Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Feature-Aware Pixel Art Animation

Ming-Hsun Kuo[1] Yong-Liang Yang[2] Hung-Kuo Chu[1]

[1]National Tsing Hua University, Taiwan [2]University of Bath, UK

Abstract

Pixel art is a modern digital art in which high resolution images are abstracted into low resolution pixelated outputs using concise outlines and reduced color palettes. Creating pixel art is a labor intensive and skill-demanding process due to the challenge of using limited pixels to represent complicated shapes. Not surprisingly, generating pixel art animation is even harder given the additional constraints imposed in the temporal domain. Although many powerful editors have been designed to facilitate the creation of still pixel art images, the extension to pixel art animation remains an unexplored direction. Existing systems typically request users to craft individual pixels frame by frame, which is a tedious and error-prone process. In this work, we present a novel animation framework tailored to pixel art images. Our system bases on conventional key-frame animation framework and state-of-the-art image warping techniques to generate an initial animation sequence. The system then jointly optimizes the prominent feature lines of individual frames respecting three metrics that capture the quality of the animation sequence in both spatial and temporal domains. We demonstrate our system by generating visually pleasing animations on a variety of pixel art images, which would otherwise be difficult by applying state-of-the-art techniques due to severe artifacts.

1. Introduction

Pixel art is a form of digital art where the details in a high resolution image are abstracted using a small number of pixels. This art form was originally invented in the 1980s to adapt to the limited graphics capabilities of computers and video games. At that time, artists showed delicate skills to organize and color pixels, one at a time, such that the pixel art image can perceptually depict the original one. Such an intriguing feature has rendered pixel art a popular contemporary art form that is widely used from computer games to advertisements [VSS12], even though the hardware constraints no longer exist in the modern graphics systems.

Since pixel art is characterized with limited resolution and reduced palette, a subtle change of a single pixel (e.g., position and color) can drastically affect the perceived image. This makes the creation of pixel art a labor intensive and skill-demanding process. Not surprisingly, the production of pixel art animation is much more complicated due to additional coherence constraints imposed in the temporal domain across frames. While there are popular editors, such as Spriter [PM12] and Piskel [Des], that support intuitive key-frame animation with rigid motions (i.e., translation and rotation), the results generated therein often suffer from severe artifacts such as broken outlines and temporal jitters due to the naive nearest neighbor sampling of individual frames. Moreover, users are usually required to manually tweak pixels within deficient frames, which is thus a tedious and error-prone process.

Motivated by the retro charm of pixel art, decades of research have investigated automatic algorithms to optimize the aesthetics of pixel art images during the creation process, such as the pixelated feature lines and color regions in-between [Yu13]. Although state-of-the-art methods have shown impressive results by either rasterizing vector graphics into high quality pixel line arts or computing optimal color palettes out of raster images, they were designed for still pixel art images. There is a lack of an effective and efficient system to produce appealing pixel art animation with multiple frames.

In this work, we present a novel animation framework tailored to pixel art images. The ultimate goal is to provide the user with intuitive control as well as efficient generation of smooth animation sequences that preserve the features of input pixel art images. To this end, our system adopts the interface of traditional key-frame based animation system and allows the user to intuitively manipulate the shape represented in the input image using a few control anchors. Once the user has specified key-frames, the system then

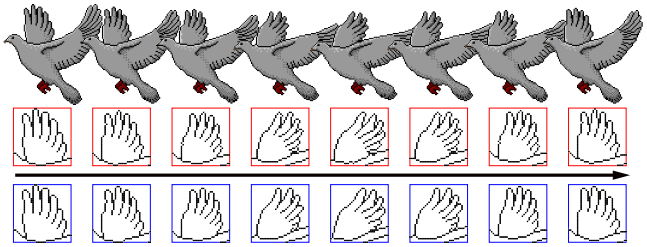


Figure 1: *An example pixel art animation sequence created by our system. Two highlighted sequences show the quality improvements of prominent feature lines in terms of shape and temporal coherence. The red boxes indicate the initial sequence, while the blue boxes are the optimized results (Input image: "Dove" © WhyMe).*

automatically interpolates the intermediate frames and optimizes the full animation sequence respecting the pixel art quality in both spatial and temporal domains. Figure 1 shows a typical example created using our system.

Our system differs from the existing ones in two aspects. First, rather than working directly on the low resolution images, our system converts the input pixel art image to a representation that enables shape-preserving image deformation. Specifically, we focus on the shape of the prominent feature lines, which are extracted from the input image and vectorized into polylines. The input image along with the vectorized feature lines are then embedded into a triangular mesh. With such a hybrid representation, our system achieves intuitive key-frame specification and smooth intermediate-frame interpolation using conventional warping-based shape manipulation. Second, the rasterized feature lines of individual frames are jointly optimized respecting the quality measurement of individual frames and the spatial coherence between frames, resulting in a smooth animation sequence that is visually pleasing. We evaluated our system by using a variety of pixel art images to generate animations of different complexity. Experimental results show that our system can produce plausible animation sequences without noticeable artifacts as those introduced by state-of-the-arts.

To summarize, we highlight here the main contributions of our work. (i) A novel animation framework for producing high-quality animation sequences for pixel art images. To the best of our knowledge, our work is the first attempt in the field. (ii) A set of quality metrics for capturing the aesthetics of feature lines within individual frames as well as the temporal coherence between consecutive frames. (iii) A joint optimization that bases on the quality metrics to refine the shape of individual feature lines and the configuration between feature lines.

2. Related Works

Pixelated shape abstraction. How to abstract geometric shapes using pixels has been studied extensively in computer graphics for decades. Early efforts focused on rasterizing simple 2D parameteric curves, such as lines [Bre65], circles [Bre77], etc. Antialiasing techniques were introduced to make the rasterized shapes visually smoothing [Wu91]. Shape abstraction dedicated to pixel art style has also received increasing attention. For example, Gerstner *et al*. [GDA*12] converted an input image to a pixelated output image by jointly optimizing a mapping of features and a reduced color palette. A general image downscaling filter by Kopf *et al*. [KSP13] incorporated a content-adaptive downsampling kernel to yield sharper and more detailed pixelated abstraction. Inglis and Kaplan [IK12] proposed a novel algorithm to pixelate vector line art where artifacts, such as jaggies and broken lines, are deliberately handled to fulfill the style of pixel art. This method is further improved in [IVK13] by preserving symmetry and adopting manual antialiasing. Recent work also investigated how to fabricate pixelated shape using physical building blocks [KLC*15]. Our work is essentially different from previous ones, and focuses on how to produce high-quality animation sequence from an input pixel art image, rather than generating pixelated abstraction from vector lines or raster images.

