2001 *IASTED International Conference on Signal and Image Processing*, 22 - 26

Marching Ridges

Jacob D. Furst
DePaul University
jfurst@cti.depaul.edu

Stephen M. Pizer
University of North Carolina
pizer@cs.unc.edu

Abstract

Marching Ridges is an algorithm for finding ridges of measurement functions defined for images, in which the ridges are defined as level sets of first derivatives of the measurement functions. Marching Ridges is both specific in its application to 2D and 3D medical images for the purpose of finding cores, and general in its theoretical application to finding ridges of intensity, boundary and medial measurements whose domains may include up to 8 dimensions. Marching Ridges uses derivative calculations, zero-crossing interpolations and topological heuristics to track a linear approximation of a ridge through a network of hypercubes.

1. Introduction

This paper presents an algorithm for identifying height ridges called *Marching Ridges* derived from earlier work on finding cores [6] and inspired by the extent to which ridges can be used for image analysis [4]. It is designed as a general-purpose algorithm that can solve for height ridges of any measurement derived from an image; examples include intensity, boundariness, and medialness. The majority of this paper is independent of the choice of function. However, parts of this paper are specific to medialness functions and optimal parameter height ridges. The Marching Ridges algorithm does not deal with the limiting cases of 0- or n-dimensional ridges, where n is the domain of the measurement. 0D ridges are local maxima of a function and, as isolated points, require no tracking, rendering the Marching Ridges algorithm superfluous and inefficient. Similarly, an nD ridge of a function of n variables is simply the graph of that function and requires no special algorithm to identify.

Marching Ridges is a method of finding linear approximations to ridges by using the lattice structure of the domain in which the ridge is contained. It is very similar to algorithms such as Marching Cubes[8, 11], Tracked Partitioning[1], and Marching Lines[5, 13] but has the added complexity of finding ridges. The marching also occurs within the lattice structure of the domain by identifying hypercubes adjoining the hypercube containing the ridge from shared hyperfaces that contain a part of the ridge. This paper is organized to first introduce the interface for Marching Ridges and then describe the algorithm for calculating ridges. The algorithm is divided into ridge finding strategies, both general (not depending on the codimension of the ridge) and specific (depending on the codimension of the ridge), and marching strategies, also both general (not depending on the dimension of the ridge) and specific (depending on the dimension of the ridge). Following this is a brief discussion of the time and space complexity of the Marching Ridges algorithm.

2. Grid elements

Marching Ridges is organized around a regular partition of the n-dimensional product space in which the ridge is to be found. The subdivision of the product space into unit n-cubes produces *grid elements*. We use the term grid element in preference to dimension specific terms such as pixel and voxel for two reasons. First, Marching Ridges has been designed for the general purpose of finding d-dimensional ridges in an n-dimensional function domain space, and the use of grid elements allows a dimensionally neutral discussion of the algorithm. Second, a grid element has vertices that are sample points of the function in its domain space rather than the more common notion of pixels and voxels, in which the pixel or voxel is centered at a sample point of the function. Thus, the grid element of a 2D space would contain four sample points, one at each vertex of a square and the grid element would reference four function values rather than the single value associated with a pixel. We will also refer to *subelements*: lower-dimensional components of grid elements. For example, grid elements of a 3D space are cubes, while the various subelements are faces (containing four sample points), edges (containing two sample points) and points (containing a single sample point). We will further use the term *border element* to distinguish the largest subelement that two grid elements share. This will be the subelement of dimension one less than the grid element; for example, the border element for a cubic grid element is a square element.

3. Interface

The interface to the Marching Ridges algorithm is a window containing a central image canvas 512 pixels square surrounded by sliders and buttons to provide parameters and instructions to the algorithm. The controls of the interface set and maintain five important quantities:

- the dimension of the original image n
- the number of optimal parameters s,
- the dimension of the grid elements $g = n - s$,
- the dimension of the ridge d, and
- the codimension of the ridge with respect to the dimension of the grid elements $c = g - d$.

Each of these quantities is initially 0 but will change as the user sets the parameters of the ridge-finding task by selecting certain buttons. As we describe each set of buttons below, we will indicate how particular choices affect the values of n, s, g, d and c.

3.1. Control buttons

The *quit* button allows the user to exit the program after completing any ridge. The *load* button allows the user to work with a new image. When the image is selected and loaded, the value of n is set to the dimensionality of the image. The *draw* button allows the user to redraw the image without any ridge points on it, or to redraw the image with the current ridge superimposed. When loading a 3D image, Marching Ridges determines how many slices of the image can be displayed on the image canvas. Marching Ridges then breaks the original image into metaslices, each containing the number of slices than can be simultaneously displayed in the interface. The *up* and *down* buttons are used to view different metaslices of the images. The *track* button initiates ridge finding based on a user supplied grid element (Section 5.1). The *trace* button provides a diagnostic tool for examining function values at a single spatial position parameterized by scale or scale and orientation. The *single* button allows the user to calculate ridges for a single grid element of the function domain space. It provides action identical to the track button but will not extend the ridge if the single element contains a ridge nor search for initial ridge points if the element does not.

