Published in Image Processing On Line on 2019–08–10.
Submitted on 2019–06–28, accepted on 2019–07–29.
ISSN 2105–1232 © 2019 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2019.269

Extraction of the Level Lines of a Bilinear Image

Pascal Monasse

Université Paris-Est, LIGM (UMR CNRS 8049), ENPC, F-77455 Marne-la-Vallée, France
(pascal.monasse@enpc.fr)

Abstract

We detail precisely an algorithm for the extraction of the level lines of a bilinear image, which is a continuous function interpolating bilinearly a discrete image. If we discard the levels of the discrete image, where topological difficulties arise, a level line is a concatenation of branches of hyperbolas. The algorithm tracks these branches and provides a sampling of the level lines in the form of closed polygons. If the level line contains a saddle point, the hyperbola degenerates to orthogonal segments, where an arbitrary but consistent choice is adopted for the tracking at the bifurcation. In any case, the extracted polygons are disjoint and enclose a bounded region. This allows to order the level lines in an enclosure tree hierarchy, which may be used for a variety of filters. Recovering this tree is a simple post-processing of the extraction algorithm.

Source Code

The ANSI C++ 03 implementation of the code that we provide is the one which has been peer reviewed and accepted by IPOL. The source code, the code documentation, and the online demo are accessible at the IPOL web page of the article[1]. Compilation and usage instructions are included in the `README.txt` file of the archive. The code is sensibly the same as the one used for computing the mean curvature map of an image [5]. The demo just outputs the level lines and their hierarchy.

Keywords: level lines; bilinear interpolation; tree of shapes

1 Introduction

Based on the observation that visual information is largely independent on the contrast of the images, it is a natural requirement that computer vision should not rely on contrast. A simple contrast change being an increasing function remapping input intensities, the elementary components of the image can be the isolevel sets, that is, the sets of points of the domain sharing the same intensity. In a digital image, these sets are composed of pixels, and most isolevel sets are composed of sparse pixels, revealing no coherent shape. A simple change is to consider instead level sets, sets of pixels below (resp. above) some intensity λ, which may be called lower level sets (resp. upper level sets). These

[1] https://doi.org/10.5201/ipol.2019.269

are the foundation of mathematical morphology, in which any tranform that maps a set to a set and preserves inclusion order of sets is directly extended to an image by applying the transform on the level sets of the original image and building the output image by stacking the transformed level sets [6]. By construction, any such transform commutes naturally with a contrast change, it is said to be equivariant with contrast.

A variant consists of considering the boundaries of the level sets, called level lines. They are the elevation lines of a topographic map. For a continuous image, the level lines are parts of isolevel sets, and most of them are curves, with an interior and an exterior, namely Jordan curves. Working with level lines is convenient, as they can be approximated by polygons. Another advantage of considering continuous images is that the level lines do not follow strictly the pixel grid, they pass *between* pixels. This is used for example in the "image curvature microscope" of IPOL [5], where an additional smoothing of the level lines is performed to be even less sensitive to pixelization. Finally, the level lines are disjoint, which makes simpler algorithms and facilitates visualization.

The best way to convert a digital image to a continuous function is by applying a convolution with a continuous function (a kernel) to the Dirac comb at pixel locations. The kernel compatible with Shannon sampling theory is the cardinal sine, but simpler kernels can be designed, in particular splines [2]. The easiest to deal with is the one of order 1, called bilinear. Bilinear interpolation is fast and quite easy to implement, as the value at a point depends only on its relative position with respect to the four nearest pixels and their data values. It is the natural, separable 2D extension of linear interpolation. An interesting property of the bilinear interpolation is that it does not create extrema between pixels, which shields us from complex singularities in the level lines.

As we will see, the so-called bilinear level lines are Jordan curves built from a concatenation of parts of hyperbola branches and straight segments. Our algorithm takes as input a set of intensity levels and yields a set of polygons that approximate the level lines. Moreover, the level lines are in a tree structure based on the enclosure relationship: level lines being disjoint, if two level lines have intersecting interiors, one is included in the other and one level line is enclosing the other; the former is an ancestor of the latter in the tree. This allows reconstructing the image from the level lines.

A higher level description of the extraction algorithm presented here was exposed in a monography [3] and the implementation was used for the image curvature microscope. The accompanying source code is a polished and clearer version of the one used there.

2 The Bilinear Level Lines

2.1 Anatomy of a Bilinear Level Line

The continuous image is noted u. Its data points are at coordinates $p \in \{0,\dots,w-1\}\times\{0,\dots,h-1\}$ and the values $u[p]$ are called the initial levels[2]. The algorithm assumes that we extract level lines at levels that are *not* initial levels: indeed, extrema of the image are at initial levels and at such levels, the level sets $[u=\lambda]$ may be reduced to a point, a segment, a full square, or a combination thereof. In other words, these are not lines. By contrast, almost all other level sets are Jordan curves as we will see, the exception being at the other critical points, the saddle points. At such points, the level line is not simple, it is self-intersecting: we get an X-junction. Tracking such lines is ambiguous, but our algorithm takes a principled approach to decide which direction to choose at such crossings, so it does not fail in such occurrences and we do not need to exclude the levels of saddle points, even though it can be ensured since they are finite in numbers. In any case, avoiding some finite number of levels, initial levels and possibly saddle levels, is not limiting, as the image is continuous and is precisely represented by a dense enough set of level lines.

[2]When taking the value of u at data points, we will use square brackets $u[.]$ instead of $u(.)$ for interpolated points.

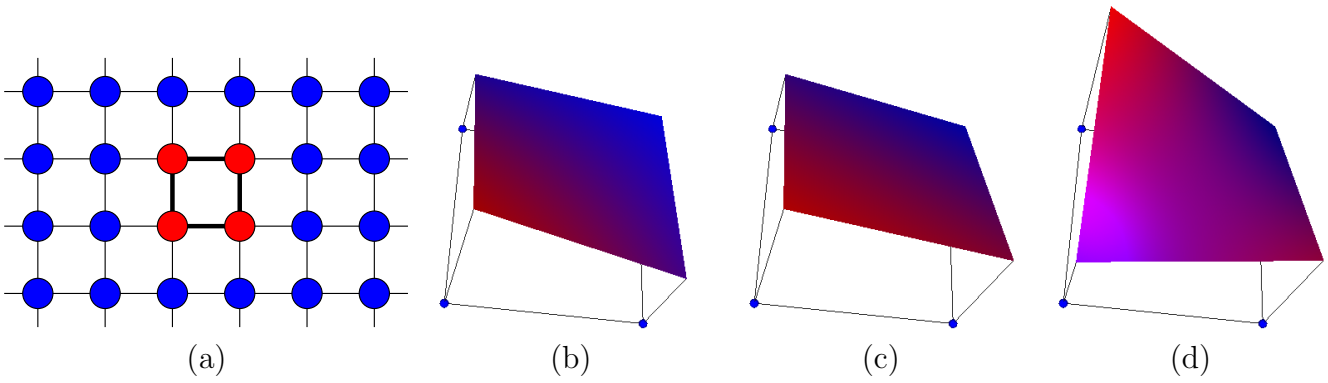


Figure 1: (a) Data points (blue and red points), a dual pixel (red points) and its four edgels (thick lines). Graph of u in a dual pixel: (b) regular, (c) planar ($d = 0$ in (3)), and (d) with saddle point ($(x_s, y_s) \in [0,1]^2$ in (6)).

