On Graph Partitioning, Spectral Analysis, and Digital Mesh Processing

Craig Gotsman
Center for Graphics and Geometric Computing
Department of Computer Science, Technion-Israel Institute of Technology
gotsman@cs.technion.ac.il

Abstract

Partitioning is a fundamental operation on graphs. In this paper we briefly review the basic concepts of graph partitioning and its relationship to digital mesh processing. We also elaborate on the connection between graph partitioning and spectral graph theory. Applications in computer graphics are described.

1. Introduction

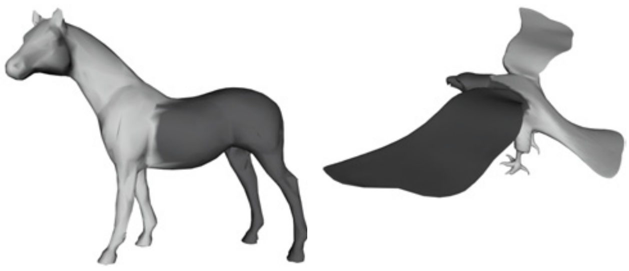
Partitioning is a fundamental operation on graphs. In its purest form, given a graph $G = <V,E>$, it is the problem of classifying the graph vertices into *two* disjoint sets satisfying the following two conditions: The vertex sets are *balanced*, meaning that neither of them is too small, and the interaction between the two vertex sets is *limited*, namely, the number of edges connecting vertices from both sets is small. This set of edges is called an *edge cut,* or just *cut*. The graph partitioning problem arises in a large number of applications, e.g. in parallel processing and numerical computation. See [24] for a complete survey. Many variants of the basic problem also exist. For example, the partition can be to more than two vertex sets, and/or the interaction between the vertex sets can be measured in a different way. Another variant on the problem is to partition the vertex set to three sets $V = U\cup S\cup W$, such that U and W are balanced and S is small. Removing S along with all edges incident on it separates the graph into two connected components. Hence S is called a *separator*. The triple $<U,S,V>$ is called a $(\beta,g(n))$-separator of a graph G containing n vertices if $V(G) = U\cup S\cup W$, $|S|<g(n)$ and $\min(|U|,|V|) > \beta n$. β is a real number in the interval [0,1].

In general, graph partitioning is NP-Hard (Problem 8.5.6. in [21]) except for a number of singular cases. The most famous is probably the family of planar graphs, for which the celebrated planar separator theorem states [17]: Every planar graph contains a $(1/3,O(\sqrt{n}))$-separator, and this separator may be computed in $O(n)$ time. This means that relatively small separators may exist even when the graph is quite large. However, finding a *minimal* $(\beta,g(n))$-separator is NP-Hard even for planar graphs.

Since graph partitioning is difficult in general, there is a need for approximation algorithms. These provide good, but not optimal, solutions to the problem, while performing quite efficiently in time and space. A popular algorithm in this respect is MeTiS [15], which has a good implementation available in the public domain.

The connection of graph partitioning to computer graphics is due to the basic fact that the vertices and edges of a 3D mesh have the structure of a graph. Many mesh operations may be described in terms of graph operations. Fig. 1 shows cuts of the connectivity structure of two 3D data sets. Note that the cut sizes are on the order of magnitude of $\sqrt{n}$, where n is the number of vertices in the graph. Note also that the cut does not necessarily partition the mesh into two meshes of equal *volume* in 3D, although the effect might be close to this, depending on the densities of the vertices on the mesh surface.

This paper provides a basic introduction to the world of graph partitioning and its relationship with digital mesh processing in computer graphics. There are interesting connections between graph partitioning and spectral graph theory which are also described in detail. Many of these issues arose in various works by the authors and his colleagues over the past few years.



(a) **(b)**

Figure 1: Some graph partitions (light and dark gray portions): **(a)** The horse model (2,048 vertices) and a balanced (996 + 1,052 vertices) small edge cut (63 edges). Note that the cut is vertical rather than horizontal, achieving a smaller value. **(b)** The eagle model (16,542 vertices) and a balanced (8,399 + 8,143 vertices) small edge cut (148 edges).

