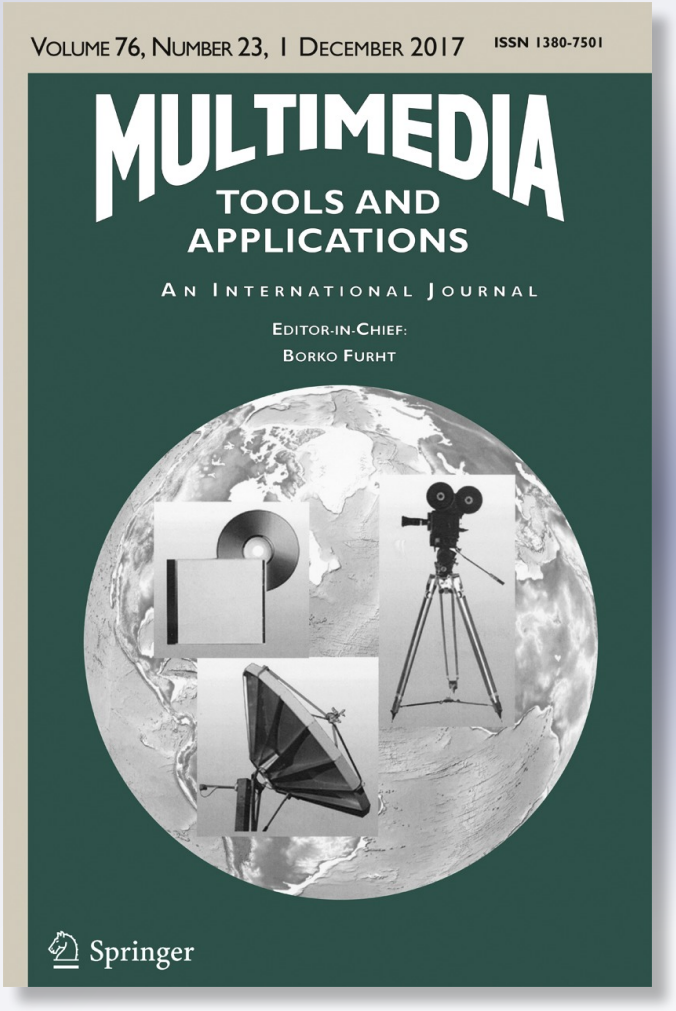
Evaluation of triangular mesh layout techniques using large mesh simplification

Hung-Kuang Chen

Multimedia Tools and Applications
An International Journal

ISSN 1380-7501
Volume 76
Number 23

Multimed Tools Appl (2017)
76:25391-25419
DOI 10.1007/s11042-017-4607-z



Springer

Multimed Tools Appl (2017) 76:25391–25419
DOI 10.1007/s11042-017-4607-z

Evaluation of triangular mesh layout techniques using large mesh simplification

Hung-Kuang Chen[1]

Received: 1 August 2016 / Revised: 8 March 2017 / Accepted: 16 March 2017 /
Published online: 1 April 2017

Abstract Highly detailed polygonal meshes nowadays were created vastly from a great number of multimedia applications. Insufficient main memory space and inferior locality-of-reference nature of the input mesh sequence incurred frequent page swaps and extremely low runtime efficiency. To address this issue, techniques based on the models of graph layout, matrix bandwidth minimization, and space-filling-curves were reported. However, such models merely considered the optimization of vertex layout on the basis of the graph distance or matrix bandwidths while leave the optimization of face layout either unoptimized or resorted an additional sorting procedure. In this paper, we propose an improved theoretic model suggesting the optimization of index-faced triangular mesh layout on the basis of a jointly consideration on both the vertex and face layout. Unlike previous attempts, we do not need additional efforts in optimizing one layout after the other by sorting, the optimized layouts are generated on the fly with each other with respect to both the vertex or triangle bandwidths. According to the experimental results, our approach outperforms the CM and SFC methods not only at yielding better theoretical results but also at improving the runtime efficiency of large mesh simplification.

Keywords Mesh layout · Graph layout · Matrix bandwidth · Mesh simplification

1 Introduction

The advancements in graphics and sensor technology enabled the creation of highly detailed polygonal meshes from a broad range of applications including product reverse manufacturing [29], medical imaging [37, 43, 44, 54, 55], remote sensing [30], and culture heritage

✉ Hung-Kuang Chen
hankchentw@gmail.com

[1] Electronic Engineering Department, National Chin-Yi University of Technology, Taichung, 40109, Taiwan

preservation [28, 45], etc. As the size of reconstructed mesh grows significantly, the memory cost for rendering or processing such mesh may eventually become larger than the size of the main memory space. Under such circumstances, traditional techniques whose main memory requirement are proportional to the input mesh size either failed in execution or ran in a virtual memory space. A conevntional approach, called the *out-of-core* approach, to processing such large meshes is to place partial or complete datasets in *external memory* space [1, 7, 15, 18, 24, 32, 34, 47]. However, such methods usually required frequent non-local accesses to its external datasets and suffered from extremely low runtime efficiency.

To deal with these problems, two types of methods were proposed. One of them solved the problem of frequent non-local accesses by reforming the algorithmic structure from global optimization to localized or streamlined processing. Well-known examples such as [26, 50] and [24] respectively for streaming simplification and compression of polygonal meshes were proposed. The other pursued better external memory access efficiency either by deploying a set of customized caches [8, 9, 23, 41, 46] or by optimizing the referential locality of the input meshes with respect to a certain set of metrics [5, 6, 14, 19, 25, 27, 51, 52]. Since the locality-of-reference nature of an input mesh sequence has significant impact to the caching performance, the optimization of the input mesh's referential locality is highly desirable in both approaches.

To allow efficient rendering by graphics hardware, three-dimensional objects are usually represented as an indexed-face mesh comprising at least an array of vertex coordinates and an array of indexed faces where each face is represented as a triplet of indices referencing the vertex array. However, such kind of meshes reconstructed either from a series of consecutive range scans over the surfaces of a real-world object or from a set of samples collected from a network of sensors were usually constructed regardless of referential locality.

To outline the problem, an example is given as follows. The Armadillo mesh downloaded from the Stanford Scanning Repository is coloured according to its vertex and triangle face layouts by assigning a color to each vertex or face using the following equations.

$$\begin{aligned} C(U) &= (1-U)^2 \times R + 2U(1-U) \times G + U^2 \times B, \\ U &= \frac{i}{|V|}, \forall v_i \in V. \end{aligned} \tag{1}$$

$$\begin{aligned} C(U) &= (1-U)^2 \times R + 2U(1-U) \times G + U^2 \times B, \\ U &= \frac{i}{|F|}, \forall f_i \in F. \end{aligned} \tag{2}$$

If the indexation of the mesh elements, namely, the vertices and triangle faces, are coherent with their distribution, the color of the rendered mesh will be in a continuous tone. In our example, the original Armadillo mesh and its optimized results from our method applying (1) and (2) are shown in the left and right columns of Figs. 1 and 2, respectively.

The discontinuous tones of the visualized images presented in Fig. 1 indicated that both the indexations of the vertices and faces of original unoptimized Armadillo mesh are not coherent with its geometrical and topological distributions, respectively. On the contrary, the continuous tones in Fig. 2 indicated that both the vertex and triangle face layouts of the optimized Armadillo mesh are coherently indexed.

Existing approaches to mesh layout optimization such as the spectral sequencing [25, 35], cache oblivious [3, 46, 52], and space-filling curves(SFC) [33, 38, 40, 48, 51] either considered the optimization only on vertex layout or applied an additional sorting of the faces following to the optimization of vertex layout. Moreover, only few of them have performed empirical evaluations other than theoretical figures such as the maximum or

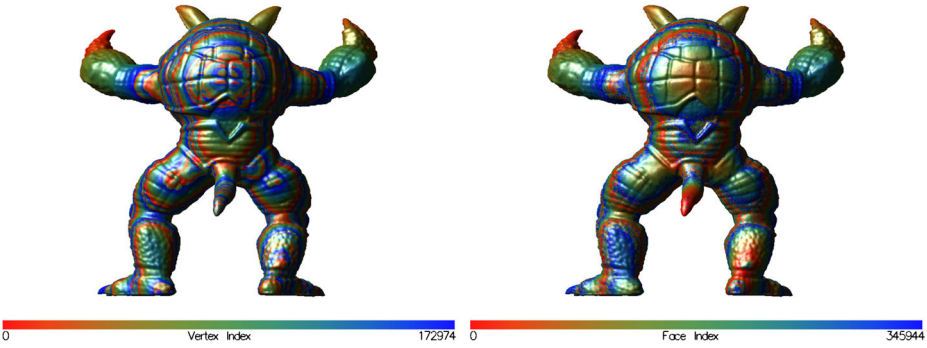


Fig. 1 The Armadillo mesh before mesh layout optimization: left, coloured according to the indexation of the vertices; right, coloured according to the indexation of the triangles

average vertex/triangle bandwidth or spans. Practical impacts of these approaches on the runtime efficiency of common geometric processing tools such as the program running times, cache-hit ratios, or the number of external memory accesses, etc., are unclear.

To addressed these issues, we propose a theoretic model improved from conventional graph layout suggesting a joint consideration on both the vertex and face layout and a level-based heuristic method on the basis of such model. Unlike previous attempts, we do not need additional efforts to optimize the layouts one after the other by sorting; instead, both the vertex and triangle face layouts are optimized on the fly with each other with respect to the vertex or triangle bandwidths.

Furthermore, since the polygonal mesh simplification is crucial to the success of a wide range of applications including level-of-detail generation(LOD), mesh optimization, remeshing, segmentation, parametrisation, skeleton extraction, animation, and morphing, etc., an evaluation over the existing methods and ours in terms of their practical impacts to the run time efficiency of large mesh simplification is conducted.

According to the experimental results, our approach outperforms the CM and SFC methods not only at yielding theoretical better layouts but also at improving the runtime efficiency of large mesh simplification. Since gigantic meshes nowadays are vastly produced from a great number of applications, we believe that our novel approach along with the empirical evaluations presented in this paper can help the practitioners in 3D multimedia related fields improving the runtime efficiency of their applications.

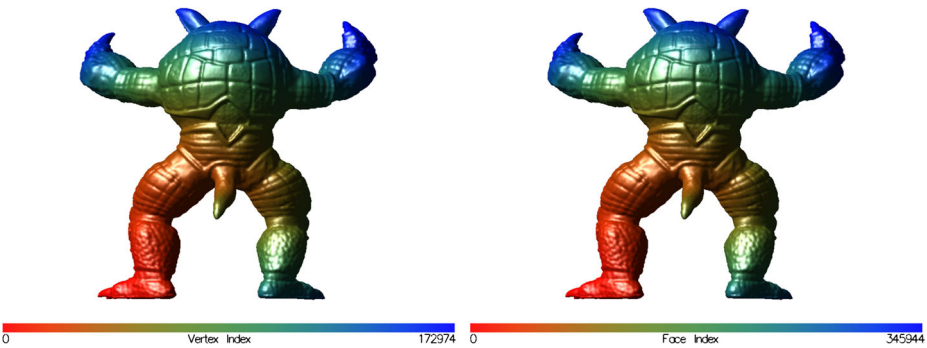


Fig. 2 The Armadillo mesh optimized with our method: left, coloured according to the indexation of the vertices; right, coloured according to the indexation of the triangles

2 Preliminaries

Prior to the discussion of our works and experimental results, we briefly review a number of related terminologies and models as follows.

2.1 Terminologies

Despite a number of advanced formats proposed for external memory processing [11, 25], indexed face mesh by far is the dominant representation for 3D surface models. An indexed-face mesh denoted as $M = \{V, F\}$ comprises a set of vertices V and a set of faces F. A vertex $v \in V$ is usually represented by a triplet of real numbers, i.e., $v = (x, y, z)$, $x, y, z \in R^3$; on the other hand, a polygon face $f \in F$ is represented by an n-tuple of indices. In particular, a face f of a triangular mesh is a triplet of indices to the vertex array V; namely, if the three vertices of a triangular face $f \in F$ is v_a, v_b, and $v_c \in V$ then the face f is represented as $f = (a, b, c)$ where a, b, c are the indices referencing the a-, b-, c-th vertices in V, respectively.

Furthermore, with respect to a vertex $v \in V$, a number of related topological entities are illustrated as follows.