We assume that the image is constant at its border:

$$|\{u[p] \,|\, p \in (\{0,\dots,w-1\} \times \{0,h-1\}) \cup (\{0,w-1\} \times \{0,\dots,h-1\})\}| = 1. \tag{1}$$

This can be achieved by explicitly setting some fixed value at the border or adding a band of 1 pixel with a constant value to each side. This ensures that all level lines are closed, so that tracking a level line can start anywhere since there is no endpoint. An auxiliary benefit is that we need not check that we stay inside the image while tracking.

We call edgel a pair of 4-adjacent data points (p, q). We call dual pixel a square of four distinct data points (p_0, p_1, p_2, p_3) such that (p_i, p_{i+1}) is an edgel for $i = 0, 1, 2$, so as (p_3, p_0), see Figure 1 (a). The bilinear interpolation being translation invariant, we just need to analyze the level lines of u inside the dual pixel $p_0 = (0,0)$, $p_1 = (0,1)$, $p_2 = (1,1)$ and $p_3 = (1,0)$. Since u is obtained by bilinear interpolation of the data points, we can write

$$\forall p = (x,y) \in [0,1]^2, u(p) = u[p_0]\,(1-x)(1-y) + u[p_1]\,(1-x)y + u[p_2]\,xy + u[p_3]\,x(1-y). \tag{2}$$

Distributing the products and refactoring, we can rewrite

$$u(p) = a + bx + cy + dxy, \tag{3}$$

with

$$a = u[p_0], \quad b = u[p_3] - u[p_0], \quad c = u[p_1] - u[p_0], \quad d = u[p_0] + u[p_2] - u[p_1] - u[p_3]. \tag{4}$$

If $d = 0$, u is affine inside the dual pixel (see Figure 1 (c)) and the portion of level line is a straight segment. The exception is when $b = c = 0$, in which case we have all points at the same level a, which is an initial level, and so excluded from level line extraction. In most dual pixels, we would have $d \neq 0$ (Figure 1 (b)) and we can rewrite

$$u(p) = d(x - x_s)(y - y_s) + \lambda_s, \tag{5}$$

with

$$x_s = -\frac{c}{d}, \qquad y_s = -\frac{b}{d}, \qquad \lambda_s = \frac{ad - bc}{d} = \frac{u[p_0]\,u[p_2] - u[p_1]\,u[p_3]}{u[p_0] + u[p_2] - u[p_1] - u[p_3]}. \tag{6}$$

This formulation reveals that the level line is locally a part of (rectangular) hyperbola. When the point $s = (x_s, y_s) \in [0,1]^2$, it is a saddle point of the bilinear image and its level is $u(s) = \lambda_s$, see Figure 1 (d). In that case, we may have two branches of the hyperbola of equation $u = \lambda$ intersecting the dual pixel, so that it is possible for a level line to go twice through a given dual pixel. A particular situation is when $\lambda = \lambda_s$, in which case the level line is locally the union of two segments, one horizontal and one vertical (degenerate hyperbola). Notice that any arbitrary choice of turn at a saddle level may yield one or two level lines depending on the surrounding, see Figure 2.

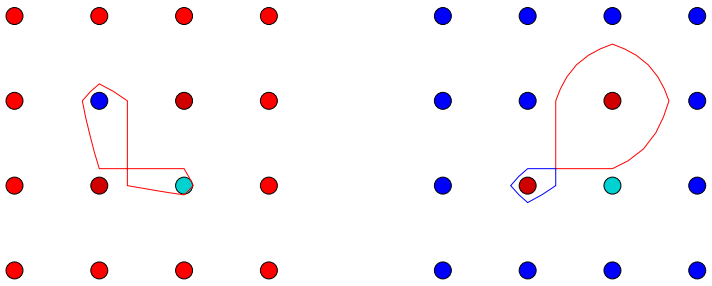


Figure 2: The same rule for the choice of turn (left or right) at a saddle level may extract a level line as a single curve with one contact point, or two curves meeting at a point. The values at the vertices of the central dual pixel are identical, but the surrounding data points are all of higher or lower value. The curves are then oriented differently, clockwise in the left figure, counter clockwise in the right figure.

2.2 Enclosure Tree Structure

The level lines being Jordan curves (except at saddle levels), they have an exterior and an interior, the connected components of its complement that meet the image frame or not. If we consider another level line, since it is disjoint from the first one, it may lie in its interior. In that case, we may consider the former as a descendant of the latter. In this way, we get a tree structure [1], or actually a forest, since there may be more than a single root. From this tree, an image can be reconstructed: each pixel gets the intensity of the deepest level line in the tree that contains it in its interior. The level lines of the reconstructed image are the ones in the tree it was built from. This is the superposition principle of mathematical morphology. This process is very useful to implement operators from mathematical morphology [4], much more precisely than any finite differences scheme [7].

3 Level Line Extraction

3.1 Level Line Tracking

Each level line L being closed, it crosses at least one horizontal edgel (p_0, q_0) with $u[p_0] < \lambda < u[q_0]$, λ being its level. Since the image level is either constant or strictly monotonous (affine) along an edgel, any edgel can be crossed at most once by L. The tracking proceeds by moving from edgel to edgel. At a given level λ, an array v is used to mark the visited horizontal edgels crossed by a level line. This ensures that we detect the loop closure while tracking, and that we do not track more than once a same level line. The procedure is summarized in Algorithm 1. The procedure `follow` jumps from edgel to edgel along the level line: a mobile dual pixel D of upper left corner p_0 is stored and a keypoint p, of the level line and on an edgel of D, is also moving. During the progression, D moves to one of its 4-neighbors, the one sharing the dual pixel where p lies. In the procedure `move`, the second step of `follow`, the dual pixel D and the keypoint p are displaced. The third step of `follow`, the procedure `sample`, returns an array of points sampling the branch of hyperbola between these edgels. Both steps make use of the parameters H of the hyperbola, computed in the first step of `follow`. Notice that it is sufficient to perform operations on v only with horizontal edgels (p_0, q_0) such that $q_0 = p_0 + (1, 0)$, and therefore we can just index v by the left endpoint p_0 of the edgel.

3.2 Hyperbola Parameters

Procedures `move` and `sample` use the parameters of the (possibly degenerate) hyperbola followed by the level line inside the dual pixel. These are computed at line 10 of Algorithm 1. They consist of:

1. the saddle point s, i.e. the center of the hyperbola;

Algorithm 1: Extracting all level lines.

