Computing Exact Discrete Minimal Surfaces: Extending and Solving the Shortest Path Problem in 3D with Application to Segmentation*

Leo Grady
Siemens Corporate Research
Department of Imaging and Visualization
755 College Road East, Princeton, NJ 08540
Leo.Grady@siemens.com

Abstract

Shortest path algorithms on weighted graphs have found widespread use in the computer vision literature. Although a shortest path may be found in a 3D weighted graph, the character of the path as an object boundary in 2D is not preserved in 3D. An object boundary in three dimensions is a (2D) surface. Therefore, a discrete minimal surface computation is necessary to extend shortest path approaches to 3D data in applications where the character of the path as a boundary is important. This minimal surface problem finds natural application in the extension of the intelligent scissors/live wire segmentation algorithm to 3D. In this paper, the discrete minimal surface problem is both formulated and solved on a 3D graph. Specifically, we show that the problem may be formulated as a linear programming problem that is computed efficiently with generic solvers.

1. Introduction

Shortest path algorithms on weighted graphs have found many applications in computer vision, including segmentation [9, 4], video summarization [12], perceptual grouping [3] and solving PDEs [14]. Since computer vision techniques have been increasingly applied to 3D data in the context of video sequences or medical acquisitions, researchers have looked for 3D extensions of many conventional 2D techniques. Although a shortest path may be found in a 3D weighted graph, the character of the path as an object boundary in 2D is not preserved in 3D. Specifically, a 3D object boundary is necessarily a (2D) surface. Therefore, extension of a shortest path approach to 3D requires computation of an appropriate surface (when the character of the path as boundary is important). We have two goals in this paper: 1) Reformulate the shortest path problem on weighted graphs to find minimal surfaces on weighted complexes (3D graphs), 2) Provide a method to solve this minimal surface problem. Our method for solving the discrete, weighted minimal surface problem will then be demonstrated in the context of 3D segmentation, since the character of the surface as an object boundary fits most naturally in the context of segmentation.

Shortest paths are used as object boundaries in several 2D image segmentation algorithms, most notably in the popular intelligent scissors/live wire algorithm [9, 4]. The intelligent scissors algorithm treats the image as a graph that is weighted to reflect intensity changes and inputs two points from a user along an object boundary. These points are then used to define the endpoints for a shortest path computation. The shortest path is then viewed as a piece of the object boundary that may be extended by the placement of additional points. Intelligent scissors has been previously applied to 3D image segmentation tasks by computing several shortest paths in 3D and using these paths to reconstruct a surface. Falcão *et al.* separate the 3D image into slabs for which the object is assumed to have constant topology. A user then employs shortest path segmentation on several cross-sections (with some constraints) which are used for surface reconstruction. Knapp *et al.* employ orthogonal cross-sections to reconstruct the surface, which may have a nontrivial topology. These cross-sections are obtained via shortest paths. In 2D intelligent scissors, shortest paths provide minimal boundaries of the segmented object. Since these extensions of intelligent scissors to 3D do not preserve this minimal character of the object boundary, we argue that the discrete minimal surfaces developed here provide a more natural extension of intelligent scissors to 3D.

Minimal surfaces have been studied extensively outside of computer vision in the fields of geometric measure theory and variational calculus [8]. However, these fields focus on finding minimal surfaces in a space that is continuous and

*Published in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition — CVPR 2006, volume 1, pages 69–78, IEEE, IEEE, June 2006.

Euclidean, rather than on the discrete, weighted lattices that arise naturally in computer vision. In order to avoid confusion with the body of existing literature, we will employ the term **minimum-weight surfaces** to refer to minimal surfaces defined on the space of discrete, weighted lattices. Although lattices are the most relevant structure for computer vision, the techniques developed here also apply to more general discrete structures.

The shortest path problem requires specification of additional constraints in order to avoid the solution of a null path. Additional constraints are typically included in one of two ways:

1. The shortest path is required to enclose one specified region of space while excluding a second specified region (i.e., separating the two regions),

2. The shortest path is required to have a specified boundary (i.e., endpoints).

These methods for specifying constraints for the minimal surface problem will be referred to as **Type I** and **Type II** constraints, respectively. Type I constraints give rise to a source separation problem that is solved efficiently on discrete spaces by Ford and Fulkerson's max-flow/min-cut algorithm [11]. Type II constraints give rise to a source connection problem that is solved efficiently on discrete spaces by Dijkstra's algorithm [11].

Type I and Type II constraints are also necessary in the specification of minimum-weight surfaces in order to avoid the null solution. The minimum-weight surface problem with Type I constraints is equivalent to solving the max-flow/min-cut problem in 3D, which has known solutions in both continuous [1] and discrete spaces [11]. Type II constraints require that the minimum-weight surface have a prescribed boundary. Surface boundaries are always given by closed contours, an example of which is the wire rim giving the boundary of a soap bubble. The minimum-weight surface problem with Type II constraints has been studied in continuous space as Plateau's problem (e.g., [8]), but there is very limited work on minimal surfaces with Type II constraints in discrete space. In fact, the only work that we are aware of is limited to those circumstances in which it is possible to translate Type II constraints into Type I constraints [5]. Since Type II constraints govern shortest path problems on 2D weighted graphs, our focus is on the solution of the Type II constrained minimum-weight surface problem on 3D weighted graphs.

