Geometric Total Variation for Image Vectorization, Zooming and Pixel Art Depixelizing [*]

Bertrand Kerautret[1] and Jacques-Olivier Lachaud[2]

[1] Univ. Lyon 2, LIRIS, CNRS, France
`bertrand.kerautret@liris.cnrs.fr`
[2] Univ. Grenoble Alpes, Univ. Savoie Mont Blanc
CNRS, LAMA, 73000 Chambéry, France
`jacques-olivier.lachaud@univ-smb.fr`

Abstract. We propose an original method for vectorizing an image or zooming it at an arbitrary scale. The core of our method relies on the resolution of a geometric variational model and therefore offers theoretic guarantees. More precisely, it associates a total variation energy to every valid triangulation of the image pixels. Its minimization induces a triangulation that reflects image gradients. We then exploit this triangulation to precisely locate discontinuities, which can then simply be vectorized or zoomed. This new approach works on arbitrary images without any learning phase. It is particularly appealing for processing images with low quantization like pixel art and can be used for depixelizing such images. The method can be evaluated with an online demonstrator, where users can reproduce results presented here or upload their own images.

Keywords: Image Vectorization · Image Super-Resolution · Total Variation · Depixelizing Pixel Art

1   Introduction

There exist many methods for zooming into raster images, and they can be divided into two groups. The first group gathers "image super-resolution" or "image interpolation" techniques and tries to deduce colors on the zoomed image from nearby colors in the original image. These methods compute in the raster domain and their output is not scalable. The second group is composed of "image vectorization" approaches and tries to build a higher level representation of the image made of painted polygons and splines. These representations can be zoomed, rotated or transformed. Each approach has its own merits, but none perform well both on cmaera pictures and on drawings or pixel art.

Image interpolation/super resolution. These approaches construct an interpolated image with higher resolution. A common trait is that they see the image as a continuous function, such that we only see its sampling in the original image.

Their diversity lies generally in the choice of the space of continuous functions. For instance linear/cubic interpolation just fits linear/cubic functions, but much more complicated spaces have been considered. For instance the method proposed by Getreuer introduces the concept of *contour stencils* [6,7] to accurately estimate the edge orientation. Roussos and Maragos proposed another strategy using a tensor driven diffusion [19] (see also [8]). Due to the increasing popularity of convolutional neural network (CNN), several authors propose super resolution networks to obtain super resolution image. For instance Shi *et al.* define a CNN architecture exploiting feature maps computed in low resolution space [22]. After learning, this strategy reduces the computational complexity for a fixed super-resolution and gives interesting results on photo pictures. Of course, all these approaches do not provide any vectorial representation and give low quality results on images with low quantization or pixel art images. Another common feature is their tendency to "invent" false oscillating contours, like in Fourier super-resolution.

Pixel art super-resolution. A subgroup of these methods is dedicated to the interpolation of pixel art images, like the classic HqX magnification filter [23] where X represents the scale factor. We may quote a more advanced approach which proposes as well a full vectorization of pixel art image [13]. Even if the resulting reconstruction for tiny images is nice looking, the proposed approach is designed specifically for hand-crafted pixel art image and is not adaptable to photo pictures.

Image vectorization. Recovering a vector-based representation from a bitmap image is a classical problem in the field of digital geometry, computer graphics and pattern recognition. Various commercial design softwares propose such a feature, like *Indesign*$^{\text{TM}}$, or Vector Magic [10] and their algorithms are naturally not disclosed. Vectorization is also important for technical document analysis and is often performed by decomposition into arcs and straight line segments [9]. For binary images, many vectorization methods are available and may rely on dominant points detection [16,17], relaxed straightness properties [1], multi-scale analysis [5] or digital curvature computation [11,15].

Extension to grey level images is generally achieved by image decomposition into level sets, vectorization of each level, then fusion (e.g. [12]). Extension to color images is not straightforward. Swaminarayan and Prasad proposed to use contour detection associated to a Delaunay triangulation [25] but despite the original strategy, the digital geometry of the image is not directly exploited like in above-mentionned approaches. Other methods define the vectorization as a variational problem, like Lecot and Levy who use the Mumford Shah piecewise smooth model and an intermediate triangulation step to best approximate the image [14]. Other comparable methods construct the vectorization from topological preservation criteria [24], from splines and adaptive Delaunay triangulation [4] or from Bézier-curves-patch-based representation [26]. We may also cite the interactive method proposed by Price and Barrett that let a user edit interactively the reconstruction from a graph cut segmentation [18].

Our contribution. We propose an original approach to this problem, which borrows and mixes techniques from both groups of methods, thus making it very versatile. We start with some notations that will be used thoughout the paper. Let Ω be some open subset of $\mathbb{R}^2$, here a rectangular subset defining the image domain. Let K be the image range, a vector space that is generally $\mathbb{R}$ for monochromatic images and $\mathbb{R}^3$ for color image, equipped with some norm $\|\cdot\|_K$. We assume in the following that gray-level and color components lie in $[0,1]$.

First our approach is related with the famous Total Variation (TV), a well known variational model for image denoising or inpainting [20]. Recall that if f is a differentiable function in Ω, its total variation can be simply defined as:

$$\mathrm{TV}(f) := \int_\Omega \|\nabla f(x)\|_K \, \mathrm{d}x. \tag{1}$$

In a more general setting, the total variation is generally defined by duality. As noted by many authors [3], different discretizations of TV are not equivalent. However they are all defined locally with neighboring pixels, and this is why they are not able to fully capture the structure of images.

In Section 2, we propose a *geometric total variation*. Its optimization seeks the triangulation of lattice points of Ω that minimizes a well-defined TV corresponding to a piecewise linear function interpolating the image. The optimal triangulation does not always connect neighboring pixels and align its edges with image discontinuities. For instance digital straight segments are recognized by the geometric total variation.

