Received 8 June 2022, accepted 15 June 2022, date of publication 24 June 2022, date of current version 30 June 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3186094

QBB: Quantile-Based Binarization of 3D Point Cloud Descriptors

DÁNIEL VARGA, JÁNOS MÁRK SZALAI-GINDL, MÁRTON AMBRUS-DOBAI, AND SÁNDOR LAKI, (Member, IEEE)
ELTE Eötvös Loránd University, 1117 Budapest, Hungary

Corresponding author: Dániel Varga (vargadaniel@inf.elte.hu)

The work of Dániel Varga was supported by the ÚNKP-21-4 New National Excellence Program of the Ministry for Innovation and Technology from the Source of the National Research, Development and Innovation Fund. The work of János Márk Szalai-Gindl and Sándor Laki was supported by the National Research, Development and Innovation Fund of Hungary Financed under the Thematic Excellence Program (National Challenges Subprogram) Funding Scheme through the Project Application Domain Specific Highly Reliable IT Solutions under Grant TKP2020-NKA-06.

ABSTRACT Local 3D point feature descriptors play an important role in many areas of computer vision, such as object recognition, registration, etc. There are many well-functioning feature descriptors, but they are typically real-valued and multidimensional vectors, leading to high computational complexity in nearest neighbor searches. To overcome this challenge, methods binarizing real-valued descriptors have emerged. In this paper, we first investigate the available binarization methods and standalone binary feature descriptors and show that existing binarization techniques cannot generally achieve good performance for arbitrary feature descriptors. To remedy this problem, we propose a new binarization method called quantile-based binarization (QBB) that can be applied to any real-valued feature descriptors. It analyses the distribution of feature descriptors that is then used to form meaningful groups along each dimension. To this end, QBB computes quantiles of the empirical distribution and the interval lengths (bin sizes) defined by quantile boundaries. Finally, it assigns a binary code to each group and concatenate them to get the final binary descriptor. QBB is able to adaptively compute the number of bits based on a capacity constraint, i.e., with the appropriate capacity setting, the resulting binary descriptor can be used on devices with lower computational power. We evaluate the descriptiveness of well-known descriptors binarized by QBB and compare them to state-of-the-art methods. According to our evaluation, QBB is able to create binary descriptors whose descriptiveness is closer to the real-valued descriptors than prior approaches. Finally, we also show that QBB can even compete with standalone binary feature descriptors.

INDEX TERMS 3D point cloud, feature descriptor, binarization.

I. INTRODUCTION

Many novel applications using 3D point clouds have recently emerged. For example, a robot manipulating the environment needs the ability to accurately sense the real world around it. Though stereo cameras and other approaches enable to obtain depth information from 2D images, similarly to human beings, point clouds collected by dedicated 3D depth sensors are more accurate. With the development of self-driving vehicles, the processing of 3D point clouds has become an even more important task. Detecting other vehicles or pedestrians around the vehicle is essential for safe and reliable control. Such methods (especially point cloud registration and object recognition algorithms) mostly rely on local feature descriptors. A local feature descriptor is a vector that characterizes the environment of a point in the point cloud and thereby makes points distinguishable from each other. The best known feature descriptors include FPFH [1], SHOT [2], SI [3], RoPS [4] and USC [5]. Survey papers like [6] have already collected and categorized these descriptors. Guo *et al.* [7] systematically compare local feature descriptors using different datasets. Their comparison covers a number of different aspects (such as computational effiency, time-crucial, and space-crucial applications, etc.) that count in real-world use cases. Their work concludes that in most application areas FPFH or SHOT

The associate editor coordinating the review of this manuscript and approving it for publication was Wenbing Zhao.

shows the best performance, depending on the number of points.

The most common operation with feature descriptors is the nearest neighbor search. The classic local descriptors are multi-dimensional real-valued features (e.g., SHOT has 352 dimensions), resulting in storage and computational costs that are too high for real-time applications. Augmented reality apps relying on 3D point clouds are very popular on mobile devices with high-quality imaging sensors. For these apps, real-time processing is utmost important for good quality of experience. Some works [8], [9] propose dimension reduction of feature descriptors to accelerate the expensive point cloud matching step. Dimension reduction techniques can compress feature descriptors so that their descriptiveness is not significantly limited.

Another way to solve this problem is to create binary descriptors (i.e., bit sequences) from existing feature descriptors. Algorithms that convert a real-valued feature vector into a binary feature vector are called binarization methods. A binary vector requires much less space than real valued ones. Furthermore, Hamming distance between bit sequences can be performed with bit operations, thus nearest neighbor searches can be executed significantly faster [10].

A typical approach for binarization is to replace the real numerical values of the original descriptor with one or more bits along a dimension. There are general binarization methods that can be applied for any real-valued descriptors. More advanced methods exploit the special properties of how a feature descriptor is computed. Finally, there are standalone binary feature descriptors, which create binary features directly.

In this work, we present a novel 3D point feature descriptor binarization method that creates groups by quantizing the distributions of each element in the original descriptor. These groups are then represented by bit sequences whose concatenation results in the binary descriptor. Using this method, the nearest neighbor searches can be performed much faster, and the descriptors require less storage. The main contributions of this paper are the following:

- We propose a new parameter-free binarization method called QBB that can generally be used to binarize any classic feaure descriptors.
- We compare the performance and properties of QBB to the available binarization methods and standalone binary feature descriptors and show that QBB overperforms them on real-world point cloud data.
- We present possible variations of QBB which in some cases have a higher descriptiveness than the original descriptor.
- The source codes of QBB and other binarization methods used in the evaluation of this paper is publicly be available on GitHub.[1]

[1]https://github.com/ELTE-IK-Point-Cloud-Group/QBB

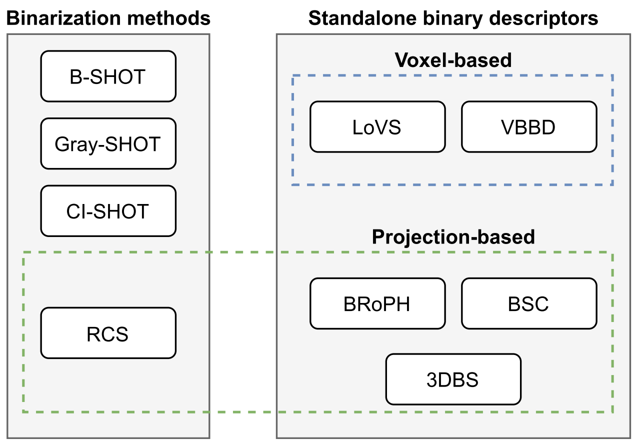


FIGURE 1. Grouping of binarization methods and standalone binary descriptors. RCS is a standalone, real-valued descriptor, but the authors gave 3 methods to binarize it. Two of these methods are general enough to be applied to any arbitrary real-valued descriptor.

II. RELATED WORK

The first method for binarizing real-valued 3D point feature descriptors was only published in 2015; it was called B-SHOT [10]. Since then, a number of works have been published that also attempted to solve the task of binarization. Binary feature descriptors can be divided into two groups (Fig. 1). One group includes methods that convert an existing real-valued descriptor into a binary one (such as B-SHOT). The advantage of these methods is that they can be applied to any real-valued feature descriptor. In general, binary feature descriptors created after binarization are less descriptive than the original descriptors, but their space requirements are much less, and nearest neighbor searches are faster on them. One drawback of this approach is that the real-valued feature descriptor must first be calculated and then converted to binary, which takes extra time.

The binarization method by Prakhya *et al.* [10] aims at binarizing SHOT, a real-valued feature descriptor. The algorithm can binarize any real-valued vector, but it was inspired by the distribution of SHOT descriptors, and works best in combination with this descriptor method. The two parameters of the algorithm are encoding length (L) and encoding ratio E_r (percentage). Each real value corresponds to one bit, i.e. the length of the descriptor does not change. The first step is to select L consecutive value from the original, real-valued descriptor and compute their sum. The corresponding bits of the values that contributed to $E_r\%$ of the sum are assigned a bit with value 1, the other bits will be 0. By concatenating the bit sequences of length L, we obtain the final descriptor. According to the authors, the B-SHOT gives the best results with $L = 4$ and $E_r = 90\%$. Later, they compared B-SHOT with B-FPFH and B-ROPS using the same algorithm. Their results show that while B-SHOT's descriptiveness is close to the original descriptor, the other two binarized descriptors are far behind. The authors acknowledge that in some cases the loss of information could be high.

