Geometric Optimization

Paul Hinker
Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545
hinker@acl.lanl.gov

Charles Hansen
Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545
hansen@acl.lanl.gov

Abstract

An algorithm is presented which describes an application independent method for reducing the number of polygonal primitives required to faithfully represent an object. Reducing polygon count without a corresponding reduction in object detail is important for: achieving interactive frame rates in scientific visualization, reducing mass storage requirements, and facilitating the transmission of large, multi-timestep geometric data sets. This paper shows how coplanar and nearly coplanar polygons can be merged into larger complex polygons and re-triangulated into fewer simple polygons than originally required. The notable contributions of this paper are: 1) a method for quickly grouping polygons into nearly coplanar sets, 2) a fast approach for merging coplanar polygon sets and, 3) a simple, robust triangulation method for polygons created by 1 and 2. The central idea of the algorithm is the notion of treating polygonal data as a collection of segments and removing redundant segments to quickly form polygon hulls which represent the merged coplanar sets.

1 Introduction

This paper describes how an optimized object[1] can be generated from the initial polygonal description of the object. The idea of reducing the polygon count of a geometric object has received increased attention in current computer graphics and image processing research. Achieving this goal benefits not only object rendering, but has direct impact on storage and communication costs as well. It is these benefits which supplied the initial motivation for the work presented in this paper. Another benefit which appeared during the course of this work was the discovery that the algorithm is highly parallel in nature and maps well to both Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) parallel architectures. This last issue will be further discussed in the future work section of this paper.

[1]Herein the terms *geometry optimization*,and *object optimization* are used to describe the process of reducing the number of polygonal primitives required to faithfully represent an object

Due to current commercial hardware and software offerings, the polygon remains the most used form of object description. Since the polygon is a planar primitive, highly detailed, complex objects can only be faithfully represented by thousands or even millions of polygons. Unfortunately, even state-of-the-art visualization systems are overwhelmed by these numbers since rendering speed, mass storage and memory requirements are directly proportional to the sum of polygons in the objects which make up a scene. These facts motivate the search for an efficient, automatic method of object optimization

Polygonal models exhibit a wide variety of characteristics depending on the type of real or simulated object they represent. This implies that no single, general purpose method of optimization is best suited for all types of objects. The characteristics of a three dimensional hydrodynamics data set differ widely from those found in medical Magnetic Resonance Imaging (MRI) data and it would be unreasonable to expect one method to produce optimum results for all cases. The geometric optimization method described in this paper performs best on geometries made up of large objects. For instance, iso-surfaces generated from three dimensional hydrodynamics simulations and terrain data. Geometries consisting of many, small surfaces; such as turbulence simulation data, do not contain large groups of coplanar primitives and, therefore, do not yield significant reduction.

A short section follows describing previous work published in the related areas of polygon decimation[7], surface re-tiling[8]and hierarchical object representation[1, 5]. The remainder of the paper describes the optimization algorithm and lists results obtained by its use. The paper concludes with a discussion of the results and a description of the anticipated direction that future work may take.

2 Related Work

Clark presents the benefits of hierarchical methods in using more than one representation of a model for geometric rendering [1]. Typically this class of methods is used to guarantee frame rate performance in applications such as flight simulators. These methods can be somewhat arbitrary when reducing polygon count because creating a reduced surface by decreasing grid resolution can easily cause small detail to be lost. Ning has developed another hierarchical method which rigorously quantifies error bounds and creates an 'ideal' surface by interpolation using a sinc function.[5].

Turk describes a surface re-tiling method which redistributes fewer vertices over an existing surface and positions them by repulsion [8]. The new vertices are then connected forming a surface made up of fewer graphical primitives. This method uses a local greedy algorithm to handle such difficulties as: points which are neighbors in a three dimensional sense but widely separated on the surface; surface bubbles created by two sets of polygons which tile the same area; and polygonal groups which meet at a sharp corner. Because new vertices are used in the creation of the final surface, the original shading information is unusable and must be recalculated or the reduced surface must be rendered with a flat shading model.

Schroeder et al. have developed a method, known as polygon decimation, which characterizes the local topology and geometry for a given vertex and determines if the vertex is a potential candidate for deletion[7]. If a vertex meets the deletion criterion, retriangulation occurs on the remaining vertices in the immediate region. This method also identifies sharp edges and sharp corners that must be retained to ensure that the resulting geometry closely resembles the original data. An important feature of this method allows the user to specify the size of the resulting geometry.

