Winter 12-2019

Event Reconstruction in the Advanced Particle-Astrophysics Telescope

Emily Ramey
Washington University in St. Louis

Washington University in St. Louis

McKelvey School of Engineering

Department of Computer Science and Engineering

Thesis Examination Committee:
Dr. Jeremy Buhler
Dr. James Buckley
Dr. Roger Chamberlain

Event Reconstruction in the Advanced Particle-Astrophysics Telescope

by

Emily Ramey

A thesis presented to the McKelvey School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

Master of Science

December 2019
Saint Louis, Missouri

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my advisors, Dr. Jeremy Buhler, Dr. James Buckley, and Dr. Roger Chamberlain, for their support of my research. I would also like to thank the National Science Foundation for funding my work. The following research is supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2018268910.

A special thanks goes to the many graduate students and distinguished faculty within the computer science and physics departments at Washington University who have reviewed this thesis and helped support the related research.

Emily Ramey

Washington University in Saint Louis
December 2019

Dedicated to my family, who made this work possible.

ABSTRACT OF THE THESIS

Event Reconstruction in the Advanced Particle-Astrophysics Telescope

by

Emily Ramey

Master of Science in Computer Science

Washington University in St. Louis, December 2019

Research Advisor: Dr. Jeremy Buhler

The Advanced Particle-Astrophysics Telescope (APT) is a concept for a gamma-ray space telescope operating in the keV to MeV energy range. Due to the nature of the telescope and the physics of detection, reconstructing initial photon trajectories can be very computationally complex. This is a barrier to the real-time detection of astrophysical transient phenomena such as Gamma Ray Bursts (GRBs), and a faster reconstruction algorithm is needed in order to effectively study them. In this project, we develop such an algorithm based on *Boggs & Jean* (2000)[2] and discuss the effects of certain algorithmic parameters on computational performance. For testing, we create a simple model of Compton scattering and generate data from a uniform source distribution. Though less representative of physical phenomena, this allows for a more straightforward development process and sets up a test framework for future iterations of the project.

Chapter 1

Introduction

1.1 The APT

The Advanced Particle-astrophysics telescope is designed to be a successor to the Fermi Gamma-ray Space Telescope, which has surpassed its expected mission time of 10 years. The main science objectives of the APT include studying high-energy transient phenomena such as supernovae and gamma-ray bursts (GRBs), continuing the search for dark matter, and conducting broader surveys of the sky in gamma rays. The APT would improve upon Fermi's sensitivity while maintaining a similar budget, allowing for a more extensive search for dark matter and an increased ability to detect transients.

1.1.1 Hardware design

The APT is designed to be used in the mid-keV to low TeV energy range, which is a significant improvement on Fermi's detecting range. The final telescope will consist of 20 repeated layers of detecting material, spaced 15 cm apart, in a cube 3 meters tall and 2.5 meters on each side. The middle section of each layer - the calorimeter - records the energy of each photon or particle that interacts with it, and the WLS fibers on the top and bottom of each layer record the corresponding position of each interaction. From this series of detected interactions, we are able to reconstruct the initial direction of the photon source using software.

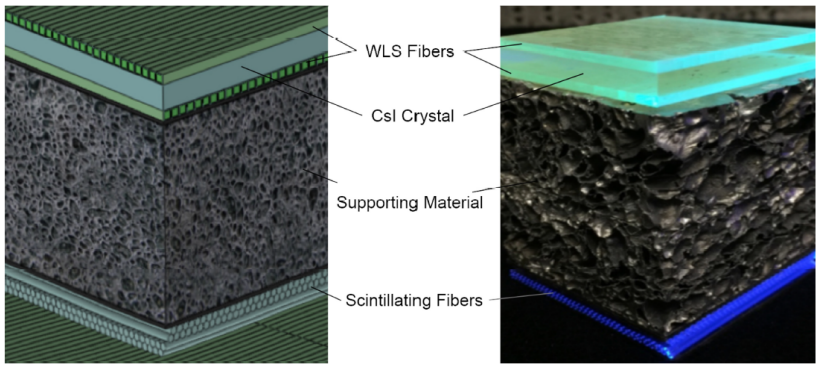


Figure 1.1: One layer of the APT, shown as a computer model on the left and as a real-world prototype on the right, with aluminum used for the support structure. The Cesium Iodide (CsI) detects photon energy while the WLS fibers detect photon position. The scintillating fibers detect the position of any matter particles that enter the detector. Each subsequent layer would have the same structure as the one shown. Reprinted from [3].

1.1.2 Software design

The APT achieves such a broad energy range by using two different software methods for reconstructing the trajectories of incoming photons, one for each of the two dominant gamma-ray interactions in this energy range. Below approximately 30 MeV, the dominant interaction is called Compton scattering, a process by which a photon transfers some portion of its momentum to an electron in a detector layer, changing its wavelength and trajectory. In the mid-MeV to GeV range, photons most commonly undergo a process called pair production upon interacting with the detector, in which a photon splits into a positron and an electron, which then interact further with the detector. As the goal of this project is to reconstruct Compton scatters, a thorough discussion of pair production is outside the scope of this paper, but a description of this phenomenon can be found in almost any particle physics textbook.

1.2 Gamma-ray bursts

Though the APT can theoretically detect many types of transients, we focus primarily on gamma-ray bursts for this project. A gamma-ray burst (GRB) is an extremely bright burst

of gamma-rays from a point source in the sky. These were first discovered in the 1960s by a US satellite that had been set up to detect radiation from nuclear blasts during the Cold War. The Compton Gamma-Ray Observatory (CGRO), launched in 1991, was used to determine that GRBs originate mainly from outside the Milky Way, and the Fermi Gamma-Ray Space Telescope, launched in 2008, has increased our understanding of the processes that cause these highly energetic blasts. A GRB can last anywhere from a few milliseconds to several hours, which is an incredibly short time window compared to most other astronomical observations. Many astrophysicists now believe that short-duration GRBs are caused by neutron-star mergers (collisions of extremely dense stellar remnants), while longer-duration GRBs originate from core-collapse supernovae (explosions of high mass stars), however much remains a mystery about how and why these events occur.

1.3 This project

1.3.1 Motivation

Though several theories exist as to their causes, the emission mechanisms of GRBs are not yet well-understood. The energies involved indicate a very efficient conversion of matter to energy, the process that drives this conversion is still an open question in the field of astrophysics. To better understand the processes that produce GRBs, it would be highly useful to observe them simultaneously with multiple different telescopes at multiple different wavelengths (gamma-ray, optical, infrared, etc.) To do this, we must be able to search for and detect a GRB in its initial stages and send out its location to other observatories. Achieving this goal could lead to future discovery in the field of astrophysics as GRBs become better understood.

One major science objective of the APT is to process gamma-ray events quickly enough to localize a source within a few seconds of the start of the burst and signal its location in that time. This means that we must significantly increase the detection capabilities of the APT in the low-energy range (keV - MeV) such that the incoming photons' directions can be reconstructed as quickly as they are received by the detector. As Compton scattering is the dominant interaction at these low energies, the primary focus of this research is to

develop and test an algorithm to reconstruct Compton-scattered photons quickly and accurately enough that the telescope can pinpoint their source in close to real time. Many Compton telescopes rely on photon events with only two interactions to reconstruct their source positions. However, a significant fraction of events in this low energy range will scatter twice or more in the detector, which means that we can achieve significant performance improvement by incorporating the reconstruction of multi-scatter events into our algorithm.

1.3.2 Scientific constraints

To eventually reach the main goal of this project, several factors be taken into account at this stage. The telescope must be able to reconstruct the trajectories of gamma rays at or near the speed at which they enter the detector, with low latency between the initial detection signal and the reconstructed solution. We expect to see between 10^5 and 10^9 photons per second during a typical gamma-ray burst[8]. Our goal for reconstruction speed is $\backsim 10^5$ photons per second, with $> 75\%$ accuracy and a latency of 1 second or less, as we expect this will give us an accurate enough localisation of the source position, but more complex simulations and source reconstructions will allow us to further constrain these target values. One of our other concerns is power consumption - the telescope will be part of a larger scientific instrument with a fixed amount of energy available to it daily. As such, we estimate that it cannot use more than 50 watts of power when running the reconstruction software.

1.3.3 Process

We base our initial Compton reconstruction algorithm on *Boggs & Jean (2000)*[2] and, with several performance improvements, we are able to meet our performance goals for this project. We start with a sequential algorithm, which enumerates each possible ordering of detector hits, and improve its performance by incorporating a tree search and pruning methods to improve the runtime. We build a simple gamma ray simulator for our development and initial tests of reconstruction speed and accuracy. Initial tests show that, with average parameters, the algorithm is able to reconstruct $\backsim 10^5$ photons per second with 80-90% accuracy, which we believe will be enough to detect and localize a gamma-ray burst once the telescope is operational. By varying parameters in the algorithm, we also examine trends

in the speed and accuracy, and discuss some possible trade-offs between the two, as well as possible improvements to the speed and accuracy for future work.

