Adaptive image vectorisation and brushing using mesh colours

Hettinga, Gerben; Echevarria, Jose; Kosinka, Jiri

Published in:
Computers & Graphics

DOI:
10.1016/j.cag.2022.05.004

Publication date:
2022

Computers & Graphics 105 (2022) 119–130

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on STAG 2021

Adaptive image vectorisation and brushing using mesh colours

Gerben J. Hettinga [a], Jose Echevarria [b], Jiří Kosinka [a,*]

[a] Bernoulli Institute, University of Groningen, The Netherlands
[b] Adobe Research, San Jose, CA, USA

ARTICLE INFO

Article history:
Received 11 February 2022
Received in revised form 8 April 2022
Accepted 2 May 2022
Available online 13 May 2022

Keywords:
Image vectorisation
Vector graphics
Mesh colours

ABSTRACT

We propose the use of curved triangles and mesh colours as a vector primitive for image vectorisation. We show that our representation has clear benefits for rendering performance, texture detail, as well as further editing of the resulting vector images. The proposed method focuses on efficiency, but it still leads to results that compare favourably with those from previous work. We show results over a variety of input images ranging from photos, drawings, paintings, all the way to designs and cartoons. We implemented several editing workflows facilitated by our representation: interactive user-guided vectorisation, and novel raster-style feature-aware brushing capabilities.

1. Introduction

Vector images define an image as a collection of geometric primitives, such as lines, ellipses, or more elaborate shapes. Vector graphics are key in many disciplines, such as graphics and web design, or textile and printing industries, due to their ability to display an image at arbitrary resolutions without loss of quality. Image vectorisation is the process of converting a bitmap (raster) image into a vector image. The manual vectorisation of a raster image is a slow process, especially for highly detailed input such as photographs, requiring expertise and immense amounts of time [1].

Many approaches to automatic image vectorisation have been proposed over the years, using various primitives. Nevertheless, they have not been widely adopted due to their performance, quality or controllability issues, although some commercial tools like Adobe's *Live Trace* [2] have become an artistic style of its own despite its limitations. Creating realistic vector graphics thus remains challenging.

In our recent paper [3], we proposed a new image vectorisation method that excels at processing time, detail control, rendering efficiency, and the resulting vector images are well suited for further editing. Our method performs well over a wide variety of inputs, specially natural images and stylised design graphics. It follows a pipeline of three main steps: image feature extraction, 2D mesh generation, and colour fitting/texture transfer; see Fig. 1. Each step has been designed with performance, quality, and control in mind, leveraging recent advances in texture representation and mesh generation. The resulting vectorised images can be rendered in real-time on a wide range of hardware.

Vector images are resolution-independent, which has compression benefits with respect to high resolution raster images. But most previous vector image representations are not editable as intuitively as raster images. For example, raster brushes allow direct editing of pixel values, but most existing 2D vector representations do not support analogous operations, especially for soft brushes. Our vectorisation pipeline effectively turns a raster image into a representation that supports hybrid workflows where users can edit colours through brushing, still retaining all the advantages of vector images.

Our original contributions in [3] included the introduction of mesh colours as an image vectorisation primitive, texture details and soft feature extraction from the input image, and a fully automatic and user-guided image vectorisation pipeline.

The present paper adds the following contributions:

- User-controllable adaptive mesh colour resolution. This allows users to choose arbitrary target error thresholds, so the method can automatically determine the resolution of the mesh colour patches.
- Improved feature extraction. We filter noisy edges by applying progressive levels of blur to the image and keeping only those edges that have correspondence in higher levels. So edge extraction parameters can be lowered to capture salient edges without worrying about introducing unwanted noise.
- Extended editing tools. This include novel raster-style brushing capabilities for our vector representation, which push vector graphics in new directions beyond typical workflows.

The remainder of the paper is organised as follows. We first present an overview of related work in vector graphics and image vectorisation (Section 2). This is followed by an overview of

* Corresponding author.
E-mail address: j.kosinka@rug.nl (J. Kosinka).

https://doi.org/10.1016/j.cag.2022.05.004

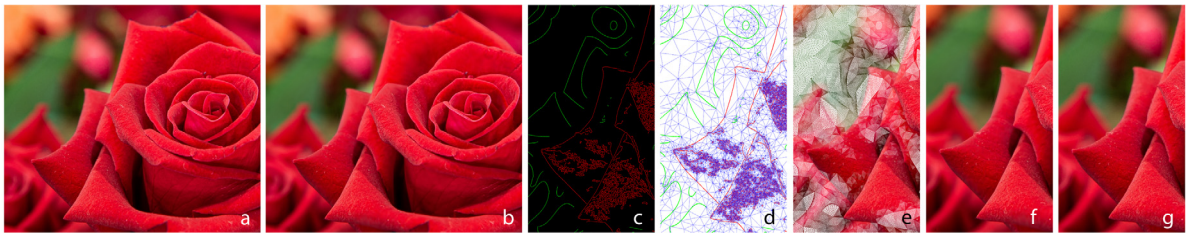

Fig. 1. An example input raster image (a) and our vectorised result (b). Please, zoom in to see the full details. The insets show the hard (red) and soft (green) image features (c) extracted in the first step of our method. These features are then used to build a curved triangular 2D mesh (d), where each triangle is later equipped with mesh colours (e) optimised from the input image. Our representation can be rendered efficiently in real-time (f), producing accurate renditions of the original image (g). Our vector images automatically adapt to varying levels of detail within the same image (e.g. petal texture versus blurry background) as seen in (c)–(e). They also capture both the infinite sharpness around salient image discontinuities and the smooth colour interpolation typical of vector graphics.
Source: Photo ©Jack Tamrong - stock.adobe.com.

our vectorisation method (Section 3) and a detailed description of its stages: feature extraction (Section 4), mesh generation (Section 5), and texture transfer (Section 6), including its new adaptive image detail control. To demonstrate the capabilities of our proposal, we show the results of applying our method to several types of raster images, edits to the resulting vector images, our improved user-guided vectorisation pipeline as well as our novel brushing functionality and compare with previous works (Section 7). Finally, we discuss our method before concluding the paper (Section 8).

2. Related work

Solid colours & linear gradients. First automatic attempts at vectorising images partition them into regions representable by flat, linear or quadratic gradients [4]. This produces stylised representations of the original image. Vectorisation of natural images based on solid colours often requires a colour quantisation step to simplify detail [5,6], affecting the detail preservation of this approach. Additionally, user guidance for this quantisation is often needed to preserve boundaries between objects [7–9]. Another early attempt relies on adaptively created image triangulations [10].

Diffusion curves. Diffusion curves [11] represent an image by a collection of curves defining sharp transitions in colour. The colours assigned to either side of these curves are then diffused over the rest of the image. When vectorising an image into the diffusion curve representation, image edges are detected and represented as smooth curves, and their colour is extracted from the underlying image. The original diffusion curves have been extended in many ways [12–15] to allow better user control and increase the fidelity of the primitive itself, and thus also in the vectorisation process. Although (generalised) diffusion curves offer a powerful set of primitives, they tend to be expensive to evaluate as this involves solving large linear systems [11], using smart solvers [16,17] or even using raytracing [18,19]. Although diffusion curves provide an excellent way to represent images, they have not seen wide adoption due to the complex nature of the solvers [20].

Parametric patches. Originally introduced in Adobe Illustrator [2], the gradient mesh primitive represents an image as a regular grid of bicubic patches [21,22], which are the result of interpolation of the colours and colour gradients assigned to the vertices of the (gradient) mesh. Early attempts at gradient mesh based vectorisation have used it in combination with (pre-)segmented regions with progressive subdivision of patches [23] or optimising meshes [24]. An automatic pipeline based on quadrangulations guided by frame fields was proposed by Wei et al. [25].

More recently, subdivision surfaces were used over triangular meshes [26,27], created using an elaborate pipeline of feature extraction, mesh generation as well as colour fitting. However, despite recent advances [28], subdivision surfaces are relatively costly to evaluate and special care is needed to ensure interpolation of the colours assigned to mesh vertices [29]. Gradient meshes and subdivision surfaces can model smooth regions of an image with high accuracy, but require many patches in regions with high-frequency changes in colour. Recently, other non-standard forms of the gradient mesh have been proposed to alleviate this problem [25,30–33], including our approach [3], extended in the present paper.

Thin-plate splines have been used in combination with cubic Bézier triangles [34]. The Bézier triangle mesh is generated by simplifying a pixel-dense triangle mesh, which is followed by optimisation and fitting. Chen et al. [35] combine dense thin-plate splines, efficiently rendered with a specialised kernel, with coarser gradient meshes generated from a manual segmentation of the image. Although most texture details are preserved, sharp features are only approximated when not preserved by the segmentation.

Our original method [3] also uses Bézier triangles for the mesh, but generated directly from detected image features. In contrast to prior art, we represent the colour inside each Bézier triangle as mesh colours [36], and further improve on this using a user-controllable adaptive approach. This allows for detailed texture preservation and provides a cheap, yet accurate representation of the original image. We note that recent work on new primitives for colour manipulation [37] presents similar triangular subdivisions, but the parametric shape of their colour distributions does not capture spatial texture detail.

