Vol. 6, No. 1: 1–6

Linear Grouping—A Method for Optimizing 3D Vertex Transformation and Clipping

Jonathan Dinerstein
Sorenson Media, Inc.

Larre Egbert and Nick Flann
Utah State University

Abstract. This paper introduces *linear gouping*, a slightly lossy optimization technique for affine transformation of rigid vertex data. This technique is beneficial when vertex transformation is of notable cost, such as in software rendering systems and applications and where visual correctness is sufficient, for example games, simulators, and other real-time applications. In practice, linear grouping (nearly visually lossless) eliminates approximately 55% of the vertices in an average polygonal mesh, reducing transform overhead by about 50% and clipping overhead by about 32%.

1. Introduction

The polygon mesh is popular in computer graphics due to ease of rendering. However, mesh representation is plagued with one particular problem: geometry processing overhead. As stated by Hoppe [Hoppe 98], in many situations "geometry processing proves to be the bottleneck", not the rasterization of polygons. It is because of this geometry-processing bottleneck that most real-time applications, such as simulation and entertainment, use sparse meshes composed of only a few large polygons. A major portion of geometry processing is vertex transformation and clipping.

Approaches to reducing transformation overhead include simplifying the mathematics [Thompson 90], level of detail [Heckbert, Garland 94],[Hoppe

© A K Peters, Ltd.

1086-7651/01 $0.50 per page

98], SIMD (such as Intel SSE), and hardware implementations. While effective in reducing computational requirements, these methods are limited because they do not eliminate any *required* transformations. Further, the standard approach to clipping requires all vertices to be transformed, regardless of whether they will be discarded.

2. Linear Groups

Surprisingly, for most polygon meshes, it is possible to group many vertices into sets of three or more that are (nearly) colinear. Each of these sets can be represented by keeping two vertices, $V1$ and $V2$, to define the line, and replacing each other vertex Q by its parameter t [O'Rourke 98]:

$$Q(t) = t * (V2 - V1) + V1. \tag{1}$$

Of course, it is rare for three vertices to be exactly colinear. We define a threshold ϵ, the maximum distance between a line and the vertices associated with it. There is a quality/time tradeoff based on this value. A good metric for the value of ϵ is the size of the polygon model being considered. We have found that $0.5\% \leq \epsilon \leq 1.5\%$ of the object coordinate space is effective (see Figures 1–2). Equations for computing point-to-line distance and projection are presented on the web site given at the end of this paper.

Linear grouping has two main steps: (1) constructing the linear groups, and (2) transforming and reconstructing the vertices from the groups. Step 1 is performed only once, at either creation or initialization; we describe our algorithm in Section 3. Step 2 is performed on a frame-by-frame basis, as discussed below.

To transform all points on the line, we first transform $V1$ and $V2$, computing

$$\begin{aligned} V1' &= M * V1, \\ V2' &= M * V2, \\ L' &= V2' - V1, \end{aligned} \tag{2}$$

at the cost of two homogeneous matrix multiplications plus a vector subtraction. The remaining points can be transformed by

$$Q'(t) = t * L' + V1' \tag{3}$$

at a cost of one scalar-vector multiplication and one vector-vector addition — significantly less than a matrix multiplication.

Clipping is computed by transforming only the defining vertices to normalized projection space. The line-segment defined by these vertices is clipped

against a canonical view volume, using an algorithm such as the Cohen-Sutherland clipper [Foley et al. 95]. If the line segment is trivially accepted or rejected, it is known that all grouped vertices must also be accepted/rejected. This not only allows many vertices to be clipped for free, but also does not require tranformation of all vertices in order to clip. If the line segment cannot be clipped trivially, then the grouped vertices must be clipped exhaustively.[1]

3. Constructing Linear Groups

We want to find a linear grouping (set of lines) that eliminates as many vertices as possible; finding a good (or best) grouping is an optimization problem. We use a branch-and-bound algorithm, performing a recursive search of possible groupings in best-first order. The best-first order is computed before searching, building a table of all possible vertex pairs, storing the number of vertices that fall within ϵ of the line — a pair that groups more vertices is considered better than one that groups fewer.

```
find_grouping(G: groups that have been constructed
    D: vertices used to define groups
    P: vertices that have been grouped
    F: vertices still free)

for each unordered pair of vertices (V1,V2) in D or F
in order of best-to-worst-quality
    consider line L = (V1, V2)

    designate candidate group Gc

    construct set C of all vertices in F that
    meet bounds and that lie within ε of L

    if candidate group Gc does not meet bounds then reject,
    else
        find_grouping(G + Gc, D + V1 + V2, P + C, F − C − V1 − V2)
if no candidate groups could be constructed
    if G has best quality so far
        BestSoFar ← G
```

[1] One might consider finding the parameters where the line intersects the view volume, and using these to clip grouped vertices. However, in our experience, this is usually just as expensive as clipping the grouped vertices exhaustively.

Without the use of bounds, this $O(n * n!)$ algorithm would be too costly to be practical. Note that by searching best-first, the bounds can eliminate searches that would not produce a higher quality solution. We use two *true bounds* that do not affect the quality of the result:

- *No nonproductive groups.* Do not accept a candidate group if C is empty.
- *Cannot be best solution.* Do not accept a candidate group if $|F + P| < |BestSoFar.P|$ (where $|s|$ is the cardinality of the set s).