Neither Type I nor Type II constraints have priority over the other, i.e., different applications call for different constraint types. As evidence for this position, we note the enduring interest in both (2D) graph cuts [2] and intelligent scissors [9, 4], despite the fact that graph cuts applies Type I constraints and intelligent scissors applies Type II constraints to the shortest path problem. Additionally, our goal is not to argue that minimum-weight surfaces are the best tool for 3D segmentation, taking the position that graph cuts has already established interest in this problem (albeit with Type I constraints). Therefore, given the interest in applying minimal surfaces to 3D segmentation and the consistent application of Type II constraints in shortest path applications, we simply recognize a problem to be solved — Computation of minimum-weight surfaces in 3D with Type II constraints. Previous attempts to extend intelligent scissors to 3D indicate a clear interest in the resolution of this problem.

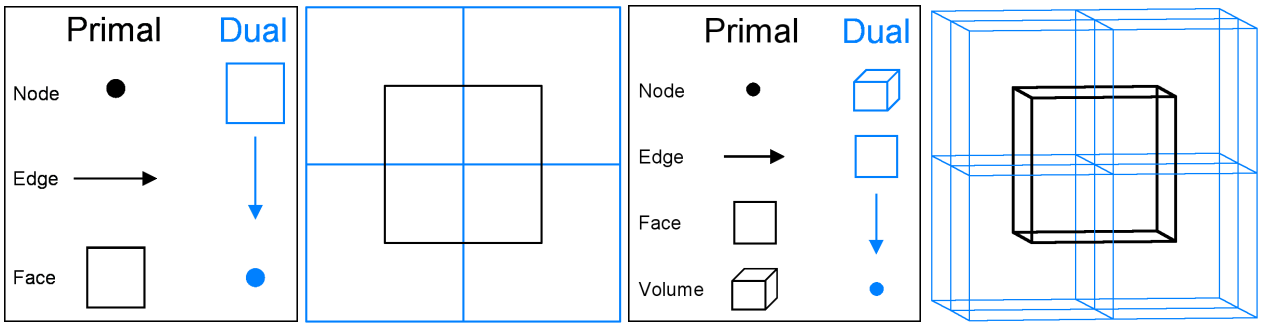
This paper is outlined as follows: In Section 2 we develop the minimum-weight surface problem in 3D and provide a general method of solution. In Section 3 we demonstrate the application of this algorithm to synthetic 3D segmentation problems of various character and then apply the algorithm to real 3D data. Finally, Section 4 provides concluding remarks and suggests directions for further research.

2. Method

In this section, we first outline a framework for viewing graph-based algorithms that produce boundaries in an image. This framework is based on the notion of a primal and dual lattice. Using this framework, we review the shortest path problem and formulate the minimum-weight surface problem. The remaining sections prove conditions under which this problem may be solved with a generic linear programming solver. Fortunately, these conditions hold for any scenario likely to be relevant to computer vision.

2.1. Duality

The notion of **duality** has played a role in graph theory (and combinatorial topology) since the time of Poincaré, in which a dual graph was defined from a planar graph by replacing each face with a dual node and connecting two nodes if their respective faces shared an edge. In this example, the **primal** graph is defined as the initial planar graph and the **dual** graph is defined as the result of the duality operation. However, this duality operation is more general in the context of algebraic topology [6] and, in fact, depends on the dimensionality of the ambient space in which the graph is embedded. In fact, a clear understanding of duality has recently come to the forefront of numerical computing (see [7] for an excellent treatment). In a general context, the standard node/face duality may be thought of as the 2-dual, in the sense that nodes are dual to 2D simplices (i.e., faces). For example, one could just as easily define a 1-dual of a graph by replacing each edge with a node and connecting nodes based upon whether or not their respective edges co-terminated at a node (this 1-dual is sometimes called the *line graph*). Figure 1 offers a picture of the relationship between the primal and dual complexes. In general, given a



(a) 2D Correspondence (b) 2D Dual Lattices (c) 3D Correspondence (d) 3D Dual Lattices

Figure 1. Duality in two and three dimensions. a) Primal and dual structures for a 2D, 4-connected lattice. b) The primal (black) and dual (blue/gray) 2D lattices. c) Primal and dual structures for a 3D, 6-connected lattice. d) The primal (black) and dual (blue/gray) 3D lattices. Treating the primal lattice as the image volume, *edges* of the dual lattice define a separating boundary of the pixels in 2D. In contrast, *faces* of the dual lattice are required to "box in" the voxels of the primal lattice in 3D.

k-simplex in p dimensions, its dual will be a $p-k$ simplex [7]. Figure 1 illustrates simplices and their duals for two and three dimensions.

