Volume 0 (1981), Number 0 pp. 1–12 COMPUTER GRAPHICS *forum*

Stylized Image Triangulation

Kai Lawonn[1] and Tobias Günther[2]

[1]University of Koblenz-Landau, Germany
[2]ETH Zurich, Switzerland

Figure 1: Stylized image triangulations that were created with our optimization-based framework. From left to right: a still life (constant color with hatching), river photograph (constant color with hatching), a concert (constant color with texture) and a giraffe (linear color gradients).

Abstract

The art of representing images with triangles is known as image triangulation, which purposefully uses abstraction and simplification to guide the viewer's attention. The manual creation of image triangulations is tedious and thus several tools have been developed in the past that assist in the placement of vertices by means of image feature detection and subsequent Delaunay triangulation. In this paper, we formulate the image triangulation process as an optimization problem. We provide an interactive system that optimizes the vertex locations of an image triangulation to reduce the root mean squared approximation error. Along the way, the triangulation is incrementally refined by splitting triangles until certain refinement criteria are met. Thereby, the calculation of the energy gradients is expensive and thus we propose an efficient rasterization-based GPU implementation. To ensure that artists have control over details, the system offers a number of direct and indirect editing tools that split, collapse and re-triangulate selected parts of the image. For final display, we provide a set of rendering styles, including constant colors, linear gradients, tonal art maps and textures. Lastly, we demonstrate temporal coherence for animations and compare our method with existing image triangulation tools.

This is the authors preprint. The definitive version is available at https://onlinelibrary.wiley.com/ and at https://doi.org/10.1111/cgf.13526.

1. Introduction

In the arts, image triangulations have recently been celebrated for combining raw geometry with organic hand-drawn structures to provide abstract, yet characteristic renditions of photographs and illustrations. As such, they raised noticeable interest in the artistic communities, ranging from exhibitions and applications in art galleries, several music videos, magazines and in public places [Bry17, Puc08, Yun13]. To create an image triangulation, the artist divides the image into triangles, which receive the average color (or a linear gradient) that approximates the covered portion of the image. An artist has three central goals when forming a triangulation: (a) to place details where needed, (b) to approximate the original image as best as possible, and (c) use as few triangles as possible. Since the manual creation of such a triangulation is tedious, a number of applications have been developed that assist in the generation of image triangulations [Ham12, Fis16, Bór17, Puc08, Oll15, Leu15, Yun13, SM16, Con16]. However, they all have in common that they only assist in goal (a): they automatically place vertices at image features and allow manual editing of a subsequently created Delaunay triangulation. To the best of our knowledge, the approximation quality (b) of an image triangulation has not been formally assessed and optimized.

In this paper, we propose a novel method to automatically generate image triangulations, which considers both the image features and the approximation quality in terms of the root mean squared distance to the underlying image. We formulate the triangulation

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

process as an energy minimization problem that automatically adjusts the vertex positions of the triangulation. The triangulation is iteratively refined by incrementally inserting further triangles until desired approximation criteria are met. The computation of the energy gradients is thereby an expensive calculation, for which we propose an efficient rasterization-based GPU implementation. To ensure artistic control over details, we provide direct and indirect manipulation tools that allow the artist to insert triangles, remove vertices, flip edges and re-triangulate selected regions. In our work, we collect several rendering styles, which include constant colors, linear gradients, as well as tonal art maps and textures. Furthermore, we show the potential of our work by applying the method to videos. Overall, this work comprises the following contributions:

- We formulate the image triangulation process as a minimization problem. The triangulation is optimized by moving the vertex positions such that the approximation error is reduced and we adaptively insert vertices in areas of high approximation errors.
- We provide an efficient rasterization-based approach to calculate the gradients of the approximation errors on the GPU.
- We provide interactive tools for the editing of the resulting image triangulation to offer the artist control over details.
- We collect several rendering styles for the triangles, including constant colors, linear gradients, tonal art maps and textures.
- We apply our method to video data and obtain frame-coherent video triangulations.

For the first time, artists are able to improve the approximation quality of a triangulation automatically. As we show later, our method leads to better results compared to existing work, both quantitatively and qualitatively. We refer to Fig. 1 for several examples of image triangulations that were generated with our tool.

2. Related Work

Much research aimed at stylization of images and videos [KCWI13], including abstraction through stylization [DS02], stippling [Sec02], pixel image abstraction [GDA*12] and diffusion curves [OBB*13]. In this paper, we focus on work that partitions an image into structures with constant colors. This comprises methods that subdivide an image into triangles, as well as images that are modified based on the L_0 gradient minimization and image segmentation.

2.1. Image Triangulation

For different stylized image representations, Grundland et al. [GGD08] sampled points based on image characteristics and calculated Delaunay triangulations as an intermediate step before applying various procedural or geometric rendering styles. The edges of the triangulation do not match the image characteristics and do not follow the objectives of our minimization. While the main goal of Grundland et al. was to generate different stylized image representations, the focus of our paper is to generate triangles only, which approximate the overall impression of the image. A number of image triangulation projects can be found online in form of commercial products or open source implementations. These projects have in common that points are placed at features and afterwards a Delaunay triangulation is performed. First, we concentrate on online tools that are available for free. Bórquez [Bór17] offered three algorithms that all detect edges, place vertices on those edges and construct a Delaunay triangulation. Two user parameters were provided: one for Laplacian smoothing of the image and one that indirectly adjusts the number of vertices. This method prioritizes strong edge features, tends to generate large triangles and provides limited artistic control. Triangulate 7 of Conceptfarm [Con16] places feature points more uniformly, which generates more well-behaved triangles. Their image blurring is edge preserving and they offered the option to manually delete points. Hamamuro [Ham12] adjusted the density of triangles to the image features, creating many small triangles along edges and a coarse triangulation in large smooth areas. This method is fully automatic. Fischer [Fis16] extended the work of Hamamuro [Ham12]. The triangles produced by this extension do not follow image edges well and vertices are only placed in regions with sufficient image features (using thresholding), which can produce a noticeable discontinuity in gradients, as demonstrated later. This tool provides an interesting stylization parameter, which creates elongated triangles to purposefully resembles strokes. This, however, produces overlapping triangles.

Polygonian by Leung [Leu15] is a mobile app that allows users to manually move, add and remove vertices that were initially placed automatically. Similar to Hamamuro [Ham12], the method adapts well to image features. Trimaginator by Ollivier [Oll15] is a commercial mobile app that provides editing tools and a collection of rendering styles that change the triangle and edge color of the resulting image triangulation. The tool not only places vertices according to image features, but also provides a radial layout. Polyshaper by Marin [SM16] initializes the triangulation either from an edge sampling or from a Poisson disc sampling. In addition, triangles can be placed manually and edge and triangle colors can be color-shifted and globally edited. Dmesh by Yun [Yun13] offers manual editing of triangulations, a parameter for controlling the density of triangles and an automatic placement according to image edges. Their tool further supports batch processing for the generation of videos, which however, are not temporally coherent. All methods above display triangles with constant colors. Delaunay Raster by Puckey [Puc08] also uses color gradients in the triangulations.

Besides all previously mentioned techniques that aim to generate a triangulated image from an aesthetic perspective, work has been done that generates triangles for various other reasons. Hoppe [Hop96] introduced a mesh simplification method that reduces the number of triangles of a given surface such that the geometry is preserved. Starting from an initial triangulation of an image, his method can be applied to reduce the number of triangles. Later, Garland and Heckbert [GH97] improved the simplification scheme by using an error metric. Cohen et al. [CDHM11] employed a greedy refinement procedure for the generation of a triangulation. Their method produces a hierarchy of triangulations. Yang et al. [YWB03] presented a novel idea for image compression. For this, they first detected the most significant pixels by applying a modified version of the Floyd-Steinberg algorithm and afterwards, these pixels were used to create a Delaunay triangulation. Finally, the color of the triangles was linearly interpolated with regard to the underlying grayscale image. Demaret and Iske [DI04] also followed the idea to construct a triangulation of the image by applying a Delaunay triangulation. Again, the most significant pixels need to be determined, which was calculated by using adaptive thinning, which was first in-

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

troduced to approximate large set of scattered data. Adams [Ada11] proposed a mesh generation framework to triangulate an image, which achieved lower computational and memory complexity. Their method produces comparable results. Later, Adams [Ada13] was able to produce better results at lower computational cost. Tu and Adams [TA13] extended the method by introducing a novel method to represent discontinuities at image edges. After an initial triangulation was computed, image edges were identified and points were inserted such that these edge were represented by triangle edges. Mostafavian and Adams [MA15] employed an optimization algorithm to reduce the approximation error in the reconstructed image. Experiments showed that their technique produces better results compared to the previously mentioned methods.

