partial t}\right]_{\text{emiss}} + \left[\frac{\partial L_{p,E}}{\partial t}\right]_{\text{sc}} + \left[\frac{\partial L_{p,E}}{\partial t}\right]_{\text{prop}}. \tag{19}$$

Each term on the right-hand side can be worked out explicitly [8, 9] to obtain:

$$\frac{\partial L_{p,E}}{\partial t} = -c_m \mu_{\text{tot}} L_{p,E} + c_m \Xi_{p,E} + \mathcal{K} L_{p,E} - c_m \vec{s} \cdot \nabla L_{p,E}, \tag{20}$$

where c_m is the speed of light in the medium, μ_{tot} is the total attenuation coefficient (with contributions from both absorption and scattering processes), $\mathcal{K}$ is an integral operator describing the angular and energy dependence of the scattering, and $\Xi_{p,E}$ describes any light source. In general, the function $\Xi_{p,E}$ depends on $\boldsymbol{r}$, $\vec{s}$, E and t. If the light source is isotropic (i.e., $\Xi_{p,E}\left(\boldsymbol{r}, \vec{s}, E, t\right)$ does not depend on $\vec{s}$) and independent of time, we can write

$$\Xi_{p,E}\left(\boldsymbol{r}, \vec{s}, E, t\right) = \frac{1}{4\pi} f(\boldsymbol{r}, E), \tag{21}$$

where the 4π term (units: ster) accounts for integration of a solid angle over a sphere. Under these hypotheses, a steady-state solution to Eq. (20) is found by setting the partial derivative $\partial L_{p,E}/\partial t$ to zero. The result is

$$c_m \mu_{tot} L_{p,E} - \mathcal{K} L_{p,E} + c_m \vec{s} \cdot \nabla L_{p,E} = \frac{c_m}{4\pi} f, \tag{22}$$

which can be further rewritten in operator form as

$$\frac{4\pi}{c_m} \mathcal{B}\, \boldsymbol{L}_{p,E} = \boldsymbol{f}, \tag{23}$$

provided that

$$\mathcal{B} = \mathrm{c_m} \mu_{\text{tot}} - \mathcal{K} + \mathrm{c_m} \vec{\mathrm{s}} \cdot \nabla. \tag{24}$$

We refer to $\mathcal{B}$ as the Boltzmann operator. If we insert Eq. (23) into Eq. (17), we get

$$\overline{\boldsymbol{u}}(\boldsymbol{f}) = \frac{4\pi}{c_m} \mathcal{L}\, \mathcal{B}\, \boldsymbol{L}_{p,E}, \tag{25}$$

which describes a practical way to obtain the function $\overline{\boldsymbol{u}}(\boldsymbol{f})$ from knowledge of the radiance function $L_{p,E}\left(\boldsymbol{r}, \vec{s}, E\right)$ inside an optical system and the Boltzmann operator $\mathcal{B}$.

6.3. Particle-processing detectors

An example of a particle-processing detector for beta particles is shown in **Figure 6**. This detector includes two layers of ultrathin phosphor foils separated by an air gap, an image intensifier, a high numerical aperture lens system and a light sensor [1, 7].

In **Figure 6**, an incoming beta particle interacts with a layer of phosphor (just a few microns thick) at location $\boldsymbol{r}_1 = \left(x_1, y_1\right)$ where it deposits some of its energy, which the layer of phosphor gives off as a flash of visible light. The particle further propagates and interacts at $\boldsymbol{r}_2 = \left(x_2, y_2\right)$

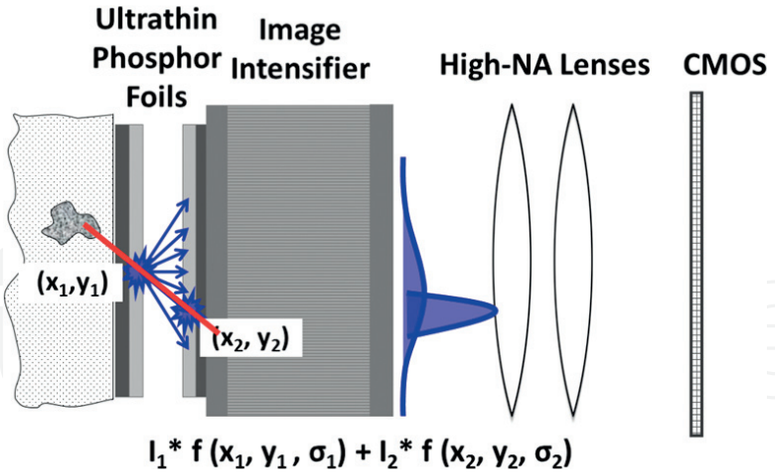

Figure 6. A particle-processing detector for beta particle (adapted from [1]).