The duality operation naturally produces an "outside face", but this outside face may be given several interpretations. For example, one might assign the outside face to a single node in the dual graph. Instead of this construction, we have chosen to subdivide the outside face in the dual into cells for two reasons: 1) A lattice is easier to work with computationally, 2) The addition of weighted extra edges/faces on the outside allow for a higher cost to be assigned to a longer path/surface enclosing the border pixels/voxels.

Duality offers a convenient taxonomy of graph-based segmentation algorithms. There are two basic types of graph-based segmentation algorithm: Implicit boundary algorithms and explicit boundary algorithms. Implicit boundary algorithms, such as graph cuts [2] or normalized cuts [13], label each node (pixel) as foreground/background and the boundary between them is implied by the labeling. In contrast, explicit boundary algorithms, such as intelligent scissors [9, 4], identify the boundary explicitly and a foreground/background labeling is given implicitly. The notion of duality provides a convenient may of viewing these segmentation algorithms in which one may treat the image data as existing at the nodes of the primal lattice. In this framework, implicit boundary algorithms operate on the primal lattice while explicit boundary algorithms operate on the dual lattice. Therefore, the components of the dual lattice are used to "box-in" the pixels in the primal lattice (e.g., edges/paths in 2D, faces in 3D). However, as illustrated in Figure 1, the corresponding dual lattice changes with dimension while the primal lattice remains constant, prompting the need for more modification of explicit boundary algorithms than implicit boundary algorithms.

Specifically, the popular graph cuts algorithm of [2] provides (Type I) fixed conditions at the nodes of the *primal* lattice and seeks a minimal cut (dual to a closed contour) between the source and sink nodes. In contrast, the intelligent scissors approach of [9, 4] fixes points along the boundary (Type II) in the *dual* lattice (sometimes referred to as the "cracks" or "bels" between the pixels [4]) and seeks the minimal boundary (path) that includes these endpoints. When considering higher dimensional images, graph cuts extends naturally, because edge cuts are always dual to $(p-1)$-surfaces, which define the boundary of a p-dimensional set of voxels. However, the explicit boundary approach given by intelligent scissors must be redeveloped for each dimension. Specifically, since the shortest path algorithm used in the 2D case is inappropriate to find a bounding surface in 3D, a minimum-weight surface must be used.

2.2. Minimum-weight surfaces

Before beginning the exposition, we fix our notation. For our present purposes, the primal and dual complexes will be three-dimensional, 6-connected lattices. Define a three dimensional **complex** [6] as consisting of a set $C = (V, E, F)$ with **vertices (nodes)** $v \in V$, **edges** $e \in E \subseteq V \times V$ and **faces** $f \in F \subseteq E \times E \times E \times E$ (since we will be dealing exclusively with 6-connected lattices). Let $n = |V|$ and $m = |E|$, where $|\cdot|$ denotes cardinality. Nodes, edges and faces will all be indexed by single subscripts. A **weighting** assigns a value to each edge called a **weight**. The weight of an edge, e_i, is denoted by w_i and considered in this work to be nonnegative. An **oriented** complex is one for which each structure is additionally assigned an ordering of its constituent vertices. Intuitively, the orientation corresponds to a "direction" of an edge or a "clockwise" or "counterclockwise" orientation of a face. A face and a bordering edge are said to have **coherent orientation** if the ordering given by the edge orientation is found in the orientation of

the face. Since the faces of a 6-connected lattice are degenerate simplices, the standard "even parity" definition of orientation is inappropriate. For our purposes, consider an orientation of a face, represented by column b indicating a signed membership of each edge to the face, as valid if $Ab = 0$, where A is defined in (4). Clearly, b may take a positive or negative sign while fulfilling this condition, which may be interpreted here as opposite orientations. Intuitively, a face is coherently oriented with a bordering edge if the edge "points" in the same direction as the "clockwise" or "counterclockwise" traversal of the face. For example, in Figure 2, edge e_1 is coherent with face f_A. In the context of image processing, nodes are associated with data elements (e.g., pixels, voxels) and the edges define a neighborhood relation. For our purposes, the 6-connected lattice is taken to have a face on every "square" of the lattice. Orientations must be assigned consistently, but may be assigned arbitrarily. An example orientation would be to orient all edges to point from nodes with smaller coordinates to nodes with larger coordinates and to orient each face in a clockwise manner (relative to a particular view of the lattice).

Graph-based segmentation algorithms typically focus on cutting a weighted graph, with weights given on the primal edges via a function of the intensity, e.g.,

$$w_i = \exp\left(-\beta(I_j - I_k)^2\right) \quad \text{for } \{v_j, v_k\} \in e_i, \tag{1}$$

where I_j indicates the image (volume) intensity at voxel v_j.

