Accelerating Bag-of-Features SIFT Algorithm for 3D Model Retrieval

Ryutarou Ohbuchi, Takahiko Furuya

4-3-11 Takeda, Kofu-shi, Yamanashi-ken, 400-8511, Japan
ohbuchi@yamanashi.ac.jp, snc49925@gmail.com

We have previously proposed a shape-based 3D model retrieval algorithm that compares 3D shape based on local visual features. The method first computes a set of multi-scale local visual features from a set of depth images rendered from multiple view orientations about the 3D model. Thousands of visual features per model are integrated into a feature vector for the model by using so-called bag-of-features approach. The algorithm performed very well, especially for models having articulation and/or global deformation. However, the method was computationally expensive; retrieving a model from a 1,000 model database took more than 10s. This because the costs of rendering depth images, extracting local visual features, quantizing these features, and computing distance among the pairs of integrated features can be quite expensive. In this paper, we significantly accelerated the algorithm by using a Graphics Processing Unit (GPU).

Keywords: 3D geometric modeling, content-based information retrieval, GP-GPU algorithms.

1. Introduction

We have previously proposed a shape-based 3D model retrieval algorithm that handles both articulated and rigid models [7]. The algorithm, called Bag-of-Features SIFT, is appearance based, so it accepts a diverse set of 3D shape representations so far as it can be rendered as range images. To achieve invariance to articulation and/or global deformation, the algorithm employs a set of multi-scale, local, visual features to describe a 3D model. We used a saliency-based multi-scale image feature called *Scale Invariant Feature Transform* (*SIFT*) by David Lowe [5] to extract the visual features. To reduce the cost of model-to-model similarity comparison, the set of thousands of visual features for the model is combined into single feature vector by using so-called *bag-of-features* approach (e.g., [2, 3, 9, 11]).

Our empirical evaluation showed that the algorithm performs very well, especially for models having articulation and/or global deformation. For rigid models, such as those found in the Princeton Shape Benchmark [6], the method performed as well as some of the best known 3D model retrieval methods, such as the Light Field Descriptor (LFD) [2] and Spherical Harmonics Descriptor (SHD) [4]. For articulated models, the BF-SIFT significantly outperformed the LFD and SHD.

However, the BF-SIFT algorithm has a high computational cost in computing the feature vector. Rendering depth images, extracting thousands of local visual features per model, and quantizing these features, can be quite expensive.

In this paper, we propose a Graphics Processing Unit (GPU) based approach to accelerate BF-SIFT algorithm. The proposed algorithm employs GPU-based rendering and GPU-based SIFT feature extraction. Along with the use of table lookup in the distance computation stage, the method achieved a query processing time of a few seconds for a hypothetical database having 100,000 models.

2. The BF-SIFT Retrieval Algorithm

We will briefly review the original *Bag-of-Features SIFT* (BF-SIFT) algorithm [7], followed by the method we employed to accelerate the algorithms.

2.1 Original BF-SIFT algorithm

The BF-SIFT algorithm compares 3D models by following the steps below;

1. **Pose normalization (position and scale):** The BF-SIFT performs pose normalization only for position and scale so that the model is rendered with an appropriate size in each of the multiple-view images. Pose normalization is not performed for rotation.
2. **Multi-view rendering:** Render range images of the model from N_i viewpoints placed uniformly on the view sphere surrounding the model.
3. **SIFT feature extraction:** From the range images, extract local, multi-scale, multi-orientation, visual features by using the SIFT [5] algorithm.
4. **Vector quantization:** Vector quantize a local feature into a visual word in a vocabulary of size N_v by using a visual codebook. Prior to the retrieval, the visual codebook is learned, unsupervised, from thousands of features extracted from a set of models, e.g., the models in the database to be retrieved.
5. **Histogram generation:** Quantized local features or "visual words" are accumulated into a histogram having N_v bins. The histogram becomes the feature vector of the corresponding 3D model.
6. **Distance computation:** Dissimilarity among a pair of feature vectors (the histograms) is computed by using *Kullback-Leibler Divergence* (KLD);

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} (y_i - x_i) \ln \frac{y_i}{x_i} \tag{1}$$

where $\mathbf{x} = (x_i)$ and $\mathbf{y} = (y_i)$ are the feature vectors and n is the dimension of the vectors. The KLD is sometimes referred to as information divergence, or relative entropy, and is not a distance metric for it is not symmetric.