Later, similar work was published, also aimed at binarizing the SHOT real-valued feature descriptor [11]. Lin *et al.* proposed a generic version of the B-SHOT algorithm (Gray-SHOT [12]) in which the encoding length (L) and the number of bits used to encode each element became input parameters. By increasing the number of bits used to represent an element, it was possible to binarize the real-valued descriptor with less loss of information, but the length of the bit sequence increased. When multiple bits are used to represent an element, Gray code was applied so that the Hamming distance of the consecutive groups is always 1. Their parameter tuning shows that the best results can be obtained when the encoding length is 352, and the number of bits used to represent each element is 2. The CI-SHOT [11] method is very similar to Gray-SHOT. This method also uses encoding length and can use more bits to represent an element. The main difference compared to the previously mentioned methods is that the CI-SHOT algorithm decides the value of bits based on a method inspired by Chebyshev's inequality. Based on their parameter tuning, the best choice for encoding length is 11, and the number of bits representing one element is 2. The authors only applied on the SHOT real-valued descriptor their binarization methods.

Yang *et al.* proposed a real-valued feature descriptor called Rotational Contour Signatures (RCS) [13]. In their work, they give three methods to convert their descriptor into a binary one. All of these methods are general enough to be applied to an arbitrary real-valued descriptor. The first method is called thresholding, which works by calculating a threshold t. For each element from the original descriptor, a bit is assigned depending on whether the current value is greater or less than the threshold. The authors calculated the median for an element and considered this value as the threshold. Their second approach is vector quantization, where one element is represented by several bits, resulting in 2^N number of groups, where N is the number of bits for one element. The advantage of this method is that it can store more information, but it uses more bits and is more sensitive to noise. The third method is called geometric binary encoding. It works with a series of real values and compares the adjacent values. It assigns bits for each real value depending on the result of the comparison. If a value is greater than the previous value, the corresponding bit is 1, otherwise 0. According to their results, the best binarization method was vector quantization, especially with 2 or 4 bits. In some cases, the binarized version performed almost as well as the real-valued descriptor itself. Our solution is based on vector quantization as well.

Another group of binary descriptors is the standalone binary descriptors. These are descriptors that produce a binary descriptor by default. The advantage of this approach is that no separate binarization step is required. The disadvantage compared to the other approach is that if a new, better descriptor is available, the binarization methods can easily produce a binary version of it too. The currently available standalone feature descriptors can be divided into two groups. The first group includes methods that build a voxel grid at the local neighborhood of the point and assign a bit to each voxel. The other group includes methods that project neighbor points onto planes or axes and process the resulting lower-dimensional data to calculate descriptors.

Two well-known voxel-based standalone descriptors are LoVS (Local Voxelized Structure) [14] and VBBD (Voxel-Based Buffer-Weighted Binary Descriptor) [15]. These two methods are very similar. In the first step, both methods compute a local reference frame (LRF) around the selected point and the points in the local neighborhood are transformed with respect to the LRF (although they use different LRF computations). In the next step, they build a voxel grid, and each voxel will correspond to one bit. In the case of LoVS, a bit value will be 1 if one or more point falls in the corresponding voxel, otherwise 0. The parameters of the method are the radius by which the neighborhood is determined and the number of voxels along an axis (m). The length of the descriptor is m^3. In the case of VBBD, the number of bits is also determined by the number of voxels, but the calculation of the value of the bits is more complex. The points closer than a radius h to a voxel center are part of the buffer region of that voxel. The Gaussian kernel density is calculated for each voxel's buffer region. The value of a bit corresponding to a voxel will be 1 if the Gaussian kernel density of the voxel's buffer region is greater than the average Gaussian kernel density. By increasing the number of voxels, more information can be stored, but if the number is too high, they become sensitive to noise. For both methods, the authors recommend setting $m = 9$ number of voxels along each ax, so the length of the descriptors will be 729 bits.

Projection-based methods have in common that they project points in their environment onto planes or axes. For this reason they need to compute an LRF and transform their neighboorhood respect to it. The 3D Binary Signatures (3DBS [16]) takes the local environment, not by a radius, but the N nearest neighbors with angular constraints. This is important because in the case of 3DBS the number of neighbors determines the length of the descriptor. First, the algorithm projects the normal vector of each point onto the x, y, and z axes. In the second step, it creates ordered point pairs and compares the projected values of their normal vectors. Each ordered point pair correspond to 3 bits (compared projected values for each ax). After concatenating the bits, the length of the final descriptor is $3 \cdot \binom{N}{2}$. The Binary Shape Context (BSC [17]) and the Binary Rotational Projection Histogram (BRoPH [18]) project the points in the local neighborhood onto xy, xz, and yz planes. The idea is to reduce the binarization of 3D points back to 2D binarization. BRoPH is similar to the RoPS [4] real-valued descriptor. The algorithm rotates the local point cloud around the x, y, and z axes with an angle (θ) and after each rotation projects the points onto three planes. These 2D image patches are then divided into $L \times L$ bins. For each bin, the points in the bin are counted (distribution matrix D) and the average of their depth values (depth image I) is calculated. The bit sequence is calculated based on these 2D image patches. The length of the final descriptor is $3 \times 3 \times d_{broph_r} \times d_{broph_{bl}}$ ($broph_r$ is the rotation size, $broph_{bl}$ is the bin length for every 2D image patches). BSC differs from BRoPH in that it projects the points onto the three planes without rotation, but also produces 2D image patches based on distribution and depth.

III. METHOD

A. MOTIVATION

Guo *et al.* highlight in their work [7] that most feature descriptors use histograms. Though there are significant differences between them, the feature elements are derived from the weights of histogram bins.

The FPFH feature descriptor is based on three angular features: θ, α and ϕ [1] that are related to the normal vectors and the difference between the vectors of two points. For each point in the point cloud, FPPH takes all the neighboring points, calculate the three angular features for that point and its neighbors, and finally, for all the features calculated for all the points, the three histograms are formed (with 11 bins which is the default parameter in the PCL [19]). The elements of the feature descriptor of a given point are computed based on how many of the feature values for the point-neighboring point pairs fall within the intervals of the bins. (If none of the feature values for the point-neighboring point pairs fall into a given bin, the element associated with that bin is 0, if all of them fall into a given bin, the element is 200, based on the Open3D FPFH implementation [20].) Since ϕ is the most interesting (cf. Fig. 3), we only show its typical histogram and empirical density functions associated with two bins in Fig. 2. If a bin has a small weight, there will be many 0 values along the corresponding dimension of the feature descriptors. This is shown in Fig. 2/(b). For high bin weights, the values along the relevant dimension may give a more interesting density function than in the previous case, which is illustrated in Fig. 2/(c). Based on our experience, it can be said in general about the different feature descriptors that, for different reasons, it is typical that along many dimensions there will be many 0 values and only a few will have a really interesting density function.

In our binarization method, we will aim to group the values of feature descriptors along a dimension. We will not distinguish between values that form a group in the following. The grouping should therefore be done in a way that preserves as much as possible the descriptive power of the original feature descriptors. When grouping along a dimension, one of the goals will be to have roughly the same amount of adjacent values in the groups, since we want each group to be equally important. Therefore, the boundaries of the groups are based on quantiles. Assuming that for a fixed feature descriptor method per point cloud type, the values along a dimension come from a similar density function, a sufficiently large training set can be used to determine the boundaries of the groups in advance. The question is how many groups should be formed. This will be explained in more detail below. For now, we just note that

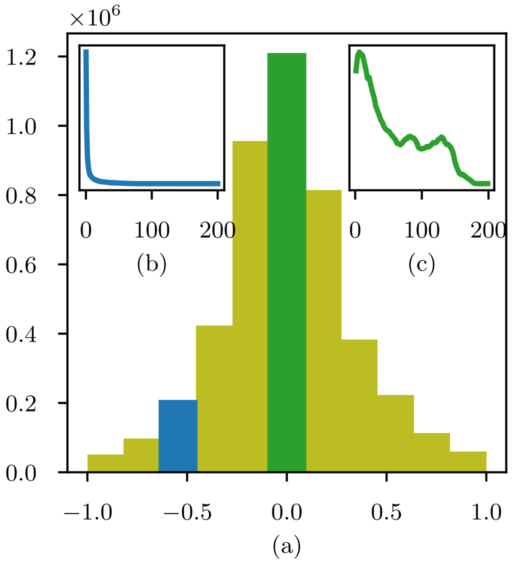


FIGURE 2. (a) Histogram of the feature ϕ used to calculate the FPFH descriptor. (b) Empirical density function of the 25th element of the FPFH descriptor (corresponding histogram bin denoted by blue) (c) Empirical density function of the 28th element of the FPFH descriptor (corresponding histogram bin denoted by green).