In Section 3, we show how to construct a vector representation from the obtained triangulation, by regularizing a kind of graph dual to the triangulation. Section 4 shows a super-resolution algorithm that builds a zoomed raster image with smooth Bezier discontinuities from the vector representation. Finally, in Section 5 we compare our approach to other super-resolution and vectorization methods and discuss the results.

2   Geometric Total Variation

The main idea of our geometric total variation is to structure the image domain into a triangulation, whose vertices are the pixel centers located at integer coordinates. Furthermore, any triangle of this triangulation should cover exactly three lattice points (its vertices). The set of such triangulations of lattice points in Ω is denoted by $\mathcal{T}(\Omega)$.

Let s be an image, i.e. a function from the set of lattice points of Ω to K. We define the *geometric total variation* of s with respect to an arbitrary triangulation T of $\mathcal{T}(\Omega)$ as

$$\mathrm{GTV}(T,s) := \int_\Omega \|\nabla \Phi_{T,s}(x)\|_K \, dx, \tag{2}$$

where $\Phi_{T,s}(x)$ is the linear interpolation of s in the triangle of T containing x. Note that points of Ω whose gradient is not defined have null measure in the above integral (they belong to triangle edges of T).

And the Geometric Total Variation (GTV) of s is simply the smallest among all possible triangulations:

$$\text{GTV}_{\mathcal{T}(\Omega)}(s) := \min_{T \in \mathcal{T}(\Omega)} \text{GTV}(T, s). \tag{3}$$

In other words, the GTV of a digital image s is the smallest continuous total variation among all natural triangulations sampling s. Since $\mathcal{T}(\Omega)$ will not change in the following, we will omit it as subscript and write simply $\text{GTV}(s)$.

One should note that classical discrete TV models associated to a digital image generally overestimate its total variation (in fact, the perimeter of its level-sets due to the co-area formula) [3]. Classical discrete TV models $\text{TV}(s)$ are very close to $\text{GTV}(T, s)$ when T is any trivial Delaunay triangulation of the lattice points. By finding more meaningful triangulation, $\text{GTV}(s)$ will often be much smaller than $\text{TV}(s)$.

Combinatorial expression of geometric total variation. Since $\Phi_{T,s}(x)$ is the linear interpolation of s in the triangle τ of T containing x, it is clear that its gradient is constant within the interior of τ. If $\tau = \mathbf{pqr}$ where $\mathbf{p}, \mathbf{q}, \mathbf{r}$ are the vertices of τ, one computes easily that for any x in the interior of τ, we have

$$\nabla \Phi_{T,s}(x) = s(\mathbf{p})(\mathbf{r} - \mathbf{q})^{\perp} + s(\mathbf{q})(\mathbf{p} - \mathbf{r})^{\perp} + s(\mathbf{r})(\mathbf{q} - \mathbf{p})^{\perp}. \tag{4}$$

We thus define the discrete gradient of s within a triangle $\mathbf{pqr}$ as

$$\boldsymbol{\nabla} s(\mathbf{pqr}) := s(\mathbf{p})(\mathbf{r} - \mathbf{q})^{\perp} + s(\mathbf{q})(\mathbf{p} - \mathbf{r})^{\perp} + s(\mathbf{r})(\mathbf{q} - \mathbf{p})^{\perp}, \tag{5}$$

where $\mathbf{x}^{\perp}$ is the vector $\mathbf{x}$ rotated by $\frac{\pi}{2}$. Now it is well known that any lattice triangle that has no integer point in its interior has an area $\frac{1}{2}$ (just apply Pick's theorem for instance). It follows that for any triangulation T of $\mathcal{T}(\Omega)$, every triangle has the same area $\frac{1}{2}$. We obtain the following expression for the GTV:

$$
\begin{aligned}
\text{GTV}(T, s) &= \int_{\Omega} \|\nabla \Phi_{T,s}(x)\|_K \, dx \\
&= \sum_{\mathbf{pqr} \in T} \int_{\mathbf{pqr}} \|\nabla \Phi_{T,s}(x)\|_K \, dx && \text{(integral is additive)} \\
&= \sum_{\mathbf{pqr} \in T} \int_{\mathbf{pqr}} \|\boldsymbol{\nabla} s(\mathbf{pqr})\|_K \, dx && \text{(by (4) and (5))} \\
&= \frac{1}{2} \sum_{\mathbf{pqr} \in T} \|\boldsymbol{\nabla} s(\mathbf{pqr})\|_K && \text{(since Area}(\mathbf{pqr}) = \frac{1}{2}) \quad (6)
\end{aligned}
$$

Minimization of GTV. Since we have a local combinatorial method to compute the GTV of an arbitrary triangulation, our approach to find the optimal GTV of some image s will be greedy, iterative and randomized. We will start from an arbitrary triangulation (here the natural triangulation of the grid where each square has been subdivided in two triangles) and we will flip two triangles whenever it decreases the GTV.
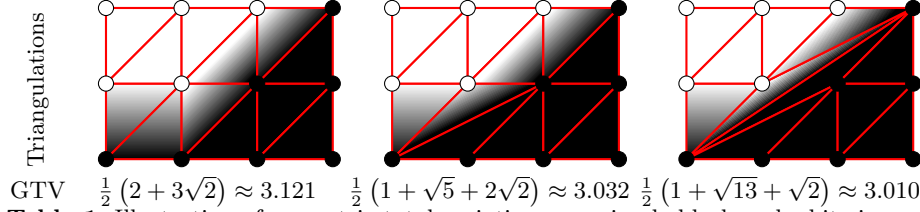
Triangulations

GTV $\frac{1}{2}\left(2+3\sqrt{2}\right) \approx 3.121$ $\frac{1}{2}\left(1+\sqrt{5}+2\sqrt{2}\right) \approx 3.032$ $\frac{1}{2}\left(1+\sqrt{13}+\sqrt{2}\right) \approx 3.010$

Table 1. Illustration of geometric total variation on a simple black and white image. Even if the pixel values of the image are the same (represented by black and white disks), different triangulations of the pixels yield different GTV. In this case, the right one is the optimal triangulation and is clearly more aligned with the underlying contour. Vectorizing or zooming the right triangulation will provide more interesting results.