Image vectorization. Extracting resolution-independent vector representations from images plays a key role in various applications, including image compression, transmission, deformation, etc. Our work also relies on the vector representation to model the shape and structure of feature lines during the optimization. Various free/commercial image vectorization tools are available online, including Potrace [Sel03], Adobe Live Trace [ADO10], and Vector Magic [Vec10]. It is also worth mentioning that Kopf and Lischinski [KL11] presented a vectorization algorithm that specifically targets pixel art images. While the previous approaches/tools have shown impressive results, we find that a simple polyline-based vectorization followed by feature-aware optimization already achieves satisfactory results in our experiments.

Stylized video and animation. Our work is closely related to the research topic of non-photorealistic rendering of 2D videos or 3D animations. Although several previous works were proposed in various domains such as painting [BNTS07,OH12], hatching [NSC*11], line drawing [BCGF10], they all share a common objective of enforcing temporal coherence according to the underlying stylization. Interested readers may refer to [BBT11] for a comprehensive survey. Our work also models the temporal coherence but aims at a novel stylization, pixel art. To the best of our knowledge, this has never been investigated before.

3. System Overview

Given an input pixel art image, our system provides intuitive tools for generating key frames and interpolating the intermediate frames. More importantly, all the frames are consecutively optimized respecting the quality of the prominent feature lines of individual frames and the spatial coherence between frames, making the final animation sequence visually pleasing. Figure 2 shows an overview of our system, which mainly comprises three major components as detailed below.

Preprocessing. In this step, we extract the prominent feature points (pixels) of the input pixel art image and reconstruct a set of feature lines for the subsequent stages (see Figure 2(a)). As pixel art images are usually generated creatively and may appear in various styles, we focus on a particular yet popular style where the pixelated feature lines are composed of nearly black pixels [Yu13]. Specifically, we set a threshold of 20 for R/G/B channel. We further filter out the outliers with only a single pixel. Then we group the feature points into a set of feature lines as follows. First, we identify all the terminal pixels, each of which has one (ending) or more than two (junction) neighboring pixels in its local 8-connected neighborhood. Then individual feature lines are formed by pixels along a path connecting two terminal pixels. We further merge adjacent feature lines that are continuously connected at the shared terminal pixel, if the difference between tangent directions is less than 45°. In rare cases where the above assumptions (i.e., color, thickness, and connectivity of pixel lines) are violated, the user is expected to infer the shape of feature lines and resolve the ambiguous connectivity between feature lines. Note that such preprocessing only needs to be executed once for each input image.

Keyframe-based animation. To generate the initial animation sequence based on the extracted feature lines, we follow the tradi-

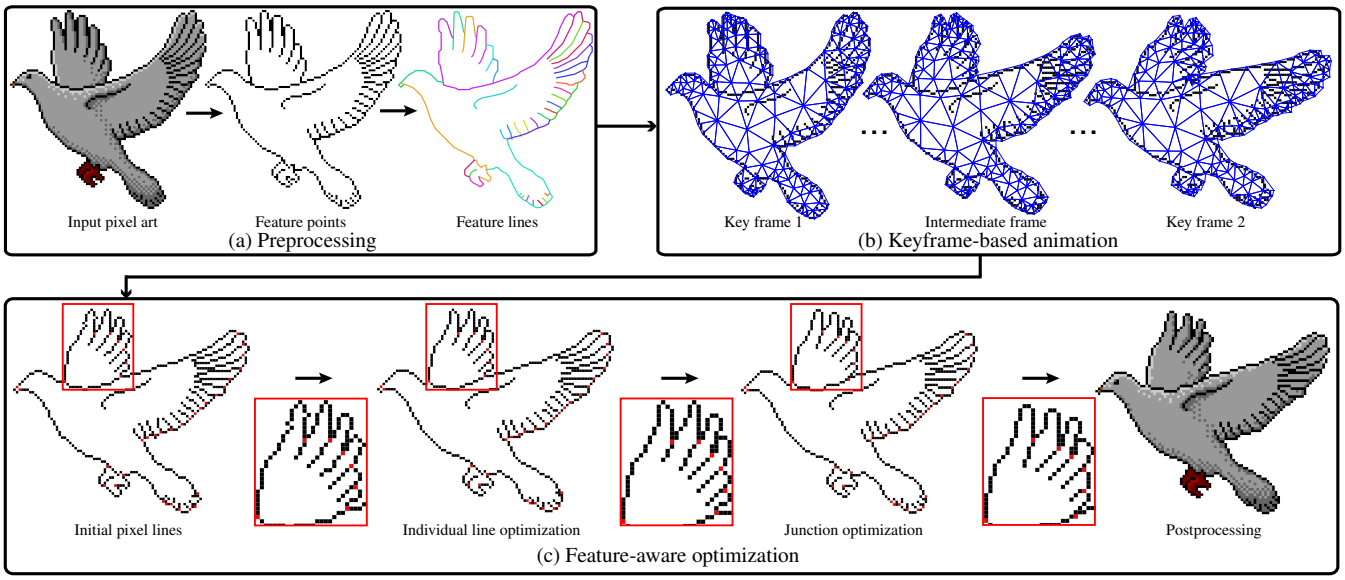


Figure 2: *Overview: Given an input pixel art image, our system first (a) extracts all feature pixels and constructs a set of feature lines in a preprocessing step. (b) Based on as-rigid-as-possible shape deformation and interpolation, key frames and intermediate frames are created as the initial animation sequence. (c) The feature lines of all the frames are further optimized to improve the visual quality and temporal coherence of the animation sequence. Finally, our system fills colors in the region bounded by feature lines (Input image: "Dove" © WhyMe).*

tional key-frame based animation framework to first specify a few key frames via interactive warping tools. Then the intermediate frames are smoothly interpolated between consecutive key frames (see Figure 2(b) and supplementary video). Specifically, each feature line is converted into a compact vector representation using polylines (see Appendix A for details). All the polylines are then embedded into a triangle mesh generated by applying Delaunay triangulation [She96] to the smoothed boundary pixels. For specifying a key frame, the user simply picks few anchors on mesh vertices and then performs intuitive drag and drop actions on arbitrary vertices. Then our system automatically updates the shape using as-rigid-as-possible deformation [IMH05]. Once all the key frames are set, the intermediate frames are interpolated using the method by [ACOL00]. The polylines of individual frames are reconstructed using the encoded barycentric coordinates. Finally, the initial animation frames are generated by rasterizing the reconstructed polylines using Bresenham's line drawing algorithm [Bre65]. Note that for each frame, we also generate a deformed image of the input pixel art image, which will be used for mapping colors to the pixels in-between feature lines after the optimization in the next stage.