- For those dimensions of the FPFH along which there are many values of 0, even the median value (or even the third quartile) may be 0, which means that we will not be able to define more than two groups. (Zero values form the first group and the the rest form the second group.) In this case, it will not even be satisfied that the same amount of values is added to the groups.
- In fact, if we take more and more groups, even for the more interesting dimensions, the boundaries of a group may be too close to each other, so there will be a stopping condition.

The groups are then represented by bit sequences that takes some account the proximity between the groups. (Essentially, the values are quantized along a dimension and the quantized values are binarized, but the specific quantized values are irrelevant to our method.) For the full feature descriptor, binarization is achieved by concatenating the bit sequences obtained along the dimensions. Our method will be called quantile-based binarization (QBB for short).

B. QUANTILE-BASED BINARIZATION (QBB)

This subsection provides a detailed exposition of our method. First of all, let us introduce the notations. D dimensional feature descriptor space is assumed. The n feature descriptors, the dth element of ith feature descriptor and the list of the dth elements of feature descriptors are denoted by $X_1, \ldots, X_n$, $X_i^{(d)}$ and $\mathbf{X}^{(d)} = (X_1^{(d)}, \ldots, X_n^{(d)})$, respectively ($i = 1, \ldots, n$, $d = 1, \ldots, D$). Write

$$\mathbf{X}_{\min}^{(d)} = \min\{X_1^{(d)}, \ldots, X_n^{(d)}\} \quad \text{and}$$
$$\mathbf{X}_{\max}^{(d)} = \max\{X_1^{(d)}, \ldots, X_n^{(d)}\}.$$

Let $Q^{(d)}(p)$ be the empirical quantile function of $\mathbf{X}^{(d)}$ for $0 \leq p \leq 1$. The tuple-builder notation:[2]

$$(f(k) \mid k = 0, \ldots, K)$$

is modelled on the set-builder notation and this generates a list with first element f(0), second element f(1), etc. Let round(x) be the nearest integer function which is the integer closest to $x \in \mathbb{R}$.

For the sake of simplicity, power-of-two groups will be formed. It is not necessarily true that the more groups we define along dimensions, the better the feature descriptor will retain its descriptive power, because we do not want to put very similar values into separate groups (for example, if the density function has a relatively narrow peak somewhere, we do not want to cut it in half). The choice of the number of groups is somewhat related to the choice of the number of bins when we want to approximate the distribution with a histogram. It is true that there should be enough bins to ensure that a bin does not hide any relevant information about the shape of the distribution, but ignores details due to random fluctuations [22].

If we want to avoid grouping the values mentioned above into two separate groups, care must be taken to ensure that the boundary of a group cannot be placed anywhere, but it should be expressed as a multiple of a unit. The unit is given by the uniform bin width of the histogram of the values, which can be determined by several methods. Because of its popularity, we have used the Freedman-Diaconis rule [23] with the addition of a limit below which the bin size cannot go in order to speed up the calculation. Let *bw* denote the bin width, then:

$$bw = \max\left(2 \cdot \frac{IQR\left(\mathbf{X}^{(d)}\right)}{\sqrt[3]{n}}, \frac{\mathbf{X}^{(d)}_{\max} - \mathbf{X}^{(d)}_{\min}}{10^4}\right) \quad (1)$$

where $IQR\left(\mathbf{X}^{(d)}\right)$ is the interquartile range, i.e. $Q^{(d)}(0.75) - Q^{(d)}(0.25)$.

In accordance with the above, on the one hand, Algorithm 1 determines, for a given dimension d, the maximum group number for which details due to random fluctuations are (expected to be) ignored. On the other hand, it computes the boundaries for each group up to the maximum group number. For a given group number *gnum*, this means that the values along dimension d of the feature descriptors are sorted in ascending order and divided into *gnum* equal parts, i.e. the *gnum*-quantiles are determined. With the addition that we can only put the boundaries where the unit boundaries are. In fact, the algorithm will only run with the specified loop as long as there is a group whose interval length is 0. The algorithm will return a set $\mathcal{G}_d$ with one element representing a group number and its associated boundaries. The reason why the algorithm does not only return boundaries for the maximum number of groups is that we also want to consider a capacity limit. We will come back to this.

[2]Similar notation can be found e.g. in Schrodt's thesis [21]

Algorithm 1 Determination of Groups for Dimension d

```
INPUT: $\mathbf{X}^{(d)}$, $bw$      ▷ the definition of $bw$ is given in Eq. 1
OUTPUT: the set $\mathcal{G}_d$ of groups and their associated boundaries up to the maximum number of groups that can be requested
 1: $\mathcal{G}_d \leftarrow \{\}$
 2: $gnum \leftarrow 2$
 3: repeat
 4:     $endps \leftarrow (\mathrm{round}(Q^{(d)}(k/gnum)/bw) \cdot bw \mid k = 0, \ldots, gnum)$
 5:     $mdiff \leftarrow \min(\{endps[k+1] - endps[k] \mid k \in [0, gnum-1]\})$
 6:     if $mdiff > 0 \vee gnum = 2$ then
 7:         $\mathcal{G}_d \leftarrow \mathcal{G}_d \cup \{(gnum, endps)\}$
 8:     end if
 9:     $gnum \leftarrow 2 \cdot gnum$
10: until $mdiff > 0$
11: return $\mathcal{G}_d$
```

In the case of the FPFH method, the maximum group numbers along the dimensions were determined separately for several point clouds (derived from the data set described in Sec. IV), which are shown in Fig. 3. It can be seen that for each dimension Algorithm 1 gives roughly similar group numbers for different point clouds. For this reason, a reasonable number of groups along the dimensions can be defined in advance, which will preserve a roughly similar descriptive power of the FPFH for future point clouds.

There are several ways to represent group indices with bit sequences. It is important to note that the similarity between binarized descriptors will be calculated by Hamming distance (and a modified version of it, see Sec. V-B). Our goal is to make the distance between binarized descriptors reflect the distance between real-valued descriptors as closely as possible while keeping the number of bits low. We need $\log_2 gnum$ bits to encode *gnum* group at least. On the other hand, to minimize information loss, if there are k groups between two groups, we want their Hamming distance to be $k + 1$. A weaker condition is that we expect the Hamming distance between bit sequences representing adjacent groups to be exactly 1. A suitable method to satisfy the weaker condition is Gray code [24], which uses $\log_2(gnum)$ number of bits. However, if we want to keep the distance between the groups accurate, it is easy to see, that we need $gnum - 1$ bits at least. The Mersenne numbers expressed in the binary numeral system are suitable for this [25]. We will be referred to it as Mersenne code. Fig. 4 and Table 1 illustrate the Gray and Mersenne code representations for 8 groups.

C. CAPACITY LIMIT

By introducing a capacity limit, our algorithm allows us to maximize the amount of memory occupied by the binarized descriptor. In this case, we choose the Gray code because it is the most compact binary representation of the information

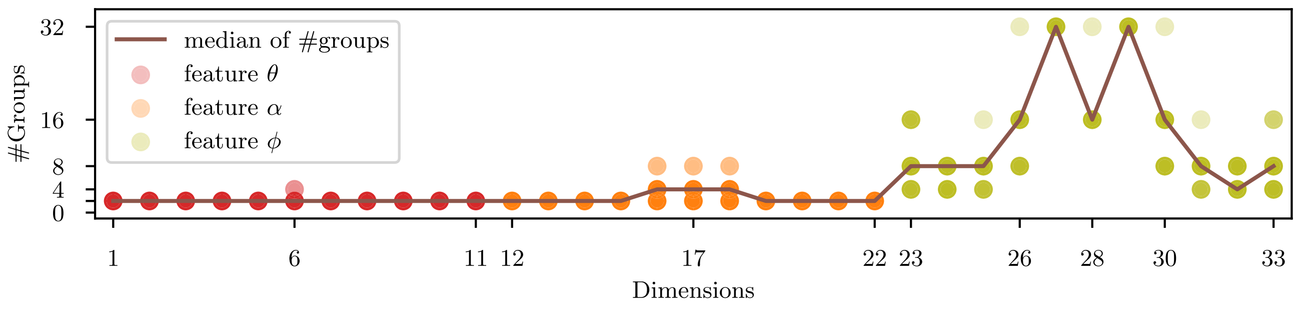


FIGURE 3. Number of groups per point cloud for FPFH. The different dimensions of the descriptor are colored according to which feature they belong to. This figure also shows why feature ϕ is the most interesting.