3.2. Image measurements

Marching Ridges allows the user to choose from three functions on which to find ridges. The first choice is an intensity function, similar to [7], in which Marching Ridges measures the intensity of the image using a zero-mean Gaussian weighting function of a specified standard deviation (Section 3.5). The second choice is a boundariness function, based on [2], in which Marching Ridges measures boundariness using either gradient magnitude or oriented first derivatives using a zero-mean Gaussian first derivative weighting function of a specified standard deviation and orientation (in the case of oriented first derivatives). The third choice is a medialness function [10], in which Marching Ridges measures medialness using either an isotropic Laplacian, an oriented Laplacian or an oriented Morse/Fritsch medialness weighting function; the Morse function is used to find 2D cores of 3D images while the Fritsch medialness is used to find 1D cores of 3D images. The algorithm uses medialness weighting of a specified standard deviation and orientation for the oriented weighting functions. The choice of measurement may also affect the dimension of the grid elements. Oriented boundariness adds $n-1$ to g, isotropic Laplacian medialness adds 1, and any of the oriented medialness measurements add n.

3.3. Ridge dimension

Marching Ridges currently allows the user to find 1D or 2D ridges. In the case of 2D images, only the 1D ridges are appropriate, while both may be sought in the case of 3D images. This choice also sets the ridge dimension d and codimension $c = g - d$.

3.4. Parameters

The radius parameter affects the size of the weighting function used to measure the image. In the case of intensity and boundariness measures, the radius is the standard deviation of the Gaussian weighting function. In the case of medialness measures, the radius is a constant mulitple of the standard deviation of the Gaussian weighting function. The theta orientation parameter is used only for oriented weighting functions: directional derivatives (boundariness) and oriented medialness. Similarly, the phi orientation parameter is used only for weighting functions oriented in 3D. All three parameters, radius, theta and phi, are set by the user to determine an initial value for optimization when finding optimal parameter ridges or to determine the initial grid element for the Marching Ridges algorithm when finding other ridges. The extent parameter determines the support of the weighting function. In most cases, the support of the weighting function is the product of radius and extent. In the case of Fritsch and Morse medialness, the size of the support is $r(1+\sigma\xi)$, where r is the radius and ξ is the extent. The rho parameter is additionally used as the ratio between radius and aperture for Morse and Fritsch medialness weighting functions. Finally, the Z parameter is the ratio of the interslice distance to the intraslice distance for 3D images.

3.5. Optimizations

The three optimization buttons provide the user the choice of no optimization, scale optimization only, or a combination of scale and orientation optimizations and correspond to choosing certain parameters as transverse directions. These optimizations are used to calculate optimal parameter ridges. The Marching Ridges algorithm was designed around the decision to make all auxiliary parameters optimal or to make none of them optimal. Thus, for example, it is not possible to find optimal scale ridges of an oriented medialness function. The choice of optimizations may also affect and s (and thus g). Scale optimization increases s by 1 and decreases g. A scale and orientation optimization increases s by n and decreases g by n.

3.6. Object/background polarity

Marching Ridges is able to find intensity and medialness ridges of white objects on black backgrounds or black objects on white backgrounds. The decision is not material in the case of ridges of boundariness; gradient magnitude is unaffected by polarity and directional first derivatives merely indicate opposite normals, depending on polarity, which does not affect the location of the ridge.

3.7. Mouse

Marching Ridges supports a three-button mouse. The left button sets a spatial position in the image for the track, trace, and single buttons. The middle mouse button reports the image value and coordinates of the location at which the button is pressed. It does not set an initial location. The right mouse button allows the user to enter an initial spatial position and auxiliary parameter values through the keyboard, rather than the mouse and sliders. These values are then used in a trace.

3.8. General ridge-finding strategies

Most of the ridge-finding strategies of Marching Ridges are dependent on the codimension c of the ridge, as that determines what particular subelement identifies ridge elements. However, there are two actions that are essentially independent of c: actions at points and averaging transverse directions.

3.9. Actions at points

Points are the lowest dimensional subelement for any product space. There are three actions that occur at every point, and form the basis for any ridge calculation:

- the calculation of derivatives in coordinate directions,
- the determination of transverse directions, and
- the alignment of transverse directions and the calculation of derivatives in transverse directions.

