Feature-Adaptive and Hierarchical Subdivision Gradient Meshes

Zhou, J.; Hettinga, G. J.; Houwink, S.; Kosinka, J.

Published in:
COMPUTER GRAPHICS FORUM

DOI:
10.1111/cgf.14442

Publication date:
2022

DOI: 10.1111/cgf.14442

COMPUTER GRAPHICS *forum*
Volume 41 (2022), number 1 pp. 389–401

Feature-Adaptive and Hierarchical Subdivision Gradient Meshes

J. Zhou, G.J. Hettinga, S. Houwink and J. Kosinka

Bernoulli Institute, University of Groningen, Groningen, The Netherlands

Abstract

Gradient meshes, an advanced vector graphics primitive, are widely used by designers for creating scalable vector graphics. Traditional variants require a regular rectangular topology, which is a severe design restriction. The more advanced subdivision gradient mesh allows for an arbitrary manifold topology and is based on subdivision techniques to define the resulting colour surface. This also allows the artists to manipulate the geometry and colours at various levels of subdivision. Recent advances allow for the interpolation of both geometry and colour, local detail following edits at coarser subdivision levels and sharp colour transitions. A shortcoming of all existing methods is their dependence on global refinement, which makes them unsuitable for real-time (commercial) design applications. We present a novel method that incorporates the idea of feature-adaptive subdivision and uses approximating patches suitable for hardware tessellation with real-time performance. Further novel features include multiple interaction mechanisms and self-intersection prevention during interactive design/editing.

Keywords: image processing, curves & surfaces, modelling

CCS Concepts: • Imaging and Video → Image Processing; • Modelling → Parametric Curves and Surfaces

1. Introduction

The *gradient mesh* is a powerful vector graphics primitive that allows for the creation and manipulation of scalable vector graphics; see, e.g. [SLWS07, BLHK18]. The traditional gradient mesh is available in several commercial design applications such as Adobe Illustrator as well as in open source alternatives such as Inkscape. The existing interfaces allow positional and colour data to be assigned to mesh vertices, where gradient handles define the curved geometry and colour transitions. The meshes require regular rectangular topology and are represented as a grid of bicubic patches. The regularity requirement is a severe restriction on the artist and consequently the resulting colour surface: adding local detail requires global mesh refinement.

The *subdivision gradient mesh* primitive improves upon the (traditional) gradient mesh primitive by allowing an arbitrary manifold topology, provided that the faces are convex [LKSD17, SL17]. The resulting colour surface is obtained by applying a single ternary subdivision step to the input mesh, followed by Catmull–Clark subdivision [CC78] to the limit. This ensures an almost everywhere C^2 continuous colour surface. Hierarchical editing is possible by allowing the artist to manipulate the geometry and colours at various levels of subdivision. Recent advances allow for the interpolation of both geometry and colour, local detail following edits at coarser subdivision levels and support for sharp colour transitions [VK18].

Methods currently incorporating subdivision gradient meshes require multiple steps of global Catmull–Clark subdivision both to represent the finest edits and to obtain a smooth surface that is close to the actual limit surface. As the number of faces grows exponentially with the number of subdivision steps and current hardware performance is mostly memory restricted, such methods are not suitable for real-time (commercial) design applications.

Our novel subdivision gradient mesh representation and rendering method address the above limitations; see Figure 1. We

1. modify and compare approximating patches for Catmull–Clark subdivision surfaces that are suitable for hardware tessellation to the setting of subdivision gradient meshes;
2. present a novel and real-time method that uses the idea of local feature-adaptive subdivision (FAS) in combination with hierarchical editing;
3. design an index mapping between vertices, (half-)edges and faces across subdivision levels for efficient detail tracking;
4. investigate and discuss trade-offs between visual quality and performance;

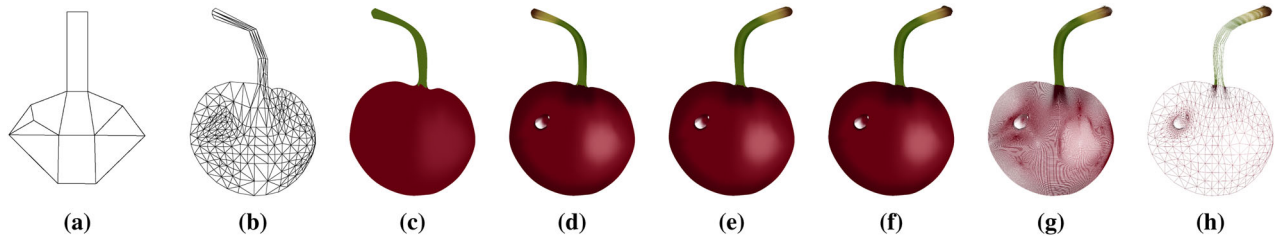

Figure 1: *Different subdivision levels and hierarchical edits of our feature-adaptive and hierarchical subdivision gradient mesh showcased on a cherry model. (a) The initial mesh consists of ten polygons. (b) The edited mesh at level 0 after geometry and colour editing. Level 0 is the mesh after an initial ternary step, here shown tessellated. (c) The corresponding rendering. (d) The edited model at level 4. (e) Back at level 0, we bend the stalk to the right by editing only 6 points; the finer edits at level 4 follow the overall shape change. (f) The locally edited model at level 5. We add some texture at the top of the stalk and adjust the shape of the bottom of the stalk. (g) The underlying mesh required for traditional mesh subdivision [VK18] has 78, 336 polygons. (h) The mesh generated using our method after the same amount of editing has only 1371 polygons. Our method supports true hierarchical yet fully local editing, which allows the artist to manipulate the geometry and colour at any level of subdivision.*

5. provide multiple user interface improvements, among which an indication of the region of influence of edits, and real-time self-intersection prevention while editing geometry.

We start by reviewing relevant related work in Section 2. A technical overview of the required building blocks is presented in Section 3. A summary of how these building blocks are adapted and incorporated in the design of our novel method is provided in Section 4. Improvements to the user interface are presented in Section 5. We then showcase the results in terms of performance and visual quality in Section 6. Several trade-offs and choices that were made, limitations and future work are discussed in Section 7. Finally we conclude the paper in Section 8.

2. Related Work

The basis of our approach is the gradient mesh [BB13]. This vector graphics primitive smoothly interpolates colour through the use of bicubic patches that are logically aligned in a rectangular structure. The first appearance of this traditional type of gradient mesh was in Adobe Illustrator [Sys98]. Modified versions are now available also in Inkscape and CorelDRAW [BLHK18], and [BHEK21] introduced a version based on mesh colours. However, the regular topology requirement and lack of support for local refinement limit its usability and expressiveness, mostly due to an extensive number of patches that are generated. The regular topology restriction has been addressed and alleviated by using either generalised barycentric coordinates [LJH13, HBK19], loop subdivision surfaces [Loo87] or Catmull–Clark subdivision surfaces [LKSD17]. The latter approach, often called the *subdivision gradient mesh*, is especially useful as it has basically the same functionality as traditional gradient meshes, but with the added advantage of unstructured topology and increased smoothness.

A recent extension [VK18] supports exact geometry interpolation, hierarchical edits and sharp colour transitions, and is therefore more versatile than the earlier mentioned techniques. However, although local edits are possible in their method, they are evaluated and rendered using global subdivision. In contrast, we introduce a truly local approach not only for editing, but also for evaluation and rendering.