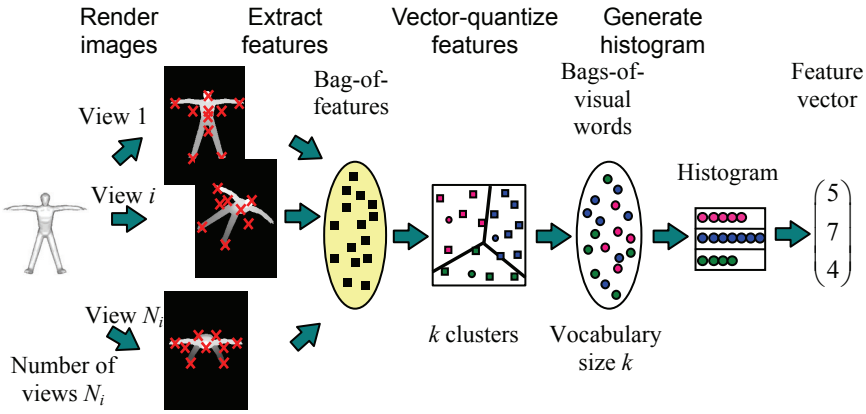


Fig. 1. Range-images of the model are rendered from multiple view angles. Local, multi-scale visual features are extracted from each one of the images using SIFT [4] algorithm. Thousands of SIFT feature per model are vector quantized by using a pre-learned *visual codebook* into *visual words*. Frequencies of visual words are accumulated into a histogram, which becomes an easy to compare and compact feature vector for the model.

2.2. GPU-accelerated BF-SIFT algorithm

The proposed algorithm employs parallelism of a Graphics Processing Unit (GPU) to accelerate two steps, the multi-view rendering step and the SIFT feature extraction step, of the six steps of the algorithm described above,. We keep the vector quantization (VQ) step unchanged. The last step, distance computation step, runs on a CPU but it is accelerated by getting rid of costly calls to logarithmic function by means of table lookups.

1. **Pose normalization (position and scale):** Pose normalization is performed on the CPU.
2. **Multi-view rendering:** Render range images of the model using the GPU, instead of the CPU.
3. **SIFT feature extraction:** As evident in Figure 2, the retrieval algorithm spends the largest amount of time in computing thousands of SIFT features from 42 images of a 3D model. To accelerate the computation, we use the *SiftGPU*, a GPU implementation of the SIFT algorithm by Wu [12]. The SiftGPU does all the work of the SIFT++, that are, construction of a multiresolution image pyramid, detection of interest points, and extraction of features at the interest points, on a GPU.
   While the SiftGPU borrows a lot from SIFT++ by Vedaldi [10], they are not the same. For example, the SIFT++ uses 64bit double precision floating point, while the siftGPU uses 32bit single precision floating point number, for the computation. Thus we compared the retrieval performances of the two SIFT

feature extraction methods. So we experimentally compare the retrieval performance as well as computational cost of the retrieval algorithm using the implementations of the SIFT algorithm.

4. **Vector quantization:** The Vector Quantization (VQ) step is a nearest point query in a very high dimensional (e.g., 1,000) space. It is implemented as a linear search into a set of values whose size N_v is the size of the vocabulary, and runs on the CPU.
5. **Histogram generation:** Quantized local features are accumulated into a histogram having bins, which becomes the feature vector of the corresponding 3D model. This step runs on the CPU.
6. **Distance computation:** Computation of the KLD can be expensive for a high dimensional feature as the computation involves a logarithmic function $\ln(x)$ per element of the feature vector. As the computation is repeated as many times as the number of models in the database, reducing the cost of the log function is quite important for a large database.
   Fortunately, a feature vector produced by the BF-SIFT is a very sparsely populated histogram, in which most of the bins have population zero, and the remaining no-zero elements are small (e.g., <512) positive integers. Thus, the $\ln(x)$ function call can be replaced by a lookup into a small table with no change in performance. The KLD computation using table lookup is performed in the CPU in our current GPU-accelerated implementation.