Input: Image u, set of levels $\mathcal{L}$
Output: Level lines $\{L\}$

foreach $\lambda \in \mathcal{L}$ **do**
  $v[.] \leftarrow$ `false` `// No edgel visited at level` λ `yet`
  foreach *horizontal edgel* (p_0, q_0) **do** `// Only for` $q_0 = p_0 + (1, 0)$
    if $u[p_0] < \lambda < u[q_0]$ **and not** $v[(p_0, q_0)]$ **then**
      $L \leftarrow \emptyset$
      repeat
        $v[(p_0, q_0)] \leftarrow$ `true`
        $D \leftarrow$ Dual pixel of upper right corner p_0
        $p \leftarrow p + (\frac{\lambda - u[p_0]}{u[q_0] - u[p_0]}, 0)$ `// Keypoint of` L
        $h \leftarrow$ Hyperbola parameters in D `// Step 1 of follow`
        $(D, p') \leftarrow$ `move`(D, λ, λ_s) `// Step 2 of follow,` λ_s `is part of` h
        $L \leftarrow L \cup$ `sample`(p, p', h) `// Step 3 of follow`
        $p \leftarrow p'$
      until $v[(p_0, q_0)]$
      Append L in $\{L\}$

2. its level λ_s in the form of a fraction, as in (6);

3. parameter $\delta = (\lambda - \lambda_s)/d$;

4. the vertex v of the hyperbola, that is the point of maximal curvature, whose coordinates (x_v, y_v) satisfy
$$(x_v - x_s)(y_v - y_s) = \delta, \tag{7}$$
so that $|x_v - x_s| = |y_v - y_s| = \sqrt{|\delta|}$, and so $\sqrt{2|\delta|}$ is the semi-major axis, distance from the vertex of the hyperbola to its center.

3.3 Finding the Exit in a Dual Pixel

The `move` procedure (Algorithm 2) is in charge of finding the exit edgel inside the dual pixel. It tracks the level line always leaving upper level data points to the left and lower level data points to the right. The dual pixel being $D = (p_0, p_1, p_2, p_3)$, with entry point p on the edgel (p_0, p_3), three exit edgels are possible: (p_0, p_1), (p_1, p_2) and (p_2, p_3). We call the first case a right turn and the last one a left turn. They are possible if $\lambda < u[p_1]$, $\min(u[p_1], u[p_2]) < \lambda < \max(u[p_1], u[p_2])$ and $u[p_2] < \lambda$ respectively. The cases are not exclusive, in particular with $u[p_2] < \lambda < u[p_1]$ and the three possibilities altogether. Notice this case implies $\max(u[p_0], u[p_2]) < \min(u[p_1], u[p_3])$. We are on a branch of a hyperbola, and the choice of which turn depends on the sign of $\lambda_s - \lambda$, with λ_s the saddle level of (6). If $\lambda < \lambda_s$ (resp., $\lambda_s < \lambda$) we have to make a right (resp., left) turn. If by chance $\lambda = \lambda_s$, the level set inside the dual pixel is the union of the two segments $x = x_s$ and $y = y_s$, and there is no natural choice, as the level line is not a simple curve. We choose arbitrarily a left turn in this case.

Notice that in Algorithm 2 p_0 needs not be the upper-left corner of the dual pixel D. Since in the implementation a dual pixel is identified by its upper-left corner, a suitable rotation of the indices $(0, 1, 2, 3)$ must be performed for correspondence.

Algorithm 2: Procedure `move` to find the exit edgel inside a dual pixel

Input: Level λ, dual pixel $D = (p_0, p_1, p_2, p_3)$ with entry in edgel $(p_0, p_3) : u[p_0] < \lambda < u[p_3]$
Output: Adjacent dual pixel D' with exit edgel $(p_i, p_{i+1}) : u[p_i] < \lambda < u[p_{i+1}]$, $i \in \{0, 1, 2\}$, subpixel exit point p' between p_i and p_{i+1}

`right`$\leftarrow \lambda < u[p_1]$? `left`$\leftarrow u[p_2] < \lambda$?
if *left* **and** *right* **then** // disambiguate according to saddle level
  `right`$\leftarrow \lambda < \lambda_s$? `left`$\leftarrow$ **not** `right` // saddle level λ_s computed from (6)
if *right* **then**
  $(p', q') \leftarrow (p_0, p_1)$ // $i = 0$
else if *left* **then**
  $(p', q') \leftarrow (p_2, p_3)$ // $i = 2$
else
  $(p', q') \leftarrow (p_1, p_2)$ // $i = 1$
$D' \leftarrow$ dual pixel adjacent to D sharing edgel (p', q')
$p' \leftarrow p' + \frac{\lambda - u[p']}{u[q'] - u[p']}(q' - p')$

3.4 Sampling Inside a Dual Pixel

When $d = 0$ in (4), the level line is a straight segment inside the dual pixel. In this case, we do not need to store any intermediate point, just entry and exit points at edgels represent perfectly the level line. We thus assume in the rest of this section that $d \neq 0$. In this case, the level line can still be a segment, but only if

$$\lambda = \lambda_s \text{ and } \begin{cases} x_s \in [0,1], y_s \notin [0,1] & \text{vertical segment } x = x_s, \text{ or} \\ x_s \notin [0,1], y_s \in [0,1] & \text{horizontal segment } y = y_s. \end{cases} \tag{8}$$

The case $\lambda = \lambda_s$ and $s = (x_s, y_s) \in [0,1]^2$ is special: whereas the true level set is a cross $(\{x_s\} \times [0,1]) \cup ([0,1] \times \{y_s\})$, we treat it as the half cross, two segments: one from the entry point to s and one from s to the exit point. Therefore, three points sample perfectly the level line.

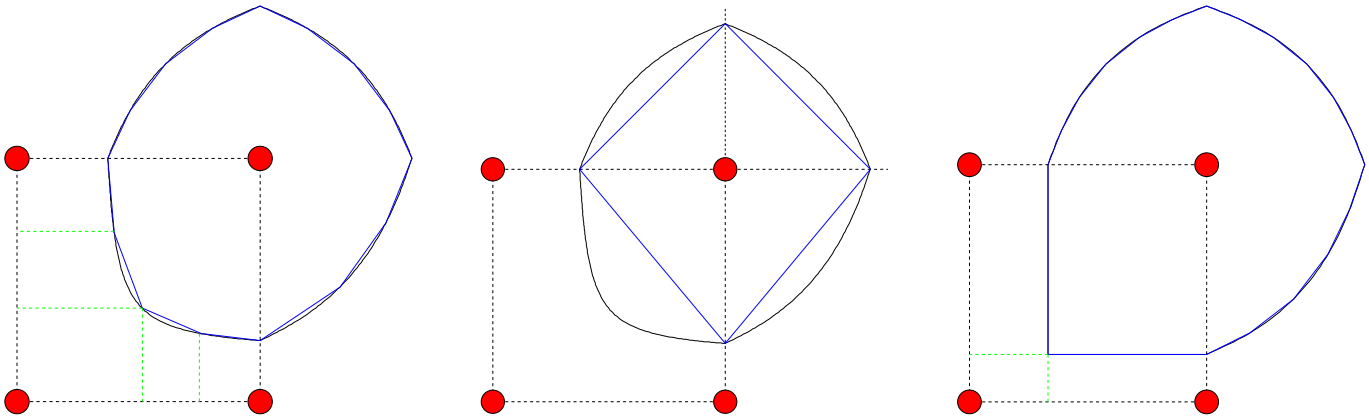
The generic case is $\lambda \neq \lambda_s$, and we have then a part of a non-degenerate hyperbola. We can choose to make the implicit equation (5) explicit in two ways:

$$x = x_s + \frac{\delta}{y - y_s} \qquad \text{or} \qquad y = y_s + \frac{\delta}{x - x_s}. \tag{9}$$

with $\delta = (\lambda - \lambda_s)/d$. Obviously, both are valid as the denominators do not vanish: $x = x_s$ or $y = y_s$ would mean $\lambda = \lambda_s$. The best approximation of the hyperbola is achieved by using the first form and sampling regularly in y if $|\partial x/\partial y| \leq 1$ and using the second form and sampling regularly in x if $|\partial y/\partial x| \leq 1$. These two quantities being inverse, there is no third alternative. The change from one regime to the other happens when $|x - x_s| = |y - y_s| = \sqrt{|\delta|}$, in which case (x, y) is a *vertex* of the hyperbola. It happens to coincide with the point of maximal curvature of the hyperbola, therefore, for optimal approximation, this point should be sampled from the hyperbola if it is located inside the dual pixel, see Figure 3. This is what we call a keypoint of the level line. The other keypoints are the points of intersection with edgels. That is, each entry point into a dual pixel is recorded.

The sampling (see Algorithm 3) is then performed by a user-defined approximate number `ppp` of sampling points per pixel unit. For example, the sampled points between keypoints (x, y) and (x', y') would be

$$x_i = x_s + \frac{\delta}{y_i - y_s}, \qquad y_i = y + i \cdot \frac{y' - y}{n}, \qquad 1 \leq i < n = \lceil |y' - y| \cdot \texttt{ppp} \rceil. \tag{10}$$



(a) ppp = 5 (b) ppp = 0 (c) $\lambda = \lambda_s$

Figure 3: Sampling of hyperbola inside a dual pixel (polygonal output in blue). (a) A regular sampling: the maximum curvature point is present, the left part is sampled as $x = x(y)$ and the right part as $y = y(x)$. (b) When ppp $= 0$, only intersections with edgels are sampled. (c) At the saddle level, only the center of the hyperbola is sampled, whatever ppp.

The bounds on i make sure not to overshoot, that is, go beyond (x', y'): 0 and n are excluded in order to not repeat the keypoints. Notice that if the integer ppp < 2, no intermediate point is sampled. If ppp $= 1$, only keypoints are kept. As an extension, the program accepts ppp $= 0$, meaning that only intersections with edgels are recorded, not the other keypoints, i.e., the vertices of hyperbolas.

Algorithm 3: Sampling of hyperbola between two keypoints

Input: s, δ: center and parameter of hyperbola, p, q: consecutive keypoints, ppp points per pixel unit

Output: A series of sampled points on the hyperbola S

$S \leftarrow \emptyset$
if $|x_q - x_p| > |y_q - y_p|$ **then**
  $n \leftarrow \lceil\, |x_q - x_p| \cdot \mathtt{ppp} \,\rceil$
  $dx \leftarrow (x_q - x_p)/n$
  for $i = 1 \ldots n-1$ **do**
    $S \leftarrow S \cup (x_i = x_p + i \cdot dx, y_s + \frac{\delta}{x_i - x_s})$
else
  $n \leftarrow \lceil\, |y_q - y_p| \cdot \mathtt{ppp} \,\rceil$
  $dy \leftarrow (y_q - y_p)/n$
  for $i = 1 \ldots n-1$ **do**
    $S \leftarrow S \cup (x_s + \frac{\delta}{y_i - y_s}, y_i = y_p + i \cdot dy)$

4 Tree Structure Computation

We take advantage of the fact that level lines are disjoint and closed. To recover the enclosure relationship, consider all horizontal edgels at a given ordinate, say $y \in \{0, \ldots, h-1\}$. That is, we consider the $w-1$ horizontal edgels (x, y)—$(x+1, y)$ with $x = 0 \ldots w-1$. A level line L intersects any such edgel at most once, therefore there are a finite number of intersection points for L. Since L is closed, this number must be even. When enumerating the edgels in order, we can count the number of edgels intersected by L. At a point $(x, y) \in \mathbb{R} \times \{0, \ldots, h-1\}$, if the number of intersections up to x is even, it means that (x, y) is outside L, while when it is odd (x, y) is inside L.

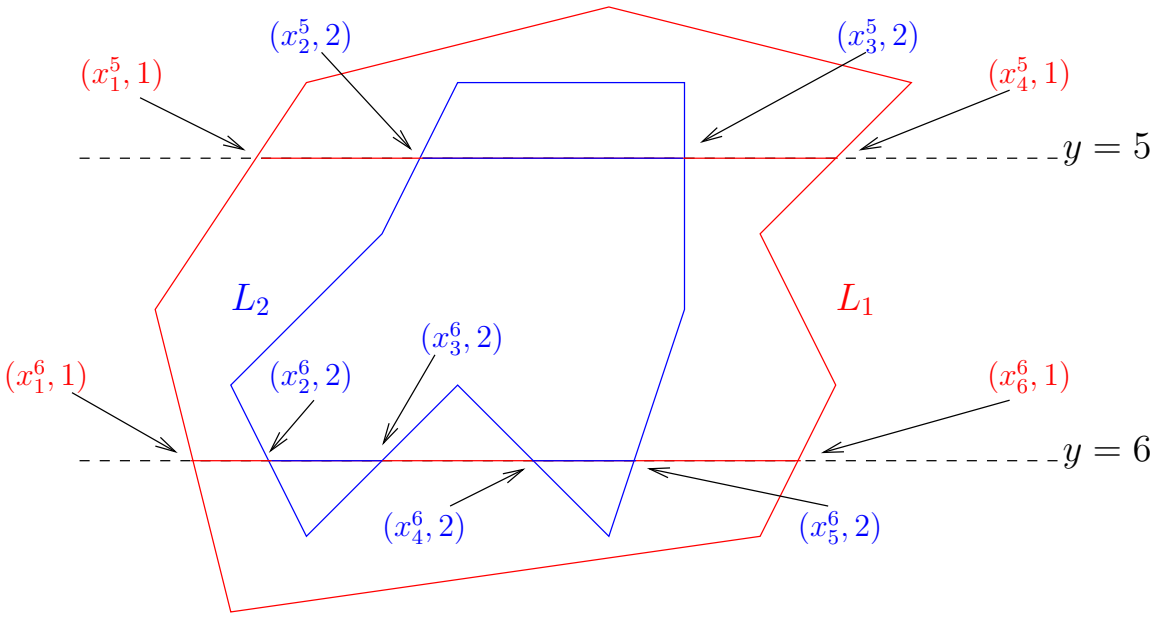


Figure 4: Two polygons, L_1 and L_2, and their enclosure relationship. At scanlines $y = 5$ and $y = 6$, all the intercepts with lines have their abscissa recorded together with the originating level line index, 1 or 2. Scanning line $y = 5$ in Algorithm 4 indicates that L_1 encloses L_2. Scanning line $y = 6$ only confirms the known information. Segments $[x_1^5, x_4^5] \times \{5\}$ and $[x_1^6, x_6^6] \times \{6\}$ are enclosed by L_1, while segments $[x_2^5, x_3^5] \times \{5\}$, $[x_2^6, x_3^6] \times \{6\}$ and $[x_4^6, x_5^6] \times \{6\}$ are enclosed by L_2.