This approach is similar in spirit to the approach of Bobenko and Springborn [2], whose aim is to define discrete minimal surfaces. However they minimize the Dirichlet energy of the triangulation (i.e. squared norm of gradient) and they show that the optimum is related to the intrinsic Delaunay triangulation of the sampling points, which can be reached by local triangle flips. On the contrary, our energy has not this nice convexity property and it is easy to see for instance that minima are generally not unique. We have also checked that there is a continuum of energy if we minimize the p-norm $\|\cdot\|_K^p$ for $1 \leq p \leq 2$. For $p = 2$ the optimum is trivially the Delaunay triangulation of the lattice points. When p is decreased toward 1, we observe that triangle edges are more and more perpendicular to the discrete gradient.

Figure 1 shows the GTV of several triangulations in the simple case of a binary image representing an edge contour with slope $\frac{2}{3}$. It shows that the smaller the GTV the more align with the digital contour is the triangulation.

Algorithm 1 is the main function that tries to find a triangulation T which is as close as possible to GTV(s). It starts from any triangulation of s (line 6 : we just split into two triangles each square between four adjacent pixels). Then it proceeds by passes (line 7). At each pass (line 10), every potential arc is checked for a possible flip by a call to CHECKARC (Algorithm 2) at line 13. The arc is always flipped if it decreases the GTV and *randomly* flipped if it does not change the GTV (line 14). Arcs close to the flip are queued (line 16) since their further flip may decrease the GTV later. The algorithm stops when the number of flips decreasing the GTV is zero (flips keeping the same GTV are not counted).

Function CHECKARC (Algorithm 2) checks if flipping some arc would decrease the GTV. It first checks if the arc/edge is flippable at line 3 (e.g. not on the boundary). Then it is enough to check only one arc per edge (line 5). Afterwards the edge may be flipped only if the four points surrounding the faces touching the edge form a strictly convex quadrilateron (line 6). Finally it computes the local GTV of the two possible decompositions of this quadrilateron using formula (6) and outputs the corresponding case (from line 7).

Note that the *randomization* of flips in the case where the energy GTV is not changed is rather important in practice. As said above, it is not a "convex"

```
1 function OptimizeGTV( s : Image ) : Triangulation ;
2 var T : Triangulation /* the triangulation that is optimized    */;
3 var Q : Queue<Arc>/* the queue of arcs currently being checked */;
4 var Q' : Queue<Arc>/* the queue of arcs to be checked after    */;
5 begin
6 │   T ← TrivialTriangulation(Pixels(s));
  │   /* All arcs may be flipped at the beginning.                */
7 │   for all arcs a of T do Push a in Q';
8 │   repeat
9 │   │   Swap(Q, Q');
10│   │   n ← 0   /* Counts the number of flips that decrease GTV */;
11│   │   while ¬IsEmpty(Q) do
12│   │   │   Pop a from Q ;
13│   │   │   c ← CheckArc(s, T, a) ;
14│   │   │   if c > 0 or (c = 0 and FlipCoin() = Heads) then
15│   │   │   │   if c > 0 then n ← n + 1;
16│   │   │   │   Push arcs of two faces around a in Q';
17│   │   │   │   Flip(T, a)   /* After, a represents the flipped arc */;
18│   until n = 0;
19│   return T;
```

Algorithm 1: Function OptimizeGTV outputs a triangulation which is as close as possible to GTV(s). It builds a trivial triangulation of the pixels of *s* then optimize its GTV by flipping its edges in a greedy and randomized way.

```
1 function CheckArc( s : Image, T : Triangulation, a : Arc ) : Integer ;
2 begin
3 │   if ¬isFlippable(T, a) then return -1;
4 │   P ← VerticesOfFacesAroundArc(T, a) ;
  │   /* P[0] is Tail(T, a), P[2] is Head(T, a), P[0]P[1]P[2] and
  │      P[0]P[2]P[3] are the two faces having a in common.       */
5 │   if P[0] < P[2] then return -1;
6 │   if ¬isConvex(P) then return -1;
7 │   $E_{\text{cur}} = \|\boldsymbol{\nabla}s(P[0]P[1]P[2])\|_K + \|\boldsymbol{\nabla}s(P[0]P[2]P[3])\|_K$ ;
8 │   $E_{\text{flip}} = \|\boldsymbol{\nabla}s(P[0]P[1]P[3])\|_K + \|\boldsymbol{\nabla}s(P[1]P[2]P[3])\|_K$ ;
9 │   if $E_{\text{flip}} < E_{\text{cur}}$ then return 1;
10│   else if $E_{\text{flip}} = E_{\text{cur}}$ then return 0;
11│   else return -1;
```

