Content-Aware Pixel Art Rendering with Non-Square Pixels

Zane Z. Wang
zz2wang@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Jose Echevarria
echevarr@adobe.com
Adobe Research
New York, New York, United States

Niviru Wijayaratne
nwijayaratne@adobe.com
Adobe Research
San Jose, California, United States

Craig S. Kaplan
csk@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

(a) original (b) naïve filtering (c) optimized tile filters (d) full colour result (e) quantized result

Figure 1: (a) A comically edited photograph of the oil painting *Beethoven with the Manuscript of the Missa Solemnis,* by Joseph K. Stieler. (b) Image rendered into a uniform regular hexagon tiling with naïve filtering. (c) Visualization of our optimized filter support shapes. (d) Full colour result with unwarped tiling geometry, exhibiting considerable improvements to the sharpness of image features, such as the lapel, scarf, and eyes. (e) Additional result after value mapping by lightness to an 8-colour palette generated by our program.

Abstract

Characterized by a deliberately limited resolution and colour palette, pixel art is an exercise in the conveyance of visual information with a limited number of samples. We present the first computational solution to a novel problem in computer graphics: the rendering of images in the pixel art style on arbitrary pixel shapes—restricted only by their ability to tile the plane—while simultaneously respecting image features. We formulate the non-square (or *any-shape*) pixel art rendering task as an energy minimization problem over tile-shaped filter supports, given a conventional raster image and geometric tiling data as input. We then demonstrate that our method produces images with superior qualitative and quantitative properties in comparison with naïve methods. Finally, we offer some stylization and artist interaction procedures for the final render, such as colour palette generation and simple adjustment of the filter vertices. This method has the potential to be useful in several artistic contexts, such as the creation of highly stylized portraiture and landscapes, and authoring of image and video for real hardware displays that use non-square pixels.

CCS Concepts

• **Computing methodologies** → **Non-photorealistic rendering**; *Image processing*; **Computer graphics**; • **Applied computing** → *Media arts.*

Keywords

Pixel Art, Stylized Rendering, Tilings, Image Abstraction, Gradient Descent, Mesh Smoothing, Rasterization

ACM Reference Format:
Zane Z. Wang, Jose Echevarria, Niviru Wijayaratne, and Craig S. Kaplan. 2026. Content-Aware Pixel Art Rendering with Non-Square Pixels. In *Proceedings of Graphics Interface 2026 (GI '26).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

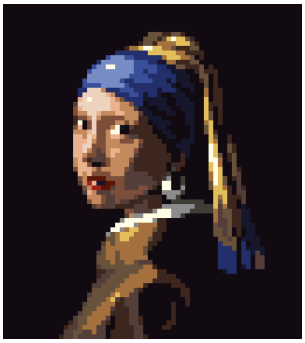
GI '26, Waterloo, ON

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Pixel art is a well-studied art form that arose from technical limitations on computing hardware beginning in the early 1980s, and whose defining characteristics are a limited number of pixels and a limited colour palette. In the modern practice of pixel art, the geometric shape of each pixel (in this case, a square) is an important contributor to the appearance of the art. Pixel shape and arrangement have an effect beyond simply bounding regions of constant

(a) Vermeer Portrait



(b) Pixel Art

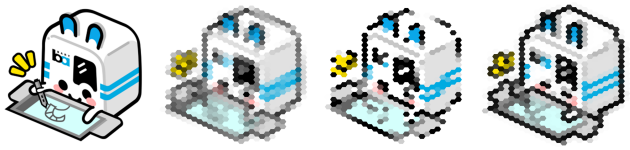
Figure 2: Side-by-side view of a) The 1665 oil painting *Girl with a Pearl Earring* by Johannes Vermeer, and b) a pixel art study of the same painting by artist Pablo Rodriguez *(www.reffpixels.com; used with permission).* To maintain the perceptual clarity of thin features like the eyes, mouth, and earring, they must be be abstracted into new large pixel representations in a way that distorts their shape.

colour: the geometry of our real or virtual display hardware influences how artists choose to represent the visual features of a pixel artwork's subject. For example, on a square pixel array, it is much harder to display diagonal lines and smooth curves than straight vertical or horizontal lines, so careful antialiasing and/or shape representation techniques must be used (Figure 2).

We are interested in exploring the extension of pixel art to other pixel shapes that tile the plane. Depending on the way that the polygonal shape(s) of a given tiling fit together, there may be certain shapes in the image subject that are much easier (or much more difficult!) to depict. Thus, one way of stating the non-square pixel artist's task is as follows: given an *image subject* and a tiling by generalized pixels of arbitrary shapes, translate the *perceptually important elements* of the image subject into their closest *analogous representations* in the tiling.

To more precisely state the problem, a *non-square pixelation* algorithm is given a raster image U and the geometric data for a finite set $\mathcal{T}$ of connected polygonal tiles. Our goal is to choose a colour for each tile $T \in \mathcal{T}$ such that $\mathcal{T}$ best represents the artistic and perceptual features of the input image U. For future reference, we refer to such an image drawn on a given tiling of the plane interchangeably as *non-square pixel art*, or a *tiling-space image*—i.e., any image where the basic unit of colour is a tile.

Naïve methods of performing this translation prove inadequate at the low resolutions that interest us—i.e., those where the shapes of the pixels are clearly visible. One might simply try to average all the pixels in the image domain U that lie underneath each tile T, or sample U at a single designated point within each tile. Figure 3 illustrates the artifacts that are produced by averaging and sampling for a cartoon-style input, which we should expect to be a good subject for (non-square) pixelation.



(a) original art (b) tile average (c) point (d) ours

Figure 3: a) The BART mascot, Barty, rendered with naïve methods (b, c) onto a hex tiling. b) Uniform averaging is extremely blurry, particularly when tiles cross colour boundaries in the source image; while c) point sampling from the centroid is riddled with holes and missed details. d) Our method preserves visual detail and effectively translates the input into the new pixel space. (No vertex brushing was applied to produce this figure. *Input image © San Francisco Bay Area Rapid Transit; used with permission.*)

We propose a first computational solution to this novel rendering problem, with the following contributions:

- We formulate the non-square pixelation task as an energy minimization problem over geometric tiling data, which aims to minimize the variance of the image data used to determine the colour of each tile. We then demonstrate both qualitative and quantitative improvements over naïve methods.
- We provide some basic means for artists to interact with the non-square pixelation process, such as mesh translation, rotation, and scaling, as well as brush-based filter shape adjustment.
- To achieve further abstraction in the style of pixel art, we propose multiple colour quantization methods, including palette generation in the OKLab colour space.
- Finally, we provide a fast GPU rasterization-based implementation of non-square pixelation, which computes finished images in seconds and allows the user to watch the pixel art evolve in real time.

