Locally refinable gradient meshes supporting branching and sharp colour transitions

Barendrecht, Pieter J.; Luinstra, Martijn; Hogervorst, Jonathan; Kosinka, Jiří

Published in:
Visual computer

DOI:
10.1007/s00371-018-1547-1

Publication date:
2018

The Visual Computer (2018) 34:949–960
https://doi.org/10.1007/s00371-018-1547-1

ORIGINAL ARTICLE

Locally refinable gradient meshes supporting branching and sharp colour transitions

Towards a more versatile vector graphics primitive

Pieter J. Barendrecht[1] · Martijn Luinstra[1] · Jonathan Hogervorst[1] · Jiří Kosinka[1]

Published online: 23 May 2018

Abstract
We present a local refinement approach for gradient meshes, a primitive commonly used in the design of vector illustrations with complex colour propagation. Local refinement allows the artist to add more detail only in the regions where it is needed, as opposed to global refinement which often clutters the workspace with undesired detail and potentially slows down the workflow. Moreover, in contrast to existing implementations of gradient mesh refinement, our approach ensures mathematically exact refinement. Additionally, we introduce a branching feature that allows for a wider range of mesh topologies, as well as a feature that enables sharp colour transitions similar to diffusion curves, which turn the gradient mesh into a more versatile and expressive vector graphics primitive.

Keywords Vector graphics · Gradient meshes · Colour interpolation · Local refinement

1 Introduction

Vector graphics provide an interesting alternative to raster graphics, as they are resolution-independent and in general easier to edit because they are composed of objects (often called *primitives*) as opposed to coloured pixels. The number of different primitives is rather limited, with the *gradient mesh* as one of the most complex ones. This primitive facilitates the creation of intricate colour propagation over regular quadrilateral meshes, contributing to the possibility of designing photorealistic illustrations in vector format.

Gradient meshes were introduced by Adobe Illustrator [1] as the *Mesh* object. CorelDRAW soon followed [7] with a similar tool called *Mesh Fill*. More recently, the primitive was added to the proposal for SVG 2 [27] and is now available in Inkscape [14] as well. Although there are subtle differences in the way gradient meshes behave in these software packages (see also Fig. 6), the overall concept is the same. In this paper we use the term *traditional gradient mesh* to refer to the primitive as implemented in these suites.

Upon using the traditional gradient mesh, it becomes clear that it comes with several limitations. The main shortcoming is its fixed rectangular topology, making it difficult to design complex objects and impossible to perform local refinement. As is well-known in Computer Graphics, the latter aspect is a desirable feature [12]. Global refinement (propagating through the entire row/column) often adds control points to the mesh where they are not needed, thereby cluttering the workspace and potentially slowing down the workflow. The conceptual difference between global and local refinement is depicted in Fig. 1. In addition, the traditional gradient mesh allows little to no control over the propagation of colour along curves and (as a consequence) in the interior, thus limiting the range of artistic styles that can be expressed with it.

It is our aim to create a more versatile and expressive primitive, in particular from the perspective of manual creation and editing. Our contributions in this direction are:

– Mathematically exact local refinement, which adds control points only where they are required

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00371-018-1547-1) contains supplementary material, which is available to authorized users.

✉ Pieter J. Barendrecht
p.j.barendrecht@rug.nl

[1] Bernoulli Institute, University of Groningen, Nijenborgh 9, 9747 Groningen, The Netherlands

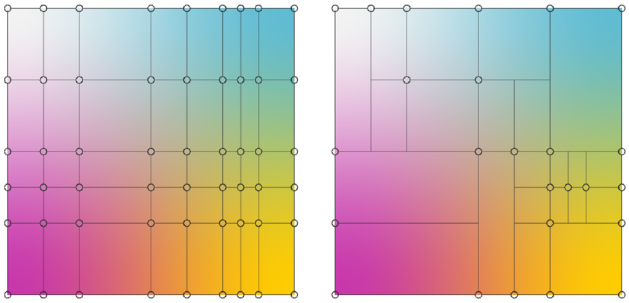


Fig. 1 Global refinement (left), where each refinement propagates through an entire row/column, is the only form of refinement available in existing implementations of the gradient mesh primitive. We propose a local refinement approach (right)

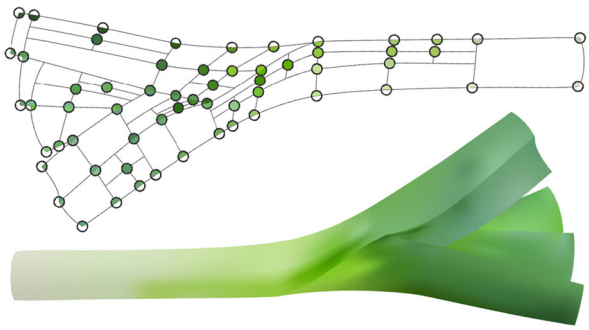


Fig. 2 Using a *single* gradient mesh to model a leek (control net top, rendering bottom). Local refinement is used to express local details, the branching feature is used to create the non-rectangular topology desired for modelling this shape, and sharp colour transitions are used to express depth

– Branching, which enables the creation of non-rectangular meshes by allowing branches, loops and (as a consequence) holes
– Assignment of colours to individual sectors of control points, which facilitates sharp colour transitions

An example gradient mesh using all three novel features is shown in Fig. 2.

The structure of the remainder of this paper is as follows. In Sect. 2, we discuss and compare related work on gradient meshes and alternative tools. We describe our interpretation of the gradient mesh primitive in Sect. 3, focusing on geometry and colour blending. Refinement is discussed in detail in Sect. 4. User interaction (i.e., editing a gradient mesh) is considered in Sect. 5. Relevant details regarding the implementation are described in Sect. 6. Branching and sharp colour transitions are described in Sect. 7 and are illustrated using a number of examples (see also the *supplementary video*). Finally, we evaluate the advantages and limitations of our contributions in Sect. 8, and look ahead to future work in Sect. 9.

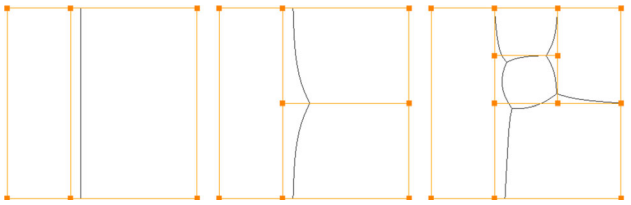


Fig. 3 Local refinement using Catmull–Clark subdivision. Vertices in the control net (orange) are generally not interpolated. Local refinement affects the valency of faces, which results in different limit surfaces (patch boundaries in black)

2 Related work

The traditional gradient mesh primitive is conceptually straightforward. As such, the existing literature focuses on extensions or applications of the tool. The prior work can be classified into three different categories, discussing either the *topology* of the gradient mesh, aspects related to *colour*, or *vectorisation*, the (semi-) automatic creation of vector graphics from raster graphics. We conclude with a concise overview of alternative techniques.