1.4 Related Work

One of the best papers available on Compton reconstruction procedures is *Boggs & Jean* (2000)[2] which details the mathematical formulae and steps required to reconstruct the initial direction of a Compton event in a layered detector. Though the paper focuses more on the physics of reconstructing Compton scatters than the algorithmic parameters, the equations listed served as a very helpful starting point for building and refining our code. We developed with the purpose of creating an algorithm to be as fast and lightweight as possible while still meeting our accuracy requirements. Many Compton telescopes must transfer data over a network connection or save observations in order to reconstruct them later, but our program is designed to process data in real time, before it leaves the telescope.

Chapter 2

Compton Reconstruction

2.1 The physics of Compton scattering

Compton scattering is a process in which a photon hits a charged particle and transfers some momentum to it in the collision. Due to conservation of momentum and energy, the photon moves away from the collision with a different wavelength and direction than it had initially, shown in Figure 2.1. In the APT, gamma rays from space scatter off of bound electrons in the detector material and kick them out of the layer that they had previously occupied. The gamma ray's energy can then be described with the following equation:

$$E' = \frac{E_0}{1 + \frac{E_0}{m_e c^2}(1 - \cos\theta)} \tag{2.1}$$

Where E' is the energy after the collision, E_0 is the energy before the collision, θ is the scattering angle of the photon, and m_e is the mass of the electron. Note that the scattering angle, the initial energy, and the final energy are the only free parameters in this equation, so determining any two of these values also determines the third. In reconstruction, we use the total energy deposited in the detector as the initial energy and are able to infer the energy and angle of each scatter from there.

Equation 2.1 is based on the assumption that the electron is at rest in the detector material prior to the collision. The effects of the electron's initial momentum lead to a phenomenon called Doppler Broadening in the resulting Compton spectrum, which causes some error in our calculations. However, this effect is relatively small ($\backsim$0.01% of initial wavelength) and

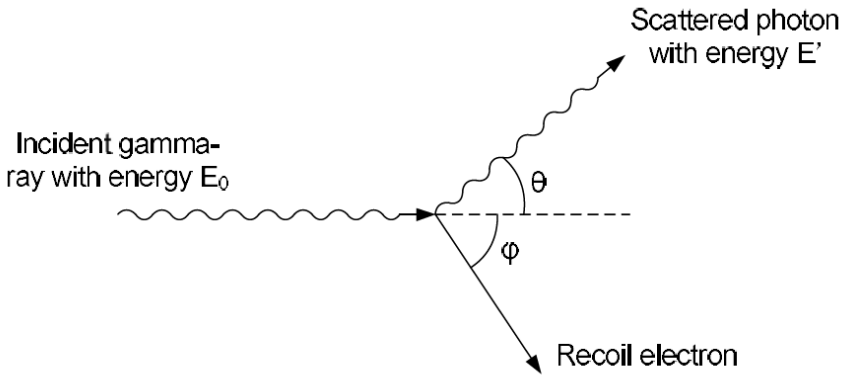


Figure 2.1: Diagram of the Compton scattering process. Reprinted from [11].

randomly distributed, so we do not take it into account in our initial calculations, though it may be useful to do so in future work.

2.2 Reconstructing photon direction

Photons are measured in the APT as a series of energy deposits, or *hits*, in the detector material, as shown in Figure 2.2. The final hit is assumed to be a photo-absorption, in which the photon is absorbed into the detector material instead of scattered off an electron. Each time the photon scatters, it deposits some energy in the detector, which is recorded through the scattered electron. It will then either scatter again or undergo photo-absorption, with the relative probability of each process depending on the remaining energy. From Equation 2.1, if we know the scattering energy and positions of the first two hits, we can use these to calculate the initial scattering angle of the photon. However, this does not give us the exact trajectory of the incident photon, but rather the "cone of possibility", defined by the scattering angle and the direction we record for the scattered photon. When taken for many photon events, these values can be used to extrapolate the position of the initial gamma-ray source.

If we already knew which hits came first in each sequence, or *event*, we could get the initial scattering angle of the photons with a simple calculation. However, the biggest challenge in

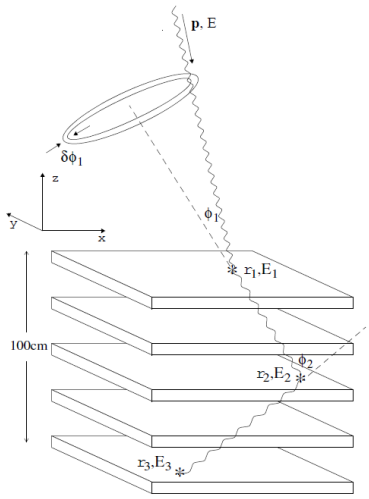


Figure 2.2: An example event showing a gamma-ray's trajectory through detector layers. Each * symbol is one hit. The dotted line shows the center of the cone of possibility while the wavy line represents the true path of the gamma-ray. Reprinted from [11].

reconstructing photon trajectory is the fact that the detector cannot give us the chronological ordering of hits. Each gamma-ray moves at the speed of light, and the distance between detector layers is small enough that we would need a much higher time resolution to determine the first two hits chronologically. However, Equation 2.1 allows us to infer the correct ordering in a different way, by comparing the spatial angles between the recorded hits to the scattering angles implied by their respective energy deposits. If there were no other factors involved, each correct spatial angle would have an exact match in energy angle, but detector noise and electronic effects mean that we must take a probabilistic approach to finding the correct sequence.

2.3 Methodology

To make things simpler in our code, we break each photon's series of hits into triples, one for each possible sequence of three hits. Each triple has a spatial angle - such as ϕ_2 in Figure 2.2 - and an energy angle, which is calculated by Equation 2.1 using the summed energy deposits of the sequence. One obstacle to our calculations is that the implied scattering angle does not depend on the energy deposited at the vertex of a given triple, but on the

total energy of the photon both before and after the hit at the vertex. This means that we cannot calculate the energy angles individually, but only as part of a sequence. If we change the hits in the sequence before a given triple, it changes the energy calculations for the triple itself. To describe these constraints mathematically, we use equations adapted from Boggs & Jean (2000)[2]:

$$W_i = \frac{1}{m_e c^2} \sum_{j=i+1}^{N} E_j \qquad \text{Unitless energy} \tag{2.2}$$

$$\eta_i' = \cos\theta' = 1 + \frac{1}{W_i} - \frac{1}{W_{i+1}} \qquad \text{Energy angle} \tag{2.3}$$

$$\eta_i = \cos\theta = \hat{r}_i \cdot \hat{r}_{i+1} \qquad \text{Spatial angle} \tag{2.4}$$

$$\hat{r}_i = \frac{\vec{x}_i - \vec{x}_{i-1}}{|\vec{x}_i - \vec{x}_{i-1}|} \qquad \text{Direction before hit } i \tag{2.5}$$

Where N is the event's total number of hits in the detector and W_i is the unitless energy of the photon after hit i (i.e. W_0 is the initial energy of the photon and $W_{i>0} < W_0$). Equation 2.3 is just a reformulation of the original Compton equation (2.1) with θ' representing the energy scattering angle at hit i. θ is the spatial angle of hit i, and is calculated using a dot product, where $\vec{x}_i$ refers to the position of hit i and $\vec{r}_i$ is the vector between hit $i-1$ and hit i. Therefore $\hat{r}_i$ is the direction of the photon immediately before hit i, and $\hat{r}_{i+1}$ is its direction immediately after. As finding the inverse cosines of Equations 2.3 and 2.4 can cost us valuable computation time, we leave them in their cosine form, η and η', and compare these values rather than the values of the angles themselves. The cosine function is uniquely determined for $\theta, \theta' \in [02\pi)$, so the two operations are mathematically equivalent.

The χ^2 metric

To compare the spatial and energy angles of a given sequence of hits, we use a χ^2 test:

$$\chi^2 = \frac{1}{N-2} \sum_{i=2}^{N-1} \frac{(\eta_i - \eta_i')^2}{\delta\eta_i^2 + \delta\eta_i'^2} \tag{2.6}$$

Where η_i and η_i' again represent the spatial and energy angles and $\delta\eta_i$ and $\delta\eta_i'$ represent the error in the spatial and energy angles. An example graph of this distribution can be found in Figure 3.2.