To the best of our knowledge, there is no method that uses an energy minimization to optimize the approximation quality by moving the vertices of an image triangulation. However, the previously mentioned approaches could serve as input for our method. Afterwards, the energy could be minimized by applying our framework.

2.2. L_0 Gradient Minimization

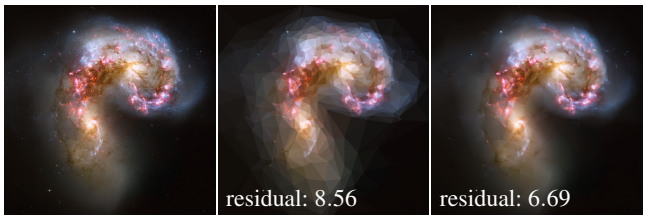
Xu et al. [XLXJ11] presented the L_0 gradient minimization method to obtain regions with constant color by letting the derivatives vanish. Their method fits an approximated image by minimizing the L_0 gradient, i.e., the non-zero gradient. Thus, it fits an image with reduced differences in color of neighboring pixels. Cheng et al. [CZL14] built up on the method by Xu et al. They employed a fused coordinate descent framework to obtain better results. Storath et al. [SWD14] introduced an optimization scheme that approximates L_0 gradient minimization based on dynamic programming and an alternating direction method of multipliers. An edge-preserving image smoothing approach was presented by Min et al. [MCL*14]. They achieved high-quality results with significantly improved runtime by minimizing a global objective function. This is accomplished by solving a sequence of 1D subsystems. Zhang et al. [ZXJ14] used a weighted median filter to imitate the L_0 filter, which reduces the computation complexity per pixel from $O(r^2)$ to $O(r)$ with r being the kernel size. Nguyen and Brown [NB15] presented an approach to minimize the L_0 gradient with improved runtime. They employed a region fusion method that unites regions of similar color to obtain regions of constant color. We focus on image partitions into triangles.

2.3. Image Segmentation

Image segmentation partitions an image into different classes, which can be represented with constant color. The area of image segmentation is a broad field that has grown over the last years. It comprises methods based on thresholding [RC78, Ots79], clustering approaches [LM99, CQM*15], histograms [OPR78, AA06], region-growing [HP74, NN04] and graph-partitioning [GPS89]. For more information about image segmentation, we refer the reader to recent surveys [ZA15, SD15, ZL16].

3. Optimization-based Image Triangulation

A key criterion for a successful image triangulation is its approximation quality of the original image. Thus, in the following, we formally pose image triangulation as an energy minimization problem that reduces the approximation error. Afterwards, we describe how to carry out the minimization, how to initialize and refine the triangulations and how users can interact in the process.



Original | Constant color | Linear gradients

Figure 2: Triangulation of Antennae galaxies (left) with 1500 triangles, using constant colors (middle) and linear gradients (right).

3.1. Problem Formulation

Given is an image $I : U \subset \mathbb{R}^2 \to C$, where U describes the image domain and C is the color space, e.g., an RGB triplet or gray value.

Subdivision into Triangles. Our goal is to approximate an image using triangles. Thus, we first subdivide the image domain U into triangles $\mathscr{T}$. Formally, the set of all triangles $\mathscr{T}$ is a simplicial complex that consists of 2-simplices (and their faces) such that the union of the triangles in $\mathscr{T}$ is U:

$$\bigcup_{T \in \mathscr{T}} T = U \tag{1}$$

Note, for every $T_i, T_j \in \mathscr{T}$ with $T_i \neq T_j$, the intersection $T_i \cap T_j$ is either empty, a 0-simplex (point), or a 1-simplex (line). Thus, adjacent triangles are not disjoint. Instead, they share the edges.

Triangle Colors. The color f of triangle $T \in \mathscr{T}$ shall approximate the underlying portion of image I that is covered by the triangle. We represent f as polynomial that is defined over the domain of T. We provide two options; a constant and a linear approximation:

$$\text{constant:} \quad f(x,y) = c \tag{2}$$

$$\text{linear:} \quad f(x,y) = ax + by + c \tag{3}$$

The coefficients a, b and c can be computed by minimizing the squared triangle approximation error $E(T)$:

$$E(T) = \frac{1}{2} \int_T (I(\mathbf{x}) - f(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x} \;\; \to \;\; \min \tag{4}$$

Varying the polynomial degree of the bivariate polynomial $f(x,y)$ gives rise to different approximation accuracies. Eqs. (2) and (3) constitute a constant color and a linear gradient, which are shown in Fig. 2. We refer to Appendix A for the computation of a, b and c. Higher-order polynomials are imaginable as well. For artistic reasons, we decided to remain with constant and linear styles for a simplifying rendition. Note that although we represent a constant and a linear representation of the triangle, we are able to define different stylizations based on this, see Sec. 5.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Optimal Triangulation. We consider the optimal vertex positions of a given triangulation to be the ones that minimize the sum of the triangle approximation errors, i.e., the triangulation approximates image I, and also minimizes an additional regularization term $p(T)$:

$$E = \sum_{T \in \mathscr{T}} E(T) + \lambda p(T) \;\rightarrow\; \min \tag{5}$$

The regularization term is detailed later and will allow the user to enforce certain artistic constraints, such as the fairness of the triangle shape. Thereby, parameter λ is a user parameter that steers how much the position of the vertices are regularized, see Sec. 6.2-Regularization. Note that Eq. (5) is an energy that is defined on a very high-dimensional space: the unknowns are not only the locations of the vertex coordinates of the triangles, but also the number of triangles is initially unknown. In the following sections, we introduce a reformulation and an iterative approximation that help us to minimize E.

3.2. Minimization

For now, we consider the number of triangles n to be fixed. We explain in Section 3.4 how n can be iteratively adapted. The remaining degrees of freedom are the vertex locations of the triangulation.

Reformulation. In the following, we denote the set of all vertices of a triangulation as $\mathscr{V}$. A first observation is that the movement of a single vertex $\mathbf{v} \in \mathscr{V}$ only affects the approximation errors of the adjacent triangles $\mathscr{A}(\mathbf{v}) = \{T \in \mathscr{T} : \mathbf{v} \subset T\}$. In other words, the impact of a variation of $\mathbf{v}$ is local. This insight leads us to a reformulation of Eq. (5) into a sum over vertices:

$$E = \sum_{\mathbf{v} \in \mathscr{V}} E(\mathbf{v}) \;\rightarrow\; \min \tag{6}$$

where $E(\mathbf{v})$ is the energy (or error) at a vertex:

$$E(\mathbf{v}) = \sum_{T \in \mathscr{A}(\mathbf{v})} \frac{E(T)}{3} + \lambda p(\mathbf{v}) \tag{7}$$

with $p(T) = \frac{1}{3}\sum_{\mathbf{v} \in T} p(\mathbf{v})$. Note that each triangle is visited three times, as we iterate over all adjacent triangles of the three corners. Hence, the division by three. The key property of Eq. (6) is that E is minimized when all summands $E(\mathbf{v})$ are minimized, since $E(\mathbf{v})$ is positive. Thus, E can be minimized for a fixed topology by optimizing the vertices $\mathbf{v} \in \mathscr{V}$ of the triangulation *in independent threads*. Here, $p(\mathbf{v})$ models the regularization term per vertex.

