Geometric Mesh Partitioning: Implementation and Experiments

John R. Gilbert* Gary L. Miller† Shang-Hua Teng‡

Revised 30 September 1994

Abstract

We investigate a method of dividing an irregular mesh into equal-sized pieces with few interconnecting edges. The method's novel feature is that it exploits the geometric coordinates of the mesh vertices. It is based on theoretical work of Miller, Teng, Thurston, and Vavasis, who showed that certain classes of "well-shaped" finite element meshes have good separators. The geometric method is quite simple to implement: we describe a Matlab code for it in some detail. The method is also quite efficient and effective: we compare it with some other methods, including spectral bisection.

Keywords: centerpoints, conformal mapping, data-parallel algorithms, geometric sampling, graph and geometric algorithms, Matlab, mesh partitioning, moment of inertia, parallel processing, scientific computing, separators, sparse matrix computations.

AMS(MOS) subject classifications: 65F50, 68Q20

Computing Reviews descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra — *Linear systems (direct and iterative methods)*, *Sparse and very large systems*; G.2.2 [**Discrete Mathematics**]: Graph Theory — *Graph algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software — *Algorithm analysis*

1 Introduction

Solving a large problem on a parallel computer with distributed memory usually requires that the data for the problem be partitioned somehow among the processors. The quality of the partition affects the speed of solution; a good partition divides the work up evenly and requires as little communication as possible.

Many problems can be represented as graphs. Examples are both direct and iterative methods for sparse linear system solution [20, 40], and, more generally, many situations in which partial differential equations are solved in physical simulation and modeling. Partitioning such a problem typically amounts

*Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304-1314 (gilbert@parc.xerox.com).

†School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (glmiller@theory.cs.cmu.edu).
This author's work was supported in part by National Science Foundation grants DCR–8713489 and CCR–9016641.

‡This author was a postdoctoral scientist at the Xerox Palo Alto Research Center when this work was done. Present address: Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455 (teng@cs.umn.edu).

to dividing the vertices of the graph into sets of equal size with few edges joining vertices in different sets. Graph partitioning has been an active field of research for several years, both theoretically [2, 9, 17, 21, 31, 32, 33], and experimentally [1, 14, 15, 16, 19, 27, 29, 37, 39, 42]. Optimal partitioning is an NP-hard problem, and finding good graph partitions in practice can be very expensive.

Graphs from large-scale problems in scientific computing are often defined geometrically. They are *meshes* of elements in d-dimensional Euclidean space (typically $d = 2$ or 3). This paper reports on experiments with a geometric mesh partitioner, which is based on theoretical work of Miller, Teng, Thurston, and Vavasis that we summarize in Section 2. The method partitions a d-dimensional mesh by finding a suitable sphere in d-space, and dividing the vertices into those interior and exterior to the sphere. The cutting sphere is found by a randomized algorithm that involves a conformal mapping of the points on the surface of a sphere in $d+1$-space. If the mesh elements are *well-shaped* in a suitable sense, the theoretical algorithm provides a strong guarantee on the quality of the partition it generates [34, 35, 41]. In practice, our implementation produces partitions that are better than the theoretical guarantees and are competitive with those produced by other modern methods.

The goal of this paper is to convince the reader of three things. First, though the theory behind the geometric partitioner is fairly complicated, the algorithms themselves are quite simple and easy to implement. Second, the implementation can be made quite efficient. Third, the partitions produced are quite good. As evidence for the first point, Section 3 discusses the engineering that makes the theoretical algorithm efficient in practice, and describes a Matlab implementation in some detail. We present experimental evidence for the second and third points in Section 4.

2 Theory of Geometric Partitioning

We now briefly review Miller, Teng, Thurston, and Vavasis's theoretical work on separators in geometrically defined graphs. For details and proofs, see their papers [34, 35].

The partitioning algorithm maps the d-dimensional mesh into a $d+1$-dimensional space. Our descriptions (and code) are correct for any $d \geq 2$, but our terminology corresponds to $d = 2$. Thus "circle" and "disk" mean "sphere in $\mathbb{R}^d$" and "ball in $\mathbb{R}^d$", while "sphere" and "plane" mean "sphere in $\mathbb{R}^{d+1}$" and "d-dimensional hyperplane".

2.1 Overlap graphs

Computational meshes are often composed of elements that are *well-shaped* in some sense, such as having bounded aspect ratio or having angles that are not too small or too large. Miller et al. define a class of so-called *overlap graphs* to model this kind of geometric constraint.

An overlap graph starts with a *neighborhood system*, which is a set of closed disks in d-dimensional Euclidean space and a parameter k that restricts how deeply they can intersect.

Definition 1 *A* k-ply neighborhood system in d dimensions *is a set* $\{D_1, \ldots, D_n\}$ *of closed disks in* $\mathbb{R}^d$*, such that no point in* $\mathbb{R}^d$ *is strictly interior to more than* k *of the disks.*

A neighborhood system and another parameter α define an overlap graph. There is a vertex for each disk. For $\alpha = 1$, an edge joins two vertices whose disks intersect. For $\alpha > 1$, an edge joins two vertices if expanding the smaller of their two disks by a factor of α would make them intersect.

Definition 2 *Let $\alpha \geq 1$, and let $\{D_1, \ldots, D_n\}$ be a k-ply neighborhood system. The (α, k)-*overlap graph *for the neighborhood system is the graph with vertex set $\{1, \ldots, n\}$ and edge set*

$$\{(i,j) : (D_i \cap (\alpha \cdot D_j) \neq \emptyset) \text{ and } ((\alpha \cdot D_i) \cap D_j \neq \emptyset)\}.$$

We make an overlap graph into a mesh in d-space by locating each vertex at the center of its disk.

Overlap graphs are good models of computational meshes because every mesh of bounded-aspect-ratio elements in two or three dimensions is contained in some overlap graph (for suitable choices of the parameters α and k). Also, every planar graph is an overlap graph. Therefore, any theorem about partitioning overlap graphs implies a theorem about partitioning meshes of bounded aspect ratio and planar graphs.

2.2 Separators for overlap graphs

The central theorem about overlap graphs is that they have good *separators*, that is, small sets of vertices whose removal divides them approximately in half. A regular cubic mesh in d-space, with n vertices in an array $n^{1/d}$ on a side, can be divided in half by removing the $n^{(d-1)/d}$ vertices on a $d-1$-dimensional slice through the middle of the array. Up to a constant factor that depends on α, k, and d, an overlap graph in d dimensions has as good a separator as the cubic mesh.

Theorem 1 (Geometric Separators [35]) *Let G be an n-vertex (α, k)-overlap graph in d dimensions. Then the vertices of G can be partitioned into three sets A, B, and C, such that*

- *no edge joins A and B,*
- *A and B each have at most $(d+1)/(d+2)$ vertices,*
- *C has only $O(\alpha k^{1/d} n^{(d-1)/d})$ vertices.*

Miller et al. gave a randomized algorithm to find the separator in the theorem, which runs in linear time on a sequential machine or in constant time on a PRAM with n processors. The separator is defined by a circle (that is, a sphere in $\mathbb{R}^d$). The algorithm chooses the separating circle at random, from a distribution that is carefully constructed so that the separator will satisfy the conclusions of Theorem 1 with high probability. The distribution is described in terms of a stereographic projection and conformal mapping on the surface of a sphere one dimension higher, in $\mathbb{R}^{d+1}$.

Here is an outline of the algorithm. Figures 1 to 6 show the steps in partitioning the 2-dimensional mesh in Figure 1.

- **Project Up.** Project the input points stereographically from $\mathbb{R}^d$ to the unit sphere centered at the origin in $\mathbb{R}^{d+1}$. Point $p \in \mathbb{R}^d$ is projected to the sphere along the line through p and the "north pole" $(0, \ldots, 0, 1)$. (See Figure 3.)
- **Find Centerpoint.** Compute a *centerpoint* of the projected points in $\mathbb{R}^{d+1}$. This is a special point in the interior of the unit sphere, as described below. (See Figure 3.)
- **Conformal Map: Rotate and Dilate.** Move the projected points in $\mathbb{R}^{d+1}$ on the surface of the unit sphere in two steps. First, rotate the projected points about the origin in $\mathbb{R}^{d+1}$ so that the centerpoint becomes a point $(0, \ldots, 0, r)$ on the $d+1$-st axis. Second, dilate the points on the

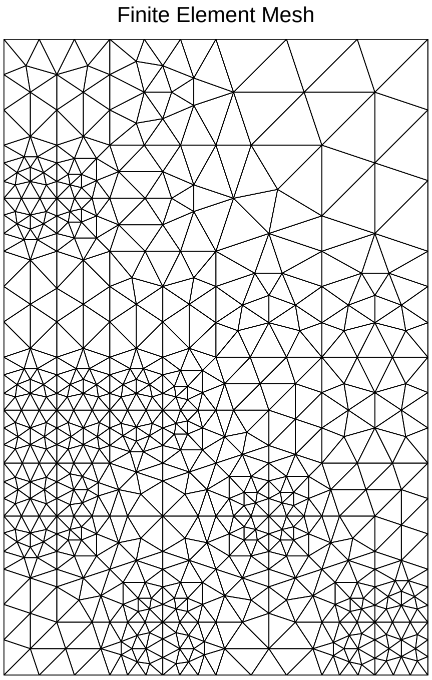

Figure 1: The input mesh.