Other vectorisations. A related class of vectorisation methods that focus on abstract/stylised imagery includes *pixel art* [38,39], where aliased raster edges can be shape, texture or shading, making that inference the core of the problem. Other recent works focus on perceptually-motivated vectorisations [40,41]. In a similar vein, vectorisation of drawings and sketches [42–45] focuses on inferring vector lines. Images can also be vectorised (and potentially edited) using their skeletons/medial axis transforms [46,47].

Vector image editing. Ultimately, the resulting vector images should allow their geometric and colour properties to be edited. Most parametric patch-based representations are easily deformed by deforming the handles of the parametric representations such as the vertices of a mesh [26] or the control points of a high-degree patch [27]. Other representations, such as diffusion curves, can be manipulated by changing the curves themselves, but it can be tedious to manipulate the image curve by curve. Local and global deformation can be achieved by discretising the image domain of diffusion curve images [48]. Our approach supports such typical geometric edits at different levels, from independent editing of vertex location and edge control points, to warping of subsets of the mesh [3].

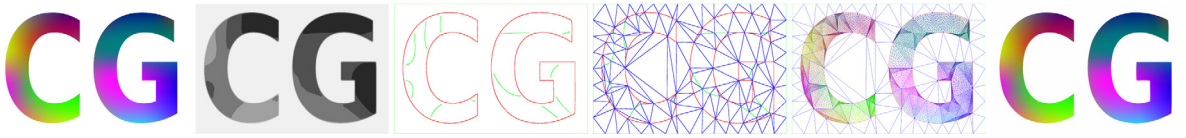

Fig. 2. Our vectorisation pipeline presented on an illustrative example. From left to right: The input raster image, banded greyscale image to extract soft edges from, extracted hard (red) and soft (green) image features, generated mesh, mesh colours with colours fitted, and the final (rasterised) result.

Many representations allow editing of colour through changing the colour distributions of a local region [25–27,35], by changing the colour and maintaining the variance. Liao et al. [26] allow direct colour editing of colour values or their neighbourhoods. However, this is dependent on the local density of the mesh. Diffusion curves can be colour edited at the curves but do not allow for arbitrary brushing. Similar representations such as Poisson vector graphics [49] and thin-plate spline based representations [50] only allow editing directly at features or by specifying entirely new features. Our proposed usage of mesh colours is more flexible than parametric solutions. Coupled with adaptive mesh resolution and hard image features, we showcase novel raster-style brushing capabilities, including soft brushes. This is different from vector brushes in some commercial drawing applications [51], where the final vector image is a collection of hard brush strokes.

3. Overview

Our method consumes a raster image (of any content) and automatically converts it into a vector image. Although the vector image needs to accurately represent the input raster content, the resulting vector representation should meet several other conditions: be editable, render efficiently, and be sparse. To that end, we extract image features that capture representative geometry, shading, and texture. In contrast to [3], we now determine image edges using a Gaussian scale-space [11], which improves control over noise filtering and keeping important edges.

Once vectorised using spline curves, the edges turn into intuitive handles for high level edits. These image features are also the constraints for our mesh generation step, where we seek a curved triangular mesh that follows them faithfully. We strive for: an efficient mesh generation, enough triangles to obtain a good topology to support detailed mesh colour patches, and a representation that is easily editable by the user.

Next, each triangle is equipped with a mesh colour patch. Although we do not intend to expose the mesh colours to the user, we allow the user to modify them using a new brushing technique not present in [3]. Our automatic and now user-controllable adaptive texture transfer process then fits the colours from the input pixels to the mesh colours of the patches. To display our vector representation, we rasterise it in real-time through tessellation shaders, whose level of detail can be controlled on the fly for detailed offline work or excellent performance for visualisation.

These steps of our pipeline are visually represented in Fig. 2:

- the main features of the image are extracted (Section 4);
- a conforming mesh is generated (Section 5);
- image colours are transferred to mesh colours (Section 6).

Some of these intermediate steps for a more complex input are shown in Fig. 1.

4. Feature extraction

The key to a successful image vectorisation is to establish which features to preserve from the input raster image. Given that we aim for a universal method applicable to any input image, we cannot make any assumptions about their content. However, we define two types of features: *hard* and *soft* edges. A similar distinction between hard and soft edges was made earlier on by Lindeberg [52] and Elder [53] who describe differing blur scales to edges which are extracted using a scale-space approach. Their approach was later used in a vectorisation setting by Orzan et al. [11] to estimate blur scales for edges.

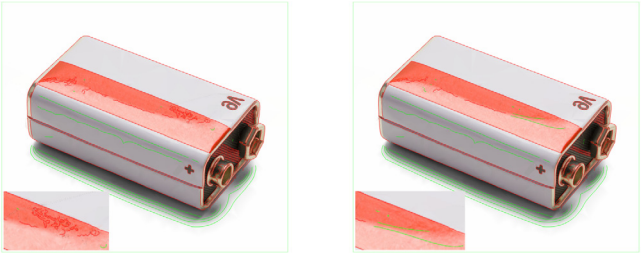

Fig. 3. Filtering edges with a scale-space approach removes edges that are the result of noise. Note the difference in the red region on the battery, where hard edges (red) on the left where removed, allowing for soft edges (green) to replace them for cleaner geometry that still preserves all the texture detail.

Hard edges come mainly from colour discontinuities that typically capture salient shapes, contours, and textures of the elements in the image; and they should remain sharp on the vector image. On the other hand, we use soft edges to model smooth but complex colour transitions (e.g. shading). Edge detection is still an active topic after decades of research [54], with recent neural approaches that do an increasingly good job at inferring geometrical edges at object level [55–57]. However, these methods are not that well suited to surfacing progressive texture detail or complex colour transitions.

Edge extraction. For simplicity and ease of control, and similarly to previous works [34,35], we use the Canny edge detector [58]. Given the performance of the rest of our method, this choice provides interactive and intuitive control over the level of detail of the resulting vectorisation. We typically set the low and high thresholds to 15 and 100 in the range $[0, 255]$, respectively. Compared to [3], where a standard single blur pass was applied, we filter further noisy edges by applying progressive levels of blur to the image and keeping only those edges that have correspondence in higher levels. Only those edges are kept that have at least one pixel in the user defined scale. Fig. 3 shows an example where the edges that are the result of noise are filtered away. While this approach is good at detecting hard edges and contours, it fails to pick up soft edges: lower thresholds lead to too much noise and/or unwanted texture detail. Thus, we propose a simple procedure to extract soft edges to complement the hard ones found by Canny.

Upon closer inspection of the vectorisations obtained using only hard edges, we noticed the missing soft edges we were interested in are typically orthogonal to the smooth colour gradients in the image. To expose those features, we quantise a greyscale version of the input image (20 levels by default), and extract the discontinuities from the resulting banding using the marching

Fig. 4. Top row: Original image and the quantised greyscale image with the extracted hard (red) and soft (green) edges overlaid. Bottom row: Our vectorised version without using soft edges (left) and with soft edges (right). Note that soft edges help to capture more detail and to avoid artefacts on the reflections of the statue and in the background. Please, zoom in for a more detailed view.

squares algorithm. This is similar to the iso-contours used to vectorise brushstrokes in [59], but our soft edges go through extra processing before being traced differently.

Edge filtering. Because hard and soft edges may overlap near areas where the image gradients change quickly, we filter the soft edges based on their distance to neighbouring hard edges, effectively removing them from areas where hard edges were already present. We do this by creating the distance transform of the previously extracted hard edges and using this as a filter for the extracted edges from the banded image. This filtering promotes the creation of sparser geometry later on in Section 5, as it removes bands that overlap with hard edges or are closely parallel to them. Fig. 4 shows the result of quantising the greyscale image, and the subsequent soft features extracted from them (top right). These features help us better capture soft image details, such as all the blurry background elements in the input image (bottom row).

5. Mesh generation

The goal of the mesh generation step is to create a mesh of cubic Bézier triangles that conform to the detected image features. We vectorise the detected edges by converting them to cubic Bézier splines, which are then used to drive the curved triangulation step.

Edge vectorisation. The extracted hard edges are traced and linked into pixel chains [60] to be vectorised as cubic Bézier splines. These splines help us capture curves and remove the aliasing present on the hard edges, and enforce C^0 or G^1 continuity as needed. In the spirit of [61], we progressively fit the splines to the chains by recursively fitting curves to each whole pixel chain.

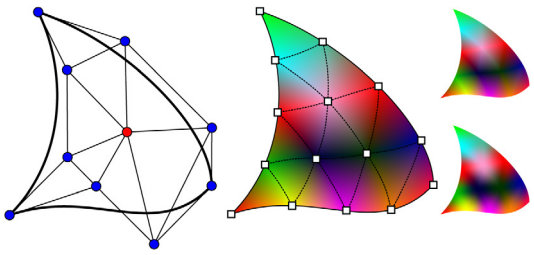

Fig. 5. Left: A cubic Bézier triangle with control points (the central one is in red). Middle: Resolution 3 mesh colour texture with $(4+1)\cdot(4+2)/2=15$ mesh colours mapped on the cubic Bézier triangle. Right: Linear (top) and quartic (bottom) interpolation.