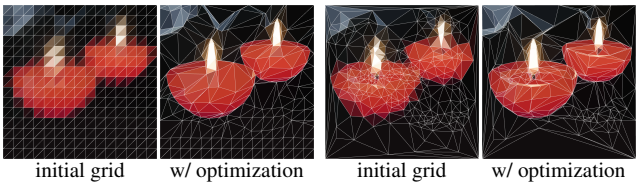
Regularization. Our optimization will assume that each vertex is always within its 1-ring, i.e., within the convex hull of vertices that are connected by an edge. Formally, the 1-ring is:

$$\mathscr{N}_1(\mathbf{v}) = \{\mathbf{w} \in (\mathscr{V} \backslash \mathbf{v}) : \mathbf{w} \in \mathscr{A}(\mathbf{v})\} \tag{8}$$

If this property is violated, degenerate or overlapping triangles might occur. To move vertices towards the center of their 1-ring, we employ a uniform Laplacian smoothing:

$$p(\mathbf{v}) = \frac{1}{2\,|\mathscr{N}_1(\mathbf{v})|} \sum_{\mathbf{w} \in \mathscr{N}_1(\mathbf{v})} (\mathbf{w} - \mathbf{v})^2 \tag{9}$$

Note that we define $\mathbf{w}^2 \coloneqq \langle \mathbf{w}, \mathbf{w} \rangle$ as the dot product. By default, we set the influence of the regularization to $\lambda = 1 \times 10^{-3}$, which makes this a subtle effect. We refer to Nealen et al. [NISA06] for alternative approaches that are popular on 2D manifolds.



(a) Regular layout (b) Importance sampling by saliency

Figure 3: Initial triangulations and their subsequent vertex optimization without refinement: regular grid (left), saliency (right). Here, with an equal number of vertices. Note that the saliency approach adapts to image features, such as the wicks and reflections.

Gradient Descent. For minimization, we use a gradient descent with step size h and an out-of-place update to avoid any dependence on ordering:

$$\mathbf{v} \leftarrow \mathbf{v} - h\frac{\mathrm{d}E(\mathbf{v})}{\mathrm{d}\mathbf{v}} \tag{10}$$

Further, we employ a trust region of 0.2 pixels (empirically chosen) that clamps the maximum change in $\mathbf{v}$ per iteration. Optionally, the trust region can decrease over time to prevent oscillation around the minimum. Section 4.1 elaborates on the efficient rasterization-based computation of the energy gradients $\frac{\mathrm{d}E(\mathbf{v})}{\mathrm{d}\mathbf{v}}$ in Eq. (10) on the GPU.

3.3. Initial Image Triangulation

The energy-minimizing method receives a triangulation as input, which is iteratively improved by moving its vertices. We provide the user two options to generate the initial triangulation: a *regular layout* and a Delaunay triangulation of an *importance sampling* of the initial vertices according to a given probability distribution.

Regular Layout. The regular layout places the vertices of the triangulation on a regular grid. To form the triangles, adjacent vertices are horizontally and vertically connected and for each grid cell a diagonal from bottom left to top right is inserted. Fig. 3a gives an example. The resulting triangulation guarantees that the valence (number of vertices on the 1-ring) is not higher than 6. A small valence can be helpful, since then a vertex influences fewer triangles and acts thus more locally.

Importance Sampling. Alternatively, the user may provide a probability distribution to guide the initial placement of vertices. The probability distribution may either be hand-drawn or it can be automatically derived, e.g., from a saliency map [HHK12] that estimates the visual importance of image features. The possibility to prescribe a general probability distribution gives the artist complete control over the sampling process. For importance sampling, we use the inverse CDF method [PJH16] in 2D. Afterwards, a Delaunay triangulation is applied to form the initial triangles, see Fig. 3b.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

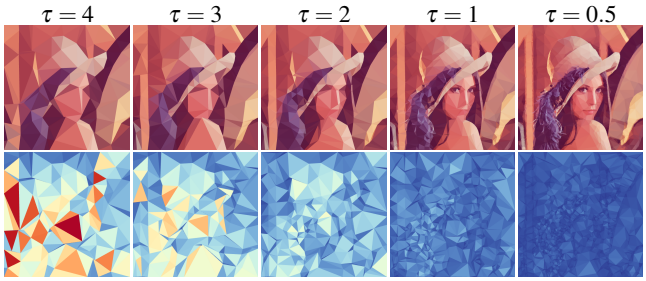


Figure 4: Triangulations (top) and plots of $\sqrt{E(T)/|I|}$ (bottom) for refinement thresholds τ. Blue color represents a low error, whereas a red color is used for a high error. Number of triangles $|\mathscr{T}|$ and the total summed up approximation error R are listed below.

3.4. Refinement Strategy

In Section 3.2, we assumed that the number of triangles was constant during the vertex updates. In the following, we lift this limitation.

Adaptive Refinement. We provide the user with the option to automatically refine triangles. When the root mean triangle approximation error $\sqrt{E(T)/|I|}$ exceeds a user-defined threshold τ, we insert a new vertex at the centroid of the triangle. This way, our method can start from a coarse triangulation (e.g., two triangles) and eventually adapts to all image features. Automatic image triangulations and their approximation errors are shown in Fig. 4 for various thresholds τ. We employ a refinement strategy that retains a Delaunay triangulation to keep the triangles as equi-angular as possible. Fortunately, there are multiple methods available to efficiently insert a point into an existing Delaunay triangulation. The option we used is to restore the Delaunay triangulation with incremental edge flips [ES96]: Whenever the sum of the two angles that are opposite to an edge exceeds 180°, an edge flip is performed until no more such edges are found. We perform this update on the CPU. Another option is the Bowyer-Watson algorithm [Bow81, Wat81], which first removes all triangles for which the inserted point is inside the circumcircle of the triangle and afterwards retriangulates holes.

Non-Delaunay Triangulations. Note that the incremental update of vertex positions in Eq. (10) can lead to triangulations that are no longer Delaunay triangulations, since triangles might deform. This is desired and necessary to adapt triangles to image features. In practice, we alternate between refinement and vertex optimization until the triangulation captures the details sufficiently. Despite the regularization term, it can happen that vertices collapse onto each other. For this reason, we remove triangles that have a small area by performing an edge collapse on their shortest edge.

3.5. User Interaction

During the energy minimization or after the algorithm terminates, we allow the user to influence the triangulation. Since our target users are artists, we provide both easy to use high-level tools, as well as methods to precisely edit individual triangles.

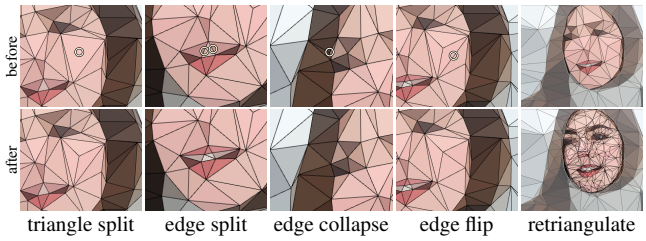


Figure 5: For artistic precision, we provide basic mesh editing operations and allow the user to retriangulate manually drawn regions.

Detail Editing. A central goal of our framework is to provide the user precise control. Thus, we implemented several mesh editing operations, which are illustrated in Fig. 5. The *edge split* (left click on edge) and *triangle split* (left click on triangle interior) allow the artist to add new vertices. An *edge collapse* (right click on vertex) collapses the selected vertex onto its nearest neighbor, which allows to remove triangles. Further, an *edge flip* (right click on edge) connects the opposite vertices of two adjacent triangles. Finally, the vertex order of a triangle can be changed (left click with shift key pressed), which determines the edge that textures are aligned with.

Brush. Refining larger areas can be tedious, when each vertex must be placed or removed manually. For this reason, we provide a brush tool, which inserts vertices in a circular area around the cursor. If artists intend to increase the triangle density, for instance to guide the attention of the viewer or to approximate visually important regions more accurately, the brush is easily applied. The radius is adjusted by scrolling the mouse wheel. Inside the circle, points are sampled at a given rate (we used 10 per second) according to a given probability distribution (uniform or normal distribution). At the sampled point, a vertex is inserted (similar to left click) or removed (right click) using the detail operations described above.

Retriangulation. Since the above brush operates only locally, we further allow the artist to draw a selection mask in which vertices can be resampled at once, which allows the user to uniformly increase or decrease the vertex density in a selected region. Using the brushes above, the artist first draws a selection mask onto the image. When pressing a button, all vertices inside the selected region are deleted and a user-defined number of vertices is uniformly sampled. The new vertices are Delaunay triangulated and connected to the triangulation outside the selected region, see Fig. 5 (right).