As is well known from the context of 3D modelling and animation [NLMD12], naive use of subdivision surfaces drastically influences memory and rendering performance. This naturally applies also in our context of subdivision gradient meshes, where interactive edits are indispensable. We borrow several techniques and ideas for real-time approximation of Catmull–Clark surfaces using hardware tessellation. Patches used in these techniques are called approximate Catmull–Clark (ACC) patches. Prime examples of such approximation schemes are ACC1 [LS08] using bicubic patches and ACC2 [LSNC09] using Gregory patches [Gre74, Lon87]. Generalised Gregory patches [HK18] can be be used to create a generalisation of ACC2 for arbitrary valency faces including their GPU treatment [HBK18]. The idea of FAS [NLMD12] allows for local subdivision near features. The OpenSubDiv library [Pix21] uses some of these elements to render subdivision surfaces efficiently using both the CPU and the GPU.

Our contribution focuses on real-time rendering of subdivision gradient meshes so that designers can interactively edit and manipulate them. Additionally, we also allow the possibility for hierarchical geometrical and colour edits following Verstraaten and Kosinka [VK18], whilst guaranteeing interactive rates of performance. Further novel features include user interface improvements and self-intersection prevention.

3. Preliminaries

We now detail some of the main building blocks of our method and its implementation, and define used terminology.

3.1. Mesh data structure

A mesh $\mathcal{M} = (V, E, F)$ is defined by a set of vertices V, a set of edges E, and a set of faces F. A vertex $v \in V$ generally contains several attributes, such as coordinates and colour. An edge $e \in E$ is directional and connects unique vertices v_i and v_j. More formally,

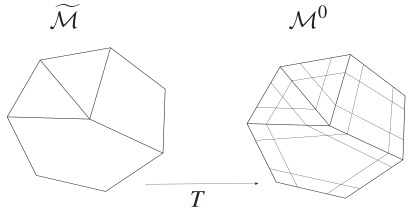

Figure 2: *An illustration of the ternary subdivision operator T.*

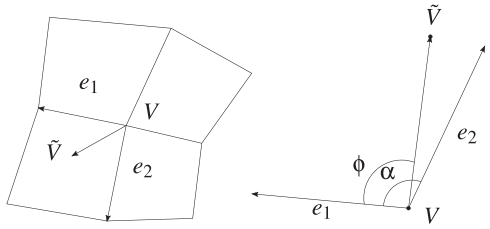

Figure 3: *Alternatives for expressing the refinement of geometry in a local frame.*

$E \subseteq \{(v_i, v_j) \in V \times V \mid i \neq j\}$. A face $f \in F$ is a minimal closed loop of edges and vertices where each subsequent pair of vertices is connected by an edge. In addition, a gradient mesh $\widetilde{\mathcal{M}}$ built on top of $\mathcal{M}$ includes a set of gradient vectors G. At each vertex, a gradient vector is assigned per incident edge. We assume that the mesh is manifold; see elsewhere [LKSD17, VK18] for more details.

A *regular vertex* is incident with quadrilateral faces only and has valency less than 4 if it is a boundary vertex, or valency equal to 4 if it is a non-boundary (internal) vertex. All other vertices are called *extraordinary*. A *regular face* is quadrilateral and is incident with regular vertices only. *Irregular* quadrilateral faces are incident with at least one extraordinary vertex, and faces of valency other than 4 are called *extraordinary*.

3.2. Ternary subdivision step

Prior to Catmull–Clark subdivision, a ternary subdivision step operator T is applied to $\widetilde{\mathcal{M}}$ to create the mesh $\mathcal{M}^0$ [LKSD17]. This operator T logically trisects each edge, and creates a smaller polygonal face for each original face, surrounded by a layer of quads; see Figure 2. This structure is useful as the colour assigned to a vertex in $\widetilde{\mathcal{M}}$ is interpolated in the Catmull–Clark limit when the one-ring neighbourhood of vertices in $\mathcal{M}^0$ corresponding to vertices in $\widetilde{\mathcal{M}}$ is assigned the same colour.

The geometry of $\mathcal{M}^0$ is obtained from $\widetilde{\mathcal{M}}$ as follows. For an edge given by V_i to V_j, its two new edge points V_{ij} and V_{ji} simulate the role of gradient handles in the traditional gradient mesh; they are initialised by $V_{ij} = (2V_i + V_j)/3$ and similarly for V_{ji}, and then adjusted by the user. Then for each vertex V_i and each of its incident faces F_j, the two edge-connected neighbours of V_i in F_j, the centroid of F_j and V_i itself are bilinearly interpolated to initialise the new face points (see Lieng *et al*. [LKSD17] for details), which can optionally also be adjusted by the user.

3.3. Subdivision gradient meshes

Arbitrary manifold topology gradient meshes can be defined by a modified Catmull–Clark subdivision scheme in both geometry and colour [LKSD17]. This method allows artists to manipulate the mesh and associated colour gradients at various levels of subdivision. Additional features such as interpolation of both geometry and colour, local detail following edits at coarser subdivision levels and support for sharp colour transitions were added as well.

The subdivision gradient mesh at subdivision level l is obtained by a single application of the ternary subdivision operator T followed by l applications of the Catmull–Clark subdivision operator C as

$$\mathcal{M}^l = C^l T \widetilde{\mathcal{M}}. \tag{1}$$

The colour component is simply stored as the colour chosen by the user. The geometry of the initial mesh $\widetilde{\mathcal{M}}$ is stored in global coordinates. Geometry edits at any level of subdivision are stored using one of the following approaches; see Figure 3. The first approach, for a vertex V edited towards a sector that corresponds to a face, stores an edited vertex $\tilde{V}$ as a displacement ΔV from the original vertex V. This displacement is locally expressed using the vectors e_1 and e_2 along the edges of the quadrant in which $\tilde{V}$ is located as $\Delta V = ae_1 + be_2$ for some a and b, which are then stored.

The second approach, for vertices V edited beyond the mesh boundary, stores the displacement ΔV as a relative angle and length with respect to the vectors e_1 and e_2 as

$$(\hat{\phi}, \hat{\rho}) = \left(\phi/\alpha, \|\Delta V\|^2/\sqrt{\|e_1\|^2\|e_2\|^2}\right). \tag{2}$$

Geometry edits at any level of subdivision are stored using these two approaches for boundary and non-boundary vertices as in Verstraaten and Kosinka [VK18], Section 4.2].

The meshes $C^l T \widetilde{\mathcal{M}}$ and $T C^l \widetilde{\mathcal{M}}$ are topologically equivalent, and therefore a natural mapping between the vertices of $C^l \widetilde{\mathcal{M}}$ and $\mathcal{M}^l$ exists. As colour edits should only affect disjoint one-ring neighbourhoods, the user is allowed to edit the colour of all vertices (or vertex sectors when using sharp colour transitions) in $\mathcal{M}^l$ that are topologically associated to $C^l \widetilde{\mathcal{M}}$.

Conventional gradient handles for a subdivision gradient mesh allow geometry edits of a subset of the vertices in $\mathcal{M}^l$ only. This unnecessarily limits the expressive freedom of the user, and therefore we allow the user to edit the geometry of all vertices in $\mathcal{M}^l$.

3.4. Feature adaptive subdivision

Our method is inspired by a computationally efficient method to evaluate the Catmull–Clark limit surface including boundaries up to machine precision [NLMD12], which adopts the idea of FAS. The mesh is iteratively subdivided only in the affected vicinity of irregular features, while the regular patches are always directly rendered as bicubic patches. Special transition patches are required to avoid cracks between adjacent patches of different subdivision levels. The FAS can be summarised as three stages [SRK*15].

