Closed Continuous Visible Domain Partitioning Of 3D Meshes

Article *in* Indonesian Journal of Educational Science (IJES) · May 2018

CITATIONS
0

READS
26

3 authors, including:

Medhat Rashad
Suez Canal University
2 PUBLICATIONS **3** CITATIONS

SEE PROFILE

RESEARCH INVENTY: International Journal of Engineering and Science
Vol. 8, Issue 2 (May 2018), PP -18-30
Issn (e): 2278-4721, Issn (p):2319-6483, www.researchinventy.com

Closed Continuous Visible Domain Partitioning of 3D Meshes

[1]Medhat Rashad, [1]Mohamed Khamiss, and [1,2]Mohamed MOUSA
(Medhat_Rashad, Mohamed_Khamiss, Mohamed_Mousa)@ci.suez.edu.eg
[1]Faculty of Computer & Informatics - Suez Canal University – Ismailia – Egypt
[2]Faculty of Computers & Information Technology, University of Jeddah, KSA

ABSTRACT - *As the computer processing power is being improved, the size of data is being increased. One of the most powerful recent enhancements is that of the Graphics Processing Unit (GPU). In last years, almost mesh operations and GPU have become linked issues. In fact, the mesh partitioning and GPU are used in several computer graphics applications. In this paper, we present an efficient GPU-based algorithm for partitioning large-scale 3D meshes. The proposed algorithm is called "Closed Continuous Visible Domain (CCVD)" where the processing time, quality and balancing between the parts are our objectives. The partitioning process is parallelized on GPU, and we have evaluated the performance of the proposed algorithm on various large benchmarks. Several experiments have been conducted to evaluate the performance of the proposed algorithm using the Princeton benchmarking. Practically, final results quality is better than the common methods, besides those sub-parts are near to the human Perception. Finally, the execution time of the proposed GPU-based CCVD partitioning is reduced by approximately 40% of CPU time*

Keywords – *GPGPU, 3D Mesh Partitioning*

Date of Submission: 22-04-2018 Date of acceptance: 26-04-2018

I. INTRODUCTION

In computer graphics and animation, devices have been improved, as the size of 3D mesh data also rapidly growing. This improvement is motivated by the needful for more points of interest and higher precision in representation and making objects. Various algorithms have been presented to partition large scale meshes. These algorithms are classified into two classes according to whether the partitioning is supervised or unsupervised. The first class needs post processing, while the second one is not. The post-processing treatments has many factors as jagged boundaries, smoothness, refinement or sharpness to purify the details of the shape [40], [6]. The set of algorithms that belongs to the first class needs additional time for enhancing the outputs. However, the algorithm's output in the other class was consistent mesh partitioning [36], [20]. Shape Diameter Function (SDF) is the most strongly and frequently in the second class. The best approach to partition a shape into disjointed subparts is the approach which final result is consistent mesh partitioning [36]. The term consistent mesh partitioning in SDF means that at mesh surface, a scalar function is defined. This function measures the diameter of the shape's volume within the neighborhood of all points on the surface. Hence, mesh partitioning is the process of decomposing a shape into meaningful parts. This kind of partitioning improve the solution of many computer graphics problems such as: modeling [10], shape compression [1], simplification [8], texture mapping [21], skeleton extraction [20], and metamorphosis [12], [43]. Geometric descriptions and semantic components, are the two ways that almost partitioning algorithms used. In the geometric characterization, the mesh is divided into a number of patches with respect to some particular geometrical attributes, such as curvature and geodesic distance [11], [35], [8], [42], and [13]. However, the mesh is divided into sub-parts that match the related features of the shape based on human perception, that led to the logic components [22], [20], [24], and [19]. In [26] discussed the various methods that used for mesh partitioning. Many techniques have been implemented over CPU. In this paper, we propose GPU-based partitioning for large-scale 3D meshes. Our method characterized by work in a parallel manner. Beside that sub-parts that represent the shape are balanced in the points.

a) 8 sub-parts b) 7 sub-parts c) 8 sub-parts

Figure1) Robust of CCVD in different Armadillo orientations

"Fig. 1" shows that the number of parts looks to be the same. Although, the orientation of the model was changed. That show our method is robust against the model orientation. The performance of the proposed method is compared with the commonly used methods implemented with CPU. We also generalize the method to process the non-manifold meshes. The algorithm presents a great enhancement in terms of processing time when compared to the CPU, especially in large scale 3d meshes.

CONTRIBUTION

We propose in this paper a parallel method for partition large-scale 3D meshes into a set of disjointed sub-parts. Each part will be parallelized and displayed by the group of kernels. In addition, GPU distributes those sub-parts over kernels (a set of threads). The distribution of these sub-parts over those kernels enhances the display and the visualization of the object. Those sub-parts near to be balanced in size. This step minimizes the workload over the stream-processors of GPU that means the load balancing problem is considered. The common and most suitable method is the SDF. This method is consistence mesh partitioning. SDF is a volume-based 3D mesh function that can manipulate features of a shape, which have similarities by using the consistent method. This method also can be used in skeletonization "skeleton extraction" and 3D mesh partitioning. The SDF is a scalar function described on the surface of a mesh, which created from the multi-rays that sent from all input points on the mesh, to measure the distance of the intersection point. Such beam/ray infusion is a deeply parallel mechanism. This task is absolutely ineffective if the processing was on the CPU. Therefore, in the proposed approach, instead of sending a single ray at a time, a parallel method for computing SDF that is implemented on GPU using MATLAB and Mex C++ utilizing the independence of the beams. GPU properties play the main role in the partitioning performance. Our implementation shows the speed difference between CPU and GPU of mesh partitioning, especially in large-scale points of mesh. We test it with several benchmarks and evaluate the performance on NVIDIA GeForce 710M GPU. In practice, GPU-based partitioning algorithm operates extremely rapidly than the traditional one "sequential algorithm on CPU".