- The *star* of a vertex $v \in V$, denoted as $S(v)$, is the set of edges connected to v.
- The *ring* of a vertex $v \in V$, denoted as $R(v)$, is the set of faces $f \in F$ adjacent to v.
- The *crown* of a vertex $v \in V$, denoted as $C(v)$, is the boundary vertices of the ring of v.

The problem of finding an optimal sequence or ordering of a given set of data with respect to an objective cost function, or the layout problem, has many equivalent forms and applications. A pioneering works for *matrix bandwidth minimization* using graph traversal techniques by Cuthill and McKee(CM) [12] is a famous example. In the fields of graph theory, another formulation for a class of similar problems is called the *graph layout* problem [16]. The two models and a number of related works will be briefly reviewed as follows. For a more comprehensive survey, readers of interest may refer to [16, 39, 49].

2.2 The problem of optimal graph layout

The *optimal graph layout* problem is considered as a particular class of combinatorial optimization problems, whose goal is to find a linear layout of the graph, i.e., a sequential discrete labelling of the vertices, that optimizes the cost of the graph with respect to an objective cost function [16]. For its effective use in a broad range of applications, the graph layout problem has many equivalent forms and applications. A brief review of the formulations and related works is introduced as follows. A more comprehensive survey of the related works can be found in [16, 39, 49].

To give a formal definition of the problem, we begin with a number of terms given as follows.

Definition 1 (Layout) Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices. A *layout* of the graph G is a bijective or labelling function $\phi : V \rightarrow \{1, \cdots, n\}$. The set of all possible layouts of a graph G is denoted as $\Phi(G)$.

Definition 2 (Edge Bandwidth) Given a layout $\phi \in \Phi$ of G and an edge $e = (u, v) \in E$, denoted as $\overline{uv}$, connecting a pair of distinct vertices $u, v \in V, u \neq v$.

We define the *bandwidth*, or the *label distance*, of $\overline{uv}$ with respect to the layout ϕ as follows.

$$\lambda(\overline{uv}, \phi, G) = |\phi(u) - \phi(v)|. \tag{3}$$

On the basis of (3), a formal definition of the *bandwidth* of a graph is given as follows.

Definition 3 (Graph Bandwidth) Given a layout ϕ of G, the *bandwidth* of a graph G with respect to ϕ, denoted as $B(\phi, G)$, is determined by

$$B(\phi, G) = \max_{\forall \overline{uv} \in E} \{\lambda(\overline{uv}, \phi, G)\}. \tag{4}$$

The problem of finding an optimal graph layout is defined accordingly as follows.

Definition 4 (Optimal Graph Layout Problem) Given a graph $G(V, E)$, the problem of *optimal graph layout* is to find an optimal layout $\phi_{min} \in \Phi$ of G such that

$$\phi_{min} \in \arg\min_{\phi} B(\phi, G). \tag{5}$$

A common way to represent a layout of a graph G is to align its vertices on a horizontal line. For example, in Fig. 3, the layout of the graph is equivalent to an alignment of its vertices on a horizontal lines of grids from 1 to $|V| = 8$, which implies $\phi : V \to \{1, 2, \cdots, 8\}$. According to this layout, $\lambda\left(\overline{AB}, \phi, G\right) = 1$, $\lambda\left(\overline{AC}, \phi, G\right) = 2$, $\lambda\left(\overline{AD}, \phi, G\right) = 3$, etc; therefore, the bandwidth of the graph $B(\phi, G) = 3$.

Note that, in this case, the minimal degree of a vertex in G is 3, i.e., $\min_{\forall v \in G}\{|S(v)|\} = 3$; hence, the illustrated layout is also an optimal layout.

2.3 The problem of matrix bandwidth minimization

By finding a permutation of the rows and columns of a sparse matrix so that the non-zero elements is as close as possible to the main diagonal, the matrix bandwidth minimization problem originated in the 1950s is mainly applied to reduce the computations

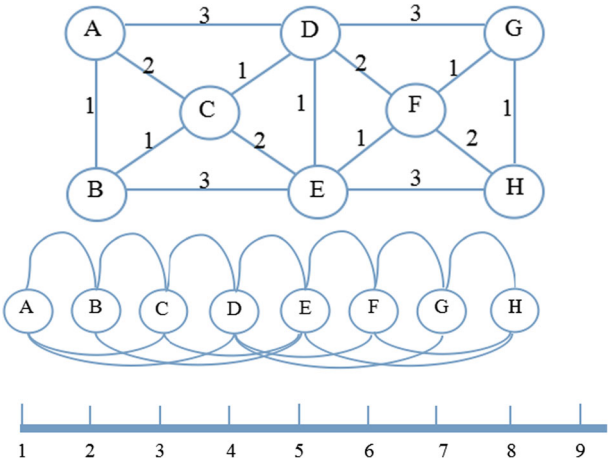


Fig. 3 An example layout of a graph with 8 nodes

in solving large linear systems [49]. A formulation of such problem is stated as follows [12].

Definition 5 (Matrix Bandwidth Minimization) Given a sparse matrix $\mathbf{A}$, find a permutation matrix $\mathbf{P}$ such that

$$\hat{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^T, \tag{6}$$

where non-zero elements $\hat{a}_{i,j}$ of $\hat{\mathbf{A}}$ is close to the diagonal as possible, i.e.,

$$\arg\min_{\mathbf{P}} \max |j - i|. \tag{7}$$

A pioneering work on *matrix bandwidth minimization* proposed by Cuthill and McKee(CM) [12] assuming a symmetric matrix input suggested treating the matrix as an adjacency matrix of an undirected graph and deriving a near optimal solution using a simple layer-based traversal. In their approach, the labelling started with the node of minimum branches then proceed to the labelling of its neighbouring nodes with respect to their number of branches in ascending order. The layer-based traversal is performed iteratively until all the vertices in the undirected graph are labelled.

An example of such symmetric sparse matrix is shown in Fig. 4, which can be treated as an adjacency matrix of a graph layout.

In spite of the fact that the traversal does not guarantee an optimal graph layout yielding minimum bandwidth, the resulting layout is often near optimal and the corresponding matrix bandwidth is effectively minimized. On the basis such concept, a number of subsequent approaches achieving better results were proposed [17, 21]. A more comprehensive study of a number of more recent approaches to this regard can be found in [49].

3 Related works

The concept of optimal ordering of rendering primitives with respect to a fix size of cache or registers is pioneered by Michael Deering [13] then formulated as the *minimum time rendering* (MTR) problem for the minimization of the rendering time by [2]. A number of later work investigated the use of a cache in the graphics processor to transparently buffer

Fig. 4 A sparse matrix corresponding to the graph layout example

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

recently referenced vertices and suggested a preprocessing phase to reorder the faces of a mesh to maximize the references to a vertex cache [1, 4, 5, 10, 23, 42].

3.1 Large mesh simplification

While highly detailed meshes created from applications such cultural heritage preservation and medical or scientific visualization commonly comprise over tens of millions of triangular faces, to effective render or process such meshes, mesh simplification are usually applied. However, traditional in-core algorithms such as the QSlim [20] usually have an input-sensitive memory footprint that requires relatively large amount of main memory space; hence, they are usually applied to models less than few millions polygons. Instead of using the main memory space, a conventional approach called *out-of-core* keeps the entire working set including the inputs, intermediate data, and outputs in external memory space.

A conventional technique called the *mesh cutting* adopted *divide-and-conquer* scheme by cutting the input mesh into smaller pieces for conventional in-core processing followed by merging [11, 22, 24]. Examples such as [22] for terrain mesh simplification, [24] for mesh compression, and [11] for large mesh simplification. However, such techniques require appropriate segmentation and re-indexation scheme for the input mesh and the sub-meshes, respectively. Furthermore, special care at the stage of merging is required in preventing from visible seams or cracks between the boundary of sub-meshes.

To process large meshes, a branch of techniques suggested streamlined and localized processing. With a small size of buffer, the incoming and in-process parts of mesh are kept in the main memory space and the finished parts are output directly to the disk file [25, 50]. To facilitate streamlined processing, a mesh layout technique called *processing sequence* suggested reordering and interleaving of vertex and face data to form a localized mesh sequence. [25–27] . Another branch of methods suggested using an external memory efficient data structure based on Octree [11].

By providing or considering a set of customized application layer software caches, the *cache-based* or *cache assisted* methods significantly improves the I/O efficiency and lowered the execution time with very limited main memory resources [8, 9, 31].

Among all the aforementioned approaches, the layout or referential locality of data has significant impact to the external memory access efficiency. If a better locality-of-reference layout was given, a great amount of external memory accesses resulting from cache misses can be avoided. Addressed on this issue, the optimization of mesh layout for better external memory efficiency were proposed similarly with those for better rendering efficiency.

3.2 Mesh layout optimization

Instead of optimizing the rendering sequence with respect to a limited amount of vertex caches inside the graphics processor, the focus of mesh layout is on optimizing external storage formats with respect to cache coherence. Existing works mainly based on the technique of spectral sequencing [25, 35], cache oblivious based on heuristics [52], graph seperator [3, 46], and space-filling curves(SFC) [33, 38, 48, 51]. A brief survey of these works are as follows.

To improve the coherence or referential locality of the mesh layouts, a linear ordering of the mesh elements, namely the vertices and the triangles is usually applied as a preprocessing stage. However, an optimal solution so far are NP-hard [16].

Isenburg et al. attempts to optimize the mesh layout for efficient external memory access by applying spatial sorting, topological sorting, or spectral sequencing over the vertex layout [25–27] followed by an external sort over the triangle layout according to the minimum or

maximum vertex index of a triangle face. To quantify the mesh quality, a set of metric, the spans and widths of a vertex or triangle, are introduce to improve caching performances.

By treating the problem of optimal mesh layout as a special case of spectral sequencing, Liu et al. proposed using the subdominant eigenvector of a kernel (affinity) matrix to for sequencing [35] where the layouts are computed by subsampling and eigenvector extrapolation.

Based on a two-level memory model, the *cache oblivious* methods resort to algorithmic and data structural reforms pursuing low external memory access complexity regardless of disk caches [46, 52]. By casting the mesh layout problem as a graph optimization problem, S.-E. Yoon et al. suggested computing cache-oblivious(CO) layouts with respect to two global cache-oblivious metrics to quantify the mesh layout quality [52]. Based on the runtime access pattern of the vertices [52], an edge span distribution of the layout is constructed to minimizes the expected number of cache misses regardless of cache parameters. Unlike [25], they compute only the vertex layout and induce the triangle layout rather than computing them separately. Their formulation can be extended to compute layouts of bounding volume and multiresolution hierarchies of large meshes [51].

Tchiboukdjian et al. introduced another CO layout algorithm for irregular but well shaped meshes with a theoretical performance guarantee [46]. Their approach mainly relies on the Binary Space Partitioning(BSP) algorithm of Miller et al. to recursively partitioning the mesh [36]. A CO layout is obtained by traversing from the first leaf of the BSP tree to the last one. However, an experimental report has shown that "even highly optimized cache-oblivious programs perform significantly worse than corresponding cache conscious programs" [53].

A more recent work based on a decomposition tree, called the relax-balanced decomposition tree, is proposed [3] where an optimal layout is obtained by performing an in-order traversal over the leaves of the decomposition tree [3]. In their work, two algorithms were proposed for finding cache-oblivious mesh layouts that guaranteed asymptotically optimal memory performance for mesh update regardless of the cache parameters.

Sagan et al. used space filling curves in computing cache coherent layouts of volumetric grids or height fields [40], which is later widely applied to improve performance of exploring very large grids [38], terrain [33], and mesh layout [51]. Such layout can be constructed by embedding the input mesh in a regular structure containing the space filling curve while neglecting its topological connectivity. Thus, it is not suitable for the application that accesses the mesh elements according to its connectivity.

Lately, Vo et al. proposed another work using space-filling curves. Their method was proposed for the computation of cache-friendly layouts of unstructured geometric data. According to their evaluations, the layouts of their algorithm is competitive with the aforementioned cache-oblivious mesh layouts. Depending on the sorting technique, their algorithm runs either in $O(|V| + |T| \log(|T|))$ or $O(|T| + |T|)$ time. However, most of their experiments are perform on an advanced working environment with large amount of main memory space. The applications under their assessment are mostly in-core algorithms; hence, the extent of improvements on cache coherence is difficult to identify.