Considering the matter of restricted topology, in related fields *subdivision surfaces* are often used to model meshes of arbitrary topology. The catch in using them for gradient meshes is that neither *interpolating* subdivision schemes nor *approximating* ones can be directly applied in a generally satisfactory way. Using the former often results in colours outside the gamut (which leads to flat or uniformly coloured spots), whereas the latter typically results in washed-out colours. The approaches taken by [17,31] use Loop's approximating subdivision scheme for editing and vectorisation. Consequently, in practice the triangular elements in the mesh need to be quite small for the intended colours to be approximated closely enough (they are not interpolated), which in turn might complicate the editing process. This issue is mostly avoided in [18] by combining augmented linear ternary subdivision and Catmull–Clark's approximating scheme supporting quad-dominant meshes of arbitrary topology. However, though colour is interpolated, by default the geometry (represented by the control net) is not. Moreover, like [17,31], the proposed method does not allow for local refinement without changing the geometry or the colour propagation. In this subdivision setting, local refinement changes the valency of faces and/or introduces vertices of irregular valencies, thereby changing the limit surface. See Fig. 3, cf. Fig. 1 (right).

Finally, neither the PDF 2.0 [21] nor the SVG 2 standard supports the surface patches required to render subdivision surfaces, which reduces the applicability of subdivision-based methods.

Methods not based on subdivision include the use of curved Bézier triangles [28] for vectorisation. The approach touches only briefly on colour continuity across two trian-

gular elements, which is not sufficient for overall continuity. Although editing of the generated elements is discussed to some extent, the method does not appear to support manual addition of elements. Finally, [16] considers vectorisation as an application of their generalised barycentric coordinates for Hermite interpolation, which allows the use of highly irregular patch shapes. While interesting for vectorisation, the method does not seem suitable for intuitive manual creation of gradient meshes.

To the best of our knowledge, no extensive comparison regarding the use of different colour spaces for gradient meshes has been published. However, preference for certain colour spaces, including CIELAB, has been mentioned before [18]. Instead, work on colour-related aspects focuses on *colour transfer*, which discusses how colour characteristics from an existing (raster) image can be used to (re-)colour a gradient mesh [29,30].

Most work primarily focuses on vectorisation. In [22], cubic Bézier patches are used to approximate individual objects extracted from raster images. Interestingly, it also discusses the need for (in this case, automatic) local refinement to efficiently express local detail. However, smoothness of the result is not guaranteed, and locally refined areas cannot be edited directly. A different approach is taken in [25], where the positions and other properties at control points of a given (manually created) gradient mesh of rectangular topology can be optimised. In contrast, [15] proposes a completely automatic approach, which is more generally applicable as it can vectorise objects with holes using a single mesh. However, like [25], it uses rectangular meshes, and holes can only be formed by separating surface patches along a chain of curve segments, which restricts their shape.

Obtaining the desired topology using a (traditional) gradient mesh can be quite a challenge. A common workflow is to use multiple gradient meshes to design the envisioned object. Still, manual creation of complex shapes remains time-consuming. *Diffusion curves* [20] alleviate this task by allowing the user to draw outlines and open or closed inner regions where changes in colour might occur, using Bézier or B-spline curves. Assigning (possibly different) colours to the two sides of the curves, the interior colour is determined using the principle of diffusion. From a mathematical point of view, this is interpreted as solving a harmonic or biharmonic [11] equation. In addition to the multigrid method used originally, existing approaches for solving these PDEs include FEM [6], BEM [26] and approximation by ray-tracing [5]. Although the workflow is closer to the classical workflow of artists, a drawback is that the solution of the diffusion process typically requires a global approach. Moreover, traditional diffusion curves do not allow much control over the colour propagation. Various methods that mitigate the latter issue, such as diffusion barriers, anisotropic diffusion and colour strength, have been proposed in [4].

Related to diffusion curves, [19] associates *shading profiles* to the curves and subsequently converts the scene to a Catmull–Clark subdivision surface which can be rendered directly. As it is subdivision-based, it comes with the same drawbacks as mentioned above.

3 The gradient mesh primitive

We now construct our interpretation of the gradient mesh primitive. First, we consider the geometrical aspects, after which we focus on blending the colours defined on top of the 2D geometry. In addition, "Appendix A" briefly discusses the use of different colour spaces.

As pointed out in [24], (semi-)transparency in vector graphics is an important feature. In addition to the R, G and B channels, a fourth channel A could be added to smoothly blend different alpha values assigned to control points. To fully appreciate this feature, a layer system should be integrated to handle overlapping meshes. Although straightforward to implement, our framework currently does not support it as it adds little value from a scientific point of view. In the following, we adopt the RGB colour space.

3.1 Geometry

In essence, the geometrical aspect consists of a 2D curve network in the xy-plane with (traditionally) the topology of a rectangular grid. An example is shown in Fig. 1 (left). The curves composing this network are commonly cubic curves. Throughout the paper we use both Bézier and Hermite formulations of these curves (and later surface patches). As such, each cubic curve $C(t)$ can be described as

$$C(t) = \sum_{k=0}^{3} P_k B_k(t) = \boldsymbol{P}^{\mathrm{T}} \boldsymbol{B}(t) \tag{1}$$

$$= \sum_{k=0}^{3} Q_k H_k(t) = \boldsymbol{Q}^{\mathrm{T}} \boldsymbol{H}(t), \tag{2}$$

where $t \in [0, 1]$. Note that P_k are the Bézier control points and $B_k(t)$ the cubic Bernstein functions, Q_k the Hermite controls (points or tangent handles) and $H_k(t)$ the cubic Hermite functions.

As two different representations are considered, a way to convert between them is desirable. Hermite control data $\boldsymbol{Q}$ can readily be obtained from the Bézier control data $\boldsymbol{P}$:

$$\boldsymbol{Q} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{P} = \boldsymbol{X}\boldsymbol{P}. \tag{3}$$

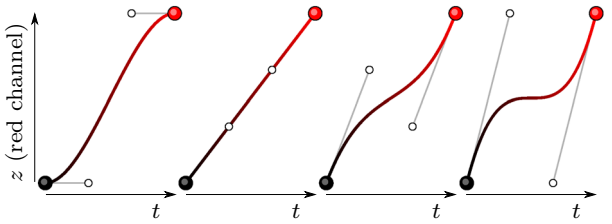


Fig. 4 Different colour blending functions with cubic parameterisation. Only the first one is G^1-compatible

Likewise, $\boldsymbol{P}$ can be obtained from $\boldsymbol{Q}$ by inverting the matrix $\boldsymbol{X}$.

3.2 Colour blending

Once the 2D geometry is defined, a colour can be assigned to each Hermite control point in the curve network. Immediately, the question arises how the colour should be propagated not only along the curves, but also within the quadrilateral regions bounded by these curves.

Assigning an RGB colour to each Hermite control point can be interpreted as extending the planar curve network to 5D. Alternatively, we can consider a separate height-field for each colour channel on top of the 2D geometry, such that the z-component takes on the R, G or B value, which we choose to map to the interval $[0, 1]$. We use this latter view, extending the curve network to 3D for each colour channel in turn.

Following the assignment of z-values to the Hermite *control points*, the z-values of the Hermite *tangents* are initialised to 0. Assuming co-linear geometrical handles on the sides of control points (see e.g., Fig. 8), this connects the curves in the 3D geometry/colour channel space with tangent continuity (G^1), and even C^1 in case these geometrical handles are of equal length. Clearly, other options for blending the colours along the curves become available when the Hermite tangents are assigned nonzero z-values, as illustrated in Fig. 4. Although this variation facilitates the use of different artistic styles, the drawback is that the colour curves no longer connect with tangent continuity.