Algorithm 2: Function CheckArc checks if flipping the arc *a* of triangulation *T* decreases GTV(*T*, *s*) (returns 1), does not change GTV(*T*, *s*) (returns 0), or returns −1 in all the other cases.
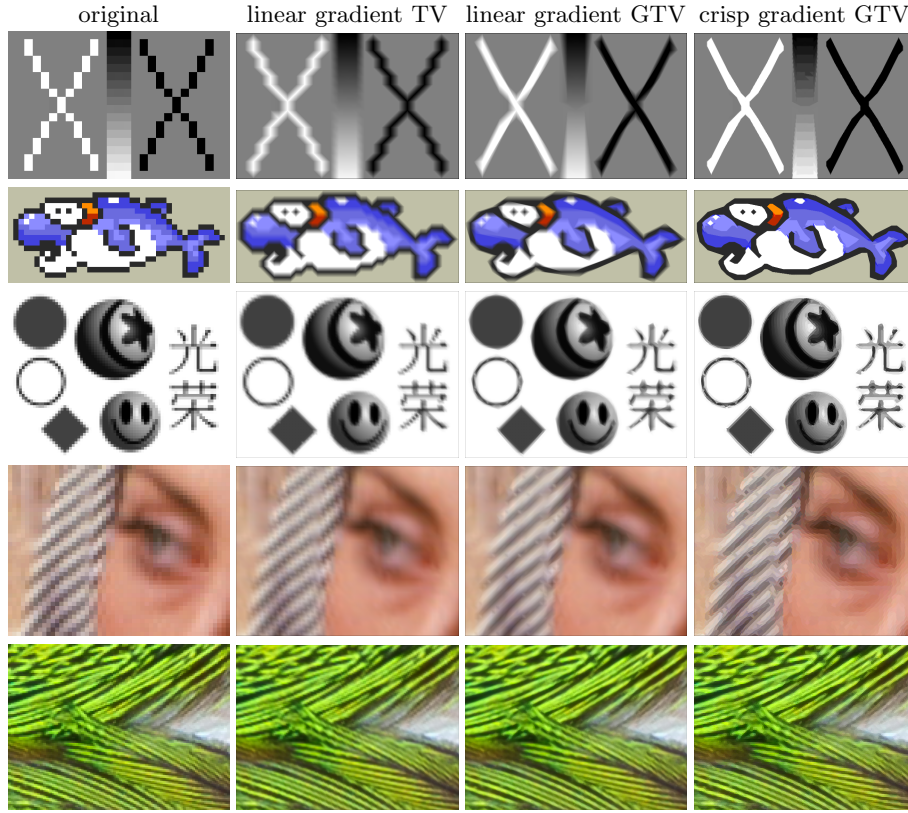
Fig. 1. Zoom ×16 of the images in leftmost column: (middle left) displaying the trivial triangulation given by a simple discretization of TV painted with induced linear gradient over each triangle, (middle right) displaying the triangulation after GTV optimization painted with induced linear gradient over each triangle, (rightmost) same as before except that triangles are painted with a crisped version of the same gradient.

energy since it is easy to find instances where there are several optima (of course, our search space being combinatorial, convexity is not a well defined notion). Randomization helps in quitting local minima and this simple trick gives nice results in practice.

Figure 1 illustrates the capacity of geometric total variation to capture the direction of image discontinuities. In this simple application, we use the discrete gradient on each triangle to display it with the corresponding linear shaded gradient. Hence we can zoom arbitrarily any image. The results show that, if we stay with the initial triangulation (as does standard discretization of TV), zoomed image are not great. On the contrary, results are considerably improved if we use the optimized triangulation of GTV. Last, we can simply render triangles with a crisped version of this gradient. Results are very nice in case of images with few colors like in pixel art.

3   Contour Regularization

Contours within images are in-between pixels. In some sense they are dual to the structure of the pixels. Since the Geometric Total Variation has structured the relations between pixels, it is natural to define the contours as a kind of dual graph to the optimal triangulation T^*. We wish also that these contour lines align nicely with discontinuities.
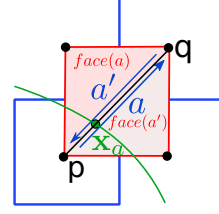
To do so, we introduce a variable t_a on each arc whose value is kept in $[0, 1]$. If the arc a is the oriented edge $(\mathbf{p}, \mathbf{q})$ then the position of the contour crossing this edge will be $\mathbf{x}_a = t_a\mathbf{p} + (1 - t_a)\mathbf{q}$ (see the upper floating figure).

We denote by a' the arc opposite to a, i.e. $(\mathbf{q}, \mathbf{p})$. The face that is to the left of a is denoted face(a) while the one to its right is face$'(a)$. We will guarantee that, at the end of each iteration, $\mathbf{x}_a = \mathbf{x}_{a'}$. We associate to each arc $a = (\mathbf{p}, \mathbf{q})$ a *dissimilarity weight* $w_a := \|s(\mathbf{q}) - s(\mathbf{p})\|_K = w_{a'}$. We introduce also a point $\mathbf{b}_f$ for each face f of T, which will lie at a kind of weighted barycenter of the vertices of f.

Algorithm 3 regularizes these points and provides a graph of contours that is a meaningful vectorization of the input bitmap image s. Note that the function INTERSECTION$(a, \mathbf{y}, \mathbf{z})$ returns the parameter t such that $\mathbf{x}_a$ lies at the intersection of straight line $(\mathbf{yz})$ and the arc a.

First it starts with natural positions for $\mathbf{x}$ and $\mathbf{b}$ (resp. middle of arcs and barycenter of faces) at line 3 and line 4. Then it proceeds iteratively until stabilization in the loop at line 8. Each iteration consists of three steps: (i) update barycenters such that they are a convex combination of surrounding contour points weighted by their dissimilarities (line 9), (ii) update contour points such that they lie at the crossing of the arc and the nearby barycenters (line 11), (iii) average the two displacements along each edge according to respective area (line 13), thus guaranteeing that $\mathbf{x}_a = \mathbf{x}_{a'}$ for every arc. The area AREA(a) associated to an arc a is the area of the quadrilateron formed by the tail of a, barycenters $\mathbf{b}_{\text{face}(a)}$ and $\mathbf{b}_{\text{face}'(a)}$ and $\mathbf{x}_a$. Note that the coefficients α and α' computed at line 14 have value 1 whenever the area is $\frac{1}{6}$.

