university of groningen / faculty of science and engineering / bernoulli institute

Error-bounded Image Triangulation with Mesh Colors

Master Thesis CS

February 2025

Student: Sjoerd Hilhorst

First supervisor: Jiří Kosinka

Secondary supervisor: Kuanhao He

Abstract

Image triangulation tries to partition an image into a set of triangles that approximate the image. The goal is to create a compact representation that can be used for various purposes, such as image compression, image editing, and scaling. In this thesis, we extend the recent paper on Error-bounded Image Triangulation by Fang et al. by introducing automatic mesh color fitting. The proposed method allows for adaptive mesh color resolution and applies this in areas of complex gradients to reduce color error. We evaluate our method on a diverse set of images and compare it to the state-of-the-art method by Fang et al. Our results demonstrate that our method generates more visually appealing results and handles complex color gradients better. We also show that our method is capable of producing high-quality results with fewer triangles than the state-of-the-art method.

Contents

1 Introduction

Raster images consist of a grid of pixels, each having specific color values, making them ideal for representing intricate details and subtle color transitions, as seen in photographs. On the other hand, vector images are created from geometric shapes on a Cartesian plane, such as points, lines, polygons, and curves. Vector images, also known as vector graphics, have several desirable qualities, including infinite scalability without quality loss, compact representation, and easy editability.

Image vectorization is the process of converting raster images into vector format. Manual vectorization is often tedious and slow and requires high expertise for complex images. A 2008 survey found that more than 7 million man-hours are spent manually vectorizing images in the United States every year [10]. Advancements in automatic image vectorization techniques have introduced various methodologies to address these challenges. One such technique is image triangulation, which breaks down a raster image into non-overlapping triangles. These meshes can be used for various artistic effects and, when sufficiently sparse, are highly editable.

When converting raster images into vector format, two quantitative indicators denote the quality of the obtained vectorization: (1) the complexity of the obtained vectorization, often quantified as the number of primitives, and (2) the color error between the obtained vectorized image and the original image. Lowering complexity leads to higher compactness of representation and editability, but in turn often increases color error. Balancing the trade-off between color error and triangulation complexity is essential for generating triangular meshes that remain recognizable while being editable and visually appealing.

Fang et al. [11] created a method to vectorize raster images using Bézier triangles under an error-bounded constraint. Curved triangles allow the generated mesh to closely follow the contours of the underlying image, reducing the error. The final mesh is obtained through a series of local remeshing operations on an initial coarse mesh. These remeshing operations reduce error and complexity. The triangulation complexity is reduced specifically by edge collapsing, which merges triangles, thereby reducing the mesh complexity. The operations are constrained by the color error bound, ensuring that no operation causes the error to exceed the bound. Computing optimal locations for the remeshing operations is achieved by a differential evolution algorithm. The method has proved effective in generating low-complexity meshes under a color error constraint for a variety of raster images.

Mesh colors is a vector graphics primitive that allows for more color detail per triangle [28]. It is an extension of vertex colors where multiple regularly spaced color samples are specified on a single face. This allows for more intricate color gradients to be represented on a single face, instead of only on the corner vertices. This approach effectively detaches the geometry from the color definition. As a result, it reduces the need for dense meshes in areas of the image with complex color gradients but simple geometry. It allows to simplify the mesh while retaining the color detail.

To further reduce mesh complexity while retaining an upper bound on the color error, this thesis proposes a novel approach that combines mesh colors with Error-Bounded Image Triangulation.

1.1 Proposed Methods and Overview

We propose the addition of mesh colors [28] to the automatic image triangulation method by Fang et al. [11] to further reduce mesh complexity. Fang et al.'s method supports quadratic color interpolation per face at most and is frequently implemented with a single constant color per face. As a result, the method generates dense meshes in regions of images with complex color gradients. These dense meshes are necessary to capture the gradients accurately and ensure the mesh remains error-bounded. However, in many cases, the geometry in these regions is relatively simple, making the meshes unnecessarily complex.

The mesh colors primitive can represent intricate color gradients on a single face by specifying multiple regularly spaced sample points in parameter space. Each face has an independent resolution, providing precise control over the level of detail needed to represent the color gradient. This decouples the geometry from the color gradient, reducing the need for dense meshes in areas with complex color transitions, simplifying overall mesh complexity.

We propose an approach that adaptively applies mesh colors to the method by Fang et al. [11] in regions with complex color gradients, while using the original color functions in regions with simple color gradients.

Moreover, we propose additional improvements to the Error-Bounded Image Triangulation method to accelerate mesh simplification. Notably, we move the pixel-to-face assignment operation to the GPU, effectively parallelizing the operation for each triangle.

By combining these methods, this thesis aims to achieve the following contributions:

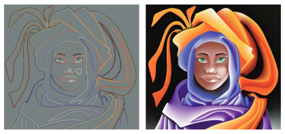
- Adaptively add the mesh colors primitive into the automatic image triangulation method by Fang et al. [11]. Mesh colors are applied in regions with complex color gradients, while the original color functions are used in regions with simple color gradients. This reduces the need for dense meshes in regions with simple geometry, simplifying the overall mesh complexity.

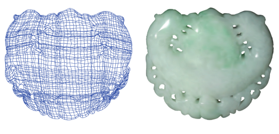
- Improve computational efficiency: The proposed method moves the pixel-to-face assignment operation to the GPU, effectively parallelizing the operation for each triangle. This reduces the time taken to assign pixels to faces, a computationally expensive operation.

The remainder of this thesis is organized as follows: Chapter 2 reviews related work in the field of image vectorization and highlights the contributions of this thesis. In Chapter 3, we provide an in depth explanation of Error-bounded Image Triangulation by Fang et al. [11] and Mesh Colors by Yuksel et al. [28], as these methods are central to our contribution. Chapter 4 describes the proposed methodology in detail, specifying the integration of Mesh Colors for Gradient Meshes with Error-Bounded Image Triangulation. Chapter 5 presents the results and evaluates the performance of the proposed method. Chapter 6 concludes the thesis, and Chapter 7 discusses potential future work.

2 Related work

The field of automatic image vectorization is vast, with methods using a variety of techniques. Figure 1 shows a few examples of automatically generated vector images. This chapter has two objectives. First, it provides an overview of vectorization primitives and automatic vectorization methods. Second, it places particular focus on automatic triangulation methods which are most relevant to our method, and relates them to our approach. For a more extensive survey of image vectorization methods, we refer to the work by Tian and Günther [22].


(a) Diffusion curve [19]


(b) Gradient mesh [14]

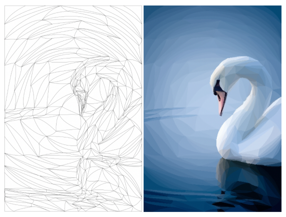

(c) Triangulation [11]

Figure 1: Vector images obtained by automatic vectorization techniques using different primitives. For each image pair the left image shows the primitives and the right image shows the corresponding generated vector image.

2.1 Diffusion Curves

One vector graphics primitive used in automatic vectorization is diffusion curves, introduced by Orzan et al. [19]. A diffusion curve consists of Bézier control points, two sets of color control points (for the left and right sides of the curve), and a set of blur control points. The curves diffuse color on each side of the curve, with the softness of the transition given by the blur. By combining multiple diffusion curves, raster images can be accurately represented. Orzan et al. [19] also introduced a method for automatic image vectorization by placing diffusion curves along edges extracted from the input image using the Canny edge detector [7]. Xie et al. [27] have improved on this method by detecting edges in the Laplacian domain, which better captures blurred edges with a large gradient. Furthermore, the edge detection is employed over multiple image scales, which allows for better detection of edges at coarse and fine scales. Other improvements have been achieved by including a depth map into the vectorization process [17], using superpixel boundaries [8]. Zhao et al. [31] propose a complementary method, where, instead of predetermining curve geometry and optimizing coloring, the geometry is iteratively optimized and the color function is fitted accordingly. Their technique also accommodates input from color fields beyond traditional pixel images, expanding the applicability of diffusion curves. Despite their expressiveness, obtaining an accurate representation involves solving a large system of linear equations [6], hindering practical use.

2.2 Gradient Meshes

Gradient meshes, introduced by Adobe Illustrator [2], are a vector graphics primitive that smoothly interpolates color across a regular bicubic quadrilateral mesh. Each vertex in the mesh contains the location, color, and tangent handles. Tangent handles define the direction of the gradient. Semi-automatic methods have been proposed where the user provides the geometry, and the color gradient is fitted automatically [21, 20]. Fully automatic techniques are achieved by automatic feature detection [14] or encoding color gradients [24]. Gradient meshes can accurately represent smooth regions in an image, but require coarse meshes to represent areas of high detail. Other forms of the gradient mesh primitive have been proposed to address these issues [4, 16, 5]. In particular, Baksteen et al. [4] propose a method that combines the gradient mesh primitive with mesh colors [28]. By integrating mesh colors intricate gradients can be captured better, therefore reducing complexity.

2.3 Image Triangulation

Triangular meshes have a long history of being used to represent images in automatic image vectorization [9, 1, 25]. Piecewise linear meshes are an obvious first step [9, 1], but often result in dense meshes in areas of high detail, and creates suboptimal results when scaling, eliminating one of the main advantages [26]. Curved triangles, bounded by Bézier curves, have been proposed to address this issue [26, 12, 13, 11, 25, 32].