Feature-aware optimization. Since the resolution of pixel art images is very low, the initial animation sequence usually contains severe artifacts, such as spiky and jaggy patterns. To improve the quality of the final animation sequence, we further optimize the shape of feature lines with respect to the visual quality and temporal coherence of the animation sequence (see Figure 2(c)). For this purpose, we design several tailor-made quality metrics that (i) preserve the deformed shape when updating feature lines; (ii) regularize the pixels of feature lines to ensure the quality of pixel art; and (iii) enforce the temporal coherence across frames. (see Section 4.1). Based on the above metrics, we optimize the shape of individual feature lines and the configuration of junctions that join feature lines using a set of local pixel-level refinement operators (see Section 4.2). Note that the initial rasterization and subsequent optimization are performed only on the feature lines. We employ a simple nearest neighbor color interpolation based on the warped pixel art image to recover pixel colors in-between feature lines and obtain the final results.

4. Feature-Aware Optimization

Formulation. The input to our feature-aware optimization is an animation sequence $\mathcal{S} = \{s_1, s_2, ..., s_N\}$, where s_n $(1 \leq n \leq N)$ denotes the n-th frame which contains M feature lines $\mathcal{L}_n = \{l_1^n, l_2^n, ..., l_M^n\}$. We formulate feature-aware pixel art animation as a feature line optimization problem, which refines the shape of individual feature lines and the configuration of the junctions among them. The objective function of the optimization is carefully designed to improve the visual quality and temporal coherence of the animation. We propose several local refinements at pixel level to mimick the pixel manipulations performed by the artists to help explore the solution space.

4.1. Quality metrics

We first define a set of quality metrics that measure the shape and appearance of feature lines for individual frames, along with feature line consistencies between neighboring frames. The proposed metrics serve as basic elements to compose the overall objective function for the subsequent optimization.

Shape similarity. This metric is defined per frame. It measures the similarity between the current feature lines under optimization (objective) and the corresponding deformed feature lines (reference)

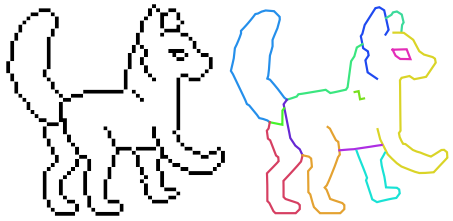


Figure 3: *The shape similarity is defined by referring the rasterized feature lines (left) to the corresponding feature lines without rasterization (right).*

without rasterization. This is to preserve the underlying shape during the optimization. Please note that the initial (rasterized) feature lines of the optimization are not used as reference because they usually contain artifacts as explained before.

Suppose l_m^n is the m-th feature line of the n-th frame and $l_m^{n\prime}$ represents the corresponding reference feature line. $l_m^{n\prime}$ is also generated based on shape deformation and interpolation, but without rasterization (see Figure 3), and it will be fixed during the optimization.

Since l_m^n and $l_m^{n\prime}$ are either polyline or can be easily converted to polyline, we employ polyline-based metrics for the similarity measurement. For simplicity, we still use l_m^n and $l_m^{n\prime}$ for the polyline representation. We consider two aspects for polyline similarity, *orientation* and *location*. The orientations of individual edges along a polyline can be represented using a turning function, where the dip angle at each point is represented as a function of the corresponding arc length [ACH*90]. The difference between two turning functions is defined as the area bounded by two function curves (see Figure 4). Note that l_m^n and $l_m^{n\prime}$ are both with arc-length parameterization. By taking all polylines into consideration, the overall orientation similarity metric of the n-th frame is defined as:

$$E_{so}^n = \sum_{m=0}^{M-1} Area[T(l_m^n), T(l_m^{n\prime})], \tag{1}$$

where T denotes the mapping which maps a polyline to its turning function. $Area(\ ,\ )$ represents the area bounded by two curves.

The above metric only preserves the orientation of feature lines and is invariant under polyline translation and scaling. To address this,

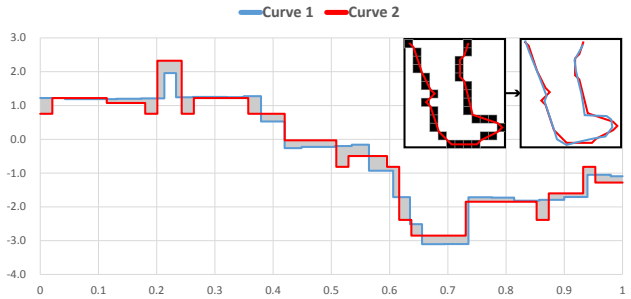


Figure 4: *We represent the orientation of a polyline using a turning function. The difference between two turning functions is measured by the area bounded by two function curves (in gray).*

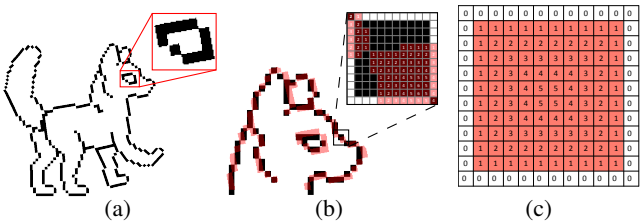


Figure 5: *(a) To accurately measure the coverage between l_m^n and $l_m^{n\prime}$, we scale $l_m^{n\prime}$ followed by high resolution rasterization to generate $L_m^{n\prime}$. (b) Different weights are assigned to pixels of $L_m^{n\prime}$ to indicate the importance of feature preserving with respect to L_m^n (in red). (c) The weight distribution over the pixels corresponding to a single pixel in the original resolution.*

we also involve the second metric to further preserve pixel locations along the polyline. The basic idea is to measure the coverage of l_m^n with respect to $l_m^{n\prime}$ so that the shape of $l_m^{n\prime}$ could be preserved by l_m^n during the optimization. To ensure estimation accuracy, we perform the computation at a higher resolution as follows. We up-sample l_m^n (10x) and achieve L_m^n, in which each pixel of l_m^n corresponds to 100 pixels. To up-sample $l_m^{n\prime}$, the simplest way is to perform rasterization first and then scale the image to high resolution, which however causes zig-zag artifacts. Instead, we up-sample the feature lines from the original pixel art image, then apply the image warping transformations for generating $l_m^{n\prime}$ to the up-sampled image, resulting in high resolution warping result $L_m^{n\prime}$ (see Figure 5(a)). For each pixel of $L_m^{n\prime}$, we assign a weight using its Manhattan distance to the boundary, indicating the importance for preserving $L_m^{n\prime}$ (see Figure 5(b)). For each pixel p_i of l_m^n, the coverage ratio c_i is measure by first accumulating the weights from pixels in $L_m^{n\prime}$, where those pixels overlaps the corresponding pixels of p_i in L_m^n. Then we normalize the coverage value using the sum of all weights corresponding to one pixel in the original resolution (see Figure 5(c)). The overall coverage of l_m^n according to $l_m^{n\prime}$ is defined as:

$$E_{sc}^n = \sum_{m=0}^{M-1} \frac{\sum_i (1-c_i)}{|l_m^n|}, \tag{2}$$

where $|l_m^n|$ is the number of pixels in l_m^n. Please note that we discard the pixels with $c_i < 0.5$ to favor improvements over pixels with less coverage.