3.9.1. Calculation of derivatives. The first action at points is the calculation of derivatives. In the case of optimal parameter height ridges, the calculation of derivatives is preceded by a local maximization of the original function *f* over the optimal parameters. The point then calculates derivatives of the original function using the coordinates of the point and any optimized parameter values. This is done using symbolic manipulation of the original weighting function to produce derivative weighting functions that are then applied to the image. Each point will apply a large number of derivative weighting functions to the image. In the case of optimal parameter ridges [9], the derivatives of the optimal parameter function $\hat{f}$ are then calculated from the derivatives of f. The result is a set of first and second derivatives at each point.

3.9.2. Determination of transverse directions. The determination of transverse directions may follow two different paths, depending on whether ridges are optimal orientation or not. In the case of optimal orientation ridges, the transverse directions are determined from the optimal orientation: transverse directions are perpendicular to the optimal orientation. If the ridge finding requires more than one transverse direction (notably 1D ridges from 3D images), they are chosen from the space perpendicular to the optimal orientation using an *a priori* choice that prevents a degenerate set of transverse directions. Given that the weighting function is isotropic in the plane spanned by the two transverse directions, the only critical need is to identify two orthogonal directions. In all other cases, the transverse directions are chosen from an eigen-analysis of the Hessian matrix of second derivatives.

3.9.3. Alignment of transverse directions. Because the Marching Ridges algorithm relies on zero-crossings of directional first derivatives, it is important to insure that the directions all share the same sense or sign. Marching Ridges accomplishes this by computing an average set of transverse directions, and then comparing each point's transverse directions with the average. If the dot product of a transverse direction and its average is negative, then the transverse direction is multiplied by -1. This action does depend on the codimension *c* of the ridge since a point has *c* transverse directions that it must align. Once each transverse direction has been aligned, the point calculates derivatives in the transverse direction(s) $\vec{v}_i$

$$D_{v_i} f = \vec{v}_i \bullet \nabla f$$

3.10. Average transverse directions

Stetten [12] has a method of performing an eigen-analysis on unit vectors in order to capture an "average" orientation of the vectors without regard to the sense of the vector. This is the method that Marching Ridges uses to create a set of average transverse directions for points.

Given h vectors $\vec{v}_i$, $1 \le i \le h$ each representing a transverse direction, construct a matrix C such that

$$C = \frac{1}{h}\sum_{i=1}^{h} \vec{v}_i^{\,t}\vec{v}_i \qquad \textbf{(Equation 1)}$$

and perform an eigen-analysis of C. The "average" transverse direction $\vec{v}$ is the eigen-vector of C corresponding to the greatest eigen-value. If there is more than one transverse direction, perform this analysis separately for each.

4. Specific ridge-finding strategies

The strategy of Marching Ridges is to use the codimension c of the ridge to push ridge finding to the lowest subelement possible. This will always result in finding ridge points contained in the subelements of a given grid element. If necessary, these points can be "sewn" together to form higher dimensional representations of a ridge (*e.g.*, splines and patches). The particular subelement identifying a ridge point is the one having dimension equal to c. This is merely a restatement of the fact that two manifolds will generically intersect at points when the sum of their codimensions equals the dimension of the space in which they lie. For example, when finding 1D ridges in a 3D grid element (ridges of codimension two), the actual identification of ridge points is done at the faces (subelements of dimension two) of the cubic grid elements. If necessary, the ridge points on the faces of the cubic grid element can be stitched together to form a curve in the grid element. Marching Ridges currently supports ridges of codimension one and two, so the following sections concentrate on the actions of edges and faces that produce ridge points.

4.1. Ridge-finding edges

Ridges of codimension one ($c = 1$) share the property of only requiring a single transverse direction. As explained above, the subelement on which to search when $c = 1$ will be the edge. Examples in which $c = 1$ include boundaries of objects and generic skeletons of objects. The edge performs three actions in the calculation of ridge points:

- the determination of average transverse directions,
- the calculation of zero-crossings, and
- a check on second derivatives.

4.1.1. Determination of average directions (Figure 1). As mentioned, points attempt to align their transverse directions with an average set of transverse directions. Since in this case, edges are identifying ridge points, the edge is responsible for averaging the transverse directions of its endpoints and reporting this information to those points. The edge queries its endpoints and uses Stetten's algorithm (Equation 1, $h = 2$) to identify an average transverse direction. Each of its endpoints then aligns its own transverse direction with the average.

