STAG: Smart Tools and Applications in Graphics (2021)
P. Frosini, D. Giorgi, S. Melzi, and E. Rodolà (Editors)

Mesh Colours for Gradient Meshes

S. D. Baksteen[1], G. J. Hettinga[1], J. Echevarria[2] and J. Kosinka[1]

[1]Bernoulli Institute, University of Groningen, the Netherlands
[2] Adobe Research, San Jose, CA, USA

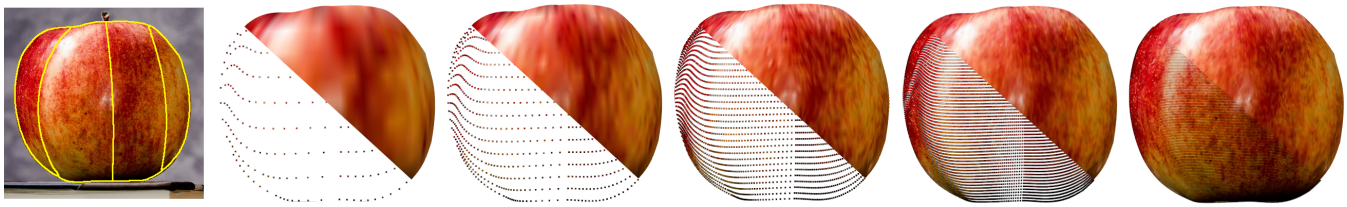

Figure 1: *An image of an apple with mesh boundaries (yellow) shown overlaid (far left). Right: The fitted apple using a gradient mesh with visualised mesh colours rendered with four patches of different resolutions:* $R = 8, 16, 32, 64, 128$ *(from left to right).*

Abstract

We present an extension of the popular gradient mesh vector graphics primitive with the addition of mesh colours, aiming to reduce the mesh complexity needed to describe intricate colour gradients and textures. We present interesting applications to user-guided authoring of detailed vector graphics and image vectorisation.

CCS Concepts

• ***Computing methodologies*** → ***Image processing;***

1. Introduction

The gradient mesh [Ado06] is a vector graphics primitive that allows the user to create a regular quadrilateral mesh that smoothly interpolates colours between vertices. The shapes of the quadrilaterals can be manipulated through tangent handles, allowing for smooth curves to bound the shape. In this way, an image can be modelled through a deformable mesh of patches.

One downside of this gradient mesh primitive is that in order to model images with complex colour gradients, or with a high level of detail, the mesh necessarily becomes dense (see Figure 8, top), as colours can be specified only at the corners of each quadrilateral; see Figure 2.

To overcome that limitation, we explore the addition of mesh colours [YKH08; YKH10] to the gradient mesh primitive, aiming at decreasing the mesh density needed to represent complex gradients and texture information. Mesh colours are an alternative to texture mapping for 3D meshes, allowing to map a texture to each polygon in the mesh, but without the need of complex global parameterizations.

Our results show that the addition of mesh colours to gradient meshes removes the burden on the designer to create complicated meshes just to be able to model colour correctly. Moreover, due to the sparsity of our meshes we create a fast vectorisation work flow, where the user has to define only a sparse mesh and the mesh colours will be automatically fit. We compare our solution with the locally refinable gradient meshes of [BLHK18].

2. Related Work

We start by briefly reviewing relevant prior work on gradient meshes and mesh colours.

2.1. Gradient Meshes

A gradient mesh is a popular vector graphics primitive for defining smooth colour surfaces. It is defined as a mesh of connected 2D patches, in which each vertex is assigned a colour. As illustrated in Figure 2, gradient meshes typically make use of bicubic patches of which the edges are cubic curves. Tangent handles at each vertex guide the geometry of these curves and the spread of colour inside the patches. As such, gradient meshes provide a way to intuitively interpolate between multiple colours. To increase their expressiveness, recent extensions include arbitrary manifold topologies [LKSD17; VK18; HBK19] and interpolation of procedural