In addition to creating stylized digital artworks, our research may be useful in real-world situations involving large, non-square pixels. For example, Project Primrose [3] is a display technology where pixels can be cut to any shape to create large low-resolution displays with limited colour palettes. In the context of Project Primrose, these pixels are called *petals*, and the corresponding content-aware rendering process is called *petalization*.[1] We believe that content authoring workflows and rendering pipelines for such displays stand to benefit significantly from our method.

2 Related Work

The problem of rendering pixel art on an array of non-square pixels is, as far as we know, novel to computer graphics research. However, similar problems appear in various forms, especially in related non-photorealistic rendering contexts. In particular, we are interested

[1] Sometimes, we refer to non-square pixelation as *petalization* for brevity, even outside of Project Primrose.

in any image abstraction method that involves dividing an image into visible discrete colour elements.

2.1 Pixel Art in Computer Graphics

A far as we are aware, Inglis and Kaplan [9] and Gerstner et al. [5] were the first authors to explicitly study algorithms for rendering pixel art. Inglis and Kaplan focus on the problem of rendering vector graphics as pixel art, placing special emphasis on the avoidance of undesirable visual artifacts categorized in their work [9]. Their method would later be extended to simulate hand-drawn antialiasing methods [10]. Meanwhile, Gerstner et al. present a method for abstracting high-resolution raster images into pixel art. They optimize a mapping of contiguous image regions (*superpixels*) to target pixels in the output, while simultaneously evolving a representational colour palette [5].

Kopf et al. [13] approach abstractive image pixelation by optimizing a bilateral combination of gaussian filter kernels for each output pixel via constrained maximum-likelihood optimization. Later, Shimizu et al. [21] introduce a full pixel art rendering pipeline for 3D scenes, including toon shading, depth-texture-based contour extraction, topological contour thinning, and image downsampling, per connected mesh object. Finally, Kaji's work [11] brings the methods of Inglis and Kaplan [2012] to the realm of 3D scene rendering. Analytical curves are fit to the contours extracted from a 3D scene render, and similar techniques are applied to remove visual artifacts from rasterization.

Some pixel art generation methods take advantage of recent advances in machine learning. Wu et al. [24] propose an architecture that decouples the learning of the colour content and pixel resolution, which allows for the pixel size to be controlled while maintaining sharp feature lines and intelligible colours—though exact adherence to the pixel structure of the target resolution isn't fully guaranteed out of the box. Binninger et al. [2] take a different approach to image pixelation, using score distillation sampling to optimize the parameters of a differentiable image generator against a semantic loss function, while enforcing hard constraints on the image resolution and colour palette.

Importantly, the preceding works all restrict their output to a standard square pixel grid. We found some artists, like Nick Loomis [17], who have experimented with non-square pixels, and certain kinds of non-square pixels (e.g. hexagons) have also been proposed as the basis for imaging systems [18]. However, we are not aware of any research that attempts to generalize pixel art rendering to other pixel geometries.

2.2 Non-Photorealistic Rendering Techniques

Ideas from previous work in non-photorealistic rendering (NPR) also prove relevant. Painterly, or *stroke-based* rendering, is a classic non-photorealistic rendering task that divides an image into discrete brushstrokes. Litwinowicz et al. [16] introduced the smoothing and subsequent division of an image into visually distinct brushstrokes, and the using of image gradients to estimate the directional information of image regions. The resulting images gain a handpainted, impressionist appearance in the style of Claude Monet. Hertzmann [7] would later expand this method to brush strokes of multiple sizes, opting to explicitly construct spline curves to represent brush strokes of varying lengths.

Stylized rendering can also be accomplished with various image-space filtering techniques. The Anistropic Kuwahara Filter [14] is well-known for its ability to turn photographs and detailed 3D scene renders into beautiful images that look a lot like oil paintings. Of particular interest to us is how the Anisotropic Kuwahara Filter uses directional image information (in this case, the edge tangent flow) and optionally depth information [1] to warp the filter to adapt to image features, allowing it to effectively stylize thin and complex details such as hair into structures that resemble directed brushstrokes.

Another area of highly relevant NPR research is image synthesis in imitation of other ornamental art forms. Hausner [6] proposes a method for turning raster images into decorative mosaics by orienting fixed tiling geometry to respect the colour boundaries in an underlying image. Song et al. [23] introduce a method that allows for even more extensive abstraction, first segmenting an image along colour boundary lines, then fitting a set of fixed shapes to the segmented regions. This can be used to produce highly abstract artworks that resemble the art of Wassily Kadinsky. Finally, Lawonn and Günther [15] present *Stylized Image Triangulation*, where an image is rendered into a connected triangle mesh with either constant colour or soft gradients per triangle. Gradient descent is used to minimize the root mean squared approximation error of the image triangle approximation. The result is an image that looks like a stylized low-poly render of the contained objects. Ideas from Lawonn and Günther's method will form the foundation for our solution to the novel non-square pixel art rendering problem.

3 Background

Our method for non-square pixel art rendering builds directly on several concepts from Stylized Image Triangulation [15]. We now briefly summarize their method, where they formulate the image triangulation problem as as the gradient descent minimization of an energy function.

Image Partition into Triangles. In an image triangulation, the image domain U is partitioned into a set $\mathfrak{R}$ of connected triangle-shaped regions with no gaps or overlaps:

$$\bigcup_{R \in \mathfrak{R}} R = U \quad \text{where} \quad \forall R_i, R_j \in \mathfrak{R},\ R_i \cap R_j = \emptyset$$

The colour of a given triangle region $R \in \mathfrak{R}$ in an image triangulation is the average of the colours in the image that lie underneath R. Expressed as an integral over the image domain:

$$\mathrm{Col}(R) = \mu_R = \frac{1}{\mathrm{Area}(R)} \int_R U(r)\ \mathrm{d}r$$

where $U(r)$ is the image contents (colour) at location r. In effect, this makes the region R a constant filter kernel with a truncated triangular filter support over the image signal U.

Triangle Approximation Error. The *energy* $E(R)$ of a given triangle (which we also refer to as the *local triangle error*) over the image R is the continuous mean squared error of the triangle colour, or equivalently the variance of the colour in the image region R:

$$E(R) = \frac{1}{\text{Area}(R)} \int_R (U(r) - \mu_R)^2 \, \mathrm{d}r$$

The objective of image triangulation is to minimize the aggregate sum of the individual triangle errors $R \in \mathfrak{R}$. In practice, $\text{Col}(R)$ and $E(R)$ are discretized into the equivalent summations over pixels in the image.