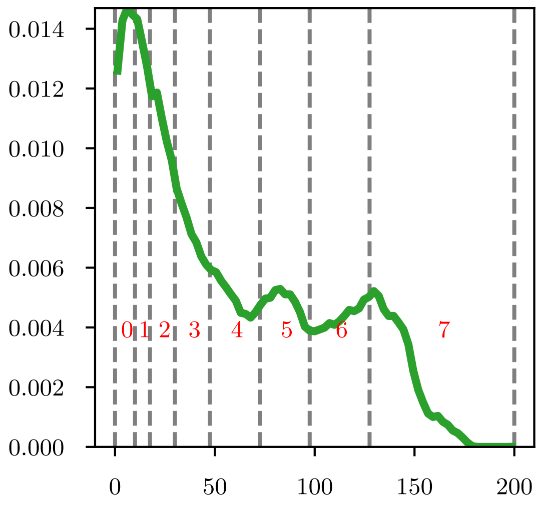


FIGURE 4. Empirical density function of the 28th element of FPFH, after grouping. Table 1 shows the Gray and Mersenne code representations of groups.

TABLE 1. Gray and Mersenne codes for 8 groups. Hamming distance of every two adjacent Gray code is 1. In case of Mersenne code, every code has the same Hamming distance as the distance between their groups.

Group index	Gray code	Mersenne code
0	000	0000000
1	001	0000001
2	011	0000011
3	010	0000111
4	110	0001111
5	111	0011111
6	101	0111111
7	100	1111111

and our evaluation shows that, in general, very similar accuracy can be achieved using the Gray code as with the Mersenne code. To avoid of information loss, each element from the original descriptor should receive one bit at least. It is assumed that $D \leq C$, otherwise the number of elements of the original descriptor can be reduced by changing its internal parameter(s) (e.g. by decreasing the number of FPFH bins). We would like to distribute the remaining $C - D$ bits among the elements in proportion to their requests. To solve the problem, let r_d the number of bits requested by d^{th} element, which is equal to below:

$$r_d = \log_2(\max\{gnum \mid (gnum, endps) \in \mathcal{G}_d\}). \tag{2}$$

TABLE 2. Descriptors and their parameters. Each descriptor were computed with ***radius*** = 0.06. Floats and bits denoted by (f) and (b) respectively.

Descriptor	Parameters	Size
FPFH [1]	deafult in PCL [19]	33 (f)
SHOT [2]	deafult in PCL	352 (f)
Spin Image [3]	deafult in PCL	153 (f)
RoPS [4]	deafult in PCL	135 (f)
VBBD [15]	$g = 4, 7, 9$ $h = 4 * radius * g$	$64, 343, 729$ (b)
LoVS [14]	$m = 4, 7, 9$	$64, 343, 729$ (b)
Gray-SHOT [12]	$L = 352, N = 2$	704 (b)
CI-SHOT [11]	$L = 11, N = 2$	704 (b)
B-FPFH	$L = 4, 11$	33 (b)
B-SHOT [10]	$L = 4$	352 (b)

Also, denote the sum of all requested bits by R:

$$R = \sum_{d=1}^{D} r_d. \tag{3}$$

If $R \leq C$, then each element can get the requested bits, since it fits within the capacity limit. The interesting case is $R > C$. Then the d^{th} element will get l_d bits:

$$l_d = 1 + \lfloor (C - D) \cdot (r_d - 1)/(R - D) \rfloor \tag{4}$$

because each element must recieve 1 bit and the remaining $C - D$ bits must be distribute by the weight of the requested bits $(r_d - 1)/(R - D)$ ($D \leq C < R$ is satisfied by the precondition, i.e. l_d will be a positive integer in any case).

IV. EVALUATION

Our method was compared with all known 3D point feature descriptor binarization methods (B-SHOT [10], CI-SHOT [11], Gray-SHOT [12]) and standalone binary descriptors (VBBD [15], LoVS [14]). To evaluate QBB, we selected real-valued descriptors from those available in Point Cloud Library [19] and Open3D [26]. The methods being compared were applied to the real-valued descriptors used by the authors themselves. Therefore not all binarization methods were run on all descriptors. The binarization methods (B-SHOT, CI-SHOT, Gray-SHOT) and the standalone binary descriptors (VBBD, LoVS) were implemented in Python by us. Our implementations and evaluation code

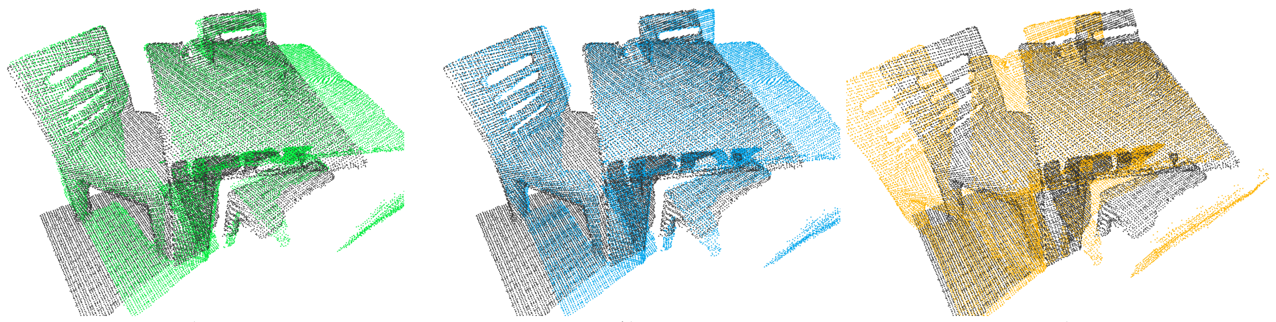


FIGURE 5. **Point cloud registration with different descriptors: (a) FPFH, (b) QBB-FPFH, (c) B-SHOT. The transofrmation matrix was calculated by RANSAC algorithm, implemented in Open3D [26]. The point clouds were cropped for better visualization.**

are available via the following link: https://github.com/ELTE-IK-Point-Cloud-Group/QBB.

The runtime of teh various algorithms strongly depends on their implementation. As stated above, all methods were implemented in Python without parallelization. The runtime of the binarization methods for a point cloud is below a second (few seconds for B-SHOT) on a notebook with regular configuration (Intel Core i5-10300H 2,50GHz; 16GB RAM, 2933MHz). Therefore, we estimate the theoretical computational complexity of each method in Sec. V-D, instead of comparing the runtimes of unoptimized implementations.

An important parameter for real-valued and binary feature descriptors is the radius, which is used to select the neighborhood of the point. The size of the radius depends on the point cloud, its noisiness, the size of the surfaces represented by the cloud, etc. Usually, a descriptor with a larger radius is able to encapsulate more information. Following Guo *et al.* [7], to ensure similar conditions for all descriptors, the same radius was used in all cases.

The publicly available 7-Scenes RGB-D *redkitchen* [27] and Redwood *livingroom* [28] datasets were used for the evaluation. In these data set, a typical point cloud contains roughly 250 000 points. For faster evaluation, the point clouds were downsampled to a voxel leaf size of 0.01, reducing the typical size of the clouds to 100 000 points. The *redkitchen* dataset contains 60 overlapping point clouds with ground truth transformation. To make our evaluation accurate, we selected overlapping cloud pairs that overlap by at least 65% (45 point cloud pairs met this criterion). Fig. 5 shows an aligned point cloud pair from the dataset with different descriptors (the point clouds are cropped for better visualization).

