J. Vis. Commun. Image R. 30 (2015) 343–349

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

G-SHOT: GPU accelerated 3D local descriptor for surface matching ☆

Linjia Hu, Saeid Nooshabadi *

Department of Computer Science, Michigan Technological University, Houghton, MI 49931, USA

ARTICLE INFO

Article history:
Received 2 December 2014
Accepted 18 May 2015
Available online 27 May 2015

Keywords:
3D Object local descriptor
Point signature
Surface matching
SHOT
Computer vision
Point cloud library
Parallel algorithm
GPU
Real-time processing
Point cloud

ABSTRACT

Signature of histogram of orientations (SHOT) as a novel 3D object local descriptor can achieves a good balance between descriptiveness and robustness in surface matching. However, its computation workload is much higher than the other 3D local descriptors. This paper investigates the development of suitable massively parallel algorithms on the graphics processing unit (GPU) for computation of high density and large scale 3D object local descriptors through two alternative parallel algorithms; one exact, and one approximate. Both algorithms exhibit outstanding speedup performance. The exact parallel descriptor comes at no cost to the descriptiveness, with a speedup factor of up to 40.70, with respect to the serial SHOT on the central processing unit (CPU). The approximate version achieves a corresponding speedup factor of up to 54 with minor degradation in descriptiveness. The proposed algorithms are integrated into point cloud library (PCL), a open source project for image and point cloud.

1. Introduction

Surface matching is a key tool in three-dimensional (3D) object recognition that locates model objects in a scene through building local correspondences between the model and the scene. It has found its way in numerous areas such as computer vision, robotics, automation, remote sensing and perception. The most common method for surface matching is to explore an effective and compact local representations of the point cloud of 3D objects, known as local 3D descriptors, and establish correspondences by matching those descriptors. In the past 20 years there has been strong research interest in local descriptors. The techniques proposed includes structural indexing [1], point signature [2], 3D point fingerprint [3], exponential mapping [4], spin images [5], local surface patches [6], shape index [7], 3D shape context [8], and intrinsic shape signatures [9]. The computation of a local descriptor depends on local reference of each key point, with respect to a normal surface vector. However, in all these proposals, the choice of local reference for each descriptor is ambiguous and not unique.

Most recent work [10,11] has analyzed the repeatability and robustness of existing local descriptor techniques, and divided them into two major categories, *viz.*, signature and histogram. The signature based descriptor describes the 3D neighborhood of a given key point by defining an invariant local reference frame (LRF), according to the local coordinates of points in the neighborhood point set. For each point in neighborhood point set, one or more geometric measurements are encoded. The histogram based descriptor describes the key point by accumulating local geometrical or topological measurements into histogram bins according to a specific quantized domain which requires the definition of either a reference axis or reference frame. Broadly, signature descriptors are potentially highly descriptive due to the use of spatially well localized information, whereas histograms descriptors trade-off descriptive for robustness by compressing geometric structure into bins. To leverage on benefits of both categories, a novel local 3D descriptor named signature of histogram of orientations (SHOT) [10] combines the merits of signature and histogram descriptors. Due to its repeatable LRFs, the SHOT descriptor exhibits a better descriptive power and robustness. However, its benefits comes at a significant increase in the computational complexity.

A point cloud is a data structure for the representation of a multi-dimensional collection of points. In a 3D point cloud, for example, a point in the surface of an object is represented by its x, y and z coordinates. The typical sources for point cloud data set are stereo camera sensors, 3D scanners, or time-of-flight cameras. They are also generated synthetically from a computer model. Fig. 3 is the classic Stanford Bunny generated by a 3D point cloud editor.

SHOT as an effective 3D descriptor has already been integrated into Point cloud library (PCL), a large scale, open source project for

☆ This paper has been recommended for acceptance by M.T. Sun.

* Corresponding author.
E-mail addresses: linjiah@mtu.edu (L. Hu), saeid@mtu.edu (S. Nooshabadi).

http://dx.doi.org/10.1016/j.jvcir.2015.05.008
1047-3203/

2D and 3D image and point cloud processing [12]. PCL framework contains numerous state-of-the-art algorithms that can be used to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract key points and compute descriptors to recognize objects in the scene based on their geometric appearance, create surfaces from point clouds and visualize them.

To overcome the performance bottleneck of the SHOT descriptor in the PCL framework, this paper proposes two alternative highly efficient parallel algorithms that target the massively parallel architecture of graphical processing unit (GPU). We have targeted implementation on GPU platform, as it is finding its way beyond graphics processing into general purpose computing. It offers massively data-parallel architecture alternative to central processing unit (CPU) through large number of computing cores. In particular, GPU has been widely employed for fast and real-time implementation of 3D image and point cloud processing algorithms [13–17]. The potential for the implementation of surface matching algorithm on the GPU comes from the fact that descriptor computations for key points are independent of each other. It is well suited for parallelization on a massively parallel programming paradigm of GPUs.[1] This work presents two efficient GPU accelerated parallel SHOT descriptors.

This paper is organized as follows. Section 2 briefly outlines the mathematical model of SHOT descriptor. Section 3 describes the complexity of SHOT descriptor and presents the exact and approximate parallel alternative implementations of SHOT on GPU (G-SHOT) as a library component in PCL framework. Section 4 presents the experimental results of performance of both algorithms. Section 5 evaluates the trades-off between the speedup and descriptiveness of two parallel G-SHOT descriptors. Section 6 concludes the paper.

