

Fast Scale Space Ridge Detection using Integral Image

Project Report

November 23, 2012

By: Tal Amir, Boris Cherevatsky
CIS Lab, Computer Science Department, Technion, Israel.

Supervisor: Dr. Renen Adar
Rafael Advanced Defense Systems LTD.

Prof. in charge: Ehud Rivlin
CIS Lab, Computer Science Department, Technion, Israel.

Contents

1	Theoretical aspects of ridge detection	2
1.1	Single scale ridges	2
1.2	Ridge strength functions	4
1.3	Scale Space and Scale Space Ridges: Lindeberg's work	4
1.4	Our method: Scale space derivatives via Integral Image	5
1.5	Developing the formulas for ridge detection	6
1.6	How to choose a direction of v_{max} ?	9
2	Ridge point detection: The basic algorithm	10
2.1	Preliminary calculations	10
2.2	Detection of zero-crossings	11
3	Phase 1: Single-scale preprocessing	13
3.1	Strength of a ridge fragment	14
3.2	Connected components	14
3.3	The importance of rank-2 points	17
3.4	A more flexible way for calculating the strength of ridge curves or connected components	17
4	Phase 2: Ridge point processing	17
4.1	Getting rid of junk	18
4.2	Sorting the ridge points from all scales	18

5	Phase 3: Ridge-curve construction	18
5.1	Picking a starting point	19
5.2	Constructing an output ridge curve	19
5.3	Jumping between scales	20
5.3.1	How to jump	20
5.3.2	To jump or not to jump?	21
5.3.3	More considerations of whether to jump or not, and how to obtain the data they require	21
5.4	Reaching a dead end	22
5.5	After having finished constructing a ridge curve	22
5.6	How to determine the ridge width	22
5.7	Maintaining a ridge domain record	23

1 Theoretical aspects of ridge detection

1.1 Single scale ridges

Suppose we are given an image $I(x, y)$ and in it we would like to detect ridges of varying widths. First, let us examine ridge points in a single scale, according to Lindeberg’s definition.

Let (x_0, y_0) be a point on the xy -plane, at which $I(x, y)$ is defined. Let us denote by $HI(x, y)$ the Hessian matrix of I at (x, y) . Let λ_{max} and λ_{min} the maximal and minimal eigenvalues of $HI(x, y)$ respectively¹, such that $|\lambda_{max}| \geq |\lambda_{min}|$. Let v_{max} and v_{min} be corresponding unit eigenvectors of $HI(x, y)$. Using these notations, we say that (x_0, y_0) is a bright / dark ridge point of I if it satisfies the following conditions:

1. $\langle \nabla I(x_0, y_0), v_{max} \rangle = 0$
2. $\lambda_{max} < 0$ (for a bright ridge point)
 $\lambda_{max} > 0$ (for a dark ridge point)

This definition can be justified using two explanations: One is by local derivatives, the other is by principal curvatures.

In terms of local derivatives, the rationale of this definition is as follows: If we look at the image I as a surface in 3d-space represented by $(x, y, I(x, y))$, we can examine a particular point $P_0 = (x_0, y_0, I(x_0, y_0))$ on that surface and look for “good” candidate directions as the direction of a ridge. For the sake of simplicity, suppose we are interested only in a bright ridge. That being said, we would expect that if $u = (u_1, u_2)$ is a vector on the xy -plane that points at a good direction for a bright ridge at P_0 , then intersecting our surface at that point with a plane that is perpendicular to the xy -plane and to the vector u , would result in a graph of a 1-dimensional function floating above the xy -plane, which obtains a local maximum at point (x_0, y_0) . First, this implies that the 1st derivative of the function at (x_0, y_0) should equal zero. Second, for the ridge to be the most pronounced, we would expect that the local maximum be the sharpest, or that the 2nd derivative of that function at (x_0, y_0) be as negative as possible.

Let us make that argument more formal. As said, we examine our surface by intersecting it with various planes which are:

¹Throughout this work, when we say maximal / minimal eigenvalue, we mean that the absolute value is maximal / minimal.

1. Perpendicular to the xy -plane.
2. Contain the point $P_0 = (x_0, y_0, I(x_0, y_0))$.
3. Parallel to a given unit vector $v = (v_1, v_2)$ (which is normal to the vector u we used before).

Each such intersection gives rise to a graph of a 1-dimensional function, rotated around the z -axis. From multivariable calculus, it can be shown that the 1st and 2nd derivatives of that function at point (x, y) are exactly the 1st and 2nd directed derivatives of I along v , which are:

1. $\frac{\partial I}{\partial v}(x_0, y_0) = \langle \nabla I(x_0, y_0), v \rangle$
2. $\frac{\partial^2 I}{\partial v^2}(x_0, y_0) = v^T H I(x_0, y_0) v$

Now, suppose we would like to choose a direction v for the intersecting plane, such that the resulting 1-dimensional function would obtain the sharpest local-maximum possible at point (x_0, y_0) . The first requirement is, of course, that the 1st derivative of the function at (x_0, y_0) be zero, meaning: $\langle \nabla I(x_0, y_0), v \rangle = 0$, which is similar to condition (1) for (x_0, y_0) being a ridge point. Now, for (x_0, y_0) to be a local maximum, we need to require that the 2nd derivative be negative, or: $v^T H I(x_0, y_0) v < 0$. In order to obtain the sharpest local minimum, we seek to minimize that expression. From the Lagrange Multipliers theorem, it turns out that the minimum and maximum of $v^T H I(x_0, y_0) v$ s.t. $\|v\| = 1$ are admitted at eigenvectors of $H I(x_0, y_0)$. Let us denote by λ_{max} and λ_{min} the maximal and minimal eigenvalues, with corresponding unit eigenvectors v_{max} and v_{min} . Note that the 2nd derivatives along v_{max} and v_{min} are exactly λ_{max} and λ_{min} , as we can easily see, for example, with v_{max} :

$$v_{max}^T H I(x_0, y_0) v_{max} = v_{max}^T \lambda_{max} v_{max} = \lambda_{max} \|v_{max}\|^2 = \lambda_{max}$$

In seeking a negative 2nd directed derivative, it would make sense to examine solely the direction v_{max} , which obtains the maximal magnitude of 2nd derivative. The reason for that is that if the strongest negative 2nd derivative is λ_{min} , with $|\lambda_{min}| < |\lambda_{max}|$ and $\lambda_{max} > 0$, then we have a perpendicular direction v_{max} along which there is a local directed minimum which is “sharper” than the local directed maximum along v_{min} . Intuitively, that makes (x_0, y_0) not a very good candidate for a ridge point. Therefore, we choose to examine the direction v_{max} and require that along it we have a zero 1st directed derivative and negative 2nd directed derivative, or: $\langle \nabla I(x, y), v_{max} \rangle = 0$ and $\lambda_{max} < 0$, which are conditions (1) and (2) respectively.