Figure 4: *Five possible constellations for transition patches [NLMD12]. Patches belonging to the current subdivision level, next subdivision level and transition patches are coloured in green, red and yellow, respectively.*

CPU preprocessing aims to produce the mesh connectivity and identify features to apply adaptive subdivision to. The input to FAS is a base control mesh, which is composed of vertices, faces and optional data containing hierarchical details and semi-sharp crease edge tags. This preprocessing yields the patches to be tessellated and the computation of the relevant control points, which are stored in index buffers. For each level of subdivision, a subdivision table is set to store all the mesh data required for the FAS process. Index buffers store patch data, which describe the patch type and the indices of all the relevant control points. The base control mesh, patch index buffers and generated subdivision tables are then sent to the GPU for further processing.

Regular patches at each subdivision level are categorised as either a *full patch* that only shares edges with patches of the same subdivision level, or a *transition patch* that is adjacent to a patch that is further subdivided. Crack-free renderings can be obtained by evaluating adjacent patches at corresponding domain locations. One approach that ensures this splits each transition patch into several subpatches using a simple case analysis. There are five possible constellations for transition patches [NLMD12], Section 4]; see Figure 4.

During **FAS**, the base mesh is subdivided iteratively by running a number of GPU kernels. At each level, the control points and the subdivision tables of current level are used for the computation of the control points at next subdivision level. These control point data are stored in a control point buffer, which is generated at the preprocessing stage. This process repeats for each subdivision level until it reaches the pre-defined maximum level.

Finally, the **patch tessellation** stage sends the patches to the GPU tessellator unit using three patch types: regular, transition and irregular. The tessellator tessellates all the patches into triangles, which are then rasterised.

FAS subdivision depth depends on the given tessellation factor t. It performs $\lceil \log_2 t \rceil$ subdivision steps. For each subdivision level l, its factor is set to $\tilde{t} = \max(1, t/2^l)$. When $\tilde{t} = 1$, the patch is evaluated only at its corners using limit stencils of Catmull–Clark subdivision.

4. Feature-Adaptive and Hierarchical Subdivision Gradient Meshes

This section presents our novel method, which adapts the idea of FAS and approximate patches to the setting of subdivision gradient meshes. Our algorithm takes a coarse base control mesh data with gradient vectors (Section 3.1), including edges explicitly tagged as being sharp for sharp colour transitions, and hierarchical editing data. The output is an adaptively refined mesh. The mesh refinement algorithm stops when there are no more faces requiring further subdivision. Afterwards, the faces of the refined mesh are sent to the GPU to be rendered using bicubic Bézier patches or Gregory patches.

Figure 5: *A tulip model. Far left: A mesh at level 0 (after the ternary subdivision step). Left: Edited mesh at level 3 using global refinement leads to 5472 patches. Right: In contrast, using our feature-adaptive and hierarchical method, the mesh at level 3 requires only 699 patches. Far right: The rendered result (using our method).*

4.1. Approximate feature adaptive rendering

We developed an OpenGL/Qt-based experimental tool ourselves instead of using FAS in OpenSubdiv due to several reasons. Although our method is inspired by FAS, it is different from FAS in the strategy for terminating the subdivision. Furthermore, FAS in OpenSubdiv does not directly support some of the features needed in our context. Finally, our own implementation gives us full control over hierarchical editing, local updates, sharp transitions and self-intersection prevention.

The surface $\mathcal{M}^\infty$ obtained after an infinite number of Catmull–Clark subdivision steps is called the limit surface. This surface is almost everywhere C^2 continuous [Hav02], and over regular regions equivalent to C^2 tensor product bicubic B-splines [DS78, LSNC09]. Boundary loops converge to uniform cubic B-spline curves [DKT98]. Irregular regions containing extraordinary vertices are composed of an infinite set of polynomial patches, and therefore expensive to evaluate.

Similar to FAS, at each level, we determine which faces to further subdivide. Faces that do not require further subdivision are rendered using bicubic Bézier patches or Gregory patches at their terminating subdivision level. Other faces are subdivided further. This choice optimises performance while obtaining the best visual quality. See the example in Figure 5: The mesh generated by our method is less dense than by traditionally *globally* subdividing the whole mesh to the highest used subdivision level.

The memory requirements for global subdivision to level k are proportional to $4^k|F|$, where $|F|$ is the number of faces in the control mesh, and can therefore be computationally prohibitive. In our method, the memory used is proportional to the number of patches, which can be at different subdivision levels. This greatly reduces the computations and memory required; see Section 6.

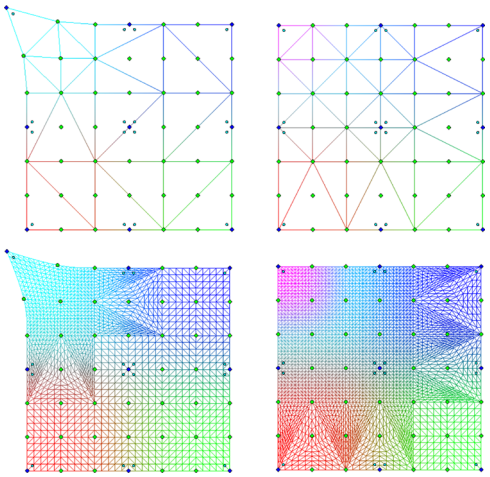

Figure 6: *Meshes resulting from feature adaptive rendering, after either a geometry edit (left) or magenta colour edit (right) of the top-left vertex in $\mathcal{M}^1$. Geometry edits affect their two-ring neighbourhood, and colour edits affect their three-ring neighbourhood. Quads are by default tessellated as two triangles. The bottom row shows the results at a higher level of tessellation.*

Our algorithm is also different to FAS in the subdivision termination condition. FAS keeps subdividing the mesh in the vicinity of irregular faces at each subdivision level until tessellation of the newly generated patches only amounts to evaluating the corner positions. In our algorithm, we introduce the concept of *affected faces* to determine at which subdivision level to terminate subdivision, as explained next.

4.2. Affected faces

A face is labelled as affected when it needs to be further subdivided. This happens for three reasons, as follows:

Edited faces. At each subdivision level, we determine which faces are affected by edits corresponding to the same or finer levels. At a specific subdivision level, a geometry edit indirectly affects a two-ring neighbourhood of faces around the edited vertex (this corresponds to the two-ring support of Catmull–Clark subdivision blending functions). Colour edits directly affect the one-ring neighbourhood (due to the colour spread step ensuring colour interpolation in the limit), and therefore indirectly affect the three-ring neighbourhood of faces around the edited vertex. Sharp colour edits produce the same effect, but only in the sector influenced by the edit.

A demonstration for both geometry and colour edits is shown in Figure 6. Note the reduced face count with respect to what global subdivision would produce. The transition patches, connecting faces at different subdivision levels as employed in FAS [NLMD12], ensure that patch edges are evaluated at identical parametric locations. The regular topology in this example ensures exact reproduction of the limit surface, which is simply a finite collection of bicubic patches.

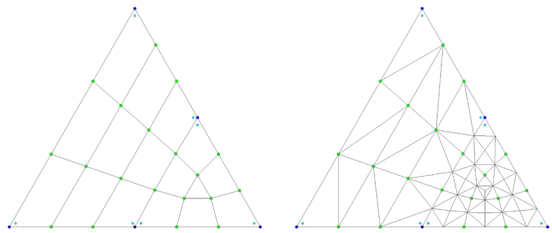

Figure 7: *Left: The mesh $\mathcal{M}^0$ obtained from an input mesh $\widetilde{\mathcal{M}}$ with one triangle with a quad next to it, after the initial ternary subdivision step. Right: $\mathcal{M}^1$, i.e.,$\mathcal{M}^0$ after one subdivision step. Note that only faces incident with the vertices of the extraordinary face are affected and thus subdivided.*

Faces affected by finer level edits are further subdivided locally. Taking a submesh of only the affected area and subdividing it produces incorrect results. This is due to the introduced boundaries and the size of Catmull–Clark subdivision stencils. This problem is solved by padding the affected area with a one-ring neighbourhood of faces before taking a submesh and subdividing it.