with a second phosphor layer, thus producing a flash of visible light here as well. Light flashes produced at each layer get amplified by an image intensifier and imaged onto a sensor. The flash of light generated at the first layer spreads out considerably as it propagates through the air gap, thus resulting in a much broader signal on the sensor than that corresponding to the flash of light generated at the second layer. This is used to determine at which layer each flash of light was generated. The direction $\vec{s}$ of the particle and its position at either phosphor foil can be estimated from the two interaction positions and the distance between the two foils (assumed known). If the particle's residual energy E is of interest, the second phosphor foil can just be replaced by a thick scintillator, so that the particle is stopped.

A CUDA algorithm for maximum-likelihood estimation of position and particle direction for the setup of **Figure 6** has been developed by Garrett Hinton at the Center for Gamma-Ray Imaging, University of Arizona. Following the same approach outlined in Section 5, the algorithm uses a four-dimensional contracting grid to simultaneously estimate location $\boldsymbol{r}_1 = (x_1, y_1)$ and $\boldsymbol{r}_2 = (x_2, y_2)$ from an image frame collected with an ultrafast CMOS camera. Experimental setup and preliminary results have been presented in [1, 7].

7. Summary and applications

This chapter provided a general overview of detector technology and algorithms for photon counting and photon processing. We started by describing detectors suitable for photon counting and photon processing. Statistical models for the data produced by these detectors were presented. We then introduced maximum-likelihood estimation and discussed its properties. We emphasized that a maximum-likelihood estimate is any parameter that maximizes the likelihood function given the detector output, and we pointed out that the likelihood function is the probability density function of the detector output conditioned on the parameter being estimated.

We then commented on graphics processing units (GPUs) and the CUDA programming environment. Through CUDA-like pseudocode, we provided a maximum-likelihood algorithm for estimation of position of interaction for gamma-ray cameras. This algorithm used a

contracting-grid approach to find a maximum of the likelihood function. Our approach heavily relied upon GPU textures to quickly retrieve calibration data. The same approach is applicable to many estimation problems.

Photon-processing detectors were introduced and defined as any system that collects multiple measurements to perform maximum-likelihood estimation of multiple event parameters (such as position, direction and energy), which are stored as a list and in full precision in the memory of a computer. The same data can also be represented as a point process, and we introduced a linear operator that maps the object being imaged to the mean of this point process. We used a dimensionality analysis to describe the advantages of photon-processing detectors over photon-counting detectors.

Particle-processing detectors are a variation of photon-processing detector. As an emerging technology, particle-processing detectors will find applications in many fields, one of them being medical imaging. In a new technique, called charged-particle emission tomography (CPET), particle-processing detectors are being evaluated for 3D *in vivo* imaging with alpha and beta particles [1, 7]. Like photon-processing detectors, particle-processing detectors convey a larger amount of information than conventional detectors for charged particles. This enables high-resolution 3D reconstruction of the distribution of radionuclides emitting charged particles without the need to kill an animal to image a collection of thinly sliced tissue sections.

Drug development will take advantage of CPET to determine drug pharmacokinetics, 3D transduction across cell membranes and targeting to tissues of interest. In the development of internal radioimmunotherapy, CPET imaging can be used to collect data on the 3D heterogeneous distributions of targeting molecules and in the estimation of delivered radiation dose. Finally, CPET will likely become a valuable technique in the emerging fields of personalized medicine and theranostics, in which diagnostics and therapy are combined in an attempt to avoid the "one-size-fits-all" approach to treatment that is often successful for some patients but not for others.

Acknowledgements

This chapter has been supported by National Institutes of Health (Grants R01 EB000803 and P41 EB002035).

Author details

Luca Caucci[1]*, Yijun Ding[2] and Harrison H. Barrett[1,3]

*Address all correspondence to: caucci@email.arizona.edu

1 Department of Medical Imaging, University of Arizona, Tucson, AZ, United States

2 Department of Radiation Oncology, University of Colorado Denver, Denver, United States

3 College of Optical Sciences, University of Arizona, Tucson, AZ, United States

References

[1] Yijun Ding. Charged-Particle Emission Tomography [dissertation]. Tucson, AZ; 2016.