In terms of principal curvatures, the explanation is easier: If we assume that P_0 satisfies the definition of a ridge point, and is not an umbilical point of the image surface, then it obtains two distinct principal curvatures with two perpendicular principal directions. It can be shown that the projection of these directions on the xy -plane are parallel to v_{max} and v_{min} , and that the corresponding principal curvatures are $\kappa_{max} = \frac{\lambda_{max}}{\sqrt{1+\|\nabla I\|^2}}$ and $\kappa_{min} = \frac{\lambda_{min}}{(1+\|\nabla I\|^2)^{3/2}}$. It makes sense to look at the principal direction with the larger principal curvature, which is v_{max} , and to require that its corresponding principal curvature κ_{max} be negative, which implies that λ_{max} is also negative. Intuitively, if we stand right at the middle of a bright ridge curve of a surface, and we walk on a short line perpendicular to the ridge, then our height increases a before crossing the ridge and decreases after crossing the ridge. Thus, the direction perpendicular to the ridge gives us a zero directed derivative of the height function, and thus is perpendicular to the gradient of that surface. Thus, $\langle \nabla I(x, y), v_{max} \rangle = 0$.

One last important thing is, as we saw in these two explanations, the ridge is perpendicular to v_{max} at each of its points, and thus the ridge direction is always parallel to v_{min} .

1.2 Ridge strength functions

To our assistance, Lindeberg defined three ridge strength functions, which we introduce here in a slightly simpler version than the original one:

1. $\mathcal{M}(x, y) = |\lambda_{max}(x, y)|$
2. $\mathcal{N}(x, y) = \lambda_{max}(x, y)^2 - \lambda_{min}(x, y)^2$
3. $\mathcal{A}(x, y) = (\lambda_{max}(x, y) - \lambda_{min}(x, y))^2$

The basic rationale behind these functions is that the 2nd directed derivative, perpendicular to the ridge, λ_{max} , should be as strong as possible. The added sophistication of \mathcal{N} and \mathcal{A} are that we require that the surface be curved along one direction and much less curved (in \mathcal{N}) or even curved the other way (in \mathcal{A}) along the perpendicular direction.

Lindeberg states that the function \mathcal{N} is more ridge-specific than the two others². Empirical observations made in this work lead to the same conclusion, as well as a theoretical reasoning: \mathcal{N} promotes points that have a strong 2nd directed derivative along v_{max} , while penalizing points that have a strong 2nd directed derivative along v_{min} , thus resembling more a vertex of an elliptic dome rather than a ridge point. Therefore, \mathcal{N} will be our strength-function of choice in this work.

1.3 Scale Space and Scale Space Ridges: Lindeberg's work

Now let us introduce the term of scale space. In his work, Tony Lindeberg used a Gaussian scale space representation of the image:

$$L(x, y, t) = I(x, y) * \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

with t being the scale parameter. Using this definition, let us define the following notations: We denote by $\nabla L(x, y, t)$ the gradient of $L(x, y, t)$ with respect to the variables x, y , $HL(x, y, t)$ is the Hessian matrix of $L(x, y, t)$ with respect to x, y and λ_{max} , λ_{min} , v_{max} and v_{min} are the eigenvalues and corresponding unit eigenvectors of $HL(x, y, t)$. By using our scale space definition of λ_{max} and λ_{min} , we can also extend the definition of our strenght function the following way:

$$\mathcal{N}(x, y, t) = \lambda_{max}(x, y, t)^2 - \lambda_{min}(x, y, t)^2$$

Now we shall seek to detect ridges of varying widths using the notation of scale space. Lindeberg's definition of a scale space ridge point (x_0, y_0, t_0) requires that the point satisfies the following conditions:

1. $\langle \nabla L(x_0, y_0, t_0), v_{max} \rangle = 0$
2. $\lambda_{max}(x_0, y_0, t_0) < 0$ (for a bright ridge point)
 $\lambda_{max}(x_0, y_0, t_0) > 0$ (for a dark ridge point)
3. $\mathcal{N}(x, y, t)$ obtains a local directed maximum at (x_0, y_0, t_0) along the t -axis.

²[Lind-96, page 28]

One can look at the set of all scale space ridge points as the intersection of two surfaces in 3D space: The first surface is the set of all points (x, y, t) that satisfy conditions (1) and (2). These are exactly the set of ridge points as defined for a single scale image, after performing the appropriate smoothing of the image. The second surface is the set of points which are local maxima of the strength function $\mathcal{N}(x, y, t)$ along the t -axis.

1.4 Our method: Scale space derivatives via Integral Image

Since working with linear scale space requires performing Gaussian smoothing for each scale, we decided to work with a different kind of scale space. Let us keep the notation of gradient, Hessian matrix, eigenvalues and eigenvectors as we used in the case of a single scale. All these measurements are calculated via convolving the image $I(x, y)$ with numerical differentiation filters. Here are the filters that we used:

$$\begin{aligned}\frac{\partial}{\partial x} &= \frac{1}{8} \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \\ \frac{\partial}{\partial y} &= \frac{1}{8} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \\ \frac{\partial^2}{\partial x^2} &= \frac{1}{4} \begin{pmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{pmatrix} \\ \frac{\partial^2}{\partial y^2} &= \frac{1}{4} \begin{pmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{pmatrix} \\ \frac{\partial^2}{\partial x \partial y} &= \frac{1}{4} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix} \\ \nabla^2 &= \frac{1}{8} \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}\end{aligned}$$

Our term of *scale* refers to a completely different idea: Instead of performing the appropriate Gaussian smoothing prior to applying the numerical differentiation filters, we apply an *augmented* version of the differentiation filters to the original image. Suppose we would like to calculate the derivatives of I in scale σ , with $\sigma \in \mathbb{N}$. Then each element of the basic differentiation filters will be replaced by a sub-block of size $\sigma \times \sigma$ that contains the number $\frac{1}{\sigma^2}$ at each of its elements. The meaning of this is quite simple: Applying a differentiation filter at scale σ means that each of the filter's elements represents a coefficient to be multiplied by the average value of the image on a square region of size $\sigma \times \sigma$. For example, the $\frac{\partial}{\partial x}$ filter at scale 3 would

be:

$$\frac{\partial}{\partial x} = \frac{1}{8} \left(\begin{array}{ccc|ccc|ccc} -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \\ -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \\ -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \\ \hline -2/9 & -2/9 & -2/9 & 0 & 0 & 0 & 2/9 & 2/9 & 2/9 \\ -2/9 & -2/9 & -2/9 & 0 & 0 & 0 & 2/9 & 2/9 & 2/9 \\ -2/9 & -2/9 & -2/9 & 0 & 0 & 0 & 2/9 & 2/9 & 2/9 \\ \hline -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \\ -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \\ -1/9 & -1/9 & -1/9 & 0 & 0 & 0 & 1/9 & 1/9 & 1/9 \end{array} \right)$$

Instead of simply convolving the image with such filters at each scale, we can do a trick to speed-up things significantly. Let us first define the Integral Image of image I :

$$II(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} I(x', y')$$

Note that by sampling the integral image at four points, we can calculate the sum of I on any square region that is parallel to the axes, using this formula:

$$\sum_{\substack{x_0 \leq x' \leq x_1 \\ y_0 \leq y' \leq y_1}} I(x', y') = II(x_0 - 1, y_0 - 1) + II(x_1, y_1) - II(x_0 - 1, y_1) - II(x_1, y_0 - 1)$$

Thus, we can use the integral image in order to apply the numerical differentiation filters to the image at any scale in a rapid way: Each of our filters consists of 9 nonzero elements at most. Each of them requires averaging the values of I on a square region, which requires sampling II at four points. Thus, calculating one numerical derivative of I at a certain pixel requires us nothing more than to sample the integral image at 45 regions at most and sum the results after multiplying them with the appropriate coefficients which appear in the filter's elements. Note that this is independent of how big the scale σ is. This means that for each scale, we can calculate all the numerical derivatives of I in linear time complexity.

