Efficient Sampling Techniques for Point Clouds

HYACINTHE HAMON
Hamon FZCO Research and Development

Abstract*—This paper introduces BOLT (Bilateral filtering and Octree Lightweight Technique), a novel, fast, and parameter-free method for up-sampling point clouds. We leverage the structural efficiency of the octree data structure and detail-preserving properties of the bilateral filter to achieve a fast and parameter-free sampling method. Unlike the current state-of-the-art techniques, BOLT does not require any parameters, deep learning, fine-tuning, or any type of training, making it a suitable candidate for real-time applications. BOLT understands the underlying structure of the point cloud by dividing the point cloud into a hierarchical octree structure. Empty children are filled in the octree, and outliers are smoothed using a bilateral filter.

I. INTRODUCTION

A point cloud is an unordered 3D representation of a set of points in space. It is commonly used in computer graphics, computer vision, and robotics. Point clouds are often generated using 3D scanners, LIDAR, and 3D applications. In this work, we wish to upsample a point cloud. Given a set of point clouds, we wish to find a new set of more dense points that still represent the same underlying surface. Further, while preserving the underlying structure, the new points should not introduce any new artifacts, should be informative, and should not cluster around the original points. The unstructured and unordered nature of point clouds makes this a challenging problem. Further, existing point cloud upsampling methods are often computationally expensive and require extensive training and parameter tuning.

To address the above challenges, we present a data-structure-driven method for point cloud upsampling that is fast and parameter-free. Our method utilizes an octree data structure without a depth limit to understand the underlying structure and initially add points to the empty children of the octree. The lack of a depth limit allows tighter fitting bounding cubes around points and allows for a more accurate representation of the underlying structure. This representation is often noisy and coarse but captures the underlying structure of the point cloud. To smooth the point cloud, we use a bilateral filter in a point cloud application

Point cloud upsampling can be used as a downstream task for various applications, such as 3D reconstruction, 3D object recognition, and 3D rendering. It can improve the quality of surface reconstruction, enhance object detection, extract features more accurately, and more.

Our method, BOLT, learns the geometry and structure of the point cloud, upsamples it, and smooths it without any parameters, deep learning, or fine-tuning.

II. RELATED WORK

Many non-deep learning-based methods for point cloud upsampling have been proposed in the past, such as moving least squares interpolation (MLS interpolation) in 2002 [2], Locally Optimal Projection (LOP) in 2007 [3], Edge-Aware Sampling (EAR) in 2013 [4], and graph total variation in 2019 [5].

MLS works by fitting a continuous surface to a set of local points using a weighted least squares fit of a polynomial surface to the points. Points are added by computing the Voronoi cells on the local surface and adding points to the vertices of the diagram.

LOP, unlike MLS, does not require fitting a local surface. Instead, it uses a projection operator to project points onto a surface in a way that minimizes the sum of the weighted distances between the original and projected points. Improvements to LOP, such as weighted LOP [6], were proposed that make LOP more robust to noise and outliers.

MLS and LOP have demonstrated promising results, but a common problem with these methods is that they do not perform well on sharp edges and corners, as the model often assumes a smooth surface.

EAR was designed to work well on edges [4]. It works by first computing the normals and relative curvature of each point. Then, if the curvature is above a certain threshold, the point is considered to be on an edge, and it is projected onto the tangent plane of the edge. If a point is considered a surface, it is projected onto the tangent plane of the surface.

Graph total variation is a method that uses a graph to represent the point cloud. First, a triangular mesh is constructed, and points are inserted at the centroids of the triangles. Assuming the point cloud is piecewise smooth, a weighted average of the L_1 norms of normals between points is minimized.

Many deep learning-based approaches also exist, such as PU-Net [7], PU-GAN [8], and PU-GCN [9]. Although these point cloud upsampling methods tackle a different problem, they are still used as a point of comparison. These methods solve different problems because they are large networks trained on large datasets and require a lot of computational power to train and run. This paper proposes a fast and parameter-free parameter-free method for point cloud upsampling.

III. BACKGROUND

A. Octree

An octree is a tree data structure with eight children in each internal node. It is often used to partition 3D space and in various applications, such as computer graphics, computer vision, and robotics.