The Precision-Recall Curve (PRC) is widely used to compare the descriptiveness of a descriptor [7]. To obtain the precision and recall values, we iterate over all of the 45 overlapping cloud pairs. From both clouds, we select *sample_num* points with random choice (we set *sample_num* to 5000, which is usually of 5% of all points). The descriptors are calculated for the selected points. In the next step, point pairs (correspondences) are created from the two clouds based on the nearest neighbor ratio [29]. We find the two nearest neighbors of each descriptor of one cloud from the descriptors of the other cloud. If the ratio between the first and the second nearest neighbor is greater than a threshold τ, we consider the point and its nearest neighbor a correspondence. We then need to check how many of the correspondences are correct. To do this, we transform the point clouds using the ground-truth transformation to align them. If two points of a correspondence are closer to each other in Euclidean space after the transformation, than a given support radius, this correspondence will count as a correct match (for support radius we used the same radius as for the descriptors: 0.06). In the next step we calculate the precision and recall values:

$$Precision = \frac{\text{Number of correct matches}}{\text{Number of correspondences}} \tag{5}$$

$$Recall = \frac{\text{Number of correct matches}}{\text{Number of possible correct matches}} \tag{6}$$

To obtain the Precision-Recall Curves we iterate through the τ threshold values from 0.5 to 1. We start from 0.5 because this means that the second nearest neighbor is twice as far away as the first nearest neighbor, which is very rare for real point clouds. However, at lower values, *precision* and *recall* can take extreme values. To illustrate the descriptiveness of a method in a compact way, we will compare the area under curve (AUC) values for different descriptors (the AUC is calculated using metrics.auc function from the scikit-learn library [30]). Since we randomly select points for the evaluation, we run the evaluator 5 times for each pair of clouds and average out the results.

V. RESULTS

In this section, we present the results of our evaluations, where we compared QBB to other relevant methods. Unless otherwise indicated, QBB methods use the default Gray code and conventional Hamming distance without capacity limit. In Fig. 6 (a) we can see the precision and recall values of real-valued descriptors. It also includes the binarized version of FPFH with our method (QBB-FPFH). Spin Image has the worst performance of the real-value descriptors. This result is consistent with the work of Guo *et al.* [7] where the performance of Spin Image also worse compared to other descriptors (FPFH, SHOT, RoPS). When threshold τ is low,

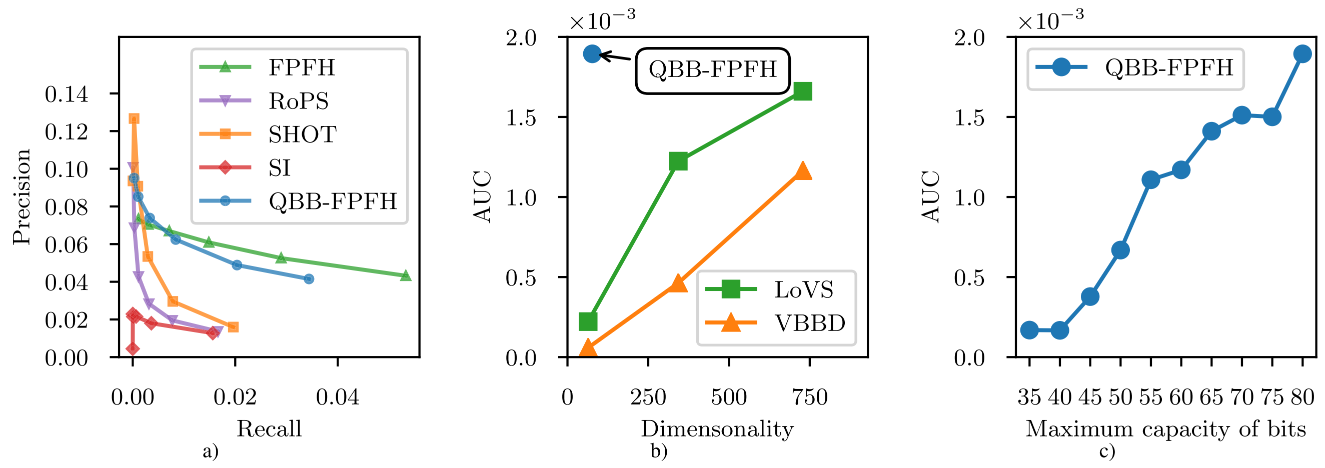


FIGURE 6. (a) PRC of QBB-FPFH and real-valued descriptors. (b) Comparing dimensionality and area under curve (AUC) values of QBB-FPFH and standalone binary descriptors. (c) The effect of capacity limit on descriptiveness of QBB-FPFH.

SHOT and RoPS can achieve high accuracy, but at a higher threshold, the accuracy decreases drastically. Only FPFH can consistently achieve good precision. It can be seen that the performance of QBB-FPFH is almost as good as the original descriptor, and better than the other real-valued descriptors. We will see later that QBB achieves the best results by binarizing the FPFH descriptor.

A. STANDALONE BINARY FEATURE DESCRIPTORS

Fig. 6 (b) shows how QBB-FPFH performs against two standalone binary descriptors. According to the authors of VBBD and LoVS, their methods give the best results when using 729 bits (9 voxels along each ax). The size of the QBB-FPFH with no capacity limit is 77 bits, yet it has a higher AUC than standalone descriptors of 729 bits. If we reduce the number of bits of the standalone methods to 64 and 343 (only cubic numbers are possible), we can see that their AUC value is much smaller. VBBD and LoVS are very similar methods, but they calculate LRF in different ways, which may explain the difference in performance. In Fig. 6 (c), we can see how the descriptiveness of the QBB-FPFH changes as the capacity limit decreases. As expected, using a capacity limit has a negative effect on performance. The largest decrease in AUC occurs when the capacity limit is changed to 50. Based on the figure, if we want to use a descriptor on a device with very limited memory and computational capacity, QBB-FPFH is suitable even with limiting its length to 65 bits.

B. POSSIBLE VARIATIONS OF QBB

The advantage of binary feature descriptors is that 1) they require less memory usage and, 2) the Hamming distance computed by bitwise operations is much faster than computing Euclidean distances between real-valued descriptors. In this subsection, we would like to describe modifications to the QBB that improves its performance at the expense of the advantages mentioned above. As described in Sec. III, we considered several different methods for encoding groups. Table 1 shows the Gray and Mersenne codes in the case of 8 groups. In Fig. 7, we can see that using Mersenne code, QBB can achieve better performance. However, the number of bits needed for one element with real value increases from $\log_2 N$ to $N - 1$. Thus, it requires more memory. However, the increase in the number of bits also creates an anomaly. The QBB may represent each element from the original descriptor by bit sequences of very different lengths. Consequently, if one element is represented by a longer bit sequence than another, the weight of these elements may be increased compared to their weight in the original descriptor in the case of conventional Hamming distance. This affects the binary descriptor created with Gray and Mersenne code too. For example, if the values are gathered into four and eight groups along the two dimensions of the original descriptor, using Gray code we get 2 and 3 bits, respectively, and using Mersenne code we get 4 and 7 bits, respectively. Using conventional Hamming distance, the representation of the second element will have a weight of $3/2 = 1.5$ times higher for the Gray-code version and $7/4 = 1.75$ times higher for the Mersenne-code version. To solve this problem, we introduce the Modified Hamming Distance metric. Let $B = b_1 b_2 \ldots b_l$ and $B' = b'_1 b'_2 \ldots b'_l$ be already binarized descriptors.

$$MHD(B, B') = \sum_{d=1}^{D} \frac{HD(s_B(d), s_{B'}(d))}{l_d}, \tag{7}$$

$$i(1) = 1 \quad \text{and} \quad i(d) = i(d-1) + l_{d-1}, \ 1 < d \leq D, \tag{8}$$

$$s_B(d) = b_{i(d)} \ldots b_{i(d)+l_d-1}, \tag{9}$$

where HD is the Hamming distance function for bit sequences, D is the number of dimensions of the original descriptor, l_d is the number of bits in the binarized descriptor corresponding to the d^{th} element of the original descriptor. To calculate the MHD, we calculate the Hamming distance for each bit sequence corresponding to each element of the original descriptor and then divide it by its length. Thus,