In order for a center pixel to be properly defined at each sub-block, we will only use odd scale numbers. Empirically, using filters with even scales yields values that are quite different than the two odd adjacent scales, which further justifies this choice.

1.5 Developing the formulas for ridge detection

In our earlier discussion, we used the notions of λ_{max} , λ_{min} , v_{max} and v_{min} . Here we shall develop closed-form expressions of these values, using the partial derivatives of the image. In order to remind ourselves that we are working in scale space, we will refer to the image as $L(x, y)$, keeping in mind that we are calculating its partial derivatives in a certain scale.

First we need to calculate the eigenvalues of HL :

$$HL = \begin{pmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{pmatrix}$$

For this, we need to solve the following equation:

$$0 = \text{Det} \begin{pmatrix} L_{xx} - \lambda & L_{xy} \\ L_{xy} & L_{yy} - \lambda \end{pmatrix} = (L_{xx} - \lambda)(L_{yy} - \lambda) - L_{xy}^2 =$$

$$\lambda^2 - (L_{xx} + L_{yy})\lambda + L_{xx}L_{yy} - L_{xy}^2$$

Which gives us:

$$\lambda_{1,2} =$$

$$\frac{L_{xx} + L_{yy} \pm \sqrt{(L_{xx} + L_{yy})^2 - 4(L_{xx}L_{yy} - L_{xy}^2)}}{2} =$$

$$\frac{L_{xx} + L_{yy} \pm \sqrt{(L_{xx} - L_{yy})^2 + 4L_{xy}^2}}{2} =$$

$$\frac{L_{xx} + L_{yy}}{2} \pm \sqrt{\left(\frac{L_{xx} - L_{yy}}{2}\right)^2 + L_{xy}^2}$$

In order to obtain the maximal eigenvalue, we need to add the root with the same sign as the sign of $\frac{L_{xx} + L_{yy}}{2}$, and in order to choose the minimal eigenvalue, we need to use the opposite sign. Let us make the following notations:

$$A = \frac{L_{xx} + L_{yy}}{2}$$

$$\sigma = \text{Sgn}(A)$$

(not to be confused with the scale σ)

$$\Delta = \frac{L_{xx} - L_{yy}}{2}$$

$$S = \sqrt{\Delta^2 + L_{xy}^2}$$

Then we have:

$$\lambda_{max} = A + \sigma S$$

$$\lambda_{min} = A - \sigma S$$

Now we need to find v_{max} . Here things get more sophisticated. Since the eigenspace of λ_{max} is a whole line that passes through the origin, a unit vector v_{max} can only be defined up to multiplication by -1 , meaning that there is no way to properly define one direction for v_{max} . We propose two candidates for vectors in the eigenspace of λ_{max} :

$$v_1 = (\Delta + \sigma S, L_{xy})$$

$$v_2 = (L_{xy}, -\Delta + \sigma S)$$

Let us now show that v_1 is indeed an eigenvector of HL with eigenvalue λ_{max} :

$$HL \cdot v_1 =$$

$$\begin{aligned}
& \begin{pmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{pmatrix} \begin{pmatrix} \Delta + \sigma S \\ L_{xy} \end{pmatrix} = \\
& \begin{pmatrix} L_{xx}(\Delta + \sigma S) + L_{xy}^2 \\ L_{xy}\Delta + L_{xy}\sigma S + L_{xy}L_{yy} \end{pmatrix} = \\
& \begin{pmatrix} L_{xx}(\Delta + \sigma S) + \sigma^2 S^2 - \Delta^2 \\ \frac{L_{xy}L_{xx} - L_{xy}L_{yy}}{2} + L_{xy}\sigma S + L_{xy}L_{yy} \end{pmatrix} = \\
& \begin{pmatrix} \sigma S(L_{xx} + \sigma S) + \Delta L_{xx} - \Delta^2 \\ L_{xy}\frac{L_{xx} + L_{yy}}{2} + L_{xy}\sigma S \end{pmatrix} = \\
& \begin{pmatrix} \sigma S(A + \Delta + \sigma S) + \Delta(A + \Delta) - \Delta^2 \\ L_{xy}A + L_{xy}\sigma S \end{pmatrix} = \\
& \begin{pmatrix} \sigma S(\Delta + \sigma S) + A\sigma S + A\Delta \\ (A + \sigma S)L_{xy} \end{pmatrix} = \\
& \begin{pmatrix} (A + \sigma S)(\Delta + \sigma S) \\ (A + \sigma S)L_{xy} \end{pmatrix} = \\
& (A + \sigma S) \begin{pmatrix} (\Delta + \sigma S) \\ L_{xy} \end{pmatrix} = \\
& (A + \sigma S)v_1 = \lambda_{max}v_1
\end{aligned}$$

In order to show that v_2 is also an eigenvector with eigenvalue λ_{max} , it is enough to show that they are both parallel. Their vector product yields:

$$(\Delta + \sigma S)(-\Delta + \sigma S) - L_{xy}^2 = \sigma^2 S^2 - \Delta^2 - L_{xy}^2 = L_{xy}^2 - L_{xy}^2 = 0$$

which proves our claim.

It is very important that we remember two facts about the vectors v_1 and v_2 , which we have just proven to be eigenvectors:

1. v_1 and v_2 are not unit eigenvectors. If we want to use one of them for the role of v_{max} in the previous discussion, we need to normalize it first, and take into consideration the fact that they are not necessarily nonzero.
2. It is still not clear which one of them to choose for the direction of v_{max} . This will be discussed in the next section.