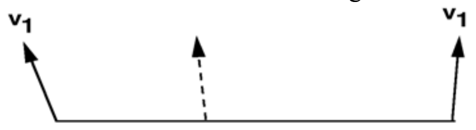

Figure1. Average transverse direction on an edge

4.1.2. Calculation of zero-crossings (Figure 2). Once the endpoints of the edge have aligned their transverse direction to the average, the edge looks for zero crossings of the first directional derivative of the ridge function in the transverse direction. Assuming that the first directional derivative is a continuous function, the intermediate value theorem ensures that, given first directional derivatives of opposite sign at each endpoint, there will exist at least one point on the edge for which the first directional derivative is zero. Assuming that the first directional derivative is linear, the edge interpolates a location for a single zero crossing. This is a potential location for a ridge point, having satisfied the first condition of being a ridge: a vanishing first derivative in the transverse direction.

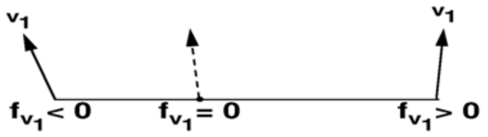

Figure 2. Interpolated zero crossing of the first directional derivative

4.1.3. Second derivative check (Figure 3). Having found a zero crossing of the first directional derivative, the edge must check the second condition for being a ridge: a negative second directional derivative. The interpolated zero crossing performs all the actions that a grid point does in calculating derivatives, finding transverse directions, aligning the transverse direction to the edge average, and recalculating the second derivative in the transverse direction

$$D_{v_i v_i} f = \vec{v}_i H(f) \vec{v}_i^t$$

This derivative is tested against zero, with a true result marking the point as a ridge point. The point is then drawn to the image window and added to a list of ridge points.

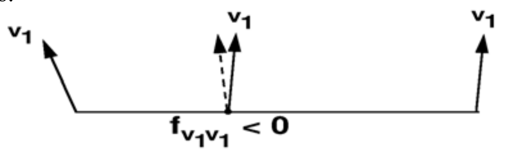

Figure 3. Positive identification of ridge point based on second directional derivative

Note: the first derivative is not recalculated at the potential ridge point. The linear interpolation is assumed to have provided a close estimate of the zero crossing of the first derivative, so no further check is actually made.

4.2. Ridges-finding faces

Ridges of codimension two ($c = 2$) share the property of requiring two transverse directions. As explained above, the subelement on which to search when $c = 2$ will be the face. An example in which $c = 2$ is the skeleton of a tubular object. The face performs four actions in the calculation of ridge points:

- the determination of average transverse directions,
- the calculation of zero-crossings of first directional derivatives in the first transverse direction,
- the calculation of zero-crossings of first directional derivatives in the second transverse direction, and
- a check on second derivatives.

4.2.1. Determination of average directions (Figure 4). As with the edge, a face identifying ridge points must determine a pair of average transverse directions formed from the transverse directions calculated at each of its vertices. Each transverse direction is averaged independently (Equation 1, $h = 4$) and the results reported to the vertices of the face.

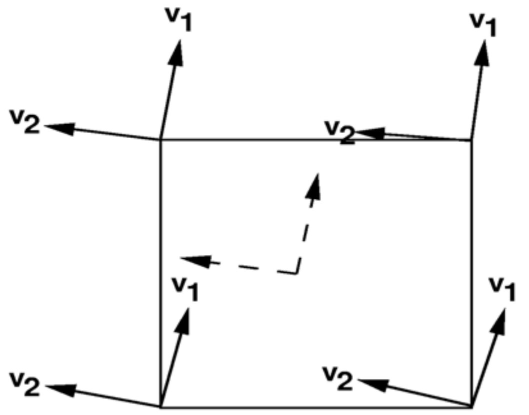

Figure 4. Average transverse directions on a face

4.2.2. Calculation of zero-crossings in first transverse direction (Figure 5). Once the vertices of the face have aligned their transverse directions to the average, the face looks for zero crossings of the first directional derivative of the ridge function in the first transverse direction on each of its four edges. At this stage, the face is identifying a codimension one ridge as determined by the first transverse direction. As before, the assumptions of continuity and linearity provide a single location for a zero crossing on any edge containing endpoints with derivatives of opposite sign.

