Time-based vertex reconstruction in the Compact Muon Solenoid

Benjamin Bartlett,[1, *] Lindsey Gray,[2] Adolf Bornheim,[1] and María Spiropulu[1]

[1] *California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, USA*
[2] *Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510, USA*

The Phase-II upgrades to the Large Hadron Collider will introduce a variety of new measurement devices to the CMS, including the High-Granularity Calorimeter (HGCAL). The increase in luminosity from these upgrades will also have the undesired side effect of vastly increasing pileup to a level at which the current machine learning vertex reconstruction (vertexing) algorithms cease to be effective. This will necessitate the development of further vertexing algorithms. Using high precision timing measurements from simulated events in the HGCAL, we design a vertex reconstruction algorithm that requires only the spatiotemporal arrival coordinates to reconstruct the interaction vertex of a collision with sub-millimeter resolution. We also analyse how particle energy and simulated time smearing affect this resolution and we apply this algorithm to more realistic $H \to \gamma\gamma$ sets. To do this, we implement a set of filters to remove poorly-reconstructed events and introduce a new algorithm capable of reconstructing interaction vertices given the pointing data and arrival data of a single cluster. Progress on this work was ultimately hindered by extensive errors in the clustering algorithms used the generation of the datasets; should these errors be resolved, further work would include integration with tracker information and the application of these algorithms to high-pileup scenarios and QCD jets.

I. INTRODUCTION

The recent upgrades to the Large Hadron Collider (LHC) have doubled the collision energy of the accelerator, increasing it from 8TeV to 13TeV, while proposed Phase-II upgrades following the next shutdown will increase the beam luminosity by an order of magnitude.[1] A side effect of both of these changes is increasing pileup - when multiple collisions happen sufficiently close enough in space and time such that it is difficult to associate the final particle states with the vertex from which they emanate. This high pileup environment poses particular challenges to distinguishing the two major production mechanisms for the Higgs boson - gluon fusion and vector boson fusion (VBF). Separating the two production mechanisms is of crucial importance for precisely understanding electro-weak symmetry breaking, among many other topics.

A novel approach, the development of which is the emphasis of this project, uses the space and time coordinates of arrival of the particles in the detector to reconstruct the interaction vertex. With a timing measurement precision of approximately 100ps or better, as attained for the first time by the Caltech CMS group[2], one can exploit the time of flight information to reconstruct the origin (interaction vertex) of the particles and test the consistency of the arrival time in the ECAL using a four-dimensional triangulation method.

Initial studies using similar techniques have been previously carried out in 2012[3] and 2014[4]. We continue these proof-of-concept studies with a new, cleaner implementation designed independently of these studies. We eventually extend this algorithm's applicability to a larger and more realistic data set, such as $H \to \gamma\gamma$ and the forward η (pseudorapidity) and lower p_T (transverse momentum) regime relevant to VBF.

To implement this on these more complex sets, we also study shower structure and evolution, specifically with the aim of investigating the feasibility of single-cluster vertexing algorithms and the efficiency of several general event filters in improving reconstruction resolution.

II. TIME-BASED VERTEXING MODEL

The time-based vertexing ("tVertexing") model is, at its core, very simple. Consider a collision emitting two particles. Let t_1 and t_2 be the recorded times of arrivals of the particles at the recorded locations (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively, as shown in Figure 1.

Note that there are multiple possible metrics by which

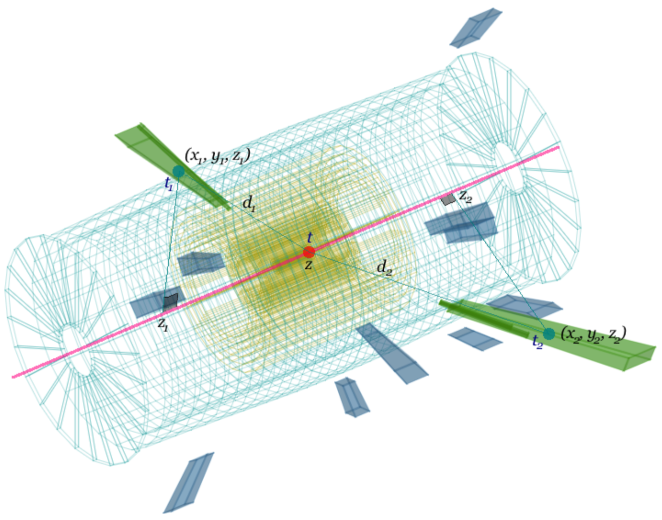

FIG. 1. Space and time arrival coordinates overlaid on a rendering of a $H \to \gamma\gamma$ event. Green clusters represent recorded hits and energy distributions in the ECAL (photon interactions), while blue clusters represent HCAL interactions.

we can measure the arrival time and coordinates of a particle in the detector. For the purposes of this study, we use the numerical average of the spatiotemporal arrival coordinates of all recorded hits ("rechits"), though previous studies have found energy-weighting the numerical average to improve performance[4,5], though this was found to be problematic when applied to various simulated datasets. For more on this, see Section VIII A.

We can assume that the two particles travelled approximately at c from the vertex origin in a straight line, an assumption which holds true for photons and high-energy massive particles. We also assume the transverse diameter of the beam is negligible (typically this is on the order of a few micrometers). Euclidean geometry gives us that:

$$t_1 - \frac{\sqrt{x_1^2 + y_1^2 + (z_1 - z)^2}}{c} = t = t_2 - \frac{\sqrt{x_2^2 + y_2^2 + (z_2 - z)^2}}{c} \tag{1}$$

Given this, we can simply solve for z to reconstruct the vertex of the event. This, of course, will produce two possible solutions, so we pick the one that better agrees with the observed data by selecting the solution z_0 that minimizes

$$\Big| c \cdot (t_1 - t_2) - \sqrt{x_1^2 + y_1^2 + (z_1 - z_0)^2} + \sqrt{x_2^2 + y_2^2 + (z_2 - z_0)^2} \Big|. \tag{2}$$

In tests on the 500GeV γ-gun set, this selects the more accurate of the two vertices 99.95% of the time. The remaining 0.05% of the time can be attributed to detector uncertainty in events with theoretical solutions very close to each other and the actual vertex.

A. Selection of Arrival Hits

For initial demonstration purposes of this algorithm in no-pileup (0PU) events, we use the arrival coordinates of the most energetic cluster in the most energetic region of interest. For photons, the initial test candidates, the energy tends to be clustered tightly in one main cluster, with a few smaller outlying clusters representing possible premature interactions in the tracker.