One last calculation we need is the ridge strength function:

$$\begin{aligned}
\mathcal{N} &= \\
& \lambda_{max}^2 - \lambda_{min}^2 = \\
& (A + \sigma S)^2 - (A - \sigma S)^2 = \\
& 4A\sigma S = \\
& 4|A|S = \\
& |L_{xx} + L_{yy}| \left((L_{xx} - L_{yy})^2 + 4L_{xy}^2 \right)
\end{aligned}$$

1.6 How to choose a direction of v_{max} ?

The basic ridge detection algorithm we perform works separately at each scale and tries to detect points which satisfy the conditions for a ridge point, as described in section 1.1 on page 2. All the partial derivatives for that purpose are calculated in the scale we're working in. The basic idea is to look for zero-crossing points of the scalar product $\langle \nabla L, v_{max} \rangle$. For this we employ an algorithm similar to marching-square. We compare the values of that product at two adjacent points, and whenever it changes sign in these two points, we detect the zero-point that lies between them via linear interpolation. So far - so good. However, there is a slight difficulty. Suppose p_1 and p_2 are two adjacent points on the plane, and that either of these two scenarios happen:

1. We use v_1 at p_1 and v_2 at p_2 , but v_1 at p_1 faces the opposite direction of v_2 at p_2 .
2. We use v_1 as the direction of v_{max} in both points, but v_1 rotates in more than 180° when moving from p_1 to p_2 .

Any of these cases would lead us to detect a "zero-point" of $\langle \nabla L, v_{max} \rangle$ that we have created on our own, due to a poor or inconsistent choice of direction for v_{max} in the two points, which made the scalar product change sign artificially. We would like to avoid this. The full details of the solution to this problem will be given in the next section. This is just to demonstrate the tremendous importance of a wise choice for v_1 and v_2 . Failure to do so would generate fake ridge points.

Let us examine the scalar product of v_1 and v_2 :

$$\langle v_1, v_2 \rangle = L_{xy} (\Delta + \sigma S - \Delta + \sigma S) = 2\sigma S L_{xy}$$

While S is always non-negative, σL_{xy} can hold negative values, meaning that v_1 and v_2 might face opposite directions even at the same point, implying that a poor choice of v_{max} would **guarantee** that we generate a false zero-crossing.

Let's examine what happens when one of the vectors, for example, v_1 , is zero. This would immediately imply that $L_{xy} = 0$, which would further imply that $S = |\Delta|$. Assigning this to $\Delta + \sigma S = 0$ would yield that $|\Delta|(\sigma + Sgn(\Delta)) = 0$.

Now, what if both vectors v_1 and v_2 would equal zero simultaneously? Can this possibly happen? If it happens, it would imply:

$$L_{xy} = 0$$

$$|\Delta|(\sigma + Sgn(\Delta)) = 0$$

$$|\Delta|(\sigma - Sgn(\Delta)) = 0$$

which would imply $\Delta = \sigma = 0$, which would further imply that $L_{xx} = L_{yy} = L_{xy} = 0$. This means that we are at a planar point, which does not satisfy the conditions for a ridge point anyway (since $\lambda_{max} = 0$). Therefore, at a ridge point, at least one of the vectors $\{v_1, v_2\}$ has to be nonzero. Which one should we choose as the direction of v_{max} ? At first guess, we might want to choose at each point the vector with the higher norm for the direction of v_{max} . Looking closely at the expressions of v_1 and v_2 , we can see that if $Sgn(\Delta) = \sigma$ then v_1 would have the higher norm of the two and if $Sgn(\Delta) = -\sigma$ then v_2 would have the higher norm.

Note that even though this choice appears to make sense, it would still not solve our problem. The reason for this is deeper than merely a difficulty in choosing a direction for an eigenvector of

HL. As this project's supervisor, Dr. Renen Adar, has shown, it is impossible to algorithmically create a single function $f(x, y)$ such that its zero crossings would give us the ridge curves of an image $I(x, y)$, using only local differential operators. A sketch of the proof of this claim can be found in the presentation slides of our project.

2 Ridge point detection: The basic algorithm

Here we present the basic algorithm for detecting ridge points and ridge curves. This algorithm is run for each scale separately. Here is a basic description of what it does:

1. Take a greyscale image I of size $M \times N$ as input.
2. Calculate its 1st-order and 2nd-order numerical differentials ($L_x, L_y, L_{xx}, L_{xy}, L_{yy}$) in the desired scale, using Integral Image, as shown in section 1.4.
3. Calculate: $\lambda_{max}, \lambda_{min}, v_1, v_2$
4. Detect zero-crossing points of the scalar products between each pair of adjacent points.

2.1 Preliminary calculations

All input and output arguments are 2d numeric arrays of size $M \times N$.

Input

$L_x, L_y, L_{xx}, L_{xy}, L_{yy}$ the partial derivatives in the current scale of our choice.

Output

$\lambda_{max}, \lambda_{min}$ The two eigenvalues of HL .

v1x, v1y, v2x, v2y The x and y components of the vectors v_1 and v_2 .

norm1, norm2 The norms $\|v_1\|$ and $\|v_2\|$.

The algorithm

Initialize double scalars **A, S, Δ, σ , NFunc**.

For each point (x, y) , do:

$$\begin{aligned}
 A &= (L_{xx}(x, y) + L_{yy}(x, y)) / 2 \\
 \Delta &= (L_{xx}(x, y) - L_{yy}(x, y)) / 2 \\
 \sigma &= \text{Sgn}(A) \\
 S &= \sqrt{\Delta^2 + L_{xy}(x, y)^2} \\
 \lambda_{max}(x, y) &= A + \sigma S \\
 \lambda_{min}(x, y) &= A - \sigma S \\
 v1x(x, y) &= \Delta + \sigma S \\
 v1y(x, y) &= L_{xy}(x, y) \\
 v2x(x, y) &= L_{xy}(x, y) \\
 v2y(x, y) &= -\Delta + \sigma S
 \end{aligned}$$

$$\begin{aligned}\text{norm1}(x,y) &= \sqrt{v_{1x}(x,y)^2 + v_{1y}(x,y)^2} \\ \text{norm2}(x,y) &= \sqrt{v_{2x}(x,y)^2 + v_{2y}(x,y)^2}\end{aligned}$$

2.2 Detection of zero-crossings

This algorithm described in the previous section gives us all the information we need in order to detect ridge points at the desired scale. Now, how do we use it to do so? Let us look again at the conditions for a point (x_0, y_0) to be a single-scale ridge point:

1. $\langle \nabla L(x_0, y_0), v_{max} \rangle = 0$
2. $\lambda_{max} < 0$ (for a bright ridge point)
 $\lambda_{max} > 0$ (for a dark ridge point)

Suppose we are seeking bright ridge points. Then these points are exactly the zero-crossing points of the scalar product $\langle \nabla L(x_0, y_0), v_{max} \rangle$, at which $\lambda_{max} < 0$. Note that if $\lambda_{max} < 0$, then at least one of $\{v_1, v_2\}$ is nonzero, which makes it possible to choose a nonzero direction for v_{max} .

We detect the zero-crossing points by comparing the values of the product above at each horizontal or vertical pair of adjacent points.

- We call (x, y) and $(x + 1, y)$ a *horizontal pair of adjacent points*. Similarly, we call (x, y) and $(x, y + 1)$ a *vertical pair of adjacent points*.
- We call the interval $[x, x + 1] \times \{y\}$ a *horizontal bar at (x, y)* , and $\{x\} \times [y, y + 1]$ a *vertical bar at (x, y)* .