4. Implementation

In this section, we introduce an efficient rasterization-based approach to compute the energy gradients in Eq. (10).

4.1. Numerical Approximation of Energy Gradient $\frac{\mathrm{d}E(\mathbf{v})}{\mathrm{d}\mathbf{v}}$

We compute the energy gradient $\frac{\mathrm{d}E(\mathbf{v})}{\mathrm{d}\mathbf{v}}$ of a vertex by determining how a variation of vertex $\mathbf{v}$ affects the adjacent triangles. For this, we need the energy gradients of the adjacent triangles, cf. Eq. (7). The energy gradient of a triangle $\frac{\mathrm{d}E(T)}{\mathrm{d}\mathbf{v}}$ requires first the identification and subsequently the variation of the respective corner using finite

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

differences. Let $\{\mathbf{a}_T, \mathbf{b}_T, \mathbf{c}_T\} = T \cap \mathscr{V}$ be the three corners of an adjacent triangle $T \in \mathscr{A}(\mathbf{v})$, then

$$\frac{\mathrm{d}E(T)}{\mathrm{d}\mathbf{v}} = \begin{cases} \mathbf{v} = \mathbf{a}_T & : \frac{E(\mathbf{a}_T+\Delta,\mathbf{b}_T,\mathbf{c}_T)-E(\mathbf{a}_T-\Delta,\mathbf{b}_T,\mathbf{c}_T)}{2\Delta} \\ \mathbf{v} = \mathbf{b}_T & : \frac{E(\mathbf{a}_T,\mathbf{b}_T+\Delta,\mathbf{c}_T)-E(\mathbf{a}_T,\mathbf{b}_T-\Delta,\mathbf{c}_T)}{2\Delta} \\ \mathbf{v} = \mathbf{c}_T & : \frac{E(\mathbf{a}_T,\mathbf{b}_T,\mathbf{c}_T+\Delta)-E(\mathbf{a}_T,\mathbf{b}_T,\mathbf{c}_T-\Delta)}{2\Delta} \end{cases} \tag{11}$$

with the notation $E(\mathbf{a}_T, \mathbf{b}_T, \mathbf{c}_T) = E(T)$ and Δ being the pixel size. Note that at some point, each of the three triangle vertices has to be moved into four directions (left, right, up, down) for the finite differences. Thus, in total there are 12 variations of each triangle, for which we have to minimize $E(T)$. The minimization of $E(T)$ is detailed in Section 4.2.

To compute the vertex energy gradient $\frac{\mathrm{d}E(\mathbf{v})}{\mathrm{d}\mathbf{v}}$, we follow Eq. (7), i.e., we iterate over the adjacent triangles $\mathscr{A}(\mathbf{v})$ of vertex $\mathbf{v}$ and sum up the triangle energy gradients, which are estimated using Eq. (11). For this, we precompute for each vertex $\mathbf{v}$ an index list of adjacent triangles $\mathscr{A}(\mathbf{v})$. Additionally, the indices of the three corners of the triangles are stored. The three corners include the vertex $\mathbf{v}$ itself, which we store at the first entry of this list.

4.2. Rasterization-based Minimization of $E(T)$

The minimization of triangle error $E(T)$ requires the approximation of image I with a polynomial, see Eq. (4). Using a geometry shader, we generate the aforementioned 12 triangles (every possible displacement of each corner) and compute $E(T)$ for each, so that the errors can be looked up in Eq. (11).

Constant approximation. For the constant approximation, we compute the mean color of the rasterized fragments for each of the 12 displaced triangles and also for the original triangle.

1. **Compute mean color**. The geometry shader outputs an ID that allows us to distinguish the $12+1$ triangles. The fragment shader of pixel (x_i, y_i) samples the underlying image $I(x_i, y_i)$ and accumulates the color and a fragment counter q for the respective triangle using atomic operations. Since atomics on floats are not natively supported on all hardware, we accumulate the 8-bit colors with values in $[0, 255]$ using integer arithmetic.
2. **Compute error** $E(T)$. In a second pass, we again rasterize all triangles and their variations. This time, the fragment shader looks up the accumulated fragment color and the number of fragments (counter) of the triangle to compute the mean color. Afterwards, the squared difference between mean color and ground truth $I(x_i, y_i)$ is computed for each fragment and again accumulated using atomic operations to obtain $E(T)$, cf. Eq. (4).

Linear approximation. The computation of a linear color gradient per triangle requires an additional computation step.

1. **Setup linear system**. Similar to the constant case, we first rasterize all triangles and their variants. In the fragment shader, we atomically add the coefficients of the linear system, described in Appendix A. This requires 6 coefficients for the symmetric 3×3 system matrix and 3 coefficients per color channel for the right-hand side. Since the pixel coordinates are integer coordinates, all atomic operations can be carried out with integer arithmetic.
2. **Solve linear system**. Afterwards, we use a compute shader to solve the linear system for each triangle. Since the system matrix is symmetric and positive-definite, we use a Cholesky factorization [PTVF96]. See Appendix A for details.
3. **Compute error** $E(T)$. In a second rasterization pass, the fragment shaders look up the coefficients a, b, c of the respective triangle, compute the fragment color using Eq. (3) and accumulate the squared difference to the ground truth using atomics.

5. Stylization Techniques

The minimization of $E(T)$ in the previous section provides us the polynomial coefficients a, b and c of the triangle color functions f, cf. Eqs. (2)–(4). To display the triangulations, we augment the colors f with several rendering styles. Fig. 6 provides an overview of the different styles used throughout the paper.

Constant colors and linear gradients. The basic styles display the colors f directly. Fig. 6b shows the constant colors from Eq. (2) and Fig. 6c displays the linear color gradients from Eq. (3). Naturally, the linear gradients approximate image I better if the image contains smooth color transitions, such as in the Antennae galaxies in Fig. 2, since constant colors introduce edges that were not present in the original image.

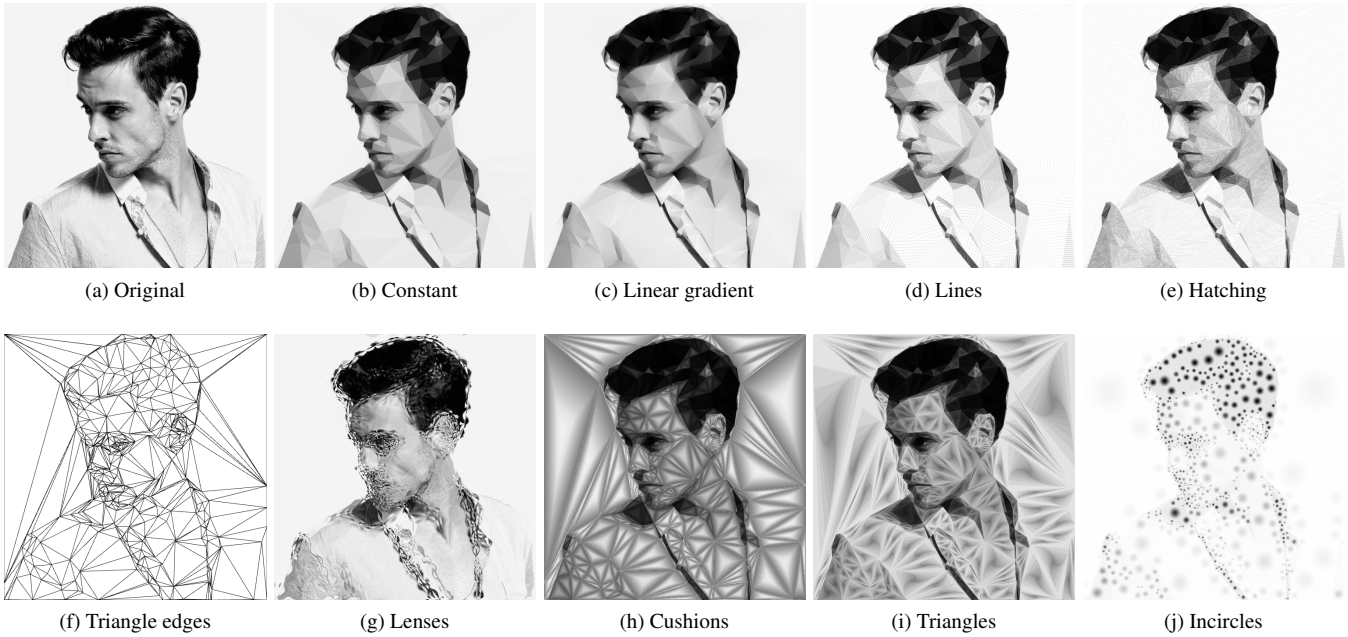
Tonal art maps. The next styles are inspired by handcrafted artwork of Josh Bryan [Bry17], who placed parallel lines inside triangles. Lines are thereby aligned with one edge of the triangle and their density is adjusted to the brightness (more lines appear darker). His hand-drawn lines are not perfectly straight, which gives the image a subtle organic feeling. In Fig. 6d, we approximate this style using tonal art maps (TAMs) of Praun et al. [PHWF01]. A TAM is a 3D texture that is seamless in each 2D slice and in which the third dimension represents the brightness. With decreasing brightness, lines are added, see Fig. 7. Aside from parallel lines, we also use cross-hatching, which is shown in Fig. 6e. For texturing, we use barycentric coordinates, which are scaled by the length of the base edge to obtain a uniform pattern size throughout the image, regardless of the size of individual triangles. From the overall appearance of Bryan's work, the base edge appears to be randomly chosen. Our users can adjust the base edge, see Section 3.5. In Fig. 1 (left), we combined the hatching with the triangle color.