When taking a submesh, we cannot retain the assignment of its original half-edge indices without a loss in performance. As the edits are indexed via half-edge indices, we either require an explicit bookkeeping of global mesh indices to submesh indices, or we require the edits to be mapped to the submesh. Our index mapping strategy, detailed in Appendix A, allows the latter to be accomplished in constant time.

In the hierarchical setting, a face is considered affected when either the face itself is affected by an edit at its subdivision level, or the face topologically contains (i.e. is an ancestor of) an affected face of a finer level edit.

Irregular faces. Due to our mesh structure given by $\mathcal{M}^l = C^l T \widetilde{\mathcal{M}}$, newly generated non-quadrilateral faces are only contained in $\mathcal{M}^0$. These non-quadrilateral faces and their one-ring neighbourhoods are both considered affected (again owing to the Catmull–Clark subdivision rules applied there). In Figure 7, left, the mesh is $\mathcal{M}^0$ with one irregular face (bottom right), which has six direct neighbours. Thus, all of the seven faces are considered affected and then subdivided; see Figure 7, right.

Affected face cascading. The used approximating (bicubic and Gregory) patches tend to create minor artefacts along edges incident with extraordinary vertices; see Figure 8. FAS of one of these sectors may create discontinuity artefacts along the edges with faces at different levels. After obtaining the affected faces, we therefore iteratively cascade the affected faces around such irregular vertices. A demonstration of affected face cascading in the vicinity of irregular vertices is given in Figure 8. There are five sectors around the extraordinary vertex in the middle. We move the red control point close to the bottom left, which affects one sector around the extraordinary vertex. This one sector is refined more, while the other sectors are not refined. To prevent the artefacts, this level mismatch may cause, we refine all the sectors around the extraordinary vertex to the same level of subdivision. Then as we can see in Figure 8, far

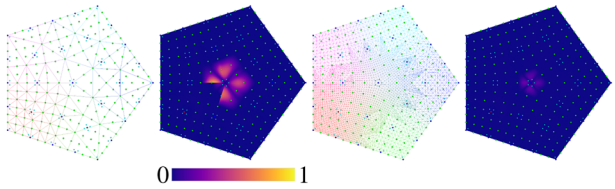

Figure 8: *Far left: The result of feature adaptive subdivision without irregular face cascading and minimal tessellation; the red control point close to the bottom left in $\mathcal{M}^2$ has been edited. Left: The resulting transition artefacts around the extraordinary vertex in the centre, shown via the plasma colour map [HDF*20]. The per-pixel difference in $[0,1]^3$ RGB space between our method and the Catmull–Clark limit surface is magnified 200 times to make it visible. Right: The mesh with irregular face cascading and an increased tessellation factor. Far right: Artefacts are greatly reduced by irregular face cascading.*

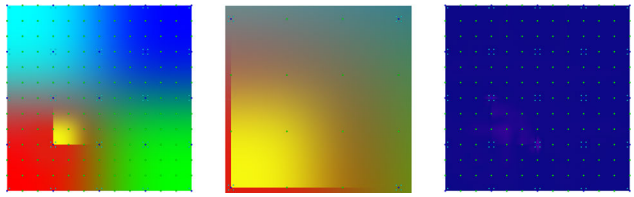

Figure 9: *Left: A colour surface with a sharp colour transition edit. Middle: A zoomed version of the edited area. Right: Artefacts corresponding to dart vertices are visible around the colour edit points in the top-left and bottom-right corner. The maximum difference in this example is 0.067% (wrt. the theoretical maximum of $\sqrt{3}$); the colour map and difference scaling are the same as in Figure 8.*

right, the sharp artefacts along edges incident with the extraordinary vertex are greatly reduced.

4.3. Patch rendering

We use the ACC2 scheme [LSNC09] to approximately render the Catmull–Clark limit surface given by the control (gradient) mesh using hardware tessellation. Our choice optimises performance while obtaining high visual quality. For each regular face in the mesh, a bi-cubic geometry/colour patch is constructed. These patches reproduce the Catmull–Clark limit surface exactly. As these patches are parametric, they are efficient to evaluate on GPU architectures with a programmable tessellation unit. For each irregular face in the mesh, a Gregory patch is constructed. These patches meet with G^1 continuity along edges incident with extraordinary vertices (in contrast to the C^0 continuity that is provided by ACC1 there). This method improves upon surface quality (Section 6) near extraordinary vertices at the expense of being slightly more computationally expensive compared to ACC1 [LSNC09]. Transition patches are handled as Niessner *et al*. [NLMD12], Section 4], see Figure 4.

4.4. Sharp transitions

All vertices along sharp transitions (often specified along chains of edges) are subdivided using the boundary rules, except its end-vertices (if any), often called *darts*, which are subdivided using the smooth rules [DKT98]. The boundary rules are only applied to the colour components of the control mesh whereas the geometry is smoothly subdivided. The approximating patches of ACC1/ACC2 were never intended to support such (colour) transitions. Although the associated artefacts turn out to be virtually invisible, a slightly improved approximation is achieved by treating the boundary/sharp edges adjacent to a dart as a complete boundary when updating its inner control points in a subdivision step. Figure 9 shows an example of sharp colour editing and the tiny artefacts resulting from using approximate patches to render them.

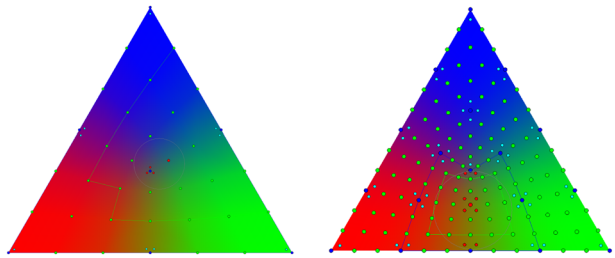

Figure 10: *Left: A visualisation of our brush editing functionality. The brush is indicated by the grey circle. The selected geometry and colour handles are shown as red bullets. Right: A visualisation of the region of influence of the currently selected handles: in green for geometry and in blue for colour.*

5. User Interface

Hierarchical editing in the context of gradient meshes was proposed in Lieng *et al*. [LKSD17] and further developed in Verstraaten and Kosinka [VK18]. It is worth noting that OpenSubdiv also supports hierarchical editing for subdivision meshes [Pix21], but for the reasons mentioned in Section 4.1, we rely on our own implementation. Improving upon Verstraaten and Kosinka [VK18], we have designed various user interface improvements that allow the artist to intuitively edit the gradient meshes in a hierarchical manner.

5.1. Editing

Handles. Existing gradient handles for a subdivision gradient mesh allow geometry edits of a subset of the vertices in $\mathcal{M}^l$ only. This unnecessarily limits the expressive freedom of the user, and therefore, we allow the user to edit the geometry of all vertices in $\mathcal{M}^l$. To this end, we require visual cues for vertices that can be edited in different ways; see Figure 10. Green bullets denote the vertices for which only geometry can be edited. Dark blue bullets define the vertices via/at which both geometry and colour can be edited. To facilitate sharp colour edits per sector, we introduce slightly smaller light blue bullets around the colour handles, which are offset in the direction of each incident sector. In our implementation, we only allow the smaller colour handles to be actually colour edited, which works well with our brushes as explained below.

The ternary subdivision step introduces an implicit three-ring separation between colour edit points, which remains the case after further Catmull–Clark subdivision. A naive approach is to use the mesh data structure and subdivision functions to explicitly keep track of these colour edit points. We present a better alternative that directly extracts the colour handles from a mesh after an arbitrary number of subdivision steps. We exploit the above-mentioned colour separation and the fact that existing vertices retain their indices during subdivision. We perform a depth-first search that is initiated with the indices of the vertices of $\mathcal{M}^0$, to account for potentially disconnected components. Only steps of three consecutive (half)edges in a single direction along the edges incident with the vertices are allowed. As a result, the points visited by the algorithm indicate the colour handles.