3 Geometric Optimization

The primary goal of any polygon reduction algorithm is to reduce the number of primitives required to faithfully represent an object. Any of the methods described in the related work section perform this function adequately but experience has shown there are several other considerations. First, it is useful to retain, as opposed to recalculating, auxiliary vertex information such as normals, texture coordinates and gradients. Vertex removal methods like the one described by Schroeder meet this requirement. Second, it is imperative that the algorithm perform the reduction quickly due to the large number of polygons involved. The geometric optimization algorithm described by this paper displays an $O(n \log n)$ time complexity, where n is the number of original primitives in a near coplanar set. Finally, the largest data sets are currently being produced on parallel and massively parallel computer hardware so the most benefit can be gained by applying the polygon reduction on the massively parallel processor before the polygonal data is moved. The algorithm presented by this paper will run well on both SIMD and MIMD parallel architectures.

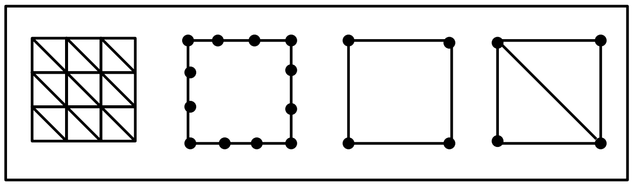

Figure 1: Basic Polygonal Flow

3.1 Overview

The geometric optimization algorithm performs the polygon reduction by first, grouping all the polygons into nearly coplanar sets. A segment list is created for each of these sets and the list is sorted. Next, redundant segments are removed from the list. This results in a segment list which contains only those segments that form the boundary of a polygon. These polygons are constructed and then re-triangulated to form the reduced object [See Figure 1].

The four main steps in the algorithm are:

- Construct nearly-coplanar sets
- Create segment list and remove duplicate segments
- Retrace polygons
- Triangulate resultant polygons

3.2 Creating Near-Coplanar Sets

Three methods of planar set construction were analyzed during algorithm development. The first exhibits linear time complexity, with respect to the number of polygons in the surface, and involves a bucket sort of the polygons on their respective normals. A unit sphere describes the sum of all possible unit normals and this sphere can be divided into 129,600 discreet partitions each of which can be described by two angles. Polygons are then placed in the buckets dictated by the polygon normal. This method generates near coplanar sets with a one degree resolution and is easy to implement and understand. Unfortunately, this method may produce inconsistent and/or non-optimal coplanar sets because of the fixed coplanar set boundaries. For example, the normals describing two polygons can be arbitrarily close to one another and still end up in different buckets. This would prevent the merge of the two polygons even though they are more nearly coplanar then other polygonal pairs which do get merged.

Another method considered involves building a representative tree [6] to form nearly coplanar sets. The idea is to choose a representative normal from the list of possible polygons and group subsequent polygons if they fall within some angular criterion (ϵ) where ϵ is a user specified angle describing the maximum angular

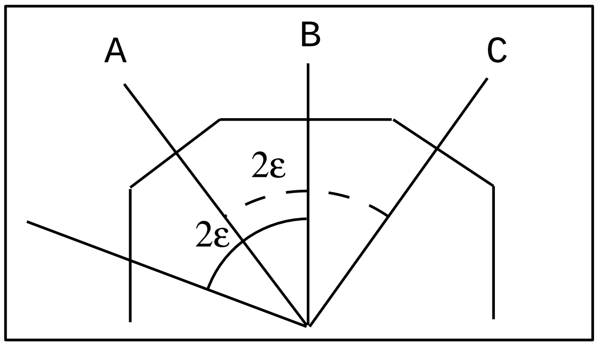

Figure 2: Choosing Representative Normals

difference between the representative normal and the normal of any other polygon in the same nearly coplanar set. This means that the angular difference between any two normals contained in the nearly coplanar set is, at most, 2ϵ. Choosing a representative normal removes the fixed boundary condition and can increase optimization efficiency however, poor representative polygon choice can miss candidates for near-coplanar set inclusion [See Figure 2]. Even though the polygons with normals A, B, and C fall within the (2ϵ) angular criterion, only choosing normal B as the representative normal will cause the three polygons to be members of the same nearly coplanar set. Multiple instances of similar cases in a single object could cause inconsistent behavior from the optimization algorithm. The representative tree method has a time complexity of $O(n \log n)$ where n is the number of polygons in the original surface.