4 Our approach to optimal triangle mesh layout

Our approach to triangle mesh layout optimization comprises two parts: namely, a theoretic model considering both the vertex and triangle layouts as well as an example heuristic approach called the *extended cost BFS*, denoted as ECBFS in this context.

4.1 Our theoretic model of mesh layout optimization

Given an indexed-face mesh $M(V, F)$, the layout of M is consisted of the layouts of its vertices V and faces $|F|$. A traditional way to characterize a mesh layout is by the *bandwidths* of its *adjacency matrix* where an optimal layout of vertices can be derived by any method for matrix bandwidth optimization. Since the face layout is left unoptimized and is inherently inconsistent with the derived vertex layout, such model of mesh layout is obviously far from complete.

In this paper, we propose using a matrix of $|V|$ rows and $|F|$ columns instead of traditional $|V| \times |V|$ adjacency matrix to represent a mesh; in which, the row and column vectors, $\mathbf{r}_i$ and $\mathbf{c}_j$, correspond to a vertex ring $R(v_i)$, $v_i \in V$, and a face $f_i \in F$. For a mesh $M(V, F)$, a sparse matrix layout of M is given by $|F|$ column vectors of $|V|$ dimension where the $i-$th column vector, $\mathbf{c}_i$, corresponds to the $i-$th face $f_i \in F$; in which, the non-zero entries of $\vec{c}_i$ indicate the *weights* of corresponding vertices of the face f_i.

As an example, a mesh and its layout are shown in Fig. 5 where all the weights are set to '1'. Since all the faces are triangles, only three entries are set to '1' in each column.

A formal definitions of these terms are given as follows.

Definition 6 (Vertex Layout) The *vertex layout* of a mesh $M(V, F)$, denoted as ϕ, is a bijective labelling function $\phi : V \rightarrow \{1, \cdots, |V|\}$, implies an ordering sequence of the vertices in V where $v_i \in V$ is placed at the $i-$th position of the vertex sequence.

Definition 7 (Face Layout) The *face layout* of the mesh $M(V, F)$, denoted as ψ, is a bijective labelling function $\psi : F \rightarrow \{1, \cdots, |F|\}$, implies an ordering sequence of the faces in F where $f_i \in F$ is placed at the $i-$th position of the sequence of triangles.

Definition 8 (Mesh Layout) Given an indexed triangular mesh $M(V, F)$, the *mesh layout* of M is determined by ϕ and ψ.

To characterize the layout of an indexed mesh, Isenburg et al. proposed the metrics of *span* and *width* respectively for the vertex and triangle layouts. According to their definitions, the *triangle span* of a vertex is the number of triangles between and including the first

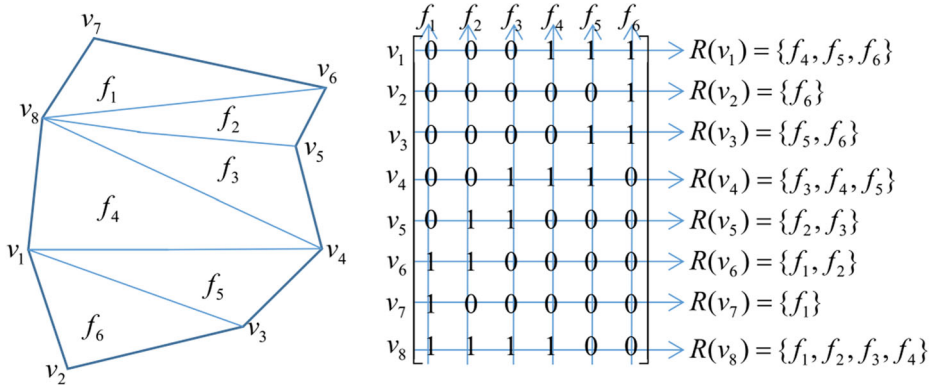


Fig. 5 An example mesh(left) and its layout matrix(right)

and last reference to the vertex; conversely, the *vertex span* of a triangle is the maximal difference of its vertex indices plus one. The *vertex span of a layout* is the maximal span of all triangles. Similarly, the *triangle span of a layout* is the maximal span of all vertices. Note that the *bandwidth* of the adjacency matrix matrix of a mesh is equivalent to the *vertex span* of the mesh layout.

We believe that a joint consideration of the optimality of both layouts regarding to the two metrics with our theoretical model could have better results rather than considering only one of them at once and left the other unoptimized or resort to an additional stage of sorting. Hence, in addition to the bandwidth metric for vertex layout, we adopted a metric similar to the *triangle span* metric for the optimization of the face layout. The formulation of the two metrics are stated as follows.

Definition 9 (Bandwidth of a Triangle) Given a layout $\phi \in \Phi$ of V and a triangle face $f \in F$. The *bandwidth* of f, denoted as $\lambda(\phi, f)$, is determined by

$$\lambda(\phi, f) = \max_{\forall v \in f}\{\phi(v)\} - \min_{\forall v \in f}\{\phi(v)\}. \tag{8}$$

Definition 10 (Bandwidth of a Mesh) Given a layout $\phi \in \Phi$ of V, the *bandwidth* of a mesh $M(V, F)$ is given by

$$B(\phi, F) = \max_{\forall f \in F}\{\lambda(\phi, f)\}. \tag{9}$$

In the example shown in Fig. 5, the bandwidth of the example mesh for the given layout is

$$\max\{|8-6|, |8-5|, |8-4, |8-1|, |4-1, |3-1|\} = 7.$$

The problem of finding optimal vertex layout on the basis of the bandwidth metric is given as follows.

Definition 11 (Optimal Vertex Layout) Given a triangular mesh $M(V, F)$, an optimal vertex layout ϕ_{min} of V with respect to $B(\phi, F)$ is determined by

$$\phi_{min} \in \arg\min_{\phi} B(\phi, F) = \arg\min_{\phi} \max_{\forall f \in F} \lambda(\phi, f), \tag{10}$$

where $B(\phi_{min}, F) = \min B(\phi, F)$.

To provide a coherent formulation, the metrics of face layout optimization are given as follows.

Definition 12 (Span of a Vertex) Given a layout $\psi \in \Psi$ of F and a vertex $v \in V$. The *span* of a vertex v, denoted as $\gamma(\psi, v)$, is determined by

$$\gamma(\psi, v) = \max_{\forall f \in R(v)}\{\psi(f)\} - \min_{\forall f \in R(v)}\{\psi(f)\} + 1, \tag{11}$$

Definition 13 (Span of a Mesh) Given a vertex layout $\psi \in \Psi$ of F, the *span* of a mesh M is given by

$$S(\psi, V) = \max_{\forall v \in V}\{\gamma(\psi, v)\}. \tag{12}$$

In the example shown in Fig. 5, the span of the example mesh for the given layout is

$$\max\{|6-4|+1, |6-6|+1, |6-5|+1, |5-3|+1, \\ |3-2|+1, |2-1|+1, |1-1|+1, |4-1|+1\} = 4.$$

On the basis of preceding definitions, an optimal face layout with respect to the span of mesh is defined as follows.

Definition 14 (Optimal Vertex Layout) Given a mesh $M(V, F)$, an optimal face layout ψ_{min} of F with respect to $S(\psi, V)$ is determined by

$$\psi_{min} \in \arg\min_{\psi} S(\psi, V) = \arg\min_{\psi} \max_{\forall v \in V} \gamma(\psi, v), \tag{13}$$

where $S(\psi_{min}, V) = \min S(\psi, V)$.

By jointly consider the optimization of both the vertex and face layout with respect to the bandwidth, $B(\phi, F)$, and the span, $S(\psi, V)$, of the mesh, an optimal mesh layout is defined as follows.

Definition 15 (The Optimal Mesh Layout) Given a triangular mesh $M(V, F)$ and the labelling functions $\phi \in \Phi$ and $\psi \in \Psi$ for its vertex and face layouts. An optimal mesh layout of M is determined by

$$\phi_{min} \in \arg\min_{\phi} B(\phi, F) = \arg\min_{\phi} \max_{\forall f \in F} \lambda(\phi, f). \tag{14}$$

$$\psi_{min} \in \arg\min_{\psi} S(\psi, V) = \arg\min_{\psi} \max_{\forall v \in V} \lambda(\psi, v). \tag{15}$$

4.2 Our method

To show how the theory works, a BFS-based heuristic method optimizing both layouts is discussed as follows.

To minimize the bandwidth and the span of the mesh at once, the algorithm optimizes the face layout on the fly with the vertex layout. During the process of labelling, the set of vertices V and set of faces F are individually divided into two disjoint sets. In this context, we denote the set of *labeled* vertices as L and the set of *unlabeled* vertices as $\bar{L}$. Likewise, the set of labelled faces are denoted as T while the set of unlabelled faces are denoted as $\bar{T}$.

Instead of using the node degree, we use an different cost function from CM to foresee the branches at next level to assist us determine the ranking of adjacent nodes of current node. The cost function is essentially a gathering operation that accumulates the number of unvisited branches from an unlabelled adjacent nodes where the accumulated value is called the *extending cost* in this paper. The definition of extending cost for an unlabelled node is given below.

Definition 16 (Extending Cost)

$$D(v) = \sum_{\forall u \in C(v) \cap \bar{L}} |C(u)| \tag{16}$$

Initially, such extending cost is evaluated on all vertices.

Algorithm 1 Extending Costs Initialization

Require: V, the set of vertices
Require: F, the set of faces
  for all $v \in V$ **do**
    $d(v) \leftarrow |C(v)|$
    $D(v) \leftarrow 0$
  end for
  for all $v \in V$ **do**
    $D(v) = \sum_{\forall u \in C(v) \cap \bar{L}} d(u)$
  end for

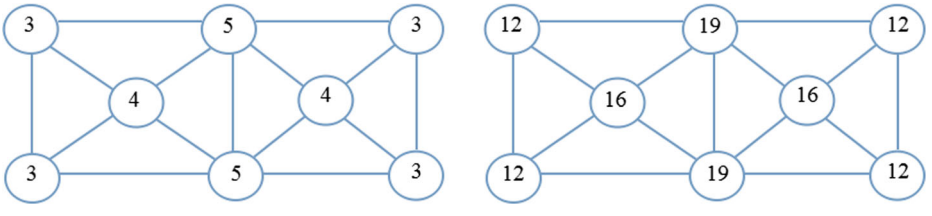
After labelling a vertex v, the extending costs of adjacent unlabelled vertices $u \in C(v) \cap \bar{L}$ are updated simply by subtracting the node degree from their extending cost as follows.

Algorithm 2 Visit and Extending Cost Update

Require: $v, v \in \bar{L}$
  Label v by $L \leftarrow L \cup \{u\}$ and $\bar{L} \leftarrow \bar{L} \setminus \{u\}$
  $\phi(v) = |L|$
  for all $u \in C(v) \cap \bar{L}$ **do**
    $D(u) \leftarrow D(u) - d(v)$
  end for
  for all $f = (a, b, c) \in R(v)$ **do**
    if $v_a, v_b, v_c \in L$ **then**
      Label f by $T \leftarrow T \cup \{f\}$ and $\bar{T} \leftarrow \bar{T} \setminus \{f\}$
    end if
  end for

Figure 6 gpan of the example mesh for the given layoutextending costs of the example graph shown in Fig. 3.