We keep each fit only if the maximum error distance between a polyline approximation of the Bézier curve and the pixel chain is half a pixel. If not, the pixel chain is split at the pixel with the largest distance, and new splines are fitted to the respective halves of the previous pixel chain, until every segment of the pixel chain is converted.

Soft edges are usually noisier and do not represent salient image features that need to be preserved as accurately as the hard edges. Therefore, we found an approximation is sufficient and we do not have to strictly enforce the start and end points of the spline piece to lie over the actual pixel-chain. Fig. 4 (top right) shows the vectorised edges obtained from the input image. As can be seen, the hard edges closely follow the discontinuous features in the image, whereas the soft edges offer an approximation of the bands from the quantisation, without affecting the quality of the reconstruction. Hard and soft edges are kept separate to be handled differently in later stages of the pipeline.

Curved triangulation. The soft and hard vectorised edges from the previous step are curved, so direct application of standard linear meshing techniques would cause degeneracies if supporting straight line segments crossed through the curved edges. One option could be to subdivide the curved segments until only accurate-enough linear elements remain, but this would quickly increase the number of faces in the subsequent meshing step. Instead, we choose to create a curved triangular mesh, as this allows us to keep a lower number of generated triangles while directly incorporating the curved edges.

To generate such a mesh we use the method of Mandad and Campen [62] based on guarding triangles to avoid intersections and employ standard constrained Delaunay triangulation. We speed up the mesh optimisation phase by inserting supporting vertices into the triangulation at regular intervals and only when the inserted position is some distance from the nearest feature. Alternatively, other curved triangulation methods [63] can be used to generate the geometry, but the density of the resulting mesh may end up being too high [64]. After the triangulation step, we are left with a non-degenerate curved triangulation. This fixes the topology of the mesh. Next, we determine its geometry and parametrisation.

On each (curved) triangle, we construct a cubic Bézier triangle (see Fig. 5, left) by using the control points of the curved edges as edge control points and converting straight supporting segments to cubic Bézier curves, which determines all the blue control points. This ensures that all vectorised (hard and soft) edges are exactly reproduced in the curved triangulation. To fix the parametrisation, we add the central control point (red in the figure) as the centroid of the edge control points of each triangle. We also keep track of which edges represent hard and soft features by tagging them. The straight segments generated by the triangulation step are always deemed to be soft.

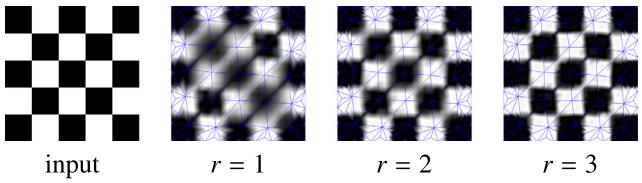

Fig. 6. From left to right: Input image and vectorisations using the same mesh but different resolution r of mesh colours. On purpose, the mesh does not capture the features of the input image correctly. Regardless, increasing r leads to increasingly better approximations of the input image.

6. Texture transfer

As mentioned before, our goal is to use vector primitives able to handle varying texture detail, while being fast to evaluate. We propose the use of *mesh colours* [36], a first in the context of image vectorisation, when not considering simple vertex colours and standard linear interpolation. Mesh colours [65] are a convenient way of storing colour and texture information in complex meshes. This makes it robust to further transformations and edits, especially when compared with parametric texture representations from previous vectorisation work. These properties make mesh colours a fitting mechanism for transferring texture information from a raster to a mesh. However, mesh colours have been used in conjunction with 3D painting tools [66], and there are no standard ways of transferring texture information to mesh colours.

Mesh colours. In our setup, we map triangular mesh colours to the cubic Bézier triangles that were generated in the previous step (Section 5). Fig. 5, middle, shows a schematic view of mapped mesh colours. The resolution r handles the number $R = \frac{(2^{r-1}+1)(2^{r-1}+2)}{2}$ of mesh colour vertices $t_{\mathbf{i}}$ per patch, where $\mathbf{i} = (i, j, k)$, $i+j+k = r$ and $i, j, k \geq 0$. We follow the procedure of [36] to interpolate mesh colour values over each patch. For clarity of presentation and to prepare the ground for our proposed mesh colour fitting scheme (Section 6.1), in the following we detail the steps for evaluating a mesh colour texture on a triangular patch.

The barycentric coordinates $\boldsymbol{\phi} = (u, v, w)$ with respect to a triangle $\triangle$ in the mesh are used to determine the three closest mesh colours: $t_{\mathbf{i}}$, $t_{\mathbf{j}}$, $t_{\mathbf{k}}$. These colours together determine a mesh colour (sub)triangle of $\triangle$. From the coordinates $\boldsymbol{\phi}$ we determine the local barycentric coordinates $\overline{\boldsymbol{\phi}}$ inside the mesh colour triangle. These local coordinates are subsequently used to either linearly or quartically interpolate between the three colour values of the mesh colour triangle. The former results in piece-wise linear C^0 colour interpolation over $\triangle$, and the latter in piecewise quartic C^1 colour interpolation at the expense of increased computational cost. We choose to use the C^0 version as there is not a lot of difference (Fig. 5, right column), except in cases where mesh colour values vary extremely. The resolution of each mesh colour texture can be changed per triangle [67], effectively adjusting the amount of texture detail that can be represented and the storage required for it. Fig. 6 shows a simple example where a static mesh approximates the same image with different mesh colour resolutions. Higher mesh colour resolutions are able to approximate the input more clearly.

6.1. Mesh colour fitting

At this stage, the mesh geometry is already fully defined and mesh colours can be fitted to them. Ideally, the colour fitting process should be implemented as a global least-squares problem over all mesh colours of the entire mesh. This would automatically generate smooth colour transitions over triangle boundaries that are marked as smooth. However, this leads to a large system of equations that needs to be solved and would not be practical both with regards to memory and performance. Our approach simplifies the problem, by fitting each triangle individually and only afterwards smoothing mesh colours where appropriate.

We first fit mesh colours to each cubic Bézier triangle T parametrised over $\triangle$ separately. We sample each T in the mesh uniformly to obtain the pairs $(p_i, \boldsymbol{\phi}_i)$ for every sample position p_i in the image with $\boldsymbol{\phi}_i$ its barycentric coordinates in $\triangle$, i.e., $T(\boldsymbol{\phi}_i) = p_i$. To effectively fit mesh colours, we need to ensure that the number of samples m satisfies $m > R$. Using the image position p_i of each evaluated pair, we look up the bilinearly interpolated colour value $I(p_i) = c_i$ in the input raster image I. Using $\boldsymbol{\phi}_i$, we determine the local barycentric coordinates $\overline{\boldsymbol{\phi}}_i$ of p_i in its mesh colour (sub)triangle. We then minimise the following function on a per-triangle basis, used once per colour channel:

$$\min \sum_i \left(T^c(\boldsymbol{\phi}_i) - c_i\right)^2,$$

where T^c evaluates the colour corresponding to T. This is a standard least squares problem that we solve for the mesh colours of T. The matrix of the system can be reused for fitting triangles with the same mesh colour resolution r, since it is independent of the actual image positions and it uses only parametric positions (expressed in terms of $\boldsymbol{\phi}$), which are generated uniformly for each triangle. We efficiently perform this sample pair creation process in parallel for each triangle using GPU compute shaders.

To increase the efficiency of this colour fitting step, we can use different mesh colour resolutions based on the size of each triangle with respect to the original image, as follows. We assume that during the feature extraction step most regions of highly varying colour are split into smaller ones, and that each resulting triangle after the mesh generation step lies in an area with small changes in colour. Therefore, the smaller the triangle, the lower resolution it needs to represent the textured area of the original image. We bin the triangles based on their pixel area in the original image into N different bins. These bins correspond to increasing resolution of mesh colours. This benefits performance as lower resolution textures require less samples to be generated, reducing CPU–GPU congestion and improving the speed of the fitting.

In this paper we introduce an improved strategy to adaptively find the best resolution of mesh colours to use for a patch, according to some user defined maximum error threshold E_{max}. We start fitting at resolution 0 and increase the resolution and only stop fitting when the mean squared error has gone below the threshold. Naturally, this process converges as the resolution of mesh colours will at some point exceed the resolution of pixels contained in the triangular area denoted by the current patch being fitted. Adaptively fitting patches takes more time than the previous simpler binning approach, as potentially multiple fits have to be done per-patch. However we can reuse the colours already sampled from the original image for each subsequent fit.

Offset sampling. We found that bilinear sampling does not fetch correct colours when sampling close to hard edges, causing colour bleeding artefacts to appear on the triangles (Fig. 7, top). Inspired by Liao et al. [11,26], we use a one-pixel padding around hard edges. When sampling at hard edges, the actual sampling is offset from the padded region. This ensures that sharp transitions on the raster image are preserved by sampling on the correct side of the edge (Fig. 7, bottom). By offsetting the sampling we are able to create crisp edges without affecting the smooth interpolation on either side of the edge.