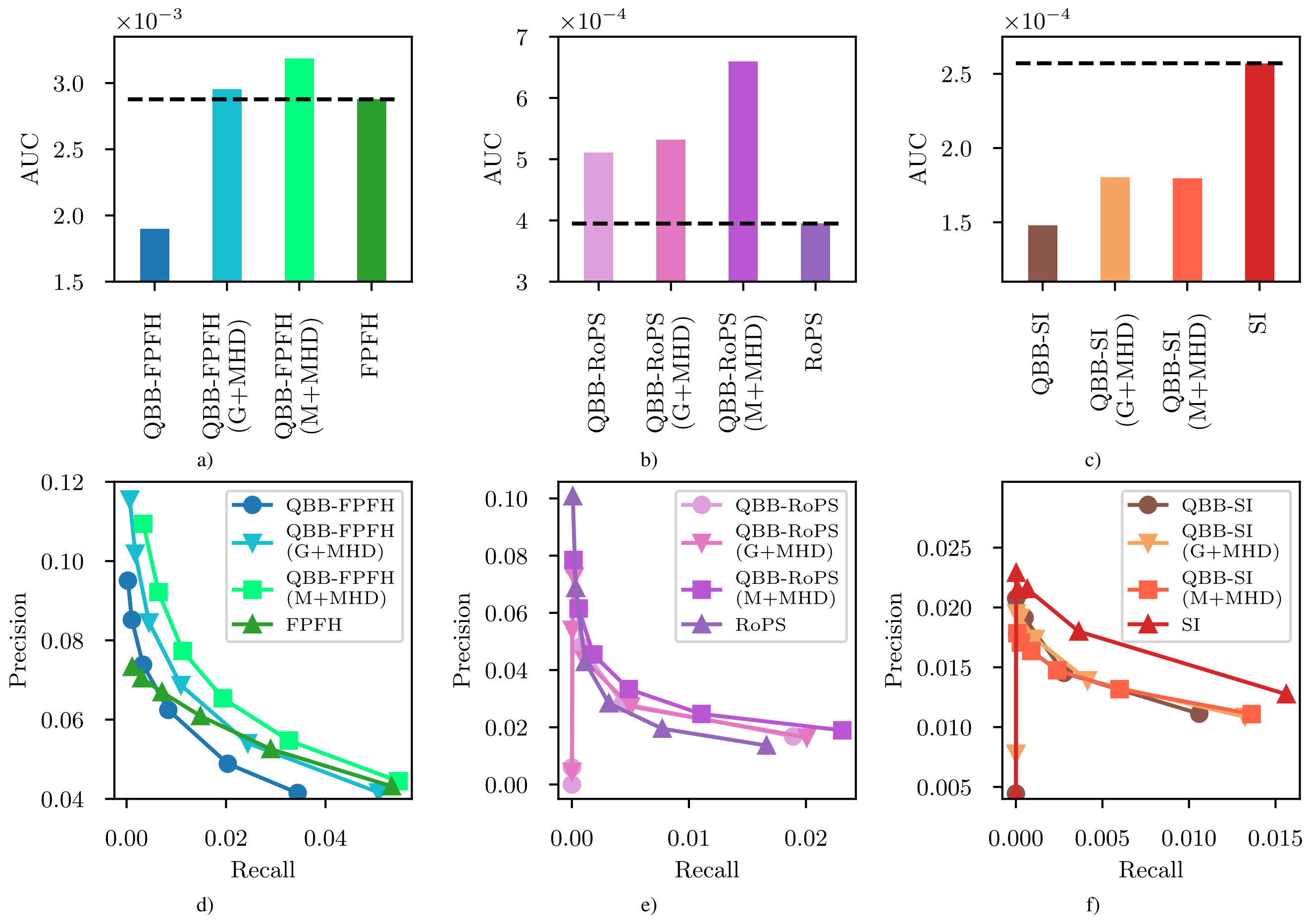


FIGURE 7. The AUC values (a), (b), (c) and corresponding Precision-Recall curves (d), (e), (f) of QBB and its variants. Notations: G - Gray code, M - Mersenne code, MHD - Modified Hamming Distance. QBB-FPFH, QBB-RoPS and QBB-SI use Gray code and conventional Hamming distance, while FPFH, RoPS and SI use Euclidean distance. It is important to note that the AUC values related to FPFH are an order of magnitude larger than the other AUC values.

each bit sequence associated with an element will contribute a value between 0 and 1 to the modified distance, i.e. each element will contribute an equal weight to the distance. To do this, we need to store the information of how many bits are used for representing each element of the original descriptor. Unfortunately, Modified Hamming distance cannot be as efficient as conventional Hamming distance (which can calculated by efficient bitwise operation). However, in some cases, QBB descriptors using MHD for nearest neighbor searches can achieve better results than the original real-valued descriptors.

Fig. 7 shows how different variants of the QBB perform. QBB-FPFH, QBB-RoPS and QBB-SI use Gray code and conventional Hamming distance, QBB-FPFH (G+MHD) / (M+MHD), QBB-RoPS (G+MHD) / (M+MHD) and QBB-SI (G+MHD) / (M+MHD) use Gray code / Mersenne code and Modified Hamming distance. The evaluation shows that QBB variations of FPFH and RoPS using Modified Hamming distance and even QBB-RoPS using conventional Hamming distance can achieve better results than the original descriptor while requiring much less storage space. However, in the case of the Spin Image descriptor, the QBB variations perform worse than the original, although better results can be obtained using the modified than the conventional Hamming distance. Consider the MHD with an optimized implementation, it can be much faster to calculate than the Euclidean distance. (It is noted that in this work we did not focus on measuring the matching speed of distance metrics, nor did we aim for a computationally optimal implementation.) We believe that the use of Modified Hamming distance, as opposed to the conventional Hamming distance, may be justified if optimizing the matching time is not the most important in the use case.

C. COMPARING BINARIZATION METHODS

The properties of the compared binarization methods are summarized in Table 3. Fig. 8 and Fig. 9 show the AUC values of original descriptors and their binarized versions using two different datasets (*redkitchen* and *livingroom*). The AUC values are on a logarithmic scale for proper visualization. For the methods with the QBB prefix, we used the Gray code and conventional Hamming distance. We also calculated the B-ROPS with different parameters ($L = 4, 5$), but precision and recall values were zeros.

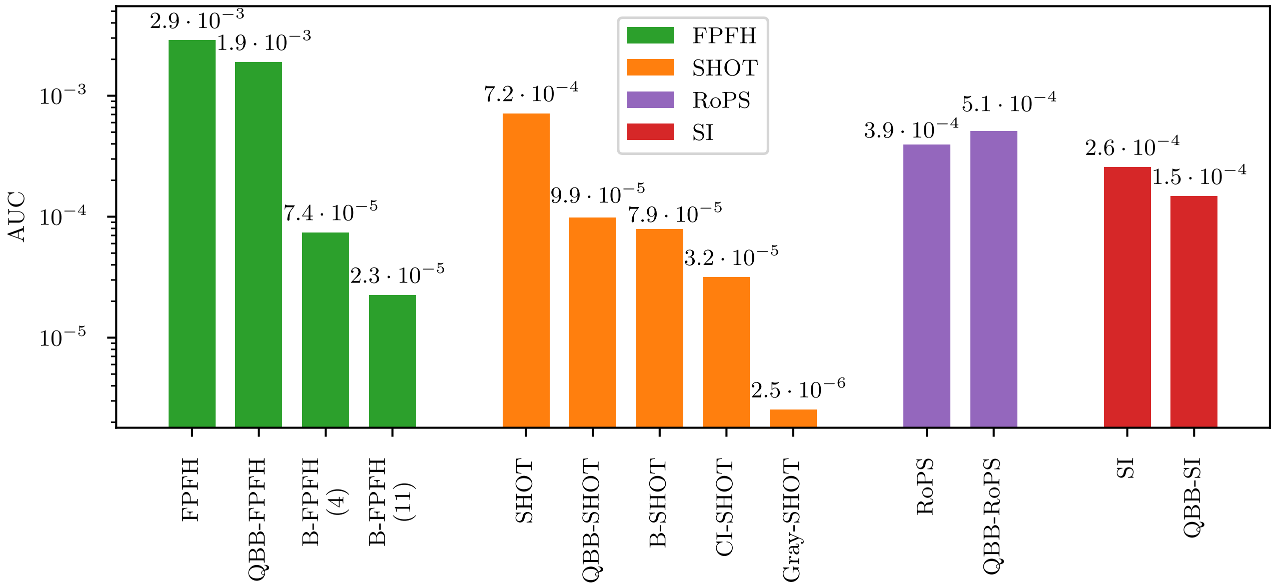


FIGURE 8. AUC of real-valued descriptors and their binarized versions with different binarization methods on the "redkitchen" dataset. The AUC values are on logarithmic scale.