By comparing the values of the scalar product at each pair of adjacent points, we can detect a zero point that lies on the bar between them, by using linear interpolation. The last missing detail of how to do this, is how to choose a direction for v_{max} in a correct manner. At last, here is the solution: Suppose we are seeking a bright ridge point between points $p_a = (x, y)$ and $p_b = (x + 1, y)$. Let $n_1 = \max\{\text{norm1}(p_a), \text{norm1}(p_b)\}$ and $n_2 = \max\{\text{norm2}(p_a), \text{norm2}(p_b)\}$. Note that if $n_1 = n_2 = 0$, as explained earlier, the best we could hope to detect is a planar point, which is not a ridge point, so we can skip this pair of points and move to the next one. If $n_1 \geq n_2$, we will use v_1 at both points. Otherwise, we will use v_2 . If $n_1 \geq n_2$, we define: $v_a = \frac{v_1(p_a)}{\|v_1(p_a)\|}$ and $v_b = \frac{v_1(p_b)}{\|v_1(p_b)\|}$. If $n_1 < n_2$, we define: $v_a = \frac{v_2(p_a)}{\|v_2(p_a)\|}$, $v_b = \frac{v_2(p_b)}{\|v_2(p_b)\|}$. v_a and v_b are our representatives of v_{max} at points p_a and p_b , and we chose them consistently. One last thing that we need to take care of, is if they are facing opposite directions. We would like the sign of the scalar product to change only because ∇L has changed its correlation with the ridge normal, not because the normal itself has rotated 180° . Thus, if $\langle v_a, v_b \rangle < 0$, we negate v_b .

Now we can compare the two products: $prod_a = \langle v_a, \nabla L(p_a) \rangle$, $prod_b = \langle v_b, \nabla L(p_b) \rangle$. If they are of opposite signs, it means we have a zero-crossing. We detect that point with sub-pixel accuracy using linear interpolation: Suppose $prod_a \cdot prod_b < 0$, and let $\alpha = \frac{prod_a}{prod_a - prod_b}$. Then our zero-point is in coordinates: $p_0 = (x + \alpha, y)$.

Now that we have the coordinates of the zero-point, we need to make sure that it's a bright ridge point. For that, it has to satisfy: $\lambda_{max}(p_0) < 0$. If it satisfies this condition, it qualifies as a bright ridge point, with strength $\lambda_{max}(p_0)^2 - \lambda_{min}(p_0)^2$. We determine the values of λ_{max}

and λ_{min} at non-integer coordinates p_0 via linear interpolation. The same goes for dark ridge points. Just replace the requirement of $\lambda_{max}(p_0)$ so that it has to be positive.

Here is a summary of the algorithm:

For each horizontal pair of adjacent points $p_a = (x, y)$ and $p_b = (x, y + 1)$:

```

n1 = max{norm1( $p_a$ ), norm1( $p_b$ )}
n2 = max{norm2( $p_a$ ), norm2( $p_b$ )}

if max(n1, n2) == 0
    There is no ridge point on the bar between  $p_a$  and  $p_b$ ,
    so move on to the next horizontal pair of points.
if n1 >= n2
    vax = v1x(x, y);    vay = v1y(x, y);
    vbx = v1x(x+1, y);  vby = v1y(x+1, y);
    vanorm = norm1(x, y);
    vbnorm = norm1(x+1, y);
else
    vax = v2x(x, y);    vay = v2y(x, y);
    vbx = v2x(x+1, y);  vby = v2y(x+1, y);
    vanorm = norm2(x, y);
    vbnorm = norm2(x+1, y);

vax = vax / vanorm;
vay = vay / vanorm;

vbx = vbx / vbnorm;
vby = vby / vbnorm;

if (vax * vbx + vay * vby) < 0
    vbx = -vbx;
    vby = -vby;

prod_a = vax * Lx(x, y) + vay * Ly(x, y)
prod_b = vbx * Lx(x+1, y) + vby * Ly(x+1, y)

if (prod_a * prod_b >= 0)
    No zero-crossing, so there's no ridge point on the bar
    between  $p_a$  and  $p_b$ , so move on to the next horizontal
    pair of points.

 $\alpha$  = prod_a / (prod_a - prod_b)
lambdaMax0 = (1 -  $\alpha$ )  $\lambda_{max}(x, y)$  +  $\alpha \lambda_{max}(x + 1, y)$ 
lambdaMin0 = (1 -  $\alpha$ )  $\lambda_{min}(x, y)$  +  $\alpha \lambda_{min}(x + 1, y)$ 

```

```

if (lambdaMax0 < 0)

    Add the point  $(x + \alpha, y)$  as a bright ridge point in the current scale,
    with strength:  $(\text{lambdaMax0}^2 - \text{lambdaMin0}^2)$ 
else

    Add the point  $(x + \alpha, y)$  as a dark ridge point in the current scale,
    with strength:  $(\text{lambdaMax0}^2 - \text{lambdaMin0}^2)$ 

```

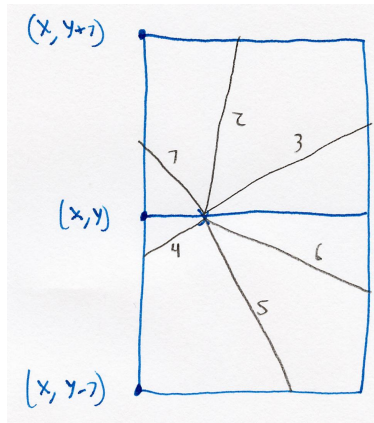
Note that here lambdaMax0 cannot equal zero, otherwise it would imply $n_1 = n_2 = 0$.

Note that this code handles only horizontal pairs of adjacent points, and therefore only detects ridge points which lie on horizontal bars. The same code, with slight adjustments, needs to be run on vertical pairs of adjacent points: $\text{Coordinate}(x + 1, y)$ should be replaced by $(x, y + 1)$ and $(x + \alpha, y)$ should be replaced by $(x, y + \alpha)$.

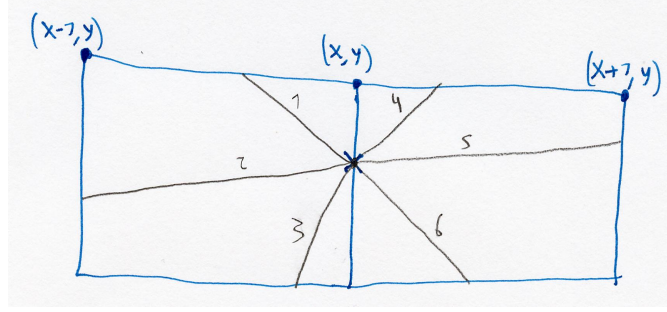
3 Phase 1: Single-scale preprocessing

Before performing the actual ridge detection, that is - stepping along the ridge-fragments and using them as the building blocks for constructing ridge-cruves, there are some preliminary calculations we can perform independently for each scale, right after we've finished to detect ridge points in that scale. Those calculations will assist us later on our job. We call this *the preprocessing phase*. This phase is performed independently on each scale, which means that it can be parallelized.