Brushes. As an alternative to individual editing at handles by direct selection, we introduce a more advanced brush; see Figure 10, left. All handles within a specified radius (adjustable by the user) from the cursor are automatically selected. A colour (that can be set interactively for the brush) is assigned to all brushed/selected colour handles while editing. Geometry edits are applied to the geometry handle closest to the cursor in the centre of the brush.

Region of influence. To provide the artist with an indication of the potentially edited region, we outline the region that would be influenced by editing the currently selected geometry (in green) and colour handles (in blue), as shown in Figure 10, right. All currently brushed colour handles and the closest geometry handle contribute to this affected area. As intended, sharp colour edits affect only specific sectors.

5.2. Interpolation and self-intersection prevention

A conventional user interface allows the user to directly edit handles at specific subdivision levels [LKSD17]. However, this approach affects the resulting limit surface in a counter-intuitive way, which is unlikely to fulfil the intentions of the artist. The reason is that, in general, control points are not interpolated in Catmull–Clark subdivision, and naive attempts to force interpolation lead to a global system of linear equations. However, as already utilised in Verstraaten and Kosinka [VK18], the initial ternary step allows for separation of these equations, which can then in turn be solved completely locally. Our improved user interface allows the user to directly edit the corresponding limit positions of the control points. Owing to unfortunate mistakes in the limit stencil inversion process in Verstraaten and Kosinka [VK18], we derive the correct formulas in Appendix B.

This (desirable) interpolation property and/or careless geometry editing may lead to self-intersections (fold-overs) in the limit surface, thereby compromising the validity of the intended design. We prevent self-intersections by employing the sufficient injectivity test of Gain and Dodgson [GD01], applied to bicubic patches in the mesh. A demonstration of the self-intersection prevention feature is presented in Figure 11. Although this feature works well, slight overlaps may occur at extraordinary vertices where ACC2 (Gregory) patches are rendered, as these are approximated by ACC1 (bicubic) patches for the injectivity tests.

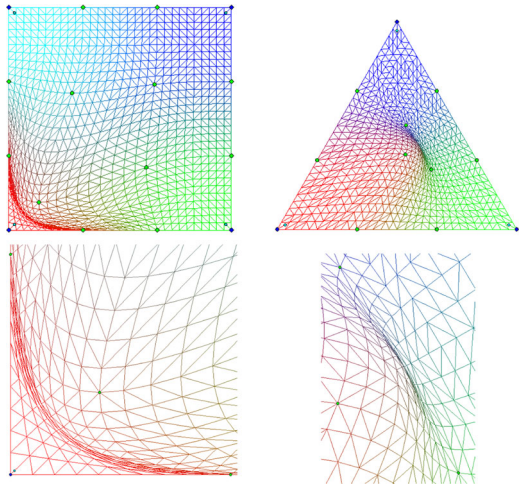

Figure 11: *A demonstration of the self-intersection prevention feature at tessellation level 10. A geometry handle is dragged in one direction as far as allowed by the method. For the quadrilateral model (left), the faces are just not overlapping at this point. For the triangular model (right), a very slight overlap might occur due to the ACC2 surface approximation. The bottom row shows insets.*

6. Results

We have designed several models using our subdivision gradient mesh, presented throughout the paper and the supplementary video. Some of them are simple and academic in nature to reveal the workings and features of our representation, and some are realistic examples to showcase the capabilities of our method. To evaluate the quality of our method, we compare our results to the corresponding Catmull–Clark limit surface both visually and statistically. The pixel-level differences are computed in the unit RGB cube, scaled up by 200 for visualisation purposes, and shown using the plasma colour map, as in Figure 8. When reporting the maximum difference as a percentage, it is with respect to the theoretical maximum of $\sqrt{3}$.

The cherry model presented in Figure 1 showcases the main features of our method: hierarchical editing. The stalk can be bent by moving only a few control points and a detail (water drop) can be easily added without having to refine the model globally. Along with the tulip model in Figure 5, it also shows the significantly reduced face count our method offers compared to previous techniques that rely on global subdivision. Further examples include the leek in Figure 12, the bowling pin in Figure 13, the beach ball in Figure 14, the butterfly in Figure 15 and the sunglasses in Figure 16.

Table 1 shows the generated face counts for the different models featured throughout this paper. Although this gives a good indication of the expected performance of our method, it is not only dependent on the number of faces. We make a distinction between bicubic and Gregory patches, and also list the number of transition patches (in brackets). To compare our method with previous methods [LKSD17, VK18], we also list the number of faces these global methods would require to represent the same models.

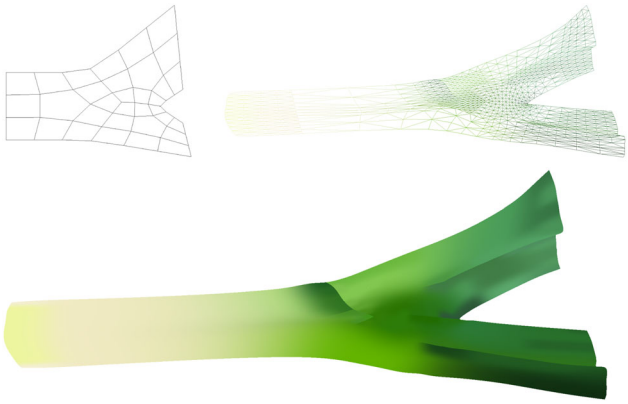

Figure 12: *An example of our method used to model a leek. Top left: A mesh of level 0 (after the initial ternary step). Top right: The mesh after 2 levels of hierarchical editing. Bottom: The final rendering of the leek model.*

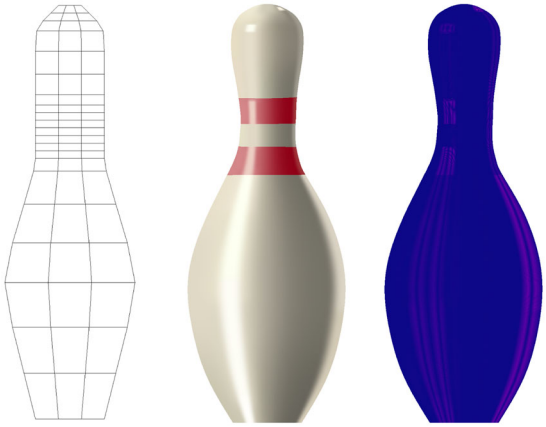

Figure 13: *A bowling pin model. Left: $\mathcal{M}^0$ after a ternary subdivision step on the initial mesh $\widetilde{\mathcal{M}}$. Middle: The edited result up to level 3. Right: A visual representation of the colour difference between our method and the Catmull–Clark limit surface; the maximum is* 0.31%.

We show simple triangular, quadrilateral and pentagonal meshes in Figure 18. On purpose, to demonstrate how our method smoothly blends colours, we assigned very distinctive colours to the colour handles (such as red, green, blue, cyan, magenta and yellow).

Sharp colour transitions and dart vertices were discussed in Section 4.4 and demonstrated in Figure 9. In cases where multiple sharp transitions meet, the difference with respect to the Catmull–Clark limit surface may become a bit larger, as shown in Figure 19. However, as these occur in very specific cases only, the difference is still relatively small, and the resulting colour surface is still smooth around the sharp transitions, we do not consider these problematic.