2. Graphs and Edge Cuts

Graph theorists have found graphs with small cuts to be particular interesting. This means that the graph can be separated into two balanced pieces by removing only a small number of edges. The extreme case, of course, is when the graph consists of two balanced connected components. Then the cut is the smallest possible – empty. If we think of the graph edges as pipes, then the existence of small cuts in a graph implies the existence of a bottleneck in the flow between the two vertex sets.

There are a number of ways to quantify the existence of small cuts in a graph. The standard way is the value of the *Cheeger constant* or *isoperimetric number* h_G of a graph [6]: For a vertex subset $S \subseteq V(G)$, define

$$h_G(S) = \frac{|E(S,\bar{S})|}{\min(|S|,|\bar{S}|)} \text{ and then } h_G = \min_S h_G(S)$$

G is connected if and only if $h_G > 0$. The smaller this value – the more "disconnected" G is, in the sense that it contains small edge cuts. The planar separator theorem implies that any planar graph G satisfies $h_G = O(1/\sqrt{n})$, and the graph partitioning problem attempts to achieve this.

2.1 Connectivity shapes

The class of manifold 3D triangle meshes with genus 0 is equivalent to the class of 3-connected planar graphs. The planar separator theorem implies that such meshes will necessarily have edge cuts of size $O(\sqrt{n})$, but in practice, it turns out that they usually have even smaller cuts. These small cuts correspond to narrow pieces of the model, tessellations of the model's real geometry, which typically has some non-trivial elongated form.

The fact that there are many small cuts in the connectivity graph of typical 3D models can be seen quite clearly when embedding the connectivity graph on a simple symmetric geometric object, such as the unit sphere. The embedding should be such that the mesh triangles form a valid *spherical triangulation* of the sphere, namely, that no two spherical triangle induced by the connectivity graph overlap, and that the triangles are well shaped (i.e. not long and skinny). Should the graph have small cuts – these result in visible clusters in the embedding. The reason the clusters form is that the small cut represents a region on the sphere enclosed by a small number of edges, hence with small area, yet containing a large number of vertices. Fig. 2 shows such an embedding of the triceratops model, where the clusters formed by the legs, head and other extremities are quite visible.

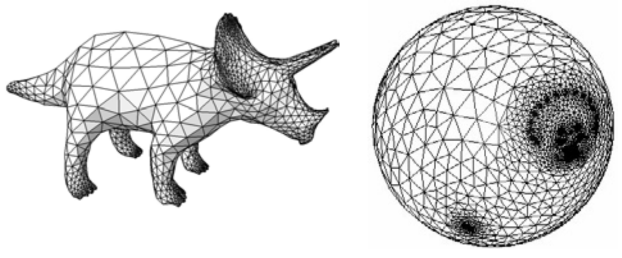


Figure 2: The triceratops model and its embedding on a sphere.

The presence of small cuts in the mesh connectivity graph conveys non-trivial information on the geometric structure of the 3D model itself. 3D models are usually tessellated from the geometric surfaces they are modeled with in the form of uniformly distributed, i.e. equilateral triangles. Assuming this, perform the following experiment: take the connectivity graph of the model, and using only this, try and reconstruct the geometry of the model. This means to find the 3D coordinates of each of the mesh vertices such that the mesh triangles form as equilateral triangles as possible in space. Quite surprisingly, the geometric shape that results from this process exhibits a close resemblance to the original 3D model, even though it was born from connectivity data only ! It is obvious that the protruding forms of the legs of the animal resulted from the small cuts in the connectivity, which "pushed" the geometry out. These shapes are called *connectivity shapes*. See Fig. 3 for an example and [12] for a detailed exposition on this topic.