Since this process is always computing convex combinations of points with convex constraints, it converges quickly to a stable point. In all our experiments, we choose $\epsilon = 0.001$ and the process converges in a dozen of iterations. Figure 2 illustrates it. The *contour mesh* is defined simply as follows: there is one cell per pixel, and for each pixel $\mathbf{p}$, its cell is obtained by connecting the sequence of points $\mathbf{x}_{a_0}, \mathbf{b}_{\text{face}(a_0)}, \mathbf{x}_{a_1}, \mathbf{b}_{\text{face}(a_1)} \ldots$ for every arc $a_0, a_1, \ldots$ coming out of $\mathbf{p}$. Each cell of the contour mesh is displayed painted with the color of its pixel. It is clear that the contour mesh is much more meaningful after GTV optimization, and its further regularization remove some artefacts induced by the discretization. Last but not least, our approach guarantees that sample points always keep their original color.

1 function REGULARIZECONTOURS*(s : Image, T : Triangulation)* ;

Result: Positions of contour points on arcs and barycenter on faces

2 begin

3   **for** *every face $f = \mathbf{pqr}$ of T* **do** $\mathbf{b}_f \leftarrow \frac{1}{3}(\mathbf{p} + \mathbf{q} + \mathbf{r})$;

4   **for** *every arc a of T* **do**

5   **if** $\neg$ISUPDATEABLE(a) **then** $t_a^{(0)} \leftarrow \frac{1}{2}$;

6   **else** $t_a^{(0)} \leftarrow$ INTERSECTION$(a, \mathbf{b}_{\text{face}(a)}, \mathbf{b}_{\text{face}'(a)})$;

7 $n \leftarrow 0$;

8   **repeat**

9   **for** *every face f of T* **do** /* Update barycenters   */

10 $\mathbf{b}_f \leftarrow \frac{1}{2}(\mathbf{b}_f + \sum_{a \in \partial f} w_a \mathbf{x}_a / \sum_{a \in \partial f} w_a)$;

11   **for** *every arc a of T with* ISUPDATEABLE(a) **do** /* Update contours

           */

12 $t_a^{(n)} \leftarrow \frac{1}{2}(t_a^{(n)} + $ INTERSECTION$(a, \mathbf{b}_{\text{face}(a)}, \mathbf{b}_{\text{face}'(a)})$;

        /* Average displacements along each edge according to area.

         */

13   **for** *every arc $a = \mathbf{pq}$ of T with $\mathbf{p} < \mathbf{q}$ and* ISUPDATEABLE(a) **do**

14 $(\alpha, \alpha') \leftarrow \frac{1}{2}(1 + 1/(6\text{AREA}(a)), 1 + 1/(6\text{AREA}(a')))$;

15 $t \leftarrow \frac{1}{2}(\alpha t_a^{(n)} + 1 - \alpha' t_{a'}^{(n)})$;

16 $(t_a^{(n+1)}, t_{a'}^{(n+1)}) \leftarrow (t, 1 - t)$;

17 $n \leftarrow n + 1$;

18   **until** $\max_{arc\ a\ of\ T} |t_a^{(n)} - t_a^{(n-1)}| < \epsilon$;

19   **return** $(\mathbf{x}, \mathbf{b})$

Algorithm 3: Function REGULARIZECONTOURS iteratively moves contour points and barycenters such that they align with the edges of the triangulation that delineate image discontinuities.
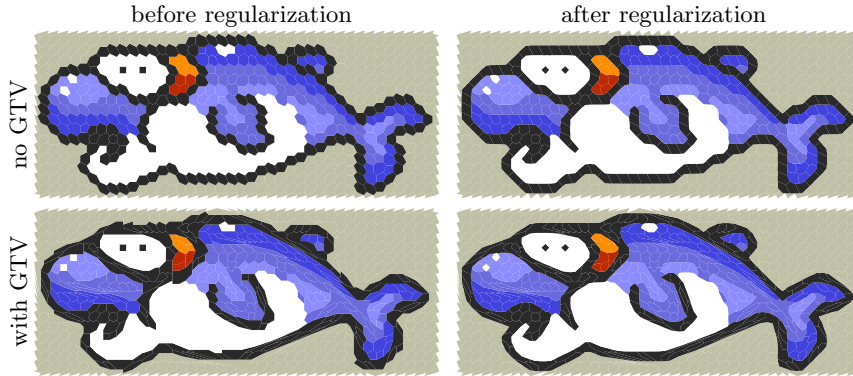

Fig. 2. Displays contour mesh: left column before regularization, right column after regularization with Algorithm 3. Top row shows contour meshes when using the initial trivial triangulation. Bottom row shows contour meshes when using the triangulation T^* that optimize GTV(s). Note that boundary triangles are displayed in white.

4   Raster Image Zooming with Smooth Contours

Contour meshes as presented in Section 3 are easily vectorized as polylines. It suffices to gather cells with same pixel color as a regions and extract the common boundaries of these regions. Furthermore, these polylines are easily converted to smooth splines. We will not explore this track in this paper but rather present a raster approach with similar objectives and features.

From the image s with lattice domain Ω, we wish to build a *zoomed image* s' with lattice domain Ω'. If the zoom factor is the integer z and Ω has width w and height h, then Ω' has width $z(w-1)+1$ and height $z(h-1)+1$. The canonical injection of Ω into Ω' is $\iota_z : (x,y) \mapsto (zx, zy)$. We use two auxiliary binary images S (for similarity image) and D for (for discontinuity image) with domain Ω'. We also define the *tangent* $\mathbf{T}_a$ at an arc a as the normalization of vector $\mathbf{b}_{\text{face}(a)} - \mathbf{b}_{\text{face}'(a)}$.