The vertex with minimum extending cost of V is labelled first. Then, the algorithm proceeds iteratively by extending cost update followed by finding and labelling a node with local minimum of extending costs with respect to the neighbourhood of the labelled vertex until all the neighbouring vertices are labelled. If the mesh contains more than one connected components or the boundary closed itself, another vertex with minimum extending cost is selected from the unlabelled group $\bar{L}$. The algorithm is summarized as follows.



node degree initial extending cost

Fig. 6 The node degrees and initial extending costs of an example graph

Algorithm 3 ECBFS

Require: V, the set of vertices
Require: F, the set of faces
Require: ϕ, the original embedding of V
Require: ψ, the original embedding of F
  Extending Costs Initialization
  Find the v with $\min_{v \in V} D(v)$
  Visit and Extending Cost Update v
  $k = 0$
  repeat
    for $i = k$ to $|L|$ **do**
      repeat
        $u = \min_{u \in C(v) \cap \bar{L}} \{D(u)\}$
        Visit and Extending Cost Update u
      until $C(v) \cap \bar{L} = \emptyset$
    end for
    $k = |L|$
    Pick an unlabelled vertex $v \in \bar{L}$
    Visit and Extending Cost Update v
  until $\bar{L} = \emptyset$

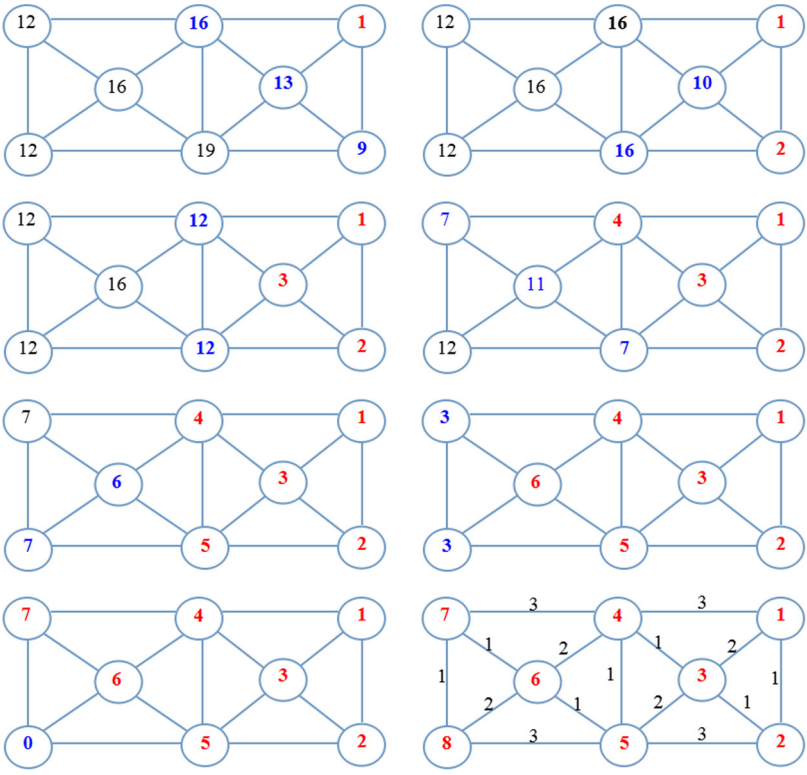


Fig. 7 **a** An example of running our algorithm on the graph shown in Figure

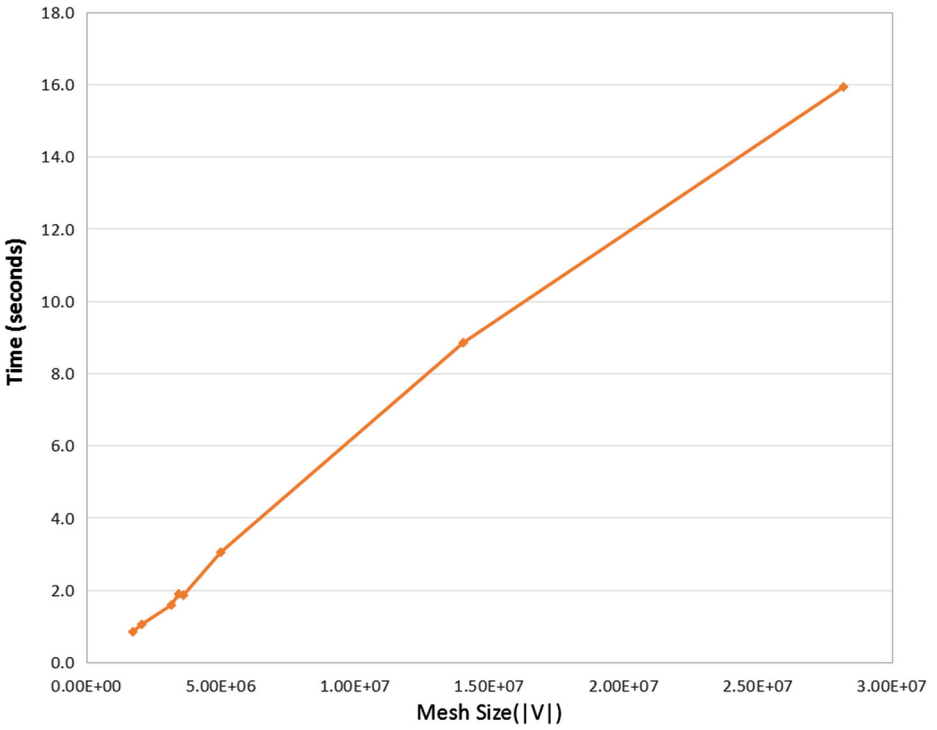


Fig. 8 A plot of the running times of our algorithm versus the mesh size $|V|$ confirms that our algorithm has linear time complexity

The labelling of faces are performed on the fly with the labelling of the vertices. During the labelling of a vertex ring, the faces in the ring is labelled if all its vertices are labelled.

Figure 7 gives an example of performing our algorithm on the graph given in Fig. 3.

Note that, in this example, the graph bandwidth of the labelling determined by our algorithm is 3, which equals the theoretical lower bound of the graph, i.e., $\max_{\forall v \in V} R(v)$.

Similar to the labelling of the vertices, the triangle faces are labelled in accordance to the completion of the labelling of its vertices, which is equivalent to a post-order layout that follows the traversal of vertex rings with no additional effort in sorting the triangle meshes.

Table 1 The test meshes

Mesh	$\|V\|$	$\|F\|$
Youthful	1,728,305	3,411,563
Awakening	2,057,930	4,060,497
Day	3,158,671	6,060,315
Dawn	3,432,236	6,594,103
Dragon(XYZRGB)	3,609,600	7,219,045
Statuette(XYZRGB)	4,999,996	10,000,000
Lucy	14,027,872	28,055,742
David	28,184,526	56,230,343

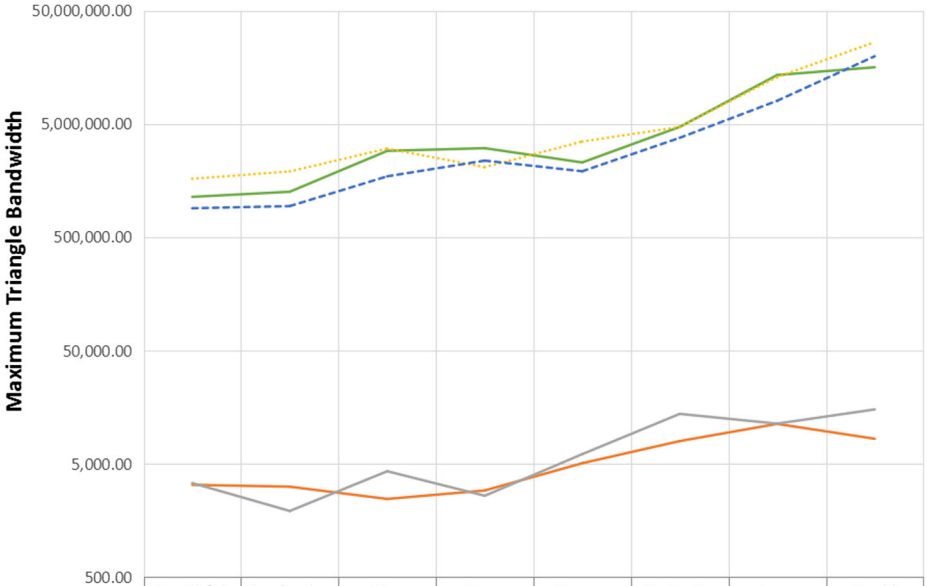
Table 2 The running times in seconds for various layout optimization schemes

Model	ECBFS	CM	SFC(Hilbert)	SFC(Z-Sort)
Youthful	0.8	2.6	5.7	4.3
Awakening	1.0	2.4	6.8	5.0
Day	1.6	12.2	10.3	7.6
Dawn	1.9	15.4	11.2	8.2
Dragon	1.9	3.8	11.8	8.7
Statuette	3.0	5.3	16.4	12.1
Lucy	8.8	15.1	46.2	33.9
David	15.9	59.2	92.4	67.7

4.3 Time complexity analysis

Succeeding to the discussion of our method, we may give an elaborate time complexity analysis over our method. Assuming that the mesh has $|v| = n$ vertices, our time complexity analysis are then performed on the basis of such input size.

Algorithm 1 as the first stage of Algorithm 3 comprises two loops scanning over the entire set of vertices of the input mesh to initialize and compute the extending costs. Since the branch degree of a vertex is a small constant factor, usually six in average, relative to



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	1135694	1260133	2895295	3056377	2285591	4707974	13503666	15821393
ECBFS	3264	3142	2449	2909	5061	7937	11256	8318
CM	3383	1919	4302	2614	6084	13796	11316	15102
SFC(Hilbert)	1640427	1905197	3034363	2077779	3497987	4702806	12946956	26360701
SFC(Z-Sort)	901040	941574	1724628	2373055	1911165	3771238	8006046	19778006

Fig. 9 The maximum bandwidths of the mesh layouts

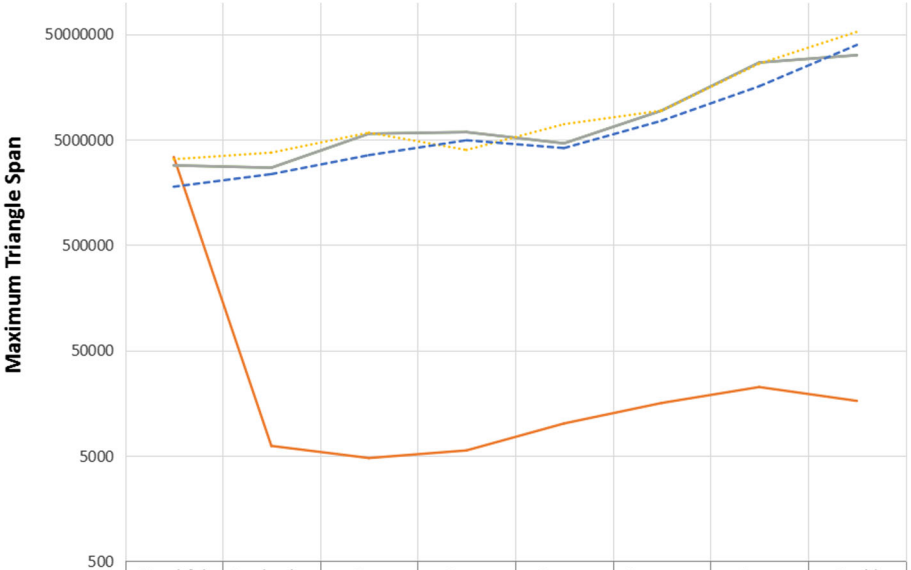
the number of vertices, the two loops takes roughly $\Theta(n)$. For the same reason, the time complexity of the two loops of Algorithm 2 are also constant.

In the third stage of Algorithm 3, an outer loop of REPEAT-UNTIL is performed to ensure the labelling of all the unlabelled vertices while an inner For loop is performed sequentially to visit the unlabelled neighbouring vertices of all the labelled vertices in a FIFO fashion. If the mesh is consisted of only one connected component, such inner For loop is able to label all the vertices of the mesh. Each time, a vertex v is picked from the set of labelled vertices, L, and its unlabelled neighbours are then labelled in the order according to their extending cost. For the size of the set of labelled vertices $L \leq |V| = n$, to visit all the labelled vertices, the algorithm takes approximately n iterations. In each iteration of the inner loop, an unlabelled vertex with minimum extending cost is labelled by scanning through the remaining unlabelled adjacent vertices of v, namely, the set of $C(v) \bigcap L$; hence, the inner iteration still takes a constant number of operations. Hence, the overall time complexity of the third stage is approximately $\Theta(n)$.

By summing the costs of three stages, we may conclude that our method takes

$$\Theta(n) + O(C) + n \times O(C) = \Theta(n) \tag{17}$$

According to the empirical results from our experiments, the running times of our algorithm is linear in proportion to the size of the mesh, which can be verified from the plot shown in Fig. 8.