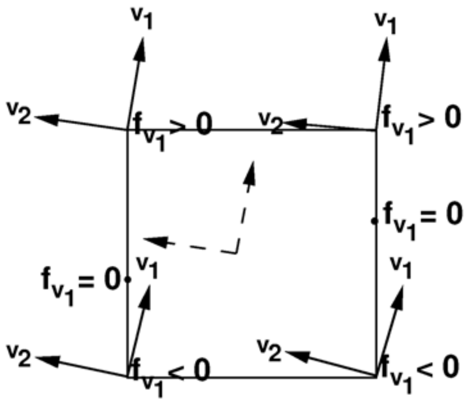

Figure 5. Interpolated zero crossings in first transverse direction

4.2.3. Calculation of zero-crossings in second transverse direction (Figure 6). If less than two edges identify zero-crossings, then there is no ridge in the face. If more than two edges identify ridge points, the face chooses the two most convex. (See Section 5.5.2.) These two points then perform the standard actions of points, resulting in first directional derivatives in the second transverse direction. These derivatives are checked for sign, and if they are opposite, a zero-crossing is interpolated along the segment connecting the two. This zero-crossing is a potential ridge point in the face, having satisfied the condition that the first derivatives vanish in each transverse direction.

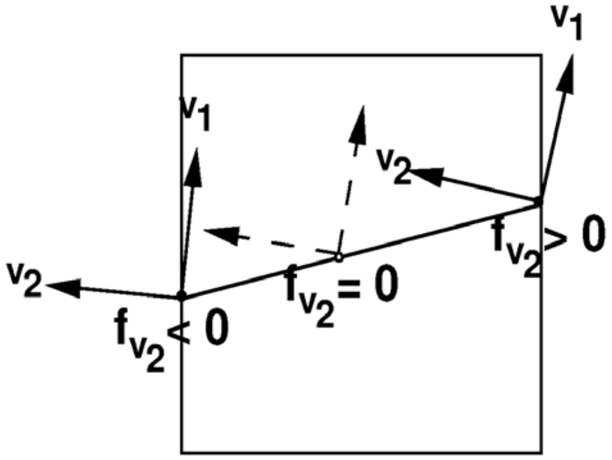

Figure 6. Interpolated zero crossing in second transverse direction

4.2.4. Second derivative check (Figure 7). Having found a zero crossing of the first directional derivative in the second transverse direction, the face must check the second condition for being a ridge: negative second directional derivatives. The interpolated zero crossing performs all the actions that a grid point does in calculating derivatives, finding transverse directions, aligning the transverse directions to the face average, and calculating the second derivatives in the transverse directions. These derivatives are tested against zero, and if both are negative, then the point is marked as a ridge point. The point is then drawn to the image window and added to a list of ridge points.

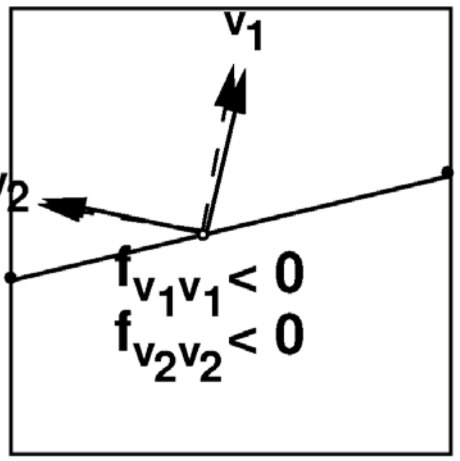

Figure 7. Positive identification of ridge point based on second directional derivatives

5. General marching strategies

Most marching strategies are a function of ridge dimension d. However, three actions are not:

- specifying the initial grid element,
- searching for the initial ridge points, and
- creating new grid elements.

5.1. Initial grid element

Marching Ridges is a semi-automatic algorithm: it requires the intervention of a user to identify a starting point. The user supplies an initial grid element in which to search for a ridge using two mechanisms, the mouse and the parameter sliders. As mentioned above, the parameter sliders are used to specify the radius and orientation of the initial grid element, while the mouse is used to indicate the spatial position of the initial grid element. (For large 3D images, the up and down buttons may be necessary to locate the correct metaslice before identifying the spatial position with the mouse.) The position so identified is called the anchor vertex of the grid element. The rest of the vertices of the grid element are calculated as a unit cube whose smallest coordinate is the anchor vertex (Figure 8). From this initial grid element, Marching Ridges will search for the closest ridge point and then extend the ridge from this identified point.

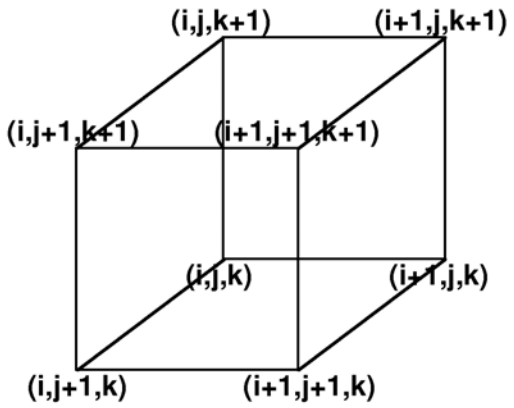

Figure 8. Initial grid element

5.2. Expanding the search

If the user-specified grid element contains a ridge, then the initial search has succeeded and extending begins. If not, then Marching Ridges begins a breadth first search of the grid elements neighboring the initial grid element until it either finds a ridge point or searches all the grid elements of the search space. If a ridge point is found, Marching Ridges immediately begins extending it. In this way, Marching Ridges finds the ridge point closest to the user specified grid element.