Pixel art quality. We measure feature line quality per frame to avoid two types of artifacts, i.e., *spiky* and *jaggy* effects. The former is caused by implausible local extrema, while the latter is due to nonmonotonic slopes (see Figure 6).

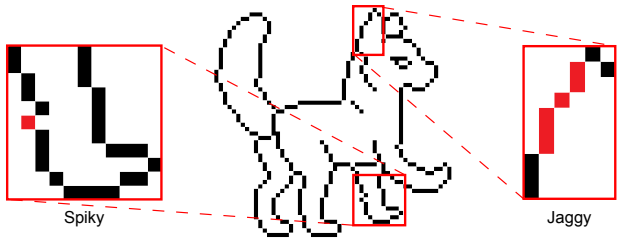
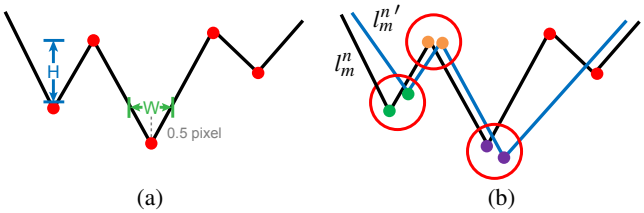


Figure 6: *Artifacts can be easily generated in the initial animation sequence, including spiky effects with implausible local extrema, and jaggy effects with nonmonotonic slope.*



(a) (b)

Figure 7: *(a) To resolve spiky effects with implausible extrema, we extract all the extrema (red dots) for each feature line. The height of an extremum is defined as the minimum vertical distance to neighboring extrema. The width of an extrema is the horizontal range with 0.5 pixel height variation. (b) The spiky effect of l_m^n is measured by comparing the extrema of l_m^n with those of $l_m^{n\prime}$ within a short distance (2 pixels). We only show vertical extrema here, while horizontal extrema are processed analogously.*

Similar as before, we measure the spiky effect of feature line l_m^n by comparing it with $l_m^{n\prime}$. Suppose l_m^n is already converted to the polyline representation, we first find the local extrema of l_m^n vertically and calculate the height and width therein as shown in Figure 7(a). The smoothness at an extremum is simply defined as its width. The above procedure is also applied for $l_m^{n\prime}$. For $l_m^{n\prime}$, we filter out the extrema where the height is less than one pixel by perturbing extreme pixels one step towards their neighboring pixels. Then for each extremum e_i in l_m^n, we search for the corresponding extremum e_i' in $l_m^{n\prime}$ within a distance threshold (2 pixels in our experiments), and store i into a set $\mathcal{P}$ if the corresponding extremum exists (see Figure 7(b)). The quality metric for avoiding spiky effect is defined as:

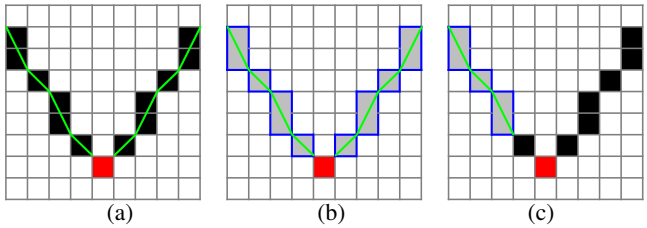
$$E_{qs}^n = \sum_{m=0}^{M-1} [\sum_{i\in\mathcal{P}} \frac{|Width(e_i) - Width(e_i')|}{Width(e_i) + Width(e_i')} + \sum_{i\notin\mathcal{P}} \frac{1}{Width(e_i)}], \quad (3)$$

where $Width(\cdot)$ denotes the width of an extremum.

To measure the quality of feature line l_m^n to avoid jaggy effect, we first use the identified extrema to divide l_m^n into a set of sub-polylines (see Figure 8(a)). For each sub-polyline, we further split its associated pixels into a sequence of horizontal or vertical segments, each of which is called a *span* (see Figure 8(b)). We measure the slope monotonicity within each sub-polyline by considering the slopes of every three consecutive spans (see Figure 8(c)). We count the total number of triplets with nonmonotonic slopes within l_m^n, denoted by $Nonmono(l_m^n)$. Then the quality metric regarding jaggy effect is defined as:

$$E_{qj}^n = \sum_{m=0}^{M-1} \frac{Nonmono(l_m^n)}{Span(l_m^n)}, \quad (4)$$

where $Span(\cdot)$ indicates the total number of spans within the initial feature line.



(a) (b) (c)

Figure 8: *(a) To resolve jaggy effects with nonmonotonic slopes, we divide a feature line into a set of sub-polylines (in green) using pixels at extrema (in red). (b) The associated pixels of each sub-polyline are further split into horizontal or vertical segments (called spans) highlighted using blue boundary lines. (c) The slope monotonicity is measured by the slopes of every three consecutive spans. One of the triplets is highlighted in gray here.*

Temporal coherence. One fundamental requirement for an animation sequence is the smooth motion transition between frames. This is rather important for pixel art animation because only limited pixels can be used to discretize the dynamic shape. In practice, artists need to pay significant efforts to fine tune pixels across frames, which is very expensive and error-prone. In this work, we carefully design a temporal coherence metric to ensure smooth shape motion between consecutive frames.

The temporal coherence between two consecutive frames s_{n-1} and s_n relies on corresponding feature lines l_m^{n-1} and l_m^n. The simplest way to measure coherence between two feature lines is to use distance based metric, e.g., Hausdorff distance. However, this is not sufficient for pixel art animation due to its low resolution nature. Hence we present a fine-grained temporal coherence metric at pixel level which quantitatively measures the cost of bringing all pixels of l_m^{n-1} to those of l_m^n. More specifically, as shown in Figure 9, we define a set of pixel manipulation operators for a single span, including *add*, *delete*, *substitute*, *merge*, and *split*, each of which is assigned a cost value according to the change on the involved pixels. Then the temporal coherence between l_m^{n-1} and l_m^n is defined as the minimal cost caused by a series of operators that map l_m^{n-1} to l_m^n. In our experiments, we perform an exhaustive search to enumerate all solutions with corresponding costs. Then the temporal coherence metric is defined as:

$$E_{tc}^{n-1,n} = \sum_{m=0}^{M-1} \frac{cost_{\min}(l_m^{n-1}, l_m^n)}{cost(l_m^{n-1}, \varnothing) + cost(\varnothing, l_m^n)}, \quad (5)$$

where $cost(\cdot,\cdot)$ indicates the cost of pixel line conversion using the proposed operators, and $\varnothing$ denotes empty pixel set. Note that we normalize the minimal cost with a brute-force cost by simply deleting all the pixels in l_m^{n-1} and adding all the pixels in l_m^n.