Performance. We list performance statistics in Figure 20 on two models: a simple hexagon model with no edits, and the complex and hierarchically edited cherry model of Figure 1. We compare

Figure 14: *A beach ball model. Left: $\mathcal{M}^0$ after a ternary subdivision on the initial mesh $\widetilde{\mathcal{M}}$. Middle: The edited result up to level 3. Right: Difference visualisation with the maximum of* 0.53%.

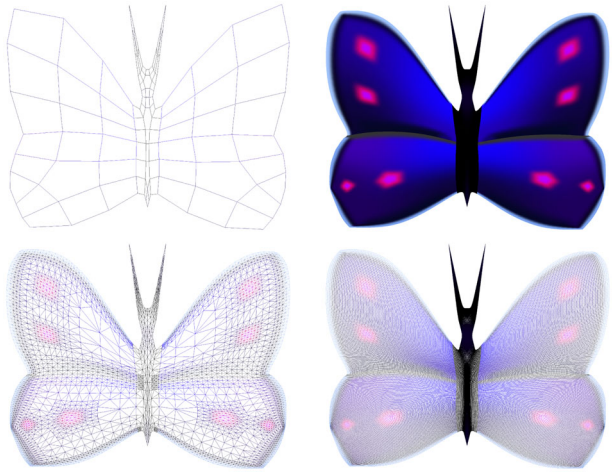

Figure 15: *A butterfly model. Top left: $\mathcal{M}^0$ after a ternary subdivision on the initial mesh $\widetilde{\mathcal{M}}$. Top right: The edited result up to level 5. Bottom left: The mesh using our method. Bottom right: The mesh of global subdivision.*

Table 1: *Generated patch counts for our models, broken down by the type of rendered patches. The numbers in brackets report the counts of transition patches. The last column gives the number of patches needed in case global/uniform subdivision is used.*

Model	Bicubic	Gregory	Total	Global
Cherry (Figure 1)	1322 (248)	49 (4)	1371	78336
Tulip (Figure 5)	642 (79)	57 (4)	699	5472
Leek (Figure 12)	930 (74)	21 (4)	951	2160
Beach ball (Figure 14)	312 (40)	24 (0)	336	6048
Bowling pin (Figure 13)	2085 (242)	0 (0)	2085	4032
Butterfly (Figure 15)	3708 (458)	33 (0)	3741	94464
Sunglasses (Figure 16)	1539 (199)	69 (6)	1608	11088
Pear (Figure 17)	3467 (827)	43 (1)	3510	313344

our method to existing approaches, both of which use global subdivision [LKSD17, VK18] for evaluation and rendering. The average timings include both rendering and CPU time calculations (of patches and their control points in case of our method), and are given in milliseconds. The listed measurements exclude our injectivity testing. When enabled, the injectivity test on the cherry model at subdivision level 5 takes approximately 7 ms. The reported mea-

Figure 16: *A sunglasses model. Top left: The edited result up to level 3 using global subdivision. Top right: The edited result up to level 3 using our method. Bottom left: The zoomed detail of the result using global subdivision. Bottom right: The zoomed detail of the result using our method.*

Figure 17: *A pear model. Top left: The final mesh up to level 6 using global subdivision. Top right: The final mesh using our method. Bottom left: The edited result up to level 6 using our method. Bottom right: The zoomed detail of the result using our method.*

surements were taken on a machine with an NVIDIA TITAN V GPU, 64 GB of RAM and an Intel XEON E5-2630 CPU.

Further performance statistics, showing both geometry and colour editing on the butterfly model (Figure 15) and the pear model (Figure 17) are shown in Figure 21. The figure shows that our algorithm's advantage increases with the subdivision levels used for

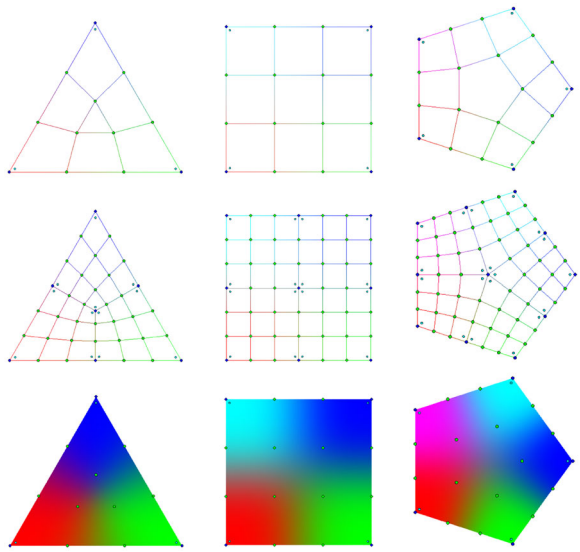

Figure 18: *From left to right are the control meshes and colour surfaces corresponding to a triangular, quadrilateral and pentagonal input mesh $\widetilde{\mathcal{M}}$, respectively. The top two rows present the limit projections of the control meshes $\mathcal{M}^0$ and $\mathcal{M}^1$, respectively. The bottom row presents the colour surfaces, including the $\mathcal{M}^0$ handles for visual guidance.*

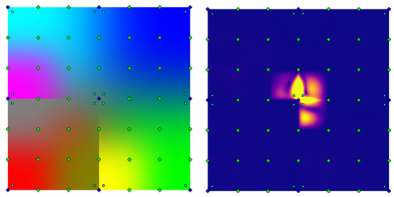

Figure 19: *Our result in the vicinity of multiple sharp colour edits. The magenta and yellow sharp colour edits create boundaries within the colour component of the surface that meet in the centre, creating an irregular vertex there. The maximum difference is* 1.50%.

adding details (such as those in the pear model near its dividing line; see Figure 17, bottom right).

7. Discussion

The design of our method mainly focused on reducing the needed face count and number of generated control points to achieve real-time performance, while preserving high visual quality. The achieved visual quality, which takes Catmull–Clark limit surfaces as ground truth, can be further improved at the expense of more rendered faces by marking all irregular faces as affected up to some desired level. In our experience and based on the examples presented throughout the paper, this is not required.

The triangular version [LS08] and the multisided version of ACC2 [HK18] are useful to fill the holes in the $\mathcal{M}^0$ meshes, but after a single subdivision step, no multisided holes remain, and thus

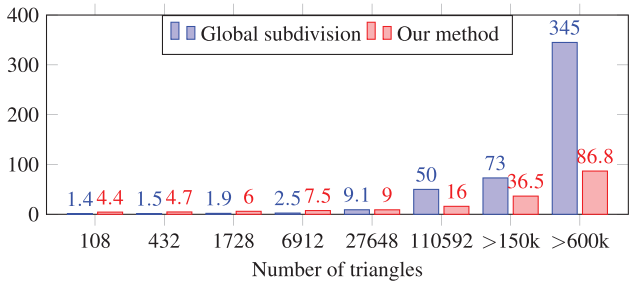

Figure 20: *Average frame time (in milliseconds) comparison between global subdivision and our method. The first six measurements correspond to a simple hexagon, for which the number of generated triangles is exactly the same for both methods. The last two measurements correspond to the cherry model of Figure 1(f) with approximately the same number of triangles (we set the tessellation levels in our method so that the numbers of generated triangles just exceed those for global subdivision at levels 5 and 6).*

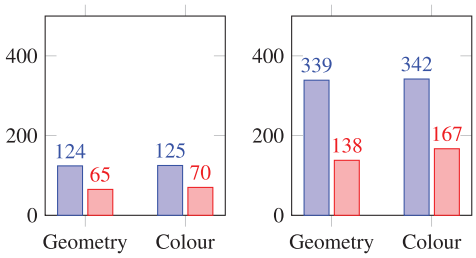

Figure 21: *Average frame time (in milliseconds) comparison between global subdivision (blue) and our method (red) on the butterfly model (left; subdivision level 5, more than 188k triangles) and the pear model (right; subdivision level 6, more than 626k triangles). Both geometry and colour editing timings are shown.*

these patches have limited usage in our setting. As they are also more computationally intensive, we have decided not to use them.