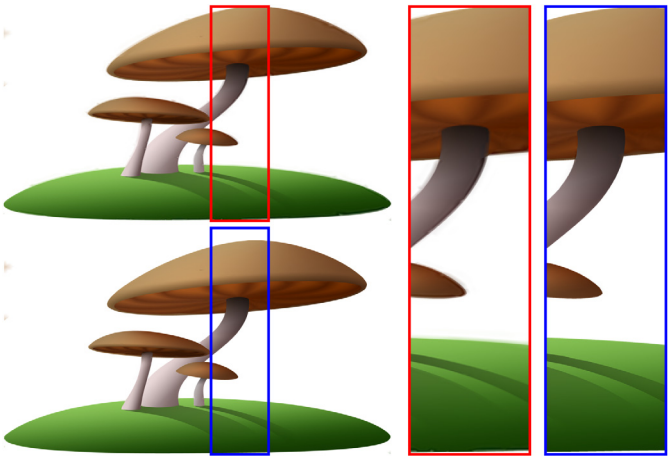

Fig. 7. Vectorised image without (top) and with (bottom) offset sampling around hard edges. Insets show the increased sharpness not only on silhouettes, but also hard edges coming from other sources like shading.

Colour smoothing. After each individual triangle in the mesh has been fitted with its associated mesh colours, we can increase the fidelity of our vectorisation by smoothing mesh colours that are on the edges of the triangles. The smoothing procedure, where mesh colour values on edges are averaged with respect to their values on adjacent mesh colour patches, only needs to be done for edges that were previously deemed to be smooth, i.e., supporting (smooth) edges that were created in the triangulation step (Section 5), or the smooth edges from the feature extraction stage (Section 4).

When faced with differences in resolution of adjacent triangles, the smoothing step cannot be achieved as a simple averaging step. In this case we choose to let the lower resolution triangle dictate the mesh colours on the edge of the higher resolution one. We achieve this by linearly interpolating the missing mesh colours so that the mesh colours of the two triangles align. The mesh colours that are already aligned can simply be copied. Averaging is used when the resolutions are equal.

This step guarantees at least C^0 interpolation of colour in smooth regions. Fig. 8 shows the difference between before and after colour smoothing. Before smoothing, the underlying triangulation is clearly visible in some regions. After smoothing, these artefacts vanish and the resulting vector image has C^0 colour interpolation everywhere except at hard features.

6.2. Rendering

The resulting vector graphics representation should lend itself to efficient rendering. To this end, we employ tessellation shaders in the modern graphics pipeline. All triangles are rendered as cubic Bézier triangles. We store the mesh colours in textures as proposed by [36] and evaluate them using the process outlined in Section 6.1 in a fragment shader. This of course leads to duplication of mesh colour data for edges and vertex positions, but it is necessary to model hard edges; this gives distinct colours on either side of the edge and no filtering should be applied there.

We have not focused our efforts on improved filtering of textures, such as trilinear filtering and MIP-mapping. There have been recent improvements and variants on the mesh colour technique that extend them with increased filtering capabilities and hardware support [66–68].

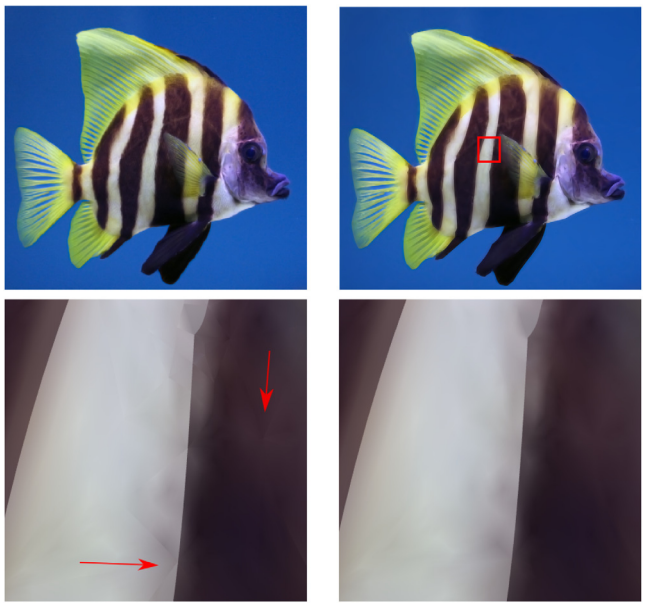

Fig. 8. Top: Input image (left) and our vectorised result (right). Bottom: Without smoothing (left) the seams of the mesh become apparent (some examples are highlighted by the red arrows), which is especially undesirable at large magnification factors typical for vector graphics. The proposed smoothing of the mesh colour values of neighbouring mesh colour patches removes such seams for a more natural and higher quality result (right).

7. Results

Fig. 1 (left) shows an intricate input image with lots of detail. Our extracted image features keep the sharpness of hard features, while the mesh colour patches capture fine texture as desired; see also the accompanying video. Fig. 2 shows a simple logo that turns into a relatively simple vector image that could be edited further to remove the background, for example. Fig. 4 shows an interesting combination of sharp and soft details, which our image features help capture accurately. Fig. 7 shows another example of clean stylised graphics faithfully captured by our representation.

In the following we focus on the improvements and extensions over the conference version [3] of this paper.

7.1. Adaptive image fitting

Adaptively fitting mesh colours allows for the mesh colours to be used in areas where they are most needed. In Fig. 9 we show different mesh colours fittings of the same geometry, features, and mesh, but with various maximum permissible mean squared error (MSE) per-patch thresholds. Naturally, by decreasing the threshold the input image is reconstructed better. Most of the small errors with respect to the original image accumulate along the edges. This can be attributed to the offset sampling we do along hard edges (Section 6) to hallucinate the infinite sharpness of vector graphics, and the fact that the vectorised edges might not perfectly align with the actual edges in the image.

The adaptivity of our solution means that lowering the error threshold does not translate in a resolution increase in every patch, since some image regions can be already well approximated by patches of lower resolution. In Fig. 10 we show the distribution of patch resolutions when increasing the maximum error threshold $E_{\max}$ for several examples. We can see that for lower thresholds the higher resolution mesh colour patches dominate, whilst increasing the threshold allows some patches to become lower resolution, with an expected gradual increase in the error.

Table 1
The performance of our vectorisation pipeline on several of the results featured in this paper. The time measurements are shown in seconds except for the rendering time, which is shown in milliseconds, and are split over the elements of our vectorisation pipeline: FE = Feature Extraction, MG = Mesh Generation, CF = Colour Fitting, RT = Rendering Time, CR = Compression Ratio. △ indicates the number of triangles and E_m stands for $E_{\max}$.

Image	Resolution	$\triangle \cdot 10^3$	FE	MG	CF	Total	RT	CR
Fig. 1	1280 × 853	∼ 9	1.0	0.2	2.5	3.7	∼ 22	0.28
Fig. 2	924 × 510	∼ 1	0.24	0.1	.4	1.1	< 1	0.03
Fig. 4 bottom left	848 × 1280	∼ 50	0.5	0.7	4	5.3	∼ 25	0.78
Fig. 7	1280 × 1181	∼ 4	1.9	0.06	1.9	3.9	∼ 2	0.06
Fig. 8	441 × 441	∼ 6	0.2	0.1	4.4	4.7	∼ 3	1.7
Fig. 9 top	1920 × 1284	∼ 15	1.5	0.3	1.5	3.3	∼ 25	0.18
Fig. 9 bottom	1280 × 853	∼ 15	1.5	0.3	13.3	15.2	∼ 30	0.65
Fig. 10 top ($E_m = 50$)	1280 × 853	∼ 8	1.1	0.3	2.8	4.3	∼ 26	0.27
Fig. 10 top ($E_m = 1$)	1280 × 853	∼ 8	1.5	0.3	15.3	17.0	∼ 26	0.93
Fig. 10 mid ($E_m = 50$)	1280 × 853	∼ 6	0.8	0.1	2.0	2.9	∼ 16	0.12
Fig. 10 mid ($E_m = 1$)	1280 × 853	∼ 6	0.8	0.1	4.7	5.7	∼ 17	0.33
Fig. 10 bottom ($E_m = 50$)	1280 × 853	∼ 41	1.2	0.7	20.3	22.6	∼ 30	1.83
Fig. 10 bottom ($E_m = 1$)	1280 × 853	∼ 33	1.2	0.5	74.9	76.9	∼ 30	4.46

7.2. Performance

Table 1 shows the performance of our vectorisation method for several results featured in this paper. In addition to the total timings, we also show timings of several of the intermediate steps, and rendering times of the rasterisation of the obtained vector images. We ran the method on a low-end laptop, with an NVIDIA MX150 GPU, 8 GB of RAM, and an Intel i5-8250 CPU; obtaining times that make our method practical.