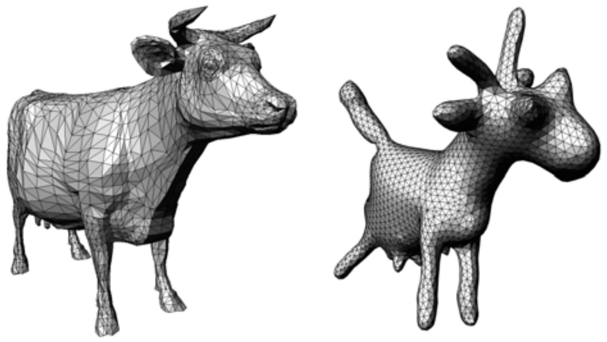
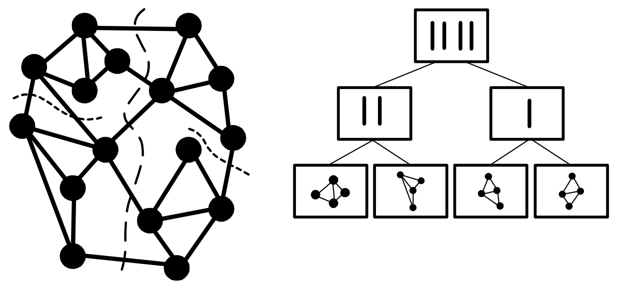


Figure 3: The cow model and its connectivity shape.

2.2 The bisection tree

Partitioning a graph into two equal halves with minimal edge cut is sometimes called *bisection.* A fundamental data structure that arises by recursive application of bisection is the *bisection tree.* These partitions the edge set of a graph among the nodes of a binary tree. Each node represents a bisection. The leaves of the binary tree are a partition of the graph vertex set. See Fig. 4.


(a) **(b)**
Figure 4: Recursive bisection of graph of 16 vertices. **(a)** Cuts of first two levels. **(b)** Corresponding bisection tree.

The bisection tree may also be viewed as a multiscale representation of the graph. If U and V are separated by a small cut, these can be viewed as two *clusters* in the graph. All vertices in each of the two sets may be collapsed into one meta-vertex in a smaller graph.

3. Connection to Spectral Analysis

The information present in a graph may be represented as the so-called *adjacency* matrix, a binary matrix $A(G)$ such that $A_{ij} = 1$ if and only if $(i,j) \in E(G)$. A related matrix is the graph *Laplacian*: $L = D\text{-}A$, where D is a diagonal matrix such that $D_{ii} = d_i$, the degree (or valence) of the i'th vertex.

3.1 The second eigenvalue

There is an intimate relationship between the combinatorial characteristics of a graph and the algebraic properties of its Laplacian. This is the essence of spectral graph theory [4,6]. In particular, there is a direct connection between the spectrum of the Laplacian and the isoperimetric number of the graph: Denote by $\{\lambda_0=0, \lambda_1, \lambda_2, .., \lambda_{n-1}\}$ the eigenvalues of L in increasing order. λ_0 is zero since L is singular due to all its rows summing to zero. For the same reason the eigenvector corresponding to λ_0 is a vector of constant values. L is connected if and only if $\lambda_1>0$. In general the co-rank of L will equal to the number of connected components of G. Moreover, λ_1 will be close to zero if G is almost unconnected, namely contains a small cut. More precisely, the following relationship between λ_1 and the isoperimetric number of the graph holds [6]:

$$\frac{h_G^2}{2} < \lambda_1 \le 2h_G$$

3.2 Spectral partitioning

Just as the second eigenvalue of $L(G)$ contains information on the presence of small cuts in the graph G, the eigenvectors of $L(G)$ play an important role in the following algorithm for partitioning G [10,2]. Denote by $\{\xi_0, \xi_1, \xi_2, ..., \xi_{n-1}\}$ the eigenvectors of L corresponding to the eigenvalues $\{\lambda_0, \lambda_1, \lambda_2, .., \lambda_{n-1}\}$. Embed G in R^d using the eigenvectors $\{\xi_1, \xi_2, ..., \xi_d\}$ as coordinate vectors, namely the vertex v_i is positioned at point $(\xi_1(i)\ \xi_2(i), ..., \xi_d(i)) \in R^d$. Now find the direction s of the largest spread of the vertices in R^d, and the $(d\text{-}1)$-dimensional hyperplane in R^d normal to s which partitions R^d into two halfspaces such that approximately half the vertices of V are in each halfspace. The graph edges that straddle the hyperplane are a small cut of G. In this way it is possible to reduce the combinatorial graph partitioning problem to a geometric space-partitioning problem. The intuition behind this algorithm can be seen by the following argument for the case d=1: First note that:

(1) $$x^T Lx = \sum_{(i,j)\in E} (x_i - x_j)^2$$

and

(2) $$\lambda_1 = \min_{x\perp(1,..,1)} \frac{x^T Lx}{x^T x}$$

where the minimum of (2) is obtained for $x = \xi_1$. Eq. (2) is the well known *Rayleigh quotient*. The condition $x \perp (1,..,1)$ is equivalent to x having zero mean, so approximately half its entries are negative and half positive. Since ξ_1 minimizes (2), (1) implies that embedding the graph vertices on the real line according to their values in ξ_1 minimizes the resulting sum of squared edge lengths. Hence partitioning the graph at any point along the real line will give a small edge cut, and the most balanced will probably be when the cut is performed at the origin. In practice using d=1 or d=2 is sufficient to produce reasonably good partitions. The case d=1 is particularly simple: Compute the eigenvector ξ_1 (also known as the *Fiedler* vector of G [8]) and partition the vertices according to the sign of the corresponding entry in the eigenvector. Fig. 5 shows the two dimensional (d=2) spectral embedding of the graph of Fig. 4a and the resulting partition. Note that while the 2D embedding of the graph is not planar (there are edge-crossings), hence not as visually pleasing as it could be, the embedding is reasonable, and in fact, embedding using the Laplacian eigenvectors is one approach to *drawing* graphs [16]. Note, however, the difference between this graph drawing approach and the connectivity shapes of Section 2.1.

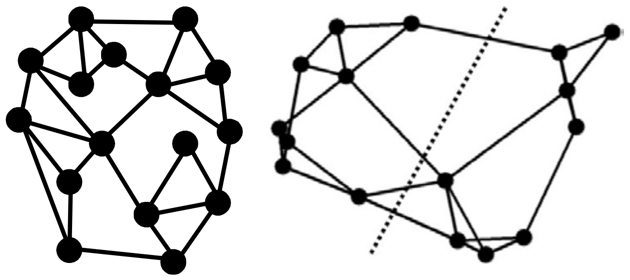


Figure 5: Graph (left) and spectral embedding (right) using first two eigenvectors of Laplacian matrix as coordinate vectors. Dashed line is resulting partition.

4. Locality-Preserving Orderings

The concept of locality in a graph is similar to the concept of locality in metric spaces. Two vertices are *close* if a small number of edges form a path between them. Two vertices are *far* apart if all paths between them are long, certainly if they are in different connected components (so that no path between them exists).

4.1 Minimal linear arrangements

A particularly interesting graph-theoretic problem is Minimal Linear Arrangement (MLA). This involves finding an ordering of the graph vertices such that vertices close to each other in the graph are not too far from each other in the ordering. More precisely, the problem is to find a one-to-one mapping $\pi: V \to \{1,..,n\}$ such that

$$\pi = \arg\min_{\pi} \sum_{(i,j)\in E} \left|\pi(i) - \pi(j)\right|$$

This is a *locality-preserving* ordering, meaning that when the graph is traversed in the order prescribed by π, there are not too many jumps within the graph. MLA is an example of a geometric *embedding* problem, where the domain is the one-dimensional grid. MLA is NP-Hard [9], but a variety of approximation algorithms exist. One very effective heuristic method to approximate MLA is to build a bisection tree for the graph, and then order the vertices as they appear in the tree leaves from left to right. The reason that this works is that if the graph contains a small cut, a good MLA ordering should first traverse the first half of the graph completely, and only then the second half. Applying this reasoning recursively (albeit greedily) leads to a good locality-preserving ordering, in analogy to the recursive constructions of the celebrated Hilbert space-filling curves on grids [20].

