Triangle Order Optimization
for Graphics Hardware Computation Culling

Diego Nehab
Princeton University

Joshua Barczak
University of Maryland, Baltimore County

Pedro V. Sander
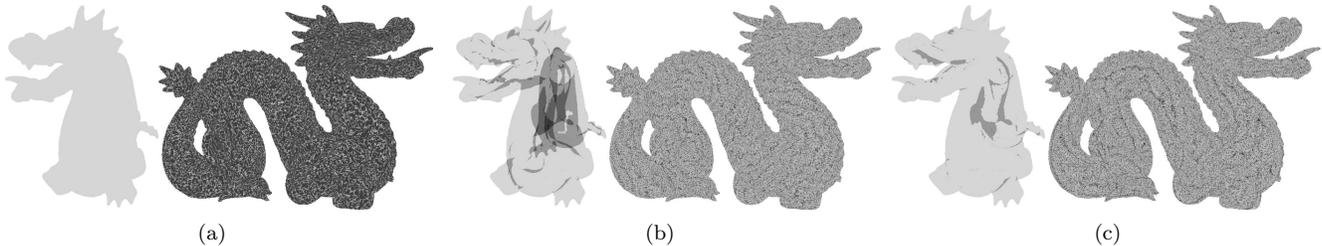ATI Research

(a)   (b)   (c)

Figure 1: Illustration of overdraw and vertex cache efficiency. In the front views, darker regions represent high-overdraw. In the side views, darker regions represent cache misses. (a) Rendering triangles in a front-to-back order maximizes the use of early Z-culling, but results in poor vertex cache performance. (b) Conversely, mesh locality optimizations produce excellent cache hit rates but can generate high levels of overdraw. (c) Our method combines both ideas into a view-independent ordering that results in excellent cache performance while minimizing overdraw.

Abstract

We describe an automatic preprocessing algorithm that reorders triangles in a mesh so as to enable the graphics hardware to efficiently cull vertex and pixel processing at rendering time.

Our method starts by dividing the mesh into planar clusters which are subsequently sorted into a view-independent order which greatly reduces overdraw. The result is an increase in the opportunities for early Z-culling, reducing pixel processing time. The clusters are then optimized for mesh locality. This produces high rates of vertex cache hits, reducing vertex processing time.

We have found that our method brings the overdraw rates of a wide range of models close to that of front-to-back order, while preserving state of the art vertex cache performance. This results in higher frame rates for pixel-bound applications with no penalty to vertex-bound applications.

1 Introduction and previous work

Modern graphics hardware pipelines organize processing in two main stages, one that processes vertices and another that processes pixels. In any given application, each one of these steps has the potential to become a bottleneck. Factors that contribute to an application being vertex- or pixel-bound may include the amount of geometric data being processed, the depth complexity, and the relative cost of the programmable components of vertex and pixel processing being used.

Naturally, graphics processors employ a series of techniques to cull both vertex and pixel computations wherever possible. We describe the two main optimizations of concern for this work. To reduce vertex overhead, modern graphics processors cache transformed vertices. When a primitive is issued that uses a vertex already in the cache, no extra work is performed for that vertex. Conversely, in the pixel side, graphics processors discard pixels that would fail the Z-test before any further processing is performed (early Z-culling). These optimizations are strongly affected by the order in which the triangles composing a mesh are issued.

Mesh locality can be optimized to ensure vertices are invoked repeatedly while transformed results are still in cache. This has been extensively investigated elsewhere [Deering 1995; Chow 1997; Hoppe 1999]. The main idea is to organize the mesh as a series of short, parallel strips that share as many vertices as possible with neighbors. State of the art methods typically bring the average number of cache misses per triangle from 3 in the random case down to 0.6–0.7 (which is very close to the 0.5 optimum).

On the other hand, drawing primitives in front-to-back order gives early Z-culling the chance to optimize away all computation that would otherwise be overwritten. Unfortunately, per-triangle real-time depth sorting on large meshes can be impractical. Furthermore, depth sorting the entire scene can ruin any gains produced by mesh locality optimizations (figure 1a). For these reasons, it is usually performed at the object level, with each object preprocessed for mesh locality. Depending on the complexity of each individual model, this approach can be far from optimal. In fact, it is often the case that mesh locality optimizations generate poor orders within objects as far as early Z-culling is concerned (figure 1b).