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

DOI: 10.2312/stag.20211478

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

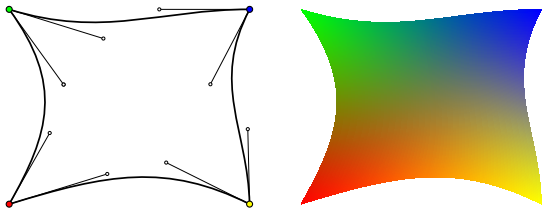

Figure 2: *A simple gradient mesh. Left: The vertex colours and tangent handles. Right: The resulting bicubic interpolant.*

noise functions [HvK19]. Traditionally, gradient meshes were only refinable globally. Locally refinable gradient meshes [BLHK18] allow to decrease mesh density. Gradient meshes have been used in the past for automatic image vectorisation [LHM09; SLWS07]. Our work aims at efficiently increasing texture detail, especially in the context of user-guided image vectorisation.

Our gradient meshes consist of bicubic patches bounded by cubic curves, expressed in Hermite form. Each curve is defined by the positions and tangents at its endpoints. This creates a type of bicubic patch known as a Ferguson patch [BLHK18].

This patch can be defined as

$$\mathbf{S}(u,v) = \mathbf{H}(u)^T \mathbf{Q} \mathbf{H}(v)$$

on $[0,1]^2$, where $\mathbf{H}(t) = \begin{pmatrix} 2t^3-3t^2+1 \\ -2t^3+3t^2 \\ t^3-2t^2+t \\ t^3-t^2 \end{pmatrix}$ are the cubic Hermite functions, and

$$\mathbf{Q} = \begin{pmatrix} \mathbf{S}(0,0) & \mathbf{S}(0,1) & \mathbf{S}_v(0,0) & \mathbf{S}_v(0,1) \\ \mathbf{S}(1,0) & \mathbf{S}(1,1) & \mathbf{S}_v(1,0) & \mathbf{S}_v(1,1) \\ \mathbf{S}_u(0,0) & \mathbf{S}_u(0,1) & \mathbf{0} & \mathbf{0} \\ \mathbf{S}_u(1,0) & \mathbf{S}_u(1,1) & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

contains the Hermite control points, namely the positions and first partial derivatives (tangents) at each vertex of the patch. $\mathbf{S}(i,j)$ are elements of $\mathbb{R}^5$ containing (x,y) position and (r,g,b) colour. Notably, the (r,g,b) components of $\mathbf{S}_v(i,j)$ and $\mathbf{S}_u(i,j)$ in the above matrix are set to 0. If neighbouring patches have gradients $\mathbf{S}_v$ and S_u in the same directions and magnitude, this creates a tangent continuous (and actually C^1) bicubic interpolation of colours and geometry across the mesh. The four zeroes in $\mathbf{Q}$ represent the mixed partial derivatives $\mathbf{S}_{uv}(i,j)$ of the Ferguson patch.

2.2. Mesh Colours

Mesh colours were originally devised as an alternative to traditional texture mapping for 3D polygon meshes [YKH08]. They are similar to vertex colours, in which each vertex in a mesh is assigned a colour, and the surface of each patch interpolates between the colours at its vertices. However, instead of colours being defined only at the vertices, they are also defined at regularly spaced points along edges and faces of the surface of each patch.

The resolution R of a mesh colour patch is defined for our purposes as the number of colour positions between each vertex on an edge, minus one. So a patch with $R = 1$ specifies colours only at the vertices, and a patch with $R = 2$ has one extra colour position on each edge and one logically in the middle of the patch. Patches can have an individually defined resolution, meaning that a mesh can contain patches of varying resolutions. This can be used to adaptively give additional detail to complex parts of the mesh, while leaving low resolution patches to describe low-detail portions, giving great texture detail with minimal memory usage [YKH10]. Mesh colours have been used in the context of interactive painting of 3D models and for image vectorisation using (curved) triangles [HEK21], but not in combination with gradient meshes.

3. Method and Implementation

We now detail our primitive, gradient meshes using mesh colours, and its implementation.

3.1. Framework

Our program renders the mesh using a tessellation shader, which partitions each patch into many triangles, which are positioned and coloured according to the parametric definition of the patch as noted in Section 2.1. A tessellation shader consists of the tessellation control shader (TCS) and the tessellation evaluation shader (TES). In the TCS, the level of tessellation is determined, which controls the number of triangles to subdivide the patch into, which can be determined by the user using a slider in the UI. The tessellation is computed from there, and the resulting triangles are then sent to the TES, which determines position and parametric position of each triangle's vertices based on local coordinates on the patch and the patch's vertex positions, and tangents, which are passed into it from the TCS. Then afterwards, the mesh colours are evaluated on a per-fragment basis using the interpolated parametric coordinates as passed from the TCS.