TABLE 3. Properties of the compared binarization methods. The second column shows the number of comparisons for one feature vector. *D* is the length (dimension) of the input vector, *L* is the encoding length, and *N* is the bit number for one real-valued element. The third column shows whether precomputations are required for the method or not. The last column shows the descriptors to which the methods were applied, according to the papers cited.

Method	Comparisons (#)	Precomp.	Applied on
QBB (ours)	$D \cdot 2^N$	Yes	SHOT, FPFH, RoPS, SI
B-SHOT	$\frac{D}{L} \cdot (L \cdot \log L + L)$	No	SHOT, FPFH
CI-SHOT	$\frac{D}{L} \cdot 2^N$	Yes	SHOT
Gray-SHOT	$\frac{D}{L} \cdot L \cdot 2^N$	Yes	SHOT

The B-SHOT, CI-SHOT, and Gray-SHOT methods were specifically designed for binarizing SHOT descriptors. The methods are general, i.e. they can binarize any real-valued descriptor, but the algorithm works well for the distribution of the values of SHOT descriptors. However, Prakhya *et al.* [31] evaluated their method also on FPFH and RoPS. B-SHOT and QBB-SHOT use 352 bits (like the original descriptor), while Gray-SHOT and CI-SHOT use 704 bits. Fig. 8 shows that none of the binarized descriptors can achieve the same performance as the original descriptor. QBB-SHOT and B-SHOT perform similarly for the SHOT descriptor.

In the case of FPFH, the performance of the QBB-FPFH is much closer to the original descriptor than B-FPFH. For B-FPFH we got similar results like the authors in their work [31]. B-FPFH was with encoding lengths of 4 and 11.

On the *redkitchen* dataset, QBB-RoPS gives a slightly better result than the original descriptor (see Fig. 8). It can happen because the binarization and Hamming distance can smooth out the noise in the original descriptor. Note that the descriptiveness of QBB-FPFH is still way higher. For the Spin Image descriptor, QBB-SI performs worse than the original, although it still has a higher AUC than the binarized versions of SHOT.

On the *livingroom* dataset, every descriptor performs worse than on the *redkitchen* dataset (see Fig. 9). The probable reason behind this phenomenon is that the *livingroom* scene is very simple, only containing few interesting surfaces. Thus, it is more difficult to distinguish, e.g., 3D objects and surfaces from each other. Note that group boundaries of QBBs are the same for both datasets, and thus recalculation of group numbers and their endpoints (Algorithm 1) for the *livingroom* dataset is not needed. The ratio of the AUC values between different binarized descriptors is very similar for both datasets. An important difference compared to the *redkitchen* evaluation is that QBB-FPFH results in a much smaller AUC than the original descriptor, the AUC value of the Gray-SHOT method increased significantly, and this time the QBB-RoPS is not better than the original RoPS.

In summary, the QBB method performs better than other binarization solutions for the considered real-valued descriptors. QBB-FPFH has achieved the best results, which could be better than other real-valued descriptors.

D. COMPUTATIONAL COMPLEXITY

B-SHOT, CI-SHOT, and Gray-SHOT binarization methods are the most similar to our method, but there is an important difference between them and QBB. As discussed in Sec. III, the group numbers and boundaries required for QBB must be determined before the binarization. This step cannot be performed during the binarization of individual feature descriptors. The QBB can work well when multiple

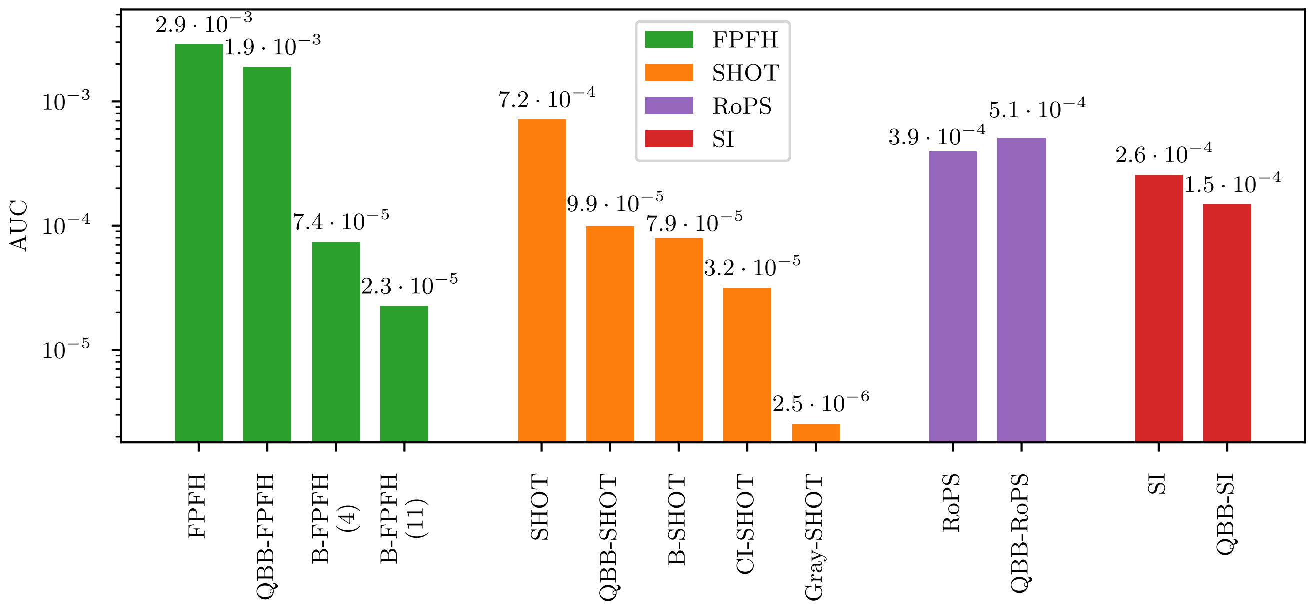


FIGURE 9. **AUC of real-valued descriptors and their binarized versions with different binarization methods on the "livingroom" dataset. The AUC values are on logarithmic scale.**

feature vectors are used to determine the groups. During the evaluation, the feature vectors of several point clouds were used to determine the group numbers, similarly to the training phase in machine learning. Accordingly, the goal is to consider as many different feature vectors as possible. Determining the necessary number of training data is not trivial and requires further work. Our experience shows that for a specific feature descriptor method it is sufficient to define the group numbers once, and these group numbers can be used well for other datasets. The group numbers we determined can be found in the given GitHub repository. For the CI-SHOT and Gray-SHOT methods, calculations need to be performed before the binarization of feature vectors as in the QBB method. For B-SHOT, no prior computations are required.

As a result of the above consideration, the cost estimates we provide in this section only take into account the computational cost of 'online' processing. In a binarization method, the input is a feature vector, so the running time depends on the length of the input feature vector (denoted by D). The most common operation in binarization is the comparison of real values. Therefore, we believe that it is sufficient to only consider those. Other important constants are the encoding length (L) and the number of bits used for encoding a dimension (N). Let $T(D)$ denote the cost of comparisons in a binarization algorithm.

The comparison cost of QBB (T_{QBB}), B-SHOT (T_B), CI-SHOT (T_{CI}) and Gray-SHOT (T_{Gray}) are the following:

$$T_{QBB}(D) = D \cdot 2^N$$
$$T_B(D) = \frac{D}{L} \cdot (L \cdot \log L + L)$$
$$T_{CI}(D) = \frac{D}{L} \cdot 2^N$$
$$T_{Gray}(D) = \frac{D}{L} \cdot L \cdot 2^N$$

The value of L is usually 4 or 11, while N is usually 4 or 5, these parameters can be considered as constants. One can see that the running time of all four methods is $\mathcal{O}(D)$, i.e., it increases linearly with the length of the input feature vector. As a concusion, QBB provides better performance compared to other binarization methods without increasing the computational complexity of the 'online' processing phase.

VI. CONCLUSION

In this work, we proposed a quantile-based binarization method of 3D point feature descriptor, called QBB. We compared our method with other well-known binarization methods and standalone binary feature descriptors. Computations on real point clouds show that QBB can compete with standalone binary descriptors, and it gives better results than other binarized descriptors. We presented possible variants of our binarization algorithm and their performances. Our results suggest that QBB is a suitable replacement for real-valued feature descriptors in certain use cases.