In Section 2, we explained in detail the algorithm that detects ridge points which lie on horizontal and vertical bars. The first thing we would like to do next is to connect points which are close to one another with edges, so that we have a graph-like data structure. Note that by the way we defined the ridge points and bars, the integer parts of a ridge point's coordinates are exactly the coordinates of the bar it lies on. Now, suppose we have a ridge point on a horizontal bar. That ridge point can have six possible neighbouring ridge point, as shown in the following sketch:



A neighbouring ridge point can be located at each of the four neighbouring vertical bars and two neighbouring horizontal bars. For each such neighbour, we add an edge to our data structure, which connects these two points. Note that we gave each type of edge an index from 1 to 6. By keeping an 8-bit integer for each horizontal bar that has a ridge point on it, we can tell which edges are attached to it. Actually, 6 bits for each bar would be enough. The same thing is done for vertical bars, using this indexing:



We call an edge between two neighbouring ridge points a *ridge fragment*.

Note: In order for the edges to make sense in our purpose of ridge detection, we only create edges that connect bright ridge points with bright ridge points or dark ridge points with dark ridge points.

3.1 Strength of a ridge fragment

A ridge fragment is an interval in the (x, y) plane that connects two neighbouring ridge points. Since each such point has a ridge strength value attached to it, it induces a strength value for the ridge fragment. That is, the mean of the strengths of the two points it connects. Note that each such ridge fragment has a length, so if we construct a curve from the ridge fragments, we can use these values to integrate the ridge strength function on the curve.

3.2 Connected components

While going over all the ridge points and creating the graph, we use a union-find data structure in order to keep track of connected components in our graph. This will assist us later in the ridge detection. Whenever we create an edge between two vertices that belong to two currently-different connected components, we tell the union-find to unite these two components.

Throughout this process of assembling the graph, we also maintain the value of the 1D integral of the strength function on each connected component. Each edge has a length, so these integrals equal the sum of each edge's length multiplied by its strength value, summed for each connected component separately. These values are updated as follows: Suppose we're creating a new edge e that connects two different connected components A and B . Then the strength of the new united connected component is $strength(A) + strength(B) + strength(e)$. Creating an edge e that connect two vertices that already belong to the same connected component A , simply requires adding $strength(e)$ to the strength of A .

Maintaining information about the strength of connected components will help us a great deal. The strength of a single ridge point only gives us local information, which is a bit too

sensitive and inconsistent, whereas the strength of the point's connected component gives us a more global and much more reliable information. Some points appear to be strong points on their own, but their connected component is rather weak, and thus these points should get a lower priority in the ridge detection. One can see this clearly on the examples shown ahead. As we shall explain later, constructing a new ridge curve will always begin from the strongest unused ridge point of the strongest connected component.

Note that by using a similar calculation as we construct the graph, we can maintain the length of each connected component. This is also a measurement that can help us later in the ridge detection.



Aerial photo of a road



Ridge points of all scales.
 coloring by scale,
 color intensity by point strength



Ridge points of all scales.
 Coloring by scale,
 color intensity by connected component strength

3.3 The importance of rank-2 points

There is a rather trivial but yet important underlying observation in the process of ridge detection: The vast majority of points we expect in the output ridge curves are ridge points which have exactly two edges attached to them, or *rank-2 points*. Points with rank 1 (*end points*) or points with a rank higher than 2 (*intersections*) are much less common, and thus interest us less, even though intersections do exhibit the necessity to choose one of the edges in the ridge construction process. We will discuss this in Section 5.

This observation motivates us to give more specific attention to rank-2 points as the main building blocks of our ridge curves. As we will see in the following sections, we will always start constructing a new ridge curve from rank-2 points, and when we consider jumping from one scale to the next, it will only be to rank-2 points, in order to avoid facing the dilemma of choosing a direction right after jumping to a different scale.

3.4 A more flexible way for calculating the strength of ridge curves or connected components

The method we introduced in Section 3.2 for calculating the strength of a connected component C can be formally described by:

$$strength(C) = \int_C strength(x) dx$$

The first modification we can introduce in order to allow us more freedom, is to give less weight to the length of the connected component itself:

$$strength(C) = \frac{\int_C strength(x) dx}{(\int_C dx)^\alpha}$$

with $0 \leq \alpha \leq 1$. With $\alpha = 0$, we have the same formula as before - a simple integration. With $\alpha = 1$, this gives us the average point strength on C . For different values of α between 0 and 1 we can give a different weight to the length of the connected component.

The second modification we can introduce is to use a generalized average:

$$strength(C) = \frac{\left(\int_C (strength(x))^\beta dx\right)^{\frac{1}{\beta}}}{(\int_C dx)^{\alpha/\beta}}$$

With $0 < \beta$. The effect of the choice of β is the same as the effect of choosing the parameter p when calculating the l_p norm of a vector: A higher value promotes connected components with high local peaks of point strength. A lower value promotes connected components in which many points have a moderately-high strength. A value of $\beta = 1$ is indifferent to the distribution of the strength among points, and simply sums it all up.

Recommended values (obtained empirically): $\alpha = 0.5$, $\beta = 0.5$.

4 Phase 2: Ridge point processing

In this phase, now that we've already detected all ridge points and created the graphs of ridge-fragments from all scales, we can perform some further processing in order to refine our database

and improve the results of the final output.

4.1 Getting rid of junk

Using the information we accumulated by now, we can perform some initial filtration and get rid of ridge fragments and ridge points that we know for sure we won't use. Two good candidates for getting rid of are:

- Connected components that contain a very small number of ridge points, i.e. 10.
- Connected components of which the lengths are very short (a few pixel-lengths).

Ridge points that belong to connected components that do not pass these criteria are good candidates to be kicked out of the process at this stage. From our experience, it is very unlikely that such points can be part of a real ridge on the image. Note that these criteria should not be too strict, otherwise it would cause us to miss useful information on the image. We're only using these criteria in order to get rid of pure junk in order to speed-up further processing. Selecting proper (and quite loose) criteria can help us get rid of almost 50% of useless points that we detected in Phase 1.

Note that we do not filter ridge points by their strengths or by the strength of their connected component, as this would imply introducing unnecessary hard thresholds to the algorithm, which we neither want nor need in order to ensure high quality output.

4.2 Sorting the ridge points from all scales

Instead of using thresholds, we chose a safer mode of operation - we always start constructing new ridge curves from the best point that yet unchecked. But how do we choose a "best point"? For this purpose, we create an array that contains all the ridge points from all scales, and sort it in a descending order according to these parameters (from more important to less important):

1. Point rank
2. Strength of connected component
3. Point strength

Sorting according to this order enables us to quickly separate the rank-2 points from all the other points, as well as sort the rank-2 points according such that points of stronger connected components come first, and stronger points within the same connected components come before the weaker ones. Once we have this data structure ready, we can move to the next phase, which is the actual ridge curve construction.