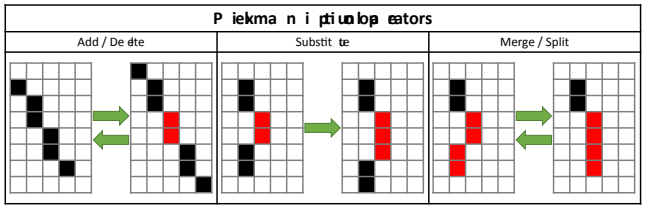


Figure 9: *The fine-grained temporal coherence metric is based on a set of pixel manipulation operators defined on a single span, including add, delete, substitute, merge, and split. A cost value is assigned to each operator based on the change of the involved pixels (highlighted in red), such as the size of the added/deleted span, the magnitude of displacement between the old and new spans, the movement when merging/spliting spans respectively.*

4.2. Optimization

Based on the proposed metrics in the previous section, we perform an optimization on each frame s_n to refine the pixels of each feature line l_m^n. The objective function guiding the optimization is defined as a weighted combination of the above quality metrics:

$$E(s_n) = w_{so}E_{so}^n + w_{sc}E_{sc}^n + w_{qs}E_{qs}^n + w_{qj}E_{qj}^n + w_{tc}E_{tc}^{n-1,n}. \quad (6)$$

In practice, the optimization is performed from the starting frame s_0 to the ending frame s_{N-1} consecutively. Note that when $n = 0$, the temporal coherence term $E_{tc}^{n-1,n}$ is not included in the objective function since it is not well defined.

Given an input frame, we apply a set of pixel refinement operators to help explore the solution space and refine the result by minimizing the objective function. In order to preserve the topology of feature lines, the operators are firstly performed on individual feature lines and then on the junctions between feature lines. The overall optimization procedure is described in Algorithm 1. The following paragraphs elaborate the key components of the optimization.

Algorithm 1 Optimization Procedure

Initial animation sequence $\mathcal{S} = (s_0, s_1, \ldots, s_{N-1})$
Initialize parameters $w_{so} = 1.0, w_{sc} = 0.5, w_{qs} = 1.0, w_{qj} = 0.5, w_{tc} = 2.0$
$n = 1$
while $n \leq N$ **do**
    Select frame s_n
    $m = 1$
    while $m \leq M$ **do**
        Select feature line l_m^n
        repeat
            Choose operator $op^* \in$ {add, delete, shift, merge, split}(see Figure 10) and apply to l_m^n such that $E(s_n)$ decreases the most
            Apply op^* to l_m^n and update s_n
        until $E(s_n)$ cannot be decreased
        $m \leftarrow m + 1$
    end while
    $n \leftarrow n + 1$
end while
repeat
    Locally perturb a new junction between feature lines
    Optimize relevant feature lines using refinement operators such that $E(s_n)$ can be decreased
    Update s_n
until $E(s_n)$ cannot be decreased

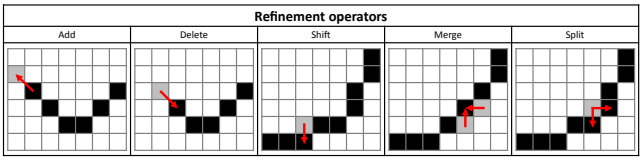


Figure 10: *We propose a set of refinement operators to explore the solution space during the optimization.*

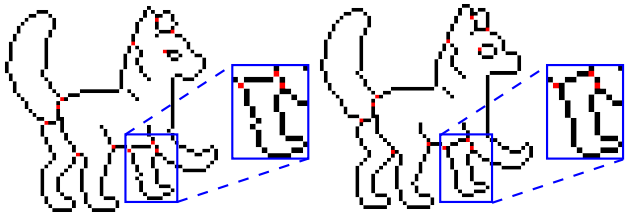


Figure 11: *Individual line optimization optimizes individual feature lines while fixing all the junction pixels (in red) according to the objective function.*

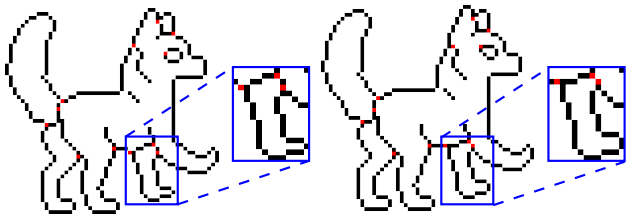


Figure 12: *Junction optimization optimizes the location of junction pixels (in red) according to the objective function.*

Refinement operators. To explore the solution space, we define a set of refinement operators, including *add*, *delete*, *shift*, *merge*, and *split* (see Figure 10). These operators are similar as those proposed for temporal coherence metric. The difference is that the operators here are defined on a single pixel other than on a span. This is to allow pixel refinement at the lowest level.

Individual line optimization. This step optimizes all feature lines separately. For each feature line l_m^n, we fix all the junction pixels and try all possible refinement operators on all pixels. To update the current solution, we select the refinement operator (on a specific pixel) with the most significant decrease of the objective function value. The above procedure repeats until no pixel can be refined. Figure 11 shows the effect of individual line optimization.

Junction optimization. After optimizing the individual lines, we perform an optimization on all the junctions between feature lines. Specifically, we iterate each junction and perturb its location in a 8-connected neighborhood while fixing other junctions. The relevant feature lines of this junction are then refined as in feature line optimization. If the objective function value decreases, we accept the result and further optimize the neighboring junctions. The whole process repeats until no junctions can be refined. Figure 12 shows the effect of junction optimization.

5. Results and Evaluation

We have tested our system on a wide range of pixel art images from different categories (human, animal, man-made object, etc.), resulting in 31 appealing pixel art animations with varying degrees of complexity. A few examples can be found in Figure 13. Since the best visual experience of our results can be achieved by displaying all the frames consecutively, we refer the reader to supplementary material for the complete animation sequences of all 31 examples.