[2] Anger HO. Scintillation Camera. Review of Scientific Instruments. 1958;**29**(1):27-33

[3] Luca Caucci. Task Performance with List-Mode Data [dissertation]. Tucson, AZ; 2012

[4] Hunter WCJ, Barrett HH, Furenlid LR. Calibration method for ML estimation of 3D interaction position in a thick gamma-ray detector. IEEE Transactions on Nuclear Science. 2009;**56**(1):189-196

[5] Llopart X, Campbell M, Dinapoli R, San Segundo D, Pernigotti E. Medipix2: A 64-k pixel readout Chip with 55-μm. IEEE Transactions on Nuclear Science. 2002;**49**(5):2279-2283

[6] Bouchami J, Gutiérrez A, Houdayer A, Jakůbek J, Lebel C, Leroy C, Macana J, Martin JP, Platkevič M, Pospíši S, Teyssierl C. Study of the charge sharing in silicon pixel detector by means of heavy ionizing particles interacting with a Medipix2 device. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2011;**633**(Supplement 1):S117-S120

[7] Ding Y, Caucci L, Barrett HH. Charged-particle emission tomography. Medical Physics. 2017;**44**(6):2478-2489

[8] Harrison H. Barrett, Kyle J. Myers. Foundations of Image Science. Hoboken, NJ: Wiley-Interscience; 2004

[9] Caucci L, Myers KJ, Barrett HH. Radiance and photon noise: Imaging in geometrical optics, physical optics, quantum optics and radiology. Optical Engineering. 2016;**55**(1):013102

[10] Miller BW, Gregory SJ, Fuller ES, Barrett HH, Barber HB, Furenlid LR. The iQID camera: An ionizing-radiation quantum imaging detector. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2014;**767**:146-152

[11] Aldrich J. R. A. Fisher and the making of maximum likelihood 1912–1922. Statistical Science. 1997;**12**(3):162-176

[12] Cramér H. Mathematical Methods of Statistics. Princeton, NJ: Princeton University Press; 1956

[13] Rao CR. Information and the accuracy attainable in the estimation of statistical parameters. Bulletin of the Calcutta Mathematical Society. 1945;**37**:81-89

[14] Zehna PW. Invariance of maximum likelihood estimators. Annals of Mathematical Statistics. 1966;**37**(3):744

[15] Moore DS. Maximum likelihood and sufficient statistics. The American Mathematical Monthly. 1971;**78**(1):50-52

[16] Huzurbazar VS. The likelihood equation, consistency and the maxima of the likelihood function. Annals of Human Genetics. 1947;**14**(1):185-200

[17] Fisher RA. Theory of statistical estimation. Mathematical Proceedings of the Cambridge Philosophical Society. 1925;**22**(5):700-725

[18] Furenlid LR, Hesterman JY, Barrett HH. Real-time data acquisition and maximum-likelihood estimation for gamma cameras. In: 14th IEEE-NPSS Real Time Conference; 4–10 June 2005. Stockholm, Sweden; 2005. p. 498-501

[19] Hesterman JY, Caucci L, Kupinski MA, Barrett HH, Furenlid LR. Maximum-likelihood estimation with a contracting-grid search algorithm. IEEE Transactions on Nuclear Science. 2010;**57**(3):1077-1084

[20] Luca Caucci, William C. J. Hunter, Lars R. Furenlid, Harrison H. Barrett. List-mode MLEM Image Reconstruction from 3D ML Position Estimates. In: IEEE Nuclear Science Symposium Conference Record (NSS/MIC), 30 Oct–6 Nov. 2010. Knoxville, TN, USA; 2010. p. 2643-2647

[21] Luca Caucci, Abhinav K. Jha, Lars R. Furenlid, Eric W. Clarkson, Matthew A. Kupinski, Harrison H. Barrett. Image Science with Photon-Processing Detectors. In: IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 27 Oct–2 Nov, 2013. Seoul, South Korea; 2013. p. 1-7

[22] Andre Lehovich. List-mode SPECT Reconstruction Using Empirical Likelihood [dissertation]. Tucson, AZ; 2005

[23] Caucci L, Barrett HH. Objective assessment of image quality. V. Photon-counting detectors and list-mode data. Journal of the Optical Society of America A. 2012;**29**(6):1003-1016