The paper is organized as follows. **Section II** addresses a brief survey of previous works, this section consists of three subsections, **subsection II-A** shows the difference between CPU and GPU mesh operations, while is **subsection II-B** explains shortly mesh partitioning types, furthermore in third **subsection II-C** shows the power and improvement of the GPU. Our motivation and proposed method are presented in **Section III**, beside the implementation and the performance. **Section IV** discussed and explained our experimental results. Finally, **Section V** analyze our approach and additional future works.

II. RELATED WORKS

3D Mesh partitioning is a fundamental issue in several mesh operations like modeling [10], shape compression[1], simplification[8], texture mapping[21], skeleton extraction[20], and metamorphosis [12], [43]. Mesh partitioning used as pre-processing step to improve and enhance the final output in many computer graphic problems [18], [41]. As well as used directly in sub-parts labeling [17].One of the most traditional and famous methods which are used for 3d mesh partitioning is The SDF [36]. One of the latest methods that used the SDF for creating closed visible region is called CVF (Continuous Visible Features) [23]. The objective for using the SDF in this method is to reduce the traversal of neighboring points (vertices) on the boundaries in the shape that represents the input mesh.

2.1. CPU VS. GPU PARTITIONING

In the next two subsections, we will focus on the two different partitioning viewpoints. The first is the approaches that depend on the visibility scope for 3d mesh decomposition. The other is the different operations on a mesh based on GPU.

1. **Mesh partitioning based on visibility points:** Analysis and partitioning of the 3D shape provide computer graphics and computational geometry with a fundamental data about the spatial data of the object. The mesh partitioning is the process of dividing a shape into many sub-parts. Each part stands alone as disjoint part. This part work as a new object. Almost points in each part are visible to each other whether direct or indirect. The visibility process creates a visible part. Additionally, this process distinguished between the points which lie on the inner, the outer, and the boundaries of a shape. The visibility process works as a filter the previous case. Many modern works use this visibility scope to determine the shape and its parts features and attributes. The visibility and the functionally "meaningful parts" is the intuition behind of these visually-based characteristics, Here we will discuss different applications based on the visible region. Firstly, The Shape Diameter Function (SDF) [36] is identified

by sampling rays in the cone in the anti-normal direction of a side. The aim of this step to obtain the thickness of a shape locally within the visible points. The value of SDF is measured be the sum of the projected distance of the rays inside the model. However, the SDF may not be matched well to visually semantic components if the thickness is not exactly distributed, as the semantic component should have the similar thickness anywhere.

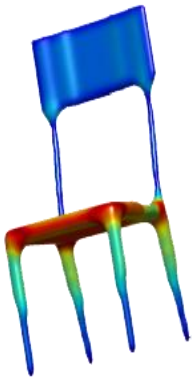


Figure 2) SDF: chair with differ features

For example "Fig. 2", the chair has completely different feature values at its middle from those on the boundary. In addition, the feature values at the ends of the leg are also completely different from those closer to the chair. Another visible region based on feature is, Volumetric Shape Image (VSI) [25], the motive of capturing the general instead of the local volumetric context of the local cone used for sampling rays, VSI attempted to sample rays in multiple directions. For computing, the VSI feature firstly determining the agent middle for all vertices and then testing ray at fixed direction. The range of VSI is obtained by comparing the difference between the sampling of a source vertex and other vertices on the mesh. Extra visible region method based on feature is Continuous Visibility Feature (CVF) [23] there are two points *x* and *y* inside the volume of a shape. Those points are visible to each other if there exists a geodesic path π connecting *x* and *y*. That means *x* and *y* are continuously visible to each other (strong visibility). But, in weak visibility, a point is visible and not by the other point (example, point *x* can see point *y*, while *y* can't see *x*). By using strong visible or weak visible the results directly affected. All previous methods are implemented by the traditional way on the CPU (Serial programming). The performance changed definitely when those method implemented on GPU. The time which the system memory and CPU shared time applications, change the exact performance time. While the dedicated memory of GPU gets the real-time execution. So, the new trend in many computer graphics and many computational geometry problems are using GPU as a platform for solving those problems. Nowadays, NVIDIA Company provides many types of GPU, according to the application applied and the utilization. In last years, there are many methods used GPU to improve the performance and enhance the final results.

2. **Mesh partitioning types:** All GPU architectures can help to solve different computer graphics problems. The goodness and the quality of those results which applied on GPU is better than those done over the CPU, whether this problem is small or large. Both tiny and very complex problem not considered. Here, the discussion about the problems which applied on GPU. Many mesh operation use GPU to increase and enhance both the performance and the output. When the parallel computing applied on the mesh operations, it minimize the time performance and increase the visualization output. In [41] mesh decompression based GPU, the prime step is to partition the shape by Edge breaker [34], then decompression the results of patches over the threads of GPU. The major advantage of partitioning step is reducing the replicated vertices among patches, and balancing the numbers of faces of the results patches. While the results of patches need post-processing step to refinement the boundaries between the faces. The extra process done because the results are jagged boundaries needed to be swapped to fix jagged boundaries. Other mesh based GPU is to recognize the objects within an image, it is a primary operation in labeling objects [18]. The connected component labeling (CCL) is the most commonly used approach to this issue. Although CCL is not easy to be implemented in a parallel way as the joined pixels can be only located primarily by graph traversal. The CCL GPU-based algorithm is a good platform for fast object identification in large-scale images. Otherwise, those mesh operations are applied on CPU or GPU. The mesh partitioning is a common mesh operation. This process known also as mesh partitioning or segmentation. Mesh partitioning divided into two main types, which different methods and procedures can be used. While in [3] presents two approaches for the

computation of the SDF on the GPU are described and compared. (The SDF is a scalar function describing the local thickness of an object. SDF can be used for consistent mesh partitioning and skeletonization). In the first approach is facilitating reorganized the tracing of the rays to be completely adapted for the rasterization hardware. The second approach utilizes parallel ray casting and an octree traversal by using OpenCL.