The original method [3] could vectorise any image with reasonable performance, taking just a few seconds for the whole process. Although our new adapted method is geared more towards quality rather than performance, it is able to vectorise most simple images within seconds and more complex images within tens of seconds. Naturally, when requiring lower error rates the performance decreases. This performance difference is due to the higher prevalence of high resolution mesh colour patches, as can be seen in Fig. 10. The higher rates of these patches also impact the compression ratio, as more mesh colours samples are needed for higher resolution patches.

The feature extraction step (Section 4) is dependent on the resolution of the input image, and the number of features in the image. The mesh generation step (Section 5) depends on the content of the image, and the number of extracted features and their orientations. For instance, curved features that lie close to each other will generate more triangles than just a single curve. Then the performance of the colour fitting step (Section 6) depends on the number of generated triangles. However, due to our parallelisation and our adaptive patch resolution, it can be achieved quite efficiently. The resulting vector images can be rendered quickly through our use of tessellation shaders. Even for images with a large number of triangles we achieve real-time performance rates (see the accompanying video).

7.3. Editing

Key to any image vectorisation method is the editability of the resulting representation. In the following, we show different types of edits and workflows, some only possible due to the intuitiveness of our structure and steps, and/or the efficiency of the whole pipeline.

Geometry editing. We support low-level deformations, i.e., dragging vertex positions and tangent handles of the cubic edges of the Bézier triangles. However, in some areas a relatively large number of elements can be generated and editing can become tedious. Instead, the mesh can be effectively manipulated using proportional editing tools [26] paired with handling of curves in the spirit of [69]. We show an example of this in Fig. 11 and the supplementary video. Because of our real-time rendering performance, users can efficiently zoom in and out for precise control.

User-guided vectorisation. We also created an interactive application that allows users to draw spline curves on top of a raster image. Replacing the automatic feature extraction stage of the pipeline, the user can mark them as hard or soft features. The created curved features are used as inputs to the rest of the pipeline. This user-guided process of vectorisation can be done fully interactively thanks to the efficiency of our pipeline. In addition, the user adds curves locally, which requires only local updates to the triangulation and mesh colours. Fig. 12 shows interactively generated vectorisations of raster images. Lastly, the same workflow could be used to clean up automatically vectorised images by adding or fixing features that were not captured correctly in the edge detection phase.

Colour editing. Due to the way mesh colours are connected to the mesh, we can replicate the expressiveness and editing capabilities of raster images by allowing the user to brush over the vector image directly, so the colours of the mesh colours are updated accordingly; see Fig. 13. Naturally, the higher the resolution of the mesh colours, the better the capacity to capture the brushstrokes, becoming akin to brushing on a raster image. After the initial vectorisation of a raster image, we allow the user to increase the resolution of the mesh colour patches arbitrarily. This is achieved without loss of quality given the new mesh colours can be created using linear interpolation from the initial mesh colours. This procedure can be used to increase the resolution locally or globally. Note that this would be harder for representations based on subdivision surfaces, as this requires global subdivision [28]; or thin plate splines, as this increases the complexity of computation (matrix-inversion). Our representation is only spatially affected as mesh colour evaluation is a constant operation.

The inclusion of the feature curve information in the vectorised representation also allows for interesting hybrid interactions (Fig. 14). Stroke colours are propagated from the centre of the brush outwards, visiting all triangle faces in the perimeter of the brush which are directly accessible from the centre of the brush, so hard features can be used as barriers to this propagation.

Free-form authoring workflows. By combining the interactive feature creation and brushing features we create novel free-form vector graphics workflows. Using the guided vectorisation process described before, the user first draws the main features of the design. Based on those features, the mesh generation step creates and initialises a blank mesh colour canvas to receive user brushstrokes. With the addition of feature-aware strokes this workflow becomes a novel and powerful tool to create vector images. A

Fig. 9. Top to bottom, colour fittings of the same image with different maximum MSE thresholds [500, 200, 100, 50, 10] resulting in mean squared errors [17.88, 11.84, 9.55, 7.43, 5.32]. Error maps scaled 20 times for visualisation purposes.

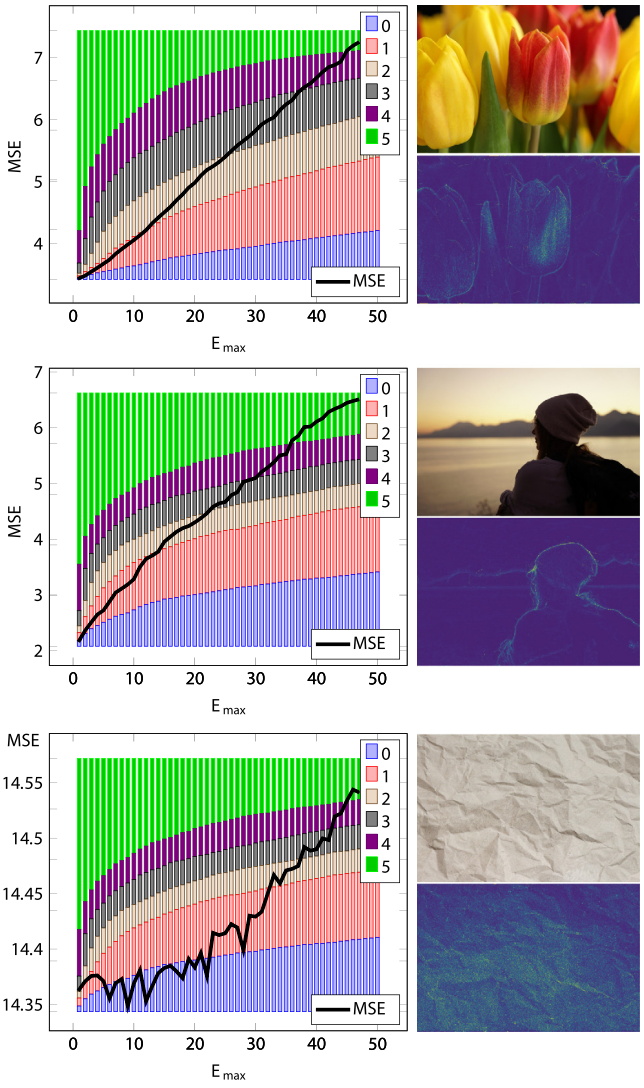

Fig. 10. Plots showing the distribution of patch resolutions from our proposed adaptive colour fitting, for the resulting vectorisations on the right. The maps show the error incurred for $E_{\max} = 1.0$, scaled 20 times for visualisation purposes.

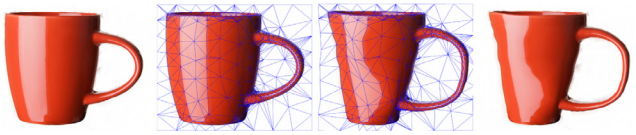

Fig. 11. Our vector images are easily editable by manipulating the curved mesh. From left to right: One of our vectorised results, the mesh of the vector, the edited mesh, and the rendered edited image. See the accompanying video for the editing session.

potential limitation comes from the fact that editing the initial features later on might update the underlying triangulation, thus changing the distribution of mesh colours unexpectedly. This could be alleviated by trying to approximate the mesh colours from the original triangulation, but this will most likely incur some loss of quality. Fig. 15 shows some examples created using the proposed free-form workflow, which would be impossible to author using previous representations.

7.4. Comparisons with previous work

We compare against recent methods and relevant primitives for image vectorisation: thin-plate splines (TPS), subdivision meshes, and diffusion curves. For all comparisons we must state that we could only approximately compare. This is because many algorithms only vectorise the images partially by first manually segmenting them and/or we were not able to find the original input images and were left with a lower quality input image. Still, we have to the best of our abilities tried to compare our method to existing methods.

Fig. 16 shows a comparison with a partially automatic TPS-based method [35]. As can be seen, their method is very good at capturing fine texture detail, but at the same time it scales similarly to a raster image, thus loosing some sharpness around hard edges. In addition, their vector patches often show seams

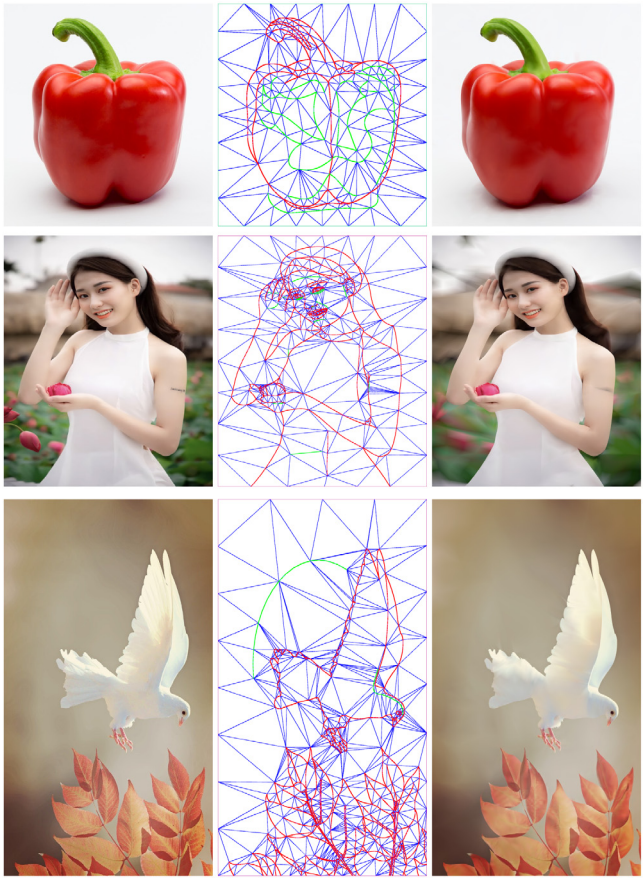

Fig. 12. User-guided vectorisations of several images. Left to right: Original image, user-placed curves in red (hard edges) and green (soft edges) with the generated mesh in blue, and the resulting vectorisation. Top: 47 curves generated 460 triangles. Middle: 155 curves generated 993 triangles. Bottom: 186 curves generated 1404 triangles. Please, see also the accompanying video.