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	2844329	2701883	5682461	5883602	4603789	9412606	26888674	31568239
ECBFS	3411562	6196	4755	5614	10117	15826	22426	16569
CM	2844329	2701883	5682461	5883602	4603789	9412606	26888674	31568239
SFC(Hilbert)	3252283	3758845	5818083	3982453	6996370	9406954	26199591	52591925
SFC(Z-Sort)	1784380	2350544	3557579	4923144	4151038	7543440	16004508	39471388

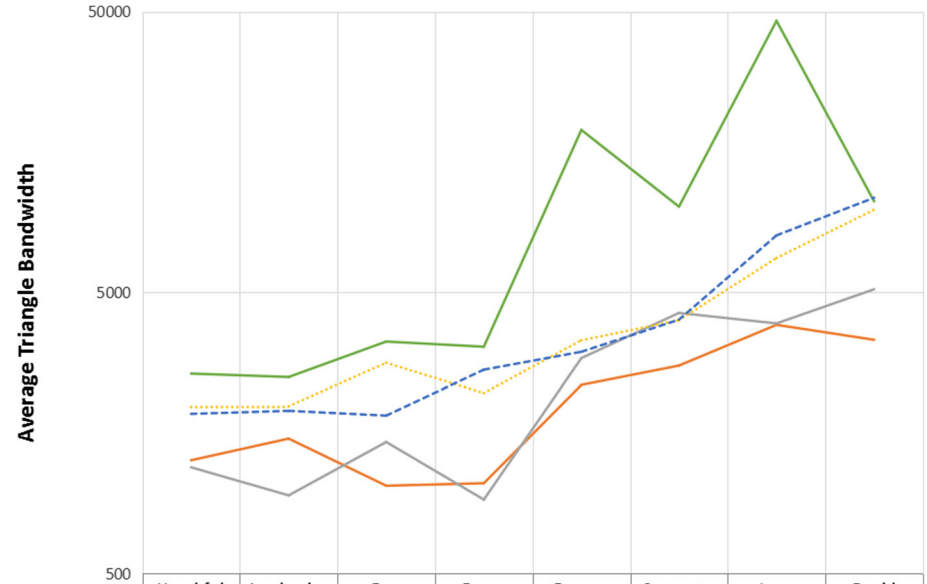
Fig. 10 The maximum spans of the mesh layouts

5 Experimental results

To evaluate our algorithm, the first problem we have confronted with is the test meshes for the experiments. For our purposes, the test meshes must be large enough to reflect the problem of low coherence. For small size of meshes, such problem may never be observed when it is completely accommodated in the main memory. Therefore, we adopted a number of meshes larger than three million triangles from the Stanford Scanning Repository and the archive of the Digital Michelangelo Project. A summary of their sizes are listed in Table 1.

To provide a baseline of comparison, a well-known heuristic technique called Cuthill–McKee (CM) [12] for optimal graph labelling and matrix bandwidth minimization and a state-of-the-art approach based on space-filling curve(SFC) by [48] are evaluated. For a more comprehensive study on the coherence of the optimized mesh layouts, in addition to the common measures of the quality of mesh layouts, the mesh layouts are visualized in four ways to reflect the extent of coherence with respect to both the vertex and triangle face layouts.

Furthermore, as we have already mentioned earlier in the first section, since we mainly focus on evaluating the practical impact to common geometric processing tools, in addition to the theoretical figures, the generated mesh layouts will be further evaluated by a cache supported mesh simplification tool originally designed for large mesh simplification [9]. To prevent from the unavoidable disturbances from the operating system and concurrently executed processes, the runtime efficiency are evaluated not only on the program running



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	2570.99	2500.34	3343.27	3203.94	18948.4	10101.8	46350.3	10551.1
ECBFS	1266.06	1511.98	1027.68	1048.93	2344.77	2746.06	3834.12	3391.56
CM	1195.33	949.495	1470.39	917.491	2919.7	4227.41	3878.35	5130.69
SFC(Hilbert)	1958.48	1971.43	2809.97	2194.58	3376.91	3983.77	6630.61	9846.17
SFC(Z-Sort)	1852	1898.07	1825.29	2655.03	3074.26	3999.98	7981.76	10865.2

Fig. 11 The average bandwidths of the mesh layouts

times of the simplification tool but also on a platform independent factor: the number of I/O read or write counts, i.e., the number of cache misses.

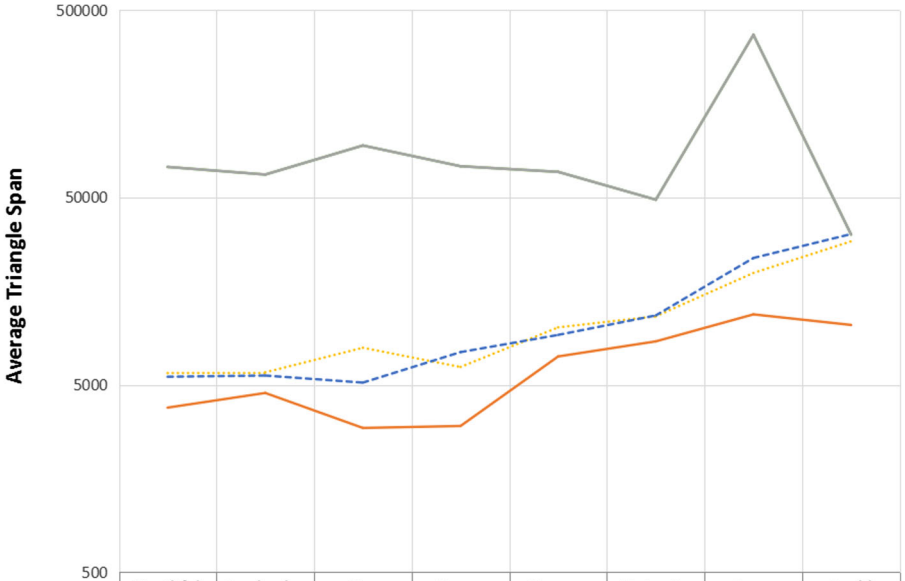
5.1 Runtime efficiency analysis

To overcome the limitation of large memory cost, we have developed two versions of code, i.e., an in-core 64-bit version and an out-of-core version. With 64-bit in-core version, we may extent the restriction of 32-bit address space to 64-bit, which allows us to make a fully use of the available main memory space. For meshes requiring larger main memory space than that is available, advanced virtual memory management supported by concurrent operating systems are utilized. On the other hand, the out-of-core version is provided for 32-bit system. In this paper, we only demonstrate the results of the 64-bit in-core version running on a PC with Intel i7-2600 with 16 GBytes main memory space. The running times of various layout optimization schemes are listed in Table 2.

With a linear time algorithm, we may observed from the Table 2 that our algorithm significantly out performs all the other schemes in terms of layout generation times.

5.2 The bandwidths and spans of the optimized layouts

The evaluation of the mesh layouts in terms of the two fundamental theoretic metrics, i.e., the bandwidths and spans of the mesh. The results are presented in this section with four



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	72839.8	66338.5	94788.2	73461.8	68545.8	48693.8	369213	31728.1
ECBFS	3777.81	4524.96	2942.5	3015.48	7089.27	8533.6	11890.2	10443.9
CM	72839.8	66338.5	94788.2	73461.8	68545.8	48693.8	369213	31728.1
SFC(Hilbert)	5780.32	5832.2	7888.14	6226.53	10135.6	11613.6	19784.5	29179.2
SFC(Z-Sort)	5525.94	5606.35	5139.03	7478.09	9240.49	11739.2	23780.1	31903

Fig. 12 The average spans of the mesh layouts

figures plotted according to their maximum bandwidths and spans respectively in Figs. 9 and 10.

According to the results shown in Fig. 9, the maximum bandwidths of the mesh layouts generated by our method has significantly lower value comparing to the original layout and those generated by the space-filling curves. Comparing to the bandwidths of the resulting

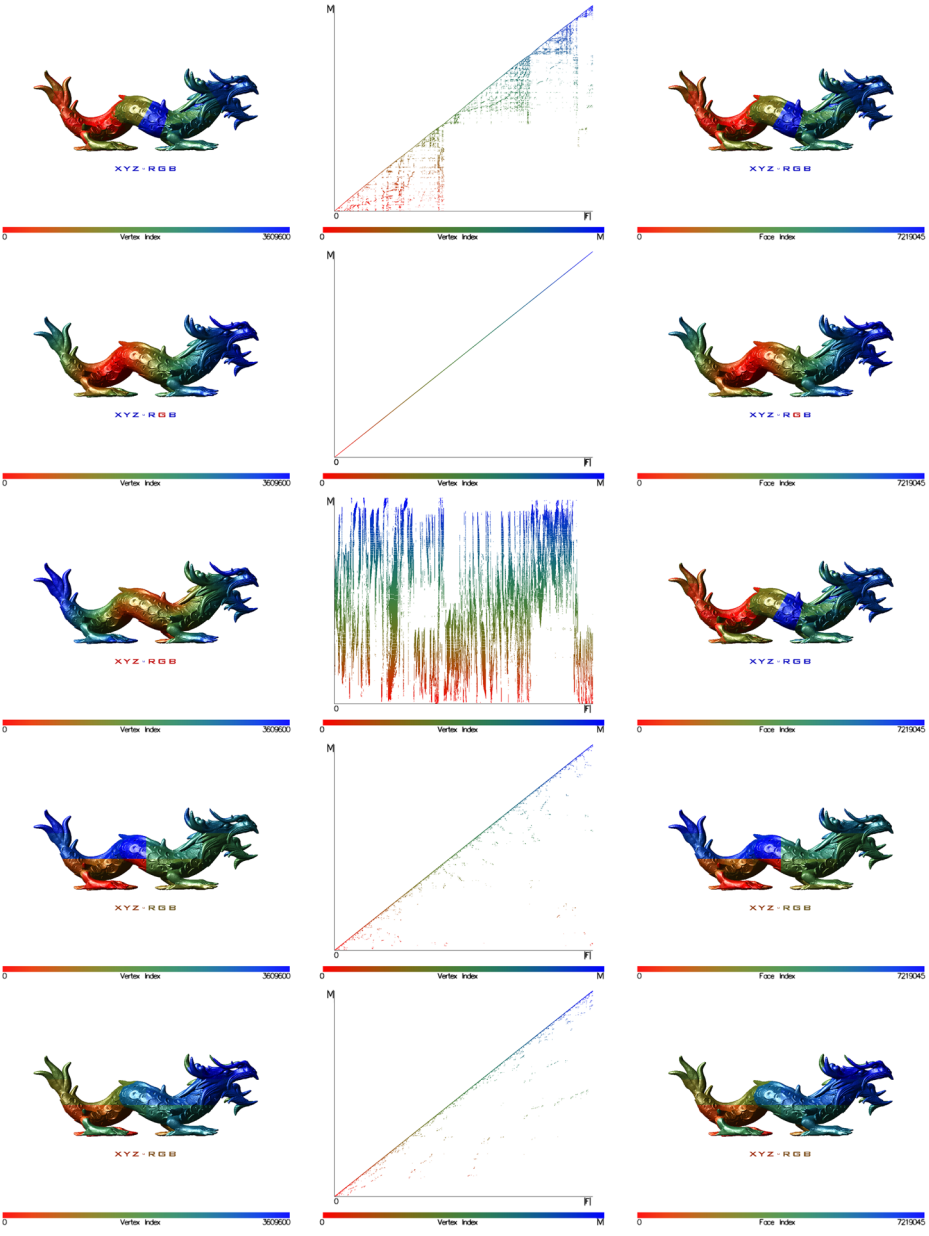


Fig. 13 The visualizations of the layouts of the Dragon meshes: *top row*, before layout optimization; *second to last rows*, after layout optimization by the ECBFS, CM, SFC(Hilbert), and SFC(Z-Sort), respectively

layouts from CM, ours are significantly lower at the cases of Day, Statuette, and David meshes and is better lower than those by CM for the Dragon and Lucy meshes.

On the other hand, the measurements over the maximum spans of the optimized layouts shows that our algorithm obviously out performs the others with significantly lower spans.