5.3. Creation of new grid elements

When a new grid element is required, either in the initial search for a ridge or in the extension of an existing ridge, Marching Ridges creates one. It then identifies the subelements of the new grid element that have already been created and assigns them to the new element. Any subelements that have not been created are then created, with points performing their three actions, and any subelements responsible for identifying ridge points (depending on c) performing their necessary actions. In this way, the creation of a new grid element automatically initiates ridge-finding procedures.

5.4. Specific marching strategies

Extending a ridge is similar to the initial search for a ridge point, except that instead of exploring all neighboring grid elements, Marching Ridges only searches grid elements into which the ridge extends and thus depends on d.

Once ridge points have been found in a particular grid element, Marching Ridges determines the existence of a ridge in each of the border elements of the grid element. Since border elements are shared by two grid elements, each border element that contains the ridge identifies a neighboring grid element into which the ridge extends. The anchor vertex of each of these grid elements is added to a queue of vertices in which Marching Ridges will search for ridge points.

5.5. Curve ridges

All 1D ridges are curves, and share the same topological problems, the same heuristics, and the same manner of identifying, continuing, and ending the ridge.

5.5.1. Topological problems. Damon [3] has shown that the maximum convexity height ridge does not generically branch while Miller [9] has shown the same for optimal scale ridges. Marching Ridges uses this assumption even in the case of more general optimal parameter height ridges, for which comparable results are not fully known. To implement this topological constraint, Marching Ridges assumes that any grid element containing a piece of a 1D ridge will only contain two ridge points among its border elements.

5.5.2. Heuristic solutions. The solution to a grid element that identifies too many ridge points is to choose the two ridge points that are most convex, where convexity C is defined as the magnitude of the product of second derivatives in each of the transverse directions.

$$C = \prod_{i=1}^{g-d} | \vec{v}_i H(f) \vec{v}_i^t |$$

This heuristic is the maximum convexity heuristic and assumes that the most convex ridge points are the ridge points of greatest interest to the user.

5.5.3. Finding an initial ridge segment. During the initial search for a ridge, a grid element will only identify a ridge if at least two of its border elements identify ridge points. Having identified at least two ridge points, it will then choose the two most convex ridge points from among all those found and report success for the search.

5.5.4. Continuing the ridge. Having identified the first grid element containing a ridge, each border element of the grid element containing a ridge point identifies a neighboring grid element in which to extend the ridge

(Figure 9). The anchor of each of these grid elements is entered into a list of grid elements in which Marching Ridges will search for ridge points. Each such grid element is then required to identify one other ridge point (the exit point) to continue the ridge. If it identifies more than one ridge point among its border elements, it chooses the most convex as the extension of the ridge. This border element in turn identifies a grid element whose anchor is entered into the list. This continues until the ridge ends.

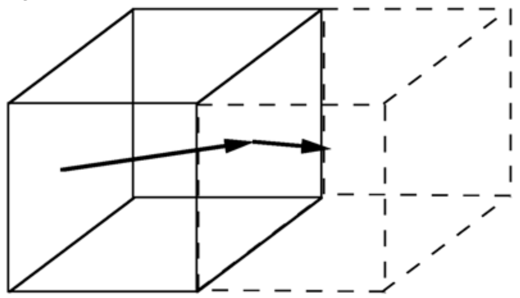
Figure 9. A curve ridge continuing into an adjacent cube

5.5.5. Ending the ridge. The ridge can end in three ways. It may exit the image when a border element containing a ridge point is unable to identify a neighboring grid element contained in the image. It may close on itself, when a grid element is created with two ridge points already identified among its border elements. Finally, the ridge may end when a grid element with one ridge point (the entry point) fails to identify another ridge point (the exit point).

5.6. Surface ridges

All 2D ridges are surfaces and share the same topological problems, the same heuristics, and the same manner of identifying, continuing, and ending the ridge.

5.6.1. Topological problems. The main topological problem with finding a piece of a surface ridge in a grid element is that each border element finds ridge segments independently of the other border elements. Ideally, the ridge segments in each border element of a particular grid element would connect to form a closed loop: the border of the ridge contained in that grid element. However, with each border element finding ridges independently, there is no guarantee that the ridge segments will connect. Further, there is the same problem of topological ambiguity that appears in the marching cubes algorithm where there are multiple possible connections among ridge segments.