2.2. MESH PARTITIONING TYPES

a) 3D volumetric Partitioning b) patch partitioning

Figure 3: 3D volumetric vs 2D surface partitioning

Here, we should refer to that we have two differ types of partitioning. The first class is Part Type partitioning, while the other is Surface Type partitioning "Fig.3". The major variation among the two classes has relied on a separate detail and the view that the object being partitioned, both a 2D surface and a 3D volumetric representation. We will concentrate on the first class. This type utilized in studying of humanitarian understanding. Testing human image perception several attempts mean that perception and shape recognition are based on structure mesh partitioning the shape into smaller components or sub-parts[15], [4], [14]. Because of this purpose, part type partitioning segments a 3D mesh to separate parts that oftentimes matched to the bodily 3D semantic components of an object. New comparable studies on the outcomes of some region type mesh decomposition procedures can be located in [2]. While in [28] and [29], region type decomposition is generated relied on investigating the junction curves/sweeps of the ball concentrated nearly all vertices. The investigation partitions a shape into related parts that are either body's parts or extended characteristics, like protrusion characteristics. Part type partitioning was useful in modeling by collecting the object's parts to inspire modern layout [10]. As well, utilized for generating toys as in [32]. Segmenting, recognizing and matching object parts can serve shape identifying and retrieval, and shape restoration [44], [30], and [31]. This part identification can be utilized in morphing [37].Lastly, segmenting object into parts has also assisted in skeleton creation[20], [38], [27], which was utilized in object deformations as well as object animation. Both 3D volume and 2D surface type not easy or difficult to be implemented in any computer environment. This environment affects the performance and the goodness of the results. Both height performance and a good result provide GPU environment. GPU considered by many computer graphics researchers the suitable environment for achieving various tasks related to those types.

2.3. HEIGHT PERFORMANCE AND PARALLEL COMPUTING USING GPU

Recent GPU provides the highest-throughput for computer graphic processors, which have a technical peak rendering of a few Tera-Flops. The generality of these GPUs work on the SIMD (single-instruction multiple data) foundation and the producers are executed concurrently by performing a large-scale number of threads. On the broad, GPU consist of multi processors (MPs), each of them has a number of streaming processors (SPs) and a small shared memory system. For example, in our implementation run on NVIDIA GeForce 710M GPU (Architecture Fermi with code name GF117-N14M-GL). This GPU has 2 Multiprocessor (MPs), each MP contains 96 CUDA cores (Compute Unified Device Architecture) or streaming multiprocessors (SMs) and each CUDA core can run 48 threads. The power of GPUs scales is linearly with the number of cores. To completely utilization of the computational capabilities of the recent GPUs, a good work decomposition scheme needs to be designed. GPU methods have been universally used for differ mesh operations acceleration, and this paper proposed the new method for decomposing the large-scale 3D mesh. To investigate the utilization of GPU in the geometrical processing purpose.

III. PARTITIONING LARGE-SCALE 3D MESHES

Recently, several computer graphics problems solved by using GPU platform in order to increase the effectiveness of the final results. Many computer graphics problems can be decomposed into disjointed sub-problems to get solved efficiently, especially large-scale problems and mesh partitioning problems. In most of those problems, GPU is the best candidate environment that can be rapidly used. Now, many mesh operations and GPU platform are considered a linked issue. Mesh partitioning is a common operation that helps and simplify many mesh analysis problems to be manipulated. Partitioning a large-scale 3D mesh based on the GPU is an integrated problem, decomposing is the problem and GPU is the environment which used as a solution platform.

3.1 Motivations of the paper

Advances in 3D scanning technologies have enabled the practical creation of meshes with hundreds of millions of polygons. One problem with 3D scanners, however, is handling the large amounts of data they produce. In fact, traditional algorithms for display, simplification, and progressive transmission of meshes are impractical for data sets of this large-scale size. Therefore a partitioning of this kind of 3D meshes should be performed. In addition, storing and displaying a collection of large-scale meshes is not suitable for traditional system memory. Therefore, the world last year's offers and suffers in many fields from the big data spatially in computer graphic. Computer graphics have many problems that in need to be treated and solved. Whereas the computer main memory which occupied by running OS and many applications concurrently. Although the system memory function is very significant, its capacity is limited. As well as, the finite numbers of cores. In order to the limitation of the memory and cores of CPUs and the large-scale size of 3D models that computer graphics used, the partitioning of many large 3D models still necessarily to be handled in the non-arbitrary algorithm. Our algorithm partitioning a large scale 3D meshes into many sub-parts, which will meet the multi-thread architecture of GPUs. Then we want to get the result of partitioning contains fewer replicate vertices. As well as, we should to made each sub-part had a balanced number of faces, that led to the partitioning speed will be optimized. Generally, in the last years, the trend toward using GPUs for general computations can be observed. The breakthrough for these approaches came with the introduction of large data set processing. "Fig. 4" shows the great difference between the architecture of CPU and GPU in amount of memory and the number of cores located.

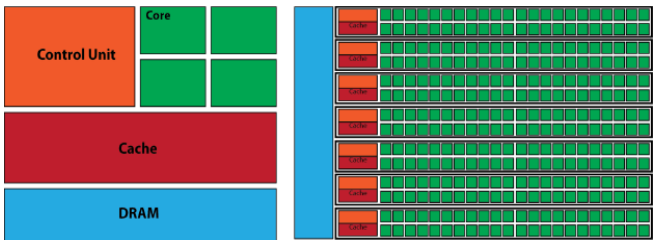


Figure 4) CPU vs. GPU Cores

While "Fig. 5" explains the difference between Fermi vs. Kepler GPU technology. If you have a GeForce 710M graphic card with 2GB of graphic memory, for example, that memory is fully separate from your computer's 8 GB of system memory. Dedicated cards are a perfect way if we are going to professional tasks of a large-scale computer graphic issues.