Triangulation approaches generally consist of two phases. The first phase is an initialization phase that defines an initial geometry from detected features. Initial triangulations can be produced creating triangles at pixel resolution [25]. Producing initial triangulations often involve applying a Delaunay [18, 30, 15] or constrained Delaunay [11, 26, 12] triangulation to the detected image features. Unconstrained Delaunay triangulations often create edges that do not represent the image, therefore, approaches that detect edges in the image and constrain these for constrained Delaunay triangulations are often preferred. The second phase applies various optimization strategies for refining the mesh geometry and color to reduce color error [11]. This phase can be categorized into two categories: (1) Adjusting geometry to align with image features and reduce color error [12, 11, 26]. (2) Reduce the color error by adding primitives and increasing mesh complexity.

2.4 Notable Recent Triangulation Methods

In recent years, several methods have been proposed to improve the quality of the triangulation. Hettinga et al. [12] propose a triangulation process with various optimization steps to minimize mesh complexity. Notably, to our knowledge, they are the first and currently only approach to use mesh colors as a primitive in automatic vectorization. Their process is as follows: (1) Two types of features are detected: hard and soft edges. Hard edges define sharp color discontinuities. Soft edges represent gradual color transitions. Hard edges are extracted using the Canny edge detector [7] directly on the input image. For soft edges, the Canny edge detector is applied to a quantized grayscale image. Subsequently, an edge filtering operation ensures non-overlapping soft and hard edges. (2) The soft and hard edges are vectorized using cubic Bézier splines, and a constrained Delaunay triangulation is applied to the resulting control points. The topology is now fixed. (3) Mesh colors [28] are adaptively fitted to the mesh. Mesh color resolution is determined per triangle based on pixel area size. The color at sample points is defined by minimizing the error between the original image and interpolated mesh colors per triangle, reducing to a least squares problem.

Xiao et al. [26] focus on minimizing color error of a mesh initialized with a bounded number of vertices and quadratic Bézier triangles. The error minimization is guided by gradient descent for relocation of vertices and control points. The method relies heavily on a good initial triangulation, as gradient descent naturally converges to local minima. A suitable initial triangulation is obtained by applying a constrained Delaunay triangulation to detected image features. Vertices are added to the circumcenter of the triangle with the largest area until a bound on the number of vertices is reached. After initialization, the mesh is optimized by gradient descent. The gradient of vertices and control points are calculated from the error function, and an ideal step size is computed.

Fang et al. [11] propose a solution less sensitive to local minima. A low mesh complexity is achieved under a color error-bound constraint by a series of remeshing operations consisting of vertex relocation. To overcome the issue of local minima, two techniques are employed: (1) optimal locations for vertices and control points during remeshing operations are found using a differential evolution algorithm. (2) Two remeshing operations, edge splitting and collapsing, are used to avoid local minima traps. Their method supports quadratic Bézier triangles and can be described in two steps. (1) A coarse initial triangulation (obtained from other methods) is brought below an error bound by a series of remeshing operations. Notably, edge splitting is performed to reduce the error, but increases mesh complexity. (2) Mesh complexity is reduced by another series of remeshing operations, where edge splitting is replaced by edge collapsing, reducing the mesh complexity. Sufficient space between the current error and error bound, needed for the error introduced by edge collapsing, is achieved by the other remeshing operations. Per-triangle colors are calculated by solving a least squares problem similar to [12].

Recently, Wang et al. [23] combined and extended the method of Fang et al. [11] and Xiao et al. [26]. Firstly, cubic Bézier curves are supported. Secondly, the method is guided by gradient descent optimization for relocation of vertices and control points, but to avoid the local minima trap, the method adaptively inserts vertices for faces during the optimization phase. Moreover, adjacent curves are merged if the cumulative error of adjacent faces is below an error threshold. Table 1 provides a comparison of the methods by Hettinga et al. [12], Xiao et al. [26], Fang et al. [11], and Wang et al. [23].

The methods discussed above showcase the recent advancements in automatic triangulation for image vectorization. Hettinga et al. [12] introduced the use of mesh colors as a primitive in automatic image vectorization. Xiao et al. [26] focused on error minimization through gradient descent, while Fang et al. [11] employed differential evolution and remeshing techniques to mitigate local minima traps, further improving the quality of the triangulation. Wang et al. [23] combined these ideas by incorporating adaptive vertex insertion into the remeshing pipeline guided by gradient descent. Table 1 provides a comparison of these methods.

While these methods make important contributions, they often face challenges in areas with complex gradients, where balancing mesh complexity and color error remains difficult. Our approach draws heavily from the method by Fang et al. [11], particularly in terms of problem formulation and solving strategy. However, we extend their work by integrating mesh colors, a technique previously explored by Hettinga et al. [12], to

Method	**Initialization**	**Optimization**	**Color Function**	**Mesh Operations**
Hettinga et al. [12]	Delaunay triangulation on soft/hard edges (Canny)	None	Mesh colors (least squares)	None
Xiao et al. [26]	Delaunay triangulation; vertex addition	Gradient descent	Constant, linear, quadratic (least squares)	Vertex/control point relocation, edge flipping
Fang et al. [11]	Obtained from other methods	Differential evolution	Constant, linear, quadratic (least squares)	Vertex/control point relocation, edge splitting/collapsing, edge flipping
Wang et al. [23]	Identical to [26]	Gradient descent + vertex insertion	Constant, linear, quadratic (least squares)	Vertex/control point relocation, vertex insertion, edge merging/flipping

Table 1: Comparison of recent automatic image triangulation methods.

adaptively refine the mesh color resolution. This allows our method to handle complex gradients more effectively and reduce color error, particularly during the edge collapsing phase. By combining these strategies, our approach can further reduce complexity under an error bound, thus providing a more accurate and visually pleasing triangulation for automatic image vectorization.

3 Background

Our approach builds upon the work by Fang et al. [11] and Yuksel et al. [28]. This section provides an overview of these works.

3.1 Error-bounded Image Triangulation

Fang et al. [11] introduced a method for generating a mesh of low complexity of image triangulations under a color error-bounded constraint. Their method works by formulating the problem as a constrained optimization problem, where the goal is to achieve the lowest mesh complexity under a certain color error bound. Because this method is the basis for our method, a detailed overview of the methodology is given below.

Overview
The triangulation algorithm consists of two phases. Figure 2 visually depicts the steps in the pipeline. First, an initial mesh is generated by applying a constrained Delaunay triangulation to detected feature edges and points. This initial mesh usually does not satisfy the error bound and quality constraints. An initialization phase performs a series of remeshing operations to reduce the color error. These remeshing operations consist of vertex relocation, control point relocation, edge flipping, and edge splitting. The initialization phase stops when the color error is error bound. The second phase is the simplification phase, where the mesh is reduced in complexity by reducing the number of triangles. This is achieved by another series of remeshing operations consisting of vertex relocation, edge relocation, edge flipping, and edge collapsing. The simplification process stops if either of the following conditions is met: (1) The maximum number of simplification iterations is reached, (2) the number of edge collapses in a single iteration is below a user-specified threshold.

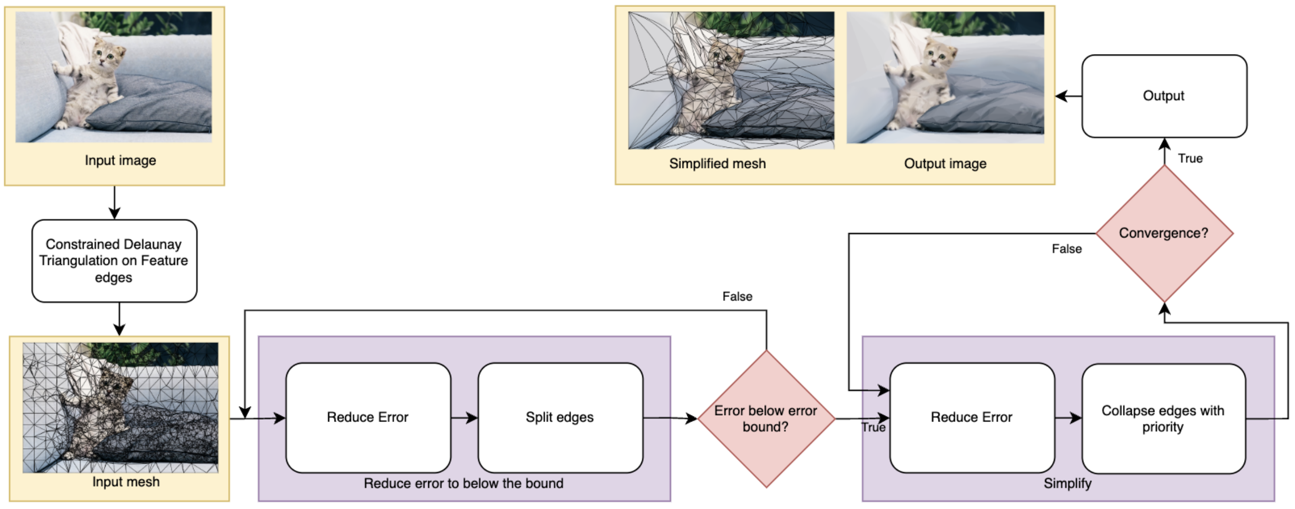


Figure 2: The pipeline of the method by Fang et al. [11]. Our method proposes changes to the collapse edges operation in the simplification step.