5.6.2. Finding an initial ridge patch. The basic step in identifying an initial ridge patch in a grid element is having the border elements identify ridge curves. This is done exactly as indicated in Section 5.5, with the exception that border elements are finding ridges segments, not the original grid element. If any border element of the grid element finds a ridge segment, the grid element contains a ridge patch, and the search has succeeded. The ridge is then continued.

5.6.3. Continuing the ridge. As with 1D ridges, having identified the first grid element containing a ridge, each border element of the grid element containing a ridge identifies a neighboring grid element in which to extend the ridge (Figure 10). The anchor of each of these grid elements is entered into a list of grid elements in which Marching Ridges will search for ridge points. Each such grid element is then required to identify at least one other boundary element containing a ridge (an exit curve) to continue the ridge. If it identifies more than one ridge curve among its border elements, the ridge continues into all such adjacent grid elements. This continues until the ridge ends.

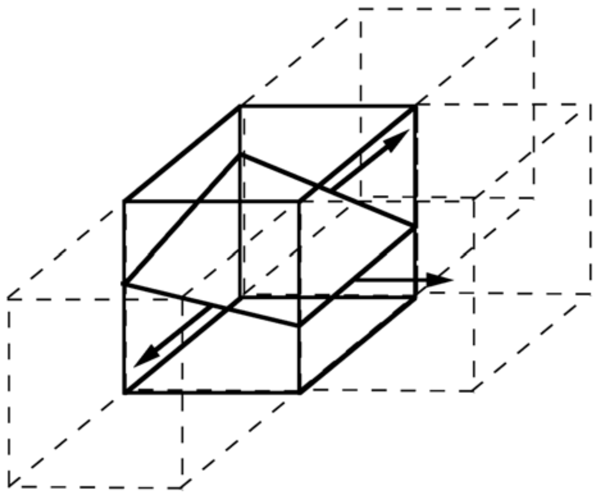
Figure 10. A surface ridge continuing into adjacent cubes

5.6.4. Ending the ridge. A 2D ridge does not “end” in the intuitive way that a 1D ridge does. Rather, the 2D ridge encounters its boundaries. This may happen in two ways. The ridge may exit the image when a border element containing a ridge is unable to identify a neighboring grid element contained in the image. Or the ridge may end when a grid element fails to identify any border elements containing the ridge except for the border elements that contain the incoming ridge. This is the same result that occurs when a 2D ridge closes.

6. Complexity of marching ridges

6.1. Time complexity

Marching Ridges runs two major loops. The first searches for a ridge and the second extends an identified ridge. Given a good approximation by the user for an initial ridge point, the search loop will end quickly, so we will concentrate on the extension loop. This loop will iterate once for each grid element that contains a ridge. Let p be the number of ridge points. Each grid element will be processed in the same fashion, in which the bulk of the work is in the optimization of optimal parameters and the calculation of derivatives of medialness. Each grid element after the first will have no more that half of its vertices performing these calculations. If g is the dimension of the search space, then 2^{g-1} is the maximum number of new vertices at each step of extension. Additionally, the identification of ridge points requires the same computations as at vertices of a grid element. Each new grid element identifies no more than 2^{g+c} potential ridge points, where c is the codimension of the ridge. Thus, there are no more than 2^{2g+c-1} optimization and derivative calculations. The number of optimizations depends on the nearness of the initial values to a local maximum, the complexity of the image and the complexity of the medialness weighting function. Experience has shown the optimizer to converge at about 100 iterations; it is coded to stop optimizing after 200 iterations. The number of derivatives calculated depends on the dimension of the search space and the kind of optimization. It will not exceed $\frac{n(n+1)}{2}$ where n is the number of arguments of the weighting function. This limits the number of weighting function applications to $\frac{200+n(n+1)}{2}2^{2g+c-1}$. The final consideration is the support of the weighting function.

6.1.1. Isotropic Laplacian and oriented Laplacian medialness. The isotropic Laplacian and the oriented Laplacian weighting functions are both used for 1D from 3D cores and 2D from 3D cores. They have identically sized supports. If r is the radius of the weighting function and ξ is the extent, then the support of weighting function is $\frac{4}{3}\pi\xi^3 r^3$. This limits the number of voxel operations in the calculation of a ridge to

$$\frac{4}{3}\pi\frac{200+n(n+1)}{2}2^{2g+c-1}\xi^3 r^3$$

6.1.2. Oriented Fritsch medialness. The oriented Fritsch weighting function is used in the calculation of 1D from 3D cores because it is easier to symbolically differentiate. If r is the radius of the weighting function, ξ is the extent, and ρ is the ratio of the radius to the weighting function aperture, then the support of the weighting function is $2\pi r^3\xi\rho(1+\xi\rho)^2$, limiting the number of voxel operations to

$$2\pi\frac{200+n(n+1)}{2}2^{2g+c-1}\xi\rho(1+\xi\rho)^2 r^3$$

The ratio of this to the oriented Laplacian weighting functions is $\frac{3\rho(1+\xi\rho)^2}{2\xi^2}$, and given typical values of $\rho = 0.25$ and $\xi = 4$, the oriented Fritsch weighting function is about 10 times as fast to apply.