An improved method involves a reformulation of the representative tree concept which improves grouping efficiency. A representative normal is chosen as before but each subsequent polygon is added to the near-coplanar group as long as adding it will not cause the range of normals in the group to exceed the user specified angular acceptance criterion. After a polygon is added to the group, the representative normal is adjusted to be the average normal of all polygons in the group. This modification ensures greater consistency in the range of normals represented between near-coplanar groupings. This method demonstrates the same time complexity as the unmodified representative tree method.

3.3 Building Segment List

The two step process of building a segment list and deleting duplicate segments effectively deletes all segments which are not part of a merged-polygon boundary. Using this segment based approach, redundant / unnecessary segments are removed in $O(n \log n)$ time where n is the number of segments created from the near-coplanar set of polygons.

The segment list can be constructed in linear time for any group of arbitrary polygons. Segments are constructed so that the greater[2] of the two endpoints is listed first. Once the segment list is sorted by coordinate it is a linear time operation to remove duplicate segments since they will appear sequentially in the sorted list.

3.4 Retrace Polygons

After duplicate segments are deleted, remaining segments form boundaries that describe the globally merged polygons since only boundary segments will escape deletion in the preceding step. A second, sorted segment list is created by copying the remaining segments and swapping endpoints so that the least[3] of the two endpoints appears first. Polygons are constructed by choosing a segment and using a binary search to find another segment with a common endpoint in one of the two lists. This process is repeated until there are no segments remaining that can be added to the current polygon, in which case a new polygon is started, or there are no segments left in the list. The retrace process is accomplished in $O(n \log n)$ where n is the number of segments not deleted by the duplicate deletion routine.

A choice must be made at this stage to decide whether or not to delete co-linear vertices. If they are deleted, optimization is increased but any gradient information associated with the remaining vertices may not accurately reflect the optimized surface and may cause severe streaking if used when rendering the resultant polygons. If, however, co-linear vertices are removed on the basis of the rate of change in their gradient, streaking can be controlled and optimization can be maximized. Co-linear vertices can be removed in $O(n)$ time where n is the number of vertices.

A problem that can arise involves a merged region that contains holes[4] [See Figure 3]. After the retrace stage, these holes will appear to be separate polygons. Discovering if one polygon contains another can be accomplished in linear time, $O(n)$ where n is the number of polygons generated by the retrace step, but the operation must be performed on each of the $O(n)$ retraced polygons so the entire operation has $O(n^2)$ time complexity. By this point in the algorithm, however, the number of polygons is greatly reduced and many cases can be decided quickly using simple tests such as bounding boxes.

When a hole is added to the description of a polygon, care must be taken to ensure that polygon traversal order remains consistent to prevent the creation of loops, bow-tie polygons and similar degenerate cases.

[2]Greater in the sense that the x-coordinate is more significant than the y-coordinate which is more significant than the z-coordinate.

[3]Least in the sense that the x-coordinate is more significant than the y-coordinate which is more significant than the z-coordinate.

[4]These holes are actual discontinuities in the surface being optimized and are not to be confused with errors that some iso-surface extraction methods may introduce into a surface.

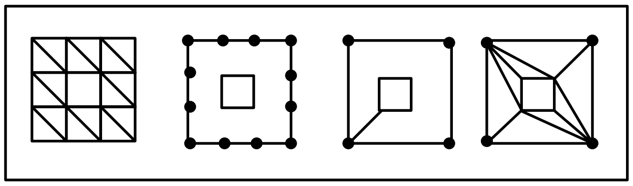
Figure 3: Handling Polygons With Holes

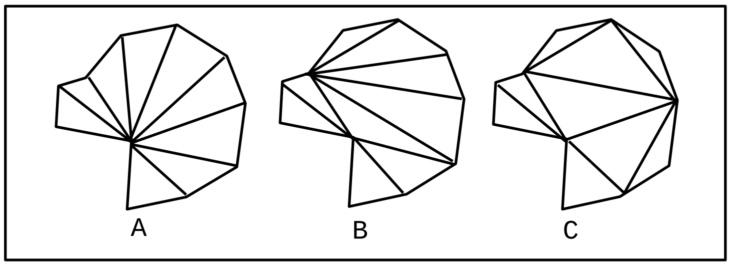

Figure 4: Triangulation Methods

If these cases are prevented, the merged polygons can be easily triangulated.