Problem statement
The problem is defined as a constrained optimization problem. The goal of

the method is to generate a triangular mesh $\mathcal{M}$ that satisfies the following requirements given an image $\mathcal{I}$ and a color threshold $\mathcal{E}_{\mathcal{I}}$;

- $\mathcal{M}$ is valid and partitions the whole image domain.
- The color error is $\leq \mathcal{E}_{\mathcal{I}}$.
- The complexity, i.e. the number of triangles N_t of $\mathcal{M}$, is low.

Mesh Definition
The mesh $\mathcal{M}$ contains a set of vertices V, edges E, and triangles T. In this method, $\mathcal{M}$ is represented using quadratic Bézier triangles, allowing for more flexible geometry that can more closely follow the contours of the underlying image. A two-dimensional quadratic Bézier triangle (degree $n_{tri} = 2$) is defined by a set of 6 control points. The surface of the triangle is defined by the following equation:

$$\mathbf{B}(u, v) = \sum_{i=0}^{2} \sum_{j=0}^{2-i} B_{i,j}(u, v)\mathbf{P}_{i,j}, \tag{1}$$

where:

- $\mathbf{P}_{i,j}$ are the control points of the Bézier triangle,
- $B_{i,j}(u, v)$ are the Bernstein basis functions for the Bézier triangle:

$$B_{i,j}(u, v) = \binom{2}{i}\binom{2-i}{j} u^i v^j (1 - u - v)^{2-i-j}, \tag{2}$$

- u, v are the barycentric coordinates with $0 \leq u, v \leq 1$ and $u + v \leq 1$.

Color Function
The image $\mathcal{I}$ is treated as a color function h over the domain $\Omega = [0, \text{width}] \times [0, \text{height}]$. The color on each triangle $t \in T$ is approximated by a polynomial P_t of degree n_{poly}. Only the the polynomial degrees $n_{\text{poly}} = 0, 1, 2$ are used, corresponding to constant, linear, and quadratic, respectively:

- $n_{\text{poly}} = 0$: constant, $P_t(\mathbf{x}) = f$,
- $n_{\text{poly}} = 1$: linear, $P_t(\mathbf{x}) = dx + ey + f$,
- $n_{\text{poly}} = 2$: quadratic, $P_t(\mathbf{x}) = ax^2 + bxy + cy^2 + dx + ey + f$,

where $\mathbf{x} = (x, y)$ and a, b, c, d, e, f are the polynomial coefficients. Solving the color function for a triangle $\mathbf{t} \in T$ consists of finding the coefficients for $P_t(\mathbf{x})$ that minimize the error.

Color Error
The ℓ_2-norm error defines the error between a triangle $\mathbf{t} \in T$ and the image $\mathcal{I}$:

$$\mathcal{E}(\mathbf{t}, \mathcal{I}) = \min_{P_{\mathbf{t}}} \int_{\mathbf{t}} (h(\mathbf{x}) - P_{\mathbf{t}}(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x}, \tag{3}$$

where $h(\mathbf{x})$ denotes the color of the image $\mathcal{I}$ at $\mathbf{x} = (x, y)$, and $P_{\mathbf{t}}(\mathbf{x})$ is the color of the triangle $\mathbf{t}$ at $\mathbf{x}$. The error is minimized by finding the optimal polynomial $P_{\mathbf{t}}$ for each triangle $\mathbf{t}$, denoted as *the best-fitting color function*: $P_{\mathbf{t}}^*$ is the one that minimizes the error. The color error over the entire image $\mathcal{E}(\mathcal{M}, \mathcal{I})$ is the sum of the errors over all triangles in the mesh $\mathcal{M}$:

$$\mathcal{E}(\mathcal{M}, \mathcal{I}) = \sum_{\mathbf{t} \in T} \mathcal{E}(\mathbf{t}, \mathcal{I}) = \sum_{\mathbf{t} \in T} \int_{\mathbf{t}} \left(h(\mathbf{x}) - P_{\mathbf{t}}^*(\mathbf{x})\right)^2 \, \mathrm{d}\mathbf{x}. \tag{4}$$

Constraining the color error to be bounded is formulated as

$$\mathrm{RMSE}(\mathcal{M}, \mathcal{I}) = \left(\frac{\mathcal{E}(\mathcal{M}, \mathcal{I})}{\text{width} \times \text{height} \times \text{channels}} \right)^{\frac{1}{2}} \leq \varepsilon_l, \tag{5}$$

where RMSE is the root mean squared error, ε_l is the color error threshold, width and height are the width and height of $\mathcal{I}$ in pixels, and channels is the number of color channels in image $\mathcal{I}$ (e.g. 3 for RGB images, 1 for grayscale). The goal is to minimize the mesh complexity N_t while satisfying the color error constraint.

Quality Constraints

Furthermore, three quality constraints are defined to ensure the validity and quality of the mesh.

- Validity constraint: Each triangle is neither degenerate nor flipped; this can be achieved by checking whether the triangle is locally injective.

$$\mathcal{Q}_v(\mathbf{t}) = \begin{cases} 0, & \text{if } \mathbf{t} \text{ is locally injective} \\ +\infty, & \text{otherwise.} \end{cases} \tag{6}$$

- Minimum angle constraint: The minimum angle of each triangle $\theta_{min}(\mathbf{t})$ is greater than a user-specified threshold θ_I or is monotically increasing, thus not smaller than the minimum angle $\theta_{\min}^{\text{prev}}(\mathbf{t})$ before updating.

$$\mathcal{Q}_\theta(\mathcal{M}, \mathbf{t}) = \begin{cases} 0, & \text{if } \theta_{\min}(\mathbf{t}) \geq \min\left\{\theta_I, \theta_{\min}^{\text{prev}}(\mathbf{t})\right\} \\ +\infty, & \text{otherwise.} \end{cases} \tag{7}$$

- Maximum curvature constraint: Each edge curvature angle $\kappa(\mathrm{e})$ is greater than or equal to a user specified threshold κ_I.

$$\mathcal{Q}_\kappa(\mathcal{M}, \mathbf{e}) = \begin{cases} 0, & \text{if } \kappa(\mathrm{e}) \geq \kappa_I \\ +\infty, & \text{otherwise.} \end{cases} \tag{8}$$

The overall quality constraint $\mathcal{Q}(\mathcal{M})$ for the mesh $\mathcal{M}$ is defined as the sum of these three individual quality constraints. This acts as a barrier function, ensuring that the mesh $\mathcal{M}$ satisfies all the specified quality constraints:

$$\mathcal{Q}(\mathcal{M}) = \sum_{\mathbf{t} \in T} \mathcal{Q}_v(\mathbf{t}) + \sum_{\mathbf{t} \in T} \mathcal{Q}_\theta(\mathcal{M}, \mathbf{t}) + \sum_{\mathbf{e} \in E} \mathcal{Q}_\kappa(\mathcal{M}, \mathbf{e}). \tag{9}$$

Optimization Problem Formulation

Given the target degree n_{tri} of the Bézier mesh, the polynomial degree n_{poly} of the color function, and the color error threshold ε_I, generating the desired triangulation $\mathcal{M}$ can be formulated as solving a constrained optimization problem:

$$\begin{aligned} \min_{\mathcal{M}} \quad & N_t(\mathcal{M}) + \mathcal{Q}(\mathcal{M}) \\ \text{s.t.} \quad & \text{RMSE}(\mathcal{M}, \mathcal{I}) \leq \varepsilon_I, \end{aligned} \tag{10}$$

where $N_t(\mathcal{M})$ is the number of triangles in the mesh $\mathcal{M}$, and $\mathcal{Q}(\mathcal{M})$ is the quality constraint.

Remeshing Operations

The initialization and simplification phases consist of a series of remeshing operations. These operations, depicted in Figure 3, are applied to the mesh $\mathcal{M}$ to reduce the color error and mesh complexity. In each operation, only the local region of the particular operation is considered. The local region, denoted as $\mathcal{L} \subset \mathcal{M}$ of an operation, consists of the set of triangles that are directly affected by the operation. The remeshing operations are as follows:

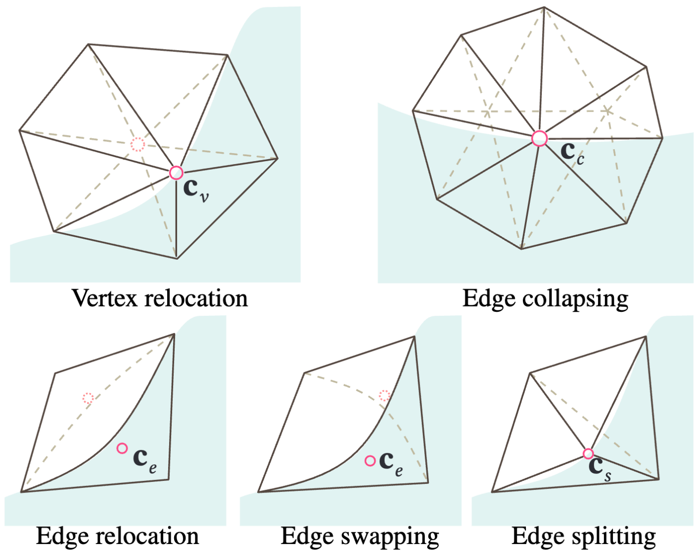


Figure 3: Remeshing operations. Image from Fang et al. [11]

- *Vertex relocation:* Vertex relocation is performed to reduce the color error. The control point on the vertex $\mathbf{c}_v$ is relocated to reduce $\mathcal{E}(\mathcal{M}, \mathcal{I})$. The local region $\mathcal{L}$ consists of the triangles sharing the vertex.