The minimum path problem may be viewed as the solution to the optimization problem

$$\min_y Q(y) = \sum_i w_i y_i, \tag{2}$$

where y_i represents an indicator vector on the set of edges, with $y_i = 1$ indicating that edge e_i belongs to the path and $y_i = 0$ indicating that edge e_i does not belong to the path.

In the absence of constraints, the solution of (2) yields a vector of $y_i = 0 \ \forall e_i \in E$, since all weights are nonnegative. Type I constraints may be introduced by specifying disjoint node subsets that must appear in separate connected components if the edges in the computed path are removed from the complex. Type II constraints may be introduced by specifying endpoints for the path. A succinct formulation of Type II constraints is given by

$$Ay = p, \tag{3}$$

where p is a vector of all zeros except for a $p_s = 1$ and $p_t = -1$, for endpoints $\{v_s, v_t\}$. The matrix A is the node-edge incidence matrix

$$A_{v,e} = \begin{cases} +1 & \text{if the vertex appears first in the edge orientation,} \\ -1 & \text{if the vertex appears second in the edge orientation,} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The node-incidence matrix in (3) plays the role of the *boundary operator* [6]. In this case, the boundary operator inputs an edge path (indicated by y) and returns the nodal boundary of that path (fixed by p). In general, the boundary operator inputs the indicator function of a complex and outputs an indicator function of its boundary. Therefore, use of the boundary operator in (3) allows us to fix the path boundary and succinctly express the Type II constraints for the minimal path problem. We note that this formulation of the minimal path problem is not new. For example, Papadimiriou [11] establishes the minimal path problem as the optimization of (2) with respect to (3) and proceeds to derive Dijkstra's algorithm as a particular optimization of these equations.

In order to pass from minimal paths to minimum-weight surfaces, we must increase the dimensionality of the above formulation. Fortunately, the dimensionality of the minimal path problem may be increased simply by using the dimension-appropriate incidence matrix (boundary operator) and boundary vector p. This dimension-increased shortest path problem therefore asks the question: *Given the boundary of a two-dimensional surface (i.e., a closed contour or series of closed contours), find the minimum-weight two-dimensional surface with the prescribed boundary*. As desired, this is the minimum-weight surface problem with Type II conditions.

In this dimension-increased problem, the incidence matrix (boundary operator) in question is the edge-face incidence matrix defined as

$$B_{e,f} = \begin{cases} +1 & \text{if the edge borders the face with coherant orientation,} \\ -1 & \text{if the edge borders the face with anticoherant orientation,} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Instead of the lower-dimension boundary vector, p, we now employ the vector r as an indicator vector of a closed, oriented contour taking values

$$r_i = \begin{cases} +1 & \text{if the edge } e_i \text{ belongs to the contour with coherant orientation,} \\ -1 & \text{if the edge } e_i \text{ belongs to the contour with anticoherant orientation,} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Therefore, our minimum-weight surface problem is

$$\min_z Q(z) = \sum_i w_i z_i, \tag{7}$$

subject to

$$Bz = r, \tag{8}$$

where z is an indicator vector indicating whether or not a face is present in the minimum-weight surface and w_i is meant to indicate the weights of a face. Since the faces in the dual lattice correspond to edges in the primal lattice (where the image data is located), (1) may be used to produce the set of face weights.

2.3. Optimization

Dijkstra's algorithm provides a fast, specialty method for optimizing the shortest path problem of (2) subject to (3). However, a generic linear programming optimization scheme could alternately be applied to (2) and (3) despite the fact that the solution y is strictly binary valued. The reason that this integer programming problem may be solved exactly with linear programming is due to the fact that the node-edge incidence matrix is always totally unimodular [11]. Recall that a matrix is totally unimodular when the determinant of all submatrices takes one of the values $\{-1, 0, 1\}$. Using a totally unimodular matrix as the constraint matrix for a linear programming problem (with an integer-valued right hand side) will always produce integer-valued solutions [11].

We would like to solve the minimum-weight surface problem described by (7) and (8) with a generic linear programming method, but there are two important questions that must first be addressed: 1) What are the conditions under which a given r will produce a solution? 2) Is the edge-face incidence matrix of (5) totally unimodular? The answer to these questions is the subject of the following two sections.

2.3.1 Feasibility

We begin by formally stating that the boundary of a boundary is zero in terms of the incidence matrices:

$$AB = 0. \tag{9}$$

Since r is the indicator vector of a closed contour, we note that

$$Ar = 0. \tag{10}$$

Now, we can make the following statement regarding the feasibility of finding a solution to (8) given a closed contour, represented by an r that satisfies (10).