4.2 Approximating spectral basis functions

Given a 3D mesh consisting of a connectivity structure and geometry associated with the vertices, it is quite obvious that the connectivity structure captures some of the correlation between the vertex geometries, in the sense that vertices closer to each other in the connectivity graph are more correlated than others. This is particularly true for smooth meshes. Hence the connectivity structure can be used to extract the correlation from the geometric data, or, in other words, code it efficiently. One way to do this, inspired by traditional transform coding in signal processing, is to express the geometry vectors x, y and z as a linear combination of a small number of basis vectors. A good choice of basis vectors turn out to be the eigenvectors of the graph Laplacian $\{\xi_0, \xi_1, \xi_2, ..., \xi_{n-1}\}$, who form an orthonormal basis of R^n [13]. The reason that these basis vectors are so suitable is that the *spectra* of x, y and z, i.e. the vector of projections onto the basis function, decay rapidly, hence may be truncated and/or aggressively quantized to encode the geometry vectors in a small number of bits. The eigenvalue corresponding to the eigenvector is the analog of the *frequency* of a harmonic basis function. Thus most of the energy in the geometric mesh signal is concentrated in the low frequencies, i.e. the eigenvectors corresponding to small eigenvalues. See Fig. 6(c).

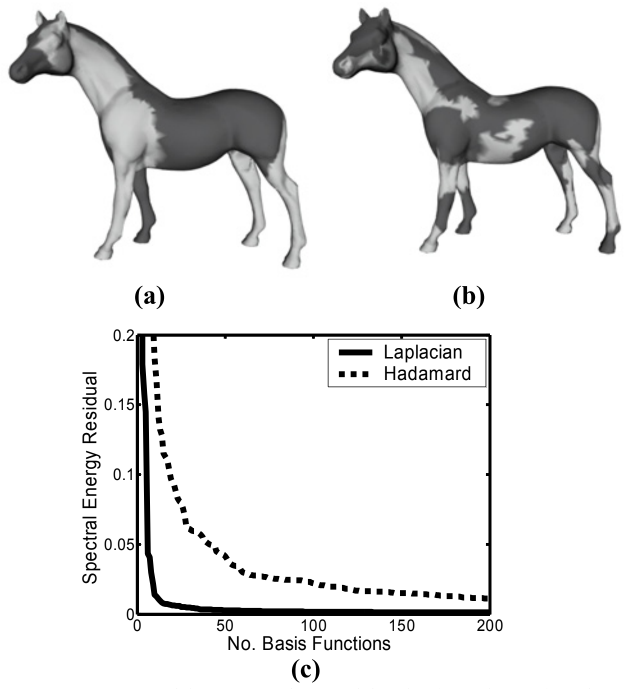


Figure 6: Two binary Hadamard basis functions for the horse model. Dark regions indicate vertices with value -1, and light regions have value +1. (a) low frequency. (b) high frequency. (c) Residual energy (accumulated energy in the rest of the basis vectors) of horse model on Laplacian and Hadamard bases.