- *Edge relocation:* Edge relocation consists of relocating the control point $\mathbf{c}_e$ on the edge to reduce $\mathcal{E}(\mathcal{M}, \mathcal{I})$. This results in a change of edge curvature. The local region $\mathcal{L}$ consists of the two triangles sharing the edge. The control points are relocated to reduce the color error.

- *Edge flipping (swapping):* Edge flipping is performed to reduce $\mathcal{E}(\mathcal{M}, \mathcal{I})$. The edge is flipped to reduce the color error. The local region $\mathcal{L}$ consists of the two triangles sharing the edge. In case of a curved edge, the control points on the edge are relocated to reduce the color error.

- *Edge splitting:* Edge splitting is performed during the initialization phase to reduce $\mathcal{E}(\mathcal{M},\mathcal{I})$. Edge splitting splits the shared edge of adjacent triangles, creating four triangles and therefore increasing mesh complexity. The edge is first split at its parameric mid-point, creating a vertex called the splitting vertex $\mathbf{v}_s$. The control point $\mathbf{c}_s$ on $\mathbf{v}_s$ is relocated to reduce $\mathcal{E}(\mathcal{R},\mathcal{I})$.

- *Edge collapsing:* Edge collapsing is performed to reduce N_t. An edge $\mathbf{e}$ is collapsed to a vertex $\mathbf{v}_c$. The control point $\mathbf{c}_c$ on $\mathbf{v}_c$ is relocated identically to the process for control point relocation. The control points $\{\mathbf{c}_e\}_{\mathbf{e}\in E_{\mathbf{v}_c}}$ on its one-ring edges $E_{\mathbf{v}_c}$ are also relocated. An error increase due to this operation is allowed, provided that the error increase is within the error bound. Notably, edge collapsing is performed using a priority queue, where edge collapses are prioritized based on the error increase. The edge that, when collapsed, generates the lowest error increase (or highest decrease, in certain cases) is chosen for the next edge collapse. This process allows to circumvent the local minima problem and rapid error increase that occurs when using a greedy approach.

These operations have two main purposes: (1) to reduce color error, which is done through relocation, splitting, and flipping, and (2) to reduce mesh complexity, which is achieved through edge collapsing. While edge collapsing is mainly used to simplify the mesh and therefore often leads to an increase in color error it can also help reduce error in certain cases.

Solving Color Function

During the solving process, the geometry of $\mathcal{M}$ changes with every remeshing operation. Consequently, the color function $P_{\mathbf{t}}$ must be updated to match the new geometry. Solving the color function for a triangle $\mathbf{t} \in T$ consists of finding the coefficients for P_t that minimize the error. Given that pixels are discrete, we can discretize the color error in Equation 3 as follows:

$$\mathcal{E}(\mathbf{t},\mathcal{I}) = \min_{P_{\mathbf{t}}} \sum_{\mathbf{x}\in\mathbf{X_t}} \left(h(\mathbf{x}) - P_{\mathbf{t}}(\mathbf{x})\right)^2, \tag{11}$$

where $X_{\mathbf{t}}$ denotes the positions of the pixels assigned to a triangle $\mathbf{t}$. This is a least squares problem; the optimal color function P_t^* can be solved using the least squares method by solving the following linear system:

$$\nabla \left(\sum_{\mathbf{x}\in\mathbf{X_t}} \left(h(\mathbf{x}) - P_{\mathbf{t}}(\mathbf{x})\right)^2 \right) = 0. \tag{12}$$

Solving this linear system involves assigning pixels to corresponding triangles. To this end, a ray-curve intersection algorithm is used. Horizontal rays are cast for each row of pixels in the image. The intersection points of these rays with the triangle edges are computed analytically, and the pixels between two intersection points are assigned to the corresponding triangle.

Differential Evolution

Remeshing operations should optimize control points to minimize the color error $\mathcal{E}(\mathcal{R},\mathcal{I})$. The error optimization problem for a remeshing operation with a local region $\mathcal{L}$ is defined as follows:

$$\begin{aligned}&\min_{\mathcal{L}} \mathcal{E}(\mathcal{L},\mathcal{I}) + \mathcal{Q}(\mathcal{L})\\ &\text{where } \mathcal{Q}(\mathcal{L}) = \sum_{\mathbf{t}\in\mathcal{L}} \mathcal{Q}_{\mathbf{v}}(\mathbf{t}) + \sum_{\mathbf{t}\in\mathcal{L}} \mathcal{Q}_{\theta}(\mathcal{M},\mathbf{t}) + \sum_{\mathbf{e}\in\mathcal{L}} \mathcal{Q}_{\mathbf{x}}(\mathcal{M},\mathbf{e}).\end{aligned} \tag{13}$$

This is discontinuous and non-differentiable. Therefore, the differential evolution (DE) algorithm is employed as it has no requirement for continuity and differentiability. In this case it is used to find the optimal positions for the control points in $\mathcal{L}$ that minimize the color error. The implementation is taken from Zhang et al. [29], who attempt to solve the same problem using DE. For our case, we consider one control point in a local region $\mathcal{L}$ at a time to optimize. Initialization of N_p control points is done by randomly sampling control point positions from the local region $\mathcal{L}$. The DE algorithm is then applied to iteratively perform mutation, crossover, and selection with mutation scale F and crossover rate C_r to the population until either the error change is below a threshold or a maximum number of iterations is reached.

3.2 Mesh Colors

When mapping a texture to a mesh, typically colors are only assigned to vertices. Mesh colors [28] is an extension of vertex colors where, in addition to assigning colors to vertices, colors are also assigned to regularly-spaced points on edges and faces. This allows for a more complex color gradient to be represented in a single primitive. This essentially allows for detaching the color gradient from geometry, defining a higher texture resolution than the model resolution.

Mesh Colors on Triangular Faces

Mesh colors work by defining a resolution $\mathcal{R}$, which determines the number of regularly-spaced color samples on a triangle face. Color sample positions of mesh colors are determined by an operation that resembles a tessellation procedure in a graphics pipeline that individually subdivides the faces.

Figure 4 depicts the color samples for increasing $\mathcal{R}$. The lowest possible resolution, $\mathcal{R} = 1$, specifies only colors at vertices of the triangle, identical to vertex colors. For $\mathcal{R} = 2$, a color sample is also defined on the middle of each edge. For $\mathcal{R} = 3$, a color sample is defined on the middle of each edge and the center of the triangle. The number of color samples on each vertex, edge, and face is given by

$$\text{colors per vertex} = 1, \tag{14}$$
$$\text{colors per edge} = \mathcal{R} - 1, \tag{15}$$
$$\text{colors per face} = \frac{(\mathcal{R}-1)(\mathcal{R}-2)}{2}, \tag{16}$$
$$\text{total colors} = \frac{(\mathcal{R}+1)(\mathcal{R}+2)}{2}. \tag{17}$$

The nice thing about mesh colors is that color samples are evenly spaced in barycentric coordinates. This allows us to easily compute the surface position of each color sample using the index of the color sample and vice versa. The position $\mathbf{P}_{ij}$ of a color sample on a face with resolution $\mathcal{R}$ in barycentric coordinates is given by

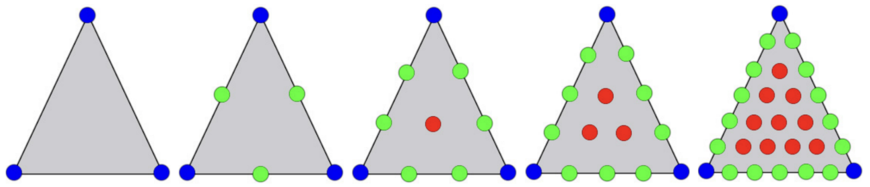

(a) $\mathcal{R} = 1$ (b) $\mathcal{R} = 2$ (c) $\mathcal{R} = 3$ (d) $\mathcal{R} = 4$ (e) $\mathcal{R} = 6$

Figure 4: Mesh color resolutions. Image from Yuksel et al. [28]

$$\mathbf{P}_{\mathrm{ij}} = \left[\frac{i}{\mathcal{R}}, \frac{j}{\mathcal{R}}, 1 - \frac{i+j}{\mathcal{R}}\right], \tag{18}$$

where i and j are integers such that $0 \leq i + j \leq \mathcal{R}$. A color sample at index $[i, j]$ is denoted as C_{ij}.