A comparison between bicubic and Gregory patches at extraordinary vertices is shown in Figure 22. While it may be the case that bicubic patches constructed by ACC1 provide a better Catmull–Clark limit surface approximation at vertices of valency 3, Gregory patches (of ACC2) provide better approximation at valencies 5 and higher. But more importantly, as discussed in Section 4.3, Gregory patches provide smoother colour surfaces with real-time performance, which justifies our choice to use them for all extraordinary vertex valencies.

The user interface shortcuts we introduced in our implementation and the brush editing functionality have allowed for a significantly quicker design process as whole swathes of control points can be edited at once. The visualisation of the affected area, explicit sector colour edit points and brushing improve user experience.

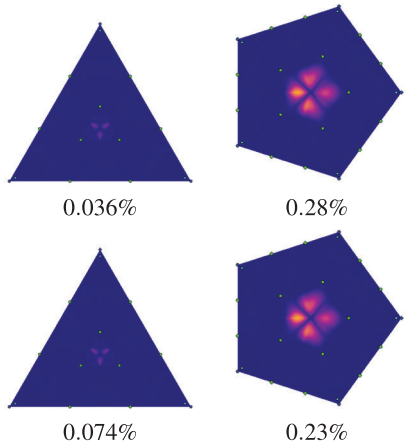

Figure 22: *A visual comparison between bicubic (top row) and Gregory patches (bottom row) at extraordinary vertices. The meshes come from Figure 18. The values below the difference images wrt. the Catmull–Clark limit surfaces report the maximum differences as percentages of* $\sqrt{3}$.

7.1. Limitations

One inconvenience of our method is creating the initial mesh of a model, which is also a necessary step in creating gradient meshes with other tools, such as Adobe Illustrator. The difference is that our approach allows for arbitrary manifold mesh connectivity, not restricted to regular rectangular arrays. This might be initially seen as a disadvantage, in that users may need to adjust their approach to (subdivision) gradient mesh design. At the same time, our approach opens the door for flexible (and locally adaptive) gradient mesh designs and more importantly image vectorisation techniques, potentially using only a single mesh.

As is the case with most, if not all, patch-based approaches, ours too may lead to incorrect geometries when the input mesh faces are not convex. One solution is to split such faces into convex ones; this is perfectly fine in our method since it supports arbitrary manifold topologies. Furthermore, our automatic fold-over detection system, when enabled, does not allow the user to create such meshes.

7.2. Future work

Concerning performance, the following topics require further attention. Our implementation recomputes the meshes and fully updates the graphics buffers after each edit, whereas a better alternative would update these structures only locally. The shaders could be optimised using for example table-driven approaches or the recent half-edge approach of Dupuy and Vanhoey [DV21]. Computationally demanding functionality like subdivision may significantly benefit from multicore processing. As these optimisations tend to significantly affect code complexity and maintainability, we have chosen to leave this as future work.

For geometry edits, a bulk editing feature may be convenient and is theoretically possible [GD01]. One would have to solve a local system of equations such that the limit projections of the vertices in

the original mesh are the desired vertices in the limit mesh. One potential drawback of such approaches is that the original mesh could be deformed quite severely in order to conform to the constraints.

Our work can be used to address the image vectorisation problem since our meshes exhibit a lot of (topological) flexibility, similar to that of meshes based on curved triangles [HEK21]. It would be a breakthrough if we could generate (feature-adaptive and hierarchical) subdivision gradient meshes automatically from raster images.

8. Conclusion

We have developed feature-adaptive and hierarchical subdivision gradient meshes, which support interactive editing and real-time rendering using hardware tessellation. Our method drastically reduces face counts while offering a nearly indistinguishable approximation with respect to global subdivision.

As a faster alternative for global subdivision, we implemented and adapted the ACC1 and ACC2 methods for the setting of subdivision gradient meshes, and borrowed ideas from FAS. A key aspect here was the design of a convenient and consistent index mapping between subdivision levels, allowing edits to be mapped to submeshes, which are subdivided only locally using our concept of affected faces.

In order to make sure that the colour surface is always valid, we have integrated a real-time self-intersection prevention mechanism to the geometry editing process. The improved user interface allows the designer to edit groups of control points at once and gives useful feedback of the effects of editing at different subdivision levels. All combined, our subdivision gradient meshes provide a significantly improved user experience over existing methods.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Appendix A: Index Mapping

In the setting of hierarchical edits and feature adaptive rendering, we require a convenient and consistent mapping between associated vertices, edges and faces across different subdivision levels. A convenient implicit mapping is obtained by a specific implementation of the Catmull–Clark subdivision step. Our mapping requires boundary halfedges in the input mesh to have indices that are greater than those of all interior halfedges. This requirement applies to the implementation of the initial ternary subdivision step and any subsequent (binary) Catmull–Clark subdivision steps.

We associate each interior halfedge e before subdivision to the face f_1 after subdivision; see Figure A.1. A consistent way of indexing the faces after subdivision is then given by $I(f_1) = I(e)$, where I returns the index. We associate each interior halfedge e before subdivision to the halfedges e_0, e_1, e_2 and e_3 after subdivision. A consistent way of indexing the interior halfedges after subdivision is then given by $I(e_i) = 4I(e) + i$.

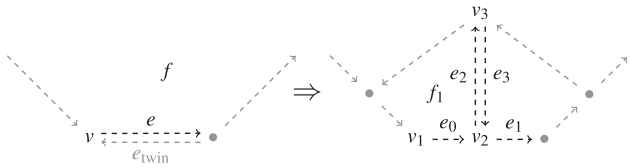

Figure A.1: *A schematic for consistent index mapping in Catmull–Clark subdivision. Not all halfedges are included to limit clutter.*

We associate each boundary halfedge e before subdivision to the halfedges e_0 and e_1 after subdivision. Let n_e be the number of interior halfedges before subdivision (which is equal to the sum of face valencies). A consistent way of indexing the boundary halfedges after subdivision is then given by $I(e_i) = 2n_e + 2I(e) + i$.

Vertex indexing is split according to the different types of generated vertices. A vertex v_i takes the index of v of the previous subdivision level. An edgepoint v_j that splits an edge $[e, e_{\text{twin}}]$ gets assigned $I(v_j) = n_v + \min(I(e), I(e_{\text{twin}})$ where n_v is the number of vertices of the previous subdivision level. Finally face points v_k that split a face f are assigned the index $n_v + \frac{n_e}{2} + I(f)$.

The introduced mapping strategies are convenient for the following reasons. The index of an interior parent halfedge e can now be obtained from the indices of the child halfedges e_0, e_1, e_2 or e_3 using integer division by 4. This allows for obtaining the affected faces at any level caused by finer level edits.

Further, the index i^n of a halfedge at subdivision level n can be expressed via the index i^m of a halfedge at level $m < n$ using hierarchically defined detail $0 \le d < 4$ as

$$i^n = 4(\dots(4(4(4i^m + d^{m+1}) + d^{m+2}) + d^{m+3})\dots) + d^n,$$

where a detail d^k indicates which child edge was chosen at subdivision level k: $d = 0$ for e_1, $d = 1$ for e_2 etc. The details d can be extracted from i^n in reverse order as they are the remainders after repeated division of i^n by 4, e.g. $d^{n-1} = \lfloor d^n/4 \rfloor$. These observations allow the following. Mapping edits towards a submesh (while preserving hierarchically defined detail) is allowed as halfedge indices i^m can be swapped. Besides this, edits can be mapped towards a subdivided mesh as $4i^m + d^{m+1}$ can be substituted by i^{m+1}. Furthermore, the affected area at a finer level can be mapped towards affected areas at coarser levels using repeated integer division by 4.