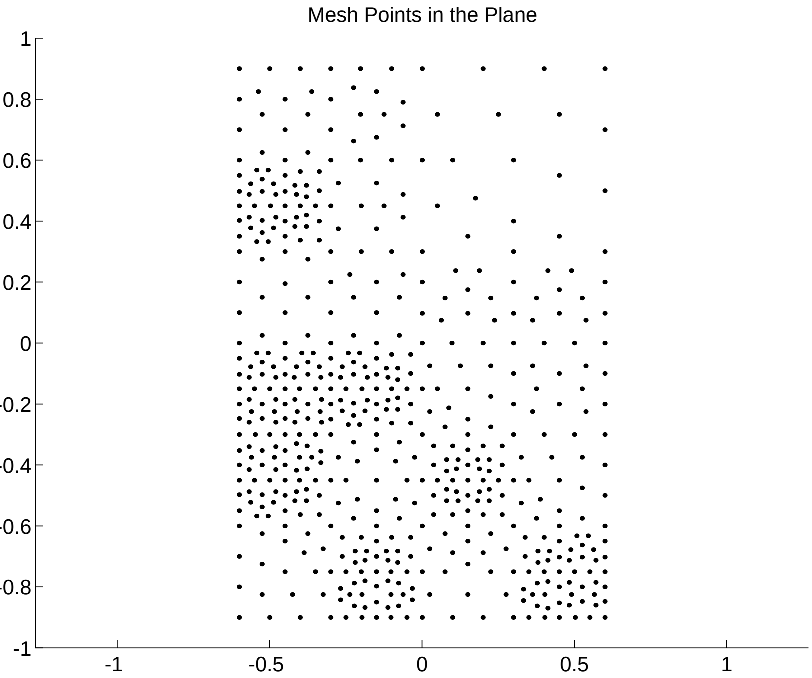

Figure 2: The mesh points.

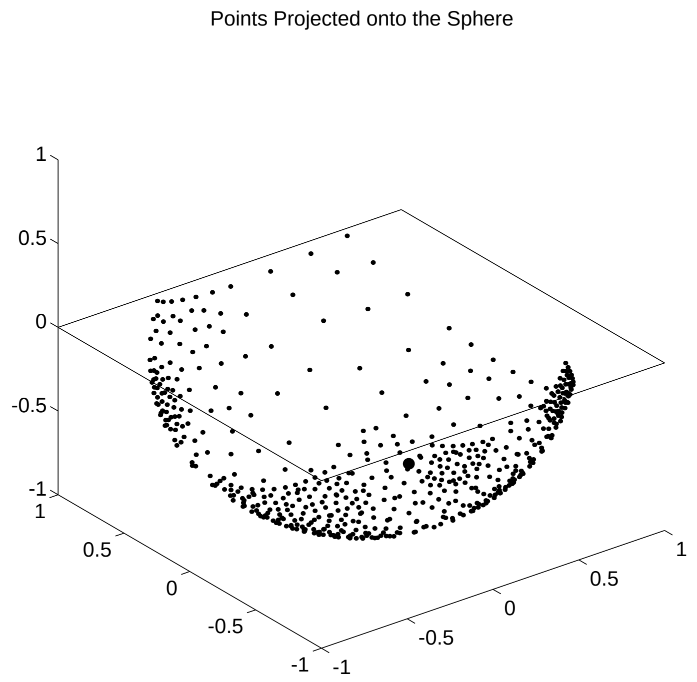

Figure 3: Projected mesh points. The large dot is the centerpoint.

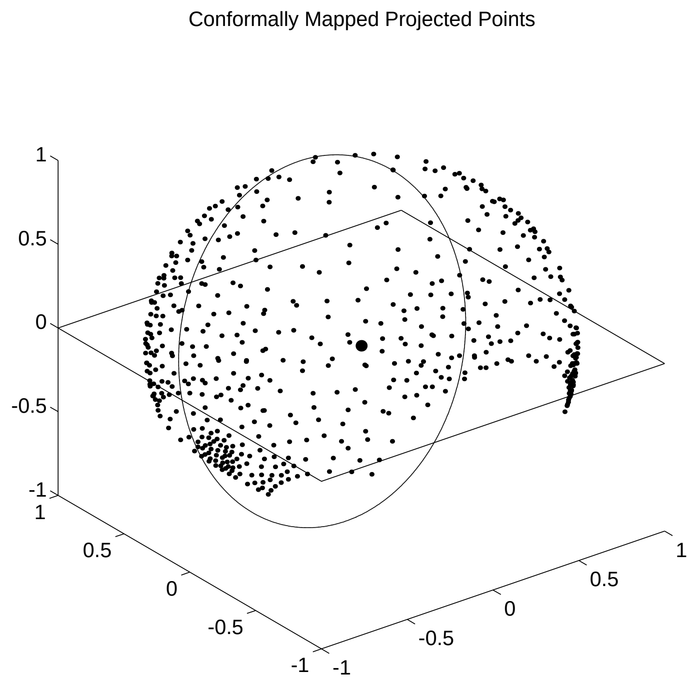

Figure 4: Conformally mapped points, with separating great circle. The centerpoint is now at the origin.

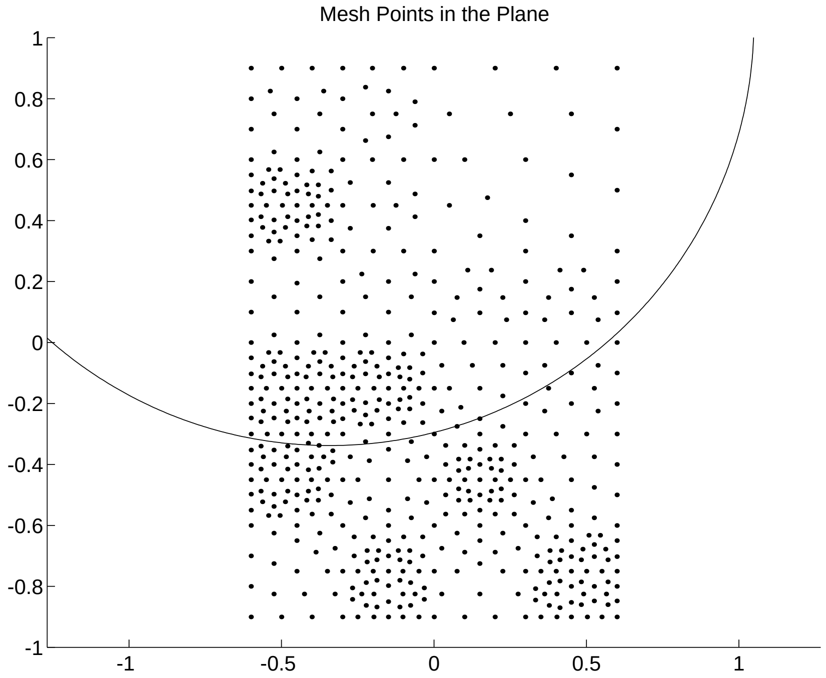

Figure 5: The separating circle projected back to the plane.