Triangle edges. On top of each style, multi-sampled triangle edges can be rendered, which reveals the triangle topology. Edges are shown in Fig. 6f and are applied for example in Figs. 9 and 10.

Lenses. For an experimental effect, we placed Gaussian-shaped lenses on the triangles' incenter. The fragment shader casts a ray onto the parametric surface and refracts the ray towards the image plane below. The effect creates larger image distortions near edges in the image, since there the triangle density is higher, see Fig. 6g.

Textures. To enable an unlimited artistic degree of freedom, we allow the user to place textures on the triangles. These textures modulate the brightness and can be used for instance to create intriguing depth illusions. Figs. 6h and 6i for instance, apply a rotationally-symmetric cushion texture and a recursively embedded triangle texture that both augment the portrait by a new depth impression.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.



(a) Original (b) Constant (c) Linear gradient (d) Lines (e) Hatching

(f) Triangle edges (g) Lenses (h) Cushions (i) Triangles (j) Incircles

Figure 6: Overview of all rendering styles, which were applied to the same portrait. In total, $|\mathscr{T}| = 880$ triangles were used. In this example, the user used the brushing tool to place more details at the eyes, mouth and ear to resemble the original image better.

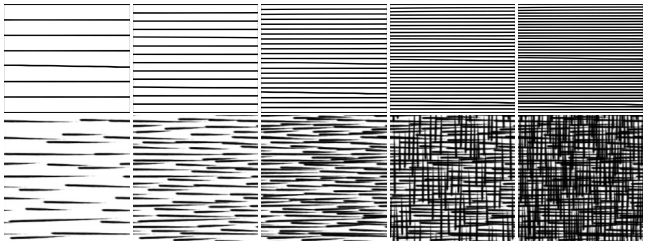


Figure 7: Tonal art maps for line patterns and hatching. The hatching textures were created by Praun et al. [PHWF01].

Incircles. Our last style uses another geometric primitive that is derived from the triangulations. In the fragment shader, we compute center and radius of the triangle's incircle and apply the common smoothstep function $3r^2 - 2r^3$ to the triangle colors f, where $r \in [0,1]$ is the relative distance to the incircle's radius. Delicate structures are typically represented with smaller triangles, which in turn increases the density of incircles. In Fig. 6j for instance, shadows at the shirt border and along facial features become apparent.

6. Results

In the following, we evaluate our method and show results for different kinds of images, including portraits, landscapes and photographs. We compare the method with existing approaches, discuss parameters and elaborate on convergence and performance.

6.1. Comparisons

In Fig. 8, we compare our method with five existing image triangulation tools [Bór17, Con16, Yun13, Fis16, Ham12]. Unfortunately, these tools do not expose a user parameter that allows us to select a desired target number of triangles. Instead, we applied the methods with default parameters and show the obtained results. We applied all methods to seven test images that contain diverse characteristics, including abstract shapes, images with locally or globally high degree of detail, a face and outdoor scenes. Furthermore, we adapt the method by Garland and Heckbert [GH97] to construct an image triangulation. Here, the desired number of triangles can be set.

The method of Conceptfarm [Con16] produces a triangulation that contains roughly equally-sized triangles. The triangle density does not increase near image features, and instead appears to be uniform. Bórquez [Bór17] and Dmesh [Yun13] both adapt the triangulation to image features (gradients). Thereby, the methods exhibit a strong variation in triangle size, leading to preservation of some details, yet large parts of the image might be unrecognizable. The method of Fischer [Fis16] also adapts to features, but involves a threshold that distinguishes between a fine-tessellated focus region and a very coarse context. The transition is noticeable for instance in the third row. The tool of Hamamuro [Ham12] aligns triangle edges with image features, which enables a much better representation of outlines. The technique by Garland and Heckbert [GH97] employs a high number of triangles and reduces them by keeping the image features. For this, we transformed the image in a grayscale image and used the gray value as the height. Afterwards, we constructed an initial triangulation, cf. Sec. 3.3. Finally, we applied the simplification algorithm by setting the number of triangles equal to our result. Among the methods from related work it almost always gave best results. In contrast, our method measures the current approximation error of each triangle and adaptively refines the triangles until a desired number of triangles or a desired approximation error is reached. Thereby, the adaptation to edges and the higher degree