It is possible to write out polygons at this stage of the algorithm. It should be noted, however, that polygons created by the preceding stages can contain hundreds or even thousands of vertices along with interior holes and most hardware renderers perform poorly, at best, on such complex polygons.

3.5 Triangulate Polygons

The triangulation step must deal with a complicated but well defined class of polygons. While the polygons might be complex, many of the usual degenerate cases will not exist. Bow-ties, recursive loops, redundant segments and co-linear segments are special cases that will not appear if the initial phases of the algorithm are correctly implemented. It has been noted that the polygons may contain holes, however, which means any triangulation scheme adopted must handle this class of polygons.

An important consideration in choosing a triangulation method is the type of triangles generated. Since the optimization method is, effectively, a point deletion method, additional information about vertices, such as gradient information, can be retained throughout the optimization process for the resulting geometry. Preliminary results indicate that, unless care is taken, triangles with poor aspect ratios can be generated yielding shaded objects that demonstrate a variety of problems including streaked shading and zbuffer rounding errors.

Methods analyzed and discarded include a local greedy triangulation and a museum view[5] triangulation method. The primary reason these two methods were discarded is because of their tendency to produce long thin triangles which can cause shading and z-buffer errors due to out-of-proportion aspect ratios.

The same polygon is triangulated using the three methods described [See Figure 4]. All algorithms start at the vertice of greatest concavity in the polygon and traverse the polygon in a clockwise rotation.

The museum view method chooses a vertex and creates all possible triangulations from that point. This not only has the tendency to generate long thin triangles, but can also create a fan pattern of triangles that is visible when rendering the complex polygon [See Figure 4 triangulation A].

[5]A museum view describes the set of points which are within the line-of-sight of the selected point

The local greedy algorithm works much the same as the museum view method but only considers creating triangulations using vertices which appear in the list of vertices near the starting vertex. The nearness consideration has no correlation to vertice proximity in 3-space but depends solely on the location of the vertices in the connection list which describes the polygon [See Figure 4 triangulation B].

A better method of triangulating the polygons involves simply traversing the polygon and storing triangles as they are discovered. For example, if we label the starting vertice 0, the triangle formed by 0, 1 and 2 is analyzed. If the segment created by the endpoints 0 and 2 does not intersect any other segment of the polygon and the segment is contained by the polygon, it is written out and vertice 1 is removed from the vertice list. The starting vertice 0 is then set to 2 and the process is repeated. If the proposed segment isn't valid, the starting vertice is incremented by one and the process repeats. This method can produce poorly proportioned triangles and fan shaped triangulations but it is easy to implement, fast and produces triangulations that are at least as good as the other two methods considered [See Figure 4 triangulation C].

Application	Polygons	Reduced	% Reduction
Oil Well Perf	304,841	66,029	78.34%
Porous Flow	1,019,373	642,204	47.37%
Turbulent Flow	315,812	295,636	6.40%

Table 1: Polygonal Reductions

Application	Reduction	ϵ	Time(sec)
Oil Well Perf	78.34%	0.1	86.73
Porous Flow	47.73%	0.0	538.41
Turbulent Flow	6.40%	2.0	95.70