We also use two *heuristic bounds* that dramatically speed up the algorithm but might impact quality:

- *Average group size.* Do not accept a candidate group if it is predicted that not enough vertices will be grouped as the partial solution is completed; $|F + P|$ is compared against the average group size in G. This reduces the number of vertices grouped by 1% on average.
- *Bounding box.* Only consider vertices for grouping on line L that are within a bounding box defined by $V1$ and $V2 \pm \epsilon$. This reduces the number of vertices grouped by 3% on average. Clipping, as presented in this paper, requires this bound to reduce groups from infinite lines to line segments.

Due to the best-first ordering, the algorithm tends to find good solutions early. To further speed operation, the search can be terminated after a predetermined amount of time or number of solutions. In fact, experimentation has shown that the optimal solution is usually found initially, so we can often accept the first solution. If there are simply too many vertices, they should be partitioned and linearly grouped separately.

3.1. Results

An experiment was performed in which the optimization algorithm described in this paper was implemented. The test material included a broad variety of 3D meshes. Using an ϵ of 0.5% of the object coordinate space, 55% of all vertices in an average mesh were eliminated. The average linear group contained about 1.5 grouped vertices. Execution time was only seconds for meshes below three hundred vertices. Minutes were required for meshes up to seven hundred vertices. Through octtree partitioning, data sets of several thousand vertices were linearly grouped in minutes. This experiment was performed on a 333 MHz processor.

Most error from linear grouping is at locations of high detail (Figures 1–2). This error can be greatly reduced (or visually eliminated) by adapting ϵ to the local topology of the model. This usually has very little impact on the total number of vertices grouped.

Figure 1. Cat statue model, rendered with a 3D marble texture. The original model is on the left, and the linear grouped model (with $\epsilon = 0.5\%$ of the object coordinate space) is on the right. As can be seen, there is very little difference. In fact, without the 3D texture, they are virtually indistinguishable. Linear grouping eliminated over 55% of the vertices.

4. Discussion

The primary weakness of linear grouping is its lossy behavior. However, since the method is meant for real-time or interactive rendering, this is acceptable. Also, linear grouping has only been applied to vertex transformation; normal transformation is not benefited. Further, only linear transformations can be performed.

Linear grouping is applicable for implementation in current rendering systems and hardware, as its implementation is simple and fits naturally into most pipelines. However, the overall benefit of linear grouping is inherently limited by how large an overhead vertex transformation and clipping are. Therefore, this method is most beneficial to software rendering systems and/or applications where vertex transformation is of notable cost.

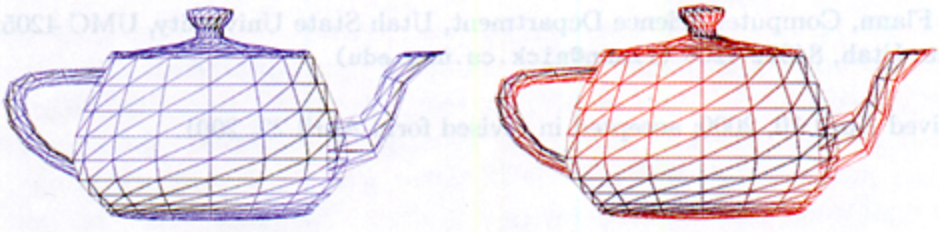

Figure 2. The original Utah teapot in blue is shown on the left. On the right, the red teapot using linear grouping with $\epsilon = 0.5\%$. Areas of very fine detail (such as the lid handle) are most prone to error.

We have primarily tested using object models. However, we expect that linear grouping could also work well for terrain fields. Most terrain fields naturally have X and Y coordinates in common, being on a grid. Therefore, finding three or more near-colinear vertices should be very common. Terrain rendering is also more tolerant to error than shapes like curved surfaces.

References

[Foley et al. 95] [1] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice.* Second edition. Reading, MA: Addison-Wesley. 1995.

[Heckbert, Garland 94] P. S. Heckbert, and M. Garland. "Multiresolution Modeling for Fast Rendering. In *Graphics Interface 94*, pp. 43–50.

[Hoppe 98] H. Hoppe. "Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering." In *IEEE Visualization 1998*, pp. 35–42.

[O'Rourke 98] J. ORourke. *Computational Geometry In C*, Second edition. Cambridge, UK: Cambridge University Press, 1998.

[Thompson 90] Kelvin Thompson. "Fast Matrix Multiplication." In *Graphics Gems*, edited Andrew Glassner, pp. 460–461, Cambridge, MA: Academic Press, 1990.

Web information:

Source code and further algorithm details and results are available at `http://www.acm.org/jgt/papers/DinersteinEgbertFlann01/`

Jonathan Dinerstein, Sorenson Media, Inc., 570 Research Parkway, Suite 102, North Logan, UT 84341 (`jon@sorenson.com`)

Larre Egbert, Computer Science Department, Utah State University, UMC 4205, Logan, Utah, 84322-4205 (`larre@cs.usu.edu`)

Nick Flann, Computer Science Department, Utah State University, UMC 4205, Logan, Utah, 84322-4205 (`flann@nick.cs.usu.edu`)

Received April 10, 2000; accepted in revised form April 22, 2001.