The zoomed image s' is constructed as follows:

Similarity set We set to 1 the pixels of S that are in $\iota_z(\Omega)$. Furthermore, for every arc $a = (\mathbf{p}, \mathbf{q})$, if $w_a = 0$ then the digital straight segment between $\iota_z(\mathbf{p})$ and $\iota_z(\mathbf{q})$ is also set to 1. Last, we set the color s' at these pixels to their color in s.

Discontinuity set For every face f, we count the number n of arcs whose weight is not null. If $n = 0$ we simply set $D(\iota_z(\mathbf{b}_f)) = 1$. $n = 1$ is impossible. If $n = 2$, let a_1 and a_2 be the two arcs with dissimilarities. We set $D(\mathbf{p}) = 1$ for every pixel $\mathbf{p} \in \Omega'$ that belongs to the digitized Bezier curve of order 3, that links the points $\iota_z(\mathbf{x}_{a_1})$ to $\iota_z(\mathbf{x}_{a_2})$, is tangent to $\mathbf{T}_{a_1}$ and $\mathbf{T}_{a_2}$, and pass through point $\iota_z(\frac{1}{2}(\mathbf{b}_f + I))$, with I the intersection of the two lines defined by $\mathbf{x}_{a_i}$ and tangent $\mathbf{T}_{a_i}$, $i = 1, 2$. If $n = 3$, for every arc a of f, we set $D(\mathbf{p}) = 1$ for every pixel $\mathbf{p} \in \Omega'$ that belongs to the digitized Bezier curve of order 2 connecting $\iota_z(\mathbf{x}_a)$ to $\iota_z(\mathbf{b}_f)$ and tangent to $\mathbf{T}_a$. Last we set the color s' at all these pixels by linear interpolation of the colors of pixels surrounding s.

Voronoi maps We then compute the voronoi map $\text{Vor}(S)$ (resp. $\text{Vor}(D)$), which associates to each pixel of Ω' the closest point $\mathbf{p}$ of Ω' such that $S(\mathbf{p}) = 1$ (resp. such that $D(\mathbf{p}) = 1$).

Image interpolation For all pixel $\mathbf{p}$ of Ω' such that $S(\mathbf{p}) = 0$ and $D(\mathbf{p}) = 0$, let $\mathbf{q} = \text{Vor}(S)(\mathbf{p})$ and $\mathbf{q}' = \text{Vor}(D)(\mathbf{p})$. We compute the distances $d = \|\mathbf{q} - \mathbf{p}\|$ and $d' = \|\mathbf{q}' - \mathbf{p}\|$. We use an amplification factor $\beta \in [0, 1]$: when close to 0, it tends to make linear shaded gradient while close to 1, it makes contours very crisp.

$$s'(\mathbf{p}) = \begin{cases} (1 - \frac{2d'}{d+d'})s'(\mathbf{q}') + \frac{2d'}{d+d'}(\beta s'(\mathbf{q}) + (1 - \beta)s'(\mathbf{q}')) \text{ when } d' \leq d, \\ (1 - \frac{2d}{d+d'})s'(\mathbf{q}) + \frac{2d}{d+d'}(\beta s'(\mathbf{q}) + (1 - \beta)s'(\mathbf{q}')) \text{ otherwise.} \end{cases}$$

In experiments, we always set $\beta = 0.75$ which gives good results, especially for image with low quantization. Of course, many other functions can be designed and the crispness could also be parameterized locally, for instance as a function of the GTV of the triangle.
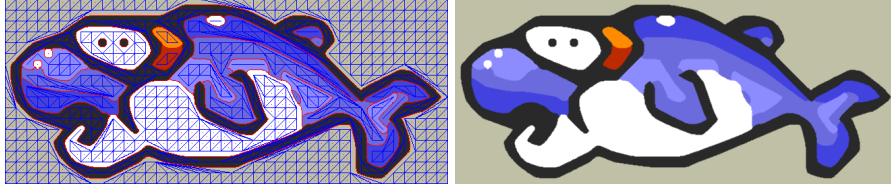
Fig. 3. Illustration of raster image zooming with smooth contours. On the left, similarity set is drawn in blue while discontinuity set is drawn in red. The final result is displayed on the right.
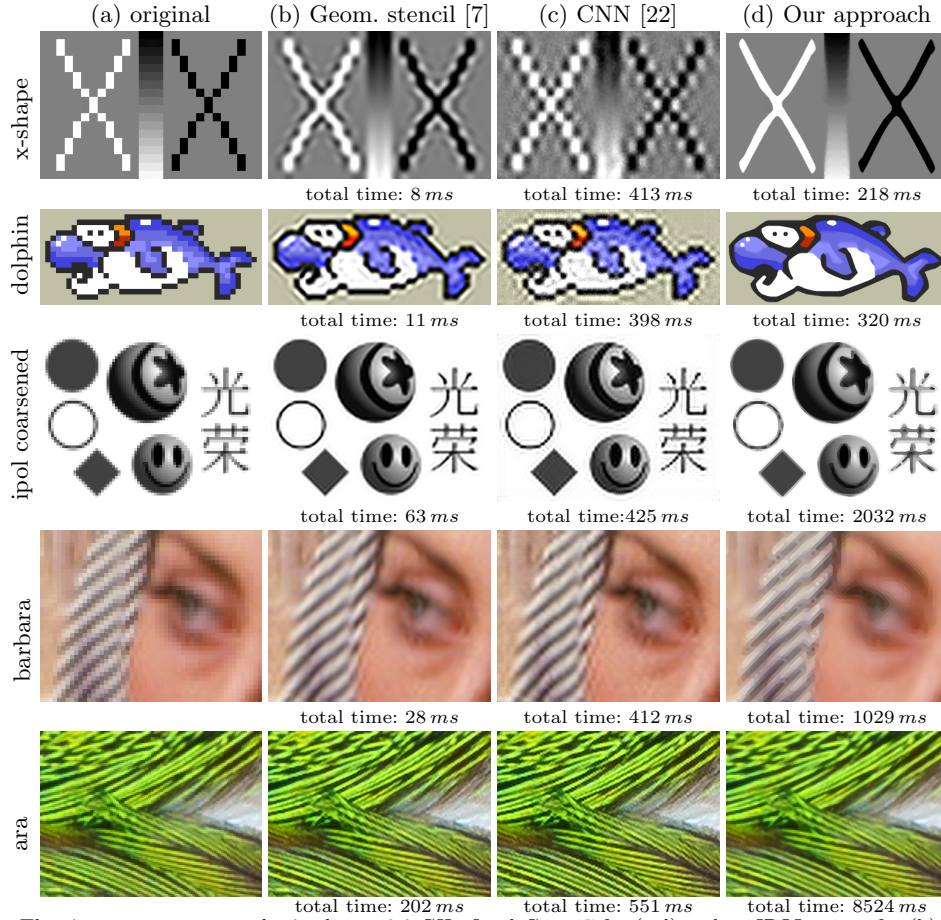
Antialiasing discontinuities To get images that are slightly more pleasant to the eye, we perform a last pass where we antialias every pixel **p** such that $D(\mathbf{p}) = 1$. We simply perform a weighted average within a 3×3 window, with pixels in S having weight 4, pixels in D having weight 0.25 and other pixels having weight 1.