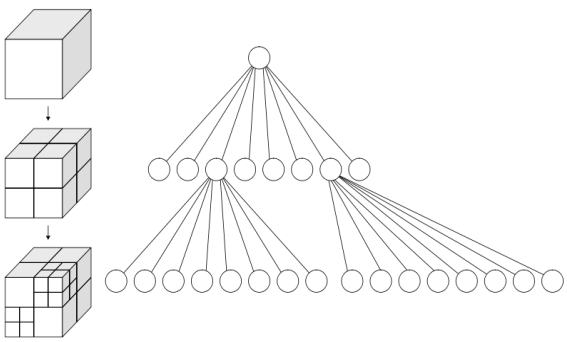

Fig. 1: An example of an octree. Each cube gets recursively divided into eight equal octants. The points are stored in the leaf nodes. Image from [10]

In point clouds, an octree is used to partition the 3D space and store the points in the leaf nodes. The octree is a hierarchical data structure that recursively divides 3D space into eight equal octants. Each node in the octree represents a rectangular prism in 3D space with a particular center, width, length, and depth. Nodes, not leaf nodes, have exactly eight children; leaf nodes store the points in the point cloud. The octree's root node represents the point cloud's bounding box. An octree has O (log n) complexity for insertion and search operations, where n is the number of points in the point cloud. The bounding cube's nearest points are much finer and tighter fitting than those further away, allowing for a more accurate representation of the underlying structure of the point cloud.

Further, these tighter bounding boxes hint at where to add new points to the point cloud. Adding points to the tighter bounding boxes will result in more informative points that are not clustered around the original points, will not introduce any new artifacts, and will preserve the underlying structure of the point cloud.

Most octrees have a depth limit, which means that the octree. will not divide the space beyond a certain depth to avoid a problem of infinite recursion. However, our method has no depth limit, and we allow the octree to divide the space as much as possible. Since the starting point clouds. They are often sparse and noisy, and the lack of a depth limit allows for a more accurate representation of the underlying structure of the point cloud. Other stops, such as checking if an existing close points are already in the node and are used to avoid infinite recursion.

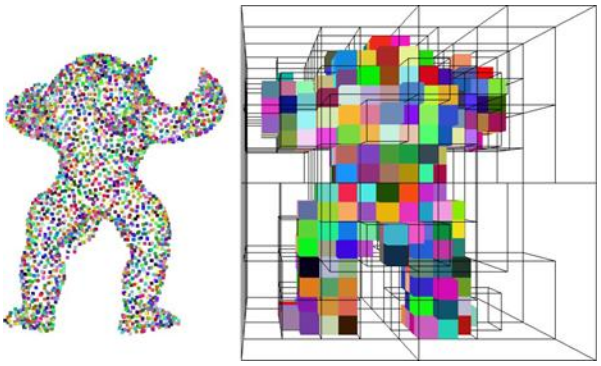

Fig. 2: An example of an octree with its point cloud and cubed representation. The figure was from a blog post [11] and generated using the Open3D library [12] with a max depth of 4.

B. Bilateral Filter

The non-linear bilateral filter smoothes images and reduces noise while preserving edges. It is a generalization of the Gaussian filter and is used in various applications such as image processing, computer graphics, and computer vision. It is defined as follows:

$$B(I, x) = \sum_{x_i \in \Omega} I(x_i) f_r(\|x_i - x\|) g_s(\|x_i - x\|)$$

Our paper uses the bilateral filter to smooth the point cloud. Similar to the image case, the bilateral filter smooths the point cloud while preserving the edges and the underlying structure of the point cloud. It works by shifting points along a normal vector, and the shift amount is a weighted average distance to its neighbors. We follow Digne et al. [1] and use the following definition of the bilateral filter for point clouds:

$$p' = p + \delta_p \cdot n_p \qquad (1)$$

Where n_p is the normal to the regression plane of some k nearest neighbors of p. Following Dinge et al. [1], our implementation computes the normal with PCA. PCA will find the regression plane that fits the data best, and the corresponding found eigenvectors will be orthogonal to said plane, which is simple to compute compared to an iterative least squares method. δ_p is the displacement of the point p. The displacement is computed as follows: let N_p be the set of k nearest neighbors of p:

$$\delta_p = \frac{\sum_{q \in N_p} w_d(\|q - p\|) w_n(\langle n_p, q - p \rangle) \langle n_p, q - p \rangle}{\sum_{q \in N_p} w_d(\|q - p\|) w_n(\langle n, q - p \rangle)} \qquad (2)$$