Minimization of $E(R)$. $E(R)$ is minimized via gradient descent over vertex positions. Since we are now in the vertex domain, we express the energy of a vertex $\mathbf{v}$ as the sum of the respective approximation errors of its connected triangles $\rho(\mathbf{v})$, plus a regularization term:

$$E_R(\mathbf{v}) = \sum_{R \in \rho(\mathbf{v})} \frac{E(R)}{3} + \lambda s(\mathbf{v})$$

where s is a uniform Laplacian smoothing that moves a vertex towards the centre of its neighbours, i.e., the centre of its topological 1-ring. (The division by three is due to the fact that triangles have three corners, so each triangle is visited thrice).

Finally, we can express the local energy minimization in gradient descent terms. For a vertex $\mathbf{v}$ and step size h:

$$\mathbf{v} := \mathbf{v} - h\vec{\nabla} E_R(\mathbf{v}) \tag{1}$$

which updates $\mathbf{v}$ for each epoch that gradient descent executes.

4 Method

As discussed in the context of Figure 2, a pixel artist will distort an image in order to improve the alignment of its features with a low-resolution pixel grid. Our technique is built upon the insight that we can harness the work of Lawonn and Günther to compute the required distortion. In Stylized Image Triangulation, they optimize the vertices of a triangulation over the domain of an image, attempting to minimize colour variance within each triangle, and present the evolved triangulation as an approximation of the image. By applying the colours found by their algorithm to the triangles in their *initial positions*, we instead induce a variance-minimizing warp of the image data into the initial triangulation. Our main challenge is to extend their algorithm to minimize variance across arbitrary polygonal tiles. These tiles must avoid straddling feature boundaries so that they do not take on unwanted variance, which would cause blurring in the final result (see Figure 3b).

4.1 Core Algorithm

The core algorithm we propose is grounded in an extension of local triangle error to arbitrary tile shapes. From now on, we discretize integrals over the image domain into sums over pixels.

Tilings and the Tile Warp. For our purposes, a tiling $\mathcal{T}$ is a finite collection of non-overlapping polygons whose union covers a subset of the image domain U. By triangulating every tile, we can also regard a tiling as a triangle mesh similar to $\mathfrak{R}$ above, whose triangles are partitioned into the individual tiles $T \in \mathcal{T}$. Our goal is to construct a *tile warp* tiling $\mathcal{T}'$ that starts out as a clone of $\mathcal{T}$, but whose vertices will move around U, guided by a gradient-descent optimization. The output of the algorithm will be a rendering of the original tiling $\mathcal{T}$ in which each tile is filled with the average colour of the pixels lying underneath the corresponding tile in $\mathcal{T}'$.

Tiles and Tile Filters. We have established that a triangle in an image triangulation is the truncated filter support for a constant filter kernel over the image domain. The same can be said of tiles. The mean colour of a tile $T \in \mathcal{T}$ is given by:

$$\text{Col}(T) = \mu_T = \frac{1}{|T'|} \sum_{t \in T'} U(t) \tag{2}$$

where T' is the underlying warped tile filter associated with T, and $U(t)$ is the image contents at location (pixel) $t \in T'$. As with image triangulation, no tiles overlap. (Note that the naïve case of simply averaging all the pixels underneath a tile is equivalent to the situation where $T = T'$.) We hypothesize that warping these filter supports to adapt to image features will preserve salient features in the source image and improve the overall clarity of the resulting non-square pixelation. For example, given an input image with thick cartoon outlines, a tile might be able to warp to span the width of the outline (Figure 3). The outline will then remain sharp in the final rendering, whereas it would be blurry with naïve averaging.

Tile and Vertex Error. Our notion of tile error (or energy) $E(T)$ is a natural extension of the triangle error. It is equivalent to the variance of the pixels underneath the corresponding tile warp T':

$$E(T) = \mathrm{E}\left[(t_i - \mu_T)^2\right] = \sum_{t_i \in T'} \frac{(t_i - \mu_T)^2}{|T'|} \tag{3}$$

Our goal is to update the tile colour μ_T by finding an optimal vertex configuration for T' that minimizes the variance of the underlying colours, while simultaneously preventing extreme distortions. We transform into the vertex domain in a similar manner:

$$E(\mathbf{v}) = \sum_{T \in \tau(\mathbf{v})} \frac{E(T)}{|\text{Verts}(T)|} \tag{4}$$

Where $\text{Verts}(T)$ is the set of vertices that belong to tile T, and $\tau(\mathbf{v})$ is the set of tiles connected to the vertex $\mathbf{v}$.

Tile Warp Regularization. Unlike in Stylized Image Triangulation, where mesh regularization is used mainly to stop degenerate (overlapping or needle-thin) triangles from forming, we also use mesh regularization of the tile warp $\mathcal{T}'$ to prevent $\mathcal{T}'$ from deviating excessively from $\mathcal{T}$. In other words, regularization limits the amount of distortion permitted between the source and tiling-space images. Because our method should be able to pixelate over myriad tile shapes, some of which could be non-convex, we propose an updated regularization scheme. Rather than uniformly moving each vertex to the centre of its 1-ring, our regularization method will instead attempt to restore the original shapes of the tiles.

Let $D_{\mathbf{v}}$ be the vertices in the 1-ring of $\mathbf{v}$, i.e., the vertices connected to $\mathbf{v}$ in the underlying triangle tessellation of the tiling. We want to find coefficients $c_1 \ldots c_n$ for each neighbouring vertex $\mathbf{d}_1 \ldots \mathbf{d}_n \in D_{\mathbf{v}}$ such that

$$\begin{bmatrix} | & | & & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \dots & \mathbf{d}_n \\ | & | & & | \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix} \tag{5}$$

If we exclude vertices of degree 2 on the frontier of the mesh (which we hold fixed), this equation is an underdetermined linear system of the form $\mathbf{Mx} = \mathbf{v}$. We solve for the coefficients $\mathbf{x}$ using the pseudoinverse $\mathbf{M}^\dagger$, giving $\langle c_1 \dots c_n \rangle = \mathbf{M}^\dagger \mathbf{v}$. We can now express the mesh regularization term $p(\mathbf{v})$ using the elements of $D_\mathbf{v}$:

$$p(\mathbf{v}) = \left(\sum_{\mathbf{d}_i \in D_\mathbf{v}} c_i \mathbf{d}_i \right) - \mathbf{v} \tag{6}$$

These coefficients are computed once upon initialization of the geometric tiling data and used at each gradient descent step to move $\mathbf{v}$ towards its starting position relative to its neighbours.