The χ^2 distribution is a sum of squared Gaussian distributions, so if we assume our noise/error, $(\eta_i - \eta_i')$, has an approximately Gaussian distribution, we can use the χ^2 metric with $N-1$ degrees of freedom (the total number of hits) to determine the likelihood of a given ordering of hits. We can see by the equation that if the spatial and energy angles of any given triple match more closely, the value of χ^2 will be lower, whereas if they are farther apart it will be higher. Therefore, in the absence of other factors, the sequence of hits that matches our data best will be the one with the lowest χ^2 value. As the factors that affect our noise and error levels result from complicated physical processes, the true noise distribution may not be perfectly Gaussian. However, based on the Central Limit Theorem[7] and a reasonable assumption of the number of independent variables that can affect our angle calculations, we can say that a Gaussian distribution is likely a reasonable approximation for our overall error.

Chapter 3

The Reconstruction Algorithm

3.1 Basis

Our reconstruction uses the χ^2 metric to determine which ordering of hits is best. Each triple represents one term in the sum of Equation 2.6. Since we assume that the ordering with the lowest χ^2 value is the correct one, we must try each possible ordering of hits in order to find it, which is very computationally complex. Our initial pseudocode for this calculation is shown in Algorithm 3.1. If there are N hits in a photon event, there are $N!$ possible orderings of those hits, and we must recalculate the η value for each hit in each ordering, adding an additional factor of N, to give us a total run-time of $O(N * N!)$ for each photon. There are several heuristic improvements we can make to shorten the average-case run-time, but we start with a basic iterative and sequential approach for simplicity's sake.

3.1.1 Data types

Data for each hit of each event is saved in a `Hit` data type, containing an x, y, z position and the energy deposit in MeV. Each photon which interacts with the detector is represented as an `Event` data type, which contains an array of `Hit` values and the number of hits total for that photon. Each reconstructed solution is contained in a `Result` data type, which contains the first two hits of the event, the scattering angle, and the error in the scattering angle (calculated from the detector noise). We save an array of Events and true `Result`

values in our simulations, then pass them to our reconstruction algorithm to get our overall accuracy.

3.2 Performance improvement

Though it is difficult to improve the theoretical time complexity of our algorithm, it is possible to make several changes that decrease the average computation time per photon. To do so, we switch from an iterative approach to a tree search and use pruning and parallelism to further decrease the run-time.

3.2.1 Tree search

The first and perhaps most important performance improvement we make is to change the structure of our program from an iterative approach - testing each sequence individually, one after another - to a recursive tree search, shown in Algorithms 3.2 and 3.3. Each photon has its own search tree containing nodes which correspond to its `Hit` values. An example search tree is shown in Figure 3.1. Each parent node has a series of child nodes corresponding to the next hit in the sequence, and to search the tree we simply have to choose a path through it, keeping track of the χ^2 value as we go. Once we have searched each path, we assume the path with the lowest χ^2 value is the correct one and use the first two hits to predict the value of η'_0 - the initial scattering angle. This is an improvement on our sequential algorithm, as we do not have to re-compute χ^2 for each node with the same parent sequence, instead simply adding onto it each time we process a new node.

Recursive algorithm

In our program, we keep track of the χ^2 value for each `Hit` we add to the sequence. As we cannot start calculating the χ^2 value without at least one triple, we first take each possible pair of hits and run them through our recursive search algorithm (line 12, Algorithm 3.2), keeping a running tally of the χ^2 value and updating it for each child node we process (line 15,

Algorithm 3.1 Sequential reconstruction algorithm for one photon

Input: `x` - a 1D array of `Hit` values for one event, `n` - the number of hits in the event

$\min\chi^2 \leftarrow \infty$ ▷ Running minimum χ^2

r $\leftarrow$ `NULL` ▷ `Result` variable

for j in 1...n! **do** ▷ N! permutations total

  $x_j \leftarrow$ permute(x, j) ▷ Returns unique permutation of hits

  $\chi^2 \leftarrow 0$

  for $k \in 1...n-1$ **do** ▷ Loop over hit sequence

    // Calculate spatial angle:

    $\vec{x}_{k-1}, \vec{x}_k, \vec{x}_{k+1} \leftarrow x_j$[k-1], x_j[k], x_j[k+1] ▷ Consecutive hit positions

    $\hat{r}_k, \hat{r}_{k+1} \leftarrow |\vec{x}_k - \vec{x}_{k-1}|, |\vec{x}_{k+1} - \vec{x}_k|$ ▷ Unit vectors along photon direction

    $\eta_k = \cos\phi_k = \hat{r}_k \cdot \hat{r}_{k-1}$ ▷ Spatial angle from inner product

    // Calculate energy angle:

    $W_k = \frac{1}{m_e c^2} \sum_{i=k+1}^{n} E_i$ ▷ Unitless energy; energy before hit k

    $\eta'_k = \cos\phi'_k = 1 + \frac{1}{W_{k-1}} - \frac{1}{W_k}$ ▷ Energy angle from Compton equation

    // Calculate Errors:

    $\delta\eta_k^2 = \delta\phi_{k,r}^2 \sin^2(\phi_k)$ ▷ $\delta\phi_{k,r}$ depends on $\delta x, \delta y, \delta z$

    $\delta\eta_k'^2 = \frac{\delta W_{k-1}^2}{W_{k-1}^4} + \delta W_k^2 \left[(\frac{1}{W_k^2} - \frac{1}{W_{k-1}^2})^2 - \frac{1}{W_{k-1}^4} \right]$ ▷ δW_k from energy and noise level

    // Add to χ^2:

    $\chi^2 \leftarrow \chi^2 + \frac{1}{n-2} \frac{(\eta_k - \eta'_k)^2}{\delta\eta_k^2 + \delta\eta_k'^2}$

  end for

  if $\chi^2 < \min\chi^2$ **then** ▷ Check against minimum

    $\min\chi^2 \leftarrow \chi^2$

    $\eta'_0 = 1 + \frac{1}{W_0} - \frac{1}{W_1} \pm \delta\eta'_0$ ▷ New η value

    r $\leftarrow x_j[0], x_j[1], \eta'_0$ ▷ Update `Result` value

  end if

end for

Output: r - `Result` with predicted η and first two hit indices

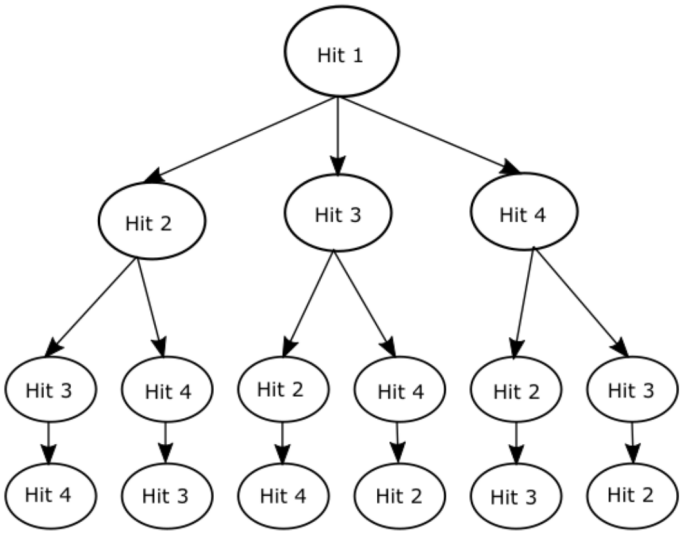


Figure 3.1: Visual representation of a tree search for one initial hit.

Algorithm 3.3). Theoretically, we could still have to process every possible sequence before reaching the minimum χ^2 value, but, as will be discussed in the following sections, there are ways to prune the tree in order to decrease the average runtime. The real performance improvement of the tree, however, comes from the recursive approach itself. In our iterative approach, we recalculated χ^2 for each node, including those which had the same parent sequence (giving them the same χ^2 value). In the recursive approach, we cut down on these repeat calculations by carrying the χ^2 value through each sequence and only adding onto it when we process a new node.

3.2.2 Pre-calculation of η values

Another performance improvement comes from pre-calculating the spatial angles for each triple, shown in Algorithm 3.2 line 2. In the sequential/iterative algorithm (3.1), we calculate η for each triple in each sequence, but, unlike η', the spatial angle does not change our calculations based on the sequence it is in. As it is only based on the hits in its given triple, we can calculate η only once for each possible triple before we start the tree search and fetch the values when they are needed. This reduces the computation time at each step in photon reconstruction, which greatly increases our reconstruction speed.