In our implementation, w*d* and w*e* are the Gaussian functions defined as follows:

$$w_i(x) = \exp\left(-\frac{x^2}{2\sigma_i^2}\right) \qquad (3)$$

In our implementation, we set $\sigma d = 0.1$ and $\sigma n = 0.1$.

IV. METHODOLOGY

We aim to upscale and smooth a sparse point cloud using an octree with a significant depth and bilateral filing on a point cloud. We start with a sparse point cloud P = {p1, . . . , p*n*}, and generate an octree T by iterating through and inserting one at a time. To generate the initial upsampling of the points, we find the parent of each for point p*i* in T, then add a new point to an empty child of the parent in T. One such iteration will double the number of points in the point cloud; then another will quadruple, and so on. This process gets repeated the number of times necessary to get the desired final number of points. Then, we extract all points from T to get our new upsampled point cloud P′. We then smooth P′ with bilateral filtering. Bilateral filtering requires hyperparameters σ*d*, σ*n*, and k. k indicates the number of neighbors used to find the normal of the regression plane, and σ*d* and σ*n* are the standard deviations for the Gaussians used in (3).

Algorithm 1 Main upsampling algorithm

Require: sparse point cloud P with n points, n_{up} number of iterations required to get the desired number of points functions UPSAMPLE(P)

T ← CONSTRUCTOCTREE(P) P_n ← EMPTYPOINTCLOUD

i ≤ nup

for p ∈ P do

parent ← p.parent child ← RANDOMEMPTYCHILD(parent)

p′ ← RANDOMPOINTINSIDE(child. Dimensions)

end for

P ← CONVERTTOPOINTCLOUD(T)

end function

V. EXPERIMENTS

Experiments were done mainly with the ShapeNet

dataset [13], a large dataset of 3D models. A random set of 1024-point clouds was sampled and then upscaled to double or 2048 points. We then compared some evaluation metrics with the ground truth to see how well our method performs quantitatively. In the appendix, we highlight the qualitative results of our method compared to other methods used in this paper.

A. *Evaluation*

We will eventually evaluate our model using the Chamfer and Hausdorff distances, as they are standard metrics used in point cloud upsampling. We will compare it to other parameter-free works and deep learning-based methods.

Algorithm 2 Bilateral smoothing algorithm, borrowed heavily from [1]

Require: point cloud P with n points, P_n new points, k neighbours, σ_d, σ_n

function BILATERALSMOOTH (P, P_n)

for $p \in P_n$ do

$N_p \leftarrow$ FINDNEIGHBOURS(P, k, p)

$n_p \leftarrow$ COMPUTEUNITNORMALTOPLANE(N)

$SW \leftarrow 0$ $\delta_p \leftarrow 0$

$q \in N_p$

$w \leftarrow w_d(\|p - q\|) \cdot w_n(\langle n_p, p - q\rangle)$ ▷ From (3)

$SW \leftarrow s_w + w$

$\delta_p \leftarrow \delta_p + w \cdot \langle n_p, p - q\rangle$

end for

$p' \leftarrow p + \delta_p \overline{s_w \cdot n_p}$

end function

The Chamfer distance is a measure of how different two shapes are and is defined as the following:

$$C(P, Q) = \frac{1}{|P|} \sum_{p \in P} \min_{q \in Q} \|p - q\|^2 + \frac{1}{|Q|} \sum_{q \in Q} \min_{p \in P} \|p - q\|^2$$

The Hausdorff distance is a measure of how similar two sets are. It is defined as the following:

H(A, B) = max(h(A, B), h(B, A))

Where:

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

In terms of comparisons, we perform comparisons with MLS as a baseline for another non-deep learning-based BILATEmReAthLoSdM, OasOTwHe(lPl,aPs*n*)with PU-GCN as a deep learning-based method. We also see how our method performs compared to

B. *Comparison with Non-Deep Methods*

Other smoothing methods, such as KNN and no smoothing, and different sampling methods, such as random sampling and octree sampling, are also used to show that our choices in smoothing and sampling are the best in this context.