Construction of Gradients. The components of the energy gradient for a given vertex are the texture-space x and y partial derivatives of the sum of the surrounding tile errors:

$$\vec{\nabla} E(\mathbf{v}) = \left\langle \frac{\partial E(\mathbf{v})}{\partial \mathbf{v}_x}, \frac{\partial E(\mathbf{v})}{\partial \mathbf{v}_y} \right\rangle \tag{7}$$

which we compute via finite differencing. The central gradient descent routine (i.e., the gradient descent of the tile warp errors) can thus be expressed similarly to Equation 1, with $E(\mathbf{v})$ in place of $E_R(\mathbf{v})$, and reflecting our new regularization scheme:

$$\mathbf{v} := \mathbf{v} - h\vec{\nabla} E(\mathbf{v}) + \lambda p(\mathbf{v}) \tag{8}$$

where h is typically the approximate width of one texel in the input image, and λ is a user-defined regularization coefficient. We also recommend enforcing a trust region with a radius between 0.1 and 0.4 texels (depending on the resolution of the input image), which limits $||h\vec{\nabla} E(\mathbf{v})||_2$ per epoch to the extent of the trust region [15]. We suggest initial values between 0.01 and 0.02 for λ; this value can also be hand-tuned by the artist for any given combination of image and tiling.

4.2 Colour Quantization

Inspired by real-time stylized rendering practices, as well as previous work on pixel art rendering, we propose multiple colour quantization schemes for stylizing the resulting non-square pixelation:

- **Value mapping by lightness** of the final non-square pixelation into a colour palette constructed by our program (or manually specified by the artist). We provide a colour palette generator which produces a palette of the specified size by varying hue and lightness in OKLab, a perceptually uniform colour space [20].
- **Nearest-neighbour colour quantization** following 1) a separate palette extraction pass, such as via k-means clustering or mean-shift segmentation, 2) our palette generation routine, or 3) manual colour specification by the artist.

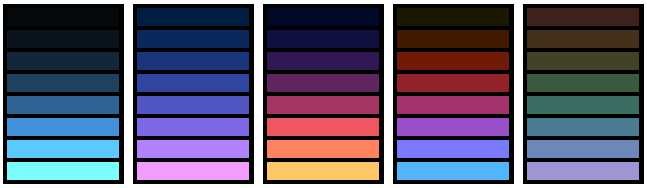


Figure 4: Examples of colour palettes generated by our program. The user can specify a starting hue and palette type, which control the extent to which we move in the hue dimension. From left to right are a monochrome, analogous, complementary, triadic, and tetradic palette. (We allow complementary palettes to either be evenly interpolated [8] or centred around opposing sides of the colour wheel.)

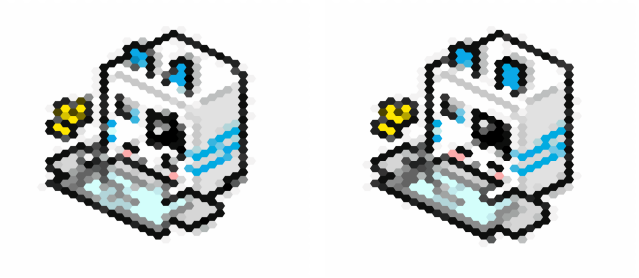
Colour Theory Basis. Our palette generation method is loosely inspired by the system presented by Hu et al. [8]. To emulate the choices a trained artist might make, we implement Ostwald's suggestions for picking harmonious colour palettes: one dimension must stay constant, and differences in the other dimensions must be evenly spaced [19]. We always[2] vary the lightness to preserve shadows and highlights in the source image. Hence, except for the case of monochromatic palettes (where hue stays constant instead), we fix the saturation.

Palette Generation. While Hu et al. use the CIELAB colour space for palette construction [8], we adopt a growing trend in stylized rendering and use OKLab for this task instead. This is due to OKLab's improved perceptual brightness uniformity across different hues with the same lightness [20]. Because we generate palettes by varying hue, chroma, and/or lightness, we perform the construction in OKLCh before transforming into OKLab and then back into linear sRGB. To construct a palette of size n:

- We start with a given hue h_0, chroma χ_0, and lightness L_0. L_0 should be low.
- The step size for brightness (step_L) should be picked such that $0 < \text{step}_L \leq \frac{(1-L_0)}{n}$. (A similar idea applies for the chroma step in the case of a monochrome palette.)
- In OKLCh, hue is specified in radians. Hence, the hue step size should be picked such that $\text{step}_h \approx \pm\frac{2\pi}{n} \cdot s_{\text{scale}}$, where $s_{\text{scale}} \in [0, 1]$ is smaller when we want the hues to be more similar.
- For complementary palettes, one may optionally skip to the other side of the colour wheel (add π to the hue) at the halfway point of the palette walk, and from there start incrementing hue in the opposite direction.

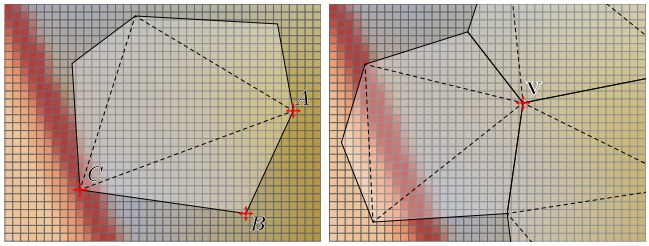
In the interest of being artistically principled (in the Ostwald sense), step_L, step_χ, and step_h should be generated once at the start and remain the same when performing the palette walk. To inject extra variety into the palettes, the step sizes can deviate slightly from the values we suggest when they are initially picked, as long as they remain the same during the walk itself. We provide suggested s

[2]For some inputs, such as rasterized vector art (logos, simple character mascots, etc.), it might be better to vary saturation rather than lightness, or default to nearest-neighbour colour quantization. As usual, human artistic judgement must be exercised.



(a) Before Brushing (b) After Brushing

Figure 5: Our algorithm does not make any hard guarantees about preserving symmetry; for example, the mascot's eyes and ears in (a). We provide a brushing tool that works over the tile warp visualization and allows the artist to move the vertices, thereby adjust the mapping to their liking; results shown in (b). Both images are quantized to a 16-colour k-means palette.