The procedure, Algorithm 4, is akin to a parenthesis balancing and is illustrated in Figure 4. We assume that intersections with horizontal edgels are recorded at line 7 of Algorithm 1. Given a set of level lines L_i, we can record all intersections with the line at ordinate y and store (x_j^y, i_j^y). In this notation, each intercept point at ordinate y, indexed by j, does not only store its abscissa x_j^y but also the index i_j^y of the level line it belongs to. Then we can sort the records at a given y by increasing value of x_j^y, and denote them as (x_j^y, i_j^y) with $(x_j^y)_j$ an increasing sequence. We can then enumerate the records and use a stack data structure S holding indices i_j^y: when i_j^y is not the top element k of S, $i_j^y \neq k$, then we push i_j^y on top (entering the interior of $L_{i_j^y}$) and record that $L_{i_j^y}$ has parent L_k; whereas if $i_j^y = k$ we pop k from S (getting outside L_k).

Algorithm 4: Building parent/child enclosures of level lines

Input: Set of level lines $(L_i)_{i=1\ldots n}$; their intersections (x_j^y, i_j^y) with edgels at line y, $j = 1 \ldots n_y$, ordered so that $x_j^y < x_k^y$ when $j < k$.
Output: Parent of each L_i (an L_k)

```
1  for y = 0 ... h − 1 do
2      Stack S = ∅
3      for j = 1 ... n_y do
4          k ← top(S), or invalid index 0 if S is empty
5          if i_j^y = k then                                  // Exit from L_k
6              pop S
7          else                                               // Entering L_{i_j^y}
8              if k > 0 then
9                  parent(L_{i_j^y}) ← L_k
10             S ← i_j^y                                      // Push on top of S
```

5 Image Reconstruction

The reconstruction of an image from its tree of bilinear level lines is interesting when the level lines have changed. For example, they have been smoothed by mean curvature motion or by affine curvature motion. Otherwise, it is just a complicated way to quantize the image. The principle is fairly similar to that of Algorithm 4: scan the lines of edgels and count the number of intersections with a level line L to see if we are inside or outside at output data points. The difficulty is that we cannot assume anymore that a level line crosses edgels at discrete points: there could be a part of the curve that is a horizontal segment, providing an infinite number of intersection points. This has to be taken care of.

The high level algorithm is explained in Algorithm 5. It consists essentially of a preorder traversal of the tree (i.e., the parent is enumerated before its children), where each level line fills the output image pixels in its interior by its gray level. The image is thus filled layer by layer, a pixel getting its value from all level lines enclosing it, newer ones painting over the previous ones. The main difficulty comes from the determination of interior/exterior of a level line.

Algorithm 5: Reconstruction of image from its level line tree (`fillTree`)

Input: Tree T whose nodes are level lines sorted by enclosure, image u to be reconstructed
Output: Pixels of u are filled with the level of the deepest line including it

```
1 for L ← root(T) do
2 |   fillCurve(L, u);
3 |   foreach C child of L do
4 |   |   fillTree(C, u);                    // Recursive call with tree rooted at C
```

The procedure `fillCurve` fills the output data points in the enclosure of the level line (in the form of a polygon) with its gray level. Reformulated, the problem is to find the pixels (with integer coordinates) in the interior of a polygon whose vertices have floating point coordinates. The first step records for each scan line of the output image the endpoints of the intervals of interior points (in floating point coordinates), which are then sorted; the second step determines for each pixel of a scan line whether it is inside or outside of the polygon, which is given by the parity of the number of endpoints at its left. The main difficulty of the first step is that the polygon may have sides along some scan lines. The algorithm enumerates the vertices of the polygon one by one, and compares each (q) with the next one (p). If the point is not on a scan line, we just need to store the point of intersections of scan lines with the segment $\alpha p + (1-\alpha)q$, $\alpha \in [0, 1)$ ($\alpha = 1$ is excluded since p may be recorded next). When the ordinate of q is an integer, i.e., it falls on a scan line, the previous point must also be taken into account following the rules of Figure 5, which determine whether q should be recorded or not.

Rules (b) and (d) indicate that q must be recorded, because it represents an endpoint of a segment on a scan line. On the contrary, (c) and (e) do not indicate an end of the scan line. The special case (f) is normally not applicable to regular polygons but can still be applied if the polygon was distorted. Rule (g) records twice the peak q. Rule (a) is applied to record intermediate points on scan lines. It is cumulative with rules where point p is not at the same ordinate. Notice that these rules assume that the region enclosed by the polygon is at its left following its orientation, which may not be the case, but they are consistent. For example, if the interior is on the right, an entry point in a scan line according to (c)-top and an exit from this scan line according to (e)-top would both be ignored, therefore missing the segment on the scan line. Nevertheless, it is acceptable as it can be argued that the fact that the polygon is tangent to the scan line is numerically unstable. Figure 6 shows an example of application of the rules.