Algorithm 3.2 Tree search reconstruction algorithm for one photon

Input: `x` - a 1D array of `Hit` values for one event, `n` - the number of hits in the event

// Pre-compute spatial angles, η:
for i,j,k in permutations(n, 3) **do** ▷ Loop over all triples
    // Calculate spatial angle:
    $\vec{x}_i, \vec{x}_j, \vec{x}_k \leftarrow x$[i], x[j], x[k] ▷ Consecutive hit positions
    $\hat{r}_i, \hat{r}_j \leftarrow |\vec{x}_j - \vec{x}_i|$, $|\vec{x}_k - \vec{x}_j|$ ▷ Unit vectors along photon direction
    triples[i][j][k] $= \hat{r}_i \cdot \hat{r}_j$ ▷ Save spatial angle
end for

$\min\chi^2 \leftarrow \infty$
$r \leftarrow$ `NULL` ▷ `Result` value

for i,j in permutations(n, 2) **do** ▷ Loop over all possible pairs of `Hit` indices
    $\chi^2 \leftarrow$ `findOptRecursive`(i, j, n, 0) ▷ Pass first two hits to tree search

    if $\chi^2 < \min\chi^2$ **then** ▷ Check for most likely permutation
        $\min\chi^2 \leftarrow \chi^2$
        $\eta'_0 = 1 + \frac{1}{W_0} - \frac{1}{W_1} \pm \delta\eta'_0$ ▷ New η value
        r $\leftarrow x[i]$, $x[j]$, η'_0 ▷ Update `Result` with first two hits and η'_0
    end if
end for

Output: r - `Result` with predicted η and first two hit indices

Algorithm 3.3 Recursive function in tree search

Input: `i`, `j` - current and next hit indices, respectively, `n` - number of hits left, χ^2 - current χ^2 value for the sequence

```
1: function findOptRecursive(i, j, n, χ²)
2:     // Base case:
3:     if n == 0 then
4:         return χ²
5:     end if
6:
7:     for each unused Hit, index k do
8:         // Calculate the new energy angle:
9:         W ← W − (1/(m_e c²)) E_2                    ▷ Calculate change in energy
10:        δW² ← δW² − (1/(m_e c²)) δE_2²
11:
12:        // Calculate η′ and δη′                      ▷ See Equation 2.3
13:        η ← triples[i][j][k]                        ▷ δη is constant spatial noise
14:
15:        Calculate new χ² value from η, η′, δη, δη′  ▷ See Equation 2.6
16:
17:        Prune the sub-tree, if possible             ▷ See Section 3.2.3
18:
19:        findOptRecursive(j, k, n − 1, new χ²)       ▷ Move on to next two hits
20:    end for
21: end function
```

3.2.3 Computational cutoffs

During our tree search, we can *prune* some sub-trees by filtering out certain sequences before we reach the final recursion depth. Depending on the input ordering of hits, we could still end up having to search the whole tree for the correct path, but generally these methods will improve our runtime on average. We can prune a sub-tree if:

1. **The χ^2 value is already greater than the running minimum.**

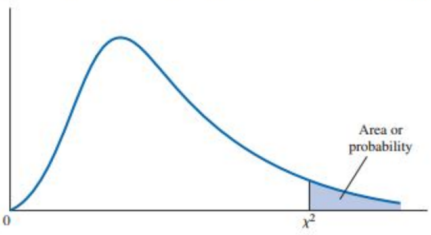
   Since we always assume the sequence with the minimum χ^2 is best, there is no need to check orderings which have already exceeded our best value.

2. **The cosine of the energetically reconstructed angle, η', is greater than 1 by some amount, the η' cutoff.**

   Our algorithm might return an η'-value greater than 1 if we either have the wrong ordering of hits or if there is significant noise in the measurement, so we only prune the tree for values significantly above 1, to be safe.

3. **The χ^2 value of a sequence exceeds what we would expect for the length of the sequence.**

   The p-value is related to the probability that the χ^2 we calculate (based on a Gaussian assumption of noise) will naturally exceed a given χ^2 value with the same degrees of freedom. In other words, it is the probability that a good ordering would have a χ^2 value higher than the expected one. We use a look-up table such as the one shown in Figure 3.2 to find the maximum allowed χ^2 value at each step in our calculation and cut off any sequence which exceeds it. The degrees of freedom we use in the χ^2 table is the number of hits in the sequence so far. For example, if we choose a p-value of 0.01, it means we will not accept any orderings that have less than a 1% chance of being correct. Using this with Figure 3.2, this means we will prune the tree for any χ^2 value greater than 6.635 after one hit, greater than 9.210 after two hits, greater than 11.345 after three hits, and so on.



Degrees of Freedom	Area in Upper Tail .10	.05	.025	.01	.005
1	2.706	3.841	5.024	6.635	7.879
2	4.605	5.991	7.378	9.210	10.597
3	6.251	7.815	9.348	11.345	12.838
4	7.779	9.488	11.143	13.277	14.860
5	9.236	11.070	12.832	15.086	16.750
6	10.645	12.592	14.449	16.812	18.548
7	12.017	14.067	16.013	18.475	20.278
8	13.362	15.507	17.535	20.090	21.955
9	14.684	16.919	19.023	21.666	23.589
10	15.987	18.307	20.483	23.209	25.188
11	17.275	19.675	21.920	24.725	26.757
12	18.549	21.026	23.337	26.217	28.300
13	19.812	22.362	24.736	27.688	29.819
14	21.064	23.685	26.119	29.141	31.319
15	22.307	24.996	27.488	30.578	32.801

Figure 3.2: An example χ^2 look-up table. [5]

4. **We have reached some maximum defined recursion depth, which we refer to as the *reconstructed hits***

   Though this is not shown directly in the pseudocode, it is implemented by changing the base case in Line 3 of our program to stop the program after a certain number of hits have been processed. We then assume that the sequence with the minimum χ^2 at the cutoff point has the minimum overall χ^2, though there is some chance that this is not the case (the exact probability depends on the cutoff point). This serves to decrease our computation time, but it also means that we do not take all of our data into account, which we expect will decrease our accuracy.

3.3 Parallelism

Our final performance improvement comes from parallelism. Each photon's computation is independent of the others', so each can be run in a parallel thread to improve the runtime of our algorithm. This also means that if any given photon's computation took longer

than expected, there would not be a backlog of events to process which could delay the localization of a gamma-ray burst or other transient. As our current level of parallelism was able to adequately meet our time constraints, we do not seek to increase the reconstruction speed further. However, depending on future time constraints, there are alternate strategies of parallelism and source localization that we could employ to do so.

Chapter 4

Experiments

4.1 Toy model simulator

Our toy model is a very basic simulator that traces the paths of virtual gamma-ray photons through our detector. This allows us to test code in a controlled environment, which is very useful in the development of our reconstruction algorithm and enables us to ensure our code is performing as we expect without the additional errors caused by outside factors. However, due to this simplicity, there will be physical effects present in the real world that we fail to account for in our simulator, such as background radiation, photon scatters in the detector support material, and various electronic effects, that will increase our overall error. We do not take the detector material into account, nor do we allow for any types of interactions other than a series of Compton scatters followed by a photoabsorption. We set up a test framework on this model simulator that will allow us to adapt our testing to more complicated gamma-ray simulators in the future.

4.1.1 Photon simulation

The simulator function, with pseudo-code shown in Algorithm 3.2, takes as its parameters the number of photons to generate and the number of hits each photon should have (including the final absorption). We pick the initial energy of each photon from a uniform distribution between 100 keV and 20 MeV (Line 12).

Algorithm 4.1 Toy Model Simulator

Constants: `minE`, `maxE` = 100 keV, 20 MeV - limits on initial photon energy **Inputs:** `numEvents` - the number of photons to simulate, `numHits` - hits per event, `events` and `results` - empty arrays for storage

```
1: function SIMULATE(numEvents, numHits, events[], results[])
2:     init alphas[], W[], k[], x[]          ▷ Arrays for the angles, energies, directions,
3:                                                  ▷ and hit positions, respectively
4:     for i in numEvents do
5:         // Initialize variables:
6:         Result & r ← results[i]                  ▷ Set up result and event references
7:         Event & e ← events[i]
8:         e.hits ← empty Hit array                          ▷ Initialize Event hits
9:         e.numHits ← numHits
10:
11:        // Generate initial energy at random:
12:        r.energy ← uniform_random(minE, maxE)                  ▷ E ∈ [minE maxE]
13:
14:        // Generate n-1 random angles:
15:        alphas ← uniform_random(−π/2, π/2, n-1)          ▷ Angle array of length n-1
16:        r.eta ← cos(alphas[0])                                  ▷ Save initial angle
17:
18:        // Calculate Hit positions, directions, and energies from angles (see §4.1.1)
19:        calcW(numHits, r.energy, alphas[], W[])      ▷ Stores energy deposits in W[]
20:        calcK(numHits, alphas[], k[])                ▷ Stores photon directions in k[]
21:        calcX(numHits, k[], x[])                        ▷ Stores Hit positions in x[]
22:
23:        perm ← random permutation of x[] indices                ▷ Randomly shuffle hits
24:        r.p0, r.p1 ← perm[0], perm[1]                          ▷ First two hit indices
25:
26:        // Record each Hit's position and energy deposit
27:        for j in numHits do
28:            E_dep ← (W[j] - W[j+1])*mc^2                  ▷ Calculate energy deposit
29:            e.hits[perm[j]] ← (x[j], E_dep)                                ▷ Save Hit
30:        end for
31:
32:        addNoise(e)          ▷ Add measurement noise to each hit in the Event
33:
34:    end for
35: end function
```

Once we have initialized our photon, we simulate its interaction with the detector by generating a random set of scattering angles according to the input number of hits (Line 15). The scattering angles are taken from a uniform distribution from -90 to +90 degrees. We do not simulate back-scatters ($\geq$ 90 degrees), since these are relatively unlikely in the real world. We then use formula 2.1 to calculate the energy deposited for each generated scattering angle and store these values in an array (Line 19, `calcW`).