Table 2: Timings

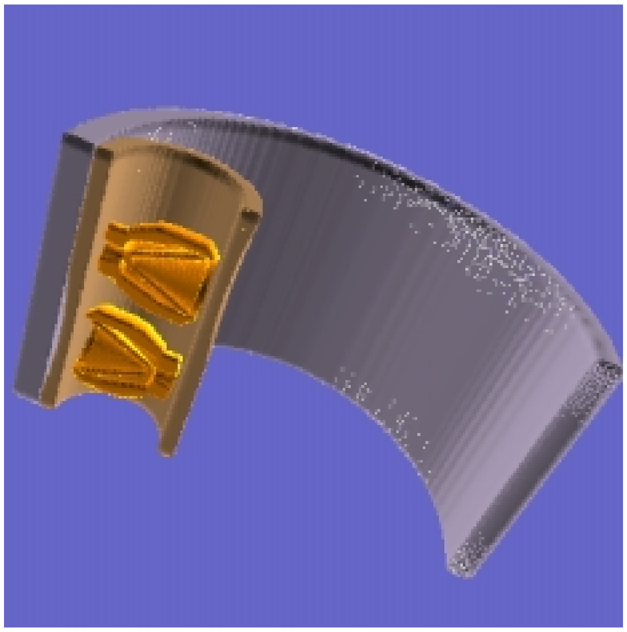

Figure 5: Original Oil Well Perferator

4 Results

The algorithm described in this paper has been applied to different scientific data sets. This section presents the results of three of these data sets chosen to be representative of many problems encountered in scientific visualization. The run-time of the algorithm is highly dependent on the number of overall polygons since the first step is to sort the entire polygon list. Also, run-time will vary widely due to the number of polygons found to be in a given coplanar or nearly coplanar set. This, again, is due to the sort that is performed on the segment list built from the coplanar set. Even though there are portions of the algorithm with quadradic time complexity, the run-time is dominated by these two O($n \log n$) functions [See Table 1 and Table 2].

4.1 Oil Well Perforator

The three dimensional hydrodynamic model *PAGOSA*[3] simulates high speed, high energy interaction between materials with various properties. One such simulation attempts to model the results of a shaped explosive charge lowered into a well casing and detonated to cause perforation in the well casing and the surrounding oil bearing strata. The simulation is conducted using approximately 10 million cells and generated over 300,000 triangles [See Figure 5. Primitive reductions of over 94% were achieved for the well casing (large semi-cylinder) using ϵ of 0.10 degree constraints for coplanar grouping. The carrier tube (small semi-cylinder) showed 87% reduction with the same grouping constraints while the charges (contained by the small semi-cylinder) demonstrated only about 2%