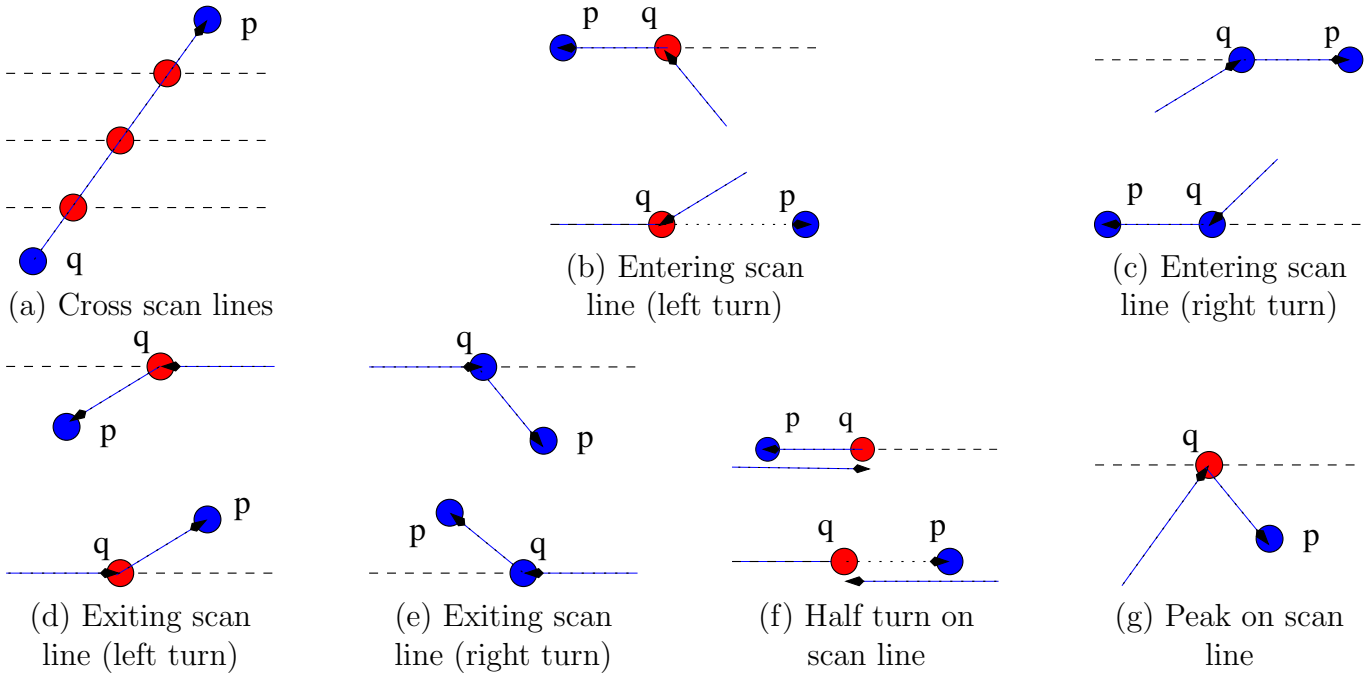


Figure 5: Rules for recording intersections of polygon side $\alpha p+(1-\alpha)q$ with scan lines. (a) Regular case with p not on scan line. (b), (c) The polygon is coming from outside the scan line containing p and q: in case of left turn (b), q is recorded as endpoint; in case of right turn (c), q is not recorded. (d), (e) The polygon is coming from the scan line containing q, which does not contain p: in case of left turn (d), q is recorded as endpoint; in case of right turn (e), q is not recorded. (f) In case of 180-degree turn on a scan line, q is recorded. (g) q is on a scan line, but next and previous points are not on the same scan line: q is recorded *twice*. Rule (a) is also applied after (d), (e) and (g).

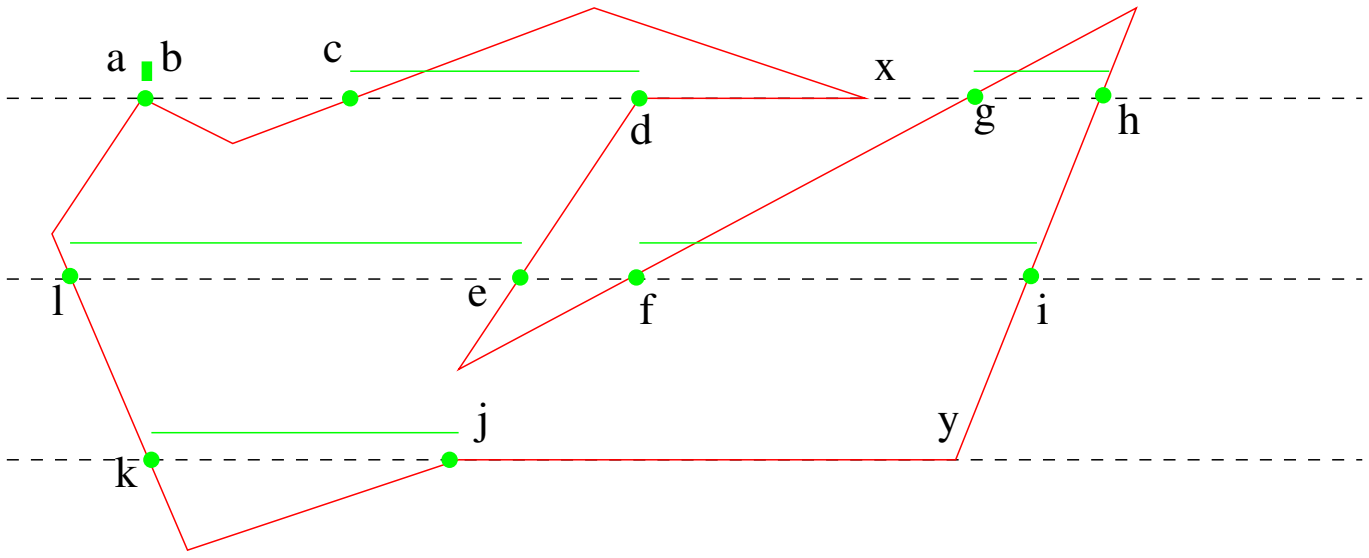


Figure 6: Segments of scanlines considered as interior of the red polygon according to rules of Figure 5. Notice that $a=b$ is a double point according to rule (g). Points x and y of the polygon are not recorded according to rule (c). This is not inconsistent, since segments dx and jy are "unstable", sensitive to precise positions of the vertices. However, if the orientation of the polygon is reversed, j and d are not recorded according to (c) while x and y are according to (d). Hence segment kj becomes ky and segment cd becomes cx.

6 Implementation

6.1 Data Structures

The main data structure is a mobile dual pixel, `DualPixel`, which stores:

- The levels at its four corners;
- The coordinates of its top-left corner;
- The cardinal direction, in the set $\{S, E, N, W\}$, of the orientation of the line.

For instance, the direction S means the line is oriented south, so that it enters the dual pixel by its upper edgel. The directions are ordered such that turning left corresponds to taking the next direction circularly, while turning right corresponds to taking the previous direction, in the ordered set (S, E, N, W). To each direction is associated a unit translation vector Δ; for instance, $\Delta(S) = (0, 1)$ since conventionally the y coordinate increases by going south. This allows moving the dual pixel with an economy of code, without testing four different cases.

The other data structure is `Hyperbola`, which records all parameters related to the hyperbola inside a dual pixel: its center $s = (x_s, y_s)$ (saddle point of the image if located inside the dual pixel), its vertex v, and the parameter δ. It also records the numerator `num` and denominator `denom` of the saddle level λ_s as a fraction in (6). The case `denom` $= 0$ means that the image level is planar inside the dual pixel, so that the level line is locally a straight segment, *not* a hyperbola, and then s, v and δ make no sense. Notice that `denom` is an integer since we assumed that the data values are integers too (they have type `unsigned char` in our implementation), so that no threshold is used to test whether it vanishes. Still, nothing prevents an implementation with floating point input levels, and in that case a threshold would be needed.