The "HGCROI-formatted" datasets used in this project have recorded hits that were clustered using the Pandora clustering algorithm[6] (this was found to be problematic eventually, as discussed in Section VIII B). The clusters have centroid properties (among much other information) which can be used in the tVertexing algorithm. However, using the flat-averaged mean of x, y, z and t from the rechits in the selected cluster was found to improve the resolution by roughly an order of magnitude on tests in various datasets. We initially attempted to energy-weight the x, y, z coordinates of the arrival hits. However, this also proved to worsen the overall accuracy of the vertexing algorithm. It is likely that this is due to a defect in the generation of the simulated events; see Section VIII A for more details.

In calculating the arrival coordinates of the cluster, we select only the hits which have timing data to them (a small minority of the hits, generally corresponding to the more energetic ones) and that occur within a 3×3 cell window on the x, y plane of the energy centroid for each layer in the HGCAL. This helps ignore lower energy, less relativistic resultant shower particles that could worsen the arrival coordinate estimates of the cluster.

III. IMPLEMENTATION ON γ-GUN DATASET

The first datasets we checked the performance of our algorithm against were high energy γ-gun datasets. These contained the detector readouts of a simple simulated zero-pileup diphoton system at various energy levels and contain the full simulated showering interactions of the particles in every layer of the tracker and HGCAL. We started with these sets because they are the cleanest ones that allow us to solve many of the detector-related problems, such as energy dispersion within the physical layers of the detector, energy weighting, possible premature splitting of the photon in tracker interactions, etc.

The algorithm was found to perform very well on these datasets, with a median error of 0.24mm and $\sigma = 0.35$mm, as shown in Figure 2. Given the 5ps time binning, with $c \cdot 5\text{ps} \approx 1.5$mm, and that there are on the order of 10^2 rechits involved in the calculation of the vertex per cluster (the hits with timing data), we would expect the optimal possible performance of this algorithm to be in the 10^{-1} mm range, which it is currently performing in.

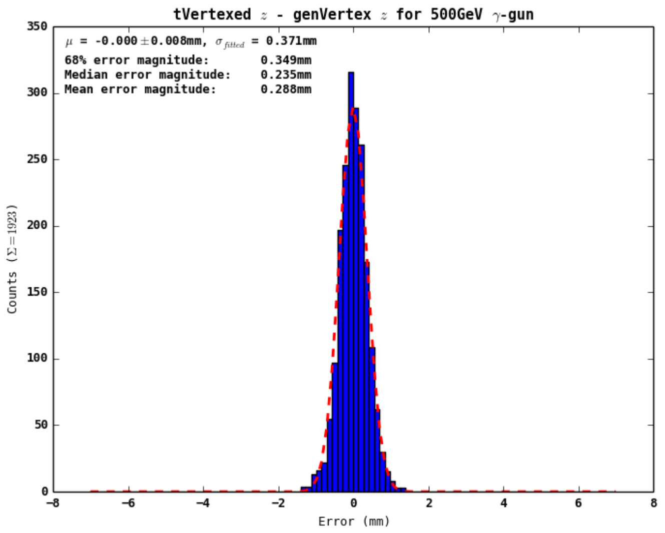

FIG. 2. Results of the tVertexing algorithm applied to a large γ-gun sample dataset, with the worst 1% of events trimmed to allow for a better Gaussian fit.

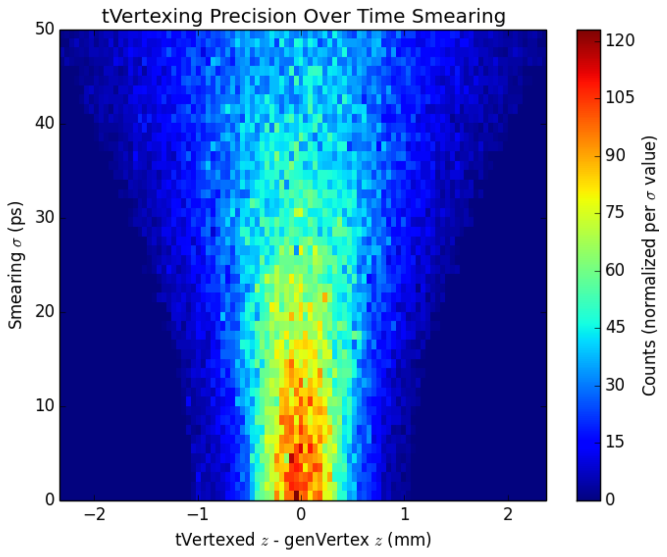

FIG. 3. Histogram showing frequency of vertexing error occurrence in the 500GeV γ-gun dataset across a range of simulated time-smearing values with the worst 1% of event reconstructions trimmed. Time smearing values are computed by applying a Gaussian smear to each individual rechit with μ as the real time-value of the hit, and σ as the value represented on the vertical axis.

In this vertexing, we discard any events without two distinct regions of interest and events which have low η separations. The reasoning for the latter condition is that very low η values increase the sensitivity of the vertexing algorithm to detection error, time binning, etc., particularly for low η separation at high values of η.

IV. EFFECT ON RESOLUTION OF SIMULATED TIME SMEARING

Our subsequent attentions were focused on analysing the effects of simulated time smearing on the resolution of the algorithm. The simulated detector readouts used in this algorithm incorporate detector time binning, discrete layer spacing, etc., but do not take into account other types of detector peculiarities, such as noise and time smearing (the variance around the actual time of a hit that the detector will report).

We analysed the same 500GeV dataset from Section III, adding a random Gaussian time smearing to each hit with μ as the actual time of arrival and σ ranging from 0-50ps. The system seemed to be remarkably stable to hit-based time smearing, with a full 50ps time smearing only doubling the median error from an unsmeared, 5ps-binned vertexing run, and tripling the median error for 50ps time smearing, as shown in Figures 3 and 4.