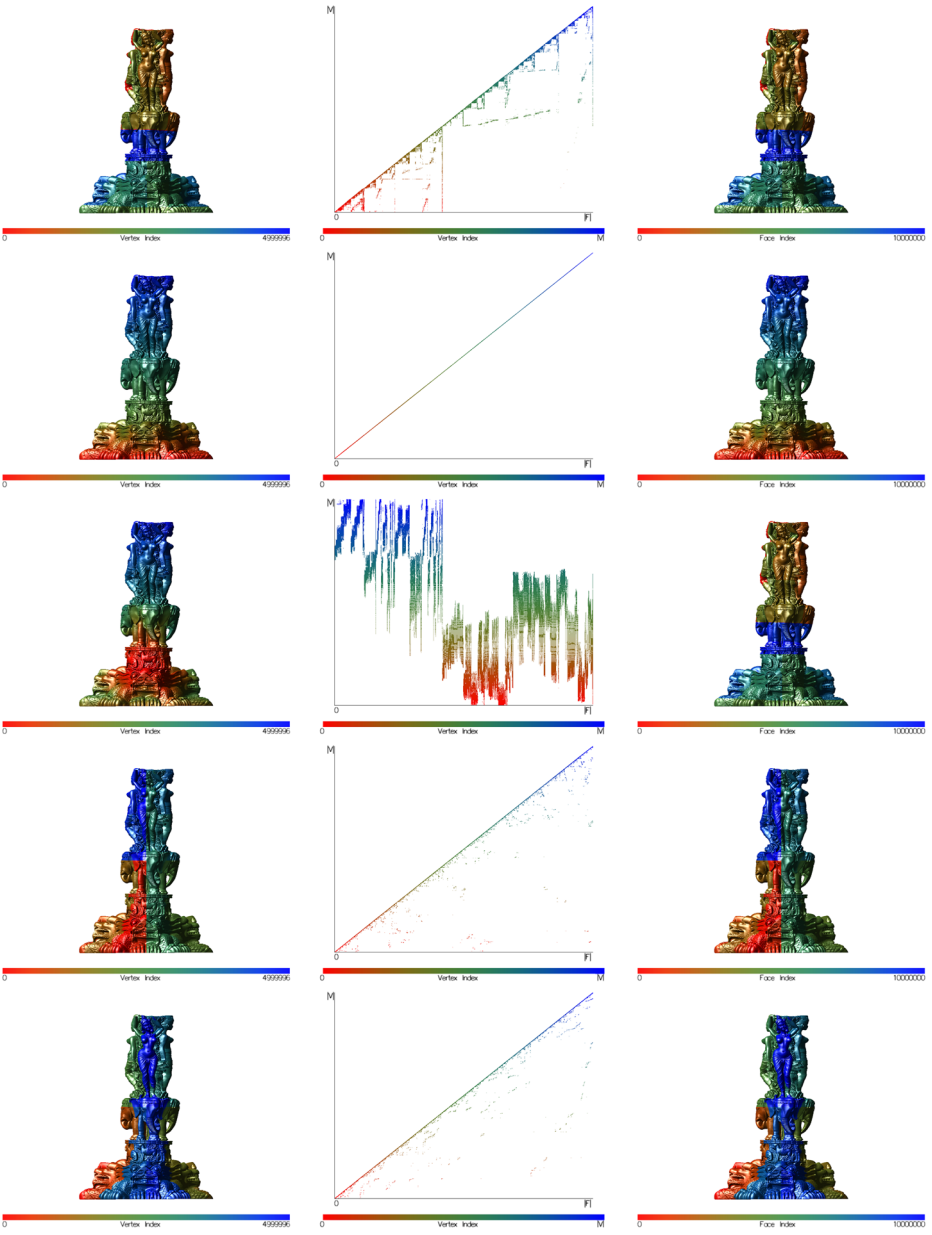


Fig. 14 The visualizations of the layouts of the Statuette meshes: *top row*, before layout optimization; *second to last rows*, after layout optimization by the ECBFS, CM, SFC(Hilbert), and SFC(Z-Sort), respectively

Note that, without follow up treatments for the face layout after the bandwidth minimization, the spans of the resulting layouts by the CM method are the same as the original. Moreover, in comparison with the spectral sequencing result presented in [25], the triangle span of the mesh layouts of the Lucy and David meshes generated by our work are also significantly

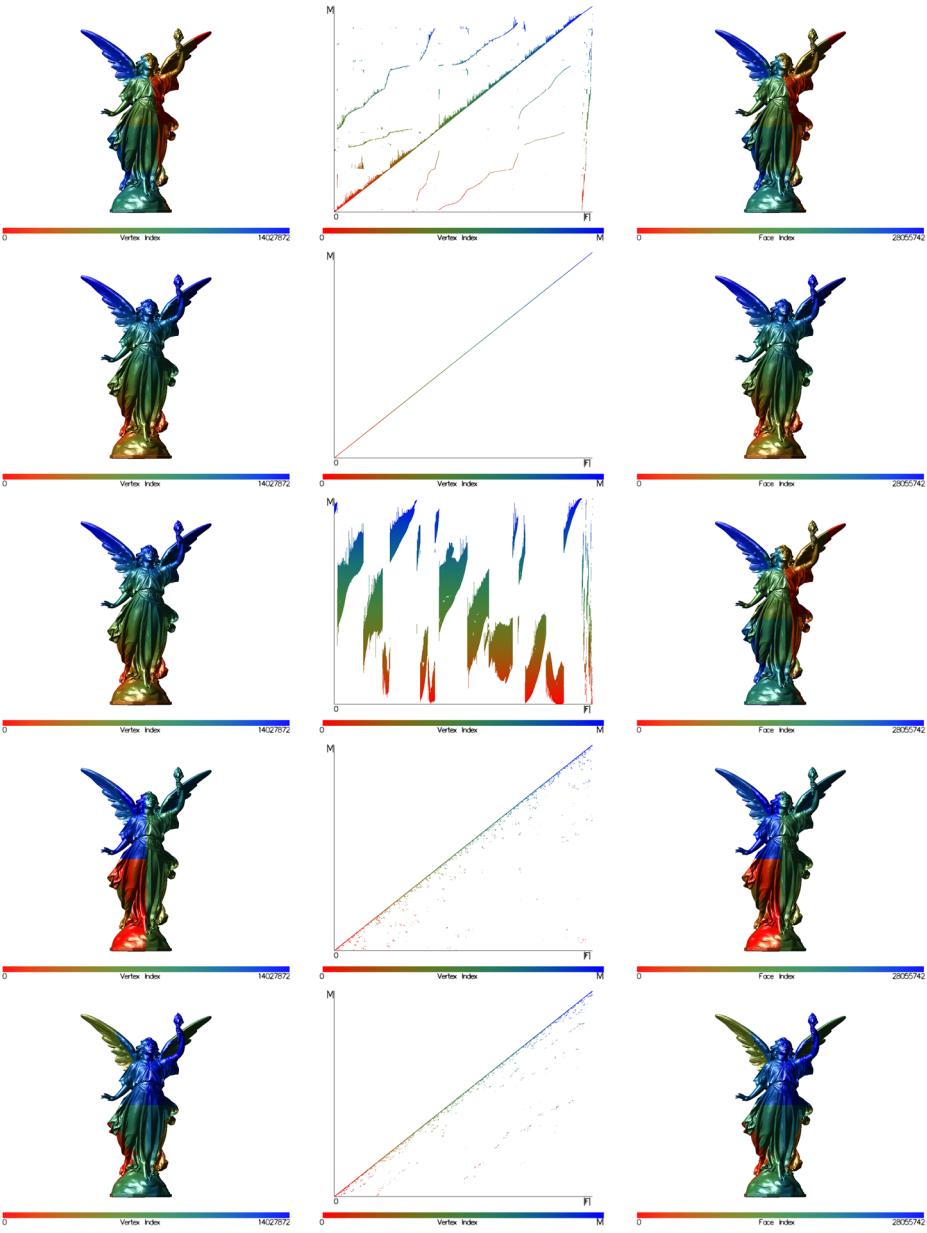


Fig. 15 The visualizations of the layouts of the Lucy meshes: *top row*, before layout optimization; *second to last rows*, after layout optimization by the ECBFS, CM, SFC(Hilbert), and SFC(Z-Sort), respectively

lower than the results of theirs, i.e, 200K and 752K respectively for the Lucy and the David meshes.

In addition to the maximum values, the average values of bandwidths and spans of the optimized layouts are given in Figs. 11 and 12, respectively.

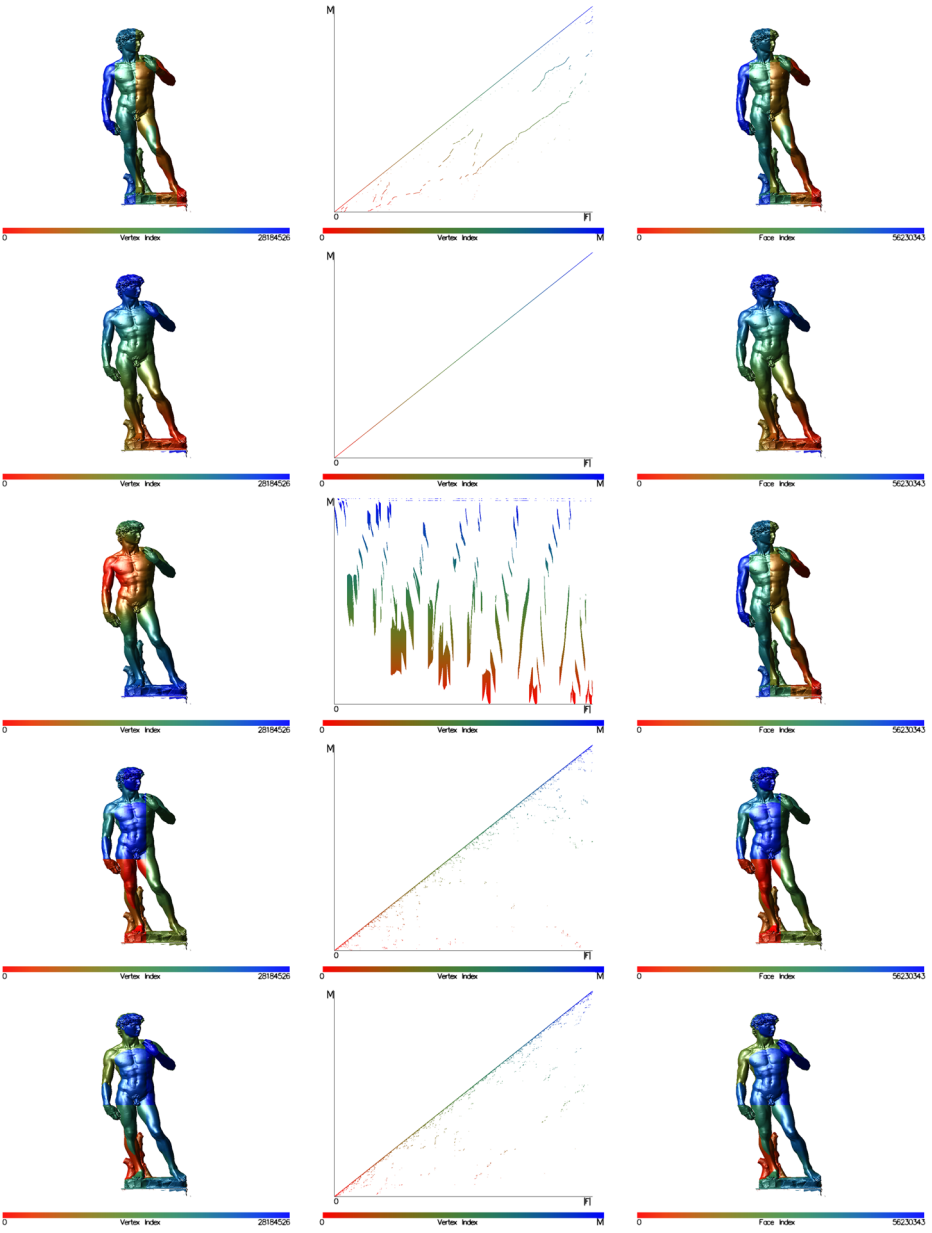


Fig. 16 The visualizations of the layouts of the David meshes: *top row*, before layout optimization; *second to last rows*, after layout optimization by the ECBFS, CM, SFC(Hilbert), and SFC(Z-Sort), respectively

According to the results shown in Fig. 11, the average bandwidths of optimized layouts by our method are lower than those by the other methods for most meshes but are sightly higher than those by CM for the Youth, Awakenning, and Dawn meshes. On the other hand, the results presented in Fig. 12 shows that the average spans of all the optimized layouts from our method are lower than those by all the other methods.