Whether tangent continuity is required or not depends on the interpretation of *colour continuity*, which is clearly different from *geometric continuity* as humans perceive spatial changes in a different way than changes in colour. This is most likely caused by the different spectral sensitivities of the three types of cone cells in the eyes [23], and might be influenced by genetic and even cultural aspects. We postulate that although tangent continuity might not be a *necessary* condition for smooth colour propagation, it is a *sufficient* condition. Therefore, we choose to use colour blending which results in at least G^1 continuity (see Fig. 4, far left).

Having fixed the colour blending along the curves, we now take a look at the propagation of colour in the regions bounded by these curves. As we only have data on the curves

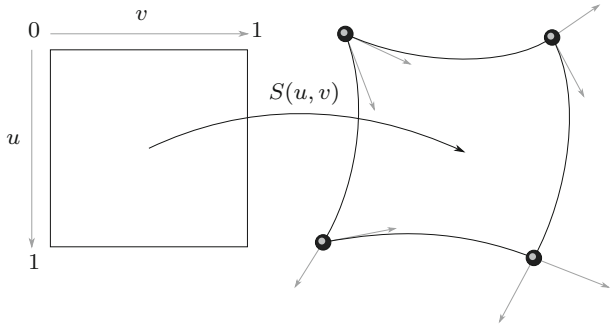


Fig. 5 A Ferguson surface patch is defined by four control points and eight tangent vectors. The visualised tangent vectors are scaled down by a factor 3

and not in the interior, a natural choice is to use some kind of *transfinite interpolation*. More specifically, we consider Coons patches that use a full set of cubics to blend the data on the boundary. As the boundary curves are also cubics, this results in a bicubic patch $S(u, v)$, which we choose to describe in Hermite form as

$$S(u, v) = \boldsymbol{H}^{\mathrm{T}}(u)\boldsymbol{C}\boldsymbol{H}(v), \tag{4}$$

with

$$\boldsymbol{C} = \begin{pmatrix} S(0,0) & S_v(0,0) & S_v(0,1) & S(0,1) \\ S_u(0,0) & S_{uv}(0,0) & S_{uv}(0,1) & S_u(0,1) \\ S_u(1,0) & S_{uv}(1,0) & S_{uv}(1,1) & S_u(1,1) \\ S(1,0) & S_v(1,0) & S_v(1,1) & S(1,1) \end{pmatrix},$$

where $S(i, j)$, $S_u(i, j)$, $S_v(i, j)$ and $S_{uv}(i, j)$ are elements of $\mathbb{R}^3$ with x and y the geometrical data and z the data from one of the colour channels.

Initially, the mixed partials $S_{uv}(i, j)$, with $i, j \in \{0, 1\}$ (commonly referred to as *twist vectors* in Hermite form) at the control points, are set to zero as we do not have interior information. This results in a subset of bicubic Hermite patches known as *Ferguson* patches, illustrated in Fig. 5.

When we cast the patch into Bézier form as

$$S(u, v) = \boldsymbol{B}^{\mathrm{T}}(u)\boldsymbol{X}^{-1}\boldsymbol{C}\boldsymbol{X}^{-\mathrm{T}}\boldsymbol{B}(v),$$

where we have used the equality

$$\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{H}(t) = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{B}(t) = \left(\boldsymbol{X}^{-1}\boldsymbol{Q}\right)^{\mathrm{T}}\boldsymbol{B}(t) = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{X}^{-\mathrm{T}}\boldsymbol{B}(t),$$

it becomes evident that vanishing twist vectors correspond to inner Bézier points positioned in such a way that they form parallelograms with the closest corner and two corresponding handles.

When editing the geometry or colour and refining (individual) patches (see Sect. 4), the twist vectors might become nonzero, which is why we adopt the full Hermite description in favour of merely the Ferguson one in the remainder of this paper.

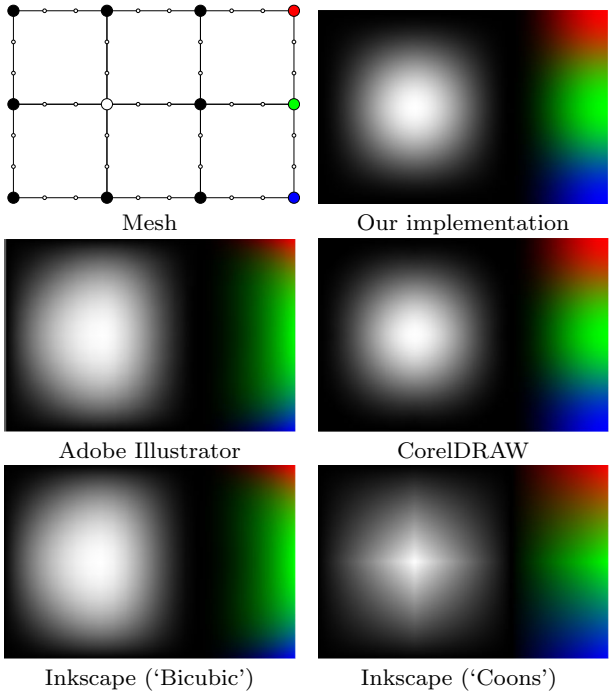


Fig. 6 Comparing renderings of a 2×3 gradient mesh in different software packages. Handles are at their default location and are visualised as small open circles

We conclude this subsection with the observation that existing implementations of the traditional gradient mesh primitive use different blending functions, see Fig. 6. Used versions are Adobe Illustrator 2017.1.0, CorelDRAW 19.0.0.328 (Graphics Suite 2017), and Inkscape 0.92.2. Based on these and other results, our approach resembles most that of CorelDRAW. Adobe Illustrator supposedly uses a different way of interpolation [25], which is also used in the bicubic option in Inkscape. The other option in Inkscape uses bilinear Coons interpolation.

4 Refinement

In our setting, (local) refinement (also referred to as *splitting*) refers to the introduction of a new curve that is the image of an u- or v-isoline of the patch to be split. In the following, we describe the splitting process along the u-direction, though the process is completely analogous along the v-direction.