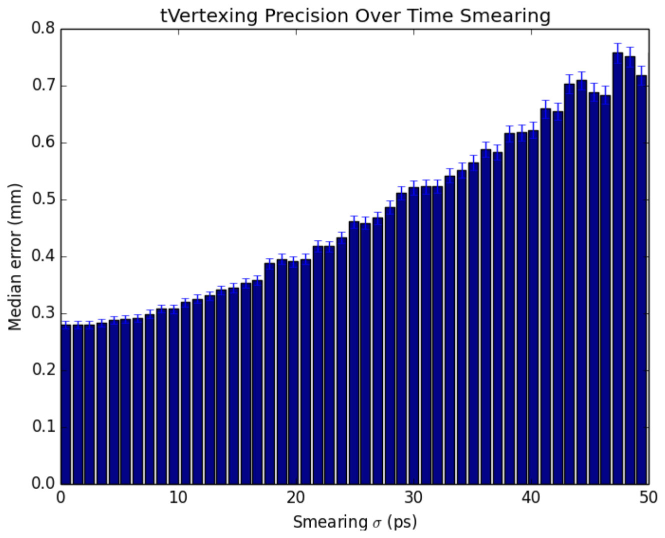

FIG. 4. Median errors of vertexing results for the same 500GeV γ-gun dataset shown in Figure 3 over the same range of time smearing values.

V. ENERGY DEPENDENCE OF VERTEXING RESOLUTION

We would expect the vertexing algorithms we develop to be more performant with higher-energy data sets, since an increase in energy would give an increase in the number of points with timing data (requiring a soft energy threshold), increasing the sample size from which to draw a conclusion about the space and time arrival coordinates.

Specifically, we would expect the resolution to be pro-

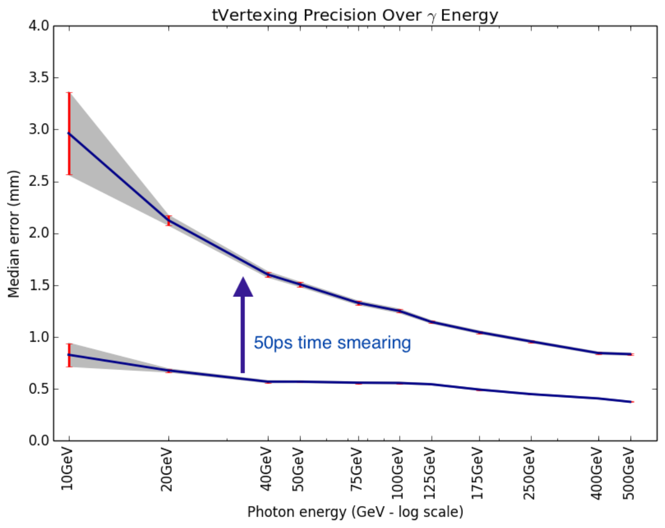

FIG. 5. Precision over photon energy of the tVertexing algorithm on γ-gun data sets at time smearing values of 0ps (lower) and 50ps (upper). Energies lower than 10GeV did not have sufficient timed hits ($\xi(E)$) to reconstruct an interaction vertex.

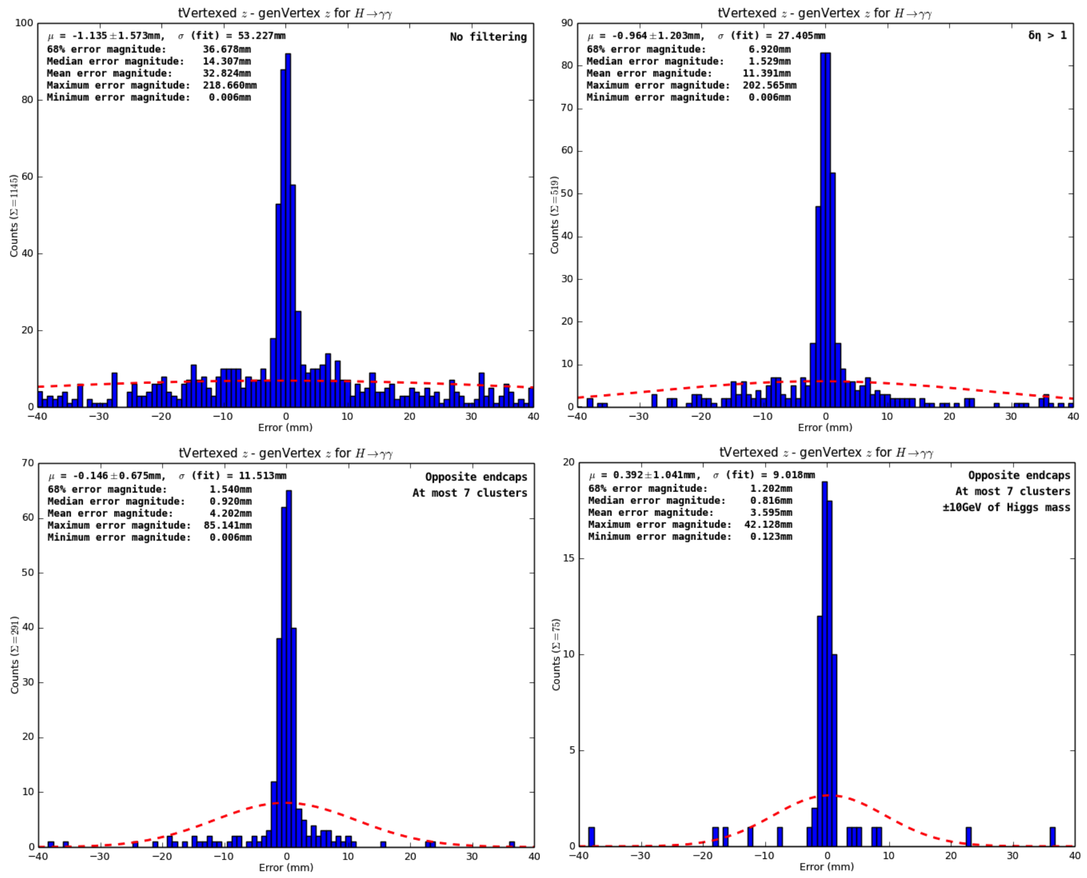

FIG. 6. Comparison of vertexing accuracy with various event filters in 0PU $H \to \gamma\gamma$. From left to right, top to bottom: a) no filtering, b) minimum η difference of 1, c) requirement for opposite endcaps ($\eta_1 > 0$, $\eta_2 < 0$) and a maximum of 7 clusters per event, and d) opposite endcaps, a maximum of 7 clusters, and invariant mass of $m_H \pm 10$GeV. While this filtering method is very effective at removing poorly-vertexed results, it also discards a large portion of the original dataset, increasing the accuracy, but decreasing the efficiency.

portional to $\frac{1}{\sqrt{\xi(E)}}$, for E the energy of the cluster and ξ some relation between cluster energy and number of timed data points. We would expect ξ to be roughly linear, but we have not yet had the chance to study this.