3.2. Texture Arrays

Previous mesh colour implementations [Yuk17] pack patch textures inside a single two-dimensional texture (with additional textures for lower mip levels) or advocate the use of bindless-textures [MSY20]. We opt to use OpenGL `Texture2DArrays`, a type of texture that stores many separate 2D textures in a three-dimensional structure, with the notable limitation that each texture in the array needs to have the same dimensions. Each patch's colour data then receives an index in this texture array. This way, operations on the mesh colours, such as brushing (Section 3.5), can be handled efficiently without resorting to recomputing the whole mesh colour texture.

Using a single `Texture2DArray`, however, causes a problem. Because each texture in the array needs to have the same dimensions, there will be wasted space in the textures of lower-resolution patches, as illustrated in Figure 3. The dimensions of the texture array will be larger than necessary for the texture of these lower-resolution patches, wasting memory space in the texture array.

In order to prevent this, multiple texture arrays were used instead of one. This requires the patch resolutions supported by the program to be restricted, as only a limited number of texture arrays

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

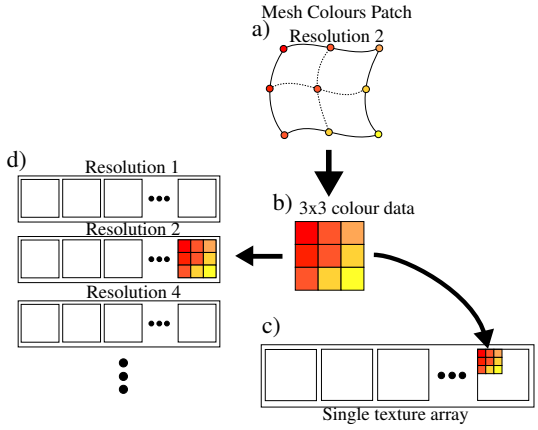

Figure 3: *a) A Ferguson patch with mesh colours shown at $R = 2$, along with (b) the representation of its mesh colours in memory. c) A diagram depicting a patch's colour data being stored inside a texture array. The new texture contains empty space surrounding the colour data, as the texture array's dimensions are larger than the patch's resolution. d) A resolution 2 patch's colour data is stored in the corresponding texture array.*

can be used. Resolutions were restricted to $R = 2^r, r \in \{0,1,\ldots,7\}$; i.e. powers of two up to and including 128.

Now, each patch has a power-of-two resolution up to 128, and its colour data is stored inside one of eight `Texture2DArrays` corresponding to each of these values. Each texture array contains N_R textures of size $R+1$ by $R+1$, where R is its corresponding mesh colour resolution. This eliminates any waste of memory space, as each patch's colour data fits perfectly in one of these texture arrays. Patches are each assigned an index in the texture array they belong to, with each texture array being resized when needed to fit new patches. When the resolution of a patch is changed, the relevant texture arrays are reconstructed and resized as well. The user can change the resolution of a patch by selecting it and using hotkeys to either double or halve the resolution. When resolution is doubled, the colour data is bilinearly interpolated between to make the required new data. This results in no loss of quality when using bilinear interpolation for rendering, as the lower resolution patch is perfectly reproduced in the higher resolution. Doubling the resolution whilst using bicubic interpolation will not exactly reproduce the lower resolution.

In order to access the texture data of a particular patch, the index of the patch's corresponding texture array is needed, which can be derived from its resolution as well as the index of the patch within the texture array. The mesh colour data can then be accessed using 3D texture coordinates (u,v,i), where i is the index of the texture within the texture array. The bottom left mesh colour can be obtained through integer coordinates $\mathbf{p} = (i,j)$ that are derived from a zero-based index in the $R+1$ by $R+1$ colour data as