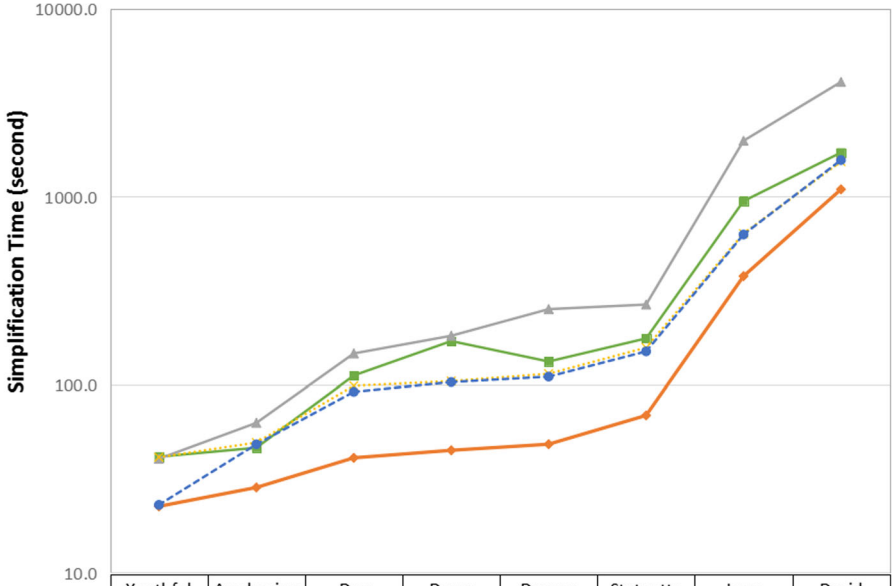
In addition, a point worth noting is that the average case results shown in Figs. 11 and 12 shows that of the space-filling-curve methods are effective in optimizing the mesh layout with respect to average bandwidth and span.

To provide alternative inspection to the quality of the layouts, visualizations of the layouts of the test meshes are given by a plot of the mesh layout and two rendered images coloured with respect to the vertex and triangle layouts in Figs. 13, 14, 15, 16: the middle column shows the plots of mesh layouts, the left and right columns respectively presents the rendered images of the meshes coloured with respect to their vertex and triangle layouts.

Note that, in Figs. 13, 14, 15, 16, the plots of mesh layout are visualized by drawing a dot on the matrix elements of value '1' where its color is assigned according to (1).

5.3 Evaluation by large mesh simplification

For the purpose of the layout optimization is to construct a cache friendly mesh layout in order to avoid cache misses and frequent I/O accesses. Unlike the evaluations performed by



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	41.5	46.3	112.0	170.6	132.8	176.3	948.8	1720.9
ECBFS	22.6	28.5	40.9	44.9	48.4	68.8	378.1	1096.0
CM	40.4	62.8	146.6	182.1	252.4	267.0	1988.5	4090.1
SFC(Hilbert)	41.2	49.3	98.8	104.6	114.3	157.4	637.3	1541.7
SFC(Z-Sort)	23.0	48.3	91.8	103.3	110.4	150.3	629.7	1569.9

Fig. 17 The simplification times for various mesh layouts

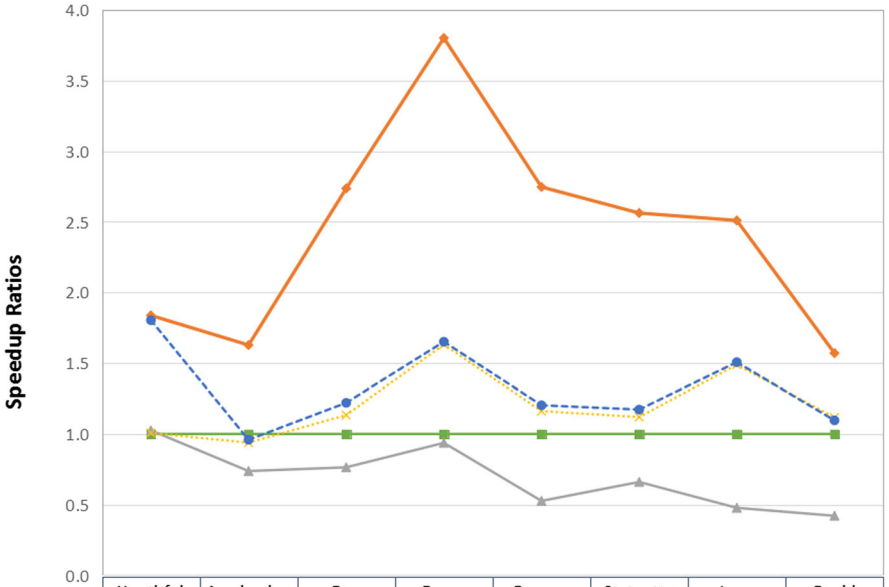
[48] for in-core mesh simplification algorithm(QSlim) [20] with large main memory space, we evaluates the extent of improvements over a cache supported large mesh simplification [9]. To evaluation the performance gain, all tests are performed on a PC with Intel I7-2600K and 16 GBytes main memory running windows 10 using only 64 MBytes cache buffer. The test meshes are reduced from its original size down to 1,000 triangles.

Since the running times depend very much on both the software and hardware platform and may vary radically in a multitasking environment, in addition to the program running times of the mesh simplification algorithm, a set of platform independent results with respect to caching performance are presented as well. Furthermore, to evaluate the performance gain over various sizes of mesh, a coherent metric is required. For this purpose, a commonly used measure called the *speedup ratio* is calculated as follows.

$$Speedup = \frac{\text{Reference Value}}{\text{Test Value}} \tag{18}$$

Figures 17 and 18 respectively shows the simplification times and the *speedup* ratios of the mesh layouts optimized with the ECBFS(our method), the Cuthill-Mckee (CM) [12], the SFC(Hilbert), and SFC(Z-Sort) [48].

It is clear to see that the simplification times of the layouts from our method(ECBFS) are significantly lower than those of the other methods and the speedup ratios are mostly greater than 2.0, which implies that the optimization of the mesh layout using our approach greatly



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ECBFS	1.8	1.6	2.7	3.8	2.7	2.6	2.5	1.6
CM	1.0	0.7	0.8	0.9	0.5	0.7	0.5	0.4
SFC(Hilbert)	1.0	0.9	1.1	1.6	1.2	1.1	1.5	1.1
SFC(Z-Sort)	1.8	1.0	1.2	1.7	1.2	1.2	1.5	1.1

Fig. 18 The speedup ratios related to the simplification times

improves the runtime efficiency of a cache-assited mesh simplification algorithm. In addition, the results of higher running times of CM relative the others indicated the importance of the optimization over the face layout.

On the other hand, the platform independent measures of runtime performance in terms of the number of cache misses and its related speedup ratios are presented in Figs. 19 and 20.

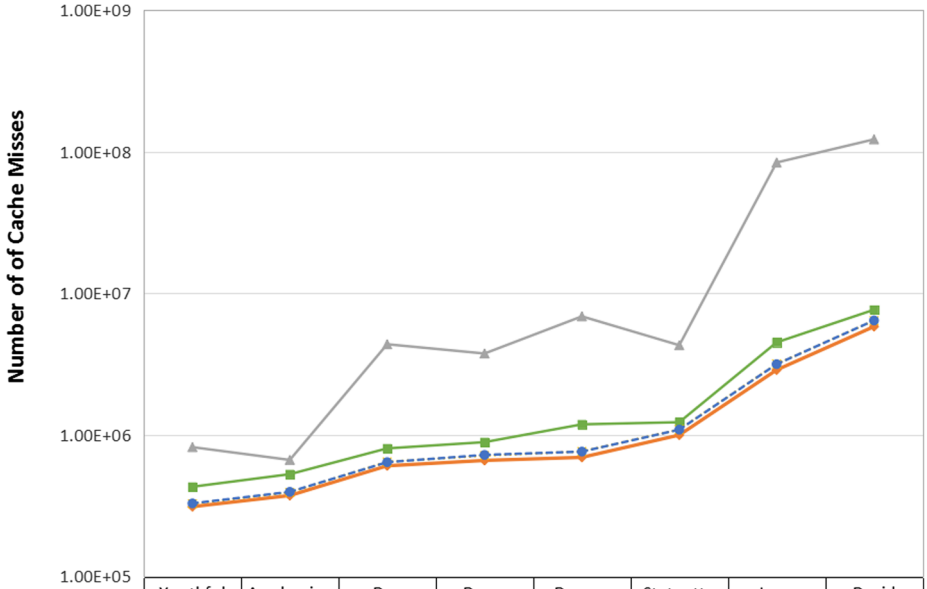
According to Figs. 19 and 20, the mesh layouts generated from our method have relatively fewer cache misses and obviously better referential locality.

Note that, in theory, the simplification time is determined mainly by the CPU as well as internal and external memory reads/writes times. If the amount of CPU and average access times for an internal/external memory operations are T_c, T_i and T_e respectively, the simplification time T_s can be approximated by

$$T_s = T_c + T_i + T_e \tag{19}$$

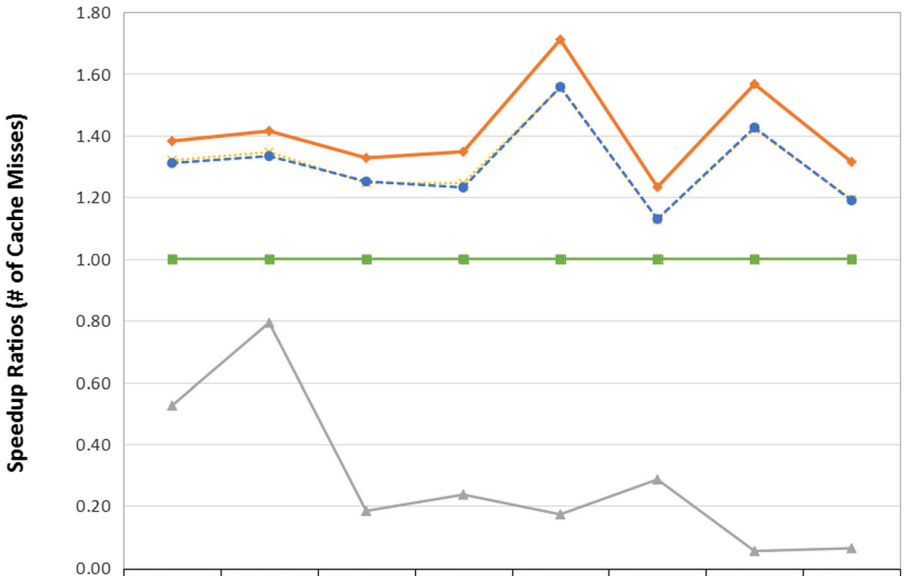
Furthermore, the amount of times for CPU and internal memory accesses are mainly determined by the problem size and the simplification algorithm's time complexity, which are irrelevant to the quality of mesh layout. On the other hand, the amount of times for external memory read or writes resulting from cache misses are directly related to the referential locality of the mesh layouts.

If the speedup ratio related to the number of cache misses is R_c, the amount of time for external memory accesses is reduced from T_e to $\frac{1}{R_c} \times T_e$. The overall simplification time is



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	427572	524566	798285	883929	1180055	1222246	4483527	7621134
ECBFS	309391	370812	601365	656042	689711	991569	2861852	5797795
CM	814487	661370	4347317	3738989	6844629	4277866	83685832	121985671
SFC(Hilbert)	324038	389660	640789	709968	759152	1084930	3154854	6386553
SFC(Z-Sort)	326214	393443	638039	717784	757592	1082424	3144184	6405454

Fig. 19 The number of cache misses during the simplification of various mesh layouts



	Youthful	Awakening	Day	Dawn	Dragon	Statuette	Lucy	David
Original	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ECBFS	1.4	1.4	1.3	1.3	1.7	1.2	1.6	1.3
CM	0.5	0.8	0.2	0.2	0.2	0.3	0.1	0.1
SFC(Hilbert)	1.3	1.3	1.2	1.2	1.6	1.1	1.4	1.2
SFC(Z-Sort)	1.3	1.3	1.3	1.2	1.6	1.1	1.4	1.2

Fig. 20 The speedup ratios related to the number of cache misses

reduced to $T_s' = T_c + T_i + \frac{1}{R_c}T_e$ and the speedup ratio related to the simplification time, R_s, is determined by

$$R_s = \frac{T_s}{T_s'} = \frac{T_c + T_i + T_e}{T_c + T_i + \frac{1}{R_c}T_e} \tag{20}$$

However, the empirical results varied greatly across different CPUs or operating systems.