Proposition 1. *If the edge-face incidence matrix has $m - n + 1$ independent columns and a nonzero vector r satisfies (10), then (8) is guaranteed to have a solution.*

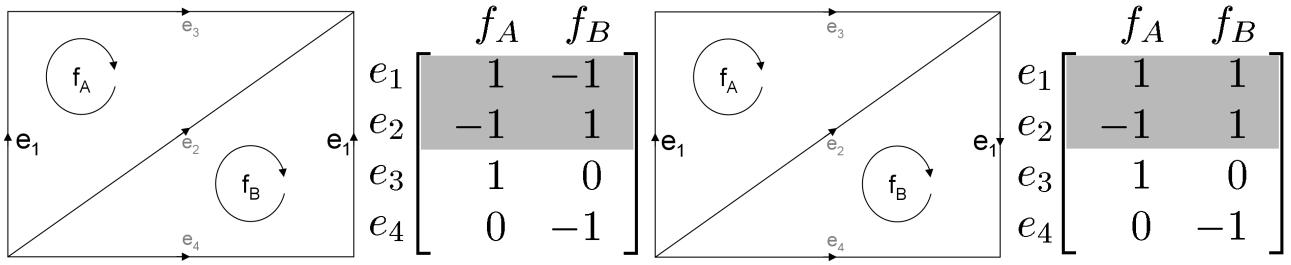
Proof. The node-edge incidence matrix is known to have a right nullspace of rank $m - n + 1$ [6]. Since (9) holds, and the edge-face incidence matrix has $m - n + 1$ independent columns, then the edge-face incidence matrix spans the right nullspace. In other words, if A r = 0, then r may be expressed as a linear combination (with constants c) of the columns of the edge-face incidence matrix r = B c, giving the proposition. □

In the context of image segmentation on a 6-connected lattice, the conditions of Proposition 1 will hold and therefore Proposition 1 settles the issue of feasibility. In a more general context, the question of feasibility hinges on whether or not the contour represented by r is a member of the homology group of the complex. For example, if the underlying complex were the triangulated surface of a torus, then an r representing a contour encircling the handle would not have a feasible solution. This situation is analogous to the minimal path problem in which two points placed in separate components of a disconnected graph will not have a path joining them.

2.3.2 Total unimodularity[1]

In order to solve the minimum-weight surface problem with a generic linear programming solver, we must show that the edge-face incidence matrix in (8) is totally unimodular. Unfortunately, Okada [10] has given examples of edge-face incidence matrices that are not totally unimodular. Consequently, we must demonstrate that the structures likely to be used in computer vision have a totally unimodular edge-face incidence matrix. As before, the structure that we are primarily interested in is the 6-connected cellular lattice. Our strategy for proving total unimodularity is to introduce two operations on a complex, show that they do not affect the total unimodularity of the edge-face incidence matrix and

[1] This note was added post-publication on 9/13/06 and amended on 11/26/08 by the author — The text within the document is reproduced as published. The topic of total unimodularity is not correctly handled in this section. The issues are: 1) Total unimodularity of the constraint matrix is mischaracterized as both sufficient *and* necessary for a constraint of the form (8) to guarantee an integral solution. In fact, total unimodularity is simply sufficient, but not necessary to guarantee an integral solution in the presence of a constraint of this form. 2) The proofs of total unimodularity for the lattice do not prove this property, since they are predicated on the (false) premise that total unimodularity is preserved via elementary matrix operations. The elementary matrix operations preserve unimodularity, but not total unimodularity. However, the two operations introduced do preserve orientability, and therefore show that the lattice is orientable. 3) The edge-face incidence matrix of the lattice is not, in fact, totally unimodular (thanks to Vladimir Kolmogorov for illuminating this point). Despite the aforementioned troubles with this section, the primary conclusion is still correct, for reasons that are explained in the upcoming TPAMI version of this paper, available online at: http://cns.bu.edu/~lgrady/grady2009minimal.pdf. Namely: On the lattice, minimizing (7), subject to the constraint (8) is guaranteed to produce an integer solution when using Linear Programming.



$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cc} f_A & f_B \end{array} \\ \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cc} f_A & f_B \end{array} \\ \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix} \end{array}$$

(a) Untwisted strip (b) Incidence matrix (c) Möbius strip (d) Incidence matrix

Figure 2. A linear programming approach may be used to solve the minimum-weight surface problem if the edge-face incidence matrix is totally unimodular. Recall that "total unimodularity" refers to the property of a matrix such that each square submatrix has determinant equal to $\det = \{-1, 0, 1\}$. As illustrated here, total unimodularity of the edge-face incidence matrix depends on the *orientability* of the underlying complex [6]. a) A triangulation of an untwisted strip. Note that the larger, darker edge, e_1, on both ends of the strip is considered attached (without a twist). b) The edge-face incidence matrix of the untwisted strip is totally unimodular. c) Triangulation of a Möbius strip. Note that the edge, e_1, on both ends of the strip is twisted and attached. This twist is reflected by the orientation of the right e_1 from (a) to (c) to reflect the twist. d) The edge-face incidence matrix of the Möbius strip of Figure (c) is not totally unimodular. The shaded submatrix formed by edges $\{e_1, e_2\}$ and faces $\{f_A, f_B\}$ has determinant equaling two, unlike the orientable case represented in (b).

then note that the 6-connected lattice (among other structures) may be built from these two operations.