$$\mathbf{p} = \mathbf{t}(u,v) = \lfloor (R+1) \cdot (u,v) \rfloor.$$

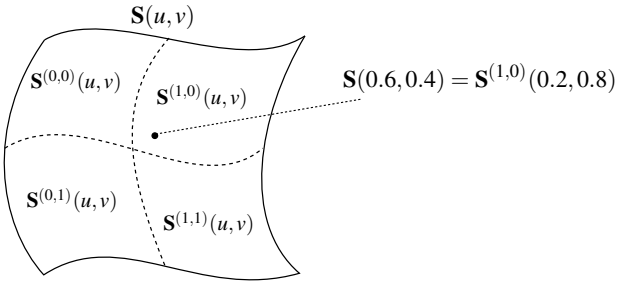

Figure 4: *An example of a mesh colours patch of resolution $R = 2$, with subpatches shown and labeled. The point $\mathbf{S}^{(1,0)}(0.2,0.8) = \mathbf{S}\left(\frac{1+0.2}{2}, \frac{0+0.8}{2}\right) = \mathbf{S}(0.6,0.4)$ is shown (position not accurate).*

3.3. Colour Interpolation

As mentioned before, our base gradient mesh implementation uses bicubic interpolation (Section 2.1). To generalise it for mesh colours for a patch with resolution R, we can consider a patch $\mathbf{S}(u,v)$ as consisting of R^2 sub-patches $\mathbf{S}^{(i,j)}(u,v)$ with $i,j \in \{0,1,\ldots,R-1\}$, $u,v \in [0,1]$ and

$$\mathbf{S}^{(i,j)}(u,v) = \mathbf{S}\left(\frac{i+u}{R}, \frac{j+v}{R}\right).$$

The spatial portion of S is determined in the same way as before, but the colours are interpolated within each sub-patch. Let us denote the colour portion of $\mathbf{S}(u,v)$ as $\mathbf{C}(u,v)$ and of $\mathbf{S}^{(i,j)}(u,v)$ as $\mathbf{C}^{(i,j)}(u,v)$. The colours at the corners of each of these $\mathbf{S}^{(i,j)}$ are exactly the mesh colours assigned to the patch: For $u,v \in \{0,1\}$, $\mathbf{C}^{(i,j)}(u,v) = \text{tex}(\mathbf{t}((i+u, j+v)))$, where tex is a function retrieving colour data from the patch's texture given (u,v) coordinates. This is illustrated in Figure 4.

For bicubic interpolation, a similar method can be used to the one used originally in the initial gradient mesh implementation: we take $\mathbf{C}^{(i,j)}(u,v) = \mathbf{H}^T(u)\mathbf{Q}\mathbf{H}(v)$ where $\mathbf{H}(t)$ are the cubic Hermite functions and

$$\mathbf{Q} = \begin{pmatrix} \mathbf{C}^{(i,j)}(0,0) & \mathbf{C}^{(i,j)}(0,1) & \mathbf{0} & \mathbf{0} \\ \mathbf{C}^{(i,j)}(1,0) & \mathbf{C}^{(i,j)}(1,1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

contains the colour values at each corner of the sub-patch. For bilinear interpolation, we simply bilinearly interpolate the four corner values.

3.4. Edge Smoothing

When working with mesh colour patches with different resolutions, it is often desired for the edges of the patches to blend smoothly with each other, without colour discontinuities at the edge. Whereas the original mesh colours implementation did this implicitly for every edge in the patch [Yuk17], we offer it an option to be toggled on and off for each edge, to allow for sharp colour discontinuities if desired.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

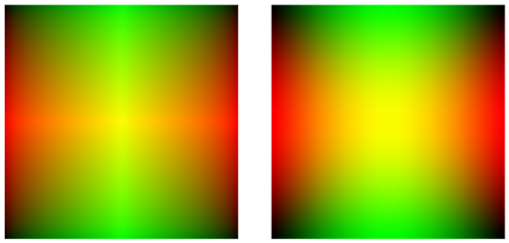

Figure 5: *A basic mesh colours patch of $R = 2$ rendered with bilinear interpolation (left) and bicubic interpolation (right).*

The program keeps track of the edges which are linked by storing the indices of the linked patches in question and the index (starting from the half-edge linked directly to the patch) of the half-edges that are linked. Then, if both patches have the same resolution, the mesh colours are copied from one of the patches to the other whenever they are modified. If one patch has a lower resolution than the other, the colours are strictly copied from the lower-resolution patch to the higher-resolution patch, linearly interpolating the colour points on the higher-resolution patch that do not exist on the lower-resolution patch from the existing colours. This means that the edge looks perfectly smooth when bilinear colour interpolation is used, but not necessarily for bicubic colour interpolation, as the colours on the higher-resolution patch do not exactly match the displayed colours on the lower-resolution patch at the same points.

