# GPU-Accelerated Expectation Maximization for Fast Point Cloud Segmentation

Benjamin Eckart, Alonzo Kelly
The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA USA
{eckart,alonzo}@cmu.edu

## Introduction

Many modern 3D range sensors, such as a Velodyne or Kinect, generate on the order of **one million data points per second** and form the foundation of many modern applications in robotic perception. Given that many of these applications demand real-time operation, care must be made for efficient algorithms that not only run adequately fast, but also make the best use of the large amounts of data available.

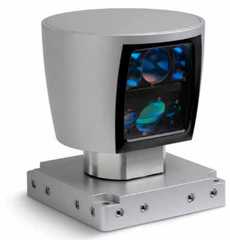


Figure: Velodyne LIDAR HDL-64E. Over 1.3 million points per second output rate *source: velodynelidar.com*

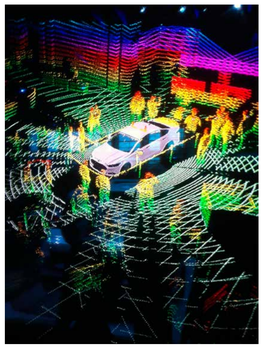


Figure: Example point cloud from 2014 Detroit Auto Show. The next generation of cars will be equipped with LIDAR sensing *source: motortrend.com*

## Motivation: A Parametric World Model

For many applications, it will be useful to transform the raw points coming from the sensor into a parametric form that conveys higher level geometric entities. A given world model not only reduces the amount of data needed for perception algorithms, but it also provides a more succinct representation that is more suitable for perceptive tasks.

Given that we have an unknown world model, $\mathbf{\Theta}$, and a set of $N$ point samples of surfaces, $\mathcal{Z}$, we would like to find the most likely $\mathbf{\Theta}$ that "explains" $\mathcal{Z}$.

We choose to parametrize $\mathbf{\Theta}$ by a convex combination of $J$ probability distribution functions (subsurfaces), which we call a mixture model. For each individual point in the point cloud, $\mathbf{z}_i$, we define its probability as

$$p(\mathbf{z}_i|\mathbf{\Theta}) = \sum_{j=1}^{J} \pi_j p(\mathbf{z}_i|\mathbf{\Theta}_j)$$

where $\pi_j$ represents our mixing weights.

The total probability of a point cloud is the product of the point probabilities, given that each point is an *iid* sample of the world.

$$p(\mathcal{Z}|\mathbf{\Theta}) = \prod_{i=1}^{N} p(\mathbf{z}_i|\mathbf{\Theta})$$

## Expectation Maximization

Directly maximizing the log data likelihood for the optimal world model is intractable. If we take the log likelihood,

$$\ln p(\mathcal{Z}|\mathbf{\Theta}) = \sum_{i=1}^{N} \ln \sum_{j=1}^{J} \pi_j p(\mathbf{z}_i|\mathbf{\Theta}_j)$$

we are still left with a sum inside the logarithm, which we cannot solve analytically for each component of $\mathbf{\Theta}$. However, if we introduce a set of binary latent variables $c_{ij}$ for each point, surface pair $\{\mathbf{z}_i, \mathbf{\Theta}_j\}$ that indicate surface ownership, we can represent the sum as a product of terms with binary exponentials and thus factorize,

$$\ln p(\mathcal{Z}, \mathcal{C}|\mathbf{\Theta}) = \sum_{ij} c_{ij} \{\ln \pi_j + \ln p(\mathbf{z}_i|\mathbf{\Theta}_j)\}$$

However, we still can't solve this factored form because we don't know $\mathcal{C}$. But, we can employ the Expectation Maximization (EM) algorithm [1] as an iterative algorithm that repeatedly performs the following two steps:

**E Step:** calculate the posterior for $\mathcal{C}$ given $\mathbf{\Theta}^{old}$:

$$E[c_{ij}] = \frac{\pi_j^{old} p(\mathbf{z}_i|\mathbf{\Theta}_j^{old})}{\sum_{j'} \pi_{j'}^{old} p(\mathbf{z}_i|\mathbf{\Theta}_{j'}^{old})}$$

**M Step:** maximize the expected log-likelihood with respect to $\mathbf{\Theta}$, using $E[c_{ij}]$:

$$\max_{\mathbf{\Theta}} \sum_{ij} E[c_{ij}]\{\ln \pi_j + \ln p(\mathbf{z}_i|\mathbf{\Theta}_j)\}$$

If we restrict our subsurfaces to be Gaussians and define $E[c_{ij}] \stackrel{\text{def}}{=} \gamma_{ij}$ we are left with:

$$\boldsymbol{\mu}_j^{new} = \frac{\sum_i \gamma_{ij}\mathbf{z}_i}{\sum_i \gamma_{ij}} \qquad \mathbf{\Sigma}_j^{new} = \frac{\sum_i \gamma_{ij}(\mathbf{z}_i - \boldsymbol{\mu}_j^{new})^T}{\sum_i \gamma_{ij}} \qquad \pi_j^{new} = \frac{\sum_i \gamma_{ij}}{N}$$

## Parallelization

Rewriting the Gaussian covariance as

$$\mathbf{\Sigma}_j^{new} = \frac{\sum_i \gamma_{ij}\mathbf{z}_i\mathbf{z}_i^T}{\sum_i \gamma_{ij}} - \frac{\boldsymbol{\mu}_j^{new}\boldsymbol{\mu}_j^{new\,T}}{\sum_i \gamma_{ij}}$$

shows us that the following quantities constitute the sufficient statistics for our segmentation:

- $\sum_i \gamma_{ij}\mathbf{z}_i\mathbf{z}_i^T$, for all $j = 1..J$ (6J numbers)
- $\sum_i \gamma_{ij}\mathbf{z}_i$, for all $j = 1..J$ (3J numbers)
- $\sum_i \gamma_{ij}$, for all $j = 1..J$ (J numbers)

The calculation of the above three quantities requires:

- Calculation of each $\gamma_{ij}$
- A weighted sum over all first and second moments

The former requires information about all $J$ surfaces but no shared information among points. The latter requires sharing data over all $N$ points but no data is needed from the $J$ surfaces once $\gamma_{ij}$ are calculated. Thus, we can use surface level parallelism in calculating $\gamma_{ij}$ and an point level parallelism to calculate the weighted moments.

## CUDA Implementation

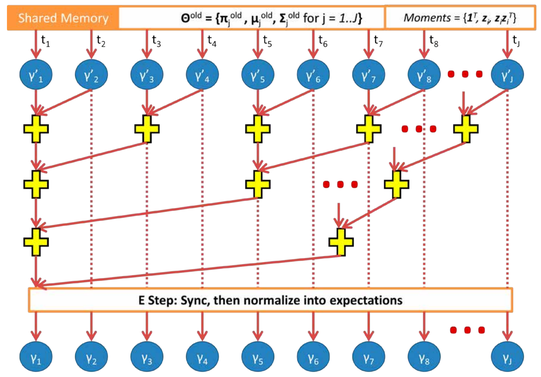



Figure: Thread blocks calculate expectations for a point with respect to all surfaces in parallel. The normalization factor is found by a sum reduction across surfaces inside the thread block. Both the point and world model are stored as shared memory.

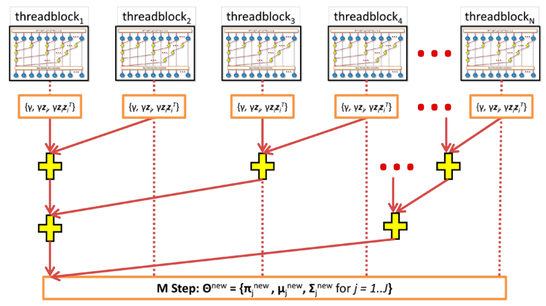
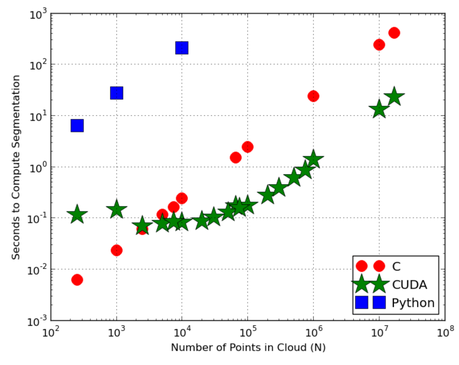



Figure: Each thread block contributes to the sufficient statistics in order to calculate a new $\mathbf{\Theta}$. With the expectations $E[c_{ij}]$ having been calculated at the thread level, each thread block contributes to the weighted moments through a sum reduction.

## Speed Comparison

We have implemented three versions of this algorithm in C, Python, and CUDA. For medium to large point clouds, CUDA's massive parallelism benefits strongly, offering over an order of magnitude improvement over C, and able to segment about **one million data points per second**.




## Segmentation Results

A good segmentation must be able to capture the salient properties of the point cloud while reducing the amount of data needed. In this way, we may think of segmentation as a kind of point cloud data compression. Given our world model, we can compress all $N$ $\mathbf{z}_i$ into weighted moments. In 3 dimensions, this is a savings of $\frac{3N}{10J}$. In general, for spatial applications of point clouds, $J \ll N$, often by three to four orders of magnitude.

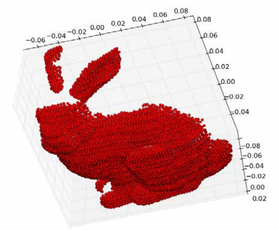


Figure: A cross-section of the Stanford bunny. This point cloud consists of about 50,000 3D points and occupies about **600 kB** of memory.

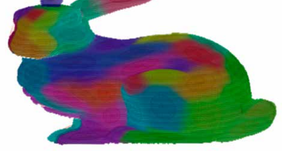


Figure: Segmentation results using 64 subsurfaces. Each subsurface is arbitrarily colored for purposes of visualization. Note that the overlap in surfaces is handled by the mixing parameter $\pi_j$. The compressed representation uses about **2.5 kB** memory.

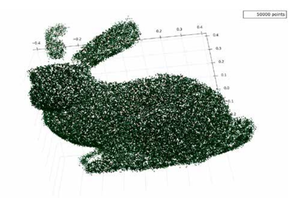


Figure: Using only our model, we can recreate the original point cloud by stochastically generating points. Note that most all fine geometric detail is preserved, indicating a successful and useful compression of data.

## Conclusions

- The EM algorithm can effectively be adapted to run on the GPU for the purpose of 3D point cloud segmentation.
- Our implementation in CUDA provides an order of magnitude increase in speed for moderate to large point clouds over a serial implementation in C.
- Our choice of world model as a mixture of Gaussians preserved fine geometric detail while lowering the amount of memory required to store the point cloud by several orders of magnitude.

## References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.