Another strategy to reduce overdraw is to prime the Z-buffer by rendering the geometry without outputting to the color buffer. On a second pass, which uses the real shaders, there will be no overdraw. Unfortunately, this approach doubles the amount of vertex processing required, which might be too high a price to pay. It is therefore worthwhile to investigate whether it is possible to reduce overdraw rates
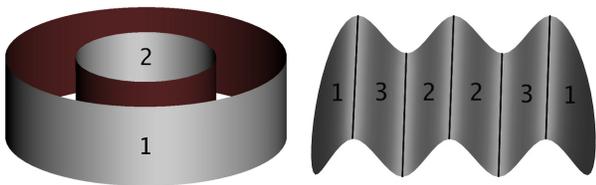
Figure 2: View independent order. Although neither of the above models is convex, rendering the labeled regions in the order implied by their respective labels ensures that, in the presence of backface culling (backfaces renderend dark), no overdraw will happen regardless of viewpoint.

without hurting cache efficiency and without changing the rendering pipeline. In that case, the resulting method could be used generally, with no disadvantages.

In this paper, we notice that in the vast majority of meshes it is possible to sort triangles in a view-independent order that greatly reduces overdraw, even if triangles are clustered into considerably large contiguous patches. These clusters can then be optimized independently to maximize mesh locality using any algorithm of choice (we use Hoppe [1999] throughout this work), with virtually no penalty with regard to optimization across the entire mesh.

The result is a pre-processing algorithm that produces a triangle order that enables the graphics hardware to cull both vertex and pixel computation within each model, leading to not only higher frame rates but more consistent frame rates in real-time applications (figure 1c).

2   View-independent sorting

As we mentioned in the introduction, the effectiveness of early Z-culling is maximized when triangles are drawn in a front-to-back order. Depth sorting, however, is fundamentally view-dependent. Although many techniques have been developed to speed up view-dependent depth sorting (see, for example, BSP related techniques [Fuchs et al. 1980; Chen and Wang 1996]), we are interested in view-independence since it is transparent to the application and does not inccur any CPU overhead.

Fortunately, in most applications, triangles facing away from the camera do not generate any pixels due to backface culling. This means that the relative order of any two triangles is irrelevant if their normals point in opposite directions. This additional freedom is what allows us to find an appropriate view-independent order that reduces overdraw.

Consider convex objects, for example. In the presence of backface culling, no triangle order can produce overdraw. Surprisingly, although order *is* important in general, for many concave objects it is also possible to find view-independent orders that produce no overdraw. Figure 2 shows two examples of concave objects and corresponding orderings. In practice, even for the cases where there is no perfect order, some orders will often be far better than average, and all we have to do is find one of them.

To simplify the problem, suppose that a given model can be clustered into non-intersecting, planar, contiguous patches. Assume further that, for each pair of patches, we are able to determine if there is any viewpoint for which one of them can obscure parts of the other. If we can then produce a patch ordering that respects most or all of the pairwise orderings, it will result in little or no overdraw, regardless of viewpoint. Because the patches are contiguous, we can then optimize each one independently for mesh local-

ity, and results will be similar in quality to that of optimizing the mesh as a whole.

We have thus divided our original problem into four smaller problems. In section 3, we describe how we cluster the mesh triangles into roughly planar patches. Then, in section 4, we describe how we generate a partial order graph representing the relative orderings between all pairs of clusters. Section 5 concludes the discussion of the algorithm, showing how we produce a cluster ordering from the graph. A final step that doesn't require discussion applies mesh locality optimization to the individual clusters.

3   Clustering into planar patches

As we mentioned before, we want to enforce the planarity of the patches produced by our clustering algorithm. That way, we minimize the amount of overdraw within each patch, which would otherwise be unavoidable regardless of the final cluster ordering.

Any of several robust algorithms for planar mesh clustering could be used to solve our problem (e.g., Garland et al. [2001], Levy et al. [2002]). We chose a method based on k-means clustering [MacQueen 1967] because it has the tendency of following global mesh features more faithfully when compared to greedy hierarchical merging methods. More specifically, we adapted the method of Sander et al. [2003] to produce clusters suitable for our application.

As in k-means clustering, the method alternates between two steps, one which assigns faces to clusters based on normal variation and distance to cluster centroids, and one which computes a new centroid for each cluster. Both achieve this objective using a breadth first search, the former starting from the centroid, and the latter starting from the cluster boundaries.