In Figure 6, this edge smoothing approach is illustrated. The patch on the bottom has a lower resolution than the patch above it, as does the patch on the right. When applying edge smoothing, the colours are copied from those patches to the central patch. This results in a smooth continuous transition between the patches. At the same time, this may result into a loss of detail; which is also why we keep this smoothing as an optional feature.

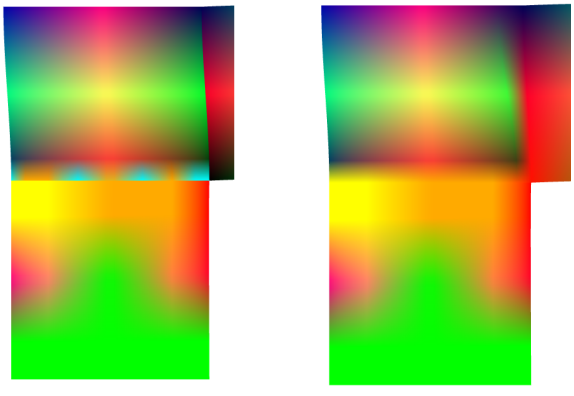

Figure 6: *A three-patch mesh using mesh colours. On the left, the mesh is seen with sharp colour discontinuities at the edges of the patches. On the right, the same mesh is shown after application of edge smoothing, eliminating discontinuities.*

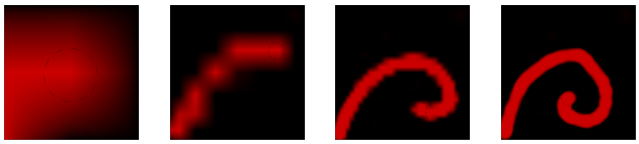

Figure 7: *The results of using the brush tool to draw with red on a completely black patch, moving the mouse along similar curves for patches of resolution $R = 2, 8, 16, 32$ (from left to right).*

3.5. Painting Tools

To allow for direct manipulation of mesh colours by the user, a pencil and brush tool have been implemented. These tools allows the user paint over the gradient mesh (Figure 7). In [YKH08] While dragging the mouse, the closest mesh colour sample point on the screen is replaced with the selected colour. This is done by iterating through each patch and then through each sample point in the patch. For each point, its local coordinates on the patch are transformed to global coordinates using the parametric description of the patch, then the distance is calculated to the mouse position. The closest point overall is then chosen to be replaced with the new colour.

3.6. Automatic Image Colour Sampling

To allow users to more easily convert raster images to gradient meshes with mesh colours, we automatically sample colours from a raster image (Figures 1 and 8). Our method iterates over each patch and each mesh colour sample point within them. For each point, its coordinates are transformed to local coordinates inside the raster image. The mesh colour is then assigned the bilinearly sampled colour of the image.

4. Results

Figure 8 shows a comparison between an image created using a traditional gradient mesh with local refinement and a recreation using mesh colours. Using only three patches instead of many, and high-resolution mesh colours, a decent approximation was created of the original using our primitive. There are more colour sample points in the mesh colours recreation than in the original version, which is needed to make up for the mesh colours not being at the same location as the vertices in the refined vertex colours mesh, also taking up more memory space overall. But the geometry of our mesh is very simple compared to the complex gradient mesh of [BLHK18].

Figure 1 shows the results of fitting a much more textured image of an apple. Notice the effects of increasing mesh colour resolution. Another result is shown in Figure 9, this time using a banana image.