3. Experiments and Results

We experimentally compared the retrieval performance of our GPU-accelerated algorithm with that of our previous CPU-based implementation by using two benchmark databases: the *McGill Shape Benchmark* (*MSB*) [13] for articulated shapes and *Princeton Shape Benchmark* (*PSB*) [8] for a set of diverse and rigid shapes. We used the same database for learning a visual codebook and for performance evaluation. That is, the codebook generated by using the MSB (PSB) is used to query the MSB (PSB). For each database, the visual codebook is generated by using a set of $N_t = 50,000$ SIFT features, which is chosen pseudo-randomly from all the features extracted from all the views rendered from all the models in the database.

For both SIFT++ (on CPU) and GPU-SIFT, we used parameters of; 42 equally spaced (in solid angle) viewpoints per model, image size of 256×256 pixels per view, scale space having octaves=6, and DOG levels=3. Other parameters for the SIFT++ and the GPU-SIFT are set to their defaults.

We wrote and run the range-image rendering part and SiftGPU feature extraction part using C++, *OpenGL* 2.1.2., and *Cg* 2.0. The vector quantizer and the KLD distance computation parts are implemented by using C++ and run on the CPU. We run the experiment under the Fedora Core Linux on a PC having an Intel Xeon 5440 quad-core CPU (Clock 2.83GHz). The code was single threaded. As for the GPU, we used a mid-range GPU, the Nvidia GeForce 9600GT with 512 MByte of memory, whose core clock is 650 MHz and memory clock is 1.8 GHz.

3.1 SIFT implementations and retrieval performance

We first compare the retrieval performances of the two SIFT feature extractors, the siftGPU [12] that runs on a GPU and the SIFT++ [10] that runs on a CPU. The comparison is done to see the impact of implementation differences, for example, the difference in the precisions of their floating number representations.

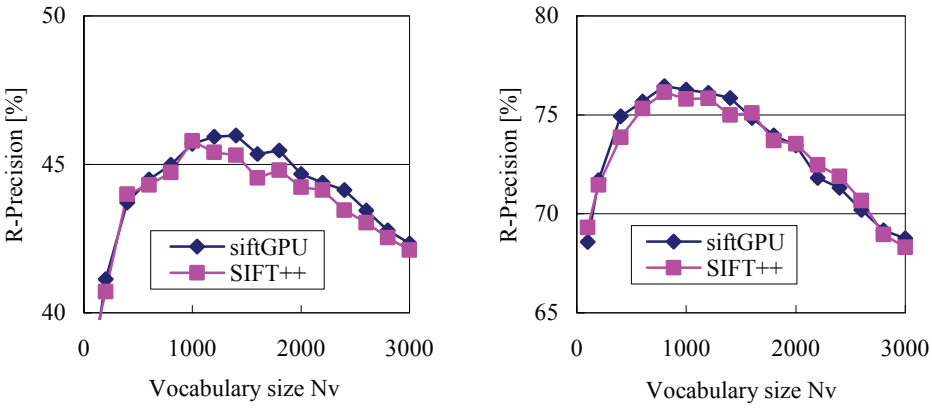
As the performance measure, we used *R-precision* [Baeza-Yates99], which is a ratio, in percentile, of the models retrieved from the desired class C_k (i.e., the same class as the query) in the top R retrievals, in which R is the size of the class $|C_k|$.

Table 1 compares among the two SIFT implementation the average numbers of interest point (i.e., the number of features) per model. For both the PSB and MSB, the number of interest points is essentially the same. The models in the PSB produced more interest points than the MSB, since the PSB models has more detailed shape features than those in the MSB.

Each curve in Figure 2 shows the retrieval performance measured in *R-precision* as a function of vocabulary size N_v. For both of the PSB (Figure 2(a)) and the MSB (Figure 2(b)), retrieval performances are virtually the same for the two implementations; the differences are less than a percentage point.

Table 1. Number of interest points, that are, features, per model for the two SIFT implementations.

	PSB	MSB
SiftGPU	1,989	1,529
SIFT++	1,967	1,570



(a) Result using the PSB. (b) Result using the MSB.