In Sander et al. [2003], since the objective is to produce clusters that are suitable for parameterization, boundary compactness of the clusters is enforced with an appropriate term in the metric that is used for determining cluster assignments. We are more concerned with planarity. Therefore, during the assignment step we use a metric that is solely based on the cluster's current average normal $\bar{\mathbf{n}}_c$ and the candidate face normal $\mathbf{n}_f$. The edge cost between a face f,

Figure 3: The Feline mesh partitioned into 32 planar patches. Notice how patches tend to follow the natural creases of the mesh. This is due to our normal-based clustering metric.

Figure 4: Viewpoint generation. A series of progressively denser viewpoint sets can be produced by subdivision of an icosahedron. From left to right, sets of 12, 42, and 162 viewpoints.

candidate for inclusion into a cluster c, and its neighboring face in that cluster is computed as follows:

$$\text{cost}(c, f) = 1 - \bar{\mathbf{n}}_c \cdot \mathbf{n}_f + \epsilon$$

where ϵ is a small constant to break ties in favor of proximity to seed in fully planar mesh sections.

Another difference is that our application does not require clusters to be topologically equivalent to disks, thus allowing us to neglect that constraint from the original algorithm.

Figure 3 shows the result of clustering the Feline mesh. Notice that the cluster boundaries have a tendency to follow the creases of the mesh.

We also considered optimizing for mesh locality *before* clustering. In that case, we included a term to the metric that favored the preservation of strip boundaries. Under certain conditions, this formed clusters whose shapes yielded slightly better vertex cache results. However, the increased curvature worsened the overdraw results. Since vertex cache performance was almost unchanged, we opted for the simpler metric, based solely on cluster and face normals. As a result, we use the number of clusters as the single parameter that controls the trade off between vertex and pixel processing.

4   Generating the partial order graph

Given the input mesh previously partitioned into clusters, we proceed to generate the partial order graph. For each pair of clusters, we must determine whether to add an arc between them, and if so, the direction and weight it should have.

Consider clusters a and b. We add an arc from a to b if drawing a before b generates less pixels than the opposite order, when rendered under a variety of viewpoints. The weight of the arc is the net difference in the number of generated pixels. If both orders generate the same number of pixels, no arc is added. The rationale is that if a can occlude large portions of b, then it should be rendered first. If only a small part of b can be occluded, the order matters less.

This is a slight simplification of the problem, because drawing a third cluster c might alter the effect on the relative overdraw between a and b (imagine a and b are inside a sphere c). In practice, this is seldom the case and the approximation is adequate.

The set of viewpoints can be generated by positioning the camera on the vertices of an icosahedron, or any of its subdivisions, as in figure 4. The model is then scaled to fit inside the icosahedron and, for each viewpoint, the frustum can be adjusted to include the entire object. We generate 512×512 images for each viewpoint and use a hardware occlusion query to determine the amount of overdraw.

If rendering with a perspective camera, the field of view and the distance to the camera affect the amount of overdraw
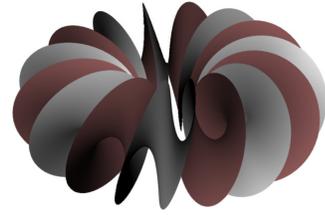

Figure 5: A model that generates cycles. If the helix is split into clusters containing each one of its twists (as shown by the alternating colors), two cycles will arise. Each cycle can be broken at one arc to generate an ordering that produces much less overdraw than random ordering.

between each pair of clusters. Ideally, the overdraw computation should take into account the camera parameters and viewpoint distribution of the target application. We have found that using an orthographic camera and 162 viewpoints evenly distributed around the model is sufficient to achieve a good approximation of overdraw for generic cases. Nonetheless, if the target application differs significantly from the generic case (e.g., the model will always be viewed from one side), the partial order graph generation can be specialized accordingly.

Since the measurements have to be performed for all pairs of clusters, the algorithm runs in $O(n^2)$ time, where n is the number of clusters. The quadratic complexity is acceptable because the number of clusters we need in practice is several orders of magnitude smaller than the number of faces in a large mesh. Even so, we can improve the efficiency of the graph construction process with the following optimizations.