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

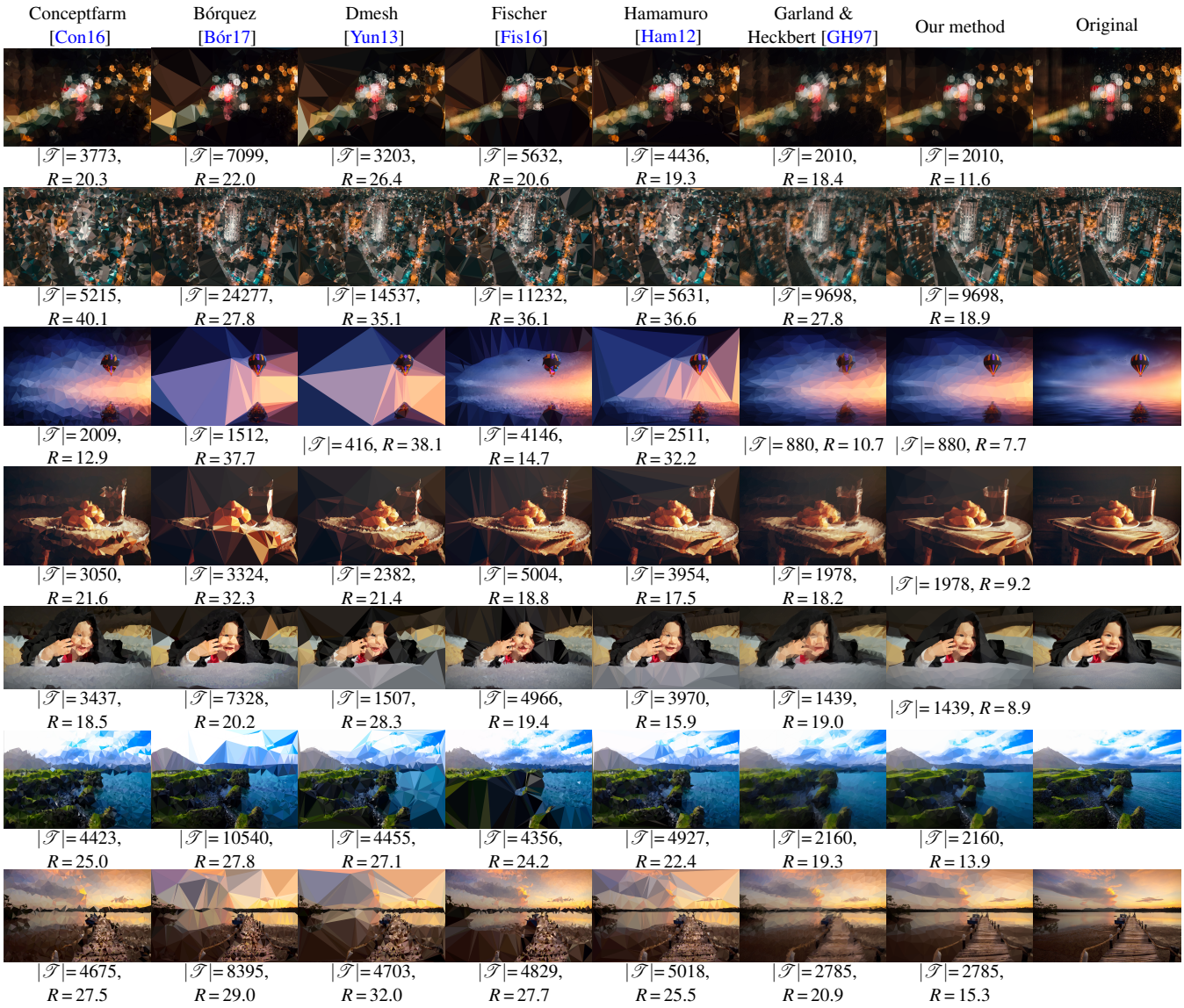


Figure 8: Comparison of our method with existing image triangulation tools. Here, $|\mathscr{T}|$ denotes the number of triangles and R is the approximation error wrt. the original image. Note that the existing tools do not allow to specify a desired number of triangles. Our method has a significantly lower approximation error compare to the best method from previous work, while still using far less triangles.

of detail near image features happen implicitly. Since none of the existing methods considers the approximation quality, our method obtains both qualitatively and quantitatively the best results, even for a much smaller number of triangles, as demonstrated.

6.2. Parameter Study

Next, we analyze the parameters of our optimization-based method.

Influence of initial grid. Since our optimization uses a gradient descent, it is likely to reach local minima. This means that the outcome of the triangulation depends on the initial triangulation. To analyze the dependence, we generated ten random initial vertex distributions in Fig. 9, which are initially Delaunay triangulated. During the first 400 optimization iterations, we iteratively subdivided the triangles with highest error until 1000 triangles were reached. Each time, the additional vertex is incrementally inserted into the Delaunay triangulation, which guarantees a well-behaved topology. As shown in the plot, the errors reduced in each run similarly. This is mainly due to the retriangulation. The visual results of the worst and best triangulation are shown on the right. As expected, the triangulations look different but are quantitatively comparable.

Refinement Threshold τ. A central user parameter is the triangle error threshold τ, which globally determines whether triangles should be subdivided. Results for varying thresholds were shown in Fig. 4. Additional local control over the triangle density is given to the user through the interaction methods, described in Section 3.5.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

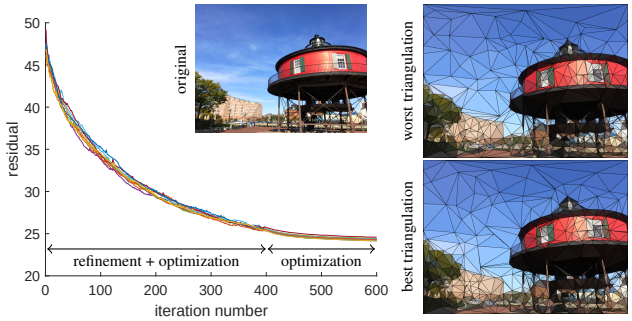


Figure 9: Residual plots for ten automatic refinements with random initial vertices (100 vertices, Delaunay triangulated). Within the first 400 iterations, the triangulation is automatically refined until 1000 triangles are reached. The last 200 iterations show the reduction of the error due to the optimization of the triangulation.

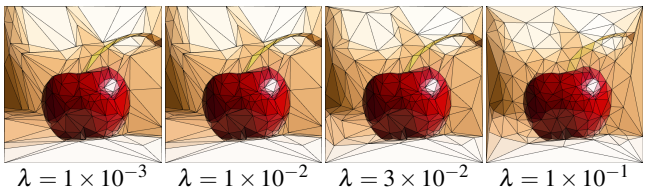


Figure 10: Different λ weights for the regularization, which balance between adaptivity to thin structures (left) and stiffness (right)

As alternative to the specification of a desired error threshold, the total number of triangles may also be bounded, as in Fig. 9. Applying the refinement every iteration leads to faster convergence. Subdividing less frequently can result in a smaller number of vertices and gives the user time for a visual feedback of the intermediate triangle density, which allows for an earlier termination.

Regularization λ. With Eq. (9), we introduced a regularizer that moves vertices toward the center of their 1-ring. Fig. 10 illustrates the effect of varying λ weights. For small or zero λ, triangles might collapse and have zero area, which can be undesirable. The difference between the two left images is small. A further reduction of λ will not change the result. When λ is too large, the triangulation becomes stiff, which hinders adaptation to small details, e.g., in the right image triangles cannot adept to the cherry stalk. Throughout all examples in the paper, we used a weight of $\lambda = 1 \times 10^{-3}$.

6.3. Video Triangulation

Existing video triangulation methods compute triangulations independently for each frame [Yun13], which creates a significant amount of incoherence and flickering, or construct a 3D tetrahedralization that does not meet our approximation goals [RD09]. Our optimization-based method, on the other hand, is able to start from the triangulation of the previous frame and optimizes the vertex positions for the next frame to minimize the approximation error. Thereby, vertices move only short distances and adaptive vertex insertion and collapses are performed to adapt to fast movements, occlusions and the appearance of features. Fig. 11 shows three frames of an elephant galloping cycle. We refer to the accompanying video for a frame coherent animation and further examples.

Figure 11: Three frames of the galloping cycle of an elephant. Our method handles occlusions of the legs and automatically refines.

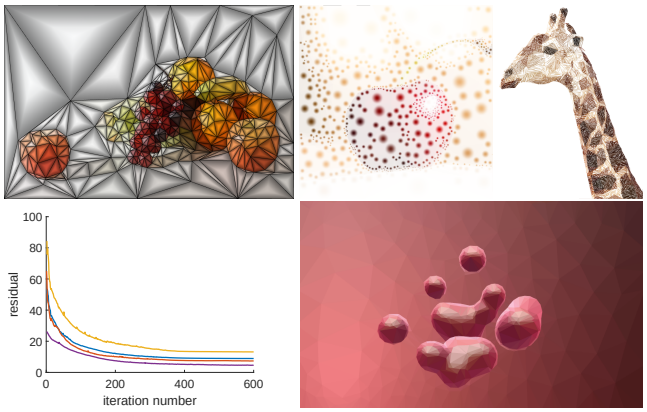


Figure 12: Four image triangulations and their corresponding convergence plots. The energy always decreases quickly.

6.4. Convergence

To measure the approximation quality, we compute the distance to the ground truth I. Since $E(T)$ is a squared distance, the following measure computes the root mean squared error for the entire image

$$R = \sqrt{\frac{1}{|I|} \sum_{T \in \mathcal{T}} E(T)} \tag{12}$$

where $|I|$ is the total number of rasterized fragments, i.e., the number of image pixels. Fig.12 shows the error over time for four different triangulations. The images in the top row (still life, cherry and giraffe) are refined until 700 triangles are reached. The meta ball was refined until an approximation error of $E(T) = 0.3$ was reached for every triangle. Within the first 400 iterations, the triangulation is automatically refined and vertices are optimized. The last 200 iterations reduce the error only by optimizing the vertex positions.