Creating a new curve (expressed in Hermite form) involves the evaluation of position and first derivatives (tangents). Moreover, we need the mixed partial derivatives (twist vectors) at the new control points to properly describe the new patches. The required computations are based on the following set of equations:

$$S(u,v) = \boldsymbol{H}^{\mathrm{T}}(u)\boldsymbol{C}\boldsymbol{H}(v), \qquad \frac{\partial S}{\partial v} = \boldsymbol{H}^{\mathrm{T}}(u)\boldsymbol{C}\boldsymbol{H}'(v),$$
$$\frac{\partial S}{\partial u} = \boldsymbol{H}'^{\mathrm{T}}(u)\boldsymbol{C}\boldsymbol{H}(v), \qquad \frac{\partial^2 S}{\partial u\,\partial v} = \boldsymbol{H}'^{\mathrm{T}}(u)\boldsymbol{C}\boldsymbol{H}'(v). \quad (5)$$

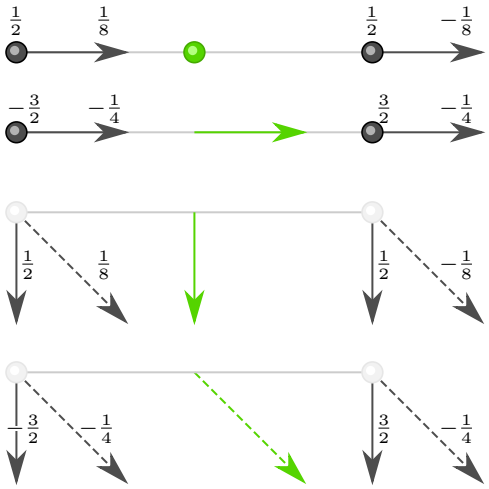


Fig. 7 Four stencils to compute the quantities in (5) for $u = 0$ or 1 and $v = \frac{1}{2}$ in Hermite form (green). Tangents are visualised as solid arrows, twist vectors as dashed arrows

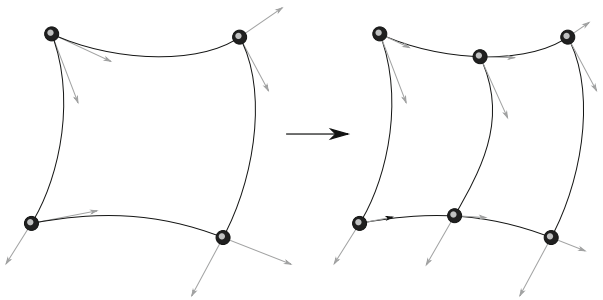


Fig. 8 Splitting a patch using the stencils from Fig. 7 to compute positions and tangent vectors. The twist vectors are not shown

It follows that the unknown quantities in (5) can be computed just from the information available on the boundary of the patches. Using the matrix $\boldsymbol{C}$ introduced in (4), this results in the *stencils* shown in Fig. 7.

At this point, patches can be split by applying the stencils on two opposite curves, see Fig. 8. For example, a new control point at $v = \frac{1}{2}$ is obtained as a weighted sum (in this case using the weights from the top stencil in Fig. 7) of the two control points and two tangent handles defining the curve of interest. Examples of refinement are shown in Figs. 2, 14, 15, 16 and 17, where individual patches have been split to locally accommodate more detail. Note that *local* refinement leads to *T-sections* in the mesh. In Sect. 5 we take a look at these from the user's point of view, whereas details regarding the implementation and book-keeping concerning T-sections are discussed in Sect. 6.

We underline that our splitting approach is *mathematically exact*, which means that no change in geometry or colour propagation occurs upon splitting. The only difference could be caused by the (adaptive) tessellation step when it is not pixel-accurate (see Sect. 6.2). In contrast, splitting meth-

ods in existing implementations are rarely exact. Figure 9 compares the accuracy of the splits in the aforementioned software packages. The renderings before and after splitting have been imported in GIMP as different layers using 8 bits per colour channel, with the *mode* of the top layer set to *difference*. The layers are subsequently merged and thresholded, where the highest value resulting in white pixels has been recorded. Lower values for thresholding have been selected to emphasise the regions with the largest difference.

Clearly, the splitting algorithms in Adobe Illustrator and CorelDRAW are not exact. In the former case, the twist vectors do not seem to be updated correctly, and in the latter case there is an issue most likely related to the computation of tangent handles. Splitting in the case of Inkscape's bicubic approach indicates severe problems regarding the computation of both position and tangent handles and possibly twist vectors. The splitting procedure for Inkscape's bilinear Coons implementation seems theoretically sound, though results in minor artefacts caused by the rendering approach. As mentioned, our implementation only shows a difference when the triangles created in the tessellation step are significantly larger than single pixels, and even in that case the difference is minimal.

5 User interaction

In this section we discuss how the continuity of the colour surface is affected by user interaction. As an initial mesh (with or without local splits) is C^1 by construction, we first look at what it takes to maintain this continuity. As it turns out that the resulting constraints can be somewhat restrictive, we subsequently discuss how the interaction can be made more flexible.

5.1 C^1 continuity

The requirement for C^1 continuity between two surface patches (mapped from the same domain) is straightforward [10]. In our bicubic setting, we have two sets of tangent handles and two sets of twist vectors that should remain co-linear and symmetric with respect to the shared curve. Of these, only the geometrical tangents can be edited by the user. Mirroring the movement of one handle to the one opposite of it is sufficient to preserve local C^1 continuity. Moving a tangent vector also updates one or more inner Bézier points, as these are related to the twist vectors which are kept constant. Note that nonzero twists might appear after movement of the tangent(s) logically orthogonal to a curve followed by (local) splitting, as is evident from the stencils in Fig. 7. Finally, as is customary in vector graphics software, tangent handles follow the movement of control points.

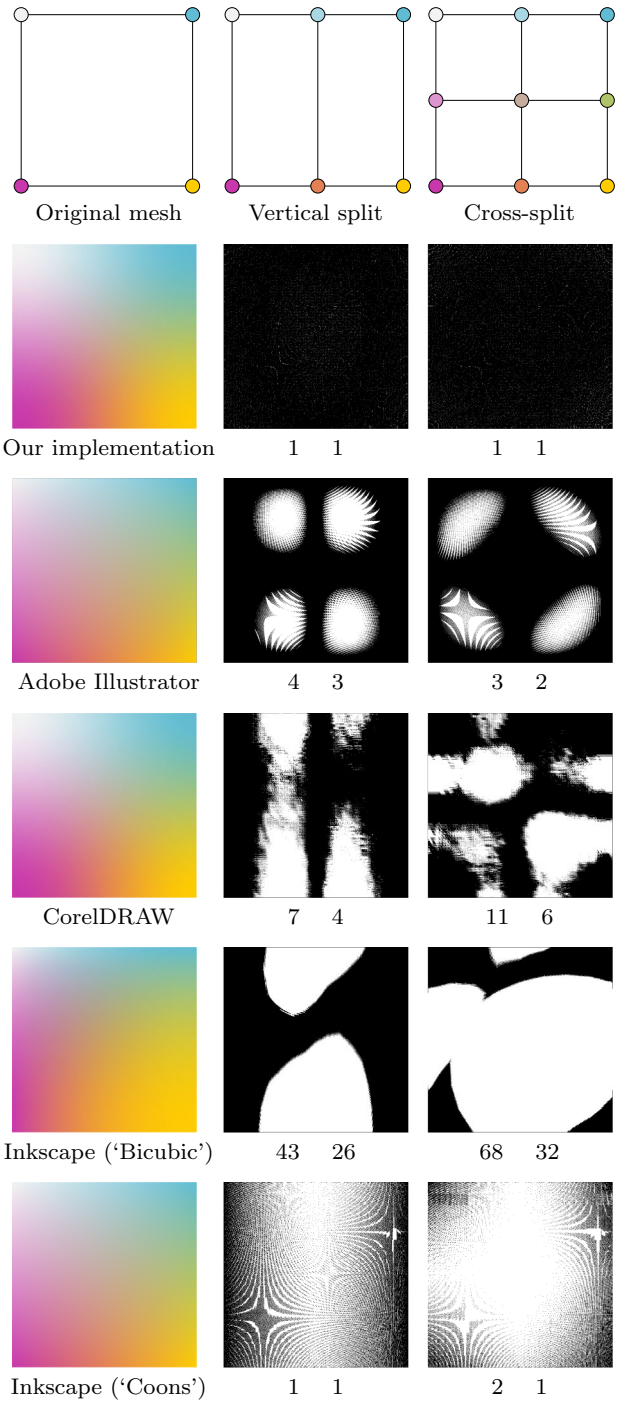


Fig. 9 Comparing refinement accuracy. The left column shows the rendering of an elementary gradient mesh (consisting of a single patch), the middle and right columns the differences after refining vertically and in both directions (cross), respectively. The two values below each result are thresholding values (minimal value and selected value for better visual comparison)

