Stripe: A Software Tool For Efficient Triangle Strips

Francine Evans   Steven Skiena   Amitabh Varshney

Abstract

The speed at which triangulated surfaces can be displayed is crucial to interactive visualization and is bounded by the rate at which triangulated data can be sent to the graphics subsystem for rendering. Partitioning polygonal models into triangle strips can significantly reduce rendering times over transmitting each triangle individually. We outline the design of our software tool, Stripe, for constructing triangle strips from partially triangulated models, and present experimental results showing these strips are 10–30% better than those from previous best known public-domain codes.

1 Introduction

The speed of rendering triangular meshes in computer graphics is limited by the rate at which triangulation data is sent to the graphics processor. Each triangle can be specified by three vertices, but to maximize the use of the available data bandwidth, it is desirable to order the triangles so that consecutive triangles share an edge. Using such an ordering, only the incremental change of one vertex per triangle need be specified. In this paper, we consider the problem of constructing good triangle strips from partially triangulated polygonal models. Our software tool, Stripe, exploits the freedom to triangulate these faces to produce strips that are $10-30\%$ better than those of previous best known public-domain codes.

To allow greater freedom in the creation of triangle strips, a "swap" command permits one to alter the FIFO queuing discipline in a triangle strip [5]. A swap command swaps the order of the two latest vertices in the buffer so that the instead of vertex i replacing the vertex $(i-2)$, vertex i replaces the vertex $(i-1)$. Although the swap command is supported in the GL graphics library [5], it is not supported in OpenGL [3, 4] for portability reasons [2]. In this paper, we evaluate our software for the more realistic OpenGL cost model where every swap costs one vertex.

2 Constructing Triangle Strips

The best previous public-domain code for constructing triangle strips which we are aware of is from SGI [1], and works only on triangulated models. The SGI algorithm begins its construction of a triangle strip by starting from an arbitrary triangle with the lowest adjacency count, which is the number of triangles that share an edge with it, and extends the strip in both directions as far as possible without overlapping any previously constructed strips. The algorithm seeks to create triangle strips that tend to minimize leaving isolated triangles. There is no reluctance to generate swaps, and understandably so, since this algorithm was aimed at generating triangle strips for Iris GL.

We first partition the model into regions that have collections of $m \times n$ quadrilaterals arranged in m rows and n columns, which we refer to as a *patch*. Each patch whose number of quadrilaterals, mn, is greater than a specified cutoff, is converted into one strip, at a cost of three swaps per turn. Further, every such strip is extended backwards from the starting quadrilateral and forwards from the ending quadrilateral of the patch to the extent possible. For extension, we use an algorithm similar to the SGI algorithm. However, we triangulate our faces "on the fly", which gives us more freedom in producing triangle strips.

3 Experimental Results and Conclusions

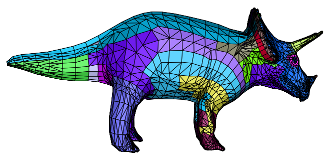
Our approach of constructing triangle strips that we have outlined above is the best out of 20 different heuristics that we have exhaustively tested on several datasets and is the one that we have selected as the method of choice in our tool, Stripe. Table 1 compares the results for our tool, Stripe, versus the SGI algorithm [1]. The cost columns show the total number of vertices required to represent the dataset in a generalized triangle strip representation under the OpenGL cost model (where each swap costs one vertex). We observe that our results are just 10% more than the theoretical lower bound of the number of triangles + 2, so there is only limited potential for better algorithms.

As can be seen from the results of Table 1, under the OpenGL cost model, we are able to outperform the SGI's public-domain code significantly. Although the SGI algorithm does have a slightly better running time, we do not believe this to be a serious drawback of our approach since the triangle-strip generation phase is typically done off-line before interactive visualization. Keeping this in mind we have not yet done any serious performance tuning of our code and there is some scope for further improving our runtimes.

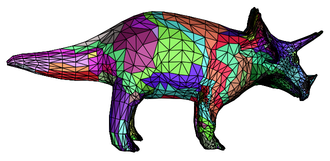
A visual comparison of the triangle strips produced by Stripe versus the SGI algorithm appears in Figure 1 where every triangle strip is colored by a different color and constant shading is used. Our software is in public-domain and more information about its release is available at: http://www.cs.sunysb.edu/~evans.

Data File	Verts	Tris	Cost		% Savings
			SGI	Stripe	
plane	1508	2992	4005	3388	15%
skyscraper	2022	3692	5621	4731	16%
triceratops	2832	5660	8267	5981	28%
power lines	4091	8966	12147	10268	15%
porsche	5247	10425	14227	11065	22%
honda	7106	13594	16599	14780	11%
bell ranger	7105	14168	19941	15011	25%
dodge	8477	16646	20561	17963	13%
general	11361	22262	31652	25912	18%

Table 1: Comparison of triangle strip algorithms.



(a) Stripe Output



(b) SGI algorithm output

Figure 1: Comparison of triangle strip outputs

Acknowledgements

We would like to acknowledge several valuable discussions with Joe Mitchell, Martin Held, Estie Arkin, Jarek Rossignac, Josh Mittleman, and Jim Helman. The datasets that we have used are from Viewpoint DataLabs. Francine Evans is supported in part by a NSF Graduate Fellowship and a Northrop Grumman Fellowship. Steven Skiena is supported by ONR award 400x116yip01. Amitabh Varshney is supported in part by NSF Career Award CCR-9502239.

References

[1] K. Akeley, P. Haeberli, and D. Burns. tomesh.c : C Program on SGI Developer's Toolbox CD, 1990.

[2] J. Helman. Personal Communication.

[3] Open GL Architecture Review Board. *OpenGL Reference Manual.* Addison-Wesley Publishing Company, Reading, MA, 1993.

[4] Open GL Architecture Review Board, J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide.* Addison-Wesley Publishing Company, Reading, MA, 1993.

[5] Silicon Graphics, Inc. Graphics Library Programming Guide, 1991.

Authors' address : evans|skiena|varshney@cs.sunysb.edu
Department of Computer Science, SUNY Stony Brook, NY 11794-4400