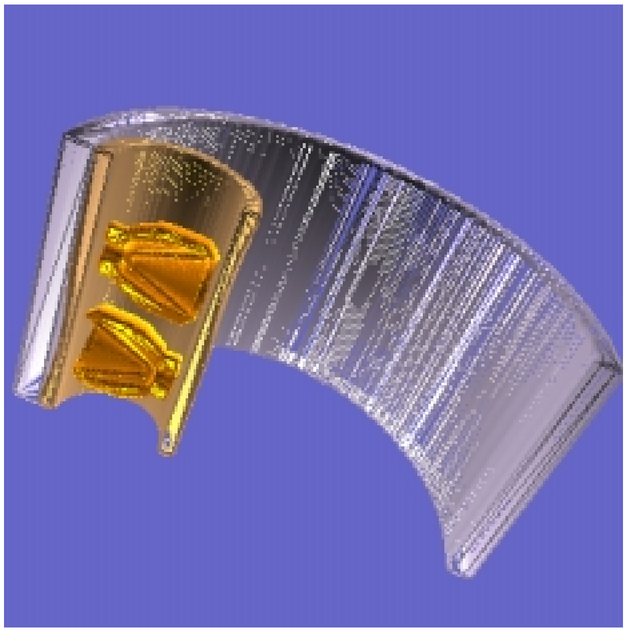

Figure 6: Optimized Oil Well Perferator

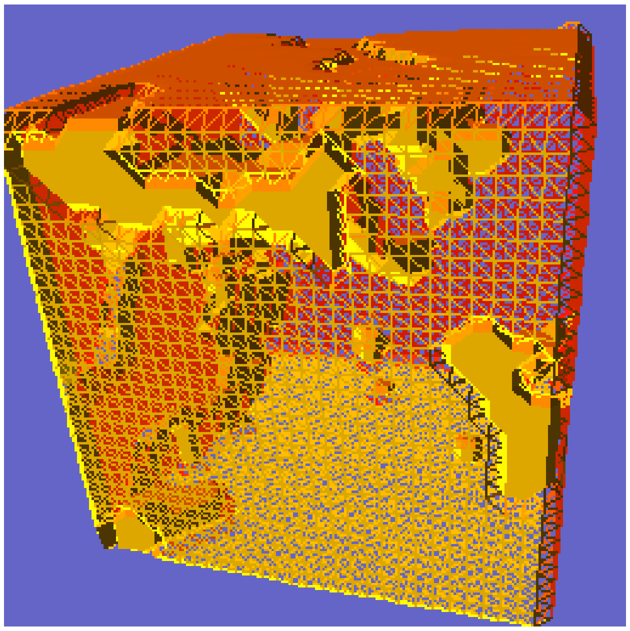

Figure 7: Original Porous Flow

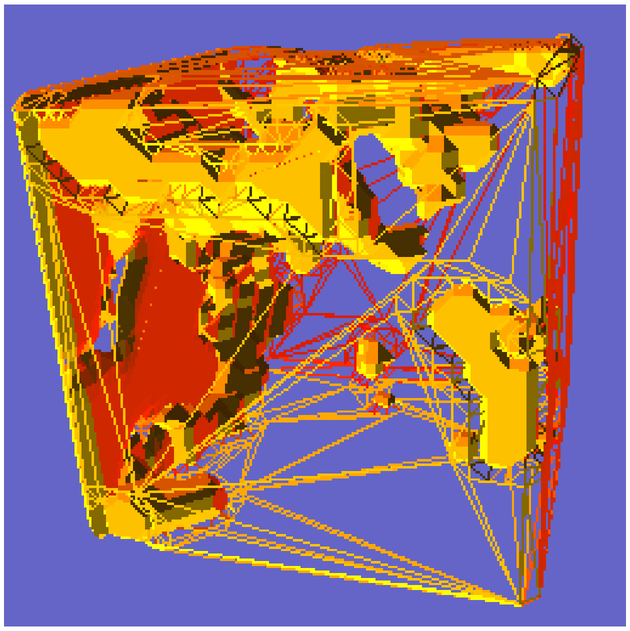

Figure 8: Optimized Porous Flow

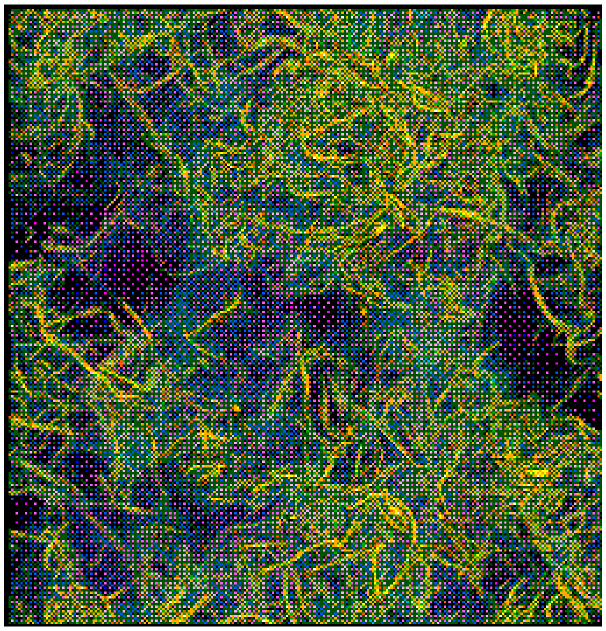

Figure 9: Original Turbulent Flow

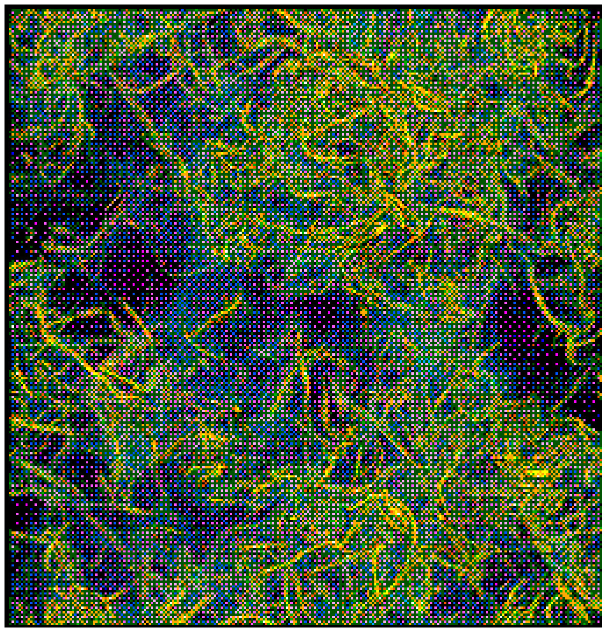

Figure 10: Optimized Turbulent Flow

reduction. Overall reduction for the five surfaces was approximately 78% [See Figure 6].