Fig. 13. A vector brushing example. *Left*: Vector image obtained with our method. *Right*: The same vector image after a localised application of a green soft brush. While the result resembles the ones achievable in standard raster editing workflows, this one leverages our whole representation, with strokes optionally blocked by hard features (like the ones outlining the red parts of the shell), and remaining in vector space during the whole process.

under magnification. In contrast, our method hallucinates vector-style sharpness around hard edges and does not show texture seams thanks to our colour smoothing around soft edges. Our texture detail is affected both by the feature extraction step and the patch resolution, obtaining less realistic abstracted looks when not sufficient.

Because our image quality is often comparable to that of [35], we chose not to include gradient meshes [24] in our comparisons, as their limitations when capturing highly detailed textures were already demonstrated by Chen et al.

Fig. 18 shows a comparison against another TPS-based method [34], where our simpler and more performant pipeline achieves

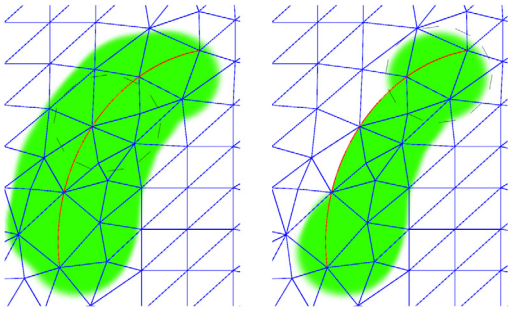

Fig. 14. Left: Brushing over our vector representation without awareness of hard features (red line) provides the same experience as when brushing over a raster image. Right: If we make the strokes aware of hard features, colour updates stop precisely at their boundaries. When the endpoint of a hard feature is in the perimeter of the brush (dashed black circle), the stroke naturally wraps around the feature. Triangle mesh shown in blue.

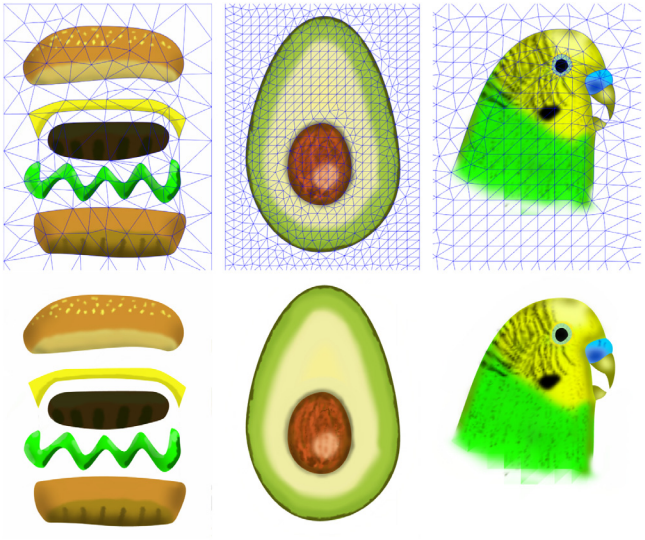

Fig. 15. Examples of vector images created using the proposed free-form workflow, by first drawing the outlines of the shapes, and using brushstrokes to paint solid colours and texture detail afterwards.

comparable results to their more intensive vectorisation method. Fig. 19 shows a comparison with another mesh-based approach [26] that uses subdivision surfaces. Our decoupling into a spatial 2D mesh and 3D (RGB) mesh colours allows finer control over colour whilst not complicating the geometry, capturing higher level of detail while achieving comparable smoothness and mesh density.

Fig. 17 shows comparisons with hierarchical diffusion curves [12]. We found that their image feature extraction translates into a global loss of clarity and detail for photos. For designed graphics, their method extracts cleaner features that produce quality closer to our method. The reported timings of the images in Fig. 17 are in the hundreds of seconds [12], our vectorisation pipeline is able to vectorise these same images at least one order of magnitude faster.

It is also worth mentioning that our precise tracing of hard edges following the contours of the objects allows straightforward cut-outs to remove image backgrounds.

7.5. Discussion

Previous vector graphics primitives have focused mostly on high colour continuity surfaces. Our vector representation is 'only'

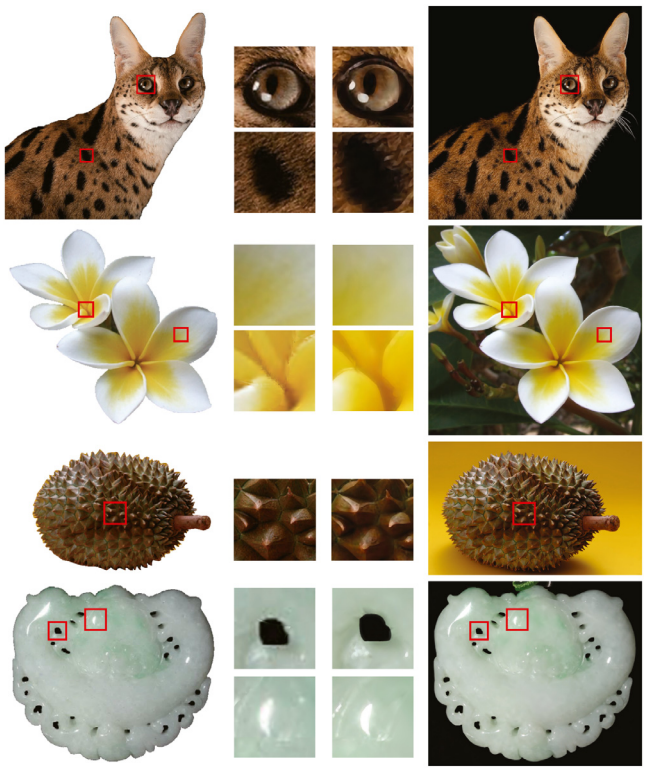

Fig. 16. Comparisons between [35] (left) and our proposed method (right). Our method seems better at hallucinating sharpness coming from geometric discontinuities (petals in the second row, holes in the last row), which are easily picked up by our extracted image features. However, ours is not that good at capturing extremely detailed textures like fur (first row). Insets also show the relevance of our colour smoothing across patches, absent in [35] (second and bottom rows). Vectorised backgrounds were not available from [35], but included for completeness.

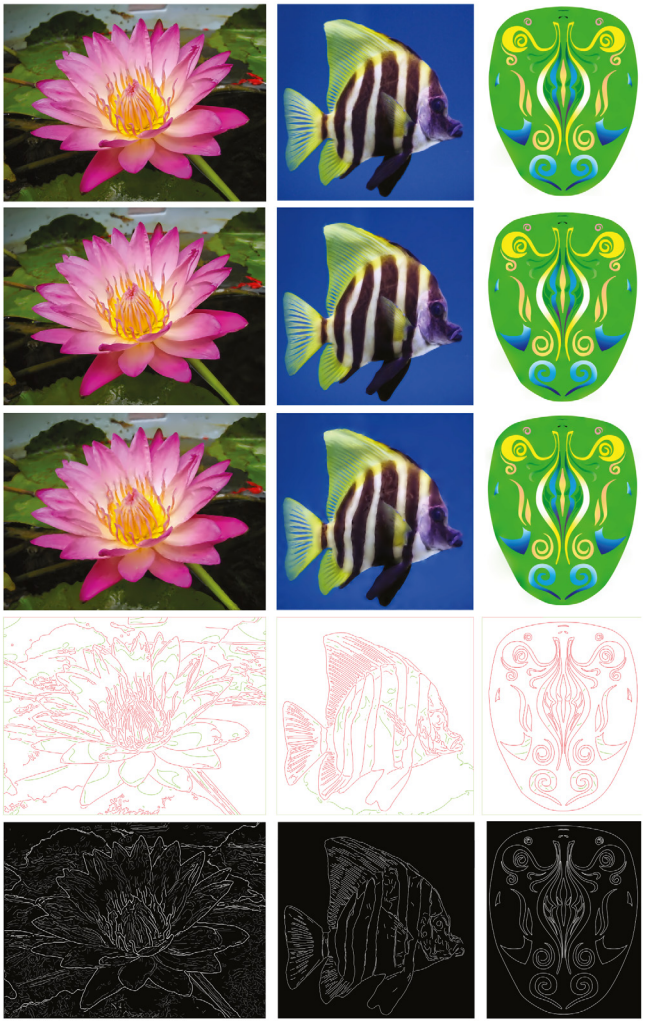

Fig. 17. From top to bottom: Input images, our results, results from [12], our extracted image features, and the ones from [12]. When applied to photos, our method produces sharper and cleaner results across the whole image (please, zoom in for details). For simpler inputs with clearer discontinuities, both methods perform similarly.