Figure 2. Vocabulary size versus retrieval performances for both CPU-based and GPU-based implementation of SIFT feature extraction. (Note that the vertical scales are different among the two graphs.)

3.2 Computational costs

Figure 3 shows the comparison of computational costs for the three variations of the BF-SIFT algorithm. In the figure, the notation X/X/X/X means that whether the CPU ("C") or the GPU ("G") is used at each of the four stages, that are, (1) depth image rendering, (2) SIFT feature extraction, (3) vector quantization, and (4) distance computation. For the distance computation step, "Cf" denotes the implementation using ln() function, while "Ct" denotes the implementation using table lookup. For example, G/C/C/Cf indicates that the rendering is done on the GPU, while the other stages, that are, the SIFT feature extraction, the vector quantization, and the distance computation using calls to ln() function, are performed on the CPU.

Note that, in this figure, the size of databases is artificially inflated to 100,000 in computing the cost of distance computation. The database size mostly impacts the cost of distance computation. The cost of rendering, feature extraction, and vector quantization are fixed, as they are averages computated from all the models of the database (PSB or MSB). It should be noted that, in reality, the 100,000 model version (if existed) of the PSB (or MSB) will probably have slightly different values of rendering, feature extracton, or vector quantization. But the impact on these values will be small.

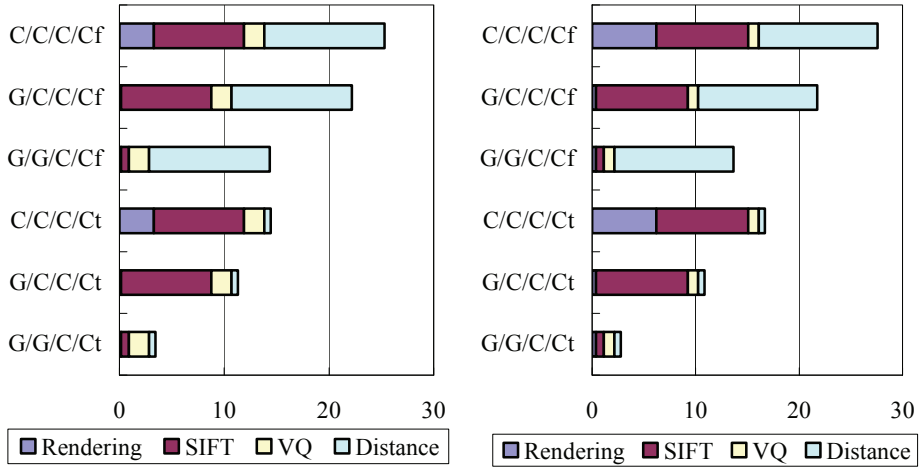
For the all-CPU case using the costly ln() function call, (the case C/C/C/Cf), the distance computation step is dominant, followed closely by the SIFT computation step, and then by the rendering step. Calling ln() function 1,000 times for the 1,000 dimensional feature vector turns out to be qute expensive.

After "modernizing" the implementation so it uses GPU-accelerated rendering, and employing a table lookup to approximate ln() function in the distance computation step running on a CPU (that is, the case G/C/C/Ct), the computational cost of the SIFT feature extraction step becomes dominant. (It is somewhat embarrassing that we did not use GPU-rendering from the start.) Note that the MSB models have higher rendering cost than those in the PSB, since the MSB models have higher polygon counts. The average polygon count of the MSB models is 13,613, compared to 4,373 for the PSB. This is because the models in the MSB are generated as iso-surface meshes from voxel-based models.

If rendering and SIFT feature extraction steps are run on the GPU, and the distance computation on the CPU employs table lookup (that is, the case G/G/C/Ct), the total computation time shrinks to 3.9s for the PSB and 2.9s for the MSB. Compared to the all-CPU implementation that uses table lookup for distance computation (the case C/C/C/Ct), the proposed GPU-based implementation (G/G/C/Ct) is about 3 to 4 times faster.

In the GPU-based implementation (G/G/C/Ct), the cost of VQ has now became the dominant factor. The VQ step for the PSB takes more time than that of the MSB. This is because, on average, a PSB model produces more features than a MSB model (see Table 1.)