This completes maintaining C^1 in regular regions, and leaves us to consider the same task around T-sections. In this setting, data at T-sections cannot be modified by the

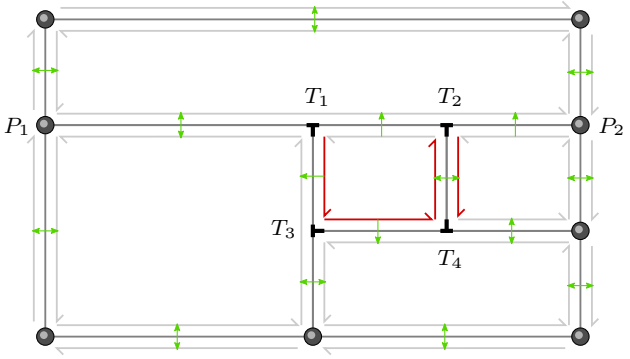


Fig. 10 A mesh with T-sections. Half-edges are indicated in grey. Green arrows indicate the *opposite* of a half-edge, which is not always a symmetrical relation near T-sections. T-sections have fixed links to the outgoing half-edge pointing in the direction of their stem, which are here indicated in dark red

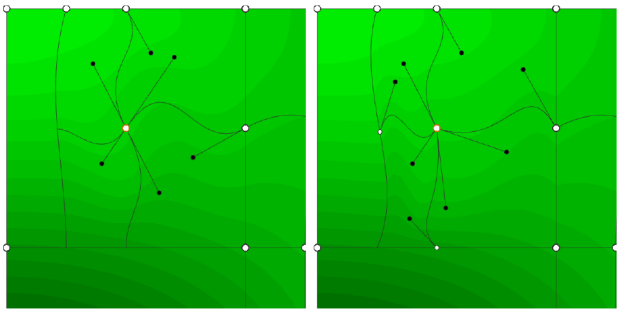


Fig. 11 User controls for the C^1 setting (left) and general (C^0) case (right), zooming in on the top-left part of the locally refined mesh in Fig. 1, here edited. Only the green colour channel is visualised, which is colour-banded in order to assess continuity. In the general setting, tangent handles at the T-sections can be edited by the user, and tangent handles at control points no longer need to be (pairwise) co-linear, resulting in C^0 transitions

user. Instead, the T-sections inherit their information from the longest curve (and associated patch) they are a part of. Moving a geometrical control point or tangent handle influencing a curve containing one or more T-sections triggers a procedure which updates this data. In the case of position and tangent vectors along the long curve, this is straightforward, whereas updating tangents in the other parametric direction and the twist vectors requires more effort because data from the adjacent patch need to be reflected and scaled by the appropriate factor. Note that modification of data at a T-section might again affect other T-sections. This only happens when the *stem* (vertical part of an upright **T**) of an updated T-section points at the *bar* (horizontal part of an upright **T**) of another T-section. For example, consider the mesh in Fig. 10, in which an update of the curve between P_1 and P_2 triggers the update of the two T-sections T_1 and T_2 on that curve. The stem of T_1 points at the bar of T_3 below it, whose stem in turn points to the bar of T_4.

Finally, we also update logically orthogonal tangent handles at T-sections (as well as adjacent twist vectors) when logically parallel tangent handles at control points on the curve containing the T-section are updated. For example, consider the tangent handle pointing down from T_1 in Fig. 10. According to the stencils in Fig. 7 this handle (as well as its adjacent twist vectors) depends on the tangent handles pointing down from P_1 and P_2.

Note that a change in geometry causes a change in colour propagation because colour is defined on top of the geometry. However, the actual colour values remain unchanged. Of these values, only the colour at Hermite control points can be updated by the user. The colour tangents directly follow from the choice of blending function along the curve (see Fig. 4) and can only be nonzero following a (local) split, automatically maintaining a smooth connection. Colour twists follow the colour tangents.

The last aspect to consider is an update of the colour of a control point that is part of a curve containing T-sections. The procedure followed is analogous to the one for geometry, including the occurrence of chain reactions which are required to correctly update the information at T-sections further away from the affected curve in order to maintain C^1 continuity.

5.2 More flexibility

Forcing co-linear tangent vectors to be of the same length is required for C^1 continuity, but might pose a restriction on the design of a desired image. Unfortunately, merely keeping them co-linear does not automatically imply G^1 continuity. The requirements for this type of continuity are more involved [9], and are at this point not discussed in more detail.

Letting go of tangent continuity altogether exposes more freedom to the user. In this setting, geometric tangent handles can be edited individually, allowing the design of sharp interior features. Tangent handles along the bar of a T-section are still updated automatically, but those in the other parametric direction (along the stem of a T-section) can now be modified by the user. Figure 11 compares the user controls for the C^1 and general C^0 setting.

6 Implementation

In this section, we consider the relevant aspects of the implementation, starting with an outline of the developed data structure, followed by an overview of the employed adaptive GPU-based tessellation. User controls such as moving points or handles, or colour picking from a background texture, are not discussed. We refer to the *supplementary video* for more details.

6.1 Data structure

Although local refinement for gradient meshes could be implemented using, e.g., (a forest of) binary trees, we instead chose to use an augmented half-edge data structure to support additional and future extensions.

Using a mixture of Bézier and Hermite representations, the Hermite control points are connected by half-edges, which in turn link to the handles (stored in Bézier format). Each half-edge also links to a twist vector, stored in Hermite format. In addition, colour information is also stored in the half-edges. Near T-sections, the *opposite* (often referred to as *twin* or *pair*) of an older half-edge is always the newer half-edge pointing to the origin of the older one. This ensures a stable cycle around non-T-sections using the *next* and *opposite* links of a half-edge. Multiple newer half-edges can therefore have the same older half-edge as their *opposite*. The level of a half-edge is increased by 1 every time it is split, where level 0 refers to the original (oldest) half-edges.

Control points contain a flag indicating whether they are a T-section or not. At T-sections, the outgoing link from a control point to a half-edge is always the half-edge logically orthogonal to the T-section, whereas for other control points it can be an arbitrary outgoing half-edge like in the classical half-edge structure. The reason for this fixed link is that it facilitates the traversal of an edge in both directions. Figure 10 shows an example mesh with T-sections in which the behaviour deviating from a classical half-edge mesh is highlighted.