Evaluating Color

For a point $\mathbf{P}$ in barycentric coordinates on the surface of a triangle, the color value is determined by linear interpolation of the colors of the three nearest color samples. Equation 18 denotes color samples are evenly spaced in barycentric coordinates and positioned based on their indices. Therefore, the three closest color samples to a point on the triangle surface can be determined using barycentric coordinates. The color value at a point $\mathbf{P}$ on the surface of a triangle $\mathbf{t}$ is determined by the following steps:

1. Multiply the barycentric coordinates of $\mathbf{p}$ by the resolution $\mathcal{R}$: $[i, j, k] = \mathcal{R} \cdot \mathbf{P}$.
2. Take the integer part $\mathbf{B} = [i, j, k] = \lfloor \mathcal{R} \cdot \mathbf{P} \rfloor$ and the fractional part $\mathbf{w} = [\mathbf{w}_i, \mathbf{w}_j, \mathbf{w}_k] = \mathcal{R} \cdot \mathbf{P} - \mathbf{B}$.
3. There are three cases to consider:
   (a) If $\mathbf{w} = 0$, the point P is on a color sample position C_{ij}, and the color value is the color of the color sample at the position.
   (b) If $\mathbf{w}_i + \mathbf{w}_j + \mathbf{w}_k = 1$, the nearest color values are the three color samples at positions C_{ij}, $C_{i(j+1)}$, and $C_{(i+1)j}$.
   (c) If $\mathbf{w}_i + \mathbf{w}_j + \mathbf{w}_k = 2$, the nearest color values are the three color samples at positions $C_{i+1,j+1}$, $C_{i+1,j}$, and $C_{i,j+1}$.
4. Finally, the color value $\mathbf{c}$ at the point $\mathbf{P}$ is the weighted average of the three color samples: $\mathbf{c} = \mathbf{w} \cdot C_{ij}$.

4 Methodology

We build upon the Error-Bounded Image Triangulation method as described in Section 3.1. The key problem, minimizing N_t under a color error constraint and quality constraint, remains the same. A low-complexity mesh is achieved by a series of local remeshing operations. Optimal vertex and control point locations for remeshing operations are achieved by a differential evolution algorithm. Minimizing error between the original image and generated mesh involves (1) assigning pixels to faces, and (2) computing the best-fitting constant, linear, or quadratic color function by solving a linear system of equations.

Our approach adaptively adds mesh colors into this method. This provides an additional approach to coloring the mesh. Mesh colors allow for non-uniform face resolutions, allowing for higher color sample density in regions of high color variation. Moreover, our approach allows mesh colors to work in conjunction with the original color functions by specifying the color function on a per-face basis. Using this property, faces with high color variation can be colored using mesh colors, while faces with low color variation can be colored using the original color functions.

4.1 Mesh Simplification using Mesh Colors

We introduce an additional parameter $\mathcal{R}$, which denotes the maximum resolution that a triangle $\mathbf{t}$ in $\mathcal{M}$ can have. This parameter is used to prevent the algorithm from increasing the resolution of a face indefinitely. This introduces a new quality constraint defining that each triangle's mesh color resolution in the mesh does not exceed a user-specified threshold:

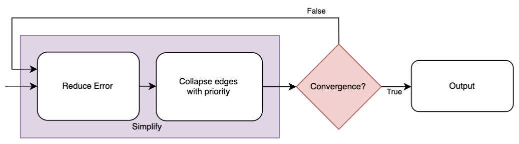
$$\mathcal{Q}_{\mathcal{R}}(\mathbf{t}) = \begin{cases} 0, & \text{if } \mathcal{R}(\mathbf{t}) \leq \mathcal{R} \\ +\infty, & \text{otherwise,} \end{cases} \tag{19}$$

where $\mathcal{R}(\mathbf{t})$ is the resolution of triangle $\mathbf{t}$. The total quality constraint $\mathcal{Q}(\mathcal{M})$ is defined as the sum of the quality constraints for each triangle, including validity, angle, curvature and resolution constraints:

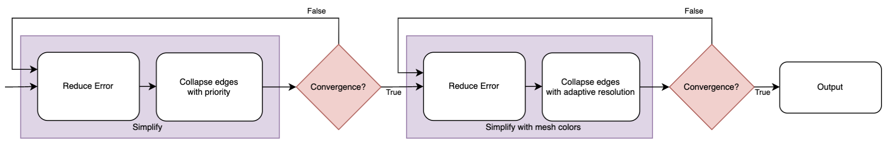
$$\mathcal{Q}(\mathcal{M}) = \sum_{\mathbf{t}\in T} \mathcal{Q}_v(\mathbf{t}) + \sum_{\mathbf{t}\in T} \mathcal{Q}_\theta(\mathcal{M}, \mathbf{t}) + \sum_{\mathbf{e}\in E} \mathcal{Q}_\kappa(\mathcal{M}, \mathbf{e}) + \sum_{\mathbf{t}\in T} \mathcal{Q}_{\mathcal{R}}(\mathbf{t}). \tag{20}$$

Updated pipeline

With the addition of mesh colors, the pipeline is updated as follows (see Figure 5): The initialization and error reduction phase remain unchanged. The simplification procedure starts off the same as well; the mesh is simplified as much as possible using the original color functions; no mesh colors are applied yet. The original pipeline terminates after this step converges. We propose an additional step, performing edge collapsing with mesh colors to further reduce the mesh complexity. In this step, vertex relocation, edge relocation, and edge flipping are again used to provide enough room between the error and error bound to perform the collapsing operation. Afterwards we propose an adapted edge collapsing procedure that incorporates mesh colors. This procedure allows collapse edges that were impossible in the previous step due to the error reduction mesh colors creates. This process is described in Algorithm 1.



(a) A zoomed-in view of the simplification step in the original pipeline by Fang et al. [11]. For the full pipeline, refer to Figure 2.


False
Reduce Error
Collapse edges with priority
Simplify
Convergence?
True
False
Reduce Error
Collapse edges with adaptive resolution
Simplify with mesh colors
Convergence?
True
Output

(b) The proposed pipeline, incorporating mesh colors into the simplification step.

Figure 5: Comparison between the original and proposed pipeline. (a) The original method by Fang et al. [11] terminates once edge collapsing has converged. (b) The proposed approach introduces an additional simplification step where mesh colors are integrated during edge collapsing, further reducing mesh complexity.

This addition introduces new convergence criteria within the simplification process. Both edge collapsing steps—priority queue-based collapsing and adaptive resolution collapsing—share the same termination conditions. This means that first priority queue-based edge collapsing is performed until the number of collapsed edges in a single iteration is below a threshold or a maximum number of iterations is reached. Then, the adaptive resolution edge collapsing is performed the number of collapsed edges in a single iteration is below a threshold or a maximum number of iterations is reached. Finally, the process is fully converged and the simplification process is terminated.

Edge Collapsing with Adaptive Resolution

We create an additional edge collapsing procedure to incorporate mesh colors, as described in Algorithm 1. The implementation is as follows: Initially, the original color function is used for each triangle; mesh colors is turned off for each face. The edge collapsing process iterates over all edges e in the mesh $\mathcal{M}$. For each edge, edge collapsing is performed, reducing the number of faces and updating the mesh topology. After collapsing an edge, the optimal locations for the remaining control points, consisting of the control point $\mathbf{c}_c$ of the collapsing vertex $\mathbf{v}_c$ and the control points $\{\mathbf{c}_e\}_{\mathbf{e}\in E_{\mathbf{v}_c}}$ on its one-ring edges $E_{\mathbf{v}_c}$, are determined using the Differential Evolution algorithm.

The error $\mathcal{E}$ is then computed for the updated mesh according to Equation 4 to evaluate the quality of the simplification. If the error is within the prescribed error bound $\mathcal{E}_{\mathcal{I}}$, the collapse is accepted, and the algorithm proceeds to the next edge.

However, if the error exceeds $\mathcal{E}_{\mathcal{I}}$, the algorithm dynamically increases the resolution of the faces containing the highest error. This is achieved by identifying the face f in the local region around the collapse with the highest error, and its current resolution $f_{\mathcal{R}}$ is below the maximum allowable resolution $\mathcal{R}_{\max}$. If such a face is found, the mesh color resolution of f is increased. The error is recalculated with the updated mesh colors.

The process of increasing resolution and recalculating the error is repeated until the error $\mathcal{E}$ falls within the acceptable range or the resolution limit $\mathcal{R}$ is reached. If the error is reduced to an acceptable level, the collapse is accepted. Otherwise, the collapse is reverted, preserving the original topology of the mesh.

Algorithm 1 Edge Collapsing with Adaptive Resolution

```
1: **Input:** Mesh $M$, error bound $\mathcal{E}_\mathcal{I}$, maximum resolution $\mathcal{R}$
2: **Output:** Simplified mesh $M'$
3: for each edge $e \in M$ do
4:     $\mathcal{L} \leftarrow$ local region around edge $e$
5:     Perform edge collapse on $e$
6:     Find optimal control point location for $\mathbf{c}_c$ and $\{\mathbf{c}_e\}_{\mathbf{e} \in E_{\mathbf{v}_c}}$ using DE
7:     Compute error $\mathcal{E}$ on mesh
8:     if $\mathcal{E} < \mathcal{E}_\mathcal{I}$ then
9:         Accept the collapse and proceed to the next edge
10:    else
11:        do
12:            Find triangle $\mathbf{t}$ with highest error in $\mathcal{L}$ region with $\mathcal{R}(\mathbf{t}) \leq \mathcal{R}$
13:            if $\mathbf{t}$ is not found then
14:                Break
15:            end if
16:            Increase resolution: $\mathcal{R}(\mathbf{t}) \leftarrow \mathcal{R}(\mathbf{t}) + 1$
17:            Recompute $\mathcal{E}$
18:        while $\mathcal{E} > \mathcal{E}_\mathcal{I}$
19:        if error $\leq$ error_bound then
20:            Accept the collapse
21:        else
22:            Revert the collapse
23:        end if
24:    end if
25: end for
```

4.2 Mesh Color Assignment

Each remeshing operation changes the geometry of the mesh, and therefore requires that the color for each triangle $\mathbf{t}$ in the local mesh $\mathcal{L}$ be updated. The original color functions are used to color the mesh. The color of each face is determined by solving a least squares problem for color fitting. For triangles that have been assigned mesh colors, the color for each mesh color sample C_{ij} is determined by bilinearly interpolating the four closest pixels in the input image.