6.5. Performance

In Fig. 13, the computation time of the individual steps from Section 4.2 is listed, as well as the time to update the positions of all vertices. One iteration of the optimization takes for the constant case $3-15\,ms$ and for the linear case $6-24\,ms$. The measurements are taken for the image of Lenna at a resolution of 512×512 pixels, with an Nvidia GeForce GTX 1080 and an Intel Core i7-6700K CPU. The runtime scales linearly in the number of pixels. Note that increasing the number of triangles improves performance. The reason for this is the sequential execution of atomic operations. The larger a triangle, the more fragments atomically add to the same

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

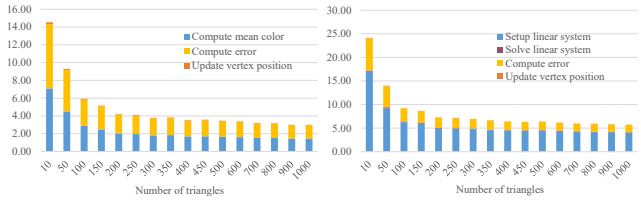


Figure 13: Performance measurements in milliseconds for the constant (left) and linear (right) approximation.

variable, which is a serialized memory access. Evidently, the rasterization (with the atomic operations) is the bottleneck. Operations that are executed per triangle (solve linear system) or per vertex (update vertex position) are with $< 0.12\,ms$ very fast in comparison. Their performance does not change for small numbers of triangles, since groups of threads run on the GPU in parallel.

For reference, we implemented the minimization of $E(T)$ on the CPU to compare the loss of precision due to the integer arithmetic on the GPU, compared to double precision on the CPU. For a constant color, the estimates of $E(T)$ deviate by 0.023 gray values (in [0,255]) and for the linear approximation by 0.029 gray values, which are both very small. Thus, we argue that integer arithmetic for the accumulation of approximation errors on the GPU is sufficient.

6.6. Discussion

Reestablishing the Delaunay properties temporarily increases the energy residual. This is, however, very often useful to eventually move toward a better minimum. On the other hand, if no additional insertion of vertices is allowed, it can prevent the minimization from reaching the minimum. For this reason, we disable the reestablishment of Delaunay triangulations in the end and apply additional vertex optimization iterations until convergence, see Fig. 9.

7. Evaluation

To assess the quality of our triangulation framework, we conducted an evaluation with two artist. Both are of age 32, the first artist A1 is male with three years of experience in digital art and the second artist A2 is female with eight years of experience in digital art as well as conventional art. The evaluation was performed in two steps. In the first step, we asked the artists to generate a triangulation of an image. Both artists were free in their choice what program they use. A1 used Inkscape (`www.inkscape.org`) and A2 used Photoshop (`https://www.adobe.com/products/photoshop.html`). Figure 14 shows the result of A1 using Inkscape only and A2 using Photoshop only in comparison with our result. After the manual generation of the triangulation, we showed the artist our framework in detail and they had time to acquaint themselves with the tool. In the second step, we asked the artists to create triangulations with our tool. During the process, we noted the spoken comments. Afterwards, we asked them to point out limitations and favored functionalities.

Results. Both artists stated that our framework is very helpful to generate a triangulation. Furthermore, they pointed out that the results are visually pleasing. A2 asked for more facilities to generate

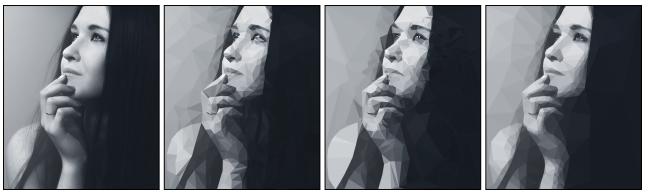


Figure 14: Comparison of manually generated triangulations by artist A1 and A2. From left to right: the original image, the result by the artist A1 with 870 triangles, the result of the artist A2 with 993 triangles, and our result with 800 triangles.

a stylized triangulation. Currently, we allow the user to add new styles by using textures, but she asked for a real-time solution. For this, she imagined a canvas, where different textures can be drawn and the result is shown immediately. The first participant A1 stated that it would be nice to have different styles in different regions. Furthermore, A1 wished to have a preview of the brush tool. After highlighting the regions and activating the brushing, the vertices are currently added randomly in this region. A1 asked for more control over this process. Additionally, both asked for an undo facility and a possibility to move the vertices manually. In summary, both participants enjoyed the framework and stated that it produces pleasing results. They would use it to generate triangulated images.

8. Conclusions

In this paper, we proposed the first optimization scheme that efficiently computes an image triangulation that approximates a given image better than existing tools, both in terms of quality and the approximation error. In comparison to other approaches, we optimize the vertex positions such that an approximation energy is minimized. Starting from an initial triangulation, our method optimizes the position of each vertex independently so that the approximation errors of all adjacent triangles are minimized. In order to preserve image features, we automatically subdivide triangles with high error and reestablish Delaunay triangulations. To provide users full control, we offer direct and indirect methods to interact with the triangulation, including edge flips, splits and collapses, brushes and selective retriangulations. Inspired from artwork, we collected a number of rendering styles, including constant colors, linear gradients, tonal art maps and textures. We believe that our tool can serve as platform to explore other stylizations. Further, we demonstrated that our method produces frame coherent animations for video input data.

In the future, we would like to improve the video triangulations to increase temporal stability for real-world videos with small details. We believe that topological changes and occlusions could be better handled and that the vertex paths could be guided by the underlying motion, for instance through optical flow. Further, we would like to explore different ways to minimize our proposed energy to obtain lower residuals. Currently, we optimize for constant and linear gradients only. It would be straightforward to include the other rendering styles directly into the optimization process or to use other color spaces or error measures.

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

Acknowledgements

We thank Robert W. Sumner and Jovan Popović for the elephant animation.

Appendix A: Triangle Color Optimization

In Section 3.1, we introduced the squared triangle approximation error $E(T)$, cf. Eq. (4), which measures how well the color f of a triangle T approximates the underlying image I. We represent f as polynomial and solve for the unknown monomial basis coefficients. To discretize the triangles, they are rasterized into q pixels:

$$E(T) \approx \frac{1}{2q} \sum_{i=1}^{q} (I(\mathbf{x}_i) - f(\mathbf{x}_i))^2 \quad \rightarrow \quad \min \tag{13}$$

The discrete energy in Eq. (13) is minimized by setting $\nabla E(T) = \mathbf{0}$ and rearranging for the unknowns.

Constant: $f(x,y) = c$

$$c = \frac{1}{q} \sum_{i=1}^{q} I(x_i, y_i) \tag{14}$$

If the triangle color f is constant, the optimal c is the mean color.

Linear: $f(x,y) = ax + by + c$

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{q} \begin{pmatrix} x_i^2 & x_i \cdot y_i & x_i \\ x_i \cdot y_i & y_i^2 & y_i \\ x_i & y_i & 1 \end{pmatrix} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \sum_{i=1}^{q} \begin{pmatrix} x_i \cdot I(x_i, y_i) \\ y_i \cdot I(x_i, y_i) \\ I(x_i, y_i) \end{pmatrix} \end{bmatrix}}_{\mathbf{b}} \tag{15}$$

If f is linear, we solve a linear system to determine the coefficients a, b and c. Since $\mathbf{A}$ is a symmetric and positive-definite matrix, we solve the linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ using a Cholesky decomposition [PTVF96]. If matrix $\mathbf{A}$ is singular, we automatically fall back to the constant case. Note that pixel colors are only contained in the right-hand side $\mathbf{b}$. Thus, if the color has multiple channels (R, G, B), the Cholesky factorization of $\mathbf{A}$ can be reused for each channel.