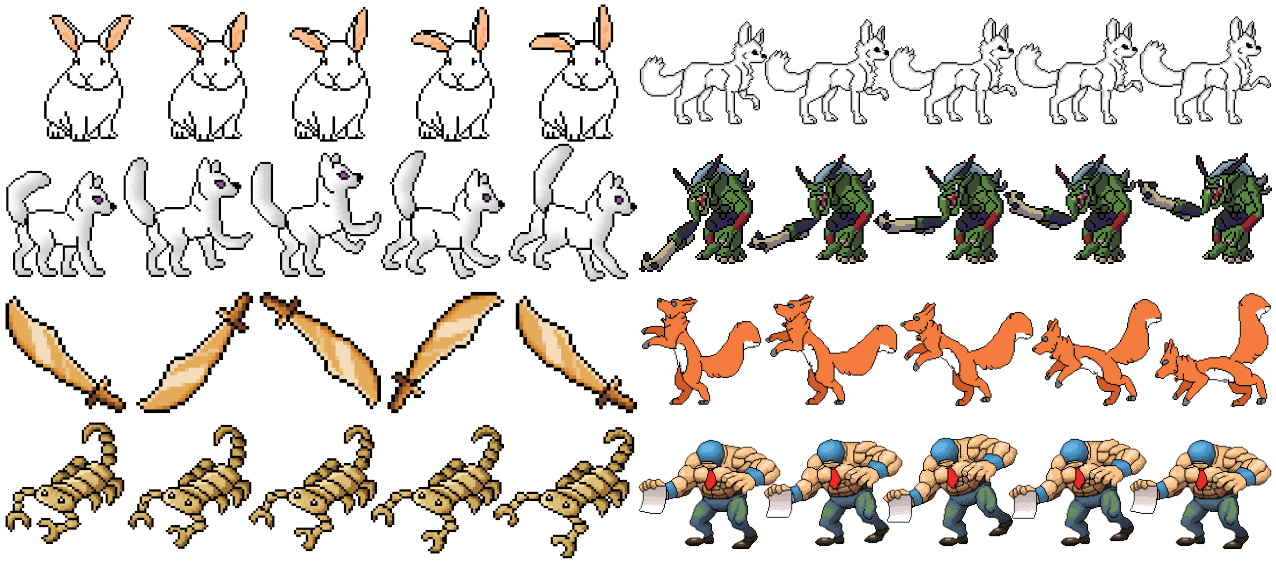


Figure 13: *Eight pixel art animations created by our system. Here we only show five intermediate frames for each sequence. The complete animation sequences can be found in the supplementary material (Input images: "Bunny" © KennaLeeStratton, "Wolf" © Coranila, "Saber" © Manning Leonard Krull, "Scorpion" © ShimmeringDream, "Dog" © SkyeFatty, "Ogremon" © BANDAI Co., Ltd, "Fox" © Mixellulu, "Man" © Derek).*

5.1. Evaluation

We also conducted several experiments to quantitively and qualitatively evaluate the performance of our system. First, we compare the visual quality of our animation sequence with those generated by three alternative methods via a user study. Then we validate our tailor-made quality metrics by comparing the results with or without a particular metric during the optimization. Lastly, we report the runtime performance of our system.

User study. Since the dominant factor to the quality of our results is the feature-aware optimization (Section 4), we evaluate its performance via a comparison with three alternative rasterization algorithms as follows. (i) Down-scaling the image comprises feature lines $L_m^{n\prime}$ $(1 \le m \le M)$ using the nearest neighbor sampling. (ii) Applying a simple down-sampling approach which first groups pixels in the original resolution that are covered by $L_m^{n\prime}$ $(1 \le m \le M)$ at high resolution (10x), and then sorts pixels in the descending order based on its coverage ratio. The final result is obtained by iteratively removing pixels with the least coverage ratio while keeping feature lines connected. (iii) Applying a state-of-the-art vector line rasterization algorithm designed for pixel art, called Superpixelator [IVK13], to $l_m^{n\prime}$ $(1 \le m \le M)$. Note that the above methods are performed for each frame s_n separately. There were 15 examples used in the designed user study. For each example, four animation sequences generated from different methods were displayed with no specific layout. The subjects were requested to pick the most visually pleasing one. Each example was evaluated by 50 subjects.

Figure 14 shows the normalized votes for the four methods. We further conducted two-proportion z-tests to verify the statistical significance of the difference between the normalized votes for all the examples. The result indicates that our system significantly outperforms the other three methods with overall p-value < 0.01. As shown in Figure 15, we can easily see that the nearest-neighbor method leads to severe artifacts of broken lines, while simple down-sampling and Superpixelator generate pixel lines with only mediocre quality. Moreover, all three alternatives result in noticeable temporal jitter during playback of the animation sequences. Our algorithm in contrast can generate high-quality pixel art animation that is visually pleasing and with guaranteed temporal smoothness. The complete animation sequences of this comparison can be found in the supplementary material.

Performance of quality metrics. In this experiment, we evaluate the effectiveness of our tailor-made quality metrics by comparing the results generated with different parameter settings as: (i) using only shape similarity metric ($w_{qs} = w_{qj} = w_{tc} = 0.0$); (ii) disabling

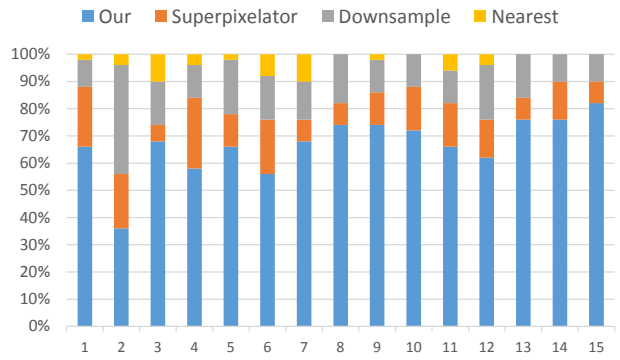


Figure 14: *Normalized votes for the visual quality of the animation sequences generated by our algorithm and three alternatives on 15 examples.*