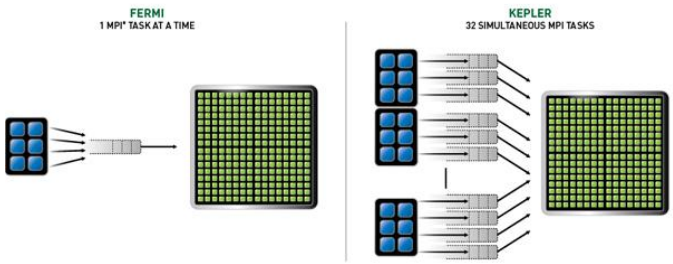


Figure 5) Fermi Vs Kepler architectures

3.2 The Proposed method

Given an input volumetric mesh $\mathcal{M}$, we use the following abbreviation: the set of *n* verities $\{v_1; v_2;...; v_n\}$, the positive domain of the shape (inside the volume of a shape) *D* meaning that " positive domain *D* is the volume of a shape but not seen by any visible vertex *v* of a triangle ΔT, the continuous visible triangle ΔT that contains the visible vertex *v*, the set of visible triangles *T* that contain the set of visible verities and finally the closed continuous visible domain CCVD. The overview of the proposed method is summarized as follows: Filter the faces: The core of this step is to ensure the outside continuous visible area of a vertex v_1, then find the set of continuous visible triangles crossbreeding to the visible vertices from positive continuous domain.

After that and recursively, getting the boundaries between the positive area of a shape and the continuous visible triangles of a visible vertex. The intention of this step to get the interior edges of a shape by applying the geodesic distance manner.

- If ΔT and v_1 of tetrahedron of the positive volume of a shape, then *T* is a positive triangle
- Continuous positive domain *D* of a vertex v_1 is a set of positive triangles *T* that directly connected to this vertex
- Clearly that, a set of vertices v_s is continuous visible area must be a subset of triangles T_v
- Constructing T_v

Finding the boundaries between contentious positive domain of a vertex *v* and continuous visible triangle T_v (recursively):

- Searching for a vertex v_2 that adjacent to T_{v1} and directly connecting to the vertex v_1, where $v_2 \notin T_{v1}$
- This process done iteratively, by searching and connecting two disjointed verities v_2 and v_3 (where v_2 and v_3 are adjacent to T_{v2}, $v_3 \notin \Delta T_{v2}$).
- By constructing the similarity matrix which each entry's value to be 1 for visibility and 0 for invisible. Basically, there exist a geodesic path π connecting v_1 and v_2 and π must be cross a boundary at least between CCVD *v1* and T_{v1}
- A geodesic path π connecting v_0 and v_1 and π must be cross a boundary at least between CCVD v_1 and T_{v0}
- Now, we will search for the third vertex v_3 that closest to π and v_1 that invisible to v_1.The next vertex v_4 be the far and last visible on π.The π geodesic path may not be the shortest path between tow disjoint vertices.

  $$\exists \pi \text{ such that } \forall \pi_1 \in \pi,\ v\pi_1 \cap m = \emptyset,$$

  where π is a geodesic path that connecting two vertices on mesh m.
- Further, v_4 is continuous visible vertex, and edge $e = \{v_4; v_3\}$ that connecting v_4 and v_3 must by boundary edge. After that, we will finding all boundaries as we get the first edge e. This process is done firstly by identifying the visibility of the third vertex $_{v3}$ of the Δt that adjacent to e. By moving on the boundaries edges $e_1 \neq e$ of Δt. [see Table I]

Table I) Weak visibility vs strong visibility

Points classify	Visibility Domain
Strong	Each points *v* and *u* is visible to each other
Weak	A point *v* can be visible by point *u* while the point u can't see point v

- The last vertices also are pair of a visible and invisible. Note that, the closed loop eliminate the vertices that not continuous visible by T_{v_1}

Algorithm 1: Closed Continuous Visible Domain (CCVD)

Data: 3D models
Result: 3D mesh partitioning
Repeat // Host procedure

∀ Face $f_0 \in \partial\, \Delta T_0$ where f_0 direct seen by is v_0 , and visibility region unallocated **Do**

Find a next vertex $v_1 \notin \Delta T_0$

Find a geodesic path π, which connecting v_0 and v_1

$e = \overline{v_0\, v_1}$

v_0 is direct see v_1

v_0 and v_1 are in the same part

Set $e_0 = \{a, b\}$ is the first edge

a visible by v_0 , while b isn't visible by v_0

$List = e$

Repeat // Kernel procedure

Let by c be a vertex can be seen by v_0 , $c \in f_i$ and $\notin e$

If c can be seen by v_0 **then**

$e = \{b, c\}$

Else

$e = \{c, b\}$

End

End

List $List \cup e$

Until List is closed loop

∀ vertices that direct close connected to v_1 without intersection // Device procedure

List has been invisible from v_0

Until all v_n are partitioned

The steps which is shown in "Fig.6" explains the 3D mesh partitioning optimization diagram. Firstly, a model 3d mesh represented (.OFF file or other suitable format) as input mesh. Then get the triangulation of the input model. Third step is responsible for eliminating the replicated vertices by minimizing the traversal vertices or points this step known as mesh simplification or optimization, as we see in "Fig. 7". The fourth step is a recursive step for partitioning the neighboring points, facets, and vertices according to the visibility term. If all points are get partitioned, we get the adaptive sub-parts. Notice that in [Table I] shows the difference between

two visibility points. The strong visibility is better than the weak one, as Points $P_i \in$ Region R_i are visible to each other. This meaning that final results are semantic components. Now we will show the step that run on GPU to increase the performance of the proposed method