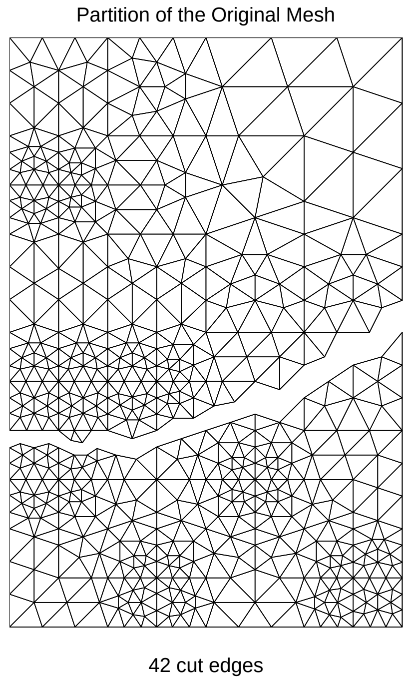

Figure 6: The edge separator induced by the separating circle. A vertex separator can also be extracted, as explained in the text.

surface of the sphere so that the center point becomes the origin. The dilation can be described as a scaling in $\mathbb{R}^d$: project the rotated points stereographically down to $\mathbb{R}^d$; scale the points in $\mathbb{R}^d$ by a factor of $\sqrt{(1-r)/(1+r)}$; and project the scaled points up to the unit sphere in $\mathbb{R}^{d+1}$ again. (See Figure 4.)

- **Find Great Circle.** Choose a random great circle (i.e., d-dimensional unit sphere) on the unit sphere in $\mathbb{R}^{d+1}$. (See Figure 4.)
- **Unmap and Project Down.** Transform the great circle to a circle in $\mathbb{R}^d$ by undoing the dilation, rotation, and stereographic projection. (See Figure 5.)
- **Convert Circle to Separator.** The vertex separator C is the vertices whose disks in the neighborhood representation (in $\mathbb{R}^d$) either (i) intersect the separating circle, or (ii) are smaller than the separating circle and would intersect it if magnified by a factor of α. The two sets A and B are the remaining vertices whose disks lie inside and outside the circle respectively. (Figure 6 shows an edge separator rather than a vertex separator.)

A *centerpoint* of a given set of points is a point (not necessarily one of the given points) such that every (hyper)plane through the centerpoint divides the given points approximately evenly (in the ratio $d:1$ or better, in $\mathbb{R}^d$). Every finite point set in $\mathbb{R}^d$ has a centerpoint, which can be found by linear programming [13, Section 4]. After the projection and conformal mapping, the origin of $\mathbb{R}^{d+1}$ is a centerpoint for the mesh points. Therefore the mapped points are divided approximately evenly by every plane through the origin—that is, by every great circle on the unit sphere in $\mathbb{R}^{d+1}$.

Every great circle determines a separator C that satisfies all the conclusions of Theorem 1 except the last one, on the size of the separator. Miller et al. show that the average size of the separators determined by all the great circles is as stated in the theorem, and therefore that a randomly chosen great circle probably gives a separator within a constant factor of the desired size.

3 Practical implementation

3.1 The mesh and its separators

The geometric separator theorem guarantees the quality of a partition if the mesh satisfies a geometric condition such as a bound on the aspect ratio of its elements. However, the geometric conditions only appear in the guarantee, not in the algorithm itself. The algorithm can be run on any mesh, with no requirements on its geometry. In practice, we observe that it generates good partitions even for meshes with badly-shaped elements. Somewhat surprisingly, it even does a reasonably good job of partitioning "$2\frac{1}{2}$-dimensional" meshes, which are meshes of triangular elements that approximate the surface of an object in 3-space.

The theorem describes a vertex separator in terms of the disks of a neighborhood system that defines the mesh. The implementation takes a simpler approach that doesn't require the neighborhood system. it just divides the vertices into those inside and those outside the separating circle. Such a vertex partition (or an *edge separator*, which is the set of edges that cross the cut) is often the goal in applications to parallel computation.

For applications like nested dissection that require a vertex separator, we compute the vertex separator from the edge separator as follows. Consider the graph G consisting only of the separating edges and their endpoints. Any vertex cover of G (that is, any set of vertices that includes at least one endpoint of every edge in G) is a vertex separator for the mesh. Since G is bipartite, we can compute the smallest vertex cover efficiently by bipartite matching [12].

The algorithm can be used to find other kinds of separators as well (though our software only includes vertex separators and edge separators). A separating set of mesh elements can be found directly from the separating circle. A partition of the mesh elements into two equal-size sets with small boundary can be found either from the separating circle, or by applying the geometric separator algorithm to a geometric dual of the mesh.

The separating circle does not necessarily split the mesh exactly in half. In theory, the centerpoint construction guarantees a splitting ratio no worse than $d + 1 : 1$; as described in Section 3.3, we actually use an approximate centerpoint construction with an even weaker guarantee. However, we observe that our approximate centerpoints nearly always lead to splits much better than the theory predicts. We almost never see splits as bad as $2 : 1$ in three dimensions, and most splits are less than 20% away from even.

We modify the splits to be exactly even, within one vertex. We do this by shifting the separating plane (in $\mathbb{R}^{d+1}$) away from the origin, in the direction normal to the plane, until it evenly splits the mapped points on the sphere. Thus the separator is a circle, but not a great circle, on the unit sphere in $\mathbb{R}^{d+1}$; this still projects back to a circle in $\mathbb{R}^d$. Our experiments show that this balancing usually affects the separator size very little. Intutitively, this is because the local geometry of a well-shaped mesh changes relatively smoothly, so a small change in the cut does not dramatically change the number of edges that cross it.

3.2 Representations

Our implementation uses very simple data structures. We never need to represent the neighborhood system or the overlap graph per se, nor do we ever use the overlap-graph parameters k and α. Most of the algorithm does not even need to know the edges of the mesh, but just manipulates the coordinates of the vertices as points in $\mathbb{R}^d$ and $\mathbb{R}^{d+1}$ (that is, as vectors). The original input points in $\mathbb{R}^d$ are scaled (isotropically) and translated to have coordinates between -1 and 1.

The implementation never actually computes a separating circle, line, or hyperplane (except to draw pictures). Rather, we represent a separating plane by its unit normal vector (in $\mathbb{R}^d$ for line separators, in $\mathbb{R}^{d+1}$ for separating hyperplanes of the mapped points). There are infinitely many planes normal to a given vector, but the one we intend is always the one that divides the points evenly. If v is the normal vector and $p_1, \ldots, p_n$ are the points (as row vectors), then the partition is into those points for which the inner product vp_i^T is less than its median value and those for which it is greater.

We do keep a representation of the graph (as a sparse adjacency matrix), but we only use it to measure the quality of a partition (which is the number of edges that cross the even cut), and to construct an explicit edge separator or vertex separator from a separating circle.

3.3 Centerpoints

The proof that every finite point set has a centerpoint yields a linear-programming algorithm that theoretically finds one in polynomial time, but would be very slow in practice. Instead, we use a version of a heuristic that was suggested by Miller and Teng [36] and was later analyzed by Clarkson et al. [10].

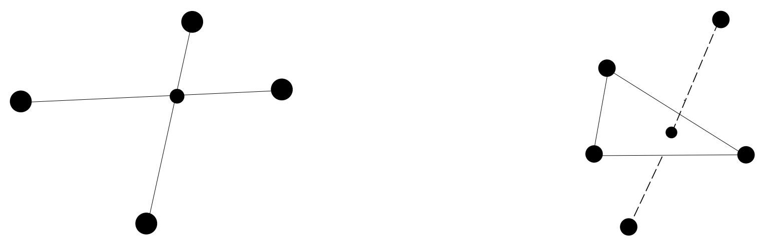

Figure 7: Radon points in two and three dimensions. The small point is the Radon point of the large points.

The heuristic uses randomization and runs in linear time in the number of sample points. It finds an approximate centerpoint by repeatedly finding *Radon points* of small point sets.

Point q is a *Radon point* [11] of a set P of points in $\mathbb{R}^d$ if P can be partitioned into two disjoint subsets P_1 and P_2 such that q lies in the intersection of the convex hull of P_1 and the convex hull of P_2. Such a partition is called a *Radon partition*. Figure 7 shows examples of Radon points in two and three dimensions.

Every set of $d+2$ points in $\mathbb{R}^d$ has a Radon point. Moreover, it can be found efficiently, as follows. Let $P = \{p_1, ..., p_{d+2}\}$, where $p_i = (p_i^1, ..., p_i^d)$. Consider the system of $d+1$ homogeneous linear equations

$$\sum_{i=1}^{d+2} \alpha_i = 0 = \sum_{i=1}^{d+2} \alpha_i p_i^j \qquad (1 \le j \le d)$$

in the variables $\alpha_1, \ldots, \alpha_{d+2}$. Since there are more variables than equations, there is a nonzero solution $(\alpha_1, \ldots, \alpha_{d+2})$. Let $U = \{i : \alpha_i > 0\}$ be the set of indices of the positive α_i's, and let $c = \sum_{i \in U} \alpha_i = \sum_{i \notin U}(-\alpha_i)$. Then we can express a single point q as a convex combination of two disjoint subsets of P:

$$q = \sum_{i \in U} (\alpha_i / c) p_i = \sum_{i \notin U} (-\alpha_i / c) p_i.$$

Finding q just requires computing a null vector α of the $d+1$ by $d+2$ matrix of the homogeneous linear system.