Unfortunately, this *spectral coding* method is not practical, especially for large meshes, since the computation of the d first eigenvectors requires $O(dn^2)$, which is prohibitive for large n. However, the following method, based on mesh partitioning, reversing the logic of Section 3.2 allows us to build an orthonormal basis which is a good approximation to the optimal Laplacian basis: Assume the number of graph vertices is a power of two. Build an (approximate) bisection tree for the graph with $\log_2 n$ levels. At level k=0 generate the constant unit basis vector. At level k>0, take the 2^{k-1} basis vectors generated at the previous level and generate another 2^{k-1} basis vectors by multiplying the first set by the pattern of 2^k alternating sequences of 1's and -1's and normalizing to unit norm. The latter operator doubles the *frequency* of the second set of basis vectors. The frequency of a vector is defined as the number of zero crossings (sign changes) in it. It is easy to prove by induction on k that the resulting set of 2^k vectors is orthonormal. It is also easy to see that if the MLA construction from the bisection tree, described in Section 4.1, is used, the resulting basis is just the 1D Hadamard basis [23] on the vertices ordered by that MLA. The rationale behind this construction is that if the first Laplacian eigenvector partitions the graph well based on the signs of its entries, then a binary sign vector constructed from a min-cut partition of the graph should be a reasonable approximation to the Laplacian eigenvector. This argument may be applied recursively. Fig. 6 shows some basis vectors constructed this way and a comparison of the decay of the coefficients of the geometry vectors when using this basis as compared to the optimal Laplacian eigenbasis. Both decay quite rapidly, which is an indicator of the suitability of the basis for coding purposes.

4.3 Rendering Sequences

Modern graphics hardware allows the use of a vertex buffer to cache vertex computations, such as projection and lighting, during rendering. To exploit this mechanism as much as possible, the mesh triangles should be rendered in an order such that as many vertices as possible are present in the finite size cache when they are needed. This means that the mesh triangles should be rendered in an order that preserves their locality as much as possible, avoiding long jumps between far mesh triangles. This ordering of the mesh triangles is called a *rendering sequence* for the mesh [11], and it is an extension of the traditional triangle stripping techniques [7]. The so-called *average cache-miss ratio* (ACMR), the average number of vertex cache misses incurred per rendered triangle, is used to measure the performance of a rendering sequence. This number can be anywhere between 0.5 and 3, so a good rendering sequence can result in frame rates which are up to six times faster than those obtained without any special rendering sequence (i.e. when the mesh triangles are rendered in an essentially random order). See [3] for some theoretical bounds on the expected speedups when using a cache of size k to render a mesh of size n.

A good rendering sequence can be obtained by generating a MLA for the dual mesh – the graph whose vertices correspond to the original (primal) mesh triangles, and edges to edges [5]. An MLA of the dual mesh is a traversal of the original mesh triangles. Fig. 7 shows a 3D mesh and a visualization of the rendering performance when a rendering sequence is and is not used. The frame rate increases by a factor of 3.7 when the rendering sequence is used.

4.4 Progressive meshes

When the mesh vertices are ordered into a *vertex sequence* using MLA, and extra care is taken such that the jumps along the sequence are not too long, they induce a natural multiresolution structure on the mesh. This vertex sequence is an (almost Hamiltonian) path traversing all mesh vertices in a locality-preserving order. The resolution of the mesh may be reduced using the *edge collapse* operation, which unites two vertices to one. With this, a natural multiresolution structure may be induced on the mesh, and even exploited for efficient coding of the mesh [14] by building *layers* as follows: Collapse every second edge in the vertex sequence to transform n vertices to $n/2$ vertices. Fig. 8 shows two levels of the multiresolution horse mesh constructed this way. It can be shown to be equivalent to the application of a one-dimensional Haar wavelet to the geometric signal ordered by the vertex sequence.

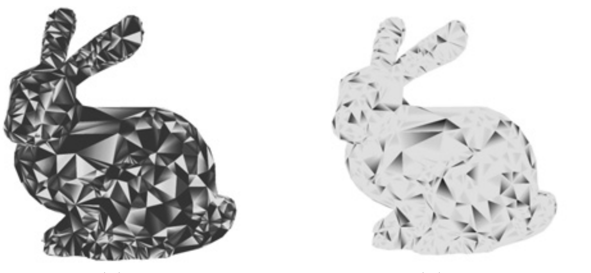


Figure 7: Visualization of rendering performance of a locality-preserving rendering sequence. Light vertices incur no cache misses. Dark vertices incur many cache misses. **(a)** Using random ordering, ACMR = 2.65. **(b)** Using MLA-induced rendering sequence, ACMR = 0.71.