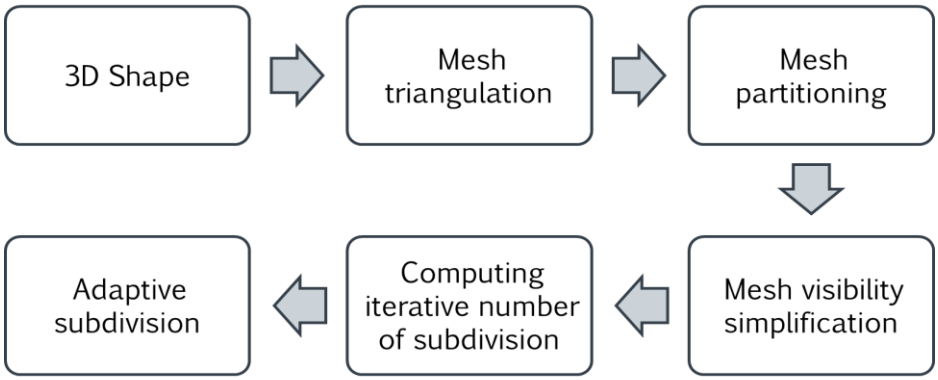
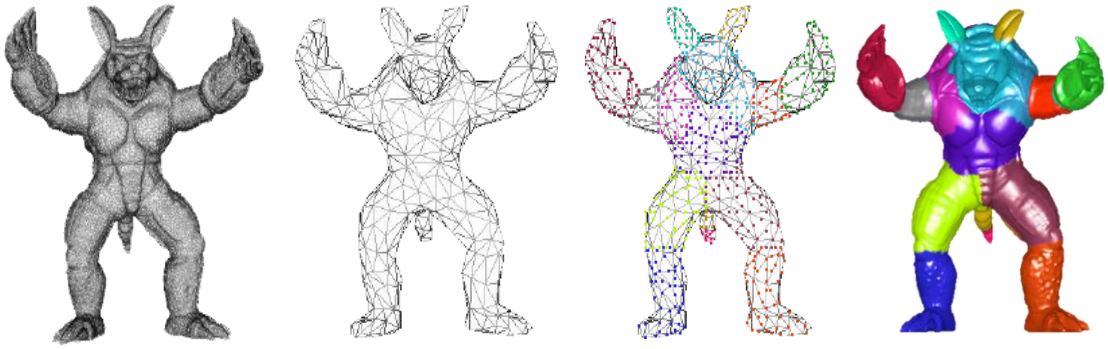


Figure 6) 3D mesh partitioning optimization flow diagram



a) Original Model b) 3D triangulation c) Visibility simplification d) 3D partitioning

Figure7) Reducing the replicated vertices by mesh simplification

3.3 **Implementation**

In this section, we will describe our implementation for partitioning large-scale 3d meshes on GPU, and the performance of our algorithm on various benchmarks. We use MATLAB R2015a (64-Bit) for the core assignment and Microsoft Visual C++ to enable C-MEX for compile C++ code in MATLAB for partitioning and parallel processes. MATLAB Executable (MEX) is designed to support using C++ codes inside the MATLAB IDE to perform rapidly executing and avoid many application bottlenecks. We call C-MEX for executing C++ codes. We work on CUDA toolkit 7.5 as the development environment for GPU, and also using NVIDIA Visual Profiler to estimate the kernel performance time, the data input and output time among GPU and the system or host memory.

We wrote our implementation c-Mex files for several goals:

1. Reusing C++ functions inside MATLAB.
2. Increasing the speed.
3. For unlimited custom extensions

The sequential version of partitioning runs on the machine with the properties that shown in [Table II] Partitioning the mesh into meaningful parts is a basic step toward many mesh operations. The familiar operation is the semantic part-based shape analysis. The SDF is the good method which was used in a shape analysis. This method was implemented in CGAL [9], [5]. In the proposed method had modified CGAL

implementation by replacing CCVD instead of The SDF. The real SDF consists of two steps, first is GMM (Gaussian Mixture Model) [39] the distribution of the feature values. The final step is to partitioning a shape by applying the k-way graph cut. The first one run in CPU, then switching to GPU to achieve the second step. Here, we have two main factors the speed and time which only GPU that can get better performance rather than CPU.

Table II) Machine and graphic card properties

Processor	Inter(R) Core i5-4200M CPU @ 2.50 GHz
RAM	4.00 GB system memory
CPU	NVIDIA Series 710M (dedicated memory 2GB DDR3) - Fermi type
OS	64-bits OS , 64x based Processor

In the previous section we had explained the proposed algorithm. Here, we will clarify GPU performance. GPUs allow manipulation of 3D dimensions faster than CPU. GPUs' memory bandwidth sometimes greater than CPUs memory. Therefore, GPUs appear to be well suitable to accelerate all operations that operate on 3D large scale volumetric data set in independently way; e.g., for transformations, filtering, aggregation, partitioning or other "Big Data" as Fig. 8.