6 Concluding remarks and future works

In this paper, we have developed a theoretical model and a level-based heuristic approach on the basis of such model to mesh layout optimization. Existing methods either merely consider the optimization of the vertex layout and left the triangle layout either unoptimized or simply resorting to an additional stage of external sort over the triangles afterwards.

However, the dereferencing of the mesh is mainly determined by the triangle ordering. Hence, the mesh layout generated by their method mostly failed to achieve better cache coherency; on the contrary by jointly considering the vertex and triangle layouts and performing the optimization over the vertex and the face layout on the fly with each other, our approach obviously have better results. As indicated by the experimental results, significant improvements are observed comparing with existing approaches with regard to both the theoretical quality measures including the bandwidth and span and the runtime efficiency of large mesh simplification.

Acknowledgments We would like to thank the 3D Scanning Repository and the Digital Michelangelo Project Archive of 3D Models for providing us all the test models. We also give our gratitude to Huy T. Vo et al. for sharing their codes in the public domain so we can have a comparison with their works.

References

1. Ahn M, Guskov I, Lee S (2006) Out-of-core remeshing of large polygonal meshes. IEEE Trans Vis Comput Graph 12(5):1221–1228. doi:10.1109/TVCG.2006.169
2. Bar-Yehuda R, Gotsman C (1996) Time/space tradeoffs for polygon mesh rendering. ACM Trans Graph 15(2):141–152. doi:10.1145/234972.234976
3. Bender MA, Kuszmaul BC, Teng SH, Wang K (2011) Optimal cache-oblivious mesh layouts. Theory Comput Syst 48(2):269–296. doi:10.1007/s00224-009-9242-2
4. Bogomjakov A, Gotsman C (2002) Universal rendering sequences for transparent vertex caching of progressive meshes. Comput Graph Forum 21(2):137–149. doi:10.1111/1467-8659.00573
5. Bolitho M, Kazhdan M, Burns R, Hoppe H (2007) Multilevel streaming for out-of-core surface reconstruction. In: Proceedings of the 5th eurographics symposium on geometry processing, SGP'07. Eurographics Association, Aire-la-Ville, Switzerland, pp 69–78. http://dl.acm.org/citation.cfm?id=1281991.1282001
6. Brodal GS (2004) Cache-oblivious algorithms and data structures. In: Hagerup T, Katajainen J (eds) Proceedings of the algorithm theory - SWAT 2004: 9th Scandinavian workshop on algorithm theory, Humlebæk, Denmark. Springer, Berlin Heidelberg, pp 3–13. doi:10.1007/978-3-540-27810-8_2
7. Cabiddu D, Attene M (2015) Large mesh simplification for distributed environments. Comput Graph 51:81–89. doi:10.1016/j.cag.2015.05.015. International Conference Shape Modeling International
8. Chen HK, Fahn CS, Lin MB (2007) The storage independent polygonal mesh simplification system. In: Proceedings of the 2nd international conference on virtual reality, ICVR'07. Springer-Verlag, Berlin, Heidelberg, pp 3–12. http://dl.acm.org/citation.cfm?id=1770090.1770092
9. Chen HK, Fahn CS, Tsai JJ, Lin MB (2006) A novel cache-based approach to large polygonal mesh simplification. J Inf Sci Eng 22(4):843–861
10. Chow MM (1997) Optimized geometry compression for real-time rendering. In: Proceedings of the 8th conference on visualization '97, VIS'97. IEEE Computer Society Press, NY, USA, p 347. http://dl.acm.org/citation.cfm?id=266989.267103
11. Cignoni P, Montani C, Rocchini C, Scopigno R (2003) External memory management and simplification of huge meshes. IEEE Trans Vis Comput Graph 9(4):525–537. doi:10.1109/TVCG.2003.1260746
12. Cuthill E, McKee J (1969) Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th national conference, ACM '69. ACM, NY, USA, pp 157–172. doi:10.1145/800195.805928
13. Deering M (1995) Geometry compression. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques, SIGGRAPH '95. ACM, USA, pp 13–20. doi:10.1145/218380.218391
14. Demaine ED (2002) Cache-oblivious algorithms and data structures. Lect Notes EEF Summer Sch Massive Data Sets 8(4):1–249
15. Derzapf E, Menzel N, Guthe M (2010) Parallel view-dependent out-of-core progressive meshes. Realismus der Echtzeitgrafik:61
16. Díaz J., Petit J, Serna M (2002) A survey of graph layout problems. ACM Comput Surv 34(3):313–356. doi:10.1145/568522.568523
17. Esposito A, Catalano MSF, Malucelli F, Tarricone L (1998) A new matrix bandwidth reduction algorithm. Oper Res Lett 23(3-5):99–107. doi:10.1016/S0167-6377(98)00040-6
18. Farias R, Silva CT (2001) Out-of-core rendering of large, unstructured grids. IEEE Comput Graph Appl 21(4):42–50
19. Frigo M, Leiserson CE, Prokop H, Ramachandran S (2012) Cache-oblivious algorithms. ACM Trans Algorithm (TALG) 8(1):4
20. Garland M, Heckbert PS (1997) Surface simplification using quadric error metrics. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques, SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., NY, USA, pp 209–216. doi:10.1145/258734.258849
21. Gibbs NE, William G, Poole J, Stockmeyer PK (1976) An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM J Numer Anal 13(2):236–250. doi:10.1137/0713023
22. Hoppe H (1998) Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proceedings of the visualization'98. IEEE, pp 35–42

23. Hoppe H (1999) Optimization of mesh locality for transparent vertex caching. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., NY, USA, pp 269–276. doi:10.1145/311535.311565
24. Isenburg M, Gumhold S (2003) Out-of-core compression for gigantic polygon meshes. ACM Trans Graph 22(3):935–942. doi:10.1145/882262.882366
25. Isenburg M, Lindstrom P (2005) Streaming meshes. In: visualization, 2005. VIS 05. IEEE, pp 231–238. doi:10.1109/VISUAL.2005.1532800
26. Isenburg M, Lindstrom P, Gumhold S, Snoeyink J (2003) Large mesh simplification using processing sequences. In: Proceedings of the 14th IEEE Visualization 2003 (VIS'03), VIS '03. IEEE Computer Society, DC, USA, p 61. doi:10.1109/VISUAL.2003.1250408
27. Isenburg M, Lindstrom P, Snoeyink J (2005) Streaming compression of triangle meshes. In: ACM SIGGRAPH 2005 Sketches. ACM, p 136
28. Koller D, Trimble J, Najbjerg T, Gelfand N, Levoy M (2006) Fragments of the city: Stanford's digital forma urbis romae project. J Roman Archaeol Suppl Ser 19(61):237–252
29. Li L, Schemenauer N, Peng X, Zeng Y, Gu P (2002) A reverse engineering system for rapid manufacturing of complex objects. Rob Comput Integr Manuf 18(1):53–67. doi:10.1016/S0736-5845(01)00026-6
30. Lillesand T, Kiefer RW, Chipman J (2014) Remote sensing and image interpretation. Wiley
31. Lin G, Yu TPY (2006) An improved vertex caching scheme for 3d mesh rendering. IEEE Trans Vis Comput Graph 12(4):640–648. doi:10.1109/TVCG.2006.59
32. Lindstrom P (2000) Out-of-core simplification of large polygonal models. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques. ACM Press, Addison-Wesley Publishing Co, pp 259–262
33. Lindstrom P, Pascucci V (2001) Visualization of large terrains made easy. In: Proceedings of the conference on visualization '01, VIS'01. IEEE Computer Society, DC, USA, pp 363–371. http://dl.acm.org/citation.cfm?id=601671.601729
34. Lindstrom P, Pascucci V (2002) Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. IEEE Trans Vis Comput Graph 8(3):239–254
35. Liu R, Zhang H, van Kaick O (2006) Spectral sequencing based on graph distance. In: Proceedings of the 4th international conference on geometric modeling and processing, GMP'06. Springer-Verlag, Berlin, Heidelberg, pp 630–636. doi:10.1007/11802914_50
36. Miller GL, Teng SH, Thurston W, Vavasis SA (1998) Geometric separators for finite-element meshes. SIAM J Sci Comput 19(2):364–386. doi:10.1137/S1064827594262613
37. Park JS, Chung MS, Hwang SB, Shin BS, Park HS (2006) Visible Korean human: its techniques and applications. Clin Anat 19(3):216–224
38. Pascucci V, Frank RJ (2001) Global static indexing for real-time exploration of very large regular grids. In: Proceedings of the 2001 ACM/IEEE conference on supercomputing, SC '01. ACM, NY, USA, p 2, doi:10.1145/582034.582036
39. Petit J (2013) Addenda to the survey of layout problems. Bullet EATCS 3(105)
40. Sagan H (1994) Space-filling curves. Universitext. Springer, New York. http://cds.cern.ch/record/1609380
41. Sajadi B, Jiang S, Gopi M, Heo JP, Yoon SE (2011) Data management for ssds for large-scale interactive graphics applications. In: Symposium on interactive 3D graphics and games, I3D '11. ACM, NY, USA, pp 175–182. doi:10.1145/1944745.1944775
42. Sander PV, Nehab D, Chlamtac E, Hoppe H (2008) Efficient traversal of mesh edges using adjacency primitives. In: ACM SIGGRAPH Asia 2008 Papers, SIGGRAPH Asia '08. ACM, NY, USA, pp 144:1–144:9. doi:10.1145/1457515.1409097
43. Spitzer V, Ackerman MJ, Scherzinger AL, Whitlock D (1996) The visible human male: a technical report. J Am Med Inform Assoc 3(2):118–130
44. Spitzer VM (2015) The visible human: A graphical interface for holistic modeling and simulation. The Digital Patient: Advancing Healthcare, Research, and Education p 51

45. Stanco F, Battiato S, Gallo G (2011) Digital imaging for cultural heritage preservation: Analysis, Restoration, and reconstruction of ancient artworks, 1st edn. CRC Press, Inc., FL, USA
46. Tchiboukdjian M, Danjean V, Raffin B (2010) Binary mesh partitioning for cache-efficient visualization. IEEE Trans Vis Comput Graph 16(5):815–828. doi:10.1109/TVCG.2010.19
47. Varadhan G, Manocha D (2002) Out-of-core rendering of massive geometric environments. In: Visualization, 2002. VIS 2002. IEEE, pp 69–76
48. Vo HT, Silva CT, Scheidegger LF, Pascucci V (2012) Simple and efficient mesh layout with space-filling curves. J Graph Tools 16(1):25–39. doi:10.1080/2151237X.2012.641828
49. Wang C, Xu C, Lisser A (2014) Bandwidth minimization problem. In: MOSIM 2014, 10ème conférence francophone de modélisation, optimisation et simulation, Nancy, France. https://hal.archives-ouvertes.fr/hal-01166658 Colloque avec actes et comité de lecture. Internationale
50. Wu J, Kobbelt L (2003) A stream algorithm for the decimation of massive meshes. In: Graphics interface, vol 3, pp 185–192
51. Yoon SE, Lindstrom P (2006) Mesh layouts for block-based caches. IEEE Trans Vis Comput Graph 12(5):1213–1220. doi:10.1109/TVCG.2006.162
52. Yoon SE, Lindstrom P, Pascucci V, Manocha D (2005) Cache-oblivious mesh layouts. ACM Trans Graph 24(3):886–893. doi:10.1145/1073204.1073278
53. Yotov K, Roeder T, Pingali K, Gunnels J, Gustavson F (2007) An experimental comparison of cache-oblivious and cache-conscious programs. In: Proceedings of the 19th annual ACM symposium on parallel algorithms and architectures. ACM, pp 93–104
54. Zhang S, Jie B, Tan L (2013) The application of chinese visible human dataset. FASEB J 27(1 Supplement):lb8–lb8
55. Zhang SX, Heng PA, Liu ZJ (2006) Chinese visible human project. Clin Anat 19(3):204–215

Hung-Kuang Chen received his Ph.D. degrees in computer science of electronic engineering from National Taiwan University of Science and Technology, Taipei, Taiwan, in 1995 and 2006. From August 1995 to July 2002, he served as a Lecturer at the Department of Electronic Engineering of Lung-Hwa University of Science and Technology, Taoyuan, Taiwan. From August 2002 to January 2007, he was a faculty of the Department of Information and Design of Asia University. Since February 2007, he served as an associate professor in the electronic engineering department of the National Chin-Yi University of Technology, Taichung, Taiwan. His research interests cover the computer graphics, virtual reality, and parallel computing.