Available online at www.sciencedirect.com
ScienceDirect
Computers & Graphics 31 (2007) 100–118
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Technical Section

Comparison of triangle strips algorithms ☆

Petr Vaněček*, Ivana Kolingerová

Department of Computer Science and Engineering, Center of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic

Abstract

Triangle surface models belong to the most popular type of geometric objects description in computer graphics. Therefore, the problem of fast visualization of this type of data is often solved. One popular approach is stripification, i.e., a conversion of a triangulated object surface into strips of triangles. This enables a reduction of the rendering time by reducing the data size which avoids redundant lighting and transformation computations.

The problem of finding an optimal decomposition of triangle surface models to a set of strips is NP-hard and there exist a lot of different heuristic stripification techniques. This paper should help to orient in the jungle of stripification algorithms. We present an overview of existing stripification methods and detailed description of several important stripification methods for fully triangulated meshes. As different authors usually use different data sets and different architectures, it is nearly impossible to compare the quality of stripification methods. For this reason we also present a set of tests of these methods to give the reader a better possibility to compare these methods.

Keywords: Computer graphics; Triangle; Triangle strip; Dual graph

1. Introduction

Triangle surface models (also called *triangle meshes*) are very often used in various applications (such as CAD, GIS, medical research, computer games) because they are easy to create, good to manipulate and edit and they have very good hardware support for visualization. In many of these applications, the rendering speed is crucial, thus the problem of fast visualization of this type of data is often being solved.

The performance of today's rendering hardware is usually very high and the speed of the rendering is bounded not only by the power of the GPU but also by the rate at which the triangulated data is sent into the GPU. To decrease the amount of data, one can use some techniques to prevent sending of unnecessary triangles, e.g., visibility culling, or some kind of simplification of complex objects, e.g., (C)LOD—continuous level of detail. Still it is important to reduce the time needed to transmit the set of triangles by compressing the topological information and decompressing at the rendering stage. As neighboring triangles share an edge, it is possible to avoid sending the common vertices twice by a special order of triangles, called a *triangle strip* or *tristrip*.

A *sequential tristrip* is a sequence of $n+2$ vertices that represents n triangles: in Fig. 1 the sequence (1,2,3,4,5,6) represents triangles △123, △234, △345, and △456. Using the sequential strip, the transmit cost of n triangles can be reduced by the factor of three (from $3 \cdot n$ to $n+2$ vertices).

There also exists a situation where the triangle adjacency does not allow a sequential encoding. In Fig. 2 the sequence (1,2,3,4,5,6) produces an invalid triangle △456. An extra vertex has to be added to change the sequence to (1,2,3,4,3,5,6). This operation is called *swap* and tristrips with swaps are called *generalized tristrips*. Still, the transmit cost is reduced more than twice, from $3 \cdot n$ to $n+2+\#swaps$ vertices.

To increase the rendering speed, it is necessary to minimize both the number of swaps and the number of

☆This work was supported by Ministry of Education of The Czech Republic—project MSM 235200005.

*Corresponding author.
E-mail addresses: pvanecek@kiv.zcu.cz (P. Vaněček), kolinger@kiv.zcu.cz (I. Kolingerová).
URLs: http://herakles.zcu.cz/~pvanecek, http://iason.zcu.cz/~kolinger.

0097-8493/$ - see front matter © 2006 Published by Elsevier Ltd.
doi:10.1016/j.cag.2006.10.003

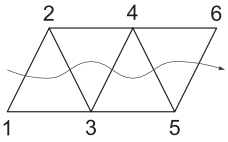

Fig. 1. A sequential strip.

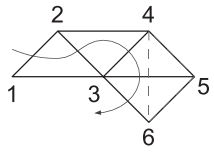

Fig. 2. A generalized strip.

triangle strips. As the triangle strips can potentially reduce the amount of data transmission, transformation and lighting computations, many graphics libraries support it, e.g., IRIX-GL, OpenGL, and Direct3D.

Evans et al., showed that covering the mesh by an optimal set of triangle strips is NP-hard [1]. To compute a stripification in a polynomial time, it is necessary to use some heuristic that finds some local optimum (usually in linear time).

As the number of triangles in meshes grow as fast as the power of GPUs and the bus bandwidths, the stripification topic is still very important and many algorithms on stripification exist. In this paper, we present an overview of existing methods and their comparison.

In Section 2 a short overview of existing stripification methods is presented. The fundamentals of some of the most important methods are described in Section 3. A comparison of these methods is discussed in Section 4. Some conclusions are given in Section 5.

2. State of the art

In this section we present a short description of existing algorithms. As the number of stripification algorithms is quite high, first we propose several ways of classification.

2.1. Classification

Triangle stripification algorithms can be categorized in several different ways. Here we enumerate five classifications that are often used:

(1) According to the type of input data (isolated vertices, triangles, etc.).
(2) According to the type of meshes (static meshes, CLOD, etc.).
(3) According to the type of optimization (minimization of number of strips, minimization of number of vertices).
(4) According to the type of heuristic function (local heuristic, global heuristic).
(5) According to the hardware support (optimization for vertex caches).

One of the possible classifications is based on the type of input data. The first category of algorithms takes only the geometrical information, i.e., only the vertices, as an input [2,3]. Typically, these algorithms work only with data sets on a plane or with a height field. The second category takes triangles of the model and tries to build triangle strips, not necessarily a single strip, without changes in topology [4–8]. The third category is more general as it takes polygons that are triangulated with respect to the stripification [9–12]. The last category takes either triangles or polygons and inserts some extra vertices (Steiner points) to achieve a single triangle strip [2,13,14]. In this paper, we will focus on category two and three according to this classification.

The majority of stripification algorithms are designed for static meshes, i.e., meshes without changes in topology. As the complexity of some industrial models are very high, the need for visualization of view-dependent meshes is growing. There are two approaches to use triangle strips in LOD meshes. First, special stripification methods that produce a stripification with some properties [6,13,15] and second, special data structures and algorithms that can manage the strips during the view-dependent visualization [16–19]. In this work, we use only static meshes for our comparison.

Furthermore, the term 'optimal stripification' is not uniquely determined. One can optimize the stripification algorithm to produce a low number of vertices needed for strips to decrease the amount of data sent through the bus to the rendering engine and speed up the rendering. As the initialization of a new triangle strip costs some extra time, it is also desirable to minimize the number of generated triangle strips [6,8,20]. It is not possible to minimize both these parameters at once—decreasing the number of triangle strips often leads to increase in the number of vertices (due to higher number of swaps, needed to preserve the strip) and vice versa. Very often, the stripification algorithms contain more heuristic functions for vertex or strip optimization. In our comparison we use both type of heuristic functions if possible, to show the influence of vertex/strip trade off on the rendering speed.

We can also classify the stripification algorithms according to the type of the heuristic function. Very often, the heuristic function only decides in which direction the strip should continue. For such a decision only some local criterion is sufficient. To obtain a better stripification, some global heuristic is necessary [9,6,14,21].

Todays GPUs contain large vertex caches and their use can significantly reduce the bandwidth. This criterion was taken into account and several algorithms that respects the vertex cache were developed [22,23].

In the next subsection we describe most of the published stripification algorithms classified according to the type of input data. As the number of stripification algorithms is quite high, the list of algorithms is probably not complete.