C^0, but given our proposed feature extraction step, the mesh is generated in a way that all triangles lie on areas with little variation in colour. These areas can then easily be handled by the mesh colour patches, removing the need for smoother interpolation strategies like subdivision surfaces or thin-plate splines, which are less efficient to evaluate than our representation.

Our pipeline uses standard and also state-of-the-art techniques for extracting features and constructing the geometry, but while the proposed steps in our method are on top of each other, the proposed algorithms for them are independent on each other. This means that the proposed pipeline can be potentially updated/upgraded in the future. We already did that to the image feature extraction and adaptive mesh colour resolution, with respect to the conference version of this paper [3]. In any case, we also proposed a variety of interactive tools that can be used if fully-automatic methods fail.

Although tessellation shaders increase the rendering performance, it is still possible to render the images using only the CPU, or alternatively using compute shaders or texture based solutions for the Web. In addition, a downgraded approximation of the vector image can be created easily by extracting each mesh colour patch as a piece-wise linear triangulation, using the evaluated mesh colour positions as vertices. This may help the compatibility with current vector format standards such as PDF [21] that do not directly support our proposed vector representation.

Theoretically, we could push our method to two extremes: on the one hand, the whole input image could be represented using only two triangles with a very high resolution of the mesh colours; on the other hand, each pixel of the input image could be captured by two triangles with a very low (in this case zero) texture resolution. Both are similar in spirit to [25], which is an efficient method, but it does not adapt its resolution to local regions with higher detail. In addition, not having explicit image features for geometric edits means the user would need to move each vertex manually, so having some sort of higher-level vectorised control features is more desirable. In summary, one needs to find a good balance between the number (and geometry) of the mesh triangles and the mesh colour resolution. We believe our pipeline achieves this balance to a very good extent, but additional optimisations are still possible, such as iterative feedback between different steps of our pipeline.

Our method does not readily support one pixel wide features. All hard edges are deemed to be ‘step‘ edges. The offset sampling thus misses one pixel wide features as the colours are not picked up by the fitting step. Single pixel wide features require separate treatment, such as in combination with some of the methods focusing on sketch vectorisation mentioned in Section 2.

8. Conclusion

We have presented a fully automatic image vectorisation method that can vectorise any image, ranging from natural

Fig. 18. From left to right: Input image, our result, result from [34], our image features and triangle mesh, and the ones from [34]. Final quality is very similar between both methods, including sharpness around hard edges due to their feature alignment, and our offset sampling and mesh colours. The main differences lie in the 2D mesh, with theirs being sparser. However, the complexity of their mesh optimisation and the evaluation of their thin-plate splines is higher than our simpler but more efficient approach. Note that the vectorised background was not available from [34], but we included it in our result for completeness.

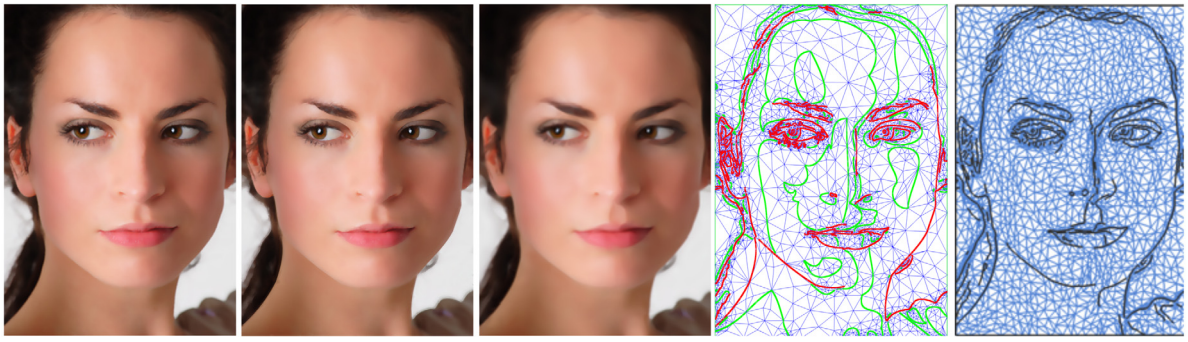

Fig. 19. From left to right: Input image, our result, result from [26], our image features and mesh, image features and mesh from [26]. While image features and triangle structure look similar, our mesh colour patches are able to capture more detail, even in smooth regions such as under the nose.

images to logos and cartoon images, with good performance and reconstruction accuracy. Through our novel use of mesh colours, we are able to adaptively transfer detailed colour textures to a mesh of curved triangles generated from our feature extraction step.

Our efficient pipeline is capable of vectorising a wide range of image types in seconds on commodity hardware. To ensure that not only hard edges but also smooth image regions are captured correctly, we add soft edges obtained from colour banding to our image features. These features are vectorised, and then respected by the mesh of curved triangles equipped with mesh colours representing the vectorised image. This results in relatively sparse vector representations that are flexible and easy to edit in a variety of ways, as demonstrated in the examples presented throughout the paper and the supplementary video.

CRediT authorship contribution statement

Gerben J. Hettinga: Conceptualisation, Methodology, Software, Investigation, Writing – original draft. **Jose Echevarria:** Conceptualisation, Methodology, Writing – original draft, Writing – review & editing. **Jiří Kosinka:** Conceptualisation, Methodology, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.05.004.

References

[1] Yang M, Chao H, Zhang C, Guo J, Yuan L, Sun J. Effective clipart image vectorization through direct optimization of bezigons. IEEE Trans Vis Comput Graphics 2016;22(02):1063–75.

[2] Adobe illustrator: Meshes. 2019, Online; https://helpx.adobe.com/illustrator/using/meshes.html. [Accessed 11 February 2022].

[3] Hettinga GJ, Echevarria J, Kosinka J. Efficient image vectorisation using mesh colours. In: Frosini P, Giorgi D, Melzi S, Rodolà E, editors. STAG: Smart tools and applications in graphics. The Eurographics Association; 2021.

[4] Lecot G, Levy B. Ardeco: automatic region detection and conversion. In: 17th Eurographics symposium on rendering. 2006, p. 349–60.

[5] Adobe illustrator: Image trace. 2019, Online; https://helpx.adobe.com/illustrator/using/image-trace.html. [Accessed 11 February 2022].

[6] Li T-M, Lukáč M, Gharbi M, Ragan-Kelley J. Differentiable vector graphics rasterization for editing and learning. ACM Trans Graph 2020;39(6).

[7] Xie J, Winnemöller H, Li W, Schiller S. Interactive vectorization. In: Proceedings of the 2017 CHI conference on human factors in computing systems. CHI '17, New York, NY, USA: Association for Computing Machinery; 2017, p. 6695–705.

[8] Richardt C, Lopez-Moreno J, Bousseau A, Agrawala M, Drettakis G. Vectorising bitmaps into semi-transparent gradient layers. Comput Graph Forum 2014;33(4):11–9.

[9] Favreau J-D, Lafarge F, Bousseau A. Photo2clipart: Image abstraction and vectorization using layered linear gradients. ACM Trans Graph 2017;36(6).

[10] Demaret L, Dyn N, Iske A. Image compression by linear splines over adaptive triangulations. Signal Process 2006;86(7):1604–16.

[11] Orzan A, Bousseau A, Winnemöller H, Barla P, Thollot J, Salesin D. Diffusion curves: A vector representation for smooth-shaded images. ACM Trans Graph 2008;35(2):71–9.

[12] Xie G, Sun X, Tong X, Nowrouzezahrai D. Hierarchical diffusion curves for accurate automatic image vectorization. ACM Trans Graph 2014;33(6):230.

[13] Lu S, Jiang W, Ding X, Kaplan CS, Jin X, Gao F, et al. Depth-aware image vectorization and editing. Vis Comput 2019;1–13.

[14] Dai W, Luo T, Shen J. Automatic image vectorization using superpixels and random walkers. In: Image and signal processing (CISP), 2013 6th international congress on, Vol. 2. IEEE; 2013, p. 922–6.

[15] Zhao S, Durand F, Zheng C. Inverse diffusion curves using shape optimization. IEEE Trans Vis Comput Graphics 2017.

[16] Jeschke S, Cline D, Wonka P. A GPU Laplacian solver for diffusion curves and Poisson image editing. Trans Graph (Siggraph Asia 2009) 2009;28(5):1–8.