After calculating the angles and energy deposits, we then must calculate the position of each hit in the detector. To do this, we need to first determine which layers of the detector the photons will scatter in and the vector directions between each pair of hits. This takes place in our function `calcK` on line 20 of our algorithm: for each scatter, we determine the layer of each hit based on an exponential probability distribution, calculate the scattering direction vector by adding the scattering angle (with a random ϕ angle) to the photon's previous direction, and save the direction vectors in array `k`. We can then use simple vector addition to determine the `Hit` positions and save them in array `x` (Line 21, `calcX`). We randomly shuffle the hits (Line 23) in order to simulate the unknown ordering of hits in the detector. If we saved the hits in sequential order for each photon it might cause our program runtime to be shorter than is realistic. Our last step, on Line 32 of our pseudocode, is to simulate some random detector noise and add it to each position and energy measurement. The distributions and parameters we use to generate this noise are discussed in the next section.

4.1.2 Errors and noise

We simulate noise in our detector using a Gaussian distribution with variable standard deviation, σ_S. For the position measurements, the user inputs the standard deviation in millimeters, and for the energy measurements, the standard deviation is input as a constant factor of the energy, a:

$$\frac{\sigma_E}{E} = a * \frac{1}{E\ (keV)} \tag{4.1}$$

We expect based on previous tests of the detector material that σ_S will be about 1 mm and a will be about 0.22.

Though the uniform distribution we generate our scattering angles from does not reflect the correct Compton cross-section, our use of Formula 2.1 ensures that no interaction we generate would be impossible in nature. As our reconstruction algorithm only compares the inferred angles to the true angles and does not take the scattering cross-section into account either, this simulator still provides a fairly good test of our performance in the absence of systematic measurement error. The true distribution of scattering angles is determined by the Klein-Nishina formula and the interaction probability by the Thomson cross-section [1], so in later iterations of this project it may be useful to incorporate these into our toy model, particularly if we choose to integrate some component of the true scattering probabilities in our reconstruction algorithm.

4.2 Hardware

The hardware we use to test our algorithm is the Raspberry Pi 3, Model B+. It uses a 4-core ARM processor with a 1.4 GHz processing speed, and has 1 GB of RAM. Due to its low number of cores and static execution, it is a slower processor than found in most laptops, but it is perfect for testing our software as it is low-power enough to be used as part of a space telescope. We use the WITRN U2 USB Power Monitor[10] for testing power usage on the Raspberry Pi. The ARM processor is commonly used for scientific applications, and because NASA has recently commissioned a space processor based on the same architecture, this platform could be the closest hardware available to what our program will actually be running on in space. We tested our parallelism on Cassini, a more advanced machine at Washington University, to get a more accurate measure of our core speedup in Section 5.3.

4.3 Performance measures

We use two main performance measures to evaluate our algorithm - the number of photons processed per second, and the accuracy. We consider a reconstruction accurate if the first two hits of the reconstruction match the first two hits in the correct sequence, and refer to this measure as the "two-hit accuracy", or simply the accuracy. We report this statistic as

the number of accurate reconstructions over the number of total reconstructions. We also consider power consumption to be a measure of performance, but any trends in power are not studied in-depth as we are well below our expected limit.

4.4 Parameters tested

There are many variables that have the potential to affect the algorithm's performance, and we want to specifically examine the trade-offs in reconstruction speed vs. accuracy for several key parameters. The tests we run are as follows:

1. Power - We use the WITRN U2 USB Power Monitor[10] to test the power used by the Raspberry Pi both when it is idle and when it is running our program.
2. Single vs. double precision - As our program is written in C++, we have a choice of whether to use `float` values (single precision) or `double` (double-precision) values in our calculations. Single precision values use fewer bytes than double values, making them faster to use in calculations but sometimes less accurate due to rounding error.
3. Simulated hits - The number of hits we simulate for each event.
4. Reconstructed hits - The number of hits we use to reconstruct a photon's trajectory. We cut off computation once the specified number of hits is reached, after which the sequence with the lowest χ^2 value so far is presumed to be the correct one. (i.e. we set a lower-depth base case in Line 3 of Algorithm 3.3).
5. χ^2 cutoff - The p-value we use for our cutoff χ^2 values. Ex: a p-value of .1 would give us a 10% probability of cutting off each good triple, whereas a p-value of .01 would give us a 1% probability of doing so.
6. η'-tolerance - The amount above 1 or below -1 that we will allow for a calculated cos(energy angle). For example, if the η'-tolerance is 0.2, any sequences containing an $\eta' > 1.2$ or < -1.2 would be cut.
7. Predicted spatial and energy noise levels - Our χ^2 calculations rely on the predicted errors of our η and η values, so changing the predicted level of error will change the

behavior of our algorithm. We vary the σ_E and a values found in Section 4.1.2 and investigate the results.

8. Simulated spatial and energy noise - We investigate the interplay between the predicted noise and the simulated noise in both the spatial and energy regimes. Though we have no control over the noise levels we will encounter in operation, it is useful to simulate a range of noise levels and adjust our predicted noise parameters accordingly. This is also useful as a proof-of-concept, as we can check to make sure our algorithm is behaving as expected under increased noise thresholds.

Chapter 5

Results

We test each of the variables specified in Section 4.4 over a range of values to examine trends in the reconstruction speed and overall accuracy. Each simulation uses the values listed in Table 5.1 unless otherwise noted, and each individual measurement comes from an average of five test runs. Refer to Section 4.4 for more detailed parameter descriptions. Our error bars represent one standard deviation away from the mean and are calculated under the assumption of Gaussian noise, which we again assume to be a reasonable approximation of the true noise based on the Central Limit Theorem[7].

Variable:	Precision	Simulated hits	Reconstructed hits	p-value	η'-tolerance	σ_{sp}	a
Value:	single	5	5	0.01	1.0	1 mm	0.22

Table 5.1: Default values used in Compton simulations.

5.1 Power

To measure the power our hardware uses when running, we use the WITRN U2 USP Power Monitor[10]. Based on past experience, our original estimate for how much power usage would be acceptable aboard a telescope array was $\backsim$50 W. In testing the Raspberry Pi's power usage, we found values ranging from about 2.52 W when idle to 2.74 W on average after running the reconstruction algorithm for a few hours. The differential power usage, found by subtracting the idle power from the power while in use, was 0.23 W on average.

Though our USB monitor did not allow us to calculate the standard deviation of our power usage, our maximum total and differential powers were 4.82 W and 2.30 W, respectively, which are both an order of magnitude less than our estimated power budget. We expect that the hardware used in final the telescope design will be similar to the Raspberry Pi, so it is likely that the power drawn by our program in practice will be similar to these values.

5.2 Precision

The precision of floating point values refers to the size in computer memory of the numbers we use in our calculations. We test both single-precision (32-bit) `float` values and double-precision (64-bit) `double` values, and display the results in Table 5.2. Our program is slightly less accurate when using float values rather than double values, but there is a large overlap in the two margins of error, so this is not very statistically significant. We also see a large decrease in reconstruction speed when using `double` values compared to `float`s. This is an expected result, as each basic operation takes slightly longer for 64-bit values than for 32-bit values regardless of the hardware. This effect would be less pronounced on Cassini as it has an out-of-order processor which can use its idle clock cycles to perform instructions for future use, decreasing the overall runtime, however the Raspberry Pi is an in-order processor and does not have such capabilities. Therefore, it is best that we use single-precision values in our final version of the program to increase the reconstruction speed.

Precision	Single	Double
Accuracy (%)	$86.12 \pm .12$	$86.15 \pm .10$
Throughput (γ/sec)	$5.08(70) \times 10^5$	$3.85(51) \times 10^5$

Table 5.2: Speed and accuracy results for single and double precision values.