6.1.3. Oriented Morse medialness. The oriented Morse weighting function is used in the calculation of 2D from 3D cores because of its very small support. If r is the radius of the weighting function, ξ is the extent, and ρ is the ratio of the radius to the weighting function aperture, then the support of the weighting function is $\frac{8}{3}\pi\xi^3\rho^3 r^3$, limiting the number of voxel operations to

$$\frac{8}{3}\pi\frac{200+n(n+1)}{2}2^{2g+c-1}\xi^3\rho^3 r^3$$

The ratio of this to the oriented Laplacian weighting functions is $2\rho^3$, and given a typical value of $\rho = 0.25$, the oriented Morse weighting function is about 32 times as fast to apply.

6.2. Space complexity

Marching Ridges use $3+g$ data structures in the calculation of ridges. The first is the list of grid elements waiting to be searched. The second is the list of ridge points found. The sum of their lengths will not exceed p, the number of ridge points in the final ridge. The other $g+1$ data structures are hash tables that contain all the grid elements and their subelements calculated in the course of finding a ridge. This limits the number of elements to a linear function of p dependent on the grid dimension g.

All of the data structures have linear access time and do not affect the time complexity of the algorithm.

7. Summary

This paper described a general algorithm for identifying optimal parameter and maximum convexity height ridges. It describes the algorithm in a dimensionally neutral way, showing that ridge finding strategies are generally dimensionally based while marching strategies are generally codimensionally based. This provides a basis for arbitrary extensions to the algorithm.

8. Acknowledgements

The authors would like to David Eberly and Daniel Fritsch for their helpful discussions on ridge marching. We would also like to thank the entire MIDAG for being willing to listen to explanations of the numerous iterations of this algorithm. The work in this paper was partially supported by NIH grant PO1 CA47982.

9. References

[1] J. Bloomenthal, "Polygonization of Implicit Surfaces", *Computer Aided Geometric Design*, 1988, pp. 341-355.
[2] J. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligenc*, 1986, pp. 679-698.
[3] D. Eberly and R. Gardner and B. Morse and S. Pizer and C. Scharlach, "Ridges for Image Analysis", *Journal of Mathematical Imaging and Vision*, v. 4, 1994, pp. 351-371
[4] J. Damon, "Properties of Ridges and Cores for Two Dimensional Images", *Journal of Mathematical Imaging and Vision*, 1999, pp. 163-174.
[5] M. Fidrich, "Following Feature Lines Across Scale", *Proceedings of the First International Conference on Scale Space*, 1997, pp. 140-151.
[6] J. Furst, S. Pizer, and D. Eberly, "Marching Cores: a Method for Extracting Cores from 3D Medical Images", *Proceedings of the Workshop on Mathematical Methods in Biomedical Image Analysis*, 1996, pp. 124-130.
[7] R.M. Haralick, "Ridges and Valleys on Digital Images", *Computer Vision, Graphics and Image Processing*, 1983, pp. 28-38.
[8] W. E. Lorenson and H.E. Cline, "Marching Cubes: a High Resolution 3D Surface Construction Algorithm", *Computer Graphics*, July 1987, pp. 163-169.
[9] J. Miller and J. Furst, "The Maximal Scale Ridge", *Proceedings of the Second International Conference of Scale Space*, 1999, pp. 83-104.
[10] S. M. Pizer and D. Eberly and B. S. Morse and D. S. Fritsch, "Zoom-Invariant Vision of Figural Shape: The Mathematics of Cores", *Computer Vision and Image Understanding*, v. 69, 1998, pp. 55-71
[11] X. Qu and X. Li, "A 3D Surface Tracking Algorithm", *Computer Vision and Understanding*, 1996, pp. 147-156.
[12] G. Stetten and S. M. Pizer, "Medial Node Models to Identify and Measure Objects in Real-Time 3D Echocardiography", *IEEE Transactions on Medical Imaging*, 1999, pp. 1025-1034.
[13] J. Thirion and A. Gourdon, "The 3D Marching Lines Algorithm", *Graphical Models and Image Processing*, 1996, pp. 503-509.