Qualitatively, Figure 5 (with energy log-spaced on the horizontal axis) seems to follow this behavior in that the resolution follows a monotonically decreasing convex curve along increasing energy, though the error bars at lower energies make this observation difficult to quantitatively verify.

VI. IMPLEMENTATION ON $H \to \gamma\gamma$

The next datasets we studied were 0PU $H \to \gamma\gamma$ sets. These are also very clean sets, like the γ-gun sets, but a few extra complications exist, such as pion generation, variation in photon arrival locations (barrel/endcaps), and various problems caused by hadronic interactions and clustering.

Initial attempts at vertexing these sets performed relatively poorly. Though a tight and well-developed core about $\mu = 0$ was present, a huge number of incorrectly vertexed events also existed.

The reasons for the initial poor performance are not certain; possible reasons could include noise created by high-energy pions from premature interactions from the $H \to \gamma\gamma$ photons or (more likely) incorrect clustering of rechits with the Pandora clustering algorithm, as discussed in Section VIII B.

To solve this, a few filters were implemented (aside from the initial filter that there are two high-energy clusters in the HGCAL endcaps). First we required there to be an η separation of at least 1. While this did reduce some of the incorrectly vertexed events, a large portion of the events that were incorrectly vertexed were found to be in the high-η regime, where η separation is less meaningful. Thus, we replaced this condition with the condition that the photons selected for vertexing needed to be in *opposite* endcaps, which further reduced the problem.

We also required that there be at most 7 clusters in the event, since a large number of clusters could indicate premature interaction with something before detection, which would tend to invalidate the results.

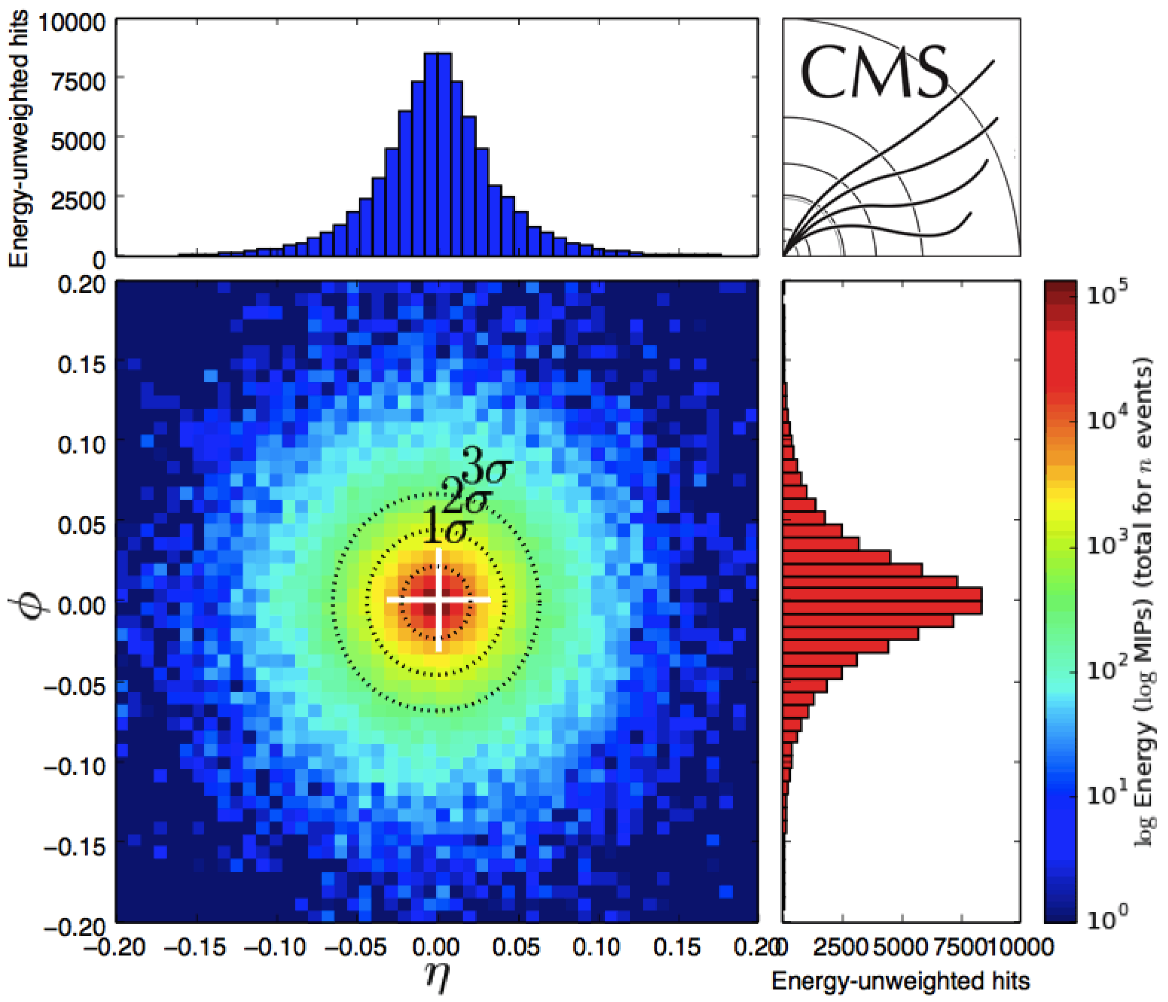

FIG. 7. Energy-weighted and unweighted hit values for a sample layer (layer 24/30) in the HGCAL endcaps for a total of 3992 photons. The 2D histogram shows a normalized η, ϕ map of the energy-weighted hits in a given layer, while the side plots show a cross-sectional unweighted hit map for the same event, with the top being a map over η and the right being a map over ϕ. The energy of the hits in a cluster tends to be more tightly centered in the cluster compared to the comparatively spread unweighted hit values. The overall tightness of the distribution ($\sigma_\eta, \sigma_\phi \approx 0.02$) indicates pointing algorithms may be a viable choice for use as preliminary filters for the tVertexing algorithm, as the length (pointing axis) of the cluster is much larger than the spread of the cluster (transverse pointing axis). An animated version of this figure is available in Appendix B.