The idea of the centerpoint heuristic is to repeatedly replace randomly chosen groups of $d+2$ points with their Radon points. Eventually the set is reduced to a single point, which is the approximate centerpoint. (Since a d-dimensional mesh uses a centerpoint in $d+1$ dimensions, the Radon reduction actually uses groups of $d+3$ points.) We implement a simple version of this; some alternatives are discussed in Section 3.8.

3.4 Geometric sampling

We use *geometric sampling* to reduce the size of the centerpoint problem for efficiency. That is, we run the centerpoint heuristic on a randomly chosen sample of the input points. Theoretically, the size of the sample necessary for a good approximation should depend on the dimension but not on the number of mesh points [35]. We find empirically that a sample of about a thousand points suffices in two or three dimensions. We find separators for a few different approximate centerpoints, derived from different random samples, and keep the best one. Since our centerpoint approximation seems very good in practice, and since it is an expensive part of the computation, we let the number of approximate centerpoints grow only logarithmically with the total number of random separator trials.

3.5 Great circles

After the points in $\mathbb{R}^d$ are mapped to the surface of the sphere in $\mathbb{R}^{d+1}$, we expect a random great circle to induce a good partition. In fact, it pays to spend some effort looking for an above-average circle. For each approximate centerpoint, we generate several circles at random and use the best one. A convenient way to generate great circles uniformly at random is to choose normally distributed random coefficients for the vector orthogonal to the plane of the great circle [30].

A special case of a separating circle is a separating line: a line in $\mathbb{R}^d$ is the projection of a circle through the north pole of the unit sphere in $\mathbb{R}^{d+1}$. Our implementation searches explicitly for a separating line as well as for a separating circle; this improves its performance on some regular meshes. The coordinate bisection methods of Heath, Raghavan, and others [24, 39, 42] also use separating lines. Teng [41, Section 5.4] gives an example of a mesh that has a good separating circle but no good separating line.

We let the user specify how many randomly generated separators to try. Of the specified number t of trials, we allocate a number proportional to $t^{d/(d+1)}$ to separating lines and the rest to separating circles. We allocate the circle trials among a number proportional to $\log t$ of approximate centerpoints. The default is to use 30 trials, which includes (in two dimensions) 6 lines and 2 centerpoints with 12 circles each.

3.6 Inertial weighting

The random choice of a separating circle or line can be improved by biasing the normal vector in the direction of the moment of inertia of the points. The idea of inertial weighting (in one form or another) has been suggested in conjuction with geometric coordinate bisection by several people [14, 42]. Gremban, Miller, and Teng [23] proved that one version of inertial weighting reduces the expected size of the separators in Theorem 1. We use inertial weighting (much more aggressively than the version analyzed by Gremban et al.) in choosing both the separating great circles in $\mathbb{R}^{d+1}$ and the separating lines in $\mathbb{R}^d$. For great circles, we simply weight our random choice of normal vector by a power (actually the square) of the inertial matrix P^TP, where P is the matrix whose rows are the coordinates of the points after conformal mapping on the unit sphere in $\mathbb{R}^{d+1}$ Thus we generate a random unit vector u, and take our separating hyperplane to be normal to $(P^TP)^2u$. For separating lines in $\mathbb{R}^d$, we weight according to a power of the matrix of coordinates that goes as the inverse of the number of choices we make—if we choose only one line, it is exactly normal to the moment of inertia, which is the first singular vector of the coordinate matrix.

3.7 Matlab implementation notes

Most of our partitioner's basic operations are from linear algebra, which makes Matlab a natural choice of language for experimental implementation. Matlab's interpreted environment and visualization tools make it easy to experiment with variations of the algorithm. The code is written in a data-parallel "vectorized" style for efficiency (since explicit loops are slow in the interpreter); this also simplifies the

Table 1: Subroutines of the geometric partitioner.

`geopart`	Geometric separator for a mesh.
`  stereoup`	Stereographic projection from plane to sphere.
`  centerpoint`	Approximate centerpoint of a set of points.
`    radon`	Radon point of a set of $d+2$ points.
`  conmap`	Conformal map on the surface of the sphere.
`    reflector`	Orthogonal transform to put point on axis.
`    stereodown`	Stereographic projection from sphere to plane.
`    stereoup`	Stereographic projection from plane to sphere.
`  sepcircle`	Separating great circle.
`    sepquality`	Number of edges cut by separator.
`      partition`	Partition points by a plane or line.
`  sepline`	Separating line.
`    sepquality`	Number of edges cut by separator
`      partition`	Partition points by a plane or line.
`vtxsep`	Convert partition to vertex separator.

process of porting the code to a parallel machine. Versions of this code have been translated to NESL [6] and Connection Machine Fortran.

To illustrate how the pieces of the partitioner fit together, and to assist the experimentally inclined reader, we discuss some of the details of the code in this section. The Appendix includes a short version of the code, and describes how to obtain the complete code by anonymous ftp.

Outline of subroutines. The top-level call `p = geopart(A,xy)` returns a list of the vertices on one side of the cut for a mesh whose adjacency structure is A and whose vertex coordinates are xy. Optional input arguments specify the number of random trials to make, and some visualizations and statistics. Optional return values are the separating circle or line, and the set of edges that cross the cut. Table 1 shows the tree of subroutines called by `geopart`.

Data structures (`geopart`). A point in $\mathbb{R}^d$ is a row vector, and a set of n points is an $n \times d$ matrix. Names that begin "xy" are in the d-dimensional mesh space, and names that begin "xyz" are in the $d+1$-dimensional mapped space.

A partitioning line or plane is represented by its normal vector; this is not ambiguous because, among the family of parallel planes, we always intend the one that divides the points evenly.

The mesh edges are represented by the adjacency matrix A of the graph. This symmetric matrix has $A(i,j) \neq 0$ if (i,j) is an edge, and $A(i,j) = 0$ otherwise. Matlab supplies sparse data structures and operations for this matrix invisibly to the user [22]. The adjacency matrix is only used in routines `sepquality` and `vtxsep` (and for visualization).

Matrix arithmetic (`stereoup, stereodown`). Most of the manipulations of point sets are done without loops, using matrix arithmetic. The stereographic projection and its inverse are examples.

Null vectors (`centerpoint, radon`). Each Radon reduction computes a null vector of a small matrix. The null vector comes from the built-in Matlab function `null`, which computes a null space basis by singular value decomposition.

Householder matrices (`conmap, reflector`). The conformal mapping on the sphere in $d+1$-space consists of an orthogonal transformation to move the centerpoint onto the axis, a stereographic projection down to d-space, a scaling, and a stereographic projection back up to $d+1$-space. The orthogonal transformation is a Householder reflection, computed by Matlab's QR factorization. The rest is matrix arithmetic.

Random directions, inertial weighting, and SVD (`sepcircle, sepline`). Separating lines and planes are represented by their normal vectors. We generate uniformly distributed random normal vectors (or directions) by generating vectors with independent normally distributed components, using Matlab's `randn`. We implement inertial weighting by multiplying the random direction by a power of the inertia matrix $M = P^T P$, where P is the matrix whose rows are the points. For separating planes in $d+1$-space (in `sepcircle`), we use the second power and compute M^2 directly. For separating lines in d-space (in `sepline`), the exponent depends on the number of trial lines. In this case we compute a fractional power of M from the singular value decomposition of P.

Sparse matrix manipulation (`partition, sepquality`). The sparse adjacency matrix A enters the partitioning computation only when we compare the quality of the randomly generated trial separators. (If we only made one trial, we wouldn't need A at all.) Subroutine `sepquality` calls `partition` to divide the points about the median of their inner products with the normal vector into sets a and b. Then the only reference to A is the one-liner that counts the crossing edges by counting nonzeros in rows from a and columns from b, namely "`cutsize = nnz(A(a,b))`".

From edge to vertex separators (`vtxsep`). This function converts the partition from `geopart` (or any partition, for that matter) into a vertex separator for the graph by finding a minimum bipartite covering as described in Section 3.1. We compute the covering from Matlab's built-in Dulmage-Mendelsohn decomposition, `dmperm`.

3.8 Possible improvements

Finally, we list some ideas that could lead to further improvement of the geometric partitioner.