2. Mathematical model of SHOT descriptor

The strength of SHOT descriptor is based on two features. First the SHOT is a 3D descriptor and has a repeatable and robust LRF. Second, it combines the merits of signature and histogram categories of descriptors to create a more effective descriptor.

2.1. Repeatable local reference frame (LRF)

To facilitate the following mathematical derivation of LRF, we assume that the radius of the neighborhood sphere for key point $\mathbf{p}$ is R, and there are K nearest neighbors $\mathbf{p}_i$ in the sphere, the covariant matrix of the K points in the neighborhood sphere is $\mathbf{M}$, and the distance between key point $\mathbf{p}$ and neighbor point $\mathbf{p}_i$ is d_i.

The repeatable LRF is based on the estimation of the normal direction of key point on a surface [11]. This estimation involves the total least square (TLS) of the normal direction computed by eigenvalue decomposition (EVD) of the covariant matrix $\mathbf{M}$ of the K points in the neighborhood sphere. TLS of normal direction is represented by the eigenvector corresponding to the smallest eigenvalue of $\mathbf{M}$. To increase the repeatability of the LRF, in the SHOT algorithm, smaller weights are assigned to distant points in the sphere and bigger weight are assigned to nearby points. Also, to improve the robustness, all the K points laying within the sphere that will be used to calculate the descriptor are included in formation of the covariant matrix as,

$$\mathbf{M} = \frac{1}{\sum_{i:d_i \leqslant R}^{K}(R-d_i)} \cdot \sum_{i:d_i \leqslant R}^{K}(R-d_i)(\mathbf{p}_i-\mathbf{p})(\mathbf{p}_i-\mathbf{p})^{\top} \tag{1}$$

[1] In this paper we use NVIDIA's compute unified device architecture (CUDA) [18] computing paradigm. The GPU used in our experiment is the NVIDIA GTX570 with 15 streaming multiprocessor (SM), with each SM having 32 stream processor (SP) cores [19,20].

To uniquely determine the sign direction of the normal at the key point the methodology described in [10] for sign disambiguation for EVD is employed. The methodology is to change the sign of each singular value or reorient each eigenvector to make it consistent with the majority of the vectors used for the computation of the normal. The sign of a normal along a local axes $s \in \{x, y, z\}$ is determined to be as s^+ or s^- in the opposite direction as,

$$\mathbf{S}_{\mathbf{s}}^{+} \doteq \{i : d_i \leqslant R \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{s}^{+} \geqslant 0\} \tag{2}$$

$$\mathbf{S}_{\mathbf{s}}^{-} \doteq \{i : d_i \leqslant R \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{s}^{-} \geqslant 0\} \tag{3}$$

2.2. Descriptor organization

Inspired by well established 2D feature descriptor, scale invariant feature transform (SIFT) [21], SHOT relies on a set of local histograms that compute on a specific subsets of points defined by a regular grid superimposed on the key point patch. For each key point, the SHOT technique uses an isotropic spherical grid partitioned along the radial, azimuth and elevation axes. The coarse partitioning of the spatial grid produces a small cardinality of the descriptors. The choice of 32 spatial volumes is proven to be adequate, resulting in eight azimuth, two radial and two elevation divisions [10,11]. Fig. 1 shows the formation of eight azimuth and the two radial divisions, and Fig. 2 exhibits the formation two elevation divisions.

Each segment within sphere in Fig. 3 encodes a descriptive entity represented by its local histogram. The formation of the local histogram is shown in Fig. 3, for a key point in the point cloud of Stanford Bunny, with one neighbor sphere encompassing the point (light green). For the local histograms of this sphere, we, separately, accumulate point counts for each of the 32 segments into bins according to function $\cos\theta_i$, with θ_i the angle between the normal at each point $\mathbf{p}_i$ within the spherical grid segment (n_{v_i}), and the normal at the key point (n_{u_i}). Choice of binning using $\cos\theta$ has the advantage that it can be easily computed by the dot production as $\cos\theta_i = n_{v_i} \cdot n_{u_i}$. Further, an equally spaced bins on $\cos\theta_i$ is equivalent to a spatial varying spaced bins on θ_i. This has a significant advantage that coarser bins are created for directions close to the reference normal direction and finer ones for the orthogonal directions. For each of 32 volumes in the neighborhood sphere of the key point there is local histograms with 10 bins. So, there is a total of 320 bins for each key point descriptor. Since the descriptor is based upon a set of local histograms, to avoid boundary effects for each point being accumulated into a specified local histogram bin, SHOT perform quadrilinear interpolation between the bin in the local histogram and the bins having the same index in the local histograms corresponding to the neighbor spherical segments within the neighborhood sphere of the key point under consideration.

3. Massive parallelization of SHOT descriptor on GPU

3.1. GPU programming model

The inherent massive-parallelism in the SHOT algorithm can be exploited for implementation on any computing platform that supports fine-grain parallelism. In this paper, we use CUDA `C/C++` extension for parallelizing the computation of SHOT descriptor for each key point. The computing model in CUDA is shown in Fig. 4. This model employs a single program multiple data (SPMD) computing paradigm [22], where parallel programs are encapsulated in *kernel* functions written in CUDA. All program copies, *viz. threads*, are executed independent of each other. Threads are further grouped into a *thread block*. Threads in a thread