Computers & Graphics 34 (2010) 424–429

Contents lists available at ScienceDirect

Computers & Graphics

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

Technical Section

Triangulation of CAD data for visualization using a compact array-based triangle data structure

Sang Wook Yang, Young Choi *

Department of Mechanical Engineering, Chung-Ang University, 221 Heukseok-dong, Dongjak-gu, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Received 21 May 2009
Received in revised form
22 January 2010
Accepted 11 February 2010

Keywords:
Mobile device
Triangulation
Visualization
Mobile 3-D
Indexed triangles

ABSTRACT

We present a triangulation method for visualization of computer-aided design (CAD) model data. The proposed method has been enhanced from a previously devised sequential triangulation method that used indexed array triangle representation. The improvement is achieved by an enhancement of the searching algorithm for array-based data structure.

The proposed triangulation is a sequential triangulation method with an optimized *k*d-tree in permutation vector form to accelerate the insertion of surface points. The permutation vector is formed by rearranging the sequence of vertex array elements comprising indexed triangle data, and the space partition information of the *k*d-tree is represented as a vertex sequence only.

The proposed triangulation method was designed for mobile devices with inadequate memory size and CPU speed compared to desktop computers. We considered both efficiency and compactness in our implementation. Topological operators are defined for querying and searching information within indexed triangle data. Experimental results empirically show that the triangulation method is of $O(n)$ time complexity, and is bounded by $O(n\log n)$ for worst-case data.

1. Introduction

Numerous studies of three-dimensional (3-D) visualization for mobile devices and graphic utilities such as OpenGL|ES and DirectX Mobile have made it easier to render 3-D models in the mobile environment. Tessellated 3-D shapes can be rendered easily on mobile devices using OpenGL|ES and DirectX Mobile; thus, various graphics applications such as 3-D games have been developed with mobile graphic engines. However, since triangulation is comparatively expensive in terms of computation, these mobile 3-D graphics applications have dealt only with pre-tessellated geometric data.

Studies of 3-D graphic visualization for mobile devices can be categorized as remote rendering architecture [1–6], visualization of tessellated data [7–9], and progressive mesh-based methods [10–12]. In remote rendering architecture, the rendered images or video streams are generated by a server and the generated scenes are transferred to mobile devices. Progressive mesh-based methods are used to reduce the size of tessellated model data. Recently, studies have been conducted to accelerate the rendering performance on mobile devices from both the hardware and software points of view [13]. Most of the previous studies of 3-D visualization on mobile devices utilize pre-tessellated models. Complex 3-D models such as 3-D CAD data are converted to approximated mesh models or images on the server side, and are transferred to the mobile devices. Then, the transferred data are simply rendered in the mobile devices. This is why previous studies focused on mobile 3-D concentrated on data conversion, transfer, and rendering efficiency.

Since the visualization of parametric surfaces with a boundary is much more complicated than the rendering of a pre-tessellated model, it is difficult to visualize 3-D CAD models directly on mobile devices. If a triangulation process that is efficient in memory usage and calculation time is used, the 3-D CAD model can be directly handled in a mobile device without preprocessing.

A 3-D model created using a solid modeler or graphic authoring software is generally represented by parametric surfaces with boundaries. Tessellation of parametric surfaces is inevitable in rendering and numerical analysis of a 3-D model; thus, it has been widely studied and applied to various fields including computer graphics, CAD/CAM, and computational analysis. From the point of view of computer graphics and CAE, approximately regular polygons are required to achieve precise analysis results and high quality rendering results. Many studies on the generation of regular meshes have been reported [14], as well as mesh refinement algorithms that calculate fine meshes from coarse meshes with respect to angles and aspect ratios of triangles [15]. However, fewer triangles are required for visualization of CAD data than for well-shaped triangulation, since the

* Corresponding author. Tel.: +82 2 820 5312; fax: +82 2 817 5101.
E-mail address: yychoi@cau.ac.kr (Y. Choi).

performance of application programs and network transfer are considered in practical use. Thus, keeping a small number of triangles while guaranteeing reasonable accuracy of the approximation is important in the visualization of a parametric surface model. Sadoyan et al. [16] proposed a uniform triangulation algorithm focused on the CAD/CAM domain by combining lattice-based triangulation [17] and Delaunay triangulation. The algorithm divides the parametric domain into rectangular cells so that the bounds of each cell form a simple polygon that triangulates each cell. The accuracy of the triangulated mesh can be adjusted with the density of the lattice.

Yang et al. [18] showed that the entire visualization process, including the triangulation of B-rep CAD data, can be executed on mobile devices with reasonable computational efficiency with the support of a new compact data structure. For efficient visualization of CAD data on mobile devices, the properties of 3-D CAD data were used as the input for the triangulation scheme. In a CAD model, a surface generally consists of a surface geometry and boundary edges. Since the boundary edges are ordered with their directions in a loop, they are sequentially traversed for computation. To reflect the curvatures of free-form surfaces such as NURBS or Bezier surfaces, it is necessary to insert points on the boundary loops and on the surfaces during the triangulation process.

