ISSN 1213-6972 Journal of WSCG http://www.wscg.eu Vol.28, No.1-2, 2020

Intermediate Representations for Vectorization of Stylized Images

D.-Amadeus J. Glöckner*
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
daniel-amadeus.gloeckner@hpi.de

Lisa Ihde*
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
lisa.ihde@student.hpi.de

Jürgen Döllner
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
juergen.doellner@hpi.de

Matthias Trapp
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany
matthias.trapp@hpi.de

Figure 1: Exemplary results of the presented approach: input image (left), vectorized result of toon stylization (middle), and vectorized result of half-tone stylization (right).

ABSTRACT

This paper presents a new approach for the vectorization of stylized images using intermediate data representations to interface image stylization and vectorization techniques. It enables the combination of efficient GPU-based implementations of interactive image stylization techniques and the advantages of vectorized image representations. We demonstrate the capabilities of our approach using half-toning and toon stylization techniques.

Keywords: Image stylization, image processing, vectorization, half-toning, toon, interaction

1 INTRODUCTION

1.1 Motivation

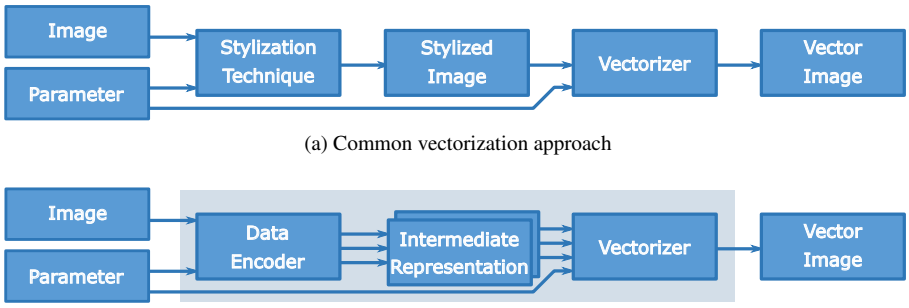
Image stylization techniques (e.g., half-toning [3] or toon [14]), and their applications offer users the opportunity to express their creativity and increasingly attracted attention during recent years. There are numerous stylization techniques that operate on raster images. Often, these can be implemented efficiently using Graphics Processing Units (GPUs). However, with respect to viewing or reproduction (e.g., for printing purposes), the resulting raster images are limited due to their limited spatial resolution.

In contrast thereto, vector images or graphics can be easily rasterized to required resolutions. However, their creation, editing and manipulation is often a cumbersome task. This work aims at combining efficient GPU-based implementation for image stylization and the advantages of vector-based presentations of their results (Figure 1).

1.2 Problem Statement

Vectorization techniques use raster images as input and generate a respective output vector image based on user defined settings such as number of colors and respective thresholds. Thus, their output often differs with respect to the number of colors being conveyed and the number of polygon required to represent image features such as colored areas or edges. However, they do not take information about the respective stylization technique into account (e.g., regions of unified color, edges, etc.). This often impacts the visual and data quality of results (e.g., high number of polygons or no explicit edge representations) and can lead to imprecise presentations of the stylization to be achieved (Figure 3).

*These two authors contributed equally



(a) Common vectorization approach