In the case of T-sections, there are two choices for the storage of handles along the adjacent pairs of half-edges of different level. One option would be to simply store two handles per half-edge. A more elegant solution is to share handles between a pair of half-edges with different levels, which in practice amounts to the multiplication of the tangent handle (in Hermite form) by a scaling factor which readily follows using de Casteljau's algorithm. The result is that a T-section always has its own handles on both sides, whereas other handles can be scaled versions of older ones. Figure 12 illustrates this approach for the central edge P_1P_2 in Fig. 10 where dyadic refinement has been applied.

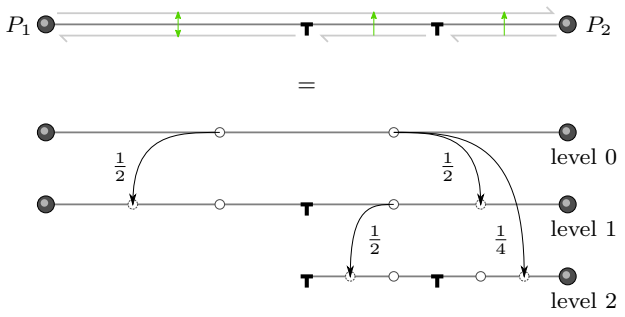


Fig. 12 Sharing handles between pairs of half-edges of different levels. The original handles are re-used and scaled by a factor $\frac{1}{2}$ after the first refinement, also resulting in two new handles on the sides of the T-section. One of these is re-used and scaled after the second refinement, whereas the other re-used handle is an original one scaled by a factor $\frac{1}{4}$

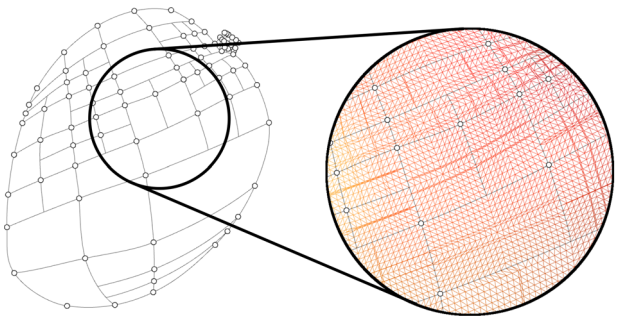


Fig. 13 Adaptive tessellation showing compatible even integer tessellation levels as well as fractional levels

6.2 Adaptive tessellation

With the advent of programmable shaders in the graphics pipeline, also the rendering of vector graphics can be accelerated [3]. In our application, the patches are rendered using the OpenGL tessellation shaders in *fractional even* mode, which is mostly straightforward. Only when the levels of two opposite half-edges are different, special care needs to be taken to select an even integer tessellation level along that edge such that no cracks will occur. Using an approximation of the on-screen length of the shortest section (associated with the newest half-edge) of the curve of interest, a tessellation level is computed in a pre-processing step, which is then multiplied by the appropriate power of 2 (based on the difference Δ in level of the two sides of the curve) for the longer sections of the edge. To illustrate this procedure, consider the short curve between T_1 and T_2 in Fig. 10. Assuming a computed tessellation level of 6 for this curve, the curve between P_1 and P_2 is assigned a tessellation level of $6 \cdot 2^{\Delta=2} = 24$, likewise the curve between P_1 and T_1 level $6 \cdot 2^{\Delta=1} = 12$, and finally the curve between T_2 and P_2 level $6 \cdot 2^{\Delta=0} = 6$.

Currently, the common hardware limit for tessellation levels is 64. In case of large patches or repeated refinement along a curve, the preferred level could exceed the maximum. In that case, the tessellation procedure can be extended such that these patches are *virtually* subdivided (subdivided only in memory), thereby mimicking levels of 128 and up.

For opposite half-edges of the same level, such as those between T_2 and T_4 in Fig. 10, the tessellation level is determined in the tessellation control shader and can be fractional. An example of the entire procedure is illustrated in Fig. 13.

A more involved alternative would be to use the pixel-accurate approach from [13], although we should mention that this only considers accuracy in geometry, which does not automatically guarantee accuracy in colour.

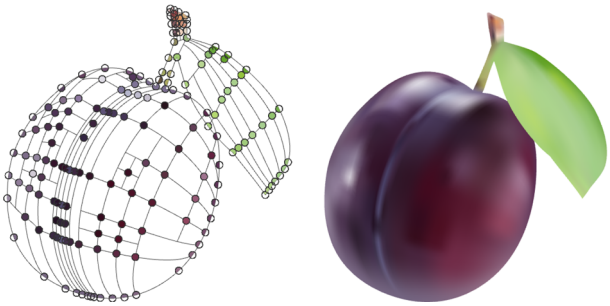


Fig. 14 A plum modelled using local refinement and branching

7 Artistic features

The two contributions considered in this section are mostly interesting from an artistic point of view, as they bring more expressiveness to the primitive without introducing new technical details. See the *supplementary video* for additional examples.

7.1 Branching

Although the use of multiple, possibly (partially) overlapping gradient meshes might still be the preferred way of designing a complex shape, creating so-called *branches* in the mesh has proven to be an interesting alternative. Clearly, as we are no longer bound to a fixed rectangular topology, this allows for the design of meshes of more complex shape. The C^1 conditions (see Sect. 5) can be enforced by reflecting tangents and twist vectors.

Moreover, multiple branches can easily be stitched together to form closed loops (and therefore holes), a feature that can also be used to design genuine *radial* gradient meshes (see Fig. 15). In existing implementations, radial gradient meshes are merely rectangular gradient meshes with multiple overlapping control points, making them troublesome to edit.

Figures 2, 14 and 17 show examples of gradient meshes with branching.

7.2 Sharp colour transitions

In Sect. 6 we mentioned that colour information is stored in half-edges. The reason to do so is that it facilitates discontinuity of colour around a control point, which can be used to model sharp colour transitions similar to diffusion curves. This obviously violates continuity properties discussed in Sect. 5, but in return provides the artist with more freedom. As images often contain colour discontinuities, it is a very useful feature to have.

Assigning the different colours is realised by only updating the colour of a specific *sector* of a control point. Each adjacent patch has its own sector, which means that up to 4 different colours can be assigned to one control point. The control net shown in Fig. 15 contains various control points with multiple colours assigned to them. Examples of this extension are illustrated in Figs. 2, 16 and 17.