Consider the operation **`face subdivision`** to be defined as the decomposition of face f_a into faces $\{f_1, f_2\}$ such that the column corresponding to f_a in B equals the sum of the columns corresponding to f_1 and f_2 and only one of $\{f_1, f_2\}$ has a nonzero entry for each row (prior to the addition of the new edge). The face subdivision process additionally requires that at least one new edge must be added, with the condition that the new edges are "internal", i.e., the rows corresponding to the new edges are zero everywhere except in the columns f_1 and f_2, and that the entries in those two columns have opposite sign. The operation **`add spanning face`** is defined as the addition of a new face to the complex, such that its column is the sum of existing face columns with the coefficients $\{-1, 0, 1\}$.

Proposition 2. *Given a complex with a totally unimodular edge-face matrix, the operations of face* `subdivision` *and* `add spanning face` *will produce a complex with a totally unimodular edge-face incidence matrix.*

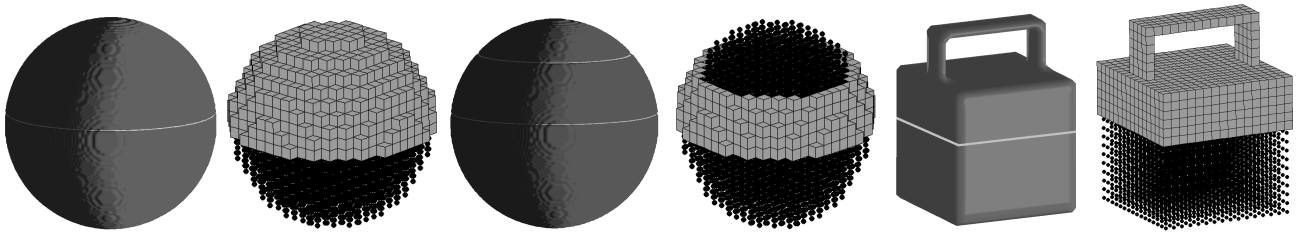
Proof. Consider the following elementary row/column operations: 1) transposition (exchange of two rows/columns), 2) change in sign of a row/column, 3) adding to the elements of a row/column the corresponding elements of another row/column multiplied by an integer. Note that each of these elementary operations corresponds to multiplication of the matrix by a totally unimodular matrix and that the product of two totally unimodular matrices is another totally unimodular matrix [6]. We now proceed to show that the elementary operations above may be used to reduce the two operations to addition of a column of zeros or addition of a row and column with a single nonzero entry belonging to the set $\{-1, 1\}$. Clearly, either addition to the matrix would not affect the total unimodularity of the matrix.

By definition of `add spanning face`, the new column is the generation of previous columns and may therefore be cancelled (i.e., reduced to zero) by existing columns using the elementary operations defined above.

The `face subdivision` operation requires two parts. First, since the internal edges have zero entries outside of f_1 and f_2 and because each row of $\{f_1, f_2\}$ has at most one nonzero entry, a row corresponding to an internal edge may be used to cancel all rows corresponding to noninternal edges. Second, since the entries of the internal edges have opposite signs, one column may now be used to cancel the other. The remaining column has nonzero entries only on internal edge rows and therefore one row (internal edge) may be used to cancel the other rows, leaving a row and a column with a single nonzero entry with value $\{-1, 1\}$. □

Clearly the 6-connected lattice may be generated with the above operations by starting with a cube (zero faces) and progressively subdividing it into a lattice on the surface of the cube. Spanning faces may then be used internally to the cube (followed by subdivision) to generate the lattice.

Intuitively, the issue of total unimodularity of the edge-face incidence matrix rests on whether or not the faces represent an **orientable manifold** [6]. Note that the terms "oriented" and "orientable" refer to different properties. The term "oriented" indicates that a sense of direction has been given to each simplex, while "orientable" indicates that the surface is torsion-free in the topological sense. To illustrate the issue of orientability, Figure 2 compares the edge-face incidence matrix for a triangulated Möbius strip with the triangulation of a standard (i.e., non-twisted) strip. If our underlying complex is nonorientable (e.g., a triangulated Möbius strip or Klein bottle), then the underlying edge-face



(a) (b) (c) (d) (e) (f)

Figure 3. Synthetic examples to illustrate correctness. Renderings of the original object (with the input contours) are shown, along with the algorithm outputs. The input volumes all had black voxels indicating voxels belonging to the object and white voxels indicating background. The white stripe in each of the rendered views shows the input contour(s). In the solution visualizations, black dots are plotted at the center of the black (object) voxels and faces are shown to indicate the computed surface. (a,b) A sphere with an input contour along a parallel. Note that, unlike 2D intelligent scissors, a single boundary input (contour) is sufficient to define a solution. (c,d) A sphere with input contours at two parallels of different heights. (e,f) A lunchbox shape with a handle on the top and a medial contour input. The algorithm will correctly find minimal surfaces with topological changes.

incidence matrix is not totally unimodular. Fortunately, for purposes of 3D image processing, the 6-connected lattice (and all structures likely to be used) is orientable and therefore has a totally unimodular edge-face incidence matrix, as shown in Proposition 2.