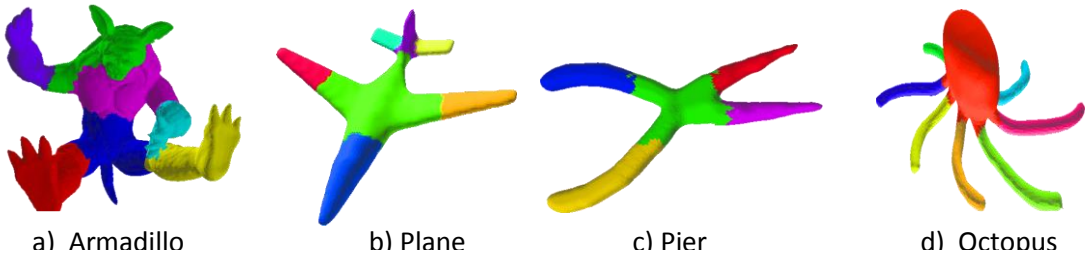


Figure 8) Partitioning created by CCVD

3.4 Performance

The proposed method is implemented in CUDA structure, NVIDIA GeForce 710M GPU which runs (65535* 65535* 64) threads concurrently in the system. In addition, the NVIDIA GeForce710M GPU has 2 streaming multiprocessors (SM), each SM contains 96 CUDA cores and each CUDA core can run 48 threads, so the peak thread of 710M is 9216. For the memory latency and limited capacity of graphics memory, the real power in maximum cases extremely lower than 9216. The machine and the NVIDIA card properties are shown in [Table II]. The machine properties play sub-role in our method, because the provided graphic card is Fermi type. There is a short time shared in the system memory. Using a high-performance computing environment will improve the execution time of the partitioning algorithm which works on processing large-scale data. A high-performance computing environment executes the parallel algorithms of the mesh partitioning on a distributed environment. A distributed environment can be applied using GPU (Graphics Processing Unit) [18].
The performance of both CUP and GPU is near to be equal in small or simple models. But, there is exist a great Difference between them when we applied the procedure on huge complex models. Although, many benchmarks avoid the complex and tiny models from the comparison. Hence, parallel implementation of CCVD algorithm have been very successful especially in large or huge scale. By using the models from the Princeton Segmentation Benchmark [7]. Our implementation run over machine with the following properties.
Both replicated vertices and running time had been minimized.

- **Acceleration of mesh partitioning**

  Our proposal method principally focuses on the speed of partitioning large-scale mesh. The method implemented on computing environments with commodity CPUs and GPUs. The experimental results were collected from the tests on a system with i5-4200M CPU 2.50 GHz CPU and GeForce 710M GPU "Fermi".

  - System memory vs dedicated memory: Shared system memory indicates sharing of the system local memory with the on-board graphics chip. While in dedicated VRAM means applications using memory for all graphics purposes (like rendering, visualization, morphing, and partitioning) will use only the memory on the discrete graphics card thus drastically improving the performance.

- CPU-based CCVD vs GPU-based CCVD: The proposed algorithm tested on both CPU and GPU. For comparing the performance of both processors. Clearly, the running time which required by CPU-based CCVD is greater than GPU-based CCVD. "Fig. 9" shows that when the processed model is about 4k vertices the running time is near to be the same, as the models (Glass, Plier, and fish). While, when models are greater than 6k vertices the running time is changed, as the models (Ant, Armadillo, and Fourlegs).

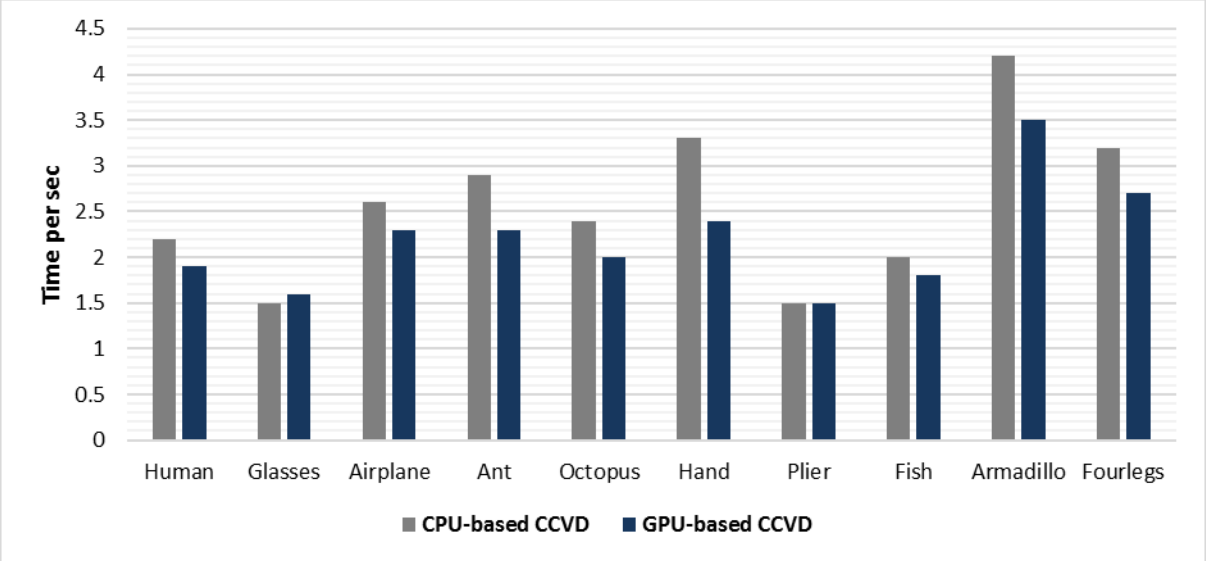


Figure 9) the proposed method performance on both CPU and GPU

IV. EXPERIMENTAL RESULTS

The recent improvement in the power of graphics processing units (GPUs) has overturned us to a viable platform for many computer graphic applications. We had given an overview of the traditional partitioning techniques in [26]. In this paper, we will focus on leveraging the power of GPUs in order to obtain high-performance partitioning of a large-scale 3D mesh. The proposed method is the interactive process, with the premise that these algorithms will lead to faster and higher-quality in 3D mesh analysis problems in the near future. Subsection 33.3 shows the computer environments that we used. In order to verify the efficiency of our method, we compared the performance of our method with the most commonly used method. SDF is the frequently method which used for mesh partitioning in CGAL computational geometry library. We run the proposed method which is called CCVD over both CPU and GPU. There is a variation in execution time especially in large-scale mesh the reference method was proposed in [36] and implemented with CGAL libraries. We will see in "Fig. 10" there exist 8 parts by using CCVD, while 23 parts for the same model. Which led to more times, storage space and other CPU resources.