(a) Triangle Perturbations (b) Vertex Energy Gradients

Figure 6: Visualization of tile-level energy minimization computations, which we express as a sum over the corresponding triangle tesselation. (a) First, the geometry shader moves each vertex of each triangle one texel in all four cardinal directions; every copy of that triangle is then rasterized. (b) At the compute shader stage, we finite difference the vertex energy gradients by summing the triangle energies within all the tiles connected to a vertex, ensuring that the movement of a given vertex is correctly reflected in the errors we look up for other triangles in a given tile.

values for each palette type in Table 1; see Figure 4 for example palettes.

4.3 Brush-Based Vertex Adjustment

We implement a simple brushing tool that allows for manual adjustment of groups of vertices in the tile warp $\mathcal{T}'$. We display a brush circle in screen space; when holding down a mouse button, vertices in the tile warp that lie within the brushing radius get either pulled towards or repelled from the centre at a speed controlled by a user-defined variable. This can be used to drag vertices around in the hopes of finding a more preferable local minimum, or to bring in more tiles to capture details the artist considers important, potentially at the cost of increased distortion. See Figure 5 for examples.

5 Implementation

Our implementation of non-square pixelation is an interactive DirectX11 application built directly on top of a subset of Lawonn and Günther's *Stylized Image Triangulation* prototype [15]. Specifically, we retain the core routines that correspond to the equations described in section 3.

5.1 Generation of Geometric Tiling Data

To generate many of the tilings used here, we developed a separate interactive tool based on the *Tactile* tiling library [12]. This tool makes it easy to create (finite patches of) isohedral tilings, a broad class of tilings by single shapes that encompass the kinds of expressive drawings created by the artist M.C. Escher [4]. We can edit the shapes of the tiles, globally rotate and scale the tiling, and draw a bounding rectangle to choose a set of tiles for export.

The tool uses ear clipping to triangulate the tiles, and exports the mesh as a JSON file to be read by the optimization software. The file contains the usual vertex and index buffers making up a triangle mesh, as well as a buffer of triangle spans, where each span records the indices of the triangles making up one tile. Because no triangle belongs to multiple tiles, we store a tile's triangles contiguously in the index buffer, and represent each span using its starting and ending triangle IDs.

5.2 GPU Computation of Tile Energy Gradients

In this subsection, we discuss the parallel computation of the tile energy gradients (Equation 8) through a combined GPU rasterization and compute shader scheme.

Approximation of Tile Error Partial Derivatives. To approximate the partial derivatives described in Equation 7 using finite differences, we need to know how movement of each vertex by one texel in each cardinal texture-space direction $\{\hat{u}, -\hat{u}, \hat{v}, -\hat{v}\}$ affects the energy of a given tile. In Stylized Image Triangulation, each triangle is copied and perturbed 12 times by a geometry shader (there are 3 vertices per triangle, and each vertex can move in 4 directions), and then each copy is rasterized twice:

(1) Once to compute the average texel colour underneath that triangle,
(2) Once more to compute the squared error by comparison of texture data against the mean.

Both of these computations are carried out with atomic addition of texel values in the fragment shader [15]. In Image Triangulation, a compute shader then runs for each vertex:

(3) Sum the squared errors of adjacent triangles and compute gradients via finite differences.

Each vertex can subsequently be moved along its gradient.

In non-square pixelation, because we do not know at compile time how many vertices a given tile may have, we found it architecturally convenient to inject a compute shader pass in between stages (1) and (2) that overwrites the triangle colours with the average tile colour before computing the squared errors. We hence leave stage (1) largely untouched. Then, in the compute shader that

Table 1: Palette Generation Coefficients

Palette Type	s_{scale} value
Monochrome	0
Analogous	0.25
Complementary	0.33
Complementary Gradient	0.50
Triadic Gradient	0.66
Tetradic Gradient	0.75

updates vertex positions (3), we compute the error gradients for the connected tiles by summing the errors in each tile's triangles. We use a series of precomputed lookup tables to associate adjacent vertices, triangles, and tiles with each other. Proof that this is equivalent to the overall tile error described in Equation 3 is provided in Appendix A.

At both the mean computation and finite differencing stages, it is important to **correctly account for how the movement of one vertex affects other triangles in a given tile** (Figure 6). We store computed means and errors for each perturbation of a given triangle contiguously in GPU memory. In the compute shader passes, we implement lookup routines that allow us to retrieve, for all the triangles within a tile, the correct triangle colours or triangle error entries given a vertex ID and which perturbation we are currently considering (+ or -, and in direction u or v).

6 Results

We have explored the world of non-square pixel art by experimenting with many combinations of source images and tilings. After loading an image and tiling into our software, a finished result typically emerges in a few seconds. If using the brush tool to coax tiling vertices into more favourable positions (subsection 4.3), interaction time might stretch to a minute as the user makes adjustments. They must switch back and forth between the tile warp $\mathcal{T}'$ and the undistorted tiling $\mathcal{T}$, applying the brush to $\mathcal{T}'$ and previewing the resulting pixel art image. The program itself runs at around 91 frames per second on a commercial laptop equipped with an Nvidia RTX 3060 GPU given a 1280×720 image and a tiling containing 6720 triangles; however, very high-resolution input images (2K or higher) can result in significantly increased frame times (down to ~30 frames per second) due to the large number of atomic adds.

Comparison With Naïve Methods. We have demonstrated numerous qualitative improvements over naïve methods throughout this paper. To provide a basic quantitative measurement of petalized image fidelity, we compute the overall integrated tile error by taking the square root of the average of the tile errors across the entire tiling-space image. Predictably, this value goes down during petalization (Figure 8). However, we notice interesting behaviour when instead defaulting to more traditional measures of perceptual similarity. When running the LPIPS [25] perceptual similarity model, comparing our results against the original image, and then doing the same for a naive render against the original image, our results have lower predicted perceptual similarity even though the images we produce are much clearer. We find this surprising, but sensible.

(a) Input **(b) Kopf et al.** **(c) Ours**

Figure 7: Though square pixel art is not our strict focus, we demonstrate square pixelation of reasonable quality. Our result shows similar clarity on the outlines, though Kopf et al.'s method [13] is better at preserving symmetry when the optimization is constrained. Though unused for this figure, the vertex brush could help make (c) more symmetrical. (*Figure (b) taken directly from the Kopf et al. manuscript.*)

When working with abstractive art forms such as pixel art, it makes sense that traditional notions of "image correctness" for continuous image signals become less applicable.

Results with Various Tilings. We provide an assortment of tilings which are rendered over by the same image in Figure 9. Not only is our method able to adapt to thin image features Figure 9d, it is also able to find analogous representations of image features that are difficult to depict in the given tiling (Figure 9e, Figure 9f).