Variants of fast centerpoint. Our Matlab experiments suggest that the simplest implementation of approximate centerpoint is the method of choice both for speed and quality. We choose a random sample of the input points (without repetition), place them onto a queue, and then repeatedly remove the first $d+3$ points from the queue and add their Radon point to the end of the queue. This performs a $d+3$-ary tree of Radon reductions, with the sample points at the leaves and the approximate centerpoint at the root. The sample size is at most $(d+3)^4$ (that is, 625 for $d=2$ or 1296 for $d=3$), and is congruent to 1 (modulo $d+2$).

The theoretical results of Clarkson et al. [10] suggest that the probability of returning a bad centerpoint decreases double-exponentially in the number of levels of Radon reduction. We use (approximately) four levels. A five-level reduction would need 3125 points for $d = 2$ or 7776 points for $d = 3$. We experimented with a variant (suggested by Clarkson et al.) that allows more levels of reduction without using more sample points. The idea is to reduce according to a directed acyclic graph instead of a complete $d+3$-ary tree. Let P_0 be a sample of L points. To construct P_h, choose L random $(d+3)$-tuples from P_{h-1} (with replacement) and let P_h be the set of Radon points of these L tuples. We can parameterize the centerpoint algorithm by L and h; the total complexity is $O(Lh^3)$. Our experiments suggest $1000 \leq L \leq 1200$ and $4 \leq h \leq 8$ work well. While this doesn't seem to beat the simpler method in Matlab, it may be useful in some settings.

From sorting to median finding. To force an even partition, we need to find the median of the dot products of the points with the normal to the partitioning plane. Our implementation uses Matlab's built-in median function, which is based on sorting. Theoretically, this is overkill; sorting takes $O(n \log n)$ time and a median can be found in linear time. In some settings (especially on parallel machines), it may be best to use a randomized median-finding algorithm [18, 28]. Indeed, one can even find an approximate median by using the one-dimensional version of the approximate centerpoint algorithm [10], which amounts to repeated median-of-three reduction.

Faster quality testing. We measure the quality of a trial separating sphere by counting the number of graph edges that cross the cut it induces. This is the only phase of the algorithm that needs to manipulate the edges (as opposed to the vertex coordinates), and it typically takes about half to two thirds of the total time.

One idea for speeding this up is to use geometric sampling again. For example, instead of examining all the edges we could look only at a random sample of $|E|^{1/(d+1)} \log |E|$ of them. This idea could be used in both sequential and parallel implementations.

A second possibility is to preprocess the mesh to represent the graph more compactly, for example, as a neighborhood system. Since a simplicial mesh has about 3 times as many edges as vertices in two dimensions or 7 times as many in three dimensions, the neighborhood system is significantly smaller than the mesh, and a quality test could be implemented by just counting intersections between neighborhood disks and the separating sphere.

Local optimization for great circles. Once the centerpoint is determined and the points are conformally mapped on the unit sphere in $\mathbb{R}^{d+1}$, each trial separating circle is selected independently at random (from an inertially biased distribution). Instead, one can imagine trying to improve each trial circle locally. Consider the quality of a trial circle, as a function of its normal vector. This is a real-valued function defined on the surface of the unit sphere in $d+1$-space. The function is not smooth—in fact, it is piecewise constant—but it might be possible to smooth it on a fine scale and then use continuous optimization methods to find a local minimum on a coarser scale. Our preliminary experiments show that this idea often improves the quality of a partition, sometimes by a significant amount.

Relaxing the 50-50 split. Most applications do not require the vertex partition to be exactly even. It may be worthwhile to search in the vicinity of an exact cut—for example, by shifting the cutting hyperplane or dilating the separating circle—for a cut whose balance is slightly uneven but whose overall quality is higher. The user would probably have to supply the definition of "overall quality", since the tradeoff between load balancing (even partition) and communication cost (small cut) depends on the application.

4 Experimental results

To assess the quality of the geometric algorithm's partitions, we compared them to coordinate bisection [24, 39, 42] and to spectral bisection [3, 25, 37] on several sample meshes.

Table 2 lists the meshes. TAPIR is a test case from a 2D mesh generation algorithm of Bern, Mitchell, and Ruppert [5] that produces triangles with sharp angles but no obtuse angles. We expect this to be a difficult mesh for the geometric algorithm, because it has many triangles with large aspect ratios. AIRFOIL2 and AIRFOIL3 are highly graded meshes of well-shaped 2D triangles around cross sections of airfoils, from Barth and Jesperson [4]. TRIANGLE is a 2D mesh of equilateral triangles, all the same size, generated by `gridt` in Matlab. PWT is a mesh of 3D elements that discretize a thin shell. We expect this to be difficult for the geometric algorithm to separate well, because its best separators should be like those of a 2D mesh but the algorithm treats it as a 3D mesh. BODY is another 3D mesh with some "thin shell" parts. We obtained these two meshes from Horst Simon at NASA. WAVE is a highly graded mesh that fills the space around an object in 3D, which we obtained from Steve Hammond at NCAR.

Table 3 shows the number of edges cut for a balanced two-way split, as found by each of the three methods. We used Matlab to implement coordinate bisection, and we used Hendrickson and Leland's Chaco package [26] to find the spectral bisections. (Chaco also implements several other bisection methods that we did not use here.) A parameter to the geometric algorithm is the number of random trials of great circles to make. The "default geometric" column reports results for 30 trials, which is the default of our Matlab code; "best geometric" reports the results for 7000 trials. (Each "default geometric" number is actually the median result of 31 separate experiments of 30 trials each.)

The results indicate that the geometric cuts are consistently smaller than the coordinate-bisection cuts. In most cases, the geometric cuts are also smaller than the spectral ones. The significant exceptions are PWT and BODY, the thin shells in 3D. These may be difficult cases for the geometric algorithm because they really should be treated as two-dimensional meshes in some sense.

It is hard to make meaningful comparisons of the running times of the various algorithms, since there are many different versions and choices of parameters for all of them, and also because the Matlab implementation runs in an interpreted environment. For a rough comparison, we note that finding a 2-way partition for the AIRFOIL3 mesh takes 46 seconds with the default Matlab geometric code, 5.9 seconds with a similar C geometric code, 0.83 seconds with a C geometric code that only computes one cutting circle, and 10.1 seconds with a good C spectral code.[1]

We do not mean to suggest that the geometric algorithm is the last word in mesh partitioning; several researchers have proposed refinements to spectral partitioning [7, 8, 25], and some purely combinatorial methods such as Hendrickson and Leland's multilevel Kernighan-Lin [27] look very promising. However, we believe this data shows that geometric partitioning is at least competitive with other modern graph partitioning methods.

[1]The Matlab code used the default 30 trials, including two centerpoints. The "similar" C code also used 30 trials with two centerpoints. The spectral time is from Chaco, using its multilevel RQI/Symmlq eigensolver and no Kernighan-Lin postprocessing. The experiments were run on an unloaded Sparc-10. All times are the median of three runs, and do not include input/output.

Mesh	Description	Mesh Type	Grading	Vertices	Edges
TAPIR	Cartoon animal	2-D triangles, sharp angles	8.5×10^4	1024	2846
AIRFOIL2	Three-element airfoil	2-D triangles	1.3×10^5	4720	13722
TRIANGLE	Equilateral triangle	2-D triangles, all same size	1.0×10^0	5050	14850
AIRFOIL3	Four-element airfoil	2-D triangles	3.0×10^4	15606	45878
PWT	Pressurized wind tunnel	Thin shell in 3-space	1.3×10^2	36519	144794
BODY	Automobile body	3-D volumes and surfaces	9.5×10^2	45087	163734
WAVE	Space around airplane	3-D volumes and surfaces	3.9×10^5	156317	1059331

Table 2: Test problems. "Grading" is the ratio of longest to shortest edge lengths.

Mesh	Spectral	Coordinate Bisection	Default Geometric	Best Geometric
TAPIR	59	55	37	32
AIRFOIL2	117	172	100	93
TRIANGLE	154	142	144	142
AIRFOIL3	174	230	152	148
PWT	362	562	529	499
BODY	456	953	834	768
WAVE	13706	9821	10377	9773

Table 3: Cut size for two-way partitions.