For the reconstruction algorithm of Section 5, a structure `PolyIterator` is used: it records the current vertex of the polygon whose enclosed region is to be filled, whether the previous side was horizontal (only relevant when currrent point is on a scan line), and the direction of the previous side: down/up if the previous point and q have different ordinates, or right/left if they are both on a scanline. When a new point p is examined, the rules of Figure 5 are applied.

6.2 Online Demo

The online demo lets the user choose a quantization step $\Delta\lambda$ for the level lines and outputs the quantized image with the level lines superimposed. Their color is linked to their depth in the enclosure tree, from purple to red, through blue, green and yellow. The levels of extraction are $0.5 + k\Delta\lambda$ with $k \in \mathbb{N}$. As it is necessary to have a constant level at the image border (1), the pixels at the border have their level replaced by their median value.

As shown by Figure 7, the reconstructed image may have some defects in the sense that a level line may seem to separate two regions at the same gray level. This is the case for the fangs of the crocodile. The problem stems from the fact that the enclosed region of the level line is put at the level of the line, whereas we could have an upper or lower level set in the interior. The orientation of the level line is not useful to deduce which, since the interior is not known at the time of extraction. To fix that, we can fill the enclosed region with level $\lambda \pm \Delta\lambda/2$, the sign being deduced from the nature of the interior level set (addition for upper, subtraction for lower). The latter can be determined by Algorithm 6: the level λ of a level line L is compared to the one of its parent λ_0. If they differ, the interior of L is of type upper when $\lambda > \lambda_0$, and lower when $\lambda < \lambda_0$. If $\lambda = \lambda_0$, the interior of L is a hole in the enclosed region of the parent, and its type is opposite to the one of the region enclosed by the parent.

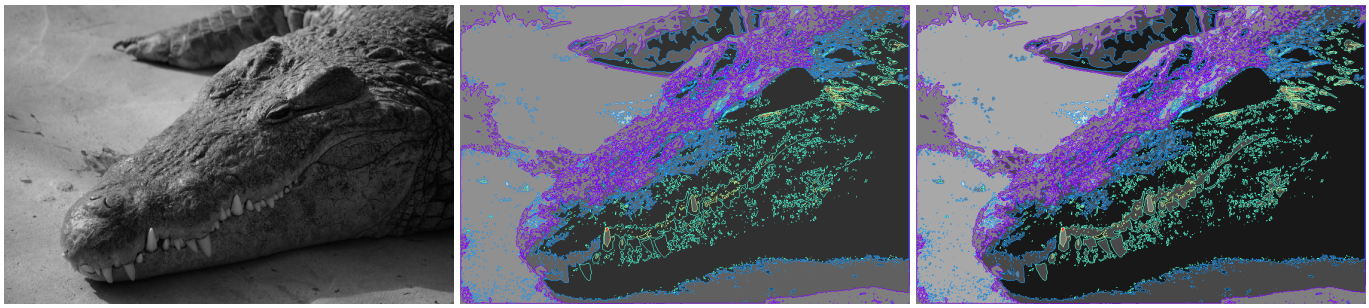


Figure 7: Image and its level lines with quantization step $\Delta\lambda = 48$. When the enclosure of a level line is filled by its level, some level lines enclose regions at same level as outside (central figure). After correcting the filling level by $\pm\Delta\lambda/2$ according to the nature of the enclosed region (see Algorithm 6), this defect does not happen anymore (right figure).

Algorithm 6: Determination of the type of the enclosed region of a level line

```
foreach level line L do                    // Iteration in preorder, parent before child
    λ ← level(L), λ0 ← level(parent(L))
    if λ > λ0 then
        type(L) = upper

    if λ < λ0 then
        type(L) = lower

    if λ = λ0 then
        type(L) = opposite(type(parent(L)))                          // Opposite type
```

Some examples of level lines obtained by the online demo with default quantization $\Delta\lambda = 32$ are shown in Figure 8.

7 Extensions

7.1 Other Interpolation Kernels

The essential property of our interpolation kernel is that it is interpolating (value 1 at 0, value 0 at other data points) and monotonous between adjacent pixels. We have

$$k(x,y) = \phi(x)\cdot\phi(y) \qquad \text{with} \qquad \phi(x) = (1-|x|)_+. \tag{11}$$

Let us state the essential properties of ϕ that were used in the algorithm:

1. ϕ is continuous.
2. ϕ is even.
3. $\phi(n) = \delta_n$ when $n \in \mathbb{Z}$.
4. $\phi(x) + \phi(1-x) = 1$ for $x \in [0,1]$ (partition of unity).
5. ϕ is strictly monotonous on $[-1,0]$ and on $[0,1]$, null outside $[-1,1]$.

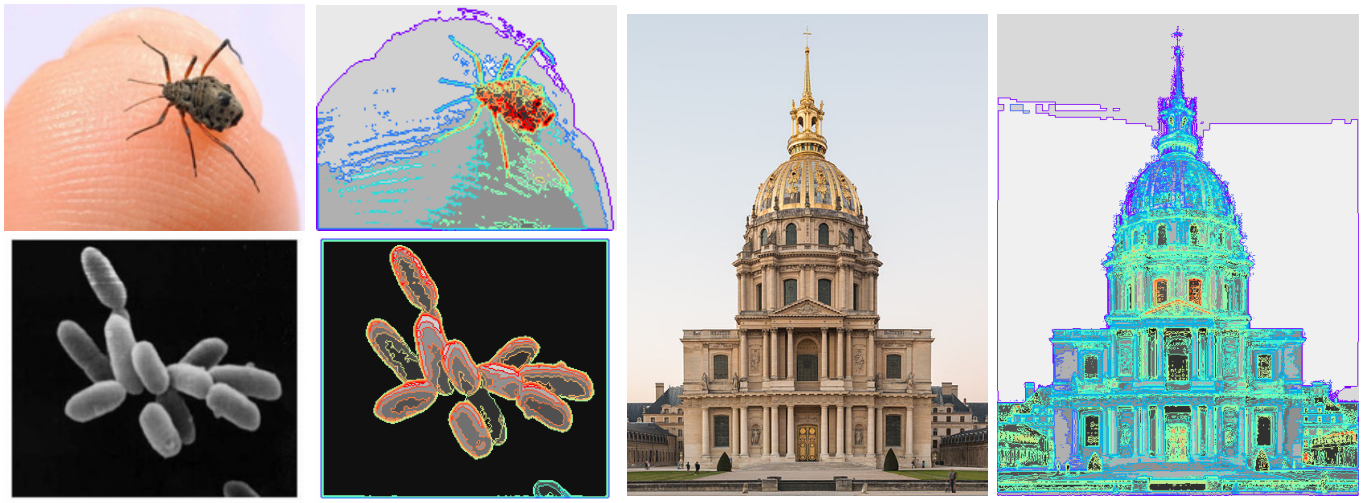


Figure 8: Examples of level lines from the online demo, with default quantization step (32).