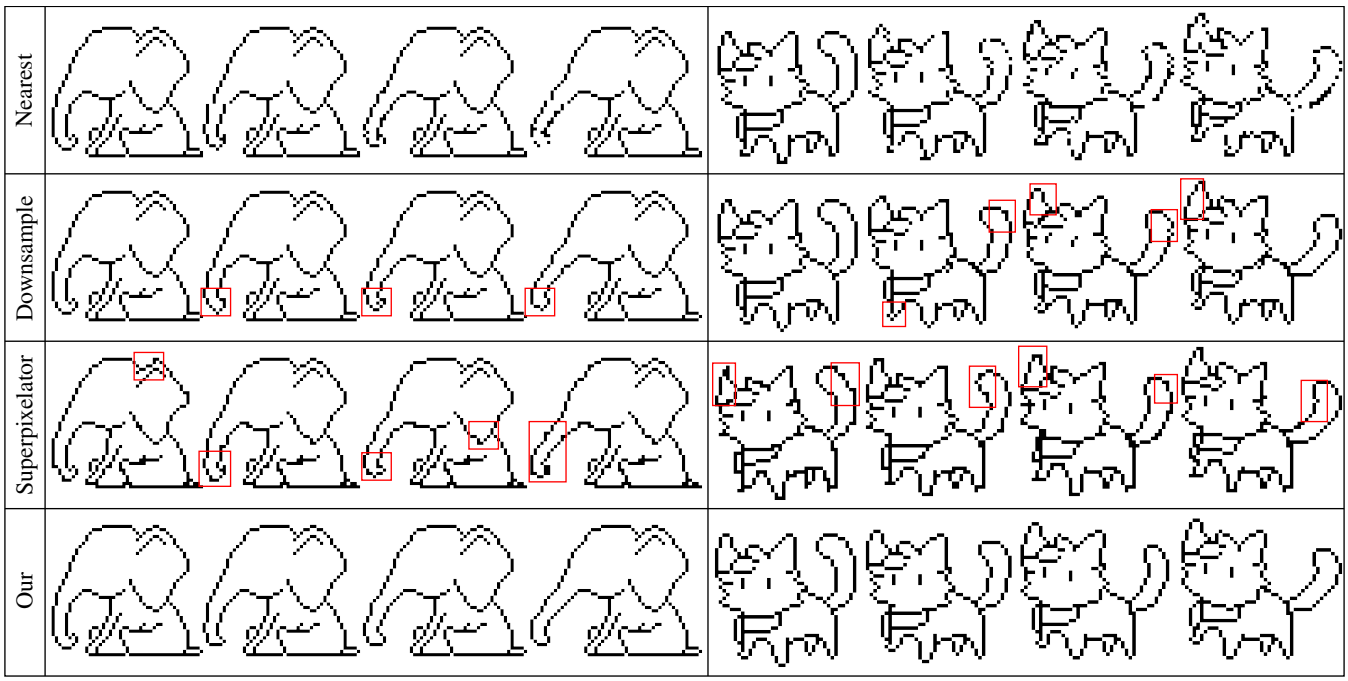


Figure 15: *Comparison of our results with the other three alternatives. Our system generates superior results (bottom row) than the other three alternatives, which suffer noticeable artifacts such as broken outlines (top row), spiky and jaggy effect (red boxes in the 2nd and 3rd rows), and temporal inconsistency (see supplementary material for complete animation sequences) (Input images: "Elephant" © LimboNoBaka, "Kitten" © Iceyspade).*

temporal coherence metric ($w_{tc} = 0.0$); and (iii) using full model with all metrics. The results show noticeable improvements on the visual quality as we augmenting the model of using only shape similarity metric (see Figure 16(a)) and adding only the pixel art quality metric (see Figure 16(b)). Using the full model leads to a further quality boost which significantly reduces the temporal jitter effects (see Figure 16(c)). Please refer to the supplementary material for the complete animation sequences of this experiment.

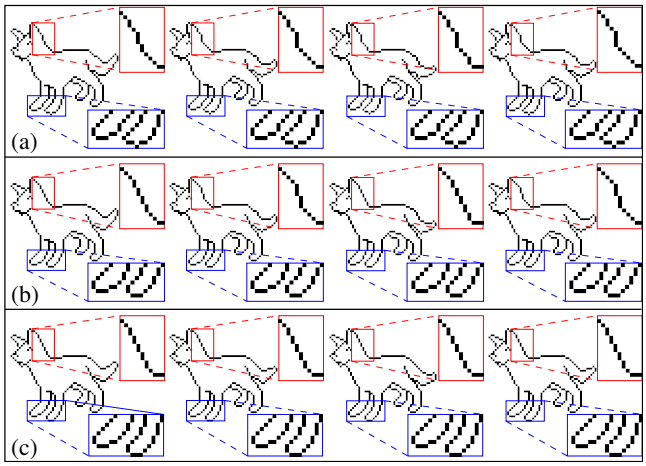
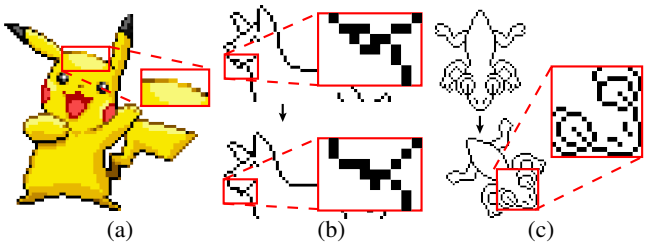


Figure 16: *Comparing results with different quality metric settings in the optimization. (a) Using only shape similarity metric. (b) Disabling the temporal coherence metric. (c) Full model. Noticeable quality improvements are highlighted within red and blue boxes (Input image: "Dog" © Karijn-s-Basement).*

Timing performance. Our system can generate high-quality animation within a reasonable amount of time. The preprocessing step takes less than one second on average. The timing of manually resolving the ambiguities (if necessary) depends on the complexity of the input pixel art, especially the junctions. Note that the preprocessing only needs to be performed once for each input image. The major computation cost of our system is due to feature line optimization. The running time here is proportional to the input image resolution as well as the structural complexity of the extracted feature lines. For example, for the wolf image shown in Figure 11, which contains 50×50 pixels, 13 feature lines, and 12 junctions, it took 30 seconds to resolve ambiguities and 5 seconds per frame on average during the optimization.

5.2. Limitations

The performance of our system is subject to the following limitations. (i) The feature lines extraction is currently heuristic and fails to handle pixel art images with strong antialiasing (see Figure 17(a)). Such images require more manual efforts to resolve ambiguities. (ii) The spatial relationship between feature lines is not modeled in the current formulation. Therefore, two nearby feature lines can merge or intersect with each other after the reconstruction (see Figure 17(b)). However, this problem rarely occurs in our experiments. (iii) The system does not account for the symmetry of input shapes, and may generate asymmetric shapes in the final



(a) (b) (c)

Figure 17: *Limitations. (a) Feature line extraction can be tricky with pixel art images containing antialiasing patterns. (b) The system fails to preserve the spatial relationship between nearby pixels from different feature lines. (c) The system does not take into account the symmetry of input shapes during the animation (Input images: "Pikachu" © Pokémon Ltd, "Dog" © Karijn-s-Basement, "Lizard" © cesarloose).*

results (see Figure 17(c)). (iv) The feature-aware optimization is formulated based on the condition that the structure of feature lines is consistent and the correspondences between feature lines are explicit between frames. Thus the system dose not support animated shapes with varying topology (e.g., heart breaking, a star-like shape evolves from a single pixel).

6. Conclusion and Future Work

How to produce a high-quality animation from a given pixel art image is an important but non-trivial problem. In this work, we present a novel key-frame based animation framework tailored to pixel art images. With the help of our system, the users can quickly generate key-frames through a set of intuitive tools. The system automatically interpolates intermediate frames and optimizes the visual quality of the prominent feature lines of individual frames, as well as the temporal coherence between frames. Such tasks could otherwise demand tremendous manual efforts using the existing systems. We demonstrate the efficiency and efficacy of system by generating high-quality pixel art animation sequences on various examples. Our system would be a practical tool for artists to generate plausible animation sequences that only require minor efforts, making the animation production process much more efficient.