2.4. Algorithm summary

Having proved Propositions 1 and 2, we can see that our generalization of the shortest path problem to the minimum-weight surface problem may be solved through the application of a generic linear programming solver. This linear programming problem inputs a closed contour (or series of closed contours) and returns a minimum-weight surface. Since closed contours are the output of standard (2D) intelligent scissors, the outputs of the 2D intelligent scissors gives inputs for a 3D intelligent scissors. Alternately, another 2D segmentation algorithm could also be used to produce the inputs to the minimum-weight surface problem.

One implementation detail must be mentioned. The shortest path optimization of (2), (3) must include two equations for each edge in order to reflect the possibility that a path could traverse that edge in either direction — each undirected edge is broken into two directed edges. This same construction must be followed in the formulation of the minimum-weight surface problem of (7), (8). In this problem, each face must be replicated as two faces with opposite orientation. Note that a pair of the same faces with opposite orientation have identical entries in the edge-face incidence matrix, except for a sign change. Explicitly including faces of both orientations leaves us with linear programming problem with $6(k+1)k^2$ equations for a $k \times k \times k$ lattice.

We may summarize the entire algorithm as follows:

1. Obtain an oriented closed contour on one or more slices through an outside algorithm (e.g., 2D intelligent scissors). This contour is represented in (8) by vector r.

2. Define face weights from the image content using (1). Note that "outside" faces must be assigned to an arbitrary value — we have employed $w = 0.5$

3. Use a generic linear programming solver to minimize (7) subject to (8) using the r defined in step one and the edge-face incidence matrix defined in (5).

4. The solution vector z indicates the faces that form the 2D minimum-weight surface separating the 3D regions, given the prescribed boundaries.

3. Results

In the previous sections, we generalized the (2D) shortest path problem with Type II constraints to the analogous (3D) minimum-weight surface problem with Type II constraints. Therefore, a natural extension of intelligent scissors to 3D has been provided. In this section, our goal is to verify the correctness of the algorithm on synthetic data and then to demonstrate its application to the segmentation of 3D data.

3.1. Correctness

We begin with three examples to demonstrate correctness. First, we use the algorithm to segment a black sphere (in a white background), given an initial contour around one parallel. Secondly, we segment the same sphere using a boundary consisting of contours around two parallels (i.e., a contour given on two slices). Finally, we segment a "lunchbox" shape given a medial contour. This experiment shows that the algorithm correctly handles topological changes.

Figure 3 shows the results of these three experiments, verifying the correctness of the algorithm. Figure 3(a) shows that a single closed contour is sufficient to define

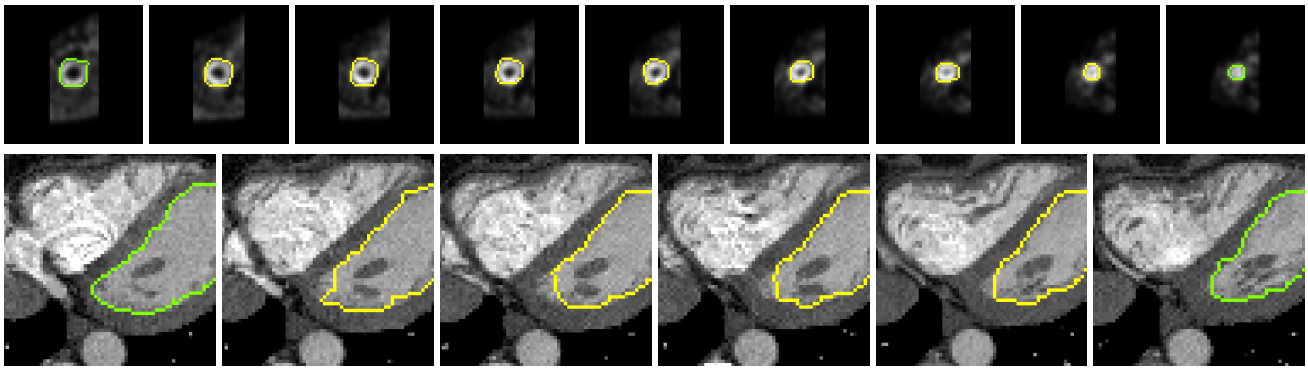


Figure 4. Application to segmentation of 3D data. In both studies, the green/gray contours were placed on the left and right slices and the intermediate yellow/white contours represent the minimum-weight surface between these contours. This figure demonstrates that the algorithm behaves as expected for a minimal surface approach to 3D segmentation. Top: SPECT cardiac data. Bottom: CT cardiac data.

the boundary of a surface, in contrast to the shortest path problem in which two points are necessary to define a path. A similarity with the shortest path problem is that the minimum-weight surface may not be unique.