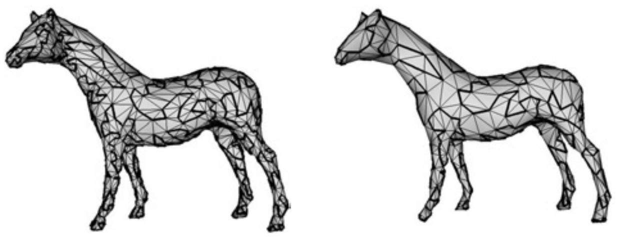


Figure 8: Two multiresolution levels of the horse model with vertex sequences (dark black lines). The model on the right (512 vertices) was generated from the model on the left (1024 vertices) by collapsing every second edge in the vertex sequence. This may be continued recursively until only a single vertex is left.

5. Mesh Connectivity Coding

With the advent of the WWW, transmitting 3D model data over the Internet is an important application, which requires representing the mesh data (connectivity + geometry) in the most compact manner. Over the past few years, a wealth of algorithms have been published for connectivity coding (e.g. [19,22]). The common denominator of many of these algorithms is that they perform a mesh *conquest,* meaning traversal of the mesh triangles or vertices, coding them into a bit stream as they proceed. At any given point in time during the coding, the mesh consists of two parts: the part that has been coded, and the part that has not yet been coded. The same holds for the decoder. The interface between these two parts is usually called the *cut-border* (or *active list*), and, as its name implies, it is actually a cut or separator of the graph into two (not necessarily connected) components.

The main data structure maintained by the coding algorithms relates to the maintenance of the dynamic cut-border during the encoding/decoding process. This usually follows from the fact that once a mesh triangle has been encoded/decoded, it can be "forgotten", in the sense that it is no longer needed to deal with the remaining triangles in the mesh. It order to maintain as small a data structure as possible, it is desirable that the cut-border be kept as small (or *tight*) as possible. This is none other than a small edge cut. So a good connectivity coding algorithm should strive to keep the cut-border as short as possible. Although the coding algorithms can only minimize the cut in a greedy manner, it turns out that the valence-coding algorithm of Touma and Gotsman [22] does a good job in this respect, and this has been optimized further in an improvement due to Alliez and Desbrun [1]. Fig. 9 illustrates the cut-borders generated in the valence coding of the hand model.

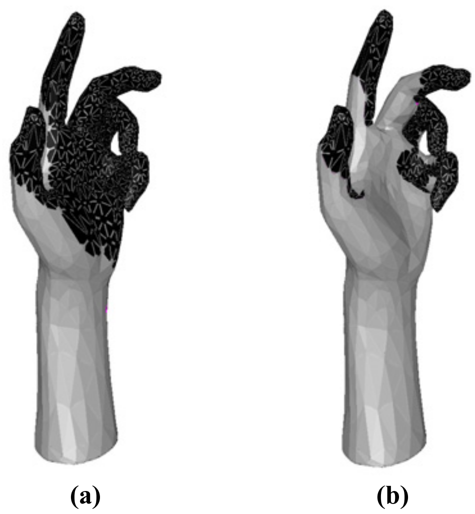

(a) **(b)**

Figure 9: Partition generated by the Touma-Gotsman connectivity coder. Light triangles have already been coded and dark triangles not yet. **(a)** Early in the coding process. **(b)** Late in the coding process.

6. Discussion and Conclusion

Graph partitioning has proven to be an effective tool for various mesh processing operations, and is intimately connected to various locality and spectral properties of the mesh connectivity structure. In some cases, it also provides a link between the connectivity and geometry components of the data set, which are not as unrelated as might seem at first glance.