If the same occlusion query object is used atomically for all queries, the system remains idle while it waits for the occlusion query result to arrive from the GPU. Accordingly, we noticed a 6x factor of improvement by issuing multiple occlusion queries consecutively before gathering their results. In our experiments, we noticed that using more than 100 simultaneous occlusion queries does not significantly improve performance.

Another key optimization is to compute the intersection of the bounding boxes of the two clusters in screen space before rendering. If that intersection is empty, the relative overdraw between the two orderings is zero. Therefore the pair can be skipped for that particular viewpoint. This optimization yields an additional 4x improvement when generating graphs for 32 clusters. Interestingly, this factor increases along with the number of clusters, since the average cluster size is smaller.

A similar method can be used to obtain the total number of pixels generated while rendering a full model, given an ordering for its clusters. We can then obtain a normalized measure of overdraw, if we divide this total by the projected area of the model. The projected area can be obtained by rendering the mesh a second time, without clearing the Z-buffer, and counting the number of pixels generated when the second pass is rendered with the depth comparison function set to equal. This is one of the tools we use to evaluate the quality of our results in section 6.

5   Ordering the clusters

If the partial order graph has no cycles, a Topological Sort [Knuth 1973] solves the ordering problem in $O(|V| + |E|)$. However, some models, such as the helix in figure 5, are likely to produce graphs with cycles. In those cases, we are interested in finding the ordering that violates the
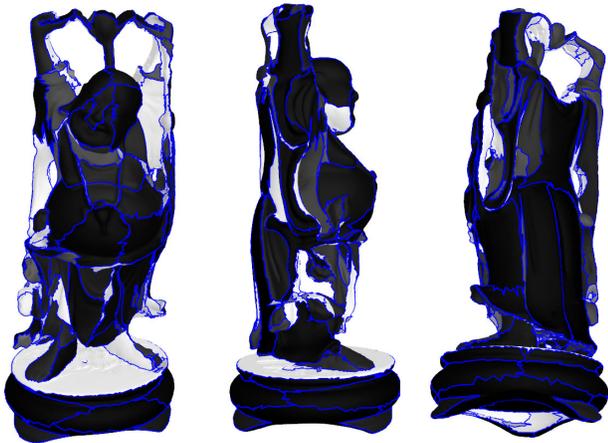
Figure 6: Result of sorting. The Buddha model has been partitioned into 256 clusters. These led to a partial order graph which was ordered with the greedy heuristic. Ordered clusters are colored in sequence from black to white. Notice how clusters that can be potentially overdrawn are white, in contrast to black clusters that are less likely to be overdrawn.

minimum number of pairwise orderings, or rather the ordering that minimizes the sum of the weights of all violations. This problem is known as the Minimum Feedback Arc Set problem [Younger 1963], and it is NP-complete [Karp 1972].

A greedy heuristic which we have found to work well in practice is described in Skiena [1997]. It has the advantages of being easy to implement, efficient to execute (it also runs in $O(|V| + |E|)$), and of agreeing with the Topological Sort whenever possible.

The idea is to find nodes with no incoming arcs or with no outgoing arcs and place them respectively in the beginning or end of the current ordering. Whenever a node is placed in the ordering, it is removed from the graph, along with its incoming and outgoing arcs. This process is repeated and the graph is progressively simplified. If there are cycles, eventually all remaining nodes will have both incoming and outgoing arcs. We then greedily select for removal the node with the greatest absolute difference in the sum of the costs of its incoming and outgoing arcs, placing it in the winning side of the ordering.

Results of this stage are shown for the Buddha model in figure 6. There, clusters have been colored from black to white, in the order determined by the heuristic. Notice that clusters that could be overdrawn tend to be white, in contrast to black clusters that are less likely to be overdrawn.

6 Results

Recall that our goal was to produce a triangle ordering that reduces overdraw while maximizing vertex cache reuse. To show that is indeed the case, we proceed in three steps. First, we show that clustering before applying mesh locality optimizations does not harm vertex cache performance. Then, we show that our clustering ordering strategy considerably reduces overdraw. Finally, we show that the reduction in overdraw translates into a frame rate improvement during real-time rendering of pixel-bound scenes.

We start with figure 7, which shows the average number of cache misses per triangle, for a variety of models. Lowest values are best, ranging from the theoretical optimum of 0.5 to the worst case of 3 cache misses per triangle. Notice how our clustering has little or no impact on the mesh local-

ity optimization (compare our results with the global vertex cache optimization).