Our conjecture is that for some dimensions less group would be enough. To prove this conjecture our future work includes applying different algorithms to determine the bin width. It would be interesting to see how the group numbers and performance would change if keypoint detection algorithms were used.

REFERENCES

[1] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 3212–3217.

[2] S. Salti, F. Tombari, and L. Di Stefano, ''SHOT: Unique signatures of histograms for surface and texture description,'' *Comput. Vis. Image Understand.*, vol. 125, pp. 251–264, Aug. 2014.

[3] J. H'roura, M. Roy, A. Mansouri, D. Mammass, P. Juillion, A. Bouzit, and P. Méniel, ''Salient spin images: A descriptor for 3D object recognition,'' in *Image and Signal Processing* (Lecture Notes in Computer Science), A. Mansouri, A. El Moataz, F. Nouboud, D. Mammass, Eds. Cham, Switzerland: Springer, 2018, pp. 233–242. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-94211-7_26

[4] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, and J. Wan, ''Rotational projection statistics for 3D local surface description and object recognition,'' *Int. J. Comput. Vis.*, vol. 105, no. 1, pp. 63–86, 2013.

[5] F. Tombari, S. Salti, and L. Di Stefano, ''Unique shape context for 3D data description,'' in *Proc. ACM Workshop 3D Object Retr. (3DOR)*, 2010, pp. 57–62.

[6] X.-F. Han, S.-J. Sun, X.-Y. Song, and G.-Q. Xiao, ''3D point cloud descriptors in hand-crafted and deep learning age: State-of-the-art,'' 2018, *arXiv:1802.02297*.

[7] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok, ''A comprehensive performance evaluation of 3D local feature descriptors,'' *Int. J. Comput. Vis.*, vol. 116, no. 1, pp. 66–89, 2015.

[8] S. M. Prakhya, B. Liu, W. Lin, K. Li, and Y. Xiao, ''On creating low dimensional 3D feature descriptors with PCA,'' in *Proc. TENCON IEEE Region Conf.*, Nov. 2017, pp. 315–320.

[9] D. Varga, J. M. Szalai-Gindl, and S. Laki, ''The descriptiveness of feature descriptors with reduced dimensionality,'' in *New Trends in Database and Information Systems* (Communications in Computer and Information Science), L. Bellatreche, M. Dumas, P. Karras, R. Matulevičius, A. Awad, M. Weidlich, M. Ivanović, and O. Hartig, Eds. Cham, Switzerland: Springer, 2021, pp. 317–322.

[10] S. M. Prakhya, B. Liu, and W. Lin, ''B-SHOT: A binary feature descriptor for fast and efficient keypoint matching on 3D point clouds,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 1929–1934.

[11] L. Du, H. Min, and Y. Lin, ''CI-SHOT: A statistical method for binary feature descriptor on 3D point clouds,'' in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2018, pp. 373–379.

[12] Y. Lin, Y. Sun, and H. Min, ''A general gray code quantized method of binary feature descriptors for fast and efficient keypoint matching,'' in *Proc. 2nd Int. Conf. Intell. Auto. Syst. (ICoIAS)*, Feb. 2019, pp. 1–7.

[13] J. Yang, Q. Zhang, K. Xian, Y. Xiao, and Z. Cao, ''Rotational contour signatures for both real-valued and binary feature representations of 3D local shape,'' *Comput. Vis. Image Understand.*, vol. 160, pp. 133–147, Jul. 2017.

[14] S. Quan, J. Ma, F. Hu, B. Fang, and T. Ma, ''Local voxelized structure for 3D binary feature representation and robust registration of point clouds from low-cost sensors,'' *Inf. Sci.*, vol. 444, pp. 153–171, May 2018.

[15] R. Zhou, X. Li, and W. Jiang, ''3D surface matching by a voxel-based buffer-weighted binary descriptor,'' *IEEE Access*, vol. 7, pp. 86635–86650, 2019.

[16] S. Srivastava and B. Lall, ''3D binary signatures,'' in *Proc. 10th Indian Conf. Comput. Vis. Graph. Image Process. (ICVGIP)*, 2016, pp. 1–8.

[17] Z. Dong, B. Yang, Y. Liu, F. Liang, B. Li, and Y. Zang, ''A novel binary shape context for 3D local surface description,'' *ISPRS J. Photogramm. Remote Sens.*, vol. 130, pp. 431–452, Aug. 2017.

[18] Y. Zou, T. Zhang, X. Wang, Y. He, and J. Song, ''BRoPH: A compact and efficient binary 3D feature descriptor,'' in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2016, pp. 1093–1098.

[19] R. B. Rusu and S. Cousins, ''3D is here: Point cloud library (PCL),'' in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1–4.

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

[21] F. Schrodt, ''Neurocomputational principles of action understanding: perceptual inference, predictive coding, and embodied simulation,'' Ph.D. dissertation,, Eberhard Karls Universität Tübingen, Tübingen, Germany, 2018.

[22] K. H. Knuth, ''Optimal data-based binning for histograms and histogram-based probability density models,'' *Digit. Signal Process.*, vol. 95, Dec. 2019, Art. no. 102581.

[23] D. Freedman and P. Diaconis, ''On the histogram as a density estimator: L_2 theory,'' *Zeitschrift fürWahrscheinlichkeitstheorie Und Verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981.

[24] F. Gray, ''Pulse code communication,'' U.S. Patent 2 632 058, Mar. 17, 1953.

[25] R. M. Robinson, ''Mersenne and fermat numbers,'' *Proc. Amer. Math. Soc.*, vol. 5, no. 5, pp. 842–846, 1954.

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

[27] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi, ''Real-time RGB-D camera relocalization,'' in *Proc. IEEE Int. Symp. Mixed Augmented Reality (ISMAR)*, Oct. 2013, pp. 173–179.

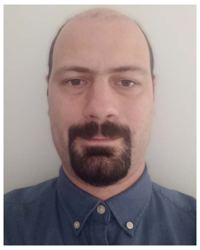
[28] S. Choi, Q.-Y. Zhou, and V. Koltun, ''Robust reconstruction of indoor scenes,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5556–5565.

[29] D. G. Lowe, ''Distinctive image features from scale-invariant keypoints,'' *Int. J. Comput. Vis.*, vol. 60, pp. 91–110, Nov. 2004.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, ''Scikit-learn: Machine learning in python,'' *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Jan. 2011.

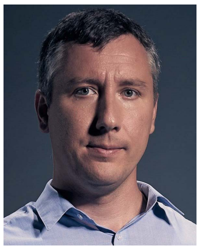
[31] S. M. Prakhya, B. Liu, W. Lin, V. Jakhetiya, and S. C. Guntuku, ''B-SHOT: A binary 3D feature descriptor for fast keypoint matching on 3D point clouds,'' *Auto. Robots*, vol. 41, no. 7, pp. 1501–1520, Oct. 2017.

DÁNIEL VARGA received the M.Sc. degree in computer science from Eötvös Loránd University, in 2017. He is currently an Assistant Lecturer with the Department of Information System, Faculty of Informatics, Eötvös Loránd University. His research interests include 3D point cloud processing related topics, such as triangulation, point cloud matching, and object recognition.



JÁNOS MÁRK SZALAI-GINDL received the M.Sc. degree in mathematics from the Budapest University of Technology and Economics, and the Ph.D. degree from Eötvös Loránd University, Budapest, in 2020. He wrote his dissertation on data-intensive methods for managing scientific data. He has recently participated in multiple research projects in those fields. He is currently working as an Assistant Professor with the Department of Information System, Faculty of Informatics, Eötvös Loránd University. His research interests include scientific databases and data science.

MÁRTON AMBRUS-DOBAI is currently an Active Student with the Faculty of Informatics, Eötvös Loránd University. He enjoys participating in research and is currently working on solving problems related to 3D point clouds as part of a scholarship application.



SÁNDOR LAKI (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Eötvös Loránd University, in 2007 and 2015, respectively. He is currently an Assistant Professor with the Department of Information Systems, Eötvös Loránd University. He has authored more than 40 peer-reviewed articles and demo articles, including publications at JSAC, ToN, INFOCOM, ICC, and SIGCOMM. His research interests include active and passive network measurement, traffic analytics, programmable data planes, and their application for new networking solutions.