The discussion in this paper has revolved around the partitioning of 3D mesh datasets based only on their connectivity graphs. In practice, 3D data sets have properties associated with the vertices, the most basic one being geometry. These properties may be used to segment the mesh to reflect its intrinsic properties better. This is useful for 3D mesh simplification, smoothing and coding but beyond the scope of this paper.

While the graph Laplacian is traditionally considered the correct operator to use for spectral analysis of graph properties, recent developments in algebraic and spectral graph theory indicate that a more appropriate set of operators might be the so-called Colin de Verdiere matrices [18]. These are a superset of the Laplacian matrices for which it is possible to better characterize the quality of the embeddings obtained using their eigenvectors.

7. Acknowledgements

Much of the material in this paper surfaced in joint work of mine with Alex Bogomjakov, Stefan Gumhold, Martin Isenburg, Zachi Karni and Alla Sheffer.

References

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding of 3D meshes. Proceedings of Eurographics, pp. 480-489, 2001.

[2] C.J. Alpert and S.Z. Yao. Spectral partitioning: The more eigenvectors, the better. In 32th DAC, ACM/IEEE, 195-200, 1995.

[3] R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. ACM Transactions on Graphics 15(2):141-152, 1996.

[4] N. Biggs. Algebraic graph theory (2nd Ed.). Cambridge University Press, 1993.

[5] A. Bogomjakov and C. Gotsman. Universal rendering sequences for transparent vertex caching of progressive meshes. Computer Graphics Forum 21(2):137-148, 2002.

[6] F.R.K. Chung. Spectral graph theory. CBMS 92, AMS, 1997.

[7] J. El-Sana, F. Evans, A. Varshney, S. Skiena, and E. Azanli. Efficient computing and updating triangle strips for real-time rendering. Computer-Aided Design 32(13):753-772, 2001.

[8] M. Fiedler. A property of eigenvectors of non-negative symmetric matrices and its application to graph theory. Czechoslovak Math. Journal, 25:619-633, 1975.

[9] M.R. Garey and D.S. Johnson. Computers and intractability: A guide to the theory of NP-completeness, W.H.Freeman, 1979.

[10] K.M. Hall. An *r*-dimensional quadratic placement algorithm. Management Science 17:219–229, 1970.

[11] H. Hoppe. Optimization of mesh locality for transparent vertex caching. Proceedings of ACM SIGGRAPH, 269-276, 1999.

[12] M. Isenburg, S. Gumhold and C. Gotsman. Connectivity shapes. Proceedings of IEEE Visualization, 2001.

[13] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. Proceedings of ACM SIGGRAPH, 279-286, 2000.

[14] Z. Karni, A. Bogomjakov and C. Gotsman. Efficient rendering of multi-resolution meshes from compressed form. Proceedings of IEEE Visualization, 2002.

[15] G. Karypis and V. Kumar. MeTiS - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 4, University of Minnesota, 1998. Available on WWW at URL http://www-users.cs.umn.edu/~karypis/ metis

[16] Y. Koren On spectral graph drawing. Preprint. Weizmann Institute of Science, 2003.

[17] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. SIAM Journal of Applied Math. 36(2):177-189, 1979.

[18] L. Lovasz and A. Schrijver. On the nullspace of a Colin de Verdiere matrix. Annales de l'Institute Fourier, 49:1017-1026, 1999.

[19] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, 5(1), 1999.

[20] H. Sagan. Space-filling curves. Springer Verlag, New York, 1994.

[21] S. Skiena. The algorithm design manual, Springer-Telos, 1998.

[22] C. Touma and C. Gotsman. Triangle mesh compression. Proceedings of Graphics Interface '98, pp. 26-34, 1998.

[23] R. Yarlagadda, K. Rao and J.E. Hershey. Hadamard matrix analysis and synthesis with applications to communications and signal/image processing. Kluwer, 1997.

[24] http://rtm.science.unitn.it/intertools/graph-partitioning/links.html