We present a triangulation method, which is an improved version of our previous work [18]. The improvement is based on separation of the construction of the space search tree from the sequential triangulation process. Additionally, we defined operations to manipulate a simple indexed triangular data structure and to formulate our improved triangulation method. Section 2 describes the operations for searching adjacency information and modifying topologies with a simple data structure called constraints embedded triangles (CETs). Section 3 contains our proposal for a sequential triangulation process based on a permutation vector for a space searching tree. Section 4 presents the results of an experiment using the proposed triangulation process. Section 5 gives our conclusions.

2. Array-based data structure for triangulation

CET is a simple modification of the *interworld* data structure that describes Voronoi diagrams of balls in 3-D space [19]. The CET data structure represents both triangle adjacency and constraint information that describes a surface boundary, and was proposed as an efficient description of constrained triangulation in a two-dimensional (2-D) parametric domain. CET consists of two arrays: a vertex array and an indexed triangle array. The CET data structure is briefly described in Section 2.1. In addition, the operations for the retrieval, traversal, and modification of entities are briefly explained, and these will be used in the discussion of our algorithm.

2.1. CET data structure

A triangulation in 2-D space can be represented with an array of n vertices $\boldsymbol{V}$={v_0, $v_1, \ldots, v_{n-1}$} and an array of m triangles $\boldsymbol{T}$={t_0, $t_1, \ldots, t_{m-1}$}. We can also represent a constrained triangulation with these two arrays by denoting a triangle in $\boldsymbol{T}$ as t=(M_0, M_1, M_2, $N_0, N_1, N_2, f_0, f_1, f_2$) where M_i represents an array index of a vertex in $\boldsymbol{V}$, N_i represents an array index of a triangle in $\boldsymbol{T}$, and f_i describes the constraint flag of the edge between t and the triangle referenced by N_i. The vertices referenced by M_0, M_1, and M_2 occur in counter-clockwise order. The triangle referenced by N_i is a neighbor to triangle t, and is not connected to the vertex referenced by M_i. If a vertex is represented as v=(p, L) where p represents the coordinates of the vertex and L is an array index of a triangle connected to v, we can traverse the triangle adjacency from a vertex. The traversal operations are described in Section 2.2. The flag value f_i is an enumeration of {0, 1, 2, 3}. The value 0 implies that the edge is not a constraint edge. The value 1 implies that the indicated edge of t is a constraint edge and that its direction in the context of the triangle is the same as the direction of the edge in the context of the boundary loop, as shown in Fig. 1(a). The value 2 implies that the indicated edge of t is a constraint edge and that its direction in the context of the triangle is opposite to the direction of the edge in the context of the boundary loop, as shown in Fig. 1(b). Fig. 1(c) shows that the constraint edge has two directions; this is represented by a flag value of 3. Since each flag can be expressed with two bits, only a single byte is needed to represent the four different flags.

2.2. Operations for CET data

Guibas and Stolfi [20] presented a quad-edge data structure and introduced a divide-and-conquer algorithm for computing a Voronoi diagram in terms of primitives for the manipulation. A description using primitives and operations provides a hierarchical and structured description of algorithms; thus, we also explain our algorithm in terms of operators related to the data structure.

The operators described in Table 1 are categorized into two types. The operators in the ‘Retrieval’ category retrieve topological information directly from the arrays in the CET data structure, and the operators in the ‘Modification’ category modify adjacency information in the array data. Since all of the operators are easily defined from the relationship between the vertex array and the triangle array described in Section 2.1, a detailed explanation is omitted. However, we emphasize the following: there is no explicit edge data in the CET data structure, but an edge can be described as a pair consisting of a triangle and an index to the adjacent triangle that shares the edge. The *edge* operator retrieves edge information and the *flag* operator retrieves edge flag information that represents a constraint edge direction.

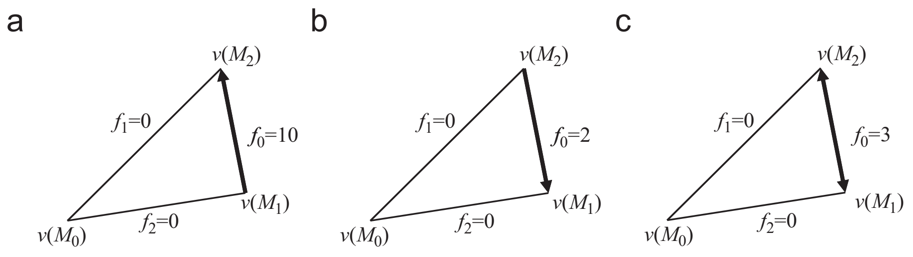

Fig. 1. Examples of constraint edge flag values.