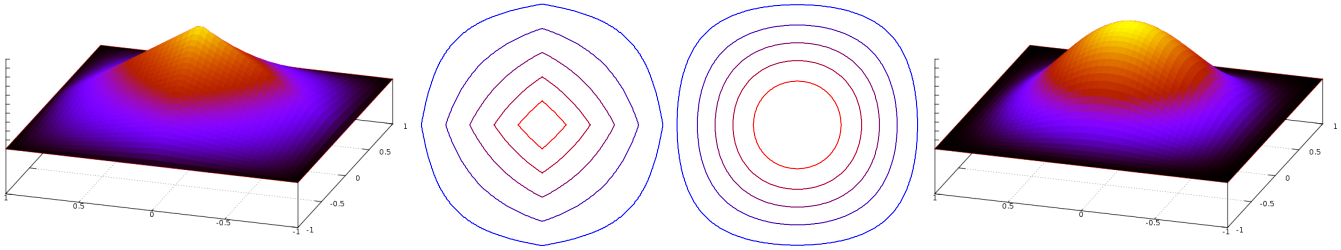


Figure 9: The regular bilinear kernel, i.e., order 1 spline (left), and a smoother alternative (12) (right), together with their level lines.

We can deduce that $\phi \geq 0$ and $\arg\max(\phi) = 0$. The continuous image inherits the properties of ϕ: continuous but not differentiable in the bilinear case. We could also consider a smoother function satisfying the same properties, such as

$$\phi(x) = \frac{1 + \cos(\pi \cdot x)}{2} \chi_{[-1,1]}(x), \tag{12}$$

which is differentiable everywhere (see Figure 9). The level lines are not anymore piecewise hyperbolas, but they are $\mathcal{C}^1$. The tangent of a level line is horizontal at a vertical edgel and vertical at a horizontal edgel. Apart from that, the extraction algorithm is fairly similar: the separability of the kernel and the fact that the function is monotonous makes the level line at a saddle level a union of two straight line segments inside a dual pixel. The explicitation of the level line as $x(y)$ or $y(x)$ can also be computed using inverse trigonomic functions. Inside the dual pixel $[0,1]^2$, the interpolated function is

$$u(x,y) = u[p_0]\,\phi(x)\phi(y) + u[p_1]\,\phi(x)(1-\phi(y)) + u[p_2]\,(1-\phi(x))(1-\phi(y)) + u[p_3]\,(1-\phi(x))\phi(y). \tag{13}$$

This can be rewritten

$$u(x,y) = a + b\,\phi(x) + c\,\phi(y) + d\,\phi(x)\phi(y), \tag{14}$$

and if $d \neq 0$:

$$u(x,y) = d\,\left(\phi(x) + \frac{c}{d}\right)\left(\phi(y) + \frac{b}{d}\right) + \frac{ad - bc}{d}. \tag{15}$$

When $ad - bc = \lambda$, the level line at level λ is the union of two segments:

$$\{\phi|^{-1}_{[0,1]}(-c/d)\} \times [0,1] \qquad\qquad [0,1] \times \{\phi|^{-1}_{[0,1]}(-b/d)\}. \tag{16}$$

7.2 Parallelization

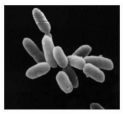
The algorithm is intrinsically fast, because its complexity is proportional to its output size: the sum of lengths of the extracted level lines. Still, for a very fine quantization of the gray levels and on large images, this can become troublesome and a parallel computation would be better. Fortunately, this is not difficult to achieve. The extraction algorithm 1 can perform the loop of levels $\lambda \in \mathbb{L}$ independently. Each thread T can handle a bunch of levels λ and use its own array v_T for visited edgels. When appending a level line L, it can do so in its own local store to avoid a lock on the global list $\{L\}$, and just merge the lists at the end. The algorithm is thus lock-free.

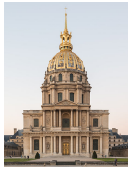
However, if the enclosure tree is required, we must record each intersection of level line with horizontal edgel at line 7 of Algorithm 1, see Section 4, and each level line must have its own unique identifier, typically its index in the final list $\{L\}$. Since it is unknown while the thread local lists are not merged, we use its local index. Therefore, each thread handles its own (I_y^T), $y = 0 \dots h-1$, each being a set of (x_j^{yT}, i_j^{yT}). When the threads merge at the end of the extraction, we can concatenate the lists (I_y^T) for all T in the same order as the $\{L\}_T$, and increasing each index i_j^{yT} by the number of elements in $\{L\}$ up to the previous concatenation.

Once the lists are merged, the parent/child relationship can be performed sequentially. In case it also needs to be parallelized, each I_y can be handled independently. However, when registering the parent of a level line L, it is preferable to get a lock in order that two threads cannot write simultaneously, even though they must agree on what to write. An alternative is that each thread keeps its view of parent/child relationship and that the results are consolidated at the end.

Image Credits

Images by the author (license CC-BY-SA-4.0) except:

 ©Rothamsted Research Center  public domain[3]

 ©Marc Pierrot-Deseilligny, CC-BY-SA-4.0  ©Daniel Vorndran, CC-BY-SA-3.0[4]

References

[1] C. Ballester, V. Caselles, and P. Monasse, *The tree of shapes of an image*, ESAIM: Control, Optimisation and Calculus of Variations, 9 (2003), pp. 1–18. https://doi.org/10.1051/cocv:2002069.

[2] T. Briand and P. Monasse, *Theory and Practice of Image B-Spline Interpolation*, Image Processing On Line, 8 (2018), pp. 99–141. https://doi.org/10.5201/ipol.2018.221.

[3] V. Caselles and P. Monasse, *Geometric Description of Images as Topographic Maps*, vol. 1984 of Lecture Notes in Mathematics, Springer, 2009. ISBN 978-3-642-04611-7.

[3] https://en.wikipedia.org/wiki/Archaea#/media/File:Halobacteria.jpg
[4] https://commons.wikimedia.org/wiki/File:Cathdrale_Saint-Louis-des-Invalides,_140309_2.jpg

[4] A. Ciomaga, P. Monasse, and J. Morel, *Image visualization and restoration by curvature motions*, Multiscale Modeling & Simulation, 9 (2011), pp. 834–871. https://doi.org/10.1137/100791269.

[5] A. Ciomaga, P. Monasse, and J-M. Morel, *The Image Curvature Microscope: Accurate Curvature Computation at Subpixel Resolution*, Image Processing On Line, 7 (2017), pp. 197–217. https://doi.org/10.5201/ipol.2017.212.

[6] R. M. Haralick, S. R. Sternberg, and X. Zhuang, *Image analysis using mathematical morphology*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9 (1987), pp. 532–550. https://doi.org/10.1109/TPAMI.1987.4767941.

[7] M. Mondelli, *A Finite Difference Scheme for the Stack Filter Simulating the MCM*, Image Processing On Line, 3 (2013), pp. 68–111. https://doi.org/10.5201/ipol.2013.53.