Our original algorithm would have taken 25s to query a 100,000 model database statistically identical to the PSB. The proposed accelerated implementation would have performed the query in about 3.9s for the same 100,000 model database.



(a) Cost breakdown for the PSB database queries.

(b) Cost breakdown for the MSB database queries.

Figure 3. Breakdown of computational cost for querying PSB and MSB database. Note that the size of database is artificially inflated to 100,000 in computing the cost of distance computation.

4. Conclusion and Future Work

We have previously published a 3D model retrieval method called Bag-of-Features SIFT (BF-SIFT) that employs a set of thousands of SIFT features [5] to describe a 3D shape. The SIFT is 2D image based, local, multi-scale, and rotation invariant. Our previous experimental evaluation showed that the method is adept at retrieving both articulated and rigid models [7]. However, the method required significant amount of computation, especially for feature extraction.

In this paper, we proposed a GPU-based algorithm that performs multi-view range image rendering and SIFT feature extraction of the BF-SIFT algorithm on the GPU. We also replaced logarithmic functions in the distance computation step with table lookups. Due to these improvements, the method achieved 3 to 4 times speedup with virtually no impact on the retrieval performance. On a hypothetical database similar to the PSB [8] but having 100,000 models, the proposed algorithm would achieve the query processing time of a few seconds.

An analysis of the accelerated implementation indicated a new target for acceleration, the vector quantization and distance computation steps. In the future, we intend to investigate better algorithms for both CPU-based and GPU-based approaches to accelerate these two steps. For example, to vector quantize more efficiently, we we intend to investigate various approximate nearest neighbor algorithms for higher dimensions, e.g., ANN [6].

Reference

1. R. Baeza-Yates, B. Ribiero-Neto, *Modern information retrieval*, Addison-Wesley (1999).
2. D-Y. Chen, X.-P. Tian, Y-T. Shen, M. Ouh-young, On Visual Similarity Based 3D Model Retrieval, *Computer Graphics Forum*, **22**(3), 223-232, (2003).
2. G. Csurka, C.R. Dance, L. Fan, J. Willamowski, C. Bray, Visual Categorization with Bags of Keypoints, *Proc. ECCV '04 workshop on Statistical Learning in Computer Vision*, 59-74, (2004)
3. R. Fergus, L. Fei-Fei, P. Perona, A. Zisserman, Learning object categories from Google's image search, Proc. ICCV'05, Vol. II, pp.1816-1823, (2005)
4. M. Kazhdan, T. Funkhouser, S. Rusinkiewicz, Rotation Invariant Spherical Harmonics Representation of 3D Shape Descriptors, *Proc. Symposium of Geometry Processing 2003*, 167-175 (2003)
5. D.G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, *Int'l Journal of Computer Vision*, **60**(2), November 2004.
6. D. Mount, S. Arya, ANN: A Library for Approximate Nearest Neighbor Searching, http://www.cs.umd.edu/~mount/ANN/
7. R. Ohbuchi, K. Osada, T. Furuya, T. Banno, Salient local visual features for shape-based 3D model retrieval, 93-102, *Proc. IEEE Shape Modeling International* (*SMI*) 2008, (2008).
8. P. Shilane, P. Min, M. Kazhdan, T. Funkhouser, The Princeton Shape Benchmark, *Proc. SMI* 2004, 167-178, (2004). http://shape.cs.princeton.edu/search.html
9. J. Sivic, A. Zisserman, Video Google: A text retrieval approach to object matching in Videos, *Proc. ICCV 2003*, Vol. 2, pp. 1470-1477, (2003)
10. A. Vedaldi, SIFT++ A lightweight C++ implementation of SIFT. http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html
11. J. Winn, A. Criminisi, T. Minka, Object categorization by learned universal visual dictionary, *Proc. ICCV05*, Vol. II, pp.1800-1807, (2005)
12. C. Wu, SiftGPU: A GPU Implementation of David Lowe's Scale Invariant Feature Transform (SIFT) , http://cs.unc.edu/~ccwu/siftgpu/
13. J. Zhang, R. Kaplow, R. Chen, K. Siddiqi, *The McGill Shape Benchmark*, (2005). http://www.cim.mcgill.ca/shape/benchMark/