Appendix B: Inverse Limit Projection

Our user interface allows direct editing of vertices of the limit meshes. The geometry edits, however, are stored with respect to vertices of the control meshes. We therefore needed to invert the limit stencil/projection [LSNC09], Section 3.2]

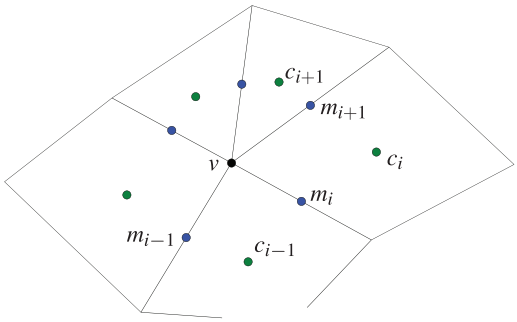

Figure B.1: *A schematic for a vertex v of valency n [LSNC09]. Edge midpoints are denoted by m_i and face centroids are denoted by c_i.*

$$v^\infty = \begin{cases} \dfrac{n-3}{n+5}v + \dfrac{4}{n(n+5)}\displaystyle\sum_{i=1}^{n}(m_i + c_i) & \text{if } v \text{ is a smooth vertex of valency } n; \\ \dfrac{v_j + 4v + v_k}{6} & \text{if } v \text{ is a crease/boundary vertex with sharp edges } v_jv \text{ and } vv_k; \\ v & \text{if } v \text{ is a sharp/corner vertex} \end{cases} \tag{B.1}$$

of Catmull–Cark subdivision; see Fig. B.1.

The inverse limit projection of a sharp/corner vertex is simply $v = v^\infty$, and that of a crease/boundary vertex reads $v = \frac{6v^\infty - v_j - v_k}{4}$. For smooth vertices, this is less trivial as the edge midpoints m_i and face centroids c_i depend themselves on v. The inverse limit projection of a smooth vertex v is then given by

$$v = \frac{v^\infty - \frac{4}{n(n+5)}\sum_{i=1}^{n}(\tilde{m}_i + \tilde{c}_i)}{\frac{n-3}{n+5} + \frac{4}{n(n+5)}\left(\frac{n}{2} + \sum_{i=1}^{n}\frac{1}{n_i}\right)}, \tag{B.2}$$

where $\tilde{m}_i = m_i - v/2$ and $\tilde{c}_i = c_i - v/n_i$ with n_i is the corresponding face valency.

References

[BB13] Barla P., Bousseau A.: Gradient art: Creation and vectorization. In *Image and Video-Based Artistic Stylisation*. London, UK: Springer (2013), pp. 149–166.

[BHEK21] Baksteen S. D., Hettinga G. J., Echevarria J., Kosinka J.: Mesh colours for gradient meshes. In *STAG: Smart Tools and Applications in Graphics*. P. Frosini, D. Giorgi, S. Melzi and E. Rodolà (Eds.). Geneve, Switzerland: The Eurographics Association (2021).

[BLHK18] Barendrecht P. J., Luinstra M., Hogervorst J., Kosinka J.: Locally refinable gradient meshes supporting branching and sharp colour transitions. *The Visual Computer 34*, 6–8 (2018), 949–960.

[CC78] Catmull E., Clark J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355.

[DKT98] DeRose T., Kass M., Truong T.: Subdivision surfaces in character animation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), New York, USA: ACM, pp. 85–94.

[DS78] Doo D., Sabin M.: Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 6 (1978), 356–360.

[DV21] Dupuy J., Vanhoey K.: A halfedge refinement rule for parallel Catmull–Clark subdivision. In *High-Performance Graphics 2021* (vol. *40*). N. Binder and T. Ritschel (Eds.). Geneve, Switzerland: The Eurographics Association (2021), pp. 1–14.

[GD01] Gain J. E., Dodgson N. A.: Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (2001), 289–298.

[Gre74] Gregory J. A.: Smooth interpolation without twist constraints. In *Computer Aided Geometric Design*. (1974), pp. 71–87.

[Hav02] Havemann S.: Interactive rendering of Catmull–Clark surfaces with crease edges. *The Visual Computer 18*, 5–6 (2002), 286–298.

[HBK18] Hettinga G. J., Barendrecht P. J., Kosinka J.: A comparison of GPU tessellation strategies for multi-sided patches. In *EG 2018—Short Papers*. O. Diamanti and A. Vaxman (Eds.). Geneve, Switzerland: The Eurographics Association.

[HBK19] Hettinga G. J., Brals R., Kosinka J.: Colour interpolants for polygonal gradient meshes. *Computer Aided Geometric Design 74* (2019), 101769.

[HDF*20] Hunter J., Dale D., Firing E., Droettboom M., the Matplotlib development team: Colormap reference. https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html (2020) [Accessed 16 January 2020].

[HEK21] Hettinga G. J., Echevarria J., Kosinka J.: Efficient image vectorisation using mesh colours. In *STAG: Smart Tools and Applications in Graphics*. P. Frosini, D. Giorgi, S. Melziand

E. Rodolà (Eds.). Geneve, Switzerland: The Eurographics Association (2021).

[HK18] Hettinga G. J., Kosinka J.: Multisided generalisations of Gregory patches. *Computer Aided Geometric Design 62* (2018), 166–180.

[LJH13] Li X. Y., Ju T., Hu S. M.: Cubic mean value coordinates. *ACM Transactions on Graphics 32*, 4 (July 2013), 1–10.

[LKSD17] Lieng H., Kosinka J., Shen J., Dodgson N. A.: A colour interpolation scheme for topologically unrestricted gradient meshes. *Computer Graphics Forum 36* (2017), 112–121.

[Lon87] Longhi L.: Interpolating Patches between Cubic Boundaries. Tech. Rep. T.R. UCB/CSD 87/313, 1987.

[Loo87] Loop C.: Smooth Subdivision Surfaces based on Triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

[LS08] Loop C., Schaefer S.: Approximating Catmull–Clark subdivision surfaces with bicubic patches. *ACM Transactions on Graphics (TOG) 27*, 1 (2008), 8.

[LSNC09] Loop C., Schaefer S., Ni T., Castaño I.: Approximating subdivision surfaces with Gregory patches for hardware tessellation. *ACM Transactions on Graphics (TOG) 28* (2009), 151.

[NLMD12] Nießner M., Loop C., Meyer M., Derose T.: Feature-adaptive GPU rendering of Catmull–Clark subdivision surfaces. *ACM Transactions on Graphics (TOG) 31*, 1 (2012), 6.

[Pix21] Pixar: Open subdiv. https://graphics.pixar.com/opensubdiv/docs/hedits.html (2021).

[SL17] Svergja J. K., Lieng H.: A gradient mesh tool for non-rectangular gradient meshes. In *Proceedings of the ACM SIGGRAPH 2017 Posters* (2017), ACM, pp. 59.

[SLWS07] Sun J., Liang L., Wen F., Shum H.-Y.: Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (TOG) 26* (2007), ACM, 11.

[SRK*15] Schäfer H., Raab J., Keinert B., Meyer M., Stamminger M., Nießner M.: Dynamic feature-adaptive subdivision. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (2015), pp. 31–38.

[Sys98] Systems A.: *Adobe Illustrator 8.0 Classroom in a Book*. San Francisco, USA: Adobe Press, 1998.

[VK18] Verstraaten T. W., Kosinka J.: Local and hierarchical refinement for subdivision gradient meshes. *Computer Graphics Forum 37* (2018), 373–383.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Video.