Square Pixel Art. We demonstrate the ability of our method to also use square tiles to pixelate images, effectively rendering content-aware pixel art in the traditional sense (Figure 13). We provide a brief comparison with previous work in Figure 7.

Aperiodic Tilings. Non-square pixelations using the hat tiling [22] are shown in Figure 11. We find that the results for hats are often slightly better if one point samples after optimizing.

Results of Varying Colour Palettes. We show results with various colour palettes in Figure 9, Figure 11, and Figure 13. Finding a pleasing colour palette to value map a given tiling-space image is a fun and challenging artistic exercise in itself.

Toon Renders and Drawings. We find non-square pixelation to be particularly effective on toon-rendered 3D scenes and cartoon drawings. These images strike a very good balance between having large regions of constant colour, while retaining geometric detail as well as salient lighting and depth cues (Figure 10, Figure 12). Often, these images come with contour lines that our algorithm can effectively extract.

7 Limitations

Because our method allows tiles in the tile warp to expand and contract according only to the error between the underlying image data and the tile average, there are inevitably some higher-level image semantics that it misses. In particular, we do not provide any hard guarantees about preservation of symmetries in the source image; we address this deficiency by allowing the artist to manually move the filter supports around via brushing (see Figure 5a). The

		
Loss Metric	**naïve averaging**	**optimized**
LPIPS	**0.2727**	**0.3358**
Integrated Tile Error	**336.107**	**207.698**

Figure 8: Rendering of Barty onto the Cairo tiling. We compare the visual loss predicted by LPIPS [25], a state-of-the-art perceptual similarity model, with our notion of the overall integrated tile error. Despite arguably looking much worse, as well as registering higher error by our measurement, LPIPS predicts lower perceptual loss for the naïve render.

continuity of thin features in the source image is also not guaranteed. While decreasing the regularization weight allows individual tiles to deform more to wrap around thin features, they cannot necessarily do so while remaining connected in a single unbroken chain of tiles. The optimization is ultimately constrained by the number of vertices in the input tiling: triangle edges must remain straight, and so the best a triangle edge can hope for is to align with the straight boundary of a feature in the image. A more complete solution might allow edges to deform away from straightness, allowing long edges to align to gently curved feature boundaries. One possible compromise is to supersample the edges of the tiling. Each edge would then gain additional degrees of freedom, at the cost of slowing down the optimization by increasing the number of triangles used.

8 Conclusion and Future Work

We have presented a technique that produces stylized, content-aware pixel art from input images. This technique is the first to generalize seamlessly from a grid of square pixels to an arbitrary division of the plane into non-overlapping tiles. We implement the technique using a fast GPU-based gradient descent optimization.

The goal of the optimization is for each tile in a tiling to minimize the colour variance of the contiguous part of the input image to which it is associated. Such an approach is certainly more content-aware than filling tiles with colours computed via naïve averaging or point sampling: specifically, tiles avoid crossing edges, and generally represent features in the source image effectively.

Still, we believe that our results could be improved by using higher level semantic information in the image. Using computer vision techniques, we might perform a figure-ground analysis by identifying the foreground subjects of an image. We could then loosen the regularization in the ground, allowing tiles to align better with subjects, in order to represent them more faithfully in the output. Ultimately, the boundary of a subject must be quantized to the edges of the tiling. In these cases, knowing the shape of that boundary might allow us to make better local decisions about tile colours. Experimentally, we see an asymmetry in boundaries. When a tile straddles the boundary of a foreground shape, we have observed that it is often better to assign it the background colour, allowing it indent the shape, rather than allowing the shape to protrude over the background. This might also help prevent excessively thick outlines from forming (e.g., in a case like Figure 7c).

We are also interested in exploring the construction of colour palettes that are more closely related to the colour information in the source image, but that still attempts to reflect the deliberate aesthetic choices a pixel artist would make. It would be interesting to explore techniques that extract an initial palette from the image and then adjust its colours to approximate the OKLab ramps described in subsection 4.2. With any fixed colour palette, it might also become possible to quantize the source image and then assign each to each tile the dominant colour of its pixels rather than the mean, thereby further reducing the blurring caused by averaging. Finally, in keeping with the pixel art aesthetic, we could also generalize standard dithering algorithms to arbitrary tilings—for example, by creating a catalogue of ordered dithering patterns for all 93 isohedral tiling types.

References

[1] Jeong-ho Ahn, Jong-Chul Yoon, and In-Kwon Lee. 2010. Depth-based Anisotropic Kuwahara Filtering. In *ACM SIGGRAPH 2010 Posters* (Los Angeles, California) *(SIGGRAPH '10)*. Association for Computing Machinery, New York, NY, USA, Article 123, 1 pages. doi:10.1145/1836845.1836977

[2] Alexandre Binninger and Olga Sorkine-Hornung. 2024. SD-XL: Generating Low-Resolution Quantized Imagery via Score Distillation. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. ACM, 1–12. doi:10.1145/3680528.3687570

[3] Christine Dierk, TJ Rhodes, and Gavin Miller. 2022. Project Primrose: Reflective Light-Diffuser Modules for Non-Emissive Flexible Display Systems. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 22, 14 pages. doi:10.1145/3526113.3545625

[4] Maurits C Escher and Doris Schattschneider. 2004. *Visions of symmetry*. Thames & Hudson.

[5] Timothy Gerstner, Douglas DeCarlo, Marc Alexa, Adam Finkelstein, Yotam I. Gingold, and Andrew Nealen. 2012. Pixelated image abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering*. https://api.semanticscholar.org/CorpusID:7158059

[6] Alejo Hausner. 2001. Simulating decorative mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 573–580. doi:10.1145/383259.383327

[7] Aaron Hertzmann. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 453–460. doi:10.1145/280814.280951

[8] Guosheng Hu, Zhigeng Pan, Mingmin Zhang, De Chen, Wenzhen Yang, and Jian Chen. 2014. An Interactive Method for Generating Harmonious Color Schemes. *Color Research Application* 39 (02 2014). doi:10.1002/col.21762

[9] Tiffany C. Inglis and Craig S. Kaplan. 2012. Pixelating vector line art. In *ACM SIGGRAPH 2012 Posters* (Los Angeles, California) *(SIGGRAPH '12)*. Association for Computing Machinery, New York, NY, USA, Article 108, 1 pages. doi:10.1145/2342896.2343021