As an alternative, least squares fitting could also be implemented to define color on each mesh color sample $C_{i,j}$. However, it was not implemented in our method. Bilinear interpolation was easier to implement and provided a reasonable balance between accuracy and computational efficiency. Moreover, the focus of our work is on demonstrating that mesh simplification can achieve good results with mesh colors, rather than striving for complete optimization in every aspect

4.3 Pixel Assignment Algorithm

Our method implements a GPU-based approach for pixel-to-face assignment, aiming to improve the efficiency of this process compared to the

CPU-based ray-curve algorithm proposed by Fang et al. [11]. Pixels are assigned to faces to solve the least squares problem for color fitting and error computation per face (see Chapter 3.2). During remeshing operations, the geometry of $\mathcal{M}$ is updated, and pixels need to be reassigned. This process is achieved using a lookup renderer. Each face is assigned a unique color and rendered using shaders. Quadratic meshes are efficiently rendered using tessellation shaders. The resulting image is read back to the CPU and looped over pixel by pixel to assign each pixel to the corresponding face. Therefore, our method is only partially GPU-based, as the pixel assignment process is still performed on the CPU.

A fully GPU-based solution is provided by [12], where triangle color assignment is implemented using compute shaders. Unfortunately, compute shaders are not supported on the system used for developing (macOS). These limitations impact the overall speed and prevent full parallelization.

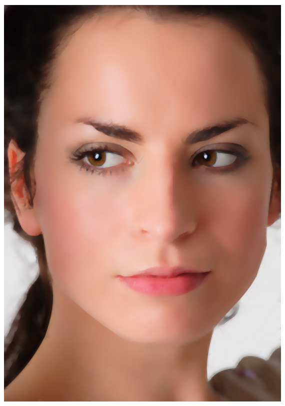
Nevertheless, the current methodology provides a structured framework for future optimization. Once fully parallelized and supported by appropriate hardware, this approach could minimize computation time, particularly for large-scale images.

5 Results

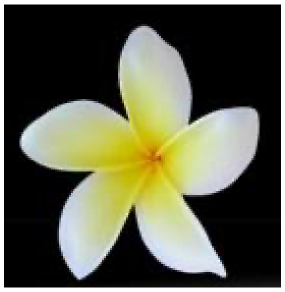
Below we present the results and discussion of our method on a variety of images of varying sizes. Our method does not generate a mesh itself, but improves upon an initial mesh. Initial meshes are obtained using the method by Hettinga et al. [12]. The images used for evaluation and the number of triangles in their initial mesh can be seen in Figure 6. We also evaluate the performance of our method. Each result is obtained using Differential Evolution parameters $F = 0.7$ and $C_r = 0.9$, which Fang et al. [11] identified as optimal. A smaller population size of $N_p = 10$ is used to reduce computation time.

(a) Curve (1033 × 1033), $N_{tri} = 248$

(b) Cat (640 × 427), $N_{tri} = 6778$


(c) Face (882 × 1262), $N_{tri} = 2706$

(d) Fish (910 × 856), $N_{tri} = 4489$


(e) Flower (141 × 144), $N_{tri} = 632$

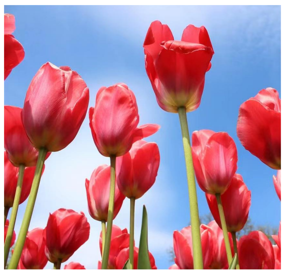

(f) Tulips (855 × 815), $N_{tri} = 6658$

Figure 6: Input images used to evaluate the proposed method. The number of triangles (N_{tri}) for each initial mesh is indicated in the subfigure captions.

An illustrative example of the correct working of our method is shown in Figure 7. Our method is able to reduce the number of triangles by a factor of $\frac{248}{25} = 9.92$. Our extracted mesh is able to retain the sharpness of the curve in the input image.

5.1 Comparison with Existing Method

We test the algorithm against the method by Fang et al. [11]. We show that our method consistently generates meshes with fewer triangles while retaining the important features of the image. Figure 8 shows a comparison of generated meshes for the face image. The input image contains subtle gradients and sharp edges. Our method captures the subtle gradients, such as those on the forehead, more effectively, creating a more visually pleasing

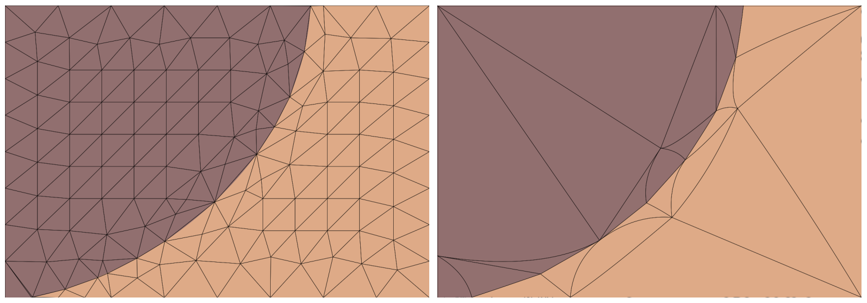

(a) Input mesh $N_t = 248$ (b) Generated mesh $N_t = 25$

Figure 7: A simple example of the simplification of a mesh. The input mesh contains (left) contains a lot of redundant triangles. The output mesh (right) has fewer triangles and retains the most imporant feature of the image.

result. For certain sharp edges, such as the left eye, our method combines constant colors with mesh colors, resulting in a washed-out region where the triangles fail to accurately represent the sharp edge. This is a limitation of our method, which is a result of the combination of the differential evolution algorithm and the mesh colors. A large mesh color resolution allows to capture more complexity on a single face, thus enabling triangles to overlap image features and still create a small error. The differential evolution algorithm is rather naive, as it does not explicitly take into account image features, which can cause the algorithm to find solutions that satisfy the error bound but do not capture the image features well. Another artifact can be observed in the same region, where a face in the white of the eye is assigned a constant color, while surrounding triangles are assigned mesh colors. This creates a discontinuity in the region which is not visually pleasing. Errors are accumulated per face, and do not consider discontinuities between faces, allowing these artifacts to appear. Despite these shortcomings, our method can be considered to create a more realistic output image and a more visually pleasing result as the gradients are better captured. Figure 9 shows a comparison of the input images, results by Fang et al. [11], and our method. Our method consistently generates meshes with fewer triangles while retaining the important features of the image.

5.2 Resolution Parameter

Figure 10 presents the results for various resolutions applied to the cat image. The artifacts mentioned above are increasingly visible as the resolution parameter $\mathcal{R}$ is increased. Exemplary is the left ear, which is increasingly worse segmented. The large mesh color resolution creates a low enough error to allow triangles to be collapsed and overlap the ear. We also observe the discontinuities created by faces containing a constant color function or mesh colors create increasingly worse artifacts for increasing $\mathcal{R}$. We note that for this particular example $\mathcal{R} = 4$ achieves the best trade-off between the number of triangles and the quality of the image. Moreover, we note that the choice of $\mathcal{R}$ must be carefully considered to create a desired result.

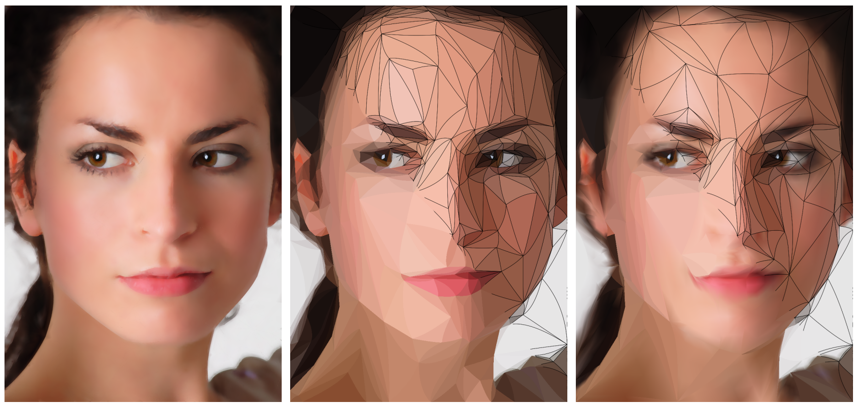

(a) Input image (b) $N_t = 616$, $\mathcal{R} = 0$ (c) $N_t = 373, \mathcal{R} = 5$

Figure 8: Comparison with [11] for RMSE = 7. Our method (right) achieves lower complexity for the same error bound than the method by Fang et al. (middle). For both methods, the constant color function is used.