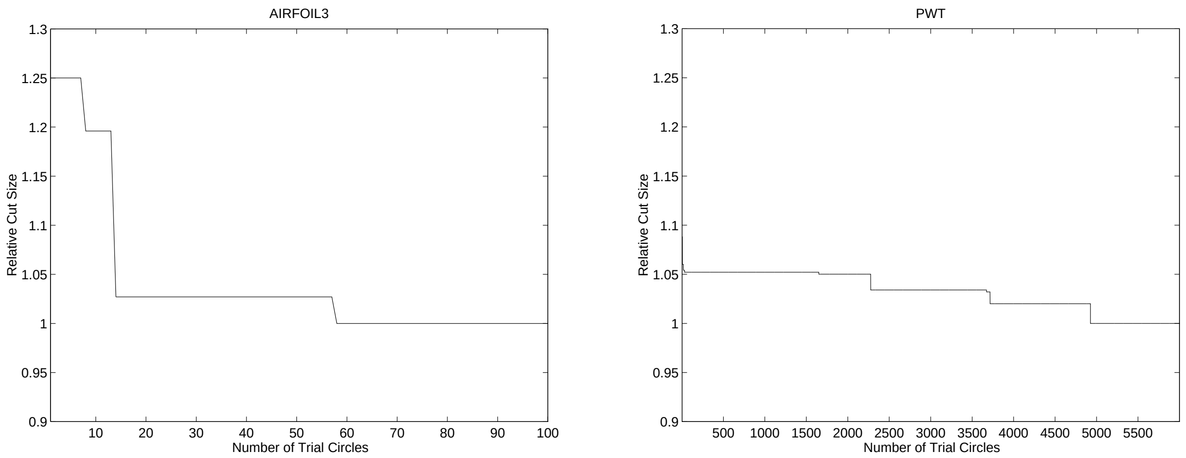

Figure 8: Improvement in relative cut size with increasing number of trials.

Mesh	After 10 Trials	After 100 Trials	Last Improvement
TAPIR	1.28	1.00	49
AIRFOIL2	1.22	1.04	1636
TRIANGLE	1.07	1.01	109
AIRFOIL3	1.20	1.00	58
PWT	1.06	1.05	4927
BODY	1.14	1.02	2989
WAVE	1.16	1.06	498

Table 4: Improvement in relative cut size with increasing number of trials.

Mesh	Spectral	Coordinate Bisection	Default Geometric
TAPIR	1278	1387	1239
AIRFOIL2	2826	3271	2709
TRIANGLE	2989	2907	2912
AIRFOIL3	4893	6131	4822
PWT	13495	14220	13769
BODY	12077	22497	19905
WAVE	143015	162833	145155

Table 5: Cut size for 128-way partitions.

Mesh	Minimum Degree	Sparspak	Spectral	Coordinate Bisection	Default Geometric	Partial Geometric
TAPIR	7786	15402	12214	12282	10314	10094
AIRFOIL2	103207	146894	102248	124075	96901	103163
TRIANGLE	130587	128995	127785	122539	123560	130106
AIRFOIL3	409392	657687	418840	472222	389232	405918
PWT	1424987	1631592	1441153	1545975	1503498	1576271

Table 6: Nested dissection: Fill.

Mesh	Minimum Degree	Sparspak	Spectral	Coordinate Bisection	Default Geometric	Partial Geometric
TAPIR	66	103	82	83	66	68
AIRFOIL2	319	415	189	287	192	201
TRIANGLE	383	269	233	223	226	231
AIRFOIL3	526	837	346	440	321	329
PWT	960	822	618	713	651	668

Table 7: Nested dissection: Height.

The data in Table 3 suggest that 30 random trials are usually enough to get close to the best separator that the geometric method will find. Table 4 and Figure 8 explore this in more detail. For each mesh, we ran 6000 random trials. The table reports the smallest cut seen in the first 10 trials, the smallest cut seen in the first 100 trials, and the number of the trial in which the smallest cut of all was first seen ("last improvement"). The cut sizes are normalized so that the smallest geometric cut seen for each mesh had size 1. Figure 8 plots the smallest cut size against trial number for two of the meshes. (Note the different scales in the two plots.) The geometric algorithm's difficulty with PWT, the thin shell in 3D, is evident here: though the improvement between 10 and 6000 trials is only 6%, small improvements continue to occur almost to the very end. On the other hand, most of the meshes "settle down" within at most a few hundred trials, and all of them are within 6% of the 6000-trial minimum after 100 trials.

Table 5 shows the total number of edges cut by using the three algorithms recursively to split the mesh into 128 pieces. For the geometric algorithm, we used the default of 30 random trials. It is striking that, for most of the problems, the cuts from the various methods differ much less in quality for 128-way than for 2-way partitions. The geometric and spectral algorithms give extremely similar sizes for all but the BODY mesh (for which we don't have an explanation of the difference).

Tables 6 and 7 illustrate "geometric nested dissection," which uses balanced 2-way geometric partitioning recursively to order a symmetric, positive definite matrix for Cholesky factorization. We tabulate both the fill, which measures the amount of storage needed for the Cholesky factor, and the height in vertices of the elimination tree, which is the number of parallel elimination steps to compute the factor with an unlimited number of processors. "Default geometric" uses the geometric algorithm to partition the graph all the way down to fragments of 3 vertices or less; "partial geometric" uses the geometric algorithm down to fragments of 100 vertices and then uses minimum degree on the fragments. We also tabulate fill and height for Sparspak's nested dissection routine [19], for Matlab's minimum degree routine [22], and for nested dissection with separators from spectral partitioning as described by Pothen et al. [38]. Sparspak's nested dissection routine uses a fast but fairly simple partitioning algorithm, which generally does not perform as well as the newer methods for either height or fill. For most of the large geometric problems there is little to choose between minimum degree and nested dissection in terms of fill, but nested dissection with the newer partitioners usually gives better height than minimum degree. Among the various spectral and geometric partitioners there is no clear winner for either height or fill.

5 Conclusions

We have described a geometric partitioning algorithm that is fairly simple to implement and seems to give excellent results on meshes from graded finite-element discretizations of 2- and 3-space. Our reference implementation is in Matlab, which makes experimenting with different versions of the algorithm quite easy. We have also implemented versions of the geometric partitioner in C and Fortran.

A chief application of graph partitioning is to distribute a computational mesh across a distributed-memory parallel machine. Can the partition itself can be found in parallel? This is challenging because most partitioners make heavy use of the edges of the graph, and therefore require a lot of communication unless most adjacent vertices share the same processor—that is, unless a good partition is already known. We expect the geometric partitioner to be reasonably efficient in parallel, because almost none of the data manipulation involves the edges. (Coordinate bisection shares this desirable property, as Heath and Raghavan's parallel implementation shows [24].) We have implemented parallel versions of the geometric partitioner in NESL [6] and Connection Machine Fortran.

An open problem is how best to handle such $2\frac{1}{2}$-dimensional meshes as our example Pwt. One possibility is to combine the geometric and spectral partitioning methods, as recently suggested by Chan, Gilbert, and Teng [7].

Appendix: Obtaining the Codes

A slightly simplified version of the Matlab geometric separator code is given below. The complete version of the Mesh Partitioning Toolbox also contains Matlab implementations of spectral bisection [37] and geometric spectral bisection [7]. It includes both edge and vertex separators, recursive bipartition, nested dissection ordering, visualizations and demos, and some sample meshes. Table 8 lists the main routines in the toolbox. The complete toolbox is available by anonymous ftp from machine `ftp.parc.xerox.com` as file `/pub/gilbert/meshpart.uu`.

Table 8: Mesh Partitioning Toolbox.