(a) CCVD 8 sub-parts (b) SDF 23 sub-parts

Figure 10) Armadillo partitioning: CCVD vs. SDF

While in Fig.1 the Armadillo model in different orientation, the number of segments extremely not changed. That means CCVD is robust to the orientation of the model. For extra results, "Fig. 10" Comparing two different methods for mesh partitioning is the factor that shows quality and performance for each them. William Rand

[33] and Hubert [16] proposed a correlation function that modified an issue of comparing two distinct partitions methods with the probable change number of categories into an issue of computing pairwise label similarities

Table III: Compare RI score between CCVD and SDF

Model	CCVD	SDF	Diff
Human	0.14	0.18	0.04
Cup	0.37	0.36	-0.01
Airplane	0.15	0.09	-0.06
Teddy	0.07	0.06	-0.01
Octopus	0.04	0.05	0.01
Plier	0.28	0.38	0.1
Table	0.12	0.18	0.06
Armadillo	0.08	0.09	0.01
Vase	0.16	0.24	0.08
Fourlegs	0.15	0.16	0.01
Ant	0.04	0.02	-0.02
Chair	0.1	0.11	0.01
Hand	0.13	0.2	0.07
Fish	0.19	0.25	0.06
Bird	0.11	0.12	0.01
Average	0.142	0.166	0.024

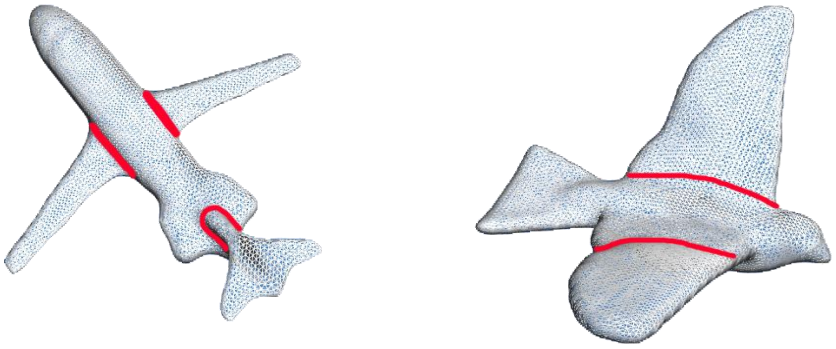
The evaluation measured by **Rand Index (RI)** metric [33], [16]. The RI scores of CCVD and SDF are shown in the [Table III], those values show the improvement that occurs on SDF significantly in the table which has 13926 vertices (from 0.38 to 0.28) and also Plier which has 4487 vertices (from 0.18 to 0.12). Our proposed method improves other categories RI values, with the exception of the airplane and the hand. Generally, we notice from fifteen models the CCVD method CCVD get better RI values than SDF. Notice that the small RI value indicates powerful similarity to the human-made partitioning. Furthermore, "Fig.11" shows the improvement of CCVD results compared to SDF.

Note that:

- The results is consistent, there is no need for post-processing process as in the other methods.
- Many methods are in need for post-processing process like smooth, over lapping, shifting, refinement, and treat boundaries, while our proposed method robustness.

Limitation

Limitation of CCVD, the set of the vertices and points separating the wings and airplane body can directly continuously see both the wing and body. The output in large-scale values can create the partitioning cuts to be in the center of the wings (plane or bird) model. This red domain may cause poor partitioning "Fig. 12".



(a) Bird (b) Airplane

Figure 12) Critical region (in red color), may cause poor partitioning

ANALYSIS

Our proposed approach maps well to recent GPUs, we evaluated the GPU performance and received high acceleration rate. The benefits and the power of our method are:

- Mapping the partitioning process to GPU architecture, keeping all patches are topologically equivalent to a disk,
- Robust to the model orientations
- Load-balanced between the patches
- Completely use the parallelism technique on commodity GPU.
- The performance scales nearly be linear with the number of sub-parts.
- The input/output data time from GPU to host memory is approximately small compared with the whole time for partitioning the large 3D models.
- The final results is consistent mesh partitioning
- Real time execution on commodity GPU
- Tiny “very small” and very complex models are not considered in our implementation
- Accuracy, visualization
- Parallel mechanism (GPU) run more faster than sequential represented methods (CPU)

Figure 11: RI score chart: CCVD and SDF

V. CONCLUSION AND FUTURE WORKS

We presented a new method for a parallel partitioning large-scale 3d meshes joined on GPU kernels. Which requires fewer boundary points/vertices repetition and balanced faces/points number for each part. Our approach is flexible and the partitioning procedure maps well to commodity GPUs. In practice, our method improves the performance of mesh partitioning on current GPU architectures. We observed that the proposed method runs extremely faster than the traditional CPU-based partitioning algorithms, especially in large-scale "Fig. 9". In 3D volumetric partitioning is a fundamental part of different mesh analysis issues, and although many mesh operations had been developed, 3d mesh partitioning is still often done by the traditional way on CPU. Unfortunately, this is a very time-consuming and tedious process. A very hopeful method to trying this problem and generation the partitioning in an interactive way.

In this paper, we present an efficient GPU implementation of partitioning large scale of 3D meshes, was proposed. The method exploits the high performance partitioning based on GPUs to improve the performance of large-scale data. The experimental results show that the proposed method can be a good solution to the mesh partition in large-scale data. There are several avenues for future and later work. We would like to perform the partitioning step faster. We would consider improvement that can get both meaningfulness and balance in the partitioning. What is extra, we can spread our partitioning approach to accelerate other mesh operation and analysis methods. The recent commodity GPU plays the significance role in almost of those steps. As the NVIDIA Company offer four types of GPU. The proposed method are implemented and testes under Fermi type. While the other three types (Kepler, Maxwell, and Pascal) get different results. Next days, a new technology is introduced, seldom the old one had been much cheaper cost and therefore continues to be a good price/performer as there is no limit of the life. The power of human-brain and imagination have no ends see "Fig. 5".

REFERENCES

1. Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. The Visual Computer, 22(3):181–193, 2006.

2. Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patané, Michela Spagnuolo, and Ayellet Tal. Mesh segmentation-a comparative study. In Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on, pages 14–25. IEEE, 2006.

3. Andrea Baldacci, Rastislav Kamenick`y, Adam Rieˇcick`y, Paolo Cignoni, Roman ˇ Durikoviˇc, Roberto Scopigno, and Martin Madaras. Gpu-based approaches for shape diameter function computation and its applications focused on skeleton extraction. Computers & Graphics, 59:151–159, 2016.