5.3 Error Bound

We evaluate our method for varying error bounds $\mathcal{E}_{\mathcal{I}}$. Figure 11 shows the results for the flower image. We observe that increasing the error bound reduces mesh complexity in the mesh and the overall quality of the image. For RMSE = 4 and RMSE = 6 (Figure 11b, and Figure 11c, respectively), the image is able to retain the sharpness of the flower petals. For RMSE = 8 and RMSE = 10 (Figure 11d, and Figure 11e, respectively), discontinuities between triangles can clearly be observed. A limitation in our mesh color assignment scheme can also clearly be observed. Consider the left petal in Figure 11e, a black pixel outside the region of the petal is bilinearly interpolated for the vertex mesh color, causing an unwanted artifact. We note that the choice of error bound must be carefully considered to create a desired output.

5.4 FLIP Metric

Visual similarity metrics are used to evaluate the quality of the generated mesh. Our method tries to minimize the root mean squared error (RMSE) between the original image and the generated mesh. A limitation of RMSE is that it does not take into account the human perception of the image. This can lead to situations where the RMSE is low, but the image is not visually pleasing.

Therefore, we use the FLIP metric [3] to evaluate the visual quality of the generated meshes. FLIP is a perceptual quality measure designed to evaluate perceived differences between images based on the human visual system. Unlike single-value metrics, FLIP generates a per-pixel difference map, offering a detailed spatial representation of visual discrepancies. For a quantitative summary, we report the mean FLIP error for the images.

In our evaluation, a PPD (Pixels per Degree) value of 33.5 was used, which corresponds to viewing a 70 cm wide screen with a horizontal resolution of 1920 pixels from a distance of 70 cm. This setup aligns with common viewing conditions.

Table 2 compares the FLIP metric for our method and the approach by

Fang et al. [11]. Our method consistently achieves similar or lower FLIP error values, indicating that the generated meshes are visually closer to the reference images. While the improvement in visual quality is modest, it demonstrates that our method can produce slightly better results compared to the method by Fang et al. [11].

Table 2: Comparison of mean FLIP error values between our method and Fang et al. [11].

Image	**FLIP (Ours)**	**FLIP (Fang et al. [11])**	**RMSE**
Cat ($\mathcal{R} = 4$, Figure 6b)	0.1251	0.147	10
Face (Figure 6c)	0.108	0.121	7
Fish (Figure 6d)	0.110	0.110	7.5
Flower (Figure 6e)	0.07	0.212	8
Tulips (Figure 6f)	0.117	0.117	8.5

5.5 Performance

We compare the performance of our method with the method by Fang et al. [11]. The results are shown in Table 3. Our method consistently outperforms the method by Fang et al. [11] in terms of runtime. The method by Fang et al. [11] is slower due to pixel assignment and rendering being implemented on the CPU. Our method is faster as it partially implements this on the GPU, resulting in a faster runtime.

However, the proposed method is still slow and not suitable for real time applications. Running the complete pipeline from error reduction to simplification can take up several minutes to hours for a single image, depending on mesh and image size. Extensive profiling has not been performed. However, experimental results have pointed out the following bottlenecks; (1) Reading back the rendered image from the framebuffer object and (2) accumulating the error over faces. As mentioned in Section 4.3, our method is partially moved to the GPU, and substantial overhead arises from transferring data between the CPU and GPU. Since performance was not a primary concern for our method, a full GPU implementation was beyond the scope of this work. Reducing the overhead from data transfer by fully implementing color assignment or error accumulation using compute shaders could significantly improve runtime performance.

Table 3: Runtime comparison between our method and Fang et al. [11]. All results are obtained using a MacBook Pro with an Apple M1 Pro chip and 16GB RAM.

Image	**Ours**	**Fang et al. [11]**
Cat (Figure 6b)	24m 10s	29m 34s
Face (Figure 6c)	13m 48s	51m 51s
Fish (Figure 6d)	49m 52s	82m 59s
Flower (Figure 6e)	2m 30s	2m 50s

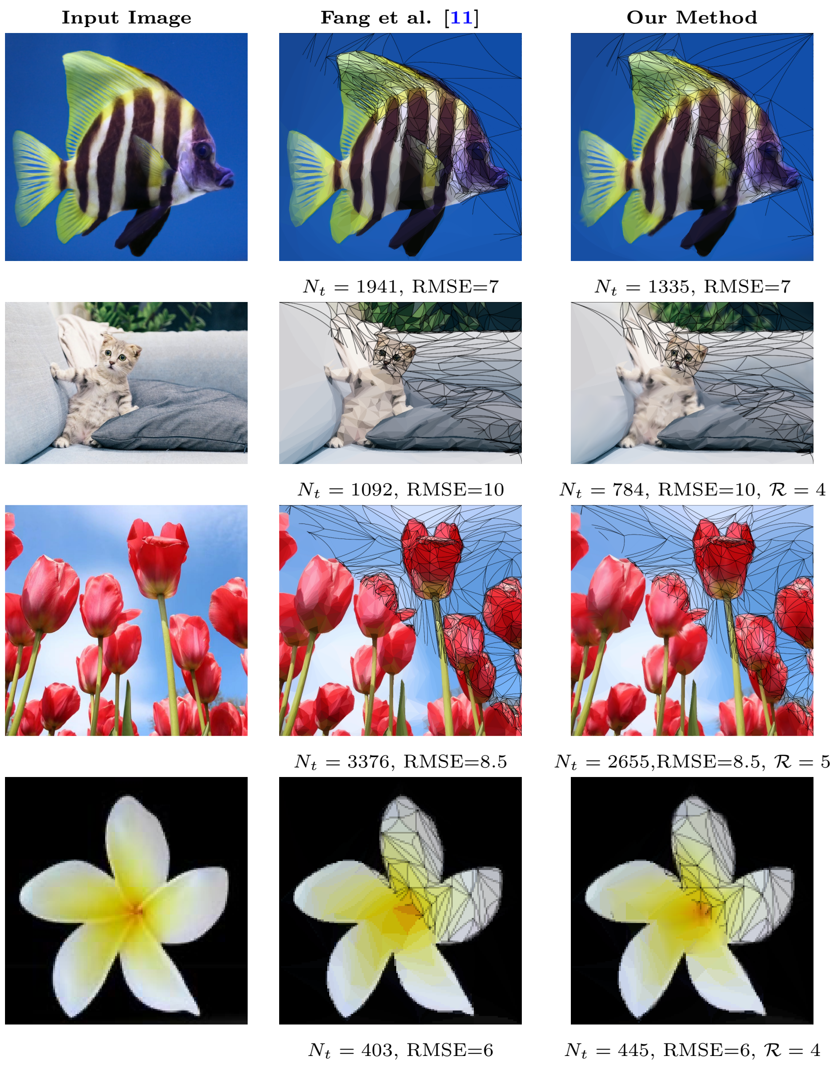
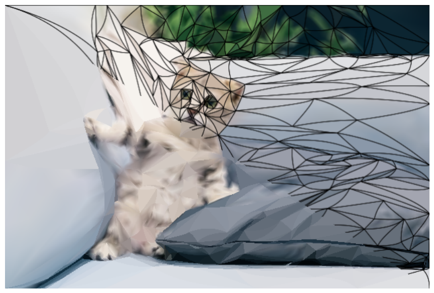
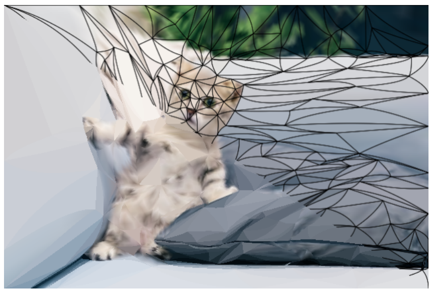


Figure 9: Comparison of input images, results by Fang et al. [11], and our method. Compared to Fang et al. [11], our method consistently generates meshes with fewer triangles while retaining the important features of the image.

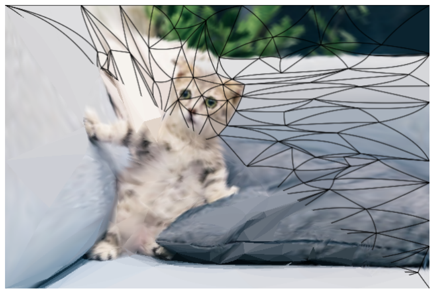
(a) Input image



(b) $\mathcal{R} = 4, N_t = 784$



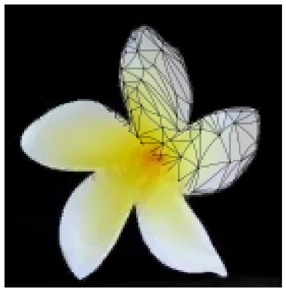
(c) $\mathcal{R} = 8, N_t = 638$



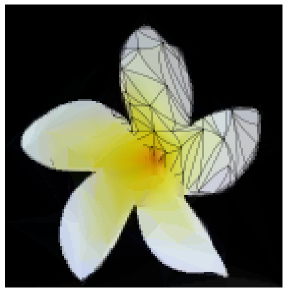
(d) $\mathcal{R} = 16, N_t = 394$

Figure 10: Results for different resolutions on the cat image. The error bound is set to RMSE = 10. Increasing the resolution parameter $\mathcal{R}$ consistently reduces the number of triangles in the mesh. Segmenation of image features results in fewer and larger segments of image features, with less precise boundaries.