Partitioning methods.	
`geopart`	Geometric.
`specpart`	Spectral.
`coordpart`	Coordinate bisection.
`gspart`	Geometric spectral.
Multiway partitions.	
`dice`	Use any 2-way partitioner to get a multiway partition.
`geodice`	Recursive geometric partitioning.
`specdice`	Recursive spectral partitioning.
`gsdice`	Recursive geometric spectral partitioning.
Vertex separators.	
`vtxsep`	Convert a 2-way partition to a vertex separator.
`geosep`	Vertex separator from geometric partitioning.
`specsep`	Vertex separator from spectral partitioning.
Nested dissection.	
`ndperm`	Use any 2-way partitioner for nested dissection.
`geond`	Geometric nested dissection ordering.
`specnd`	Spectral nested dissection ordering.
`gsnd`	Geometric spectral nested dissection ordering.
`analyze`	Predict fill, opcount, etc. for an elimination ordering.
Meshes and graph generators.	
`meshes.mat`	Three sample meshes with coordinates.
`grid5`	2D square 5-point mesh.
`grid7`	2D square 7-point mesh.
`grid9`	2D square 9-point mesh.
`gridt`	2D triangular mesh.
`grid3d`	3D cubical mesh.
`grid3dt`	3D cubical simplicial mesh.
`badmesh`	A mesh that has no good straight-line cut.
Visualization and graphics.	
`gplotpart`	Draw a 2-way partition.
`gplotmap`	Draw a multiway partition.
`highlight`	Draw a mesh with some vertices highlighted.
`gplotg`	Draw a 2D or 3D mesh (replaces Matlab's `gplot`).
`etreeplot`	Draw an elimination tree (replaces Matlab's `etreeplot`).
Utilities.	
`components`	Connected components of a graph.
`cutsize`	Find or count edges cut by a partition.
`other`	Other side of a partition, or change representations.
`intersection`	Intersection of two sets.
`union`	Union of two sets.
`ranks`	Replace matrix elements by their ranks.
`fiedler`	Fiedler vector of a graph.
`laplacian`	Laplacian matrix of a graph.

```
function [part1,part2] = geopart(A,xy,ntries)
% GEOPART : Geometric partition of a finite element mesh.
%
% [part1,part2] = geopart(A,xy,ntries) returns a vertex partition of
% the mesh whose structure is A and whose coordinates are the rows of xy.
% ntries (optional, default 30) is the number of separating circles to try.

[npoints,dim] = size(xy);
if nargin < 3, ntries = 30; end;

% The "ntries" tries will include "nouter" centerpoint computations,
% each with "ninner" hyperplanes;
% and also "nlines" cut planes in the dim-dimensional mesh space.
% The following division is ad hoc but seems about right.
nlines = floor((ntries/2) ^ (dim/(dim+1)));
nouter = ceil(log(ntries-nlines+1)/log(20));
ninner = floor((ntries-nlines) / nouter);

% Size of sample for approximate centerpoint computation.
csample = min(npoints,(dim+3)^4);

% Scale points to have mean zero and max coordinate magnitude 1.
xy = xy - ones(npoints,1)*mean(xy);
xy = xy / max(max(abs(xy)));

% Project points stereographically up to the sphere.
xyz = stereoup(xy);

circlequality = Inf;
for i = 1:nouter
    cpt = centerpoint(xyz,csample);
    xyzmap = conmap(cpt,xyz);
    [greatcircle,gcquality] = sepcircle(A,xyzmap,ninner);
    if gcquality < circlequality
        circlequality = gcquality;
        bestcircle = greatcircle;
        bestcpt = cpt;
    end;
end;

% Also try separating with a straight line in mesh space.
[bestline,linequality] = sepline(A,xy,nlines);

if linequality <= circlequality
    [part1,part2] = partition(xy,bestline);
else
    [part1,part2] = partition(conmap(bestcpt,xyz),bestcircle);
end
```

```
function xyz = stereoup(xy)
% STEREOUP : Stereographic projection from plane to sphere.
%
% xyz = stereoup(xy):
% Project points xy stereographically from the dim-1 dimensional space
% onto the unit sphere in dim dimensions.
% The number of dimensions of the mesh is the number of columns of xy.
% This is the inverse of stereodown.

[n,dim] = size(xy);
dim = dim+1;        % dim is the number of dimensions of the sphere.

edim = [zeros(n,dim-1) ones(n,1)];
normsquared = [(xy.^2) ones(n,1)] * ones(dim,dim);

% Each row of "edim" is the dim-dimensional unit vector [0 0 ... 0 1].
% "normsquared" is the square of the norm of (xy - edim), if we think
% of xy as embedded in dim dimensions.
%
% Thus xyz = edim + 2*(xy - edim) ./ normsquared.

xyz = edim + 2 * [xy -ones(n,1)] ./ normsquared;
```

```
function xy = stereodown(xyz)
% STEREODOWN : Stereographic projection from sphere to plane.
%
% xy = stereodown(xyz):
% Project points xyz stereographically from the unit sphere in dim dimensions
% to the dim-1 dimensional space.
% The number of dimensions of the mesh is the number of columns of xyz minus 1.
% This is the inverse of stereoup.

[n,dim] = size(xyz);   % dim is the number of dimensions of the sphere.

xy = xyz(:,1:dim-1) ./ ((ones(n,1)-xyz(:,dim)) * ones(1,dim-1));
```

```
function c = centerpoint(xyz,n)
% CENTERPOINT : Approximate centerpoint in any number of dimensions.
%
% c = centerpoint(xyz,n) uses repeated Radon reduction to compute
% an approx centerpoint for the points in xyz, using a sample of size n.

% Determine sample size n, and round it to be congruent to 1 mod d+1.
[npoints,d] = size(xyz);
n = min(n,npoints);
n = (d+1) * floor((n-1)/(d+1)) + 1;

% Sample points without replacement.
xyzsize = ceil(n*(1+1/(d+1)));
xyzs = zeros(xyzsize,d);
sample = randperm(npoints);
sample = sample(1:n);
xyzs(1:n,:) = xyz(sample,:);

% Perform Radon reduction according to a full d+2-ary tree.
queuehead = 1;
queuetail = n;
while queuehead < queuetail
    queuetail = queuetail+1;
    xyzs(queuetail,:) = radon(xyzs(queuehead:queuehead+d+1,:));
    queuehead = queuehead+d+2;
end;

c = xyzs(queuetail,:);
```

```
function r = radon(xyz)
% RADON : Radon point of d+2 points in d dimensions.
%
%      r = radon(xyz);
%      Each row of xyz is the coordinates of a point.

[dplus2, d] = size(xyz);

% Express zero as a homogeneous linear combination of the input points.
nullvec = null([ones(d+2,1) xyz]');
nullvec = nullvec(:,1);

% The positive coefficients identify the Radon point.
positives = find(nullvec>0);
r = nullvec(positives)' * xyz(positives,:) / sum(nullvec(positives));
```

```
function [xyzmap,xymap] = conmap(c,xyz)
% CONMAP : Conformal map for geometric partitioning.
%
% [xyzmap,xymap] = conmap(c,xyz):
% Input is points xyz on a unit sphere in d-space
% (one point per row of matrix xyz),
% and a point c in d-space represented as a row vector.
%
% We compute the following conformal mapping:
%    First reflect the sphere so that c is on the last axis.
%    Second stereographically map the xyz points onto the d-1 - space.
%    Third scale the xy points so that c stretches to the origin.
%    Fourth stereog'y map the xy points back to the sphere.
%
% The first output argument is the new points on the sphere in d-space;
% the second output argument is the new points in the d-1 - space.

% Compute the reflection and stretch.
[Q,r] = reflector(c);
alpha = sqrt((1+r)/(1-r));

% Reflect.
xyzref = xyz * Q;
xyref = stereodown(xyzref);

% Stretch.
xymap = xyref / alpha;
xyzmap = stereoup(xymap);



function [Q,r] = reflector(c)
% REFLECTOR : Orthogonal transformation to put point on last axis.
%
% [Q,r] = reflector(c):
% Given a point c in d-space (as a row vector),
% return an orthogonal Q such that c*Q is on the last axis,
% and has magnitude r.  Q is a Householder reflection.

d = length(c);
p = d:-1:1;
[Q,r] = qr(c(p)');
Q = Q(p,p);
r = r(1);
```

```
function [gc,gcquality] = sepcircle(A,xyz,ntries)
% SEPCIRCLE : Good great circle for geometric partitioning.
%
% [gc,gcquality] = sepcircle(A,xyz,ntries)
% Generate a great circle on the unit sphere in d dimensions,
% trying to find one that gives a good edge separator of the graph A.
% "xyz" is the input points, conformally mapped on the unit sphere
%       in d-space so that the approximate centerpoint is at the origin.
% We return the best of ntries attempts.
%
% The circle is returned only as a direction, a d-vector normal to its plane.
% The second output is the partition quality as measured by sepquality.

[npoints,d] = size(xyz);

% For inertial weighting, we weight the randomly chosen great circles
% according to a power (say 2) of the inertial matrix of the points.

M = (xyz' * xyz) ^ 2;

% Now choose all ntries random great circles.
% Each circle is represented by a vector normal to its plane.
% Normally distributed points give vectors with uniformly distributed
% directions, which we then weight by the matrix M from above.

vv = randn(ntries,d) * M;

quality = Inf * ones(ntries,1);
for i = 1 : size(vv,1)
    v = vv(i,:);
    if norm(v) ~= 0
        quality(i) = sepquality(v,A,xyz);
    end;
end;

[gcquality,i] = min(quality);
gc = vv(i,:)';
```

```
function [bestline,linequality] = sepline(A,xy,ntries)
% SEPLINE : Good separating line (or plane) for geometric partitioning.
%
% [bestline,linequality] = sepline(A,xy,ntries)
% Generate cutting planes in the d-dimensional mesh space,
% trying to find one that gives a good edge separator of the graph A.
% "xy" is the input points, in the mesh space (not conformally mapped).
% "ntries" is the number of planes to try.
%
% We generate the planes at random, weighted toward the first
% singular vector of the matrix of coordinates.  The weighting
% becomes weaker as the number of tries goes up.
%
% "bestline" is returned as a vector normal to the best line/plane.
% "linequality" is the quality of its partition as measured by "sequality".

[npoints,d] = size(xy);

[U,S,V] = svd(xy,0);

if ntries == 1
    % With only one try, use the singular vector.
    vv = V(:,1)';
else
    % "exponent" determines the weighting.
    % This formula is ad hoc, but seems pretty good.
    exponent = 2*(d+1)/(ntries-1);
    s = diag(S).^exponent;
    W = (V * diag(s) * V');
    vv = randn(ntries,d) * W;
    rownorms = sqrt(sum((vv.*vv)'));
    vv = diag(1./rownorms) * vv;
end;

quality = Inf * ones(ntries,1);
for i = 1 : ntries
    v = vv(i,:);
    quality(i) = sepquality(v,A,xy);
    end;
end;

[linequality,i] = min(quality);
bestline = vv(i,:)';
```

```
function cutsize = sepquality(v,A,xyz)
% SEPQUALITY : Separator quality.
%
% cutsize = sepquality(v,A,xyz)
% Return the number of edges crossing a partition of the vertices of A,
% at positions xyz, by the plane described by v.

[a,b] = partition(xyz,v);
if min(length(a),length(b))
    cutsize = nnz(A(a,b));
else
    cutsize = 0;
end



function [a,b] = partition(xyz,v)
% PARTITION : Partition points by a plane.
%
% [a,b] = partition(xyz,v):
% Each row of xyz is an input point in d-space.
% Input v is a vector, giving a direction normal to the partitioning plane.
%
% The output is two vectors of integers,
% the indices of the points on each side of the plane.
% Points on the plane are assigned to balance the cut.

[n,d] = size(xyz);

v = v(:); % Make v a column vector

dotprod = xyz * v;
split = median(dotprod);
a = find(dotprod <  split);
b = find(dotprod > split);
c = find(dotprod == split);
nc = length(c);
if nc
    na = length(a);
    nca = max([ceil(n/2)-na, 0]);
    nca = min(nca,nc);
    if nca > 0
        a = [a; c(1:nca)];
    end;
    if nca < nc
        b = [b; c(nca+1:nc)];
    end;
end;
a = a';
b = b';
```

```
function [sep,parta,partb] = vtxsep(A,a,b)
% VTXSEP : Convert an edge separator (or node partition) to a vertex separator.
%
% sep = vtxsep(A,a,b):
% A is a symmetric 0-1 matrix representing an undirected graph.
% a and b partition the vertices of A.
% This function returns a vertex separator sep of minimum size
% such that a-sep and b-sep are in different components of A-sep.
%
% Optional outputs: [sep,parta,partb], where sep is as above
% and parta = a - sep, partb = b - sep.

% We use dmperm to find a maximum matching on the bipartite subgraph
% of A induced by the edges joining a to b, and then use the matching
% to construct a minimum vertex cover of that subgraph, which is the
% desired vertex separator.

A = spones(A);
aborder = a(find(max(A(b,a))));  % points of a adjacent to b
bborder = b(find(max(A(a,b))));  % points of b adjacent to a

if length(aborder) == 0

    % The parts are disconnected, so the separator is empty.
    sepa = [];
    sepb = [];

else

    % Use dmperm to find a matching of the birpartite graph.
    % The separator is points of a in the horizontal subgraph,
    % plus points of b not in the horizontal subgraph.
    [p,q,r,s] = dmperm(A(aborder,bborder));
    sepa = aborder(p(r(1):(r(2)-1)));
    sepb = bborder(q(s(2):(s(length(s))-1)));

end;

sep = [sepa sepb];

t = zeros(1,length(A));
t(a) = ones(size(a));
t(sepa) = zeros(size(sepa));
parta = find(t);

t = zeros(1,length(A));
t(b) = ones(size(b));
t(sepb) = zeros(size(sepb));
partb = find(t);
```