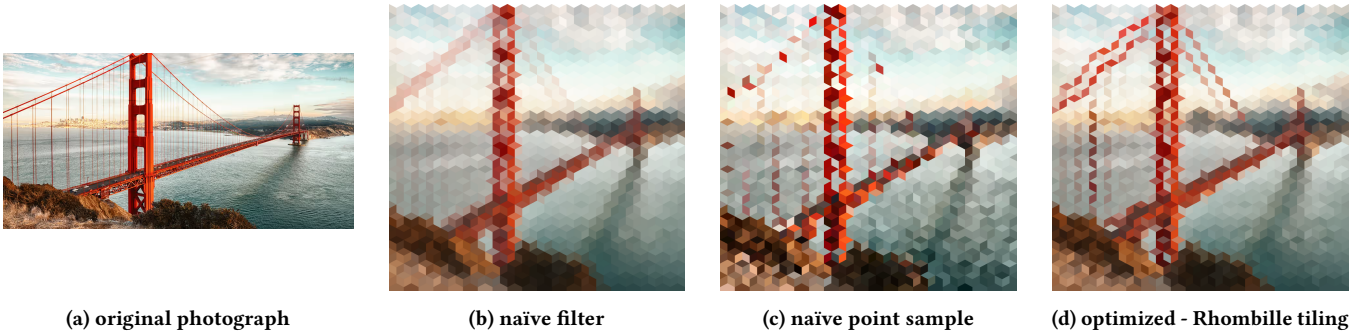
[10] Tiffany C. Inglis, Daniel Vogel, and Craig S. Kaplan. 2013. Rasterizing and antialiasing vector line art in the pixel art style. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Anaheim, California) *(NPAR '13)*. Association for Computing Machinery, New York, NY, USA, 25–32. doi:10.1145/2486042.2486044

[11] Yuki Kaji. 2021. Generating Contours in the Pixel Art Style from 3D Meshes. *Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo* (2021). https://vcl.jp/publications/2015/2015-01-n-shimizu.html

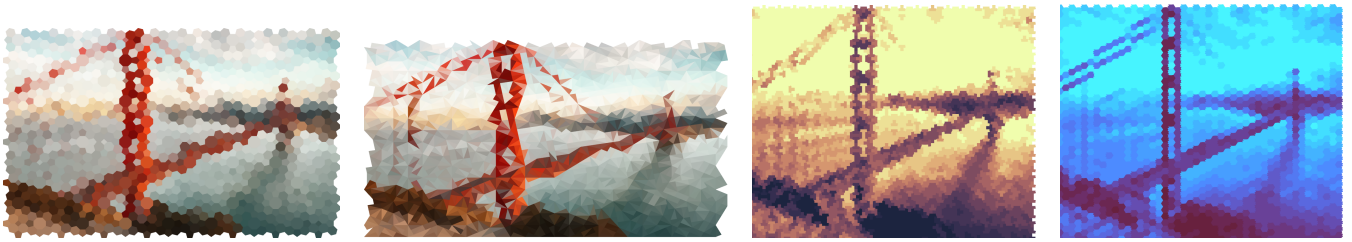
[12] Craig S. Kaplan. 2022. Tactile. https://github.com/isohedral/tactile Accessed on 09 October 2025.

[13] Johannes Kopf, Ariel Shamir, and Pieter Peers. 2013. Content-adaptive image downscaling. *ACM Trans. Graph.* 32, 6, Article 173 (Nov. 2013), 8 pages. doi:10.1145/2508363.2508370

[14] Jan Eric Kyprianidis, Henry Kang, and Jürgen Döllner. 2009. Image and Video Abstraction by Anisotropic Kuwahara Filtering. *Computer Graphics Forum* 28, 7 (2009), 1955–1963. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01574.x doi:10.1111/j.1467-8659.2009.01574.x

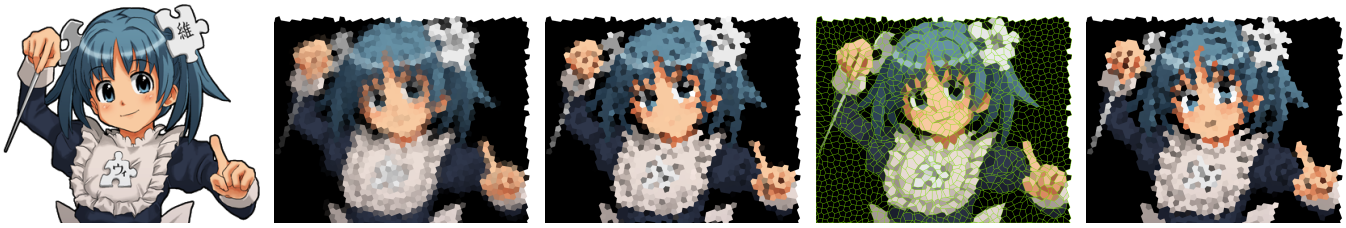


(a) original photograph (b) naïve filter (c) naïve point sample (d) optimized - Rhombille tiling



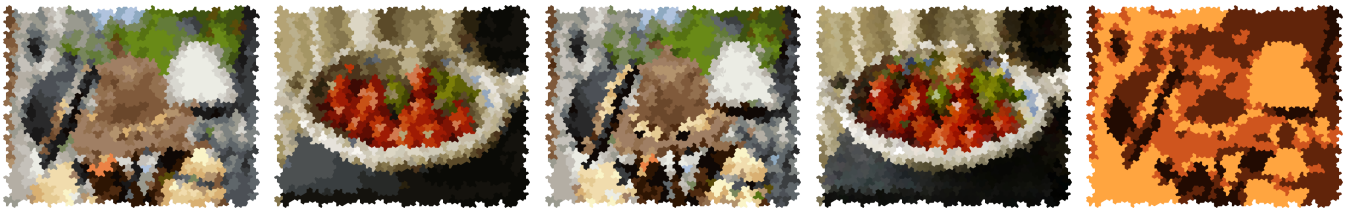
(e) optimized - Altair tiling (f) optimized - Pinwheel tiling (g) optimized - T tiling (h) optimized - fish tiling

Figure 9: Various renders of the Golden Gate Bridge using naïve methods (b, c) and our method (d–h). (d) shows a optimized render on the Rhombille tiling; note how extremely thin features are effectively recovered. (e) and (f) show interesting results on tilings of multiple shapes (the Altair and Conway-Radin Pinwheel tilings respectively). Our program finds a reasonable approximation of image structures even when exact representations are impossible (e.g., straight vertical lines). (g) and (h) show renders with different 16-colour palettes, first on a T-shaped tiling, and then on a fish-shaped tiling.