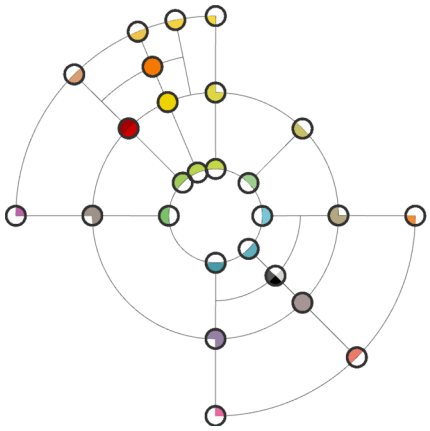


Fig. 15 The control net of a radial gradient mesh with local refinement, branches and multiple colours per control point enabling sharp colour transitions. The control points are enlarged for visualisation purposes

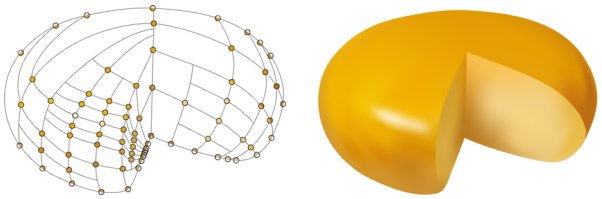


Fig. 16 A wheel of cheese modelled using local refinement and sharp colour transitions

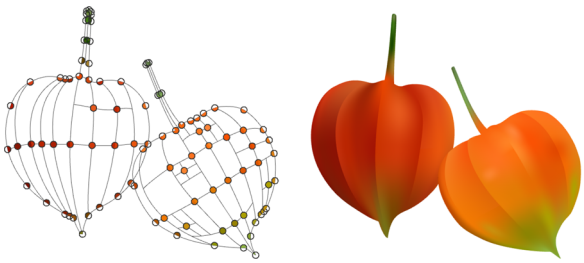


Fig. 17 Two husks of *Physalis alkekengi* ('Chinese lantern') modelled using local refinement, branching and sharp colour transitions

8 Discussion and conclusion

In this paper we have described our interpretation of a gradient mesh primitive extended with mathematically exact local refinement, branching capabilities and sharp colour transitions.

Local refinement allows the user to add detail only where needed, such as local reflections of a light source. It helps the designer to maintain an overview of the topology of the mesh,

Table 1 Comparing the number of patches required to model the examples shown in this paper (figure numbers are indicated between brackets) using our version (local refinement and branching) and traditional gradient meshes (global refinement and possibly using multiple meshes per object)

	Leek (2)	Plum (14)
Ours	59	154
Traditional	93	274
	Cheese (16)	Lanterns (17)
Ours	69	76
Traditional	126	145

instead of getting lost in a clutter of points and curves which might slow down the workflow either directly or indirectly. Table 1 compares global and local refinement by listing the number of patches required to model the examples shown in this paper; see also Fig. 1.

Branching takes away some of the topological restrictions of the traditional gradient mesh, enabling the design of more complex objects using a single gradient mesh without reducing editability.

The possibility to assign multiple colours to a single control point provides the artist with a user-friendly way to include sharp colour transitions in a design, one of the attractive characteristics of diffusion curves.

Using adaptive crack-free tessellation allows for real-time editing, which remains interactive even for complex illustrations and (near) pixel-accuracy on our setup (i.e., an NVIDIA Quadro K420 GPU at a resolution of 2560×1440 pixels).

Ultimately, the result is an efficient, versatile and expressive primitive, which is compatible with the PDF and new SVG 2 standards. This compatibility makes it widely applicable, in contrast to several existing methods, including subdivision-based ones.

Experience and user feedback with regard to editing has shown that certain topologies are difficult or even impossible to create using a single extended gradient mesh. Using multiple objects is currently the only available alternative (see Sect. 9 for future work on this aspect). In addition, certain aspects of illustrations would benefit from the availability of other vector primitives. Integration of our tool in a general vector graphics editor is therefore desirable.

9 Future work

We conclude the paper with some promising ongoing and future work topics. First, more flexibility in topology of the curve network is highly desirable. Although an approach based on subdivision surfaces for quad-dominant meshes is available [18], it comes with several drawbacks as mentioned in Sect. 2. An alternative method based on polynomial or rational patches, instead of the special patches required for subdivision surfaces, could work. However, as tangent continuity of the colour surface (in each channel) might be required, this is not a trivial task.

Secondly, improved colour control would add to the range of styles supported by the primitive. Combining this with a G^1 blending function hints at introducing additional degrees of freedom along individual curves.

Combining both topics with vectorisation of raster graphics would be a very challenging direction of research. Besides the artistic applications, it might also be interesting from the perspective of (image) compression, certainly in conjunction with local refinement.

With the upcoming support of SVG 2 in most web browsers, *animated* gradient meshes using, e.g., the data structure introduced in [8] would make an interesting topic for research.

Finally, as mentioned in [2], the next step in this field could very well be the development of a primitive combining the positive characteristics from both gradient meshes and diffusion curves.

Acknowledgements Parts of this paper are based on, or were inspired by, the BSc theses of the second and third authors. The real-life examples of gradient meshes were inspired by existing photographs (from third parties) under a *Creative Commons* license or similar and were created manually by the first author.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

A Colour spaces

In addition to the colour blending functions discussed in Sect. 3, we considered the use of different colour *spaces*. We briefly discuss the main characteristics of HSV, HSL, HSI, CIELCh$_{ab}$ (cylindrical spaces), and CIEXYZ, CIELUV, CIELAB, sRGB as well as linear sRGB.

Cylindrical colour spaces represent *hue* in a single channel H. Because of the periodical nature of H, interpolation can be performed either clockwise or counter-clockwise, which turns out to be problematic. For example, interpolation in

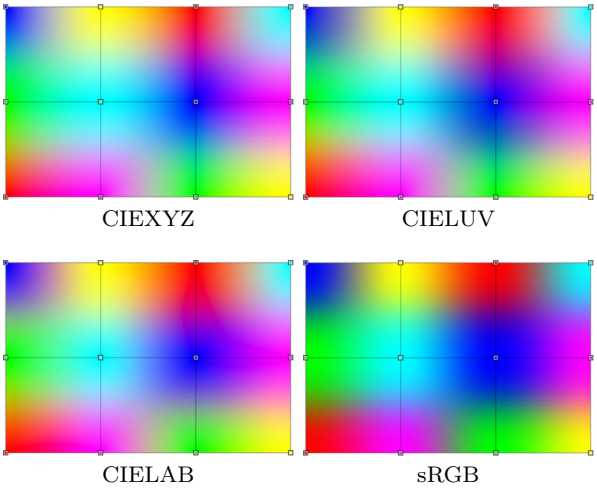


Fig. 18 Renderings of a 2×3 gradient mesh interpolated in different colour spaces using G^1 colour blending

HSV between red ($H = 0°$) and magenta ($H = 300°$) yields an undesired colour gradient including yellow, green, cyan and blue when going counter-clockwise. Clearly, clockwise interpolation is preferred in this case. Ambiguity of direction occurs with a 180° difference between two colours, whereas subsequently tweaking one of these colours results in unstable and non-intuitive behaviour. An alternative would be to linearly interpolate within the hue/saturation disc in HSV, though also in this case the results are often undesirable. We conclude that cylindrical colour spaces are not suitable in the context of gradient meshes.

Results of using the other colour spaces are shown in Fig. 18. Note that this overview does not include linear sRGB, as it is a linear transformation of CIEXYZ and therefore produces the same interpolation results. CIELUV, CIELAB and sRGB produce slightly darker results than CIEXYZ as they include a nonlinear gamma correction. Dark regions can occur when interpolating in sRGB between markedly different hues. Both CIELUV and CIELAB seem to be suitable spaces for colour interpolation, which makes sense as they were specifically designed to be perceptually uniform. Of these two, CIELAB introduces some undesirable purple when interpolating from blue to yellow, which makes CIELUV the best choice among the colour spaces considered.

In practice, the difference between rendering in the default sRGB colour space and CIELUV is hardly perceptible in our experience. The reason is that the difference in colour of adjacent control points is usually much less extreme than depicted in Fig. 18, and moreover, the patches are typically much smaller. In addition, conversion between sRGB and CIELUV (on the GPU) is non-trivial. For these reasons, our current implementation uses sRGB.