[17] Jeschke S, Cline D, Wonka P. Estimating color and texture parameters for vector graphics. In: Computer graphics forum, Vol. 30, no. 2. Wiley Online Library; 2011, p. 523–32.
[18] Bowers JC, Leahey J, Wang R. A ray tracing approach to diffusion curves. In: Computer graphics forum, Vol. 30, no. 4. Wiley Online Library; 2011, p. 1345–52.
[19] Prévost R, Jarosz W, Sorkine-Hornung O. A vectorial framework for ray traced diffusion curves. Comput Graph Forum 2015;34.
[20] Boyé S, Barla P, Guennebaud G. A vectorial solver for free-form vector gradients. ACM Trans Graph 2012;31(6).
[21] Adobe PDF. 2006, Online; https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdf_reference_archive/pdf_reference_1-7.pdf. [Accessed 11 February 2022].
[22] Sun J, Liang L, Wen F, Shum H-Y. Image vectorization using optimized gradient meshes. ACM Trans Graph (TOG) 2007;26(3):11.
[23] Price B, Barrett W. Object-based vectorization for interactive image editing. Vis Comput 2006;22(9):661–70.
[24] Lai YK, Hu SM, Martin RR. Automatic and topology-preserving gradient mesh generation for image vectorization. ACM Trans Graph (TOG) 2009;28(3):85.
[25] Wei G, Zhou Y, Gao X, Ma Q, Xin S, He Y. Field-aligned quadrangulation for image vectorization. In: Computer graphics Forum, Vol. 38, no. 7. Wiley Online Library; 2019, p. 171–80.
[26] Liao Z, Hoppe H, Forsyth D, Yu Y. A subdivision-based representation for vector image editing. IEEE Trans Vis Comput Graphics 2012;18(11):1858–67.
[27] Zhou H, Zheng J, Wei L. Representing images using curvilinear feature driven subdivision surfaces. IEEE Trans Image Process 2014;23(8):3268–80.
[28] Zhou J, Hettinga G, Houwink S, Kosinka J. Feature-adaptive and hierarchical subdivision gradient meshes. Comput Graph Forum 2022;41:389–401.
[29] Verstraaten TW, Kosinka J. Local and hierarchical refinement for subdivision gradient meshes. Comput Graph Forum 2018;37(7):373–83.
[30] Lieng H, Kosinka J, Shen J, Dodgson NA. A colour interpolation scheme for topologically unrestricted gradient meshes. In: Computer graphics forum, vol. 36, no. 6. Wiley Online Library; 2017, p. 112–21.
[31] Li XY, Ju T, Hu SM. Cubic mean value coordinates. ACM Trans Graph 2013;32(4):126–:10.
[32] Barendrecht PJ, Luinstra M, Hogervorst J, Kosinka J. Locally refinable gradient meshes supporting branching and sharp colour transitions. Vis Comput 2018;34(6–8):949–60.
[33] Baksteen SD, Hettinga GJ, Echevarria J, Kosinka J. Mesh colours for gradient meshes. In: Frosini P, Giorgi D, Melzi S, Rodolà E, editors. STAG: Smart tools and applications in graphics. The Eurographics Association; 2021.
[34] Xia T, Liao B, Yu Y. Patch-based image vectorization with automatic curvilinear feature alignment. ACM Trans Graph (TOG) 2009;28(5):115.
[35] Chen K, Luo Y, Lai Y, Chen Y, Yao C, Chu H, et al. Image vectorization with real-time thin-plate spline. IEEE Trans Multimed 2020;22(1):15–29.
[36] Mallett I, Seiler L, Yuksel C. Patch textures: Hardware implementation of mesh colors. In: High-performance graphics. The Eurographics Association; 2019.
[37] Shugrina M, Kar A, Fidler S, Singh K. Nonlinear color triads for approximation, learning and direct manipulation of color distributions. ACM Trans Graph 2020;39(4).
[38] Kopf J, Lischinski D. Depixelizing pixel art. ACM Trans Graph 2011;30(4).
[39] Silva MAG, Montenegro A, Clua E, Vasconcelos C, Lage M. Real time pixel art remasterization on GPUs. In: 2013 XXVI Conference on graphics, patterns and images. 2013, p. 274–81.
[40] Hoshyari S, Dominici EA, Sheffer A, Carr N, Wang Z, Ceylan D, et al. Perception-driven semi-structured boundary vectorization. ACM Trans Graph 2018;37(4).
[41] Dominici EA, Schertler N, Griffin J, Hoshyari S, Sigal L, Sheffer A. PolyFit: Perception-aligned vectorization of raster clip-art via intermediate polygonal fitting. ACM Trans Graph 2020;39(4).
[42] Favreau JD, Lafarge F, Bousseau A. Fidelity vs. Simplicity: A global approach to line drawing vectorization. ACM Trans Graph 2016;35(4).
[43] Najgebauer P, Scherer R. Inertia-based fast vectorization of line drawings. Comput Graph Forum 2019;38(7):203–13.
[44] Egiazarian V, Voynov O, Artemov A, Volkhonskiy D, Safin A, Taktasheva M, et al. Deep vectorization of technical drawings. In: Vedaldi A, Bischof H, Brox T, Frahm JM, editors. Computer vision. Cham: Springer International Publishing; 2020, p. 582–98.
[45] Stanko T, Bessmeltsev M, Bommes D, Bousseau A. Integer-grid sketch simplification and vectorization. Comput Graph Forum 2020;39(5):149–61.
[46] Wang J, Kosinka J, Telea A. Spline-based medial axis transform representation of binary images. Comput Graph 2021;98:165–76.
[47] Wang J, Kosinka J, Telea A. Spline-based dense medial descriptors for lossy image compression. J Imaging 2021;7(8).
[48] Lu S, Ding X, Gao F, Chen J. Shape manipulation of diffusion curves images. IEEE Access 2020;8:57158–67.
[49] Hou F, Sun Q, Fang Z, Liu YJ, Hu SM, Qin H, et al. Poisson vector graphics (PVG). IEEE Trans Vis Comput Graphics 2018;26(2):1361–71.
[50] Finch M, Snyder J, Hoppe H. Freeform vector graphics with controlled thin-plate splines. ACM Trans Graph 2011;30(6):1–10.
[51] Adobe fresco: Vector brushes. 2020, Online; https://helpx.adobe.com/fresco/using/vector-brushes.html.[Accessed 11 February 2022].
[52] Lindeberg T. Edge detection and ridge detection with automatic scale selection. In: Proceedings CVPR IEEE computer society conference on computer vision and pattern recognition. 1996, p. 465–70.
[53] Elder JH, Zucker SW. Local scale control for edge detection and blur estimation. IEEE Trans Pattern Anal Mach Intell 1998;20(7):699–716.
[54] Maini R, Aggarwal H. Study and comparison of various image edge detection techniques. Int J Image Process (IJIP) 2009;3(1):1–11.
[55] Xie S, Tu Z. Holistically-nested edge detection. In: Proceedings of the IEEE international conference on computer vision. 2015.
[56] Liu Y, Cheng M, Hu X, Wang K, Bai X. Richer convolutional features for edge detection. In: 2017 IEEE conference on computer vision and pattern recognition. 2017, p. 5872–81.
[57] He J, Zhang S, Yang M, Shan Y, Huang T. BDCN: Bi-directional cascade network for perceptual edge detection. IEEE Trans Pattern Anal Mach Intell 2020;1–14.
[58] Canny J. A computational approach to edge detection. IEEE Trans Pattern Anal Mach Intell 1986;PAMI-8(6):679–98.
[59] Benjamin MD, DiVerdi S, Finkelstein A. Painting with triangles. In: NPAR 2014, Proceedings of the 12th international symposium on non-photorealistic animation and rendering. 2014.
[60] Kovesi P. MATLAB and octave functions for computer vision and image processing: Edge linking and line segment fitting. 2020, Online; https://www.peterkovesi.com/matlabfns/index.html#edgelink. [Accessed 31 March 2022].
[61] Schneider PJ. An algorithm for automatically fitting digitized curves. In: Graphics gems. USA: Academic Press Professional, Inc.; 1990, p. 612–26, https://dl.acm.org/doi/10.5555/90767.90941.
[62] Mandad M, Campen M. Bézier guarding: Precise higher-order meshing of curved 2D domains. ACM Trans Graph 2020;39(4).
[63] Hu Y, Schneider T, Gao X, Zhou Q, Jacobson A, Zorin D, et al. TriWild: Robust triangulation with curve constraints. ACM Trans Graph 2019;38(4):52:1–52:15.
[64] Mandad M, Campen M. Guaranteed-quality higher-order triangular meshing of 2D domains. ACM Trans Graph 2021;40(4).
[65] Yuksel C, Keyser J, House DH. Mesh colors. ACM Trans Graph 2010;29(2).
[66] Yuksel C, Lefebvre S, Tarini M. Rethinking texture mapping. Comput Graph Forum 2019;38(2):535–51, (Proceedings of Eurographics 2019).
[67] Yuksel C. Mesh colors with hardware texture filtering. In: ACM SIGGRAPH 2016 Talks. SIGGRAPH '16, New York, NY, USA: Association for Computing Machinery; 2016.
[68] Mallett I, Seiler L, Yuksel C. Patch textures: Hardware support for mesh colors. IEEE Trans Vis Comput Graphics 2020. [in press].
[69] Liu S, Jacobson A, Gingold Y. Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces. ACM Trans Graph 2014;33(6):1–9.