References

[AA06] ARIFIN A. Z., ASANO A.: Image segmentation by histogram thresholding using hierarchical cluster analysis. *Pattern Recogn. Lett. 27*, 13 (Oct. 2006), 1515–1521. 3

[Ada11] ADAMS M. D.: A flexible content-adaptive mesh-generation strategy for image representation. *IEEE Transactions on Image Processing 20*, 9 (Sept 2011), 2414–2427. 3

[Ada13] ADAMS M. D.: A highly-effective incremental/decremental delaunay mesh-generation strategy for image representation. *Signal Processing 93*, 4 (2013), 749 – 764. 3

[Bór17] BÓRQUEZ J.: The Delaunay triangulation image generator. http://snapbuilder.com/code_snippet_generator/triangulator_triangulation_image_generator/, 2017. Accessed: 2017-09-16. 1, 2, 7, 8

[Bow81] BOWYER A.: Computing Dirichlet tessellations. *The Computer Journal 24*, 2 (1981), 162–166. 5

[Bry17] BRYAN J.: Hand-drawn Image Triangulations. https://www.joshbryanart.com/triangulations, 2017. Accessed: 2017-09-16. 1, 6

[CDHM11] COHEN A., DYN N., HECHT F., MIREBEAU J.-M.: Adaptive multiresolution analysis based on anisotropic triangulations. *Mathematics of Computation 81* (Sept. 2011), 789–810. 2

[Con16] CONCEPT FARM: Triangulate 7. https://www.behance.net/gallery/10543937/Image-Triangulator-App, 2016. Accessed: 2017-09-16. 1, 2, 7, 8

[CQM*15] CHEN Z., QI Z., MENG F., CUI L., SHI Y.: Image segmentation via improving clustering algorithms with density and distance. *Procedia Computer Science 55* (2015), 1015 – 1022. 3rd International Conference on Information Technology and Quantitative Management, ITQM 2015. 3

[CZL14] CHENG X., ZENG M., LIU X.: Feature-preserving filtering with l0 gradient minimization. *Computers & Graphics 38* (2014), 150 – 157. 3

[DI04] DEMARET L., ISKE A.: Advances in digital image compression by adaptive thinning. *Annals of the Marie-Curie Fellowship Association 3* (2004), 105–109. 2

[DS02] DECARLO D., SANTELLA A.: Stylization and abstraction of photographs. *ACM Trans. Graph. 21*, 3 (July 2002), 769–776. 2

[ES96] EDELSBRUNNER H., SHAH N. R.: Incremental topological flipping works for regular triangulations. *Algorithmica 15*, 3 (Mar 1996), 223–241. 5

[Fis16] FISCHER G.: Triangulate images. https://snorpey.github.io/triangulation/, 2016. Accessed: 2017-09-16. 1, 2, 7, 8

[GDA*12] GERSTNER T., DECARLO D., ALEXA M., FINKELSTEIN A., GINGOLD Y., NEALEN A.: Pixelated image abstraction. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Goslar Germany, Germany, 2012), NPAR '12, Eurographics Association, pp. 29–36. 2

[GGD08] GRUNDLAND M., GIBBS C., DODGSON N. A.: Stylized multiresolution image representation. *Journal of Electronic Imaging 17*, 1 (2008), 013009:1–17. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), SIGGRAPH '97, pp. 209–216. 2, 7, 8

[GPS89] GREIG D. M., PORTEOUS B. T., SEHEULT A. H.: Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)* (1989), 271–279. 3

[Ham12] HAMAMURO A.: Triangulation image generator. http://jsdo.it/akm2/xoYx, 2012. Accessed: 2017-09-16. 1, 2, 7, 8

[HHK12] HOU X., HAREL J., KOCH C.: Image signature: Highlighting sparse salient regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 1 (Jan 2012), 194–201. 4

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. 2

[HP74] HOROWITZ S. L., PAVLIDIS T.: Picture Segmentation by a directed split-and-merge procedure. *Proceedings of the 2nd International Joint Conference on Pattern Recognition, Copenhagen, Denmark* (1974), 424–433. 3

[KCWI13] KYPRIANIDIS J. E., COLLOMOSSE J., WANG T., ISENBERG T.: State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics 19*, 5 (May 2013), 866–885. 2

[Leu15] LEUNG H.: Polygonian. http://www.polygonian.com/, 2015. Accessed: 2017-09-16. 1, 2

[LM99] LUCCHESE L., MITRA S. K.: Unsupervised segmentation of color images based on k -means clustering in the chromaticity plane. In *Proceedings of the IEEE Workshop on Content-Based Access of Image and Video Libraries* (1999), pp. 74–. 3

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.

[MA15] MOSTAFAVIAN S., ADAMS M. D.: An optimization-based mesh-generation method for image representation. In *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (Aug 2015), pp. 234–239. 3

[MCL*14] MIN D., CHOI S., LU J., HAM B., SOHN K., DO M. N.: Fast global image smoothing based on weighted least squares. *IEEE Transactions on Image Processing 23*, 12 (Dec 2014), 5638–5653. 3

[NB15] NGUYEN R. M. H., BROWN M. S.: Fast and effective l0 gradient minimization by region fusion. In *IEEE International Conference on Computer Vision (ICCV)* (Dec 2015), pp. 208–216. 3

[NISA06] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Laplacian mesh optimization. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), GRAPHITE '06, ACM, pp. 381–389. 4

[NN04] NOCK R., NIELSEN F.: Statistical region merging. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 11 (Nov. 2004), 1452–1458. 3

[OBB*13] ORZAN A., BOUSSEAU A., BARLA P., WINNEMÖLLER H., THOLLOT J., SALESIN D.: Diffusion curves: A vector representation for smooth-shaded images. *Commun. ACM 56*, 7 (July 2013), 101–108. 2

[Oll15] OLLIVIER P.: Trimaginator. http://trimaginator.com/, 2015. Accessed: 2017-09-16. 1, 2

[OPR78] OHLANDER R., PRICE K., REDDY D. R.: Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing 8*, 3 (1978), 313 – 333. 3

[Ots79] OTSU N.: A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics 9*, 1 (1979), 62–66. 3

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH, ACM, pp. 581–. 6, 7

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 4

[PTVF96] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P.: *Numerical recipes in C*, vol. 2. Cambridge university press, Cambridge, 1996. 6, 11

[Puc08] PUCKEY J.: Delaunay raster imagery. https://www.jonathanpuckey.com/projects/delaunay-raster/, 2008. Accessed: 2017-09-16. 1, 2

[RC78] RIDLER T. W., CALVARD S.: Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man, and Cybernetics 8*, 8 (Aug 1978), 630–632. 3

[RD09] RICHARDT C., DODGSON N. A.: Voronoi video stylisation. In *Computer Graphics International Short Papers* (May 2009), pp. 103–108. 9

[SD15] SONAWANE M., DHAWALE C.: A brief survey on image segmentation methods. In *IJCA Proceedings on National conference on Digital Image and Signal Processing* (2015), Citeseer. 3

[Sec02] SECORD A.: Weighted voronoi stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2002), NPAR '02, ACM, pp. 37–43. 2

[SM16] SÁNCHEZ-MARÍN J.: Polyshaper. http://polyshaper.co/, 2016. Accessed: 2017-09-16. 1, 2

[SWD14] STORATH M., WEINMANN A., DEMARET L.: Jump-sparse and sparse recovery using potts functionals. *IEEE Transactions on Signal Processing 62*, 14 (July 2014), 3654–3666. 3

[TA13] TU X., ADAMS M.: Improved mesh models of images through the explicit representation of discontinuities. *Canadian Journal of Electrical and Computer Engineering 36*, 2 (Spring 2013), 78–86. 3

[Wat81] WATSON D. F.: Computing the n-dimensional delaunay tessellation with application to voronoi polytopes*. *The Computer Journal 24*, 2 (1981), 167–172. 5

[XLXJ11] XU L., LU C., XU Y., JIA J.: Image smoothing via l0 gradient minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 174:1–174:12. 3

[Yun13] YUN D. Y.: DMesh. http://dmesh.thedofl.com/, 2013. Accessed: 2017-09-16. 1, 2, 7, 8, 9

[YWB03] YANG Y., WERNICK M. N., BRANKOV J. G.: A fast approach for accurate content-adaptive mesh generation. *IEEE Transactions on Image Processing 12*, 8 (2003), 866–881. 2

[ZA15] ZAITOUN N. M., AQEL M. J.: Survey on image segmentation techniques. *Procedia Computer Science 65* (2015), 797 – 806. International Conference on Communications, management, and Information technology (ICCMIT'2015). 3

[ZL16] ZOU Y., LIU B.: Survey on clustering-based image segmentation techniques. In *Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on* (2016), IEEE, pp. 106–110. 3

[ZXJ14] ZHANG Q., XU L., JIA J.: 100+ times faster weighted median filter (wmf). In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2014), pp. 2830–2837. 3

© 2018 The Author(s)
Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.