Our final requirement was that the two photons selected for vertexing had an invariant mass, given by

$$m^2 = \Big(\sum_i p_{T_i} \cosh \eta_i\Big)^2 - \Big|\Big|\sum_i (p_{T_i} \cos \phi_i + p_{T_i} \sin \phi_i + p_{T_i} \sinh \eta_i)\Big|\Big|^2 \qquad (3)$$

that was within 10GeV (or any arbitrary range) of $m_H \approx$ 125.09GeV. The (untrimmed) results of these filters are shown in Figure 6.

VII. POINTING-BASED VERTEXING MODEL

Though the tVertexing algorithm gives an excellent resolution when it is fed the proper cluster information, a major disadvantage is that it requires two clusters created from two distinct particles to reconstruct the interaction vertex. Here we introduce a secondary algorithm which can estimate the interaction vertex given only the spatial and timing data of a single cluster. We call this algorithm “pVertexing”, as it primarily uses the pointing information given by the shape of a single cluster to reconstruct the vertex.

Current algorithms like this exist using a principal component analysis on the shape of the cluster (but no timing information) to attempt to reconstruct the vertex, but these tend to have a very poor resolution, on

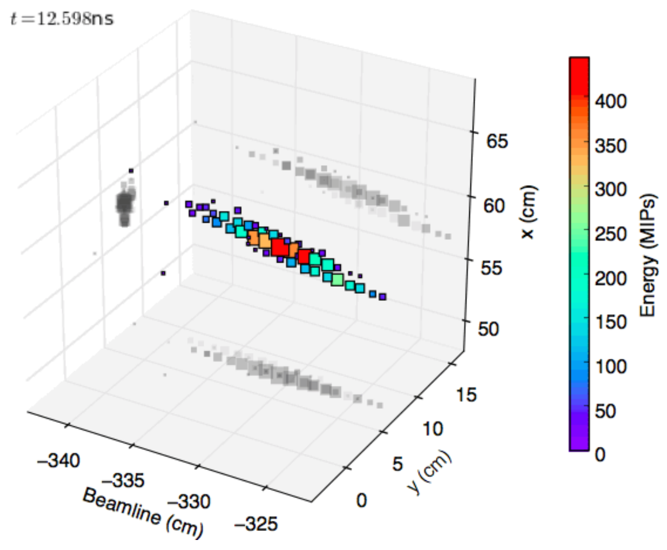

FIG. 8. Three-dimensional projection of the energy-weighted rechits contained in a single cluster in one of the HGCAL endcaps. The high linearity of the cluster gives reasonably good pointing data. An animated version of this figure showing rechit arrivals over time is available in Appendix B.

the order of a few centimeters.[5] These extract longitudinal data from the relatively high axis length to transverse axis length, shown in projected 3D in Figure 8 and use the pointing data to trace back to the location closest to the beamline.

The algorithm performs a BFGS nonlinear least squares minimisation to solve for the interaction vertex of a single cluster. We give an initial guess in cylindrical coordinates of

$$\begin{cases} \theta = \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right) \\ \phi = \arctan\left(\frac{y}{x}\right) \\ z_0 = 0 \\ t_0 = 0 \end{cases}$$

and fit the x, y, z, t arrival data to the functions

$$\begin{cases} x_f = (t - \delta t) \cdot c \sin\theta \sin\phi \\ y_f = (t - \delta t) \cdot c \sin\theta \cos\phi \\ z_f = (t - \delta t) \cdot c \cos\theta + z_0 \end{cases}$$

(with δt the time of interaction). We then numerically minimise the error function $\varepsilon = (x_f - x)^2 + (y_f - y)^2 + (z_f - z)^2$ via a least squares method.

A. Implementation

The algorithm was tested on the same 500GeV γ-gun sets in Section III, yielding interesting results. The pVertexing algorithm gave a median error of 0.338mm, almost as good as the results from Section III, when it was run on each of the two "main" (highest energy) clusters of

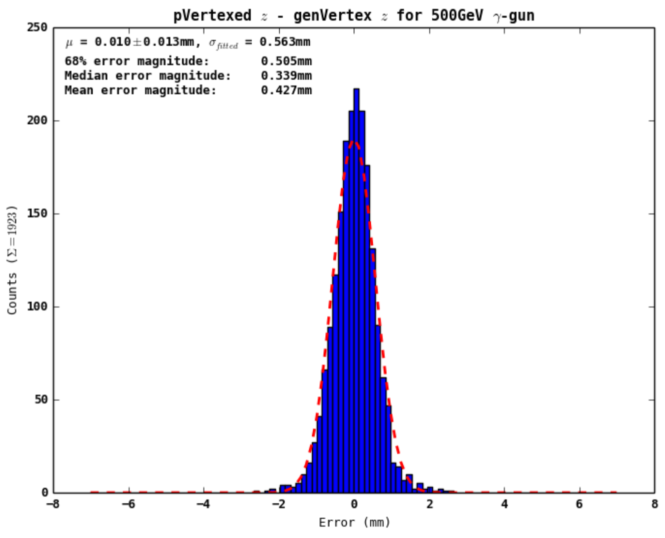

FIG. 9. Accuracy of the pVertexing algorithm when numerically averaged over the two vertices reconstructed from the two primary clusters. The median error magnitude is 0.339mm, almost as accurate as the tVertexing algorithm. However, note that this technically holds no advantage over the tVertexing algorithm, as two clusters are still required to give accurate results, as the results in Figure 10, while technically still at a usable resolution, are obviously flawed.

the set and the results were averaged, as shown in Figure 9. However, each individual cluster pVertexing seems to have a median offset of ±3.5mm, shown in Figure 10.