References

[1] Ajit Agrawal and Philip Klein. Cutting down on fill using nested dissection: Provably good elimination orderings. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*. Springer-Verlag, 1993.

[2] Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and applications. In *Proceedings of the 22th Annual Symposium on Theory of Computing*. ACM, 1990.

[3] Stephen T. Barnard and Horst D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Technical Report RNR–92–033, NASA Ames Research Center, 1992. A short version is in *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, SIAM, 1993, pp. 711–718.

[4] Timothy J. Barth and Dennis C. Jespersen. The design and application of upwind schemes on unstructured meshes. In *27th Aerospace Sciences Meeting*. AIAA, 1989.

[5] Marshall Bern, Scott Mitchell, and Jim Ruppert. Linear-size nonobtuse triangulation of polygons. In *Proceedings of the 10th Symposium on Computational Geometry*, pages 221–230. ACM, 1994.

[6] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, Cambridge, Mass., 1990.

[7] Tony F. Chan, John R. Gilbert, and Shang-Hua Teng. Geometric spectral bisection, July 1994. Manuscript.

[8] Tony F. Chan and W. K. Szeto. A sign cut version of the recursive spectral bisection graph partitioning algorithm. In *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 562–566, Snowbird, UT, June 1994.

[9] F. R. K. Chung and S.-T. Yau. A near optimal algorithm for edge separators. In *Proceedings of the 26th Annual Symposium on Theory of Computing*. ACM, 1994.

[10] K. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng. Approximating center points with and without linear programming. In *Proceedings of 9th ACM Symposium on Computational Geometry*, pages 91–98, 1993.

[11] L. Danzer, J. Fonlupt, and V. Klee. Helly's theorem and its relatives. *Proceedings of Symposia in Pure Mathematics, American Mathematical Society*, 7:101–180, 1963.

[12] A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian J. Math.*, 10:517–534, 1958.

[13] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, NY, 1987.

[14] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. Num. Meth. Eng.*, 36:745–764, 1993.

[15] C. Farhat and H. Simon. TOP/DOMDEC—A software tool for mesh partitioning and parallel processing. Technical report, NASA Ames Research Center, 1993.

[16] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. Technical Report 82CRD130, General Electric Co., Corporate Research and Development Center, Schenectady, NY, 1982.

[17] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovakian Mathematics Journal*, 23:298–305, 1973.

[18] Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18:165–172, 1975.

[19] Alan George and Joseph W. H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal on Numerical Analysis*, 15:1053–1069, 1978.

[20] J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

[21] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5:391–407, 1984.

[22] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13:333–356, 1992.

[23] Keith D. Gremban, Gary L. Miller, and Shang-Hua Teng. Moments of inertia and graph separators. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 452–461. SIAM, 1994.

[24] Michael Heath and Padma Raghavan. A cartesian parallel nested dissection algorithm, 1994. To appear in *SIAM Journal on Matrix Analysis and Applications*.

[25] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. Technical Report SAND92-1460, Sandia National Laboratories, Albuquerque, NM, 1992.

[26] Bruce Hendrickson and Robert Leland. The Chaco user's guide, Version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque, NM, 1993.

[27] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, 1993.

[28] C. A. R. Hoare. Algorithm 63 (PARTITION) and Algorithm 65 (FIND). *Communications of the ACM*, 4:321, 1961.

[29] B. W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–308, February 1970.

[30] D. E. Knuth. *The Art of Computer Programming*, volume 3 of *Computer Science and Information Processing*. Addison-Wesley, second edition, 1981.

[31] F. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[32] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. of Appl. Math.*, 36:177–189, April 1979.

[33] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, June 1986.

[34] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Finite element meshes and geometric separators. to appear 1994.

[35] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, IMA Volumes in Mathematics and its Applications. Springer-Verlag, 1993.

[36] Gary L. Miller and Shang-Hua Teng. Center points and point divisions. manuscript, School of Computer Science, Carnegie Mellon University, 1990.

[37] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11:430–452, 1990.

[38] Alex Pothen, Horst D. Simon, and Lie Wang. Spectral nested dissection. Technical Report CS–92–01, Pennsylvania State University Department of Computer Science, 1992.

[39] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[40] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[41] Shang-Hua Teng. *Points, Spheres, and Separators: A Unified Geometric Approach to Graph Partitioning.* PhD thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, August 1991.

[42] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3:457–481, 1991.