5.3 Parallelism

We test the parallelism of our program on Cassini, as the Raspberry Pi does not have enough cores to see trends. The speedup of our algorithm, which is the execution time of our program when run on x cores divided by the execution time when run on one core, is

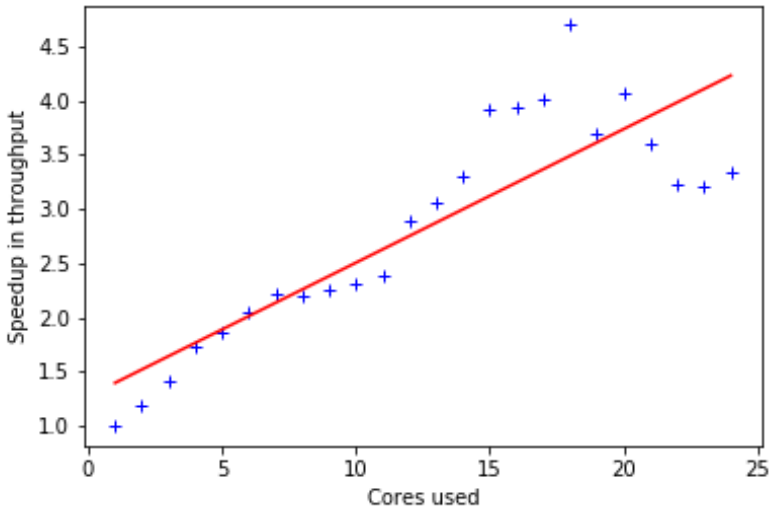


Figure 5.1: Core speedup of reconstruction algorithm, fit with a linear trendline (red).

shown in Figure 5.1. We can see that we have close to linear speedup when the number of cores is small and a linear trend on average over the whole range, despite seeing some nonlinear effects at higher numbers of cores. This indicates that our program scales well with increased execution capacity.

5.4 Total hits used

To ensure that our program is behaving correctly, we investigate its behavior using events with increasing numbers of hits. We can see from the left plot of Figure 5.2 that our throughput decreases exponentially with increasing numbers of simulated hits. While not ideal, this is an improvement on our original estimate, which predicted a factorial increase in computation time for increasing hits. Though, theoretically, the worst-case time complexity of our program remains the same as in the iterative version, tree search improves the average-case complexity of our search algorithm and displays significantly improved performance when used with large numbers of simulated photons.

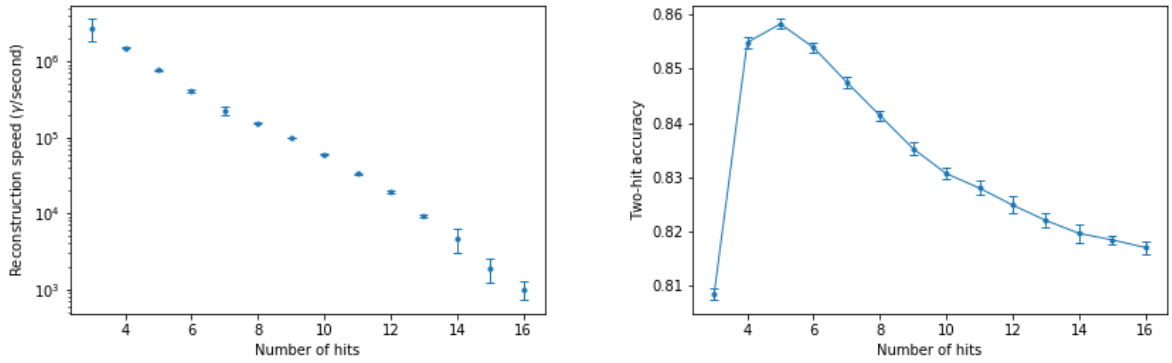


Figure 5.2: Reconstruction speed (left) and accuracy (right) with increasing numbers of simulated hits.

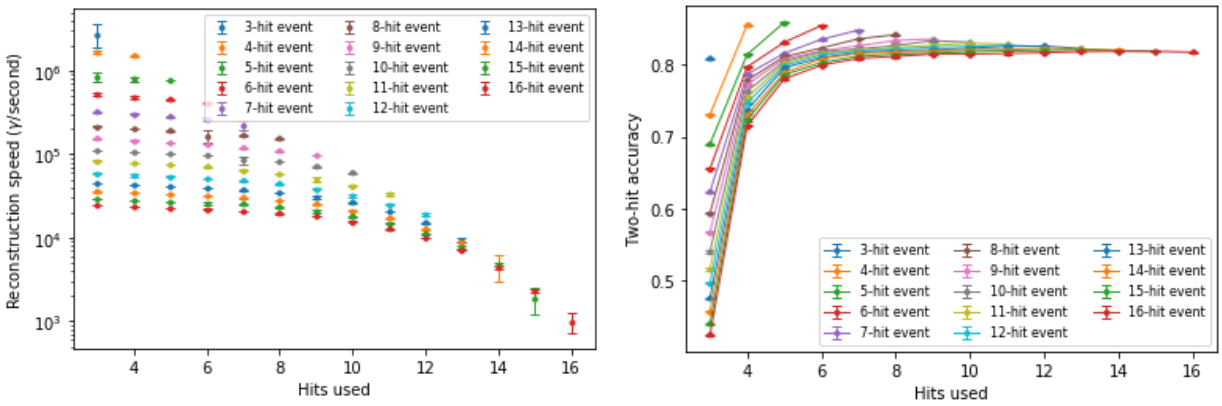


Figure 5.3: Reconstruction speed (left) and accuracy (right) with increasing numbers of reconstructed hits

In plotting the accuracy of our program for increasing numbers of hits (Figure 5.2, right) we find some interesting results in that the accuracy peaks at 5 simulated hits. It is unclear from the algorithm exactly why this is, but it is possible that increasing the number of hits past a certain point introduces more error to our calculation instead of less. If a similar trend holds for testing on more accurate simulators, we can consider weighting our angle reconstructions based on these probabilities in future work.

5.5 Reconstructed hits

We next investigate our program's performance varying both the total number of hits for each event and the number of hits we use in our reconstruction. When we stop computation at a certain number of hits, the ordering with the lowest χ^2-value we have found so far becomes our chosen reconstruction. While this would in theory decrease the computation time for each event, it also has the potential to also decrease the overall accuracy due to the fact that we are not using all the information available to us in our reconstruction. To determine how big the trade-off is in accuracy and reconstruction speed, we simulate each possible combination of total hits versus reconstructed hits and plot one line for each type of event. So, each set of events with N hits is its own line, and each point in the line indicates a different number of hits used in its reconstruction. The results in speed and accuracy are displayed in Figure 5.3.

As expected, our throughput decreases with increasing hits used in reconstruction while accuracy increases. For throughput, this trend is especially pronounced for events with more hits, and seems to drop sharply around 10-12 hits. The opposite is true of our accuracy, with gains tailing off at around 6-8 hits. Beyond this number, using more hits in reconstruction only gives us marginal gains in accuracy while significantly decreasing our speed. It would be reasonable, therefore, to set a maximum number of reconstructed hits for all events, increasing our reconstruction speed by sacrificing a small amount of accuracy. It is not immediately clear from our data what the optimal cutoff value should be, as we would need the correct energy distribution of our source to predict any losses in our overall accuracy. In a real-world test setting, events with high numbers of hits are much less probable than events with low numbers of hits, so our losses in overall accuracy would likely be less severe than those predicted for a certain event type.

Despite these considerations, we can perform some preliminary calculations assuming a flat source distribution (i.e. all event types are equally probable), shown in Figure 5.4. For each possible cutoff value in reconstructed hits, we calculate the residual accuracy, or the accuracy lost by ignoring higher orders of hits. This way, we can decide how much accuracy we are willing to trade for an increase in speed and set the maximum cutoff accordingly. Similar calculations can be done for residuals in speed.

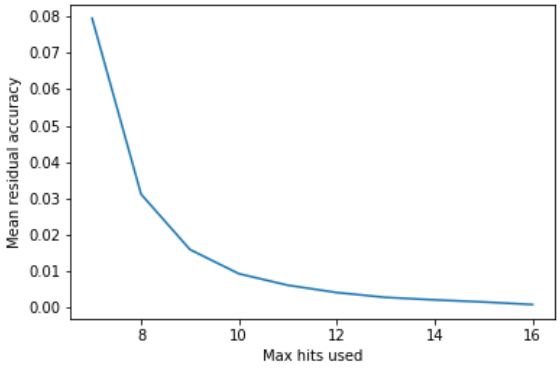


Figure 5.4: Residual accuracy for an increasing max cutoff in reconstructed hits

5.6 Cutoff values

χ^2 cutoff

Changing the p-value of our calculation changes the level of certainty we have in our results. If we use a smaller p-value, we can be more certain that any orderings we exclude are truly bad ones, but we also cannot exclude as many orderings, so we expect that our speed will decrease and our accuracy will increase as our p-values get smaller. Our results confirm this expectation, as shown in Figure 5.5. There is a rather large uncertainty on the last p-value (0.1), but as we have an overall linear trend prior to that value we expect that this is simply due to noise. Over the tested range of p-values, our accuracy has a range of about 5%, while our reconstruction speed only changes by about 1-2%. Based on these tests, it would be most efficient to choose one of the smaller p-values tested for our final calculations, as it would not significantly affect our overall runtime.