(a) original drawing (b) naïve filtering (c) naïve point sampling (d) optimized tile shapes $\mathcal{T}'$ (e) point sample from (d)

Figure 10: Rendering of Wikipedia's anime mascot, Wikipe-tan, into the Cairo tiling. Here, we show how our algorithm can also greatly improve the results obtained by point sampling. (e) Notice how the tiles will contort to idiomatically represent the gaps in the fingers holding the baton, as well as the schlera and irises of the eyes. Meanwhile, (b) and (c) exhibit the usual detail loss symptoms. Because the optimization routine allows tiles to wrap around thin colour regions, even ultra-thin details such as the mouth lines and hair shading (e) can be recovered by point sampling from (d) $\mathcal{T}'$. We choose not to quantize here as point sampling preserves the already-limited colour palette.



(a) hats on hats (b) bowl of tomatoes (c) hats on hats (point) (d) bowl of tomatoes (point) (e) petalized hats (4-colour)

Figure 11: Various non-square pixelations on the aperiodic *hat* [22] tiling. (a, b) optimized, using full averaging and 32 k-means colours. (c, d) point sampling after optimization, also 32 k-means colours. (e) quantized to 4 analogous colours, akin to the constraints imposed by a Primrose display [3].

(a) engine screenshot (b) naïve filtering (c) optimized

Figure 12: Non-square pixelation is particularly effective on toon renders. Notice how the eyelashes, ribbon, and patterns on the clothes and floor are not only recovered, but translated into their closest analogous representation in the new tiling space. (*Input 3D scene © Unity Technologies Japan/UCL*. Image (a) rendered in Unity Game Engine.)

(a) no optimization (b) optimized - 16 colours

Figure 13: Our method can also be used to render content-aware pixel art with square pixels; it performs best at this task when we choose tile sizes that are close to (but not necessarily exactly) the width of important image features, like hair lines and eye schlera. In (b), we particularly enjoy how the creases and shadows on the coat are emphasized by the combined action of the complementary palette and the warped tile filters.

[15] Kai Lawonn and Tobias Günther. 2019. Stylized Image Triangulation. *Computer Graphics Forum* 38 (2019), 221–234. Issue 1. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13526 doi:10.1111/cgf.13526

[16] Peter Litwinowicz. 1997. Processing images and video for an impressionist effect. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 407–414. doi:10.1145/258734.258893

[17] Nick Loomis. 2013. Non-square Pixelations. https://loomsci.wordpress.com/2013/06/27/non-square-pixelations/. Accessed: 2026-02-28.

[18] Paola Magillo. 2025. Non-square grids: A new trend in imaging and modeling? *Computer Science Review* 56 (2025), 100695. doi:10.1016/j.cosrev.2024.100695

[19] W. Ostwald and F. Birren. 1969. *The Color Primer: A Basic Treatise on the Color System of Wilhelm Ostwald*. Van Nostrand Reinhold Company. https://books.google.com/books?id=une8QOa_vYQC

[20] Björn Ottosson. 2020. OkLab. https://bottosson.github.io/posts/oklab/. Accessed: 2026-02-01.

[21] Nobutoshi Shimizu, Yuki Morimoto, Tomoaki Moriya, and Tokiichiro Takahashi. 2015. Automatic Pixel Art Generation from 3DCG Models. *IWAIT* (2015). https://vcl.jp/publications/2015/2015-01-n-shimizu.html

[22] David Smith, Joseph Samuel Myers, Craig S. Kaplan, and Chaim Goodman-Strauss. 2024. An aperiodic monotile. *Combinatorial Theory* 4, 1 (July 2024). doi:10.5070/c64163843

[23] Yi-Zhe Song, Paul L. Rosin, Peter M. Hall, and John Collomosse. 2008. Arty Shapes . In *Computational Aesthetics in Graphics, Visualization, and Imaging*, Douglas W. Cunningham, Victoria Interrante, Paul Brown, and Jon McCormack (Eds.). The Eurographics Association. doi:10.2312/COMPAESTH/COMPAESTH08/065-072

[24] Zongwei Wu, Liangyu Chai, Nanxuan Zhao, Bailin Deng, Yongtuo Liu, Qiang Wen, Junle Wang, and Shengfeng He. 2022. Make Your Own Sprites: Aliasing-Aware and Cell-Controllable Pixelization. *ACM Trans. Graph.* 41, 6, Article 193 (Nov. 2022), 16 pages. doi:10.1145/3550454.3555482

[25] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

A Appendix

We present a proof that the local tile energy can be computed as a weighed sum of the triangle errors of the triangles that belong to that tile. As previously established, the error (energy) of the pixel samples underneath a tile filter T' is equivalent to the variance of the pixels it contains (See Equation 3, reproduced below). In this proof we use T in place of T', as the derivation holds true whether $T = T'$ or not. By the definition of variance:

$$\mathrm{Var}(T) = \mathrm{E}\left[(t_i - \mu_T)^2\right] = \sum_{t_i \in T} \frac{(t_i - \mu_T)^2}{|T|}$$

where t_i is the ith pixel within the filter support of T, $|T|$ is the number of pixels (i.e., the area) of the filter support, and μ_T is the mean colour of T. Expressed as the union of the connected non-intersecting regions $R_{1\ldots n}$ (i.e., triangles) used to tessellate T:

$$\mathrm{Var}(T) = \mathrm{Var}(R_1 \cup R_2 \ldots \cup R_n) = \sum_{r_i \in R_1 \cup \ldots R_n} \frac{(r_i - \mu_T)^2}{|R_1 \cup R_2 \ldots \cup R_n|}$$

By the linearity of the expectation:

$$\mathrm{Var}(T) = \sum_{r_i \in R_1} \frac{(r_i - \mu_T)^2}{|R_1 \cup \ldots R_n|} + \cdots + \sum_{r_i \in R_n} \frac{(r_i - \mu_T)^2}{|R_1 \cup \ldots R_n|}$$

$$= \frac{|R_1|}{|T|}\left(\sum_{r_i \in R_1} \frac{(r_i - \mu_T)^2}{|R_1|}\right) + \cdots + \frac{|R_n|}{|T|}\left(\sum_{r_i \in R_n} \frac{(r_i - \mu_T)^2}{|R_n|}\right)$$

Where $|T| = |R_1 \cup \ldots R_n|$. Since the triangle area is given by $|R_i|$ and the tile area is given by $|T|$, we can effectively compute the tile variance by computing an area-weighed sum of triangle variances with the triangle mean colour μ_R overwritten with the tile mean colour μ_T.

Received 3 April 2026; revised 19 May 2026; accepted