Figure 8 shows a comparison of the worst-case overdraw ratio for any viewpoint. Overdraw ratios measure the number of pixels generated divided by the projected area of the model in pixels. Again, lowest values are best, and a value of one means *no* overdraw (as in front-to-back order). Notice how our ordering produces less overdraw when compared to the global vertex cache optimization and with random triangle ordering.

In figures 7 and 8, the models ordered by our method use the smallest number of clusters that enables our sorting strategy to reduce overdraw to acceptable levels. It turns out that this usually happens long before the number of clusters starts to harm the vertex cache performance. Figure 9 shows the relationship between number of clusters with overdraw for the Dragon and Buddha models with 150k triangles. Notice how, after a certain number of clusters, little is gained in terms of overdraw reduction. In practice, an appropriate number of clusters can be found by trying the method on successive powers of two, until the average overdraw stops improving consistently.

Finally, figure 10 shows the improvement in rendering time for a real-time, pixel-bound application due to our ordering scheme. For that, we used an application that renders models using a computationally intensive, albeit reasonable, procedural texture and per pixel lighting. This application was used to render a model of the Dragon, with resolutions of 20k and 150k triangles. The values shown are the ratios between measurements for our ordering scheme and the ordering produced by direct use of Hoppe [1999].

It is clear from the graph that lower overdraw rates correlate with lower render times. The 20k model is entirely pixel-bound and exhibits a strong correlation. If the application is not as pixel-bound, as in the 150k example, this correlation will not be as strong. Nevertheless, improvements should be expected. Notice that for some viewpoints our method results in slightly higher rendering times. In those few cases, the vertex cache optimization result outperforms our method because the view direction coincides with the direction in which the triangle strips were generated. This causes triangles to be rendered in front-to-back order. Even so, the performance difference for those viewpoints is less than 10% and these cases are significantly outnumbered by cases in which our occlusion-aware method produces much better results. Furthermore, as mentioned earlier, if the target application has a restricted set of camera positions, our optimization could take that into account to further reduce overdraw from these key viewpoints.

As final note, we also observed a considerable reduction in the variance of the overdraw rates for varying viewpoints, when we compare our results to those of the global mesh locality optimization. For example, the variances for the Dragon and Buddha models are reduced by 50 and 10 times, respectively. This leads to more consistent frame rates.

7 Conclusions

We presented an automatic preprocessing method that produces a view-independent ordering of mesh triangles which has the property of greatly reducing overdraw while preserving mesh locality. This ordering immediately results in considerably higher and more stable frame rates for pixel-bound applications, and presents no risk to vertex-bound applications. Therefore, the method can be used in a wide range of applications.
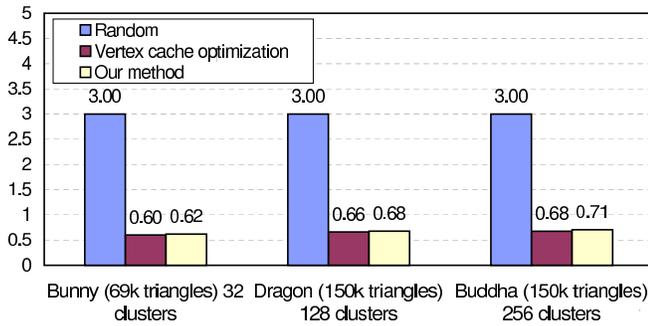
Figure 7: Vertex cache performance. Bars show the average number of of cache misses per rendered triangle. Lowest values are best. Notice that clustering does not significantly harm cache performance.
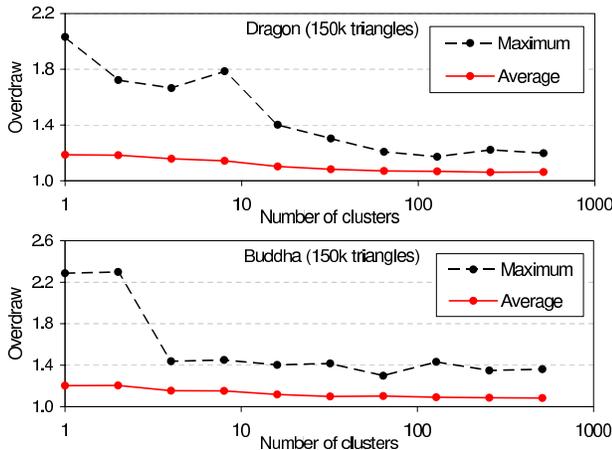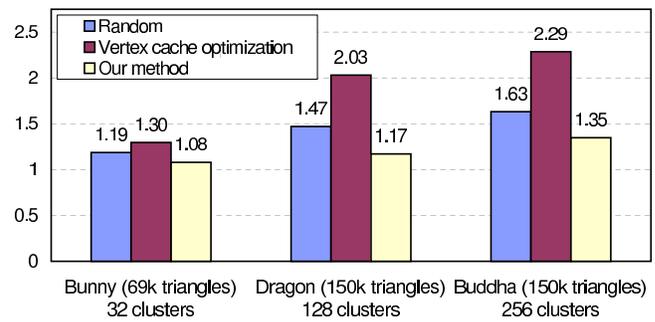