Figure 3 illustrates this method of raster image zooming which provides crisp discontinuities that follow Bezier curves. Comparing with Figure 2, image contours are no more polygonal lines but look like smooth curves. Last this method still guarantees that original pixel colors are kept.

5   Experimental Evaluation and Discussion

Our method can both produce vectorized image or make rasterized zoomed images, either from camera pictures or tiny image with low quantization like pixel art images. To measure its performance, several other super-resolution methods have been tested on the set of images given in the first column of Figure 4, with parameters kept constant across all input images. First, we experimented the method based on geometric stencils proposed by Getreuer [6] with default parameters, which is implemented in the online demonstrator [7]. As shown in the second column of the figure, results appear noisy with oscillations near edges, which are well visible on pixel art images (e.g. x-shape or dolphin images). Such defaults are also visible on ara or barbara image near the white area. Another super-resolution method was experimented which uses on a convolutional neural network [22]. Like the previous method, numerous perturbations are also visible both on pixel art images (x-shape or dolphin) but also in homogeneous areas close to strong gradients of ipol-coarsened image. We have tried different parameters but they lead to images with comparable quality. On the contrary, due to its formulation, our method does not produce false contours or false colors, and works indifferently on pixel art images or camera pictures.

Other comparisons are presented on Figure 5 in order to give an overview of the behavior of five other methods. The two first methods complement the comparisons on pixel art image with respectively the depixelizing method proposed by Kopf and Lischinski [13] implemented in *Inkscape*, and a commercial sofware

	(a) original	(b) Geom. stencil [7]	(c) CNN [22]	(d) Our approach

x-shape — total time: 8 ms — total time: 413 ms — total time: 218 ms

dolphin — total time: 11 ms — total time: 398 ms — total time: 320 ms

ipol coarsened — total time: 63 ms — total time: 425 ms — total time: 2032 ms

barbara — total time: 28 ms — total time: 412 ms — total time: 1029 ms

ara — total time: 202 ms — total time: 551 ms — total time: 8524 ms

The time measures were obtained on a 2.9 GHz *Intel Core i7* for (c,d) and on IPOL server for (b).

Fig. 4. Comparison of the proposed approach (d) with other methods on Geometric Stencils [7] (b) and based on Convolution Neural Network [22] (c).

proposed by *Vector Magic Inc* [10]. Our method captures better the direction of discontinuities of the underlying shape with less contour oscillations (see the border of dolphin or x-shape). Two other methods were tested on barbara image: Roussos and Maragos tensor-driven method [8] and Potrace vectorization software [21]. Again, results appear with oscillations around strong gradients with Roussos-Maragos algorithm, while Potrace software tends to smooth too much the image. Finally we also applied the Hqx magnification filter proposed by Stepin [23] that provides interesting zoom results but presents some artefacts (see for instance the X center of Hq4x) and limited scale factor (i.e. 4). Note that other comparisons can easily be done with the following online demonstrator: `https://ipolcore.ipol.im/demo/clientApp/demo.html?id=280` and source code is available on a *GitHub* repository: `https://github.com/kerautret/GTVimageVect`.

Depixelizing [13]   vector magic [10]

Roussos-Maragos [8]   Potrace [21]