The obvious structure shown in this figure (roughly symmetric peaks about an origin with exactly zero hits on it), combined with the high accuracy of the results when

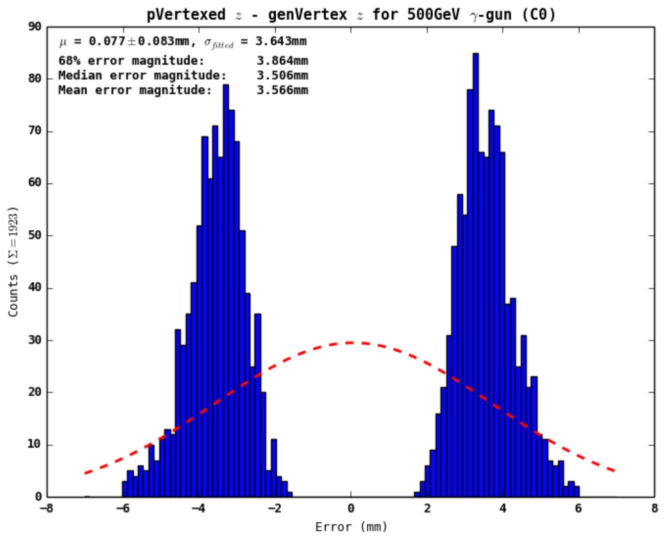

FIG. 10. Accuracy of the pVertexing algorithm using only the first primary cluster. The median error magnitude is 3.864mm, and while this is still a usable resolution, the characteristic widths of each half of the curve is much smaller than this value, with $\sigma_{\text{half}} \approx 0.7$mm.

the individual pVertexed results are averaged means that these offsets always happen in pairs, with one cluster being offset in either the positive or negative z direction, while the other is offset a near-equal amount in the opposite direction. This suggests either a systematic error with the algorithm or an error with the data set, such as a spatial offset of the data along z, though as of the writing of this analysis note, we have not been able to locate this error.

The resolutions of each half of the results shown in Figure 10 are actually quite good: $\sigma \approx 0.7$mm. If we were able to isolate what was wrong with the data sets or algorithm, this would potentially be a very effective secondary algorithm for use with the tVertexing algorithm for removing pileup and other unwanted interactions. The dependency of the algorithm on only one cluster would allow us to generate probable regions of interaction for each observed particle (cluster), then use the tVertexing algorithm on pairs of them to obtain a finer resolution.

The primary disadvantage of this algorithm, however, is that it requires the interaction time of the collision. While the tVertexing algorithm relies on the difference of arrival times of two clusters, the pVertexing algorithm looks at only one set of arrival times, requiring the production time.

A possible solution to this would be to use tracker information to generate a set of possible collision times and choose the one that produces the lowest error when minimising ε, which takes both the spatial structure and arrival time data into account.

VIII. PROGRESS IMPEDIMENTS

A. Energy Weighting

Previous studies[4,5] have shown in various types of vertex reconstruction algorithms that resolution tends to be optimized by energy-weighting the spatial positions of the recorded hits per cluster. The specific energy weighting scheme found to yield optimal results in[5] was an arithmetic mean of $\log\left(\frac{E}{E_{\text{total}}}\right) + k$ for some cutoff threshold k (in Monet's study, $k = 3$, though this value requires further optimization and may vary by photon energy). The time coordinates are not energy weighted due to variability in energy deposition amount by layer, though this has not been thoroughly studied.

We were unable to replicate these results on the simulated HGCAL datasets. Specifically, when using even a simple linear energy weighting on the spatial coordinates, the resolution was worsened by over an order of magnitude, depending on the interaction type and energy in the dataset. This dramatic worsening of performance for what should be a simple modification indicated an error in the generator-level energies in the datasets we were using, which have had a history of various errors, including incorrect generator-level vertices and previous errors with generator-level energies, as discussed in the next section. A large number of other weighting schemes were tested as well, ranging from log weighting to exponential weighting; none performed better than a simple unweighted numerical average.

B. Clustering Algorithm

While searching for further energy problems in the generation of the datasets, we discovered a large problem with how Pandora was clustering the rechits. Specifically, multiple clusters from distinct arrival events would frequently be merged into an overly large single cluster. The present HGC clustering using Pandora fails to preserve the finer details of energy deposits in more complex environments. In particular, even in zero pileup, photon clusters can be contaminated by hadron remnants of the proton-proton collision, and the existing clustering algorithms can often group together energy that should be in different separated into disjoint clusters. This is an artifact of all clustering algorithms used in the HGCAL so far being completely binary as to whether or not the include a reconstructed hit in a cluster.

Since multiple particles can overlap even in simple LHC collisions, especially in the high-η endcaps, it is easily possible to distort clusters unless the algorithms that build them are aware of how to share energy between overlapping energy deposits. Once this is achieved and implemented in the HGCAL clustering code, a better interpretation of the particle content impacting the end-cap calorimeter can be realized, and more delicate quantities, such as the cluster time, can be preserved. This is now an ongoing task in the HGCAL software group and will likely take the next six months to achieve.

Since the vertexing algorithms written in this study use the detector-level information of the simulated datasets, which is dependent on the Pandora clustering algorithm, we suspect this clustering problem is the source of the energy-weighting problems discussed in the previous section. Additionally, it is quite possible this contributed to some of the noise exhibited in Figure 6, both before and after the filters were applied. This would not be present in the simulated γ-gun sets shown in Figure 2, as the hadronic proton-proton remnants of the collision are not simulated. Given proper clustering, we can expect the $H \to \gamma\gamma$ results (Figure 6) to conform much more like the results of the γ-gun sets (Figure 2) with submillimeter median resolution.

IX. DISCUSSION

The huge increase in pileup that will accompany the Phase-II upgrades to the LHC[1] will require as many vertex reconstruction constraints as possible maintain a reasonable vertexing resolution and efficiency. The current boosted decision tree algorithms used for tracker vertex reconstruction break down horribly in high-pileup environments (with a vertex assignment accuracy rate of around 30% in 140PU) and are unable to track uncharged particles.

In this study we have provided two new algorithms capable of reconstructing interaction vertices from charged or uncharged particles to submillimeter precision independently of any tracker information. While this precision is still about an order of magnitude worse than the current tracker resolution even in zero-pileup datasets, this algorithm could, ideally, be used in conjunction with the current tracker algorithms to better assign vertex locations in high-pileup environments.

The algorithms were found to reconstruct vertices in low-pileup datasets to a median precision of about 300μm and is quite resilient to simulated time smearing, though the simulated hit-based Gaussian smearing might not be an accurate model of the time smearing present in the HGCAL. Our results also indicate these algorithms retain a useful resolution at energies as low as 10GeV, with the resolution roughly doubling in the absence of time smearing and tripling with 50ps time smearing applied.

Though the errors found in the current CMS clustering algorithms (discussed in sectionVIII B) prevented us from extending this study to QCD jets and high-pileup environments, a combination of tracker data to estimate likely times of collisions, single-cluster vertexing with the pVertexing algorithm, and two- or multiple-cluster vertexing with the tVertexing algorithm or a slightly modified variant using a numerical minimisation procedure may prove superior to the current vertex reconstruction algorithms when used in high-pileup environments.

A. Future Work

A variety of future work can be performed based on or to further this research.