We compared our results with MLS, a non-deep learning-based method of upsampling point clouds using local surface fitting.

Chamfer distance $\times 10^3$

class	MLS	Ours
plane	14.8	15.8
helmets	26.7	29
cap	22.4	24.4
car	28.6	31
headset	25.3	23.8

TABLE I: Comparison of our method with MLS with chamfer distance ×103. Note that lower is better

Hausdorff distance $\times 10^3$

class	MLS	Ours
plane	282.9	80.5
helmets	454.8	182.2
cap	150.2	156.3
car	75.3	73.5
headset	171.3	166.5

TABLE II: Comparison of our method with MLS with Haus- Dorff distance ×103. Note that lower is better.

Our method generally performed better than MLS regarding the Hausdorff distance but worse in the Chamfer distance. In some cases, our method performed better in both metrics, such as the headset class. In terms of execution time, our method was also slower than MLS, taking 0.5 seconds to run compared to 0.2 seconds for MLS. Note, however, that the MLS implementation was written entirely in C++. Our method only implements the bilateral filter in C++; the rest are implemented in Python. This difference in overhead may account for the difference in execution time.

A worse Chamfer distance but better Hausdorff distance indicates that our method is better at preserving the global shape but worse at preserving the

local shape. It also implies that our method is better at preserving the overall structure of the point cloud but worse at preserving the details. It also implies that our method is less sensitive to outliers than MLS. This sensitivity to outliers for MLS can be reflected in the car example in the appendix, where the MLS method has a few points very far from the shape, whereas our method does not have this issue.

Depending on the task, one may use MLS or our method. Our method is better if the task requires preserving the overall structure of the point cloud, while MLS is better if the task requires preserving the details of the point cloud.

C. Comparison of Other Smoothing Methods

We compared our method with other smoothing methods, such as the bilateral filter and a K-nearest neighbors-based method, as well as no smoothing and just the octree sampling.

Chamfer distance $\times 10^3$

class	KNN	Bilateral	None
plane	16.2	15.8	16
helmets	29.3	29	29.8
cap	24.5	24.4	25.5
car	31.3	31	31
headset	24.4	23.8	24.2

TABLE III: Comparison of our method with MLS with cham- fer distance ×103. Note that lower is better In general, our method performed better than the KNN method in terms of both the Chamfer and Hausdorff distance, except in the cap and car classes. Bilateral also performed better than no smoothing regarding the Chamfer distance but worse in the Hausdorff distance. Hausdorff distance x10^3

class	MLS	Ours	None
plane	81.3	80.5	78.4
helmets	186.2	182.2	182.1
cap	149.4	156.3	155.6
car	72.1	73.5	74
headset	168.3	166.5	166.5

TABLE IV: Comparison of our method with MLS with Hausdorff distance $\times 10^3$. Note that lower is better.

D. Comparison of Different Sampling Methods

In this subsection, we compare our octree sampling method with random sampling, and both cases use bilateral smoothing.

Chamfer distance $\times 10^3$

class	Random	Octree
plane	49.1	15.8
helmets	40	29
cap	38.7	24.4
car	36.7	31
headset	46	23.8

TABLE V: Comparison of our method with random sampling with chamfer distance $\times 10^3$. Note that lower is better.

Hausdorff distance $\times 10^3$

Class	Random	Octree
plane	86.2	80.5
helmets	172.4	182.2
cap	153.4	156.3
car	74.6	73.5
headset	178.3	166.5

TABLE VI: Comparison of our method with random sampling with Hausdorff distance $\times 10^3$. Note that lower is better

The octree sampling method outperforms the random sampling method in terms of Chamfer distance but is slightly better regarding the Hausdorff distance. This shows that using an octree to voxelize and add points nearby points is a better method than randomly sampling points.