In the future, it would be interesting to explore the following directions. (i) Since the potential users of our system are professional artists, we plan to conduct user study and fine tune the system according to their feedback. (ii) We would like to investigate other characteristics of pixel art, such as color dithering and antialiasing patterns, for further quality improvements. (iii) The robustness of feature line extraction can be further improved using sophisticated image vectorization method [KL11]. (iv) It is also worth exploring how to enhance the current system with 2.5D modeling [RID10, YJL*15] and more advanced deformation tools (e.g., [YFW12]), such that animations with part occlusion and topology change can also be handled.

Acknowledgements

We are grateful to the anonymous reviewers for their comments and suggestions. The work was supported in part by the Ministry of Science and Technology of Taiwan (103-2221-E-007-065-MY3, 105-2221-E-007-104-MY2 and 104-2218-E-004-003), EPSRC Grant EP/K02339X/1 and EP/M023281/1.

Appendix A: Feature Line Vectorization

In our implementation, we employ a simple feature line vectorization algorithm to construct a compact geometric representation of the underlying 2D shape for subsequence warping and interpolation. More specifically, the feature line vectorization follows a three-step procedure as shown in Figure 18. First, we divide each feature line composed by pixels into a set of horizontal/vertical segments, each of which is called a *span*. Second, we extract all the points connecting neighboring spans and link these points within each span using a straight line segment, resulting in an initial vectorized polyline. Finally, we insert additional points into the polyline at each extremum span by creating a new point at the center of each pixel. Note that after vectorizing individual feature lines, we further connect each vectorized feature line to its associated junction pixels to form the final vectorization result of the entire shape.

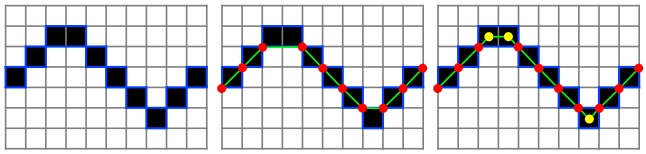


Figure 18: *Three-step polyline vectorization.*

References

[ACH*90] ARKIN E. M., CHEW L. P., HUTTENLOCHER D. P., KEDEM K., MITCHELL J. S. B.: An efficiently computable metric for comparing polygonal shapes. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (1990), SODA '90, pp. 129–137. 4

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH '00, pp. 157–164. 3

[ADO10] ADOBE, INC.: Adobe illustrator cs5. www.adobe.com/products/illustrator.html, 2010. 2

[BBT11] BÉNARD P., BOUSSEAU A., THOLLOT J.: State-of-the-art report on temporal coherence for stylized animations. In *Comp. Graphics Forum* (2011), vol. 30, pp. 2367–2386. 2

[BCGF10] BÉNARD P., COLE F., GOLOVINSKIY A., FINKELSTEIN A.: Self-similar texture for coherent line stylization. In *Proc. of NPAR* (2010). 2

[BNTS07] BOUSSEAU A., NEYRET F., THOLLOT J., SALESIN D.: Video watercolorization using bidirectional texture advection. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3 (2007). 2

[Bre65] BRESENHAM J. E.: Algorithm for computer control of a digital plotter. *IBM Syst. J. 4*, 1 (1965), 25–30. 2, 3

[Bre77] BRESENHAM J.: A linear algorithm for incremental digital display of circular arcs. *Commun. ACM 20*, 2 (1977), 100–106. 2

[Des] DESCOTTES J.: Piskel. www.piskelapp.com. 1

[GDA*12] GERSTNER T., DECARLO D., ALEXA M., FINKELSTEIN A., GINGOLD Y., NEALEN A.: Pixelated image abstraction. In *Proc. of NPAR* (2012), Eurographics Association, pp. 29–36. 2

[IK12] INGLIS T. C., KAPLAN C. S.: Pixelating vector line art. In *Proc. of NPAR* (2012), pp. 21–28. 2

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3 (2005), 1134–1141. 3

[IVK13] INGLIS T. C., VOGEL D., KAPLAN C. S.: Rasterizing and antialiasing vector line art in the pixel art style. In *Proc. of NPAR* (2013), pp. 25–32. 2, 7

[KL11] KOPF J., LISCHINSKI D.: Depixelizing pixel art. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4 (2011), 99:1–99:8. 2, 9

[KLC*15] KUO M.-H., LIN Y.-E., CHU H.-K., LEE R.-R., YANG Y.-L.: Pixel2brick: Constructing brick sculptures from pixel art. *Comp. Graphics Forum (Proc. Pacific Graphics) 34*, 7 (2015), 339–348. 2

[KSP13] KOPF J., SHAMIR A., PEERS P.: Content-adaptive image downscaling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 32*, 6 (2013), 173:1–173:8. 2

[NSC*11] NORIS G., SỲKORA D., COROS S., WHITED B., SIMMONS M., HORNUNG A., GROSS M., SUMNER R. W.: Temporal noise control for sketchy animation. In *Proc. of NPAR* (2011), pp. 93–98. 2

[OH12] O'DONOVAN P., HERTZMANN A.: Anipaint: Interactive painterly animation from video. *IEEE Trans. Vis. Comput. Graph. 18*, 3 (2012), 475–487. 2

[PM12] PARENT M., MUNIZ E.: Spriter. brashmonkey.com, 2012. 1

[RID10] RIVERS A., IGARASHI T., DURAND F.: 2.5D cartoon models. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 59:1–59:7. 9

[Sel03] SELINGER P.: Potrace: a polygon-based tracing algorithm. potrace.sourceforge.net, 2003. 2

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Selected Papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering* (1996), FCRC '96/WACG '96, pp. 203–222. 3

[Vec10] VECTOR MAGIC, INC.: Vector magic. vectormajig.com, 2010. 2

[VSS12] VERMEHR K., SAUERTEIG S., SMITAL S.: eboy. hello.eboy.com, 2012. 1

[Wu91] WU X.: An efficient antialiasing technique. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991), SIGGRAPH '91, pp. 143–152. 2

[YFW12] YANG W., FENG J., WANG X.: Structure Preserving Manipulation and Interpolation for Multi-element 2D Shapes. *Comp. Graphics Forum (Proc. Pacific Graphics) 31*, 7 (2012), 2249–2258. 9

[YJL*15] YEH C.-K., JAYARAMAN P. K., LIU X., FU C.-W., LEE T.-Y.: 2.5D cartoon hair modeling and manipulation. *IEEE Trans. Vis. Comput. Graph. 21*, 3 (2015), 304–314. 9

[Yu13] YU D.: Pixel art tutorial. makegames.tumblr.com/post/42648699708/pixel-art-tutorial, 2013. 1, 2