5 Phase 3: Ridge-curve construction

Now that we have the point array ready, as well as the graphs that contain the ridge points and ridge-fragments of each scale, we can finally start to construct actual ridge curves that will be returned in the output of the algorithm. Our method of operation is as follows: We iteratively construct ridge curves by picking a starting point, stepping through ridge-fragments from one

point to the other, and sometimes jump from one scale to the other when it's justified. A ridge point can be used in more than one ridge curve. A ridge-fragment, on the other hand, can only be used in one ridge curve. Thus, when we add a new ridge curve to the output, all the ridge-fragments it uses become unavailable. After we finish constructing a ridge, we accept it to the collection of ridge curves that will be returned in the output, and mark its domain on scale space as occupied. More about the ridge domain records in Section 5.7.

The basic order of operations of this phase is as follows:

1. Pick the best unchecked ridge point to start from, from an array that keeps the points sorted in a special order.
Note: This should always be a rank-2 point. If no such points are left, this phase is done.
2. Start constructing a ridge curve at one of its two available directions, jumping from one scale to the other when needed, while avoiding stepping into the domain of a ridge that we have already created and accepted to the output.
3. When the construction needs to end, start constructing a ridge curve from the same starting point at its other direction (if it's still available).
4. Connect the two resulting curves to form one ridge curve.
5. Mark the ridge-fragments of that curve as unavailable, preventing the algorithm from using them when constructing the next curves.
6. Mark the scale space domain that this curve passes through as occupied by that ridge curve, so ridges that are too close to it (in scale space as well as direction) cannot invade its domain.
7. Go to 1.

5.1 Picking a starting point

The point from which we start the tracking is always a ridge point. At the beginning of this phase, we create an array that contains all the ridge points from all scales and sort them in a descending order according to the following parameters (from more important to less important parameter): Point rank (i.e. the number of fragments attached to it), strength of connected component, point strength. This enables us to quickly separate the rank-2 points from all the other points. It helps us avoid a lot of difficulties as well as obtain a better output if we only use rank-2 points as starting points.

Each time, we pick the strongest rank-2 point of the strongest connected component. Since we created the array at the beginning of the tracking phase, that point might no longer be rank-2, as we might have already used one or both of its ridge-fragments. Therefore, if that point is no longer of rank 2, we proceed to the next point in the array. The point we pick in this phase will be used as the starting point for the current ridge construction.

5.2 Constructing an output ridge curve

Now that we have a starting point, we should start tracking from it along the ridge fragments and construct a ridge curve.

We pick one of the two edges that are attached to that point and start stepping along it. Whenever we reach a point that has a rank higher than 2, we always choose the edge which best correlates with our current direction of progress. We can accumulate a direction of progress by exponentially smoothing the last step we made with the “current” direction of progress at the previous step, using this formula: Suppose the current direction of progress is v_d and that we’ve just made a new step, denoted by v_s . We update v_d using this formula:

$$v_d \leftarrow \alpha v_d + (1 - \alpha) v_s$$

with $0 < \alpha < 1$.

5.3 Jumping between scales

So far we haven’t discussed jumping between scales. Imagine we have just made a step from point p_1 to point p_2 which are both at the same scale, using the rules defined above. What if, at an adjacent scale, we have a “better” point than p_2 ? How do we define a better point? And if it is indeed better, how do we jump to it?

We have found that the best way to do so, is right after stepping from p_1 to p_2 . If p_2 is in scale σ (which we assume to be an odd integer, since, as said before, we only work with odd scales), we sniff for better points at its two adjacent odd scales $\{\sigma - 2, \sigma + 2\}$. At these two scales, we look for ridge points which are at a certain proximity to p_2 . The best way to look for them is by searching on both sides of our ridge curve, along a straight line that is perpendicular to our current ridge curve, at a certain short length of a few pixels, and check if it is intersecting with a ridge fragment.

5.3.1 How to jump

Suppose there is such an intersection at point q in one of the two adjacent scales. Note that q does not necessarily lie directly on a ridge point in our database: It is merely the intersection between our scanning line and a ridge fragment. Thus, let q_2 be the ridge point on the ridge-fragment containing q , for which the vector $q_2 - q$ correlates best with our current direction of progress.

First, let us ignore the question of whether we actually want to change scale and jump from p_2 to q or q_2 , and assume that we do. How do we jump? The simplest way we found, in order to keep the curves consistent, is to “step back” from p_2 to p_1 , from there add q as the next point on the ridge curve, and from there add q_2 . Note that due to the drifting aside that is sometimes caused by changing scale, this means that the step from p_1 to q might contain a component that is normal to the ridge curve. We might want to compensate for that offset, but since the offset accumulates as more and more scale-jumps are performed, it might shift the ridge curve significantly away from the actual object on the image. A good solution might be not to let all the normal offset caused by jump manifest itself immediately, but rather divide it between a few steps during the construction in order to have a more consistent output curve that on one hand, does not suddenly “jump aside”, and on the other hand does not drift away from the objects on the image which the ridge points originated from.

5.3.2 To jump or not to jump?

There are two main factors which we consider important in order to answer the question of whether to jump or not. First one, obviously, is the point strength. Does q have a better ridge point strength than p_2 ? Second is the point's local direction: If we continue stepping from q_2 , will our step direction correlate well with the current direction of progress? Note that since q is not an actual ridge point in our database, we need to evaluate its ridge point strength by using a linear interpolation between the two ridge points at the ends of its fragment. If we want to be strict in our requirement for strength improvement between p_2 and q , we might require that it gives us at least a certain percentage of improvement, e.g. 5%. Empirically, though, adding such a requirement was found a bit too restrictive. We might also want to require the connected component of q to be stronger than the connected component of p_2 .

In order to avoid complications, we might require that the two ridge points which are connected to q both be rank-2 points, so there is no question of which edge to choose right after jumping. We can also require this only from q_2 . Another limitation that we impose is to require a minimal number of steps that should be made in our current scale before we consider jumping from it to a different scale. If we haven't made this number of steps yet, we don't look for optional scale-jump destinations and continue stepping only in our current scale. If, however, this limitation causes us to get stuck in a dead end (i.e. a ridge point of rank 1 that we stepped into), we might not want to impose this limitation.

One last and important thing is that we do not jump if our current direction of progress is marked as occupied at the point q_2 . See Section 5.7 for more details.

5.3.3 More considerations of whether to jump or not, and how to obtain the data they require

What if we could ask a few more questions before we actually decide to jump from p_2 to q ? What if we could know in advance how far we would be able to walk after having jumped from p_2 to q before reaching a dead end, or how jumping can contribute to the added strength of the output curve? Surely we wouldn't want to jump to a point from which we can only take one or two more steps forward, or that guarantees us a significantly lower contribution to the strength, compared to stepping within our current scale.