References

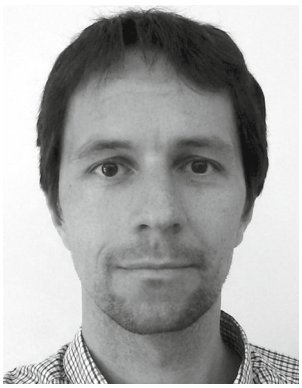
1. Adobe Systems. Adobe Illustrator 8.0 Classroom in a Book. Adobe Press (1998)
2. Barla, P., Bousseau, A.: Gradient art: creation and vectorization. In: Rosin, P., Collomosse, J. (eds.) Image and Video-Based Artistic Stylisation, pp. 149–166. Springer, New York (2013)
3. Batra, V., Kilgard, M.J., Kumar, H., Lorach, T.: Accelerating vector graphics rendering using the graphics hardware pipeline. ACM Trans. Graph. **34**(4), 146 (2015)
4. Bezerra, H., Eisemann, E., DeCarlo, D., Thollot, J.: Diffusion constraints for vector graphics. In: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, pp. 35–42. ACM (2010)
5. Bowers, J.C., Leahey, J., Wang, R.: A ray tracing approach to diffusion curves. In: Computer Graphics Forum, vol. 30, pp. 1345–1352. Wiley (2011)
6. Boyé, S., Barla, P., Guennebaud, G.: A vectorial solver for free-form vector gradients. ACM Trans. Graph. **31**(6), 173 (2012)
7. Coburn, F.D., McCormick, P.: CorelDRAW 9: The Official Guide. Osborne/McGraw-Hill, Berkeley (1999)
8. Dalstein, B., Ronfard, R., Van de Panne, M.: Vector graphics animation with time-varying topology. ACM Trans. Graph. **34**(4), 145 (2015)
9. DeRose, T.D.: Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces. Comput. Aided Geom. Des. **7**(1–4), 165–179 (1990)
10. Farin, G.E.: Curves and surfaces for CAGD: a practical guide. Morgan Kaufmann, Burlington (2002)
11. Finch, M., Snyder, J., Hoppe, H.: Freeform vector graphics with controlled thin-plate splines. In: ACM Trans. Graph., vol. 30, p. 166. ACM (2011)
12. Forsey, D.R., Bartels, R.H.: Hierarchical B-spline refinement. ACM Siggraph Comput. Graph. **22**(4), 205–212 (1988)
13. Hjelmervik, J.M., Fuchs, F.G.: Interactive pixel-accurate rendering of LR-splines and T-splines. In: Eurographics (Short Papers), pp. 65–68 (2015)
14. Inkscape Wiki. Mesh gradients (2018). http://wiki.inkscape.org/wiki/index.php/Mesh_Gradients. Accessed 01 Feb 2018
15. Lai, Y.-K., Hu, S.-M., Martin, R.R.: Automatic and topology-preserving gradient mesh generation for image vectorization. In: ACM Trans. Graph., vol. 28, p. 85. ACM (2009)
16. Li, X.-Y., Ju, T., Hu, S.-M.: Cubic mean value coordinates. ACM Trans. Graph. **32**(4), 126:1–126:10 (2013)
17. Liao, Z., Hoppe, H., Forsyth, D., Yu, Y.: A subdivision-based representation for vector image editing. IEEE Trans. Vis. Comput. Graph. **18**(11), 1858–1867 (2012)
18. Lieng, H., Kosinka, J., Shen, J., Dodgson, N.A.: A colour interpolation scheme for topologically unrestricted gradient meshes. In: Computer Graphics Forum, vol. 36, pp. 112–121. Wiley (2017)
19. Lieng, H., Tasse, F., Kosinka, J., Dodgson, N.A.: Shading curves: vector-based drawing with explicit gradient control. In: Computer Graphics Forum, vol. 34, pp. 228–239. Wiley (2015)
20. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion curves: a vector representation for smooth-shaded images. ACM Trans. Graph. **27**(3), 92:1–92:8 (2008)
21. PDF Association. ISO 32000-2 (PDF 2.0). Technical report, International Organization for Standardization, (2017)
22. Price, B., Barrett, W.: Object-based vectorization for interactive image editing. Vis. Comput. **22**(9), 661–670 (2006)
23. Reinhard, E., Khan, E.A., Akyuz, A.O., Johnson, G.: Color imaging: fundamentals and applications. CRC Press, Boca Raton (2008)
24. Richardt, C., Lopez-Moreno, J., Bousseau, A., Agrawala, M., Drettakis, G.: Vectorising bitmaps into semi-transparent gradient layers. In: Computer Graphics Forum, vol. 33, pp. 11–19. Wiley (2014)

25. Sun, J., Liang, L., Wen, F., Shum, H.-Y.: Image vectorization using optimized gradient meshes. In: ACM Trans. Graph., vol. 26, p. 11. ACM (2007)
26. Sun, X., Xie, G., Dong, Y., Lin, S., Xu, W., Wang, W., Tong, X., Guo, B.: Diffusion curve textures for resolution independent texture mapping. ACM Trans. Graph. **31**(4), 74 (2012)
27. W3C SVG Working Group. Scalable Vector Graphics (SVG) 2 (2018). https://www.w3.org/TR/SVG2/. Accessed 01 Feb 2018
28. Xia, T., Liao, B., Yu, Y.: Patch-based image vectorization with automatic curvilinear feature alignment. In: ACM Trans. Graph., vol. 28, p. 115. ACM (2009)
29. Xiao, Y., Wan, L., Leung, C.-S., Lai, Y.-K., Wong, T.-T.: Example-based color transfer for gradient meshes. IEEE Trans. Multimed. **15**(3), 549–560 (2013)
30. Xiao, Y., Wan, L., Leung, C.S., Lai, Y.-K., Wong, T.-T.: Optimization-based gradient mesh colour transfer. In: Computer Graphics Forum, vol. 34, pp. 123–134. Wiley (2015)
31. Zhou, H., Zheng, J., Wei, L.: Representing images using curvilinear feature driven subdivision surfaces. IEEE Trans. Image Process. **23**(8), 3268–3280 (2014)

Pieter J. Barendrecht obtained his B.Sc and MSc degrees in Mechanical Engineering from Eindhoven University of Technology, the Netherlands. He is currently pursuing a Ph.D. in Computer Graphics at the University of Groningen, the Netherlands, focusing on the application of splines in vector graphics, numerical analysis and 3D modelling.

Martijn Luinstra is a Computing Science student at the University of Groningen, the Netherlands. He plans on attending the M.Sc in Computing Science at the University of Groningen, with focus on computer graphics and data science.

Jonathan Hogervorst received his B.Sc degrees in Artificial Intelligence and Computing Science from the University of Groningen, the Netherlands in 2017. He plans on following the M.Sc in Security & Network Engineering at the University of Amsterdam, the Netherlands.



Jiří Kosinka is an Assistant Professor in the Scientific Visualization and Computer Graphics research group at the Johann Bernoulli Institute of the University of Groningen, the Netherlands. He received his Ph.D. degree from Charles University in Prague in 2006. His research interests include computer graphics, geometric modelling, and computer-aided design.