4.2 Flow Through Porous Media

The porous flow application models fluid flow through a three dimensional digitized core sample. A core sample is obtained by drilling into the porous, oil-bearing material in an oil field. The core sample is digitized at a 256^3 resolution [See Figure 7]. Isosurfacing the core sample produced over 1 million triangles using a massively parallel version of the *Marching Cubes* surface generation algorithm. Since this particular data set contains large flat areas, a 47% reduction was achieved specifying strictly coplanar sets (ϵ of zero) [See Figure 8].

4.3 Turbulent Flow

The turbulent flow model is currently running on the CM-5 massively parallel supercomputer with a grid resolutions of 256^3 and 512^3 (soon to become 1024^3) [See Figure 9]. The data set used in our experiments was 256^3. This 16 million cell model produces over 310,000 triangle primitives per surface per time step. The surface created from this volume represents the magnitude of the vorticity found in the simulated volume. Due to the many small surfaces of high curvature this model exhibits, noticeable geometry degeneration occurs before a significant reduction in geometry size can be achieved. Choosing ϵ of 2 degrees yielded reduction of slightly more than 6% [See Figure 10].

5 Conclusions

The geometry optimization algorithm can significantly reduce the amount of geometric primitive information required to faithfully reproduce an object. By reducing nearly coplanar polygonal sets to segments, the algorithm can remove redundant segments in a single pass and triangulate the remaining polygons to achieve object optimization. This method has been used successfully on many scientific applications with results presented on three representative applications. The results indicate that the algorithm described in this paper works best when optimizing areas of gradually changing orientation and is largely ineffective when faced with surfaces of high curvature.

6 Future Work

A similar, segment based approach to optimization is being developed for the CM-5 MIMD massively parallel computer located at the Advanced Computing Laboratory at Los Alamos National Laboratory. The resultant code will be coupled with a massively parallel version of the *Marching Cubes* algorithm [2]. The resulting geometry will be streamed via FDDI link to an ONYX/RE-2 Silicon Graphics renderer for visualization.

7 Acknowledgements

This research was performed in part using the resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory, Los Alamos, NM 87544.

References

[1] James H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," In *Communications of the ACM*, volume 19, number 10, pages 547–554, 1976.

[2] Charles Hansen and Paul Hinker, "Massively Parallel Isosurface Extraction," In *Proceedings Visualization '92*, pages 77–83, October 1992.

[3] D. B. Kothe, J. R. Baumgardner, J. H. Cerutti, B. J. Daly, K. S. Holian, E. M. Kober, S J. Mosso, J. W. Painter, R. D. Smith and M. D. Torrey, "PAGOSA: A Massively-Parallel, Multi-Material Hydrodynamics Model for Three-Dimensional High-Speed Flow and High-Rate Material Deformation" In *Proceedings of the 1993 Simulation Multiconference on the High Performance Computing Symposium*, March 29, 1993, pages 9–14, 1993.

[4] W. Lorensen and H. Cline, "A high resolution 3d surface construction algorithm," In *Computer Graphics*, volume 21, pages 163–169, 1987.

[5] Paul Ning and Lambertus Hesselink, "Octree Pruning for Variable-Resolution Isosurfaces," In *Proceedings of Computer Graphics International - Tokyo*, June 22-26, 1992.

[6] D. Salesin and F. Tampieri, "Grouping Nearly Coplanar Polygons Into Coplanar Sets," In *Graphics Gems III*, 1992, ISBN 0-12-409671-9.

[7] William J. Schroeder, Johnathan A. Zarge and William E. Lorensen, "Decimation of Triangle Meshes," In *Computer Graphics*, volume 26, number 2, pages 65–69, July 1992.

[8] Greg Turk, "Re-Tiling Polygonal Surfaces," In *Computer Graphics*, volume 26, number 2, pages 55–64, July 1992.