At each rank-2 points, there are exactly two adjacent points we can step to. In the tracking process, it will be useful to know what these directions promise us: What length can we accumulate if we continue to step forward until we reach a dead end? How many points will we meet along the way? What contribution to the strength can we accumulate in that direction? We can answer these questions already in this stage, by actually performing this walk in a preliminary tracking process that does not jump between scales and does not generate output, but rather collects all that information and updates it to the rank-2 points. Then we can perform the actual tracking, based on that information.

As we step forward in the preliminary tracking, with each new step, we update the new point we reach with all the information we accumulated so far. Whenever we reach intersections, we simply choose the direction which best correlates with our previous step direction. We can also exponentially-smooth the previous step direction with its predecessors in order to have a more accurate measure.

Another piece of information that we might want to keep at each rank-2 point is its actual local direction. We can obtain that direction directly from the direction of v_{min} , but that is

a local measure which might be sensitive to noise, and thus cannot be 100% reliable. Another way is to measure the actual local direction, by simply taking a few steps forward and backward and using the difference vector as the direction vector. A more time-efficient way to do so is, as we walk along the path, we keep a “current direction” variable, say v_d , and with each new step v_s , update it via the formula:

$$v_d \leftarrow \alpha v_d + (1 - \alpha) v_s$$

with $0 < \alpha < 1$.

Using Heron’s formula, we can also approximate the curvature at each rank-2 point. For this we need to choose points which are at a sufficient number of steps away from the point where we measure the curvature. However, in this work we couldn’t find any meaningful use for the curvature for deciding whether to jump to a point or not. Still, intuitively, it might seem safer to jump from one point to another when both these points admit low curvatures.

5.4 Reaching a dead end

What do we do when we cannot step forward anymore? There are some cases when the ridge curves are “cut” in the middle, due to noise or small artifacts on the object that generated the ridge. We can try to compensate for that by looking ahead in a certain range for the beginning of the next part of the curve. If we find a ridge point that is close enough to the end point, and is located at an angle from that point which correlates well with the current ridge direction, we can simply add this point as the next point on the curve and continue tracking from there. If we want to create a more consistent output curve, we can bridge the gap between these points using a few “phantom points”, in order to prevent large gaps (more than 1 pixel long) between adjacent points on an output ridge curve.

5.5 After having finished constructing a ridge curve

Through out the ridge tracking phase, we maintain a list of all the output ridge curves. Whenever we finish constructing a new ridge curve, we add it to the list. Before adding it, we calculate some measurements of the ridge curve, such as its length and its strength. The strength is calculated by integrating the point strength along that curve, in the same manner we did for connected components on Section 3.4. After adding the curve to the list of output curves, we mark all its edges as occupied. Therefore, they cannot be used or stepped through in further ridge constructions.

Once we’ve reached the last rank-2 point in the point array, it means that we ran out of starting points and thus we’ve finished the ridge tracking phase. The last thing left is to sort the ridge curve list according to the ridge strength and return the desired number of best ridges.

5.6 How to determine the ridge width

At each ridge point, by knowing the scale and ridge angle, we can determine the width of the ridge around that point and do so quite accurately. Note that we need the angle for that purpose since this algorithm is highly non rotation-invariant. We have managed to work out the dependence of the width in the ridge angle empirically. Suppose the ridge angle at point p is θ with $\theta \in [0, \pi]$ (it is important to actually keep θ within that range), and the scale at p is

σ . Then the distance from p to the exterior of the ridge (i.e. the local ridge width divided by 2) is given by $\sigma \cdot \gamma$ with:

$$\gamma = 2^{1/4} \sqrt{\cos \left(\min \left(\left| \theta - \frac{1}{4}\pi \right|, \left| \theta - \frac{3}{4}\pi \right| \right) \right)}$$

5.7 Maintaining a ridge domain record

Usually, long objects on the image generate similar ridge curves in multiple adjacent scales. Although we don't use the same ridge fragments in more than one ridge, we still might use ridge fragments from similar locations in adjacent scales and create multiple ridge curves that represent the same object. Moreover, this multiplicity of copies of similar ridge curves might prevent us from returning other ridges, which have a slightly weaker strength but are still good and should be contained in the output. In order to avoid those repetitions, we maintain a data structure we call *ridge domain record*. The ridge domain record aims in preventing the ridge tracking process to step into regions which are already occupied by other ridges that were accepted to the output, if in our current tracking we are close enough to the occupied ridge in three senses:

1. Location on the x, y plane
2. Scale
3. Angle

If we are about to pass in the domain occupied by an already existing ridge curve, in one of the scales that are adjacent to it, and in a similar direction, we stop the tracking process.

In order to keep track of which directions are occupied and which are not, we divide the interval $[0, \pi)$ into k bins, with bin no. i ($i \in \{0, \dots, k-1\}$) corresponding to the range of directions $[\frac{i}{k}\pi, \frac{i+1}{k}\pi)$. The data is kept as follows: For each scale space voxel we maintain a bit array of k bits. Whenever we finish constructing a ridge curve and accept it to the output, we run along all the points that this ridge occupies. These are not merely the ridge points that the curve passes through, but all the ridge points in our database that are located in that ridge's domain, taking into account its width (see Section 5.6 for how to calculate the ridge width). We do this by scanning along a short line that is perpendicular to the ridge curve, with its length equal to the ridge width at the current point. Moreover, we might also want to take a safety range of scales, and occupy all the points that are 1,2 or 3 scales up and down.

In each of the points we encounter in the mentioned process, we update its appropriate bin according to the current ridge direction, and set its flag to 1, meaning "occupied".

That process can be sketched in the following pseudo-code. Let us make the following notations: Let $\gamma[i]$, $i \in \{1, \dots, n\}$ be the array of points denoting a ridge curve we just accepted to the output, with the unit-normals $N[i]$ and the ridge angles $\theta[i]$. Denote by $\sigma[i]$ the ridge scales and $w[i]$ the ridge half-width at point i . Let $d \in \{0, \dots, k\}$ denote the range of angle bins to occupy and $r \in \{0, 1, 2, \dots\}$ denote the safety range of scales to occupy - both of which are constant parameters of the algorithm. We do the following:

```

For  $i \in \{1, \dots, n\}$ 
Let  $\bar{\theta} = \theta[i]$  with its value taken in  $[0, \pi)$ .

    Let  $b = \left\lfloor \frac{\bar{\theta}}{\pi} k \right\rfloor$ .
    For  $j \in \{-\lfloor w[i] \rfloor, \dots, \lfloor w[i] \rfloor\}$ 
        For  $s \in \{\sigma[i] - r, \dots, \sigma[i] + r\}$ 

            Let point  $p$  be the point with integer coordinates  $p =$ 
             $\text{round}(\gamma[i] + jN[i])$ .
            If there is a ridge point in point  $p$ , scale  $s$ ,
                set the following flags at that point to 1:
                 $\{b - d, \dots, b + d\} \pmod k$ 

```

References

- [Lind-96] Tony Lindeberg - Edge detection and ridge detection with automatic scale selection, 1996.