5. Discussion

As shown in the previous section, the mesh colours implementation can produce high complexity gradients using low mesh density, compared to traditional gradient meshes which need higher density meshes to capture the same. A mesh using mesh colours will also use less data than an exactly equivalent traditional mesh, because if the mesh colours match up exactly with the vertex control points on

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

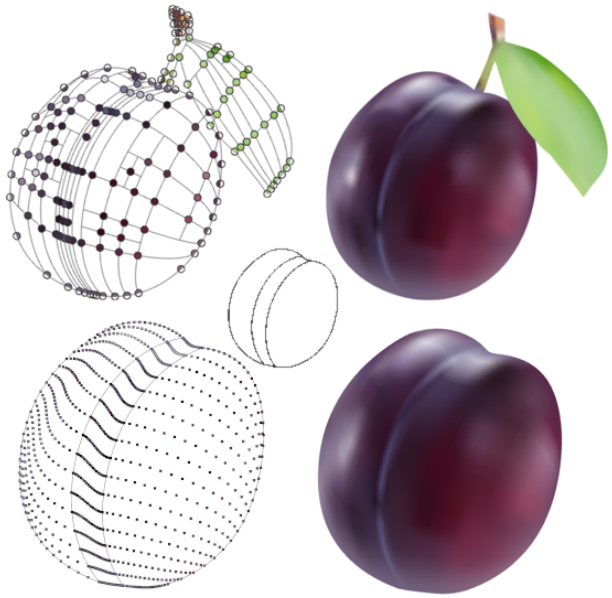

Figure 8: *Top: An image of a plum created using a previous gradient mesh implementation [BLHK18]. On the left, the mesh boundaries are shown with vertex colours visible, while on the right the interpolated result is shown. Image and mesh adopted from [BLHK18]. Bottom: The same image of a plum recreated with mesh colours, using a combination of image fitting and manual adjustments with the pencil tool in order to avoid capturing the overlapping leaf. Each patch in the mesh has a mesh colours resolution* $R = 16$ *for a total of* 867 *mesh colour samples. In the middle, the mesh boundaries are shown, and on the right is the interpolated result (bilinear interpolation).*

the traditional mesh, the traditional mesh will have additional data in tangents and vertices associated with these points, which is not needed in the mesh colours mesh.

However, in practice, when adapting a traditional gradient mesh with mesh colours using significantly reduced mesh complexity, the new mesh will likely need more mesh colours than there are vertices in the original, to make up for the fact that the simpler geometry sacrifices some flexibility where colour control points are located, as seen in Figure 8, where the adapted mesh using mesh colours contains significantly more colour points than the original. In this case, the new mesh may use more storage space.

Compared to traditional gradient mesh implementations, the mesh colours version can capture significantly more detail for each patch, and therefore capture the same image using significantly less complex mesh geometry. However, this also sacrifices flexibility, in that mesh colours are distributed evenly across the surface of each patch, and cannot be individually moved, whereas users can exactly specify the position of each vertex in a traditional mesh. The properties of the gradient are also specified entirely by the larger, less dense patches, creating less opportunity for fine-tuning the specifics of the gradient than an approach using dense mesh geometry.

Figure 9: *Far left: The original image of a bunch of bananas. Left: One banana manually traced, with mesh overlaid. Right: The banana rendered using image fitting with resolution* $R = 1$ *for all patches, totalling* 84 *mesh colour samples. Far right: The banana rendered using bilinear interpolation with mesh colour resolutions ranging from* $R = 4$ *for most patches to* $R = 16$ *for the brown tip of the banana, totalling* 1357 *mesh colour samples.*

These restrictions, however, enable an intuitive implementation of edge smoothing, as all possible patch resolutions are multiples of each other, and edge smoothing using patch resolutions that are not multiples of each other would need an entirely different implementation to properly remove discontinuities. In addition, the limit on the number of possible resolutions serves a purpose in increasing (texture) storage efficiency of the implementation. Previous implementations of mesh colours [Yuk17] use packing algorithms to fit texture data of arbitrary resolutions in the same texture, which causes some wasted storage space, as this packing is not fully efficient, creating up to 16% storage overhead for large meshes [Yuk17]. Our implementation has no storage overhead, at the cost of restricting the space of possible patch resolutions.

6. Conclusion and Future Work

Our main goal was to add mesh colours to the gradient mesh primitive, in the hopes of reducing the amount of complexity needed in the mesh geometry to represent complicated colour gradients and textures. A memory-efficient implementation of mesh colours has been created for gradient meshes, as well as basic tools for the manipulation and creation of such meshes.