4. Irving Biederman. Recognition-by-components: a theory of human image understanding. Psychological review, 94(2):115–147, 1987.

5. Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. Triangulations in cgal. In Proceedings of the sixteenth annual symposium on Computational geometry, pages 11–18. ACM, 2000.

6. Bernard M Chazelle. Convex decompositions of polyhedra. In Proceedings of the thirteenth annual ACM symposium on Theory of

7. Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. In Acm transactions on graphics (tog), volume 28, page 73. ACM, 2009.

8. David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. ACM Transactions on Graphics (TOG), 23(3):905–914, 2004.

9. Andreas Fabri, Geert-Jan Giezmann, Lutz Kettner, et al. On the design of cgal the computational geometry algorithms library. 1998.

10. Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. ACM Transactions on Graphics (TOG), 23(3):652–663, 2004.

11. Michael Garland, Andrew Willmott, and Paul S Heckbert. Hierarchical face clustering on polygonal surfaces. In Proceedings of the 2001 symposium on Interactive 3D graphics, pages 49–58. ACM, 2001.

12. Arthur Gregory, Andrei State, Ming C Lin, Dinesh Manocha, and Mark A Livingston. Interactive surface decomposition for polyhedral morphing. The Visual Computer, 15(9):453–470, 1999.

13. Sumanta Guha. 3d mesh segmentation using local geometry. International Journal of Computer Graphics & Animation, 5(2):37, 2015.

14. Donald D Hoffman and Whitman A Richards. Parts of recognition. Cognition, 18(1):65–96, 1984.

15. Donald D Hoffman and Manish Singh. Salience of visual parts.Cognition, 63(1):29–78, 1997.

16. Lawrence Hubert and Phipps Arabie. Comparing partitions. Journal of classification, 2(1):193–218, 1985.

17. Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. ACM Transactions on Graphics (TOG), 29(4):102, 2010.

18. Young-Min Kang. A parallel approach to object identification in largescale images. In The Second International Conference on Electronics and Software Science(ICESS2016), Japan 2016, 2016.

19. Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. The Visual Computer, 21(8-10):649– 658, 2005.

20. Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts, volume 22. ACM, 2003.

21. Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. ACM Transactions on Graphics (TOG), 21(3):362–371, 2002.

22. Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In Proceedings of the 2001 symposium on Interactive 3D graphics, pages 35–42. ACM, 2001.

23. Guilin Liu, Yotam Gingold, and Jyh-Ming Lien. Continuous visibility feature. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1182–1190. IEEE, 2015.

24. Rong Liu and Hao Zhang. Segmentation of 3d meshes through spectral clustering. In Computer Graphics and Applications, 2004. PG 2004.Proceedings. 12th Pacific Conference on, pages 298–305. IEEE, 2004.

25. Rong Liu, Hao Zhang, Ariel Shamir, and Daniel Cohen-Or. A partaware surface metric for shape analysis. In Computer Graphics Forum, volume 28, pages 397–406. Wiley Online Library, 2009. computing, pages 70–79. ACM, 1981A

26. Mohamed MOUSA Medhat Rashad, Mohamed Khamiss. A review on mesh segmentation techniques. ijeit, 6:18–26, February 2017.

27. Michela Mortara, Giuseppe Patané, and Michela Spagnuolo. From geometric to semantic human body models. Computers & Graphics, 30(2):185–196, 2006.

28. Michela Mortara, Giuseppe Patané, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. Algorithmica, 38(1):227–248, 2004.

29. Michela Mortara, Giuseppe Patané, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In Proceedings of the ninth ACM symposium on Solid modeling and applications, pages 339–344. Eurographics Association, 2004.

30. David Lon Page, Mongi A Abidi, Andreas Koschan, and Yan Zhang. Object representation using the minima rule and superquadrics for under vehicle inspection. In Proceedings of the 1st IEEE Latin American Conference on Robotics and Automation, pages 91–97, 2003.

31. Sylvain Petitjean. A survey of methods for recovering quadrics in triangle meshes. ACM Computing Surveys (CSUR), 34(2):211–262, 2002.

32. Roni Raab, Craig Gotsman, and Alla Sheffer. Virtual woodwork: Making toys from geometric models. International Journal of Shape Modeling, 10(01):1–29, 2004.

33. William M Rand. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical association, 66(336):846–850, 1971.

34. Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. IEEE transactions on visualization and computer graphics, 5(1):47–61, 1999.

35. Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 409–416. ACM, 2001.

36. Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. The Visual Computer, 24(4):249–259, 2008.

37. Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. In Computer Graphics Forum, volume 21, pages 219–228. Wiley Online Library, 2002.

38. Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, Bing-Yu Chen, and Ming Ouhyoung. Domain connected graph: the skeleton of a closed 3d shape for animation. The Visual Computer, 22(2):117–135, 2006.

39. Guorong Xuan, Wei Zhang, and Peiqi Chai. Em algorithms of Gaussian mixture model and hidden markov model. In Image Processing, 2001. Proceedings. 2001 International Conference on, volume 1, pages 145–148. IEEE, 2001.

40. Jie-Yi Zhao, Min Tang, and Ruo-Feng Tong. Connectivity-based segmentation for gpu-accelerated mesh decompression. Journal of Computer Science and Technology, 27(6):1110, 2012.

41. Jieyi Zhao, Min Tang, and Ruofeng Tong. Mesh segmentation for parallel decompression on gpu. In Computational Visual Media, pages 83–90. Springer, 2012.

42. Kun Zhou, John Synder, Baining Guo, and Heung-Yeung Shum. Isocharts: stretch-driven mesh parameterization using spectral analysis. In Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, pages 45–54. ACM, 2004.

43. Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Fast and intuitive generation of geometric shape transitions. The Visual Computer, 16(5):241–253, 2000.

44. Emanoil Zuckerberger, Ayellet Tal, and Shymon Shlafman. Polyhedral surface decomposition with applications. Computers & Graphics, 26(5):733–743, 2002.