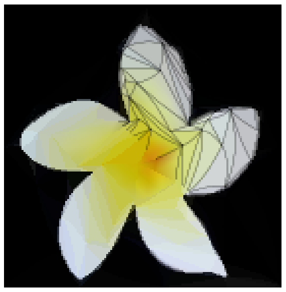
(a) Input image



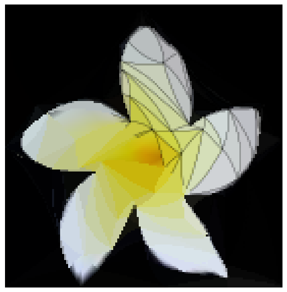
(b) RMSE = 4, N_{faces} = 714



(c) RMSE = 6, N_{faces} = 403



(d) RMSE = 8, N_{faces} = 264



(e) RMSE = 10, N_{faces} = 176

Figure 11: Results for different error bounds on the flower image. The mesh color resolution is set to $\mathcal{R} = 4$.

6 Conclusion

In this thesis, we proposed a method for adaptively adding mesh colors to the method by Fang et al. [11]. Mesh colors can represent more complex color gradients than the original method, allowing for a lower mesh complexity under the same color error bound. Overall, it performed well as it is able to reduce the mesh complexity of the original method by a considerable amount under the same color error bound. However, various improvements can be made to the method. The current collapsing method is naive and does not take into account the underlying image features. As a result, the method can distort edges in the image. Visual artifacts can also be observed between neighboring faces, as mesh colors are not shared between edges.

Moreover, while our method proves to be faster than the method by Fang et al. [11], it is certainly not ready for real-time applications. Fully moving the color fitting and error accumulation to the GPU could potentially reduce the runtime significantly. Ultimately, the addition of mesh colors to the method by Fang et al. [11] is promising. For the method to be viable in practical applications, further research is needed. The suggestions outlined in Chapter 7 should be considered to further improve the method.

7 Future work

Various improvements can be made to our proposed methods. They can be divided into two categories: improvements that further optimize the generated mesh and improvements on the performance of the algorithm.

7.1 Mesh Optimization

- Supporting cubic Bézier curves: Our method currently supports quadratic Bézier curves. Extending the method to support cubic Bézier curves would allow for more complex shapes to be vectorized with fewer triangles, reducing complexity.

- Mesh color fitting: Our method currently only supports bilinear interpolation of the underlying image. Extending the method to support solving a least squares problem for the mesh colors would allow for more accurate color representation.

- Improving adaptive resolution: Our method currently adaptively changes the resolution for faces in a local neighborhood of a remeshing operation. A number of improvements can be made to this operation. For example, the resolution could be adapted based on the area of the face, the curvature of the face, or the color gradient of the face. Furthermore, the adaptive updating could be extended to consider all faces in the mesh, not just those in the neighborhood of a remeshing operation, so that the face in the global mesh that contributes the most to the error can be updated with a higher resolution. More intelligent approaches could also detect regions of the image with high texture detail and increase the resolution in those regions.

- More intelligent approaches to remeshing, taking into account image features: Our method currently remeshes faces based on the error in the image and is quite naive, not taking into account the underlying image features. More intelligent approaches could be developed that take into account and constrain the underlying image features. For example, the method could detect and constrain edges to ensure that they are not distorted by the collapsing operation.

7.2 Performance Improvements

- Further GPU acceleration of pixel assignment and color fitting: Our method has partly moved this to the GPU, but there are still parts executed on the CPU. The pixel assignment operation is using a lookup renderer. The result is then used to solve a least squares problem for color fitting on the CPU. This process could be fully moved to the GPU using compute shaders similar to the method by Hettinga et al. [12].

- Parallelization of the remeshing operation: Our method currently remeshes faces sequentially. This operation could be parallelized for non-overlapping regions in the mesh to improve performance.

References

[1] Michael D. Adams. A flexible content-adaptive mesh-generation strategy for image representation. *IEEE Transactions on Image Processing*, 20(9):2414–2427, 2011.

[2] Adobe Illustrator. Using meshes in adobe illustrator. https://helpx.adobe.com/illustrator/using/meshes.html, n.d. Accessed: 2024-12-03.

[3] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), August 2020.

[4] S. D. Baksteen, Gerben Hettinga, J. Echevarria, and Jiri Kosinka. *Mesh Colours for Gradient Meshes.* Eurographics Association, October 2021. Smart Tools and Applications in Graphics 2021, STAG ; Conference date: 26-10-2021 Through 29-10-2021.

[5] Pieter J. Barendrecht, Martijn Luinstra, Jonathan Hogervorst, and Jiří Kosinka. Locally refinable gradient meshes supporting branching and sharp colour transitions: Towards a more versatile vector graphics primitive. *Visual computer*, 34(6):949–960, June 2018. 35th Computer Graphics International conference (CGI) ; Conference date: 11-06-2018 Through 14-06-2018.

[6] Simon Boyé, Pascal Barla, and Gaël Guennebaud. A vectorial solver for free-form vector gradients. *ACM Trans. Graph.*, 31(6), November 2012.

[7] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[8] Wen Dai, Tao Luo, and Jianbing Shen. Automatic image vectorization using superpixels and random walkers. 2:922–926, 2013.

[9] F. Davoine, M. Antonini, J.-M. Chassery, and M. Barlaud. Fractal image compression based on delaunay triangulation and vector quantization. *IEEE Transactions on Image Processing*, 5(2):338–346, 1996.

[10] James Richard Diebel. *Bayesian image vectorization: the probabilistic inversion of vector image rasterization.* PhD thesis, Stanford, CA, USA, 2008. AAI3332816.

[11] Zhi-Duo Fang, Jia-Peng Guo, Yanyang Xiao, and Xiao-Ming Fu. Error-bounded image triangulation. *Computer Graphics Forum*, 42(7):e14967, 2023.

[12] Gerben Hettinga, Jose Echevarria, and Jiri Kosinka. *Efficient Image Vectorisation Using Mesh Colours.* Eurographics Association, 2021. Italian Chapter Conference 2021 : Smart Tools and Apps in Graphics ; Conference date: 28-10-2021 Through 29-10-2021.

[13] Gerben J. Hettinga, Jose Echevarria, and Jiří Kosinka. Adaptive image vectorisation and brushing using mesh colours. *Computers & Graphics*, 105:119–130, 2022.

[14] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3), July 2009.

[15] Kai Lawonn and Tobias Günther. Stylized image triangulation. *Computer Graphics Forum*, 38(1):221–234, 2019.

[16] Henrik Lieng, Jiří Kosinka, Jingjing Shen, and Neil A. Dodgson. A colour interpolation scheme for topologically unrestricted gradient meshes. *Computer Graphics Forum*, 36(6):112–121, 2017.

[17] Shufang Lu, Wei Jiang, Xuefeng Ding, Craig S. Kaplan, Xiaogang Jin, Fei Gao, and Jiazhou Chen. Depth-aware image vectorization and editing. *Vis. Comput.*, 35(6–8):1027–1039, June 2019.

[18] Yiting Ma, Xuejin Chen, and Yu Bai. An interactive system for low-poly illustration generation from images using adaptive thinning. pages 1033–1038, 07 2017.

[19] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, Holger Winnemöller, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. *Commun. ACM*, 56(7):101–108, July 2013.

[20] Brian Price and William Barrett. Object-based vectorization for interactive image editing. *Vis. Comput.*, 22(9):661–670, September 2006.

[21] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11–es, July 2007.

[22] Xingze Tian and Tobias Günther. A survey of smooth vector graphics: Recent advances in representation, creation, rasterization, and image vectorization. *IEEE Transactions on Visualization and Computer Graphics*, 30(3):1652–1671, 2024.

[23] Wanyi Wang, Zhonggui Chen, Lincong Fang, and Juan Cao. Curved image triangulation based on differentiable rendering. *Computer Graphics Forum*, 43(7):e15232, 2024.

[24] Guangshun Wei, Yuanfeng Zhou, Xifeng Gao, Qian Ma, Shiqing Xin, and Ying He. Field-aligned quadrangulation for image vectorization. *Computer Graphics Forum*, 38(7):171–180, 2019.

[25] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, New York, NY, USA, 2009. Association for Computing Machinery.

[26] Yanyang Xiao, Juan Cao, and Zhonggui Chen. Image representation on curved optimal triangulation. *Computer Graphics Forum*, 41(6):23–36, 2022.

[27] Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Trans. Graph.*, 33(6), November 2014.

[28] Cem Yuksel, John Keyser, and Donald H. House. Mesh colors. *ACM Transactions on Graphics*, 29(2):15:1–15:11, 2010.

[29] Wen-Xiang Zhang, Qi Wang, Jia-Peng Guo, Shuangming Chai, Ligang Liu, and Xiao-Ming Fu. Constrained remeshing using evolutionary vertex optimization. *Computer Graphics Forum*, 41(2):237–247, 2022.

[30] Wenli Zhang, Shuangjiu Xiao, and Xin Shi. Low-poly style image and video processing. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 97–100, 2015.

[31] Shuang Zhao, Frédo Durand, and Changxi Zheng. Inverse diffusion curves using shape optimization. *IEEE Transactions on Visualization and Computer Graphics*, 24(7):2153–2166, 2018.

[32] Hailing Zhou, Jianmin Zheng, and Lei Wei. Representing images using curvilinear feature driven subdivision surfaces. *IEEE Transactions on Image Processing*, 23(8):3268–3280, 2014.