The results have shown that this method can create geometrically simple meshes with a great level of colour detail, as well as the possibility of vectorisation of raster images using image fitting and manually created meshes with simple geometry, yielding good results. Basic direct manipulation of mesh colours is also provided through a brush tool. More elaborate brush tools would be interesting future work.

Other interesting avenues for future research improved colour fitting beyond single pixel colours or bilinear colour sampling and automatic patch resolution selection by comparing the result of image fitting on a patch to the input raster image.

Acknowledgements This paper is in part based on the first author's BSc thesis project at the University of Groningen.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

References

[Ado06] ADOBE. *Adobe PDF*. https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdf_reference_archive/pdf_reference_1-7.pdf. 2006 1.

[BLHK18] BARENDRECHT, PIETER J, LUINSTRA, MARTIJN, HOGERVORST, JONATHAN, and KOSINKA, JIŘI. "Locally refinable gradient meshes supporting branching and sharp colour transitions". *The Visual Computer* 34.6 (2018), 949–960 1, 2, 4, 5.

[HBK19] HETTINGA, GERBEN J., BRALS, RENÉ, and KOSINKA, JIŘÍ. "Colour interpolants for polygonal gradient meshes". *Computer Aided Geometric Design* 74 (2019), 101769. ISSN: 0167-8396 1.

[HEK21] HETTINGA, GERBEN J., ECHEVARRIA, JOSE, and KOSINKA, JŘÍ. "Efficient Image Vectorisation Using Mesh Colours". *STAG: Smart Tools and Applications in Graphics*. Ed. by FROSINI, P., GIORGI, D., MELZI, S., and RODOLÀ, E. The Eurographics Association, 2021 2.

[HvK19] HETTINGA, GERBEN J., VAN BECKHOVEN, ROWAN, and KOSINKA, JIŘÍ. "Noisy gradient meshes: Augmenting gradient meshes with procedural noise". *Graphical Models* 103 (2019), 101024. ISSN: 1524-0703 2.

[LHM09] LAI, YU-KUN, HU, SHI-MIN, and MARTIN, RALPH R. "Automatic and Topology-Preserving Gradient Mesh Generation for Image Vectorization". *ACM Trans. Graph.* 28.3 (July 2009). ISSN: 0730-0301 2.

[LKSD17] LIENG, HENRIK, KOSINKA, JIŘÍ, SHEN, JINGJING, and DODGSON, NEIL A. "A Colour Interpolation Scheme for Topologically Unrestricted Gradient Meshes". *Computer Graphics Forum* 36.6 (2017), 112–121 1.

[MSY20] MALLETT, IAN, SEILER, LARRY, and YUKSEL, CEM. "Patch Textures: Hardware Support for Mesh Colors". *IEEE Transactions on Visualization and Computer Graphics* (2020). ISSN: 1077-2626. DOI: 10.1109/TVCG.2020.3039777 2.

[SLWS07] SUN, JIAN, LIANG, LIN, WEN, FANG, and SHUM, HEUNG-YEUNG. "Image Vectorization Using Optimized Gradient Meshes". *ACM Trans. Graph.* 26.3 (July 2007), 11–es. ISSN: 0730-0301. DOI: 10.1145/1276377.1276391. URL: https://doi.org/10.1145/1276377.1276391 2.

[VK18] VERSTRAATEN, TEUN W. and KOSINKA, JŘÍ. "Local and Hierarchical Refinement for Subdivision Gradient Meshes". *Computer Graphics Forum* 37.7 (2018), 373–383 1.

[YKH08] YUKSEL, CEM, KEYSER, JOHN, and HOUSE, DONALD H. *Mesh Colors*. Tech. rep. Department of Computer Science, Texas A&M University, 2008 1, 2, 4.

[YKH10] YUKSEL, CEM, KEYSER, JOHN, and HOUSE, DONALD H. "Mesh colors". *ACM Transactions on Graphics* 29.2 (2010), 15:1–15:11. ISSN: 0730-0301 1, 2.

[Yuk17] YUKSEL, CEM. "Mesh Color Textures". *High-Performance Graphics (HPG 2017)*. Los Angeles, CA: ACM, 2017. ISBN: 978-1-4503-5101-0/17/07 2, 3, 5.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.