We stress that the minimum-weight surface will not necessarily equal the solution to the minimal surface problem in continuous, Euclidean space. For example, the solution to the continuous minimal surface is a catenoid when given a boundary of two, identical, closed contours at different heights. In contrast, the minimum-weight surface is a cylinder when the underlying complex is a 6-connected graph. This contrast between the discrete and continuous domains is analogous to the fact that the shortest path on a lattice will not necessarily be the same as a straight line in the plane.

3.2. Real data

In this section we illustrate the application of minimum-weight surfaces to the segmentation of real data. Minimum-weight surfaces with Type I boundary conditions have been previously applied to image segmentation via graph cuts [2]. Therefore, we simply illustrate the use of our method for computing minimum-weight surfaces with Type II boundary conditions in the context of image segmentation (i.e., as a 3D extension of intelligent scissors). Two 3D datasets were used in these experiments — SPECT cardiac data and CT cardiac data. Type II boundary conditions (closed contours) were generated using standard intelligent scissors on two slices of each dataset, and we computed the minimum-weight surface that had these contours as a boundary. Figure 4 shows the result of these experiments.

4. Conclusion

The increasing use of minimum-weight surfaces for segmentation in 3D, and ubiquity of shortest path problems in computer vision, led us to define an algorithm for finding exact, minimum-weight surfaces in 3D weighted graphs. In contrast to the max-flow/min-cut approach to finding minimum-weight surfaces by defining Type I boundary conditions, our algorithm finds minimum-weight surfaces through the specification of Type II boundary conditions. Both Type I and Type II boundary conditions find continuing application in 2D for the computation of shortest paths. We have shown that it is possible to use a generic linear programming solver to find the minimum-weight surface unless the image is defined on an unusual underlying complex (e.g., a Möbius strip) rather than a standard, 6-connected, lattice. A natural application of our results is the extension of the powerful intelligent scissors/live wire approach to three dimensions. An attractive feature of this 3D extension of intelligent scissors is that the output given by standard (2D) intelligent scissors provides an input for the computation of a minimum-weight surface.

Future work will the develop along three lines: 1) Search for a specialty linear programming solver, analogous to Dijkstra's algorithm for the shortest path problem, 2) Application of the 3D minimum-weight surfaces to extend other shortest path-based computer vision algorithms to higher dimension, 3) Extension of this approach to find minimum-weight hypersurfaces in higher dimensional complexes.

References

[1] B. Appleton and H. Talbot. Globally optimal surfaces by continuous maximal flows. *IEEE PAMI*, 28(1):106–118, Jan. 2006. 2

[2] Y. Boykov and M.-P. Jolly. *Interactive graph cuts* for optimal boundary & region segmentation of objects in N-D images. In *Proc. of ICCV 2001*, pages 105–112, 2001. 2, 3, 8

[3] L. Cohen and T. Deschamps. Grouping connected components using minimal path techniques. Application to reconstruction of vessels in 2D and 3D images. In *Proc. of CVPR 2001*, volume 2, pages 102–109. IEEE Comp. Soc., 2001. 1

[4] A. X. Falcão, J. K. Udupa, S. Samarasekera, S. Sharma, B. H. Elliot, and R. de A. Lotufo. User-steered image segmentation

paradigms: Live wire and live lane. *Graphical Models and Image Processing*, 60(4):233–260, 1998. 1, 2, 3

[5] D. Kirsanov. *Minimal Discrete Curves and Surfaces*. PhD thesis, Harvard University, Cambridge, MA, 2004. 2

[6] S. Lefschetz. *Algebraic Topology*, volume 27. American Math. Soc. Col. Pub., 1942. 2, 3, 4, 5, 6

[7] C. Mattiussi. The finite volume, finite element and finite difference methods as numerical methods for physical field problems. In *Advances in Imaging and Electron Physics*, pages 1–146. Academic Press Inc., April 2000. 2, 3

[8] F. Morgan. *Geometric Measure Theory*. Academic Press, London, 3rd edition, 2000. 1, 2

[9] E. Mortensen and W. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models in Image Processing*, 60(5):349–384, 1998. 1, 2, 3

[10] S. Okada. On mesh and node determinants. *Proc. of the IRE*, 43:1527, 1955. 5

[11] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover, 1998. 2, 4, 5

[12] S. V. Porter, M. Mirmehdi, and B. T. Thomas. A shortest path representation for video summarisation. In *Proc. of 12th ICIAP*, pages 460–465. IEEE Comp. Soc., Sept. 2003. 1

[13] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22(8):888–905, Aug. 2000. 3

[14] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control*, 40(9):1528–1538, Sept. 1995. 1