E. Comparison with Deep Methods

We will compare our method with the deep learning-based method PU-GCN[9]. First, the computational cost of using PU-GCN will be analyzed. When experimenting with an Nvidia 4090, we found that the PU-GCN took 10GB of memory during training. We trained the PU-GCN model for 10 epochs, which took 2 hours. The original paper trained the model for 100 epochs. This is a significant amount of time and memory and is a disadvantage of the PU-GCN method. In the PU-GCN paper, the authors claimed a chamfer distance of $\sim 0.5\times 10^3$ and a Hausdorff distance of $\sim 1\times 10^3$. This is much lower than our method, but the computational cost is much higher.

This is also much lower than what we found in our experiments but is likely because we only trained for 10 epochs. This comparison was done with the PU1K dataset. Each point cloud was a sample of 256 points and was upsampled to 1024 points.

Hausdorff $\times 10^3$

Class	PU-GCN	Ours
eight	96.1	221.3
elephant	74.1	122.6
elk	94.6	52
Sandisk	86	169.3
genus3	127.1	378.6

TABLE VII: Comparison of our method with PU-GCN with chamfer distance $\times 10^3$. Note that lower is better

Our method performs much worse in terms of the Hausdorff distance. However, its computational cost is much lower than that of PU-GCN.

VI. FUTURE WORK

Since this method does not require any parameters and is very light, it is suitable for real-time applications. One issue, however, is that it is slower than methods such as MLS. This is due to the overhead in Python and the lack of concurrency. Thus, one potential future work is to implement this method in C++ to reduce the overhead since, in this work, the bilateral filter was already implemented in C++. Further, this method can benefit significantly from concurrency. Numerous works parallelize the creation of octree structures [14]. One can also parallelize the bilateral filter using an octree, as shown in [1]. The method can be made faster and more suitable for real-time applications with both optimizations.

CONCLUSION

In conclusion, this paper presents BOLT, a fast and parameter-free method for upsampling point clouds. Our method leverages the octree data structure's structural efficiency and the bilateral filter's detail-preserving properties to achieve a fast and parameter-free upsampling method. Unlike the current state-of-the-art deep methods, BOLT does not require any parameters, deep learning, fine-tuning, GPU, or any type of training, making it a suitable candidate for real-time applications. We evaluate BOLT on various point clouds, compare it with the current state-of-the-art methods, and show competitive results. In future work, we plan to implement BOLT in C++ and add concurrency to reduce the overhead and improve the execution time.

REFERENCES

[1] J. Digne and C. de Franchis, "The Bilateral Filter for Point Clouds," *Image Processing On Line*, vol. 7, pp. 278–287, 2017, https://doi.org/10.5201/ipol.2017.179.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Trans. Vis. Comput. Graph.*, vol. 9, pp. 3–15, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:792977

[3] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer, "Parameterization- free projection for geometry reconstruction," *ACM Trans. Graph.*, vol. 26, no. 3, p. 22–es, Jul 2007. [Online]. Available: https://doi.org/10.1145/1276377.1276405

[4] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, "Edge-aware point set resampling," *ACM Trans. Graph.*, vol. 32, no. 1, Feb 2013. [Online]. Available: https://doi.org/10.1145/2421636.2421645

[5] C. Dinesh, G. Cheung, and I. V. Bajic, "3d point cloud super-resolution via graph total variation on surface normals," 2019.

[6] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM Trans. Graph.*, vol. 28, no. 5, p. 1–7, Dec 2009. [Online]. Available: https://doi.org/10.1145/1618452.1618522

[7] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-net: Point cloud upsampling network," 2018.

[8] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-gan: a point cloud upsampling adversarial network," 2019.

[9] G. Qian, A. Abualshour, G. Li, A. Thabet, and B. Ghanem, "Pu-gcn: Point cloud upsampling using

graph convolutional networks," 2019.

[10] "Octree," https://en.wikipedia.org/wiki/Octree.

[11] "Open3d octree ," https://blog.csdn.net/qq_41068877/article/details/ 124242265.

[12] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[13] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li,

[14] S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," 2015.

[15] C. V., K. K., A. P., and A. J. K., "Parallel manipulations of octrees and quadtrees," 1992.