Figure 8: Overdraw. The plot shows the worst-case overdraw ratio. Lowest values are best. Notice how our method produces results closer to the optimal value of one.

Figure 9: Effect of number of clusters on overdraw. After a certain number of clusters, little is gained in term of overdraw reduction. This happens long before the small cluster sizes start harming vertex cache performance.
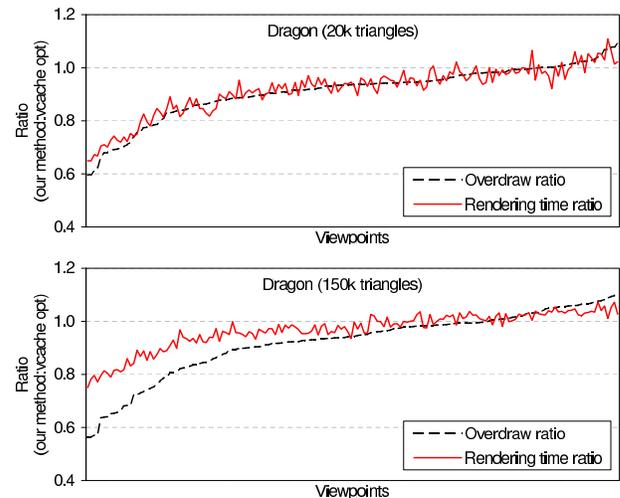

Figure 10: Correlation of overdraw with rendering time. The plot shows overdraw rates and rendering times for a model of the Dragon at two mesh resolutions rendered with an expensive pixel shader.

Acknowledgements

References

CHEN, H.-M. and WANG, W.-T. 1996. The feudal priority algorithm on hidden-surface removal. In *SIGGRAPH*, ACM Press, pages 55–64.

CHOW, M. M. 1997. Optimized geometry compression for real-time rendering. In *Visualization '97*, IEEE, pages 347–354, 559.

DEERING, M. 1995. Geometry compression. In *SIGGRAPH*, ACM Press, pages 13–20.

FUCHS, H., KEDEM, Z. M., and NAYLOR, B. F. 1980. On visible surface generation by a priori tree structures. In *SIGGRAPH*, ACM Press, pages 124–133.

GARLAND, M., WILLMOTT, A., and HECKBERT, P. S. 2001. Hierarchical face clustering on polygonal surfaces. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, ACM Press.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. In *SIGGRAPH*, ACM Press, pages 269–276.

KARP, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, Plenum Press, pages 85–103.

KNUTH, D. E. 1973. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley.

LEVY, B., SYLVAIN, P., NICOLAS, R., and JÉROME, M. 2002. Least squares conformal maps for automatic texture atlas generation. In ACM, editor, *SIGGRAPH 02, San-Antonio, Texas, USA*.

MACQUEEN, J. B. 1967. Some methods for classification and analysis of multivariate observations. In U. o. C. P. Berkeley, editor, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., and HOPPE, H. 2003. Multi-chart geometry images. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, Eurographics Association, pages 146–155. ISBN 3-905673-06-1.

SKIENA, S. S. 1997. *The Algorithm Design Manual*. Springer.

YOUNGER, D. 1963. Minimum feedback arc sets for a directed graph. *IEEE Transactions on Circuits and Systems*, 10(2):238–245.