Of course, the first major step to continue this research is the application to pileup and QCD jet datasets. Unfortunately, this will require fixing the clustering errors in the Pandora algorithm, as described in Section VIII B, a comparatively giant task.

Several shower slimming methods developed by Flamant, et. al.[4] can potentially improve vertexing resolution by attaining a better definition of arrival time.

Integration with tracker data could allow for an estimation of interaction time, allowing for a preliminary pass of the pVertexing algorithm over data sets to give initial vertex estimates. A secondary pass of the tVertexing algorithm using clusters that appear to have common vertices could better reveal vertex locations. This could potentially be used in conjunction with the current boosted decision trees to more precisely narrow the vertex locations and assign more likely and pileup-resistant probability distributions.

The timing information used in these algorithms may also assist in reconstructing vertices created in the tracker by photon conversions, long-lived particles, or bremsstrahlung.

Appendix A: Algorithm Details

Here we discuss in much greater detail how the file system for this project is organized and how the important programs work. Additional documentation can be found in the actual source code (normal comments along comments side-aligned at the 100 character mark to give context), as well as in the “Documentation” directory.

1. Vertexing.py

This “master” file contains the main set of vertexing algorithms and is used to vertex data sets and run various tests described in this paper, such at time smearing, energy resolution analyses, etc. Below is a description of each of the functions in this module in the order in which they appear in the source code (roughly organised by topic).

`Writer()`: Helper class for making multiprocessed progress bars.

`timeSmearing()`: Simulates time-smearing in the rechit arrival times by applying a Gaussian with μ as the actual arrival time and σ as the input value.

`timeSmearingTest()`: Iterator function used to run a resolution analysis over time smearing σ values and keep track of the errors. Calls `Plotter.tVertexErrorHist2D()` to create a plot like the one in Figure 3.

`tVertex()`: Given centroid locations and arrival times of two clusters, triangulates their interaction vertex, returning the physical vertex location and the relative time of interaction.

`XYZtoEtaPhi()`: Converts arrays of x, y, z coordinates to an η, ϕ map using the transformation

$$\begin{cases} \phi = \arctan\left(\frac{y}{x}\right) \\ \eta = -\log\left(\frac{\sqrt{x^2+y^2}}{2z}\right). \end{cases}$$

`getClusterArrivalTimes()`: Averages the arrival times of rechits in a cluster. While the cluster data in the root files includes an arrival time for the cluster, we believe this time to be faulty in the sets, as it always increases the error by an order of magnitude or so when used.

`getClusterXYZ()`: Computes the x, y, z arrival coordinates. The `(cluster.centerX,cluster.centerY,cluster.centerZ)` coordinates were found to give faulty arrival data that greatly increased the median error. Arrival coordinates are computed with a flat average over x, y, and z, since the energy data in some of the sets seems to be bugged.

`pVertex()`: Performs a BFGS nonlinear least squares minimisation to solve for the interaction vertex of a single cluster. We give an initial guess in cylindrical coordinates of

$$\begin{cases} \theta = \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right) \\ \phi = \arctan\left(\frac{y}{x}\right) \\ z_0 = 0 \\ t_0 = 0 \end{cases}$$

We fit the x, y, z, t arrival data to the functions

$$\begin{cases} x_f = (t - \delta t) \cdot c \sin\theta \sin\phi \\ y_f = (t - \delta t) \cdot c \sin\theta \cos\phi \\ z_f = (t - \delta t) \cdot c \cos\theta + z_0 \end{cases}$$

(with δt the time of interaction) and numerically minimise the error function $(x_f - x)^2 + (y_f - y)^2 + (z_f - z)^2$ via a least squares method.

`getClusterPointer()`: Deprecated method to perform a similar procedure to a PCA analysis to retrieve pointing data.

`pVertexPCA()`: Deprecated method. Uses the PCA data from the clusters to reconstruct the vertex location.

`gammaGunFilter()`: Applies a set of filters to γ-gun events. Similar to `HggFilter`, but separately modifiable.

`HggFilter()`: Applies a set of filters to a $H \to \gamma\gamma$ data set, including selecting for at least two ROI's, using at most n clusters, selecting events with cluster energy criteria, filtering events by invariant mass, and imposing η separations.

`HiggsInvariantMassFilter()`: Returns a boolean representing if the invariant mass of an event is within ε of m_H.

`invariantMass()`: Calculates the invariant mass of the two most energetic regions of interest in an event, as shown in equation 3.

`fourVector()`: Simple class with `add()` and `dot()` properties that represents the energy-momentum 4-vector of a physics object. Used in invariant mass filtering.

`energyResolution()`: Multiprocessed function for analysing resolution as a function of energy, such as in Figure 5.

`singleThreadedEnergyResolution()`: Singly-processed version of `energyResolution()` which is easier to debug.

`ROIEnergyAnalysis`: Deprecated troubleshooting function testing the linearity of the observed energy summed over all rechits in a ROI compared to the generator-level energies.

`genEnVsEn`: Another troubleshooting function testing detector energy response linearity.

`vertexData`: Main "vertexing loop" of the program. This vertexes data sets using pVertexing and tVertexing methods, keeps track of errors, and can be used to plot relations, calculate statistics, trim events, etc.

2. Plotter.py

This set of functions contains all relevant plotting procedures to the algorithm; it does very little computation itself. This program makes extensive use of libraries that, as of August 2015, are not contained in lxplus by default, so it either needs to be run locally or in a virtual environment. We briefly discuss each function below in the order they appear in the code.

`vertexPlot()`: Initial function used to plot a crude representation of a vertex location relative to the size of the CMS. Mainly used for visualization and sanity checking purposes.

`showerAnimator()`: Plots an animated 5D (x, y, z, t, energy) graph and renders it to an animated gif. Each frame is rendered independently as a 3D scatter plot, with the color and size of each point adjusted relative to its energy. By default, shadows are also rendered on the xz, yz, and xy axe. The frames are combined into a gif using the ImageMagick plugin. If you don't have this plugin, you can still render the frames and use an external program to combine them. The `delete` option allows you to delete the frames when the combined gif is made.

`XYZtoEtaPhi()`: Simple function to convert x, y, z coordinates to an η, ϕ map.

`layerVarianceFrame()`: Function that renders the individual frames for `layerVarianceAnalysis()` like the one shown in Figure 7. Four subplots are constructed to contain the 2D histogram, the two side plots, and the colorbar. 1σ, 2σ, and 3σ curves are drawn on the data, and the background is colored the same value as the zero value (the background is normally white; this is an aesthetic preference), and the frame is saved as an image in a folder.