η' cutoff

There is no clear trend in our results for η' tolerance, shown in Figure 5.6. Though there are sharp drops in reconstruction speed for certain η' values, this could very easily be due to noise, and the large uncertainties make it difficult to draw any conclusions about an overall

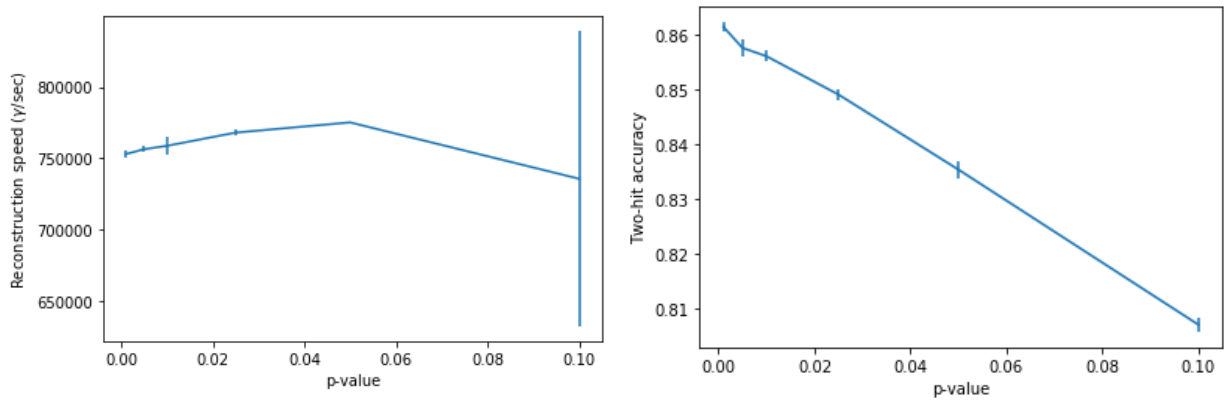


Figure 5.5: Reconstruction speed (left) and accuracy (right) with increasing p-value.

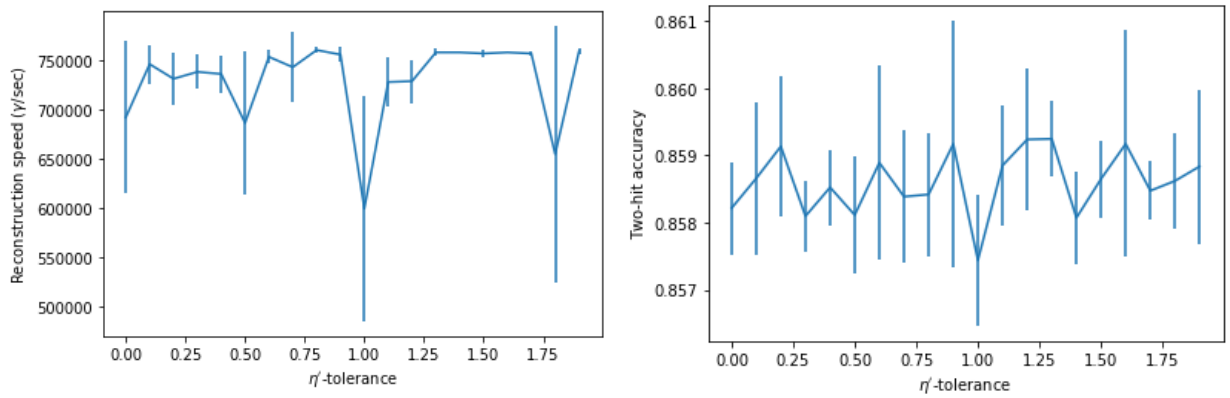


Figure 5.6: Reconstruction speed (left) and accuracy (right) with an increasing η'-cutoff.

trend. In accuracy, there is similarly no clear trend, and the range is so small as to be insignificant for our purposes.

5.7 Noise

Predicted Noise

We simulate different levels of predicted noise by changing the a parameter in our energy calculations and the σ_{sp} parameter in our spatial calculations, and our results are shown in Figures 5.7 and 5.8. There are no significant trends in reconstruction speed when the

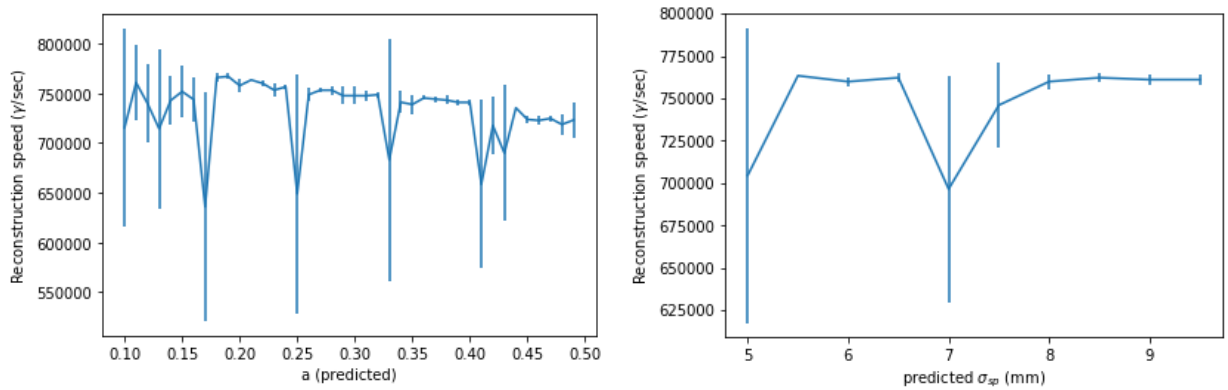


Figure 5.7: Reconstruction speed with increasing levels of predicted noise.

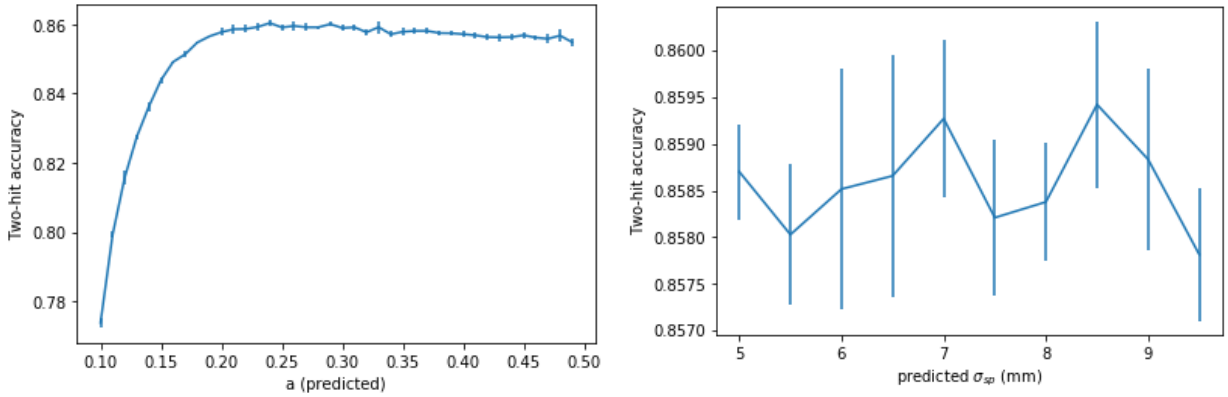


Figure 5.8: Reconstruction accuracy with increasing levels of predicted noise.

simulated noise is changed, which stands to reason as our calculation is the same either way. The same goes for the accuracy of our predicted spatial noise, which has too small a range to be considered an effective parameter. However, the accuracy of our predicted energy noise does show a clear trend which flattens out as the predicted energy noise approaches the true energy noise. If we overestimate the noise, we see the same or very similar accuracy to a correct noise estimate, but if we underestimate the noise we see a severe decrease in accuracy. If this trend holds for more robust simulator data, it could be a good way to probe the true background noise when the telescope is in operation.