approx. time: $< 0.5\,s$ approx. time: $2\,s$

approx. time: $< 0.5\,s$ approx. time: $2\,s$

total time : $503\,ms$ approx. time: $5\,s$

$Hq4x$ [23] $Hq4x(Hq4x)$ $Hq4x$ [23] $Hq4x(Hq4x)$

total time : $158\,ms$ total time: $152\,ms$ total time : $142\,ms$ total time: $320\,ms$
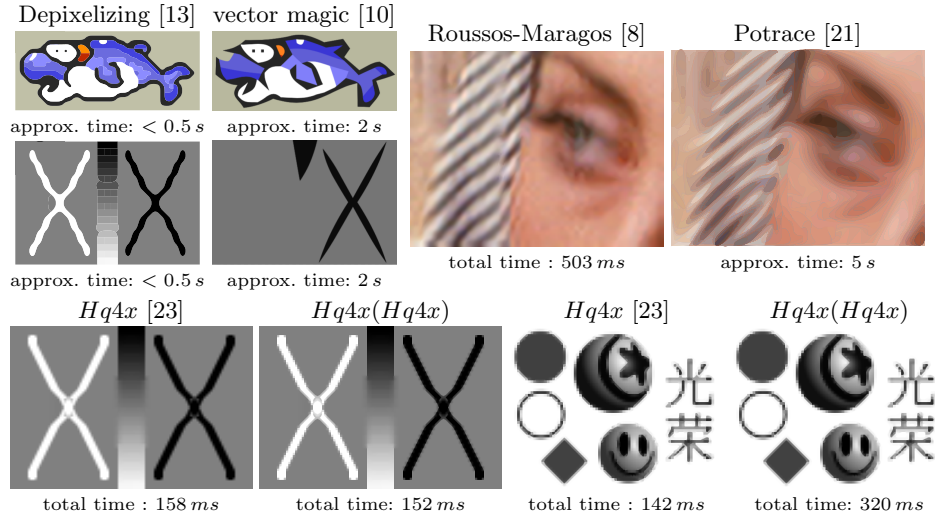
Fig. 5. Other complementary comparisons on five other approaches. The time measures were obtained on a 2.9 GHz *Intel Core i7* for all experiments expect Roussos-Maragos obtained on IPOL server. We use the default parameters expect for Potrace: we select 512 passes with color option.

6   Conclusion

We have presented an original approach to the problem of image vectorization and image super-resolution. It is based on a combinatorial variational model, which introduces geometry into the classical total variation model. We have compared our method both to state-of-the-art vectorization and super-resolution methods and it behaves well both for camera pictures and pixel art images. We have also provided an online demonstrator, which allows users to reproduce results or test our method with new images. In future works, we plan to compare quantitatively our method with state-of-the-art techniques, and also use our approach to train a CNN for zooming into pixel art images.

References

1. Partha Bhowmick and Bhargab B. Bhattacharya. Fast Polygonal Approximation of Digital Curves Using Relaxed Straightness Properties. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1590–1602, 2007.
2. Alexander I Bobenko and Boris A Springborn. A discrete laplace–beltrami operator for simplicial surfaces. *Discrete & Computational Geometry*, 38(4):740–756, 2007.
3. Laurent Condat. Discrete total variation: New definition and minimization. *SIAM Journal on Imaging Sciences*, 10(3):1258–1290, 2017.
4. Laurent Demaret, Nira Dyn, and Armin Iske. Image compression by linear splines over adaptive triangulations. *Signal Processing*, 86(7):1604–1616, 2006.

5. Fabien Feschet. Multiscale analysis from 1d parametric geometric decomposition of shapes. In IEEE, editor, *ICPR*, pages 2102–2105, 2010.
6. Pascal Getreuer. Contour Stencils: Total Variation along Curves for Adaptive Image Interpolation. *SIAM J. on Imaging Sciences*, 4(3):954–979, January 2011.
7. Pascal Getreuer. Image Interpolation with Geometric Contour Stencils. *Image Processing On Line*, 1:98–116, September 2011.
8. Pascal Getreuer. Roussos-Maragos Tensor-Driven Diffusion for Image Interpolation. *Image Processing On Line*, 1:178–186, September 2011.
9. Xavier Hilaire and Karl Tombre. Robust and accurate vectorization of line drawings. *IEEE TPAMI*, 28(6):890–904, 2006.
10. Vector Magic Inc. Vector magic, 2010. `http://vectormagic.com`.
11. B. Kerautret, J.-O. Lachaud, and B. Naegel. Curvature based corner detector for discrete, noisy and multi-scale contours. *IJSM*, 14(2):127–145, December 2008.
12. Bertrand Kerautret, Phuc Ngo, Yukiko Kenmochi, and Antoine Vacavant. Greyscale image vectorization from geometric digital contour representations. In *DGCI 2017*, pages 319–331, 2017.
13. Johannes Kopf and Dani Lischinski. Depixelizing Pixel Art. In *ACM SIGGRAPH 2011 Papers*, pages 99:1–99:8, 2011. Vancouver, British Columbia, Canada.
14. Gregory Lecot and Bruno Levy. Ardeco: automatic region detection and conversion. In *17th Eurographics Symposium on Rendering-EGSR'06*, pages 349–360, 2006.
15. Hairong Liu, Longin Jan Latecki, and Wenyu Liu. A Unified Curvature Definition for Regular, Polygonal, and Digital Planar Curves. *Int. J. Comput. Vision*, 80(1):104–124, 2008.
16. Majed Marji and Pepe Siy. Polygonal representation of digital planar curves through dominant point detection – a nonparametric algorithm. *Pattern Recognition*, 37(11):2113–2130, 2004.
17. Thanh Phuong Nguyen and Isabelle Debled-Rennesson. A discrete geometry approach for dominant point detection. *Pattern Recognition*, 44(1):32–44, 2011.
18. Brian Price and William Barrett. Object-based vectorization for interactive image editing. *The Visual Computer*, 22(9-11):661–670, September 2006.
19. Anastasios Roussos and Petros Maragos. Vector-valued image interpolation by an anisotropic diffusion-projection PDE. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 104–115. Springer, 2007.
20. Leonid I. Rudin, Stanley Osher, and Emad Fatemi. *Nonlinear total variation based noise removal algorithms*. 1992.
21. Peter Selinger. Potrace: `http://potrace.sourceforge.net`, 2001-2017.
22. Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. In *CVPR*, pages 1874–1883, Las Vegas, NV, USA, June 2016. IEEE.
23. M. Stepin. Hqx magnification filter, 2003. `http://web.archive.org/web/20070717064839/www.hiend3d.com/hq4x.html`.
24. Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. In *Trans. on Graphics*, volume 26, page 11. ACM, 2007.
25. S. Swaminarayan and L. Prasad. Rapid Automated Polygonal Image Decomposition. In *35th Workshop AIPR'06*, pages 28–28, October 2006.
26. Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. In *Trans. on Graphics*, volume 28, page 115. ACM, 2009.