`layerVarianceAnalysis()`: Invoker function to call `layerVarianceFrame()` repeatedly to generate a layer-by-layer animation of energy distribution in the detector cells. Combined necessary data at the beginning of the function to a single properly formatted array using a horribly inefficient, lazy, one-time use method, then passes this data to `layerVarianceFrame()` and generates a new frame for each layer, finally combining the frames into an animated gif using ImageMagick.

`tVertexErrorHist()`: Plots an error histogram for vertexed z values, calculates some relevant statistics, fits a scaled Gaussian to the curve, and plots the data, as shown in Figure 2.

`tVertexErrorHist2D()`: Plots a 2D error histogram for the vertexed z values along space and some other axis, such as time smearing, as shown in Figure 3.

`smearingBarPlot()`: Plots median errors of vertexing algorithm with respect to time smearing σ, as in Figure 4.

`energyResolutionBarPlot()`: Plots median errors of vertexing algorithm with uncertainties as a function of particle energy.

`energyResolutionLinePlot()`: Plots median errors of vertexing algorithm with uncertainties over particle energy as a line plot, as shown in Figure 5.

`energySpectrum()`: Plots p_T spectrum for a dstribution.

`fourVector()`: Mathematical 4-vector for calculating invariant masses of systems.

`invariantMass()`: Calculates the invariant mass of a two-particle system. Copied from file `Vertexing.py`.

`invariantMassDistribution()`: Plots a histogram of invariant masses for a (usually filtered) event.

`invariantMassErrorPlot`: Plots error magnitude as a function of invariant mass, used in analytics.

`sumEnergyVsGenEnergy()`: Plots the sum of energies in a cluster against the generator level energies of the cluster. Used for verifying a linear maximum energy response in the HGCAL, which helped with debugging some errors.

3. RootProcessor.py

The purpose of this script is to convert HGCROI-formatted root files from `EDMProcessor.py` to an identically-structured numpy array, which is faster and easier to work with for computational purposes.

This script is relatively straightforward. After some initial argument parsing, we load the necessary platform-specific root libraries and open a new processing job for each file to process (these will be processed concurrently). In each job to process, we set up a structured numpy array with a depth of four layers that mirrors the root tree. The reason for using numpy arrays instead of root trees is that they seem to be more platform independent, work well with python and outside lxplus, and do not require initial loading past opening the file.

The structure of the array is `array[eventNo][dataClass][dataType][index]`. The first index, `eventNo`, simply represents the index of the event of interest, ranging from 0 to the length of the data set. For the data class, we use indices from 0-4 to represent the data class in increasing level of abstraction (due to numpy not liking multiple layers of record arrays). This is given below:

- 0 = Recorded hits (rechits)
- 1 = Cluster data
- 2 = Regions of interest (superclusters)
- 3 = Vertex data (not always present)
- 4 = Generator-level vertex data

The vertex data (3) is representative of a reconstructed vertex using other algorithms unrelated to this analysis note and is usually not present in datasets.

The third index, `dataType` is indexed by a structured array with keywords corresponding to the respective names in the root files. The fourth index corresponds to the n^{th} element of the array.

For example, to access the energy of the 27th rechit of the 21st event (zero indexed), we would simply use: `array[21][0]['en'][27]`.

A structured array like this is built for each event, which is appended to the overall output array. This final array is saved in binary form to a `.npy` file using `np.save()`. After completing all running processes and saving the relevant files, the program terminates.

4. EDMProcessor.py

This script takes the full EDM readout of a simulated dataset and converts it to a HGCROI-formatted root file (a general TTree structure containing relevant information to the HGCAL).

This script is also simple. We initially parse the arguments and have a small module to support wildcard usage, then simply call the relevant programs from the HGCanalysis package, which process the EDM files.

This script was modified from a previously existing version.

Appendix B: Animated Figures

Several figures in this note represent systems evolving over time and feature animated content. All of the figures with animation in the analysis note can be viewed in an album at `gfycat.com/bencbartlett/tvertexing`.

Appendix C: Source Code

Any of this code written in this analysis note is available upon request from the first author or in an open-source GitHub repository (`github.com/bencbartlett/tVertexing`).

Appendix D: EOS, AFS Data and Dependencies

All data sets used in this project are available on the CMS EOS server at `/store/cmst3/group/hgcal/CMSSW/`. All files used in this project are also available at `/afs/cern.ch/user/b/bbartlet`, with the main project files being found at `/afs/cern.ch/user/b/bbartlet/public/tVertexing`.

All code in this project was run on lxplus using the `CMSSW_6_2_0_SLHC25_patch6` framework and Python 2.7.10rc1 or Python 2.6.6 with NumPy 1.4.1 and SciPy 0.7.2. Code that was run locally was run using the `CMSSW_7_4_5_FWLITE` framework and Python 2.7.10 with NumPy 1.8.0rc1 and SciPy 0.13.0b1.

ACKNOWLEDGMENTS

We would like to thank the family of Jean J. Dixon for their support of the Caltech SURF program, as this project has been in part made possible by their generous funding. We are also indebted to the Caltech CMS group for a number of illuminating conversations over the course of our group meetings. Finally, we would like to thank the Swiss Heatwave 2015, whose 40°C temperatures kept us in our office, expediting the results of this study.

* Electronic address: bartlett@caltech.edu

[1] F. Zimmermann, "CERN Upgrade Plans for the LHC and its Injectors," in *Proceedings of Science*, 2009.
[2] C. Collaboration, "Time reconstruction and performance of the CMS electromagnetic calorimeter," *Journal of Instrumentation*, vol. 5, pp. T03011–T03011, Mar. 2010.
[3] S. G. Nezami, A. Apresyan, and A. Bornheim, "Vertex Reconstruction Using Timing Information from CMS ECAL," *CMS Analysis Note*, 2012.
[4] C. Flamant, A. Bornheim, A. Apresyan, and M. Spiropulu, "Four-Dimensional Shower Profles for Picosecond Time-of-Flight Measurements and Time-Based Vertex Reconstruction with the CMS ECAL," *CMS Analysis Note*, 2014.
[5] G. Monet, "Study of pointing capabilities with photons," in *HGCAL Software Meeting*, 2015.
[6] J. S. Marshall, A. S. T. Blake, and M. A. Thomson, "Pandora : Particle Flow Reconstruction," in *AIDA Project*, 2014.