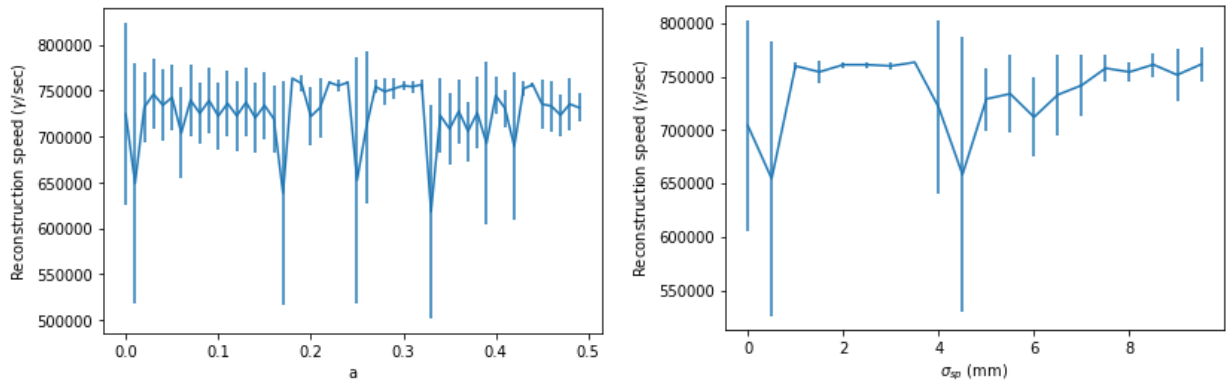


Figure 5.9: Reconstruction speed with increasing simulated noise level.

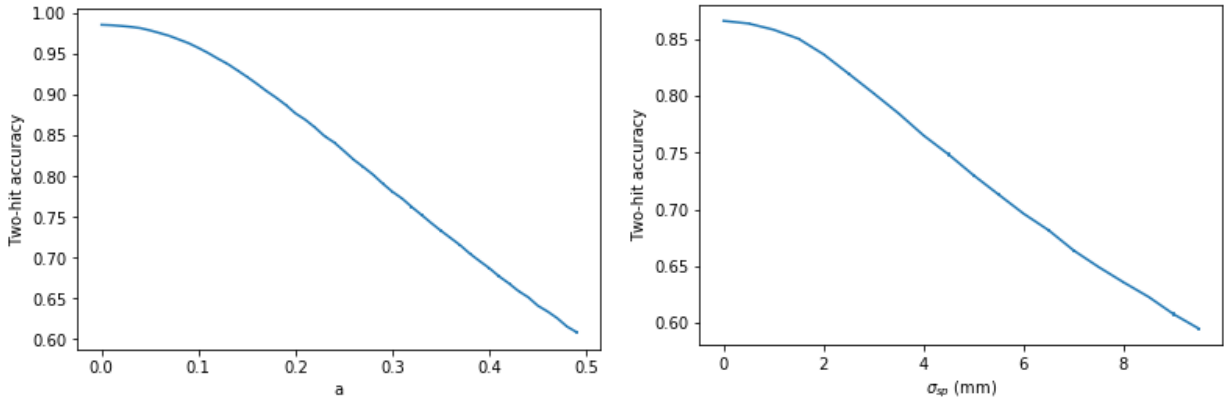


Figure 5.10: Reconstruction accuracy with increasing simulated noise level.

Simulated Noise

We also test our program's speed and accuracy with differing levels of simulated noise, shown in Figures 5.9 and 5.10. We see fairly expected results, that there are no clear trends in reconstruction speed but that the accuracy goes down with increasing noise in our detected events. This is useful as a check on our toy model simulator, as less clear trends in the accuracy might prompt us to evaluate whether our simulator works as we expect it to.

Chapter 6

Conclusion

6.1 Discussion

As part of this project, we developed and tested a model simulator for gamma-ray trajectories, wrote a Compton reconstruction algorithm, and set up a framework for testing and investigating trends in our data. We have met our constraints for power and better constrained our accuracy and reconstruction speeds based on possible input parameters and simulated environmental factors.

Our overall goal was to be able to reconstruct at least 50,000 photons per second with greater than 75% accuracy, and, based on our findings in sections 5.4 and 5.5, we will be able to achieve this goal by implementing a maximum cutoff in the number of reconstructed hits for each event. Based on our data in Figure 5.2, we have also been able to improve the average-case time complexity of our program from factorial to exponential with our tree search algorithm. Our program has, on average, a linear trend in speedup, indicating that it scales well, though we will likely have a maximum of only four cores to work with if the telescope uses a standard ARM processor in its hardware.

Though we saw mostly expected behavior when testing the various parameters, there are a few interesting features which may prove useful to future work on this project. In varying the number of hits for each event, we find an unexpected trend in accuracy that may help us to better reconstruct the position of a gamma-ray source in future iterations of the project, as events with a higher predicted accuracy can be given greater weight in the final source localization. Investigating the reconstructed hits, we find that only using a certain number

of hits per event can save us computation time without significant loss of accuracy. Finally, in varying our predicted noise levels, we find that the point at which the accuracy flattens out is a fairly good predictor of the simulated energy noise in our system. This could help us better determine the true noise level when performing tests with real telescope equipment.

6.2 Future work

Though this project laid the groundwork for the simulation and reconstruction of Compton scatters for our detector, there is much left to do before the telescope becomes fully operational. One of the primary goals of future projects will be to repeat the test cases we have performed on more accurate data from CERN's Geant 4 simulator to make sure that we see the same behavior from both. An alternative approach could be to make our toy model simulator more robust by simulating events from a given gamma-ray source profile and by including the correct probability distributions for Compton scattering angles. Once this is finished, the next logical step would be to include the Klein-Nishina formula as a prior on our reconstruction algorithm, which would prioritize triples of hits more likely to be Compton scatters based on the angle itself, rather than just comparing the energy and spatial angles to see if they match. Compton scatters are much more likely to happen at small angles for our energy range, so including this in our calculations could further improve our reconstruction accuracy.

Testing on a better simulator would also allow us to examine the overall accuracy of our program for different source distributions rather than just the accuracy for events with a given number of hits. In our current tests, all events have equal weight, whereas Compton scatters in nature are much more likely to have 2-3 hits than any larger number. This means that the overall accuracy would be a weighted mean of those we found in section 5.4, with the overall shape of the distribution depending on the emission distribution of the source. With these probabilities taken into account, we would potentially be able to optimize our program for the expected source distribution. Correct figures for the accuracy distribution would also be very useful as a prior on the source reconstruction algorithm.

The use of parallelism in our algorithm also requires some further investigation. The current program runs each photon in parallel, but theoretically each branch of our tree search could

also be done simultaneously. There would be a small amount of overhead associated with the creation of each parallel thread, so at some point the extra parallelism would cease to be worth the decreased speed. Though we are meeting our speed goals currently, it would be interesting to investigate the optimal level of parallelism for a program such as this, in case it is needed in the future.

References

[1] S. Blinder. Klein-nishina formula for compton effect. https://demonstrations.wolfram.com/KleinNishinaFormulaForComptonEffect/, 2009.

[2] Boggs, S. E. and Jean, P. Event reconstruction in high resolution compton telescopes. *Astron. Astrophys. Suppl. Ser.*, 145(2):311–321, 2000.

[3] Buckley, J. H. and Chen, W. The advanced particle-astrophysics telescope. memo, unpublished, 2018.

[4] GLAST Facility Science Team, N. Gehrels, and P. Michelson. GLAST: the next-generation high energy gamma-ray astronomy mission. *Astroparticle Physics*, 11:277–282, June 1999.

[5] C. Inc. Selected values of the chi square distribution. https://www.chegg.com/homework-help/questions-and-answers/table-124-selected-values-chi-square-distribution-area-probability-area-upper-tail-degrees-q28841985, 2019.

[6] J. Keller. Boeing to develop next-generation radiation-hardened space processor based on the arm architecture. https://www.militaryaerospace.com/computers/article/16726308/boeing-to-develop-nextgeneration-radiationhardened-space-processor-based-on-the-arm-architecture, 2017.

[7] W. Lamorte. The role of probability. http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability12.html, Jul 2016.

[8] R. Mushotzky. Lecture notes in high-energy astrophysics. https://www.astro.umd.edu/~richard/ASTR480/GRB_lec_1_2_Astro480.pdf, 2013.

[9] V. Schönfelder, H. J. M. Aarts, et al. Instrument description and performance of the imaging gamma-ray telescope comptel aboard the compton gamma-ray observatory. 1993.

[10] Wordpress. Web- u2 usb meter/monitor upm quick look. https://usbchargingblog.wordpress.com/2018/08/11/web-u2-usb-meter-monitor-upm-quick-look/, Nov 2019.

[11] D. Xu. Gamma-ray imaging and polarization measurement using 3-d positionsensitive cdznte detectors. *University of Michigan, Ann Arbor, MI, PhD Thesis*, 2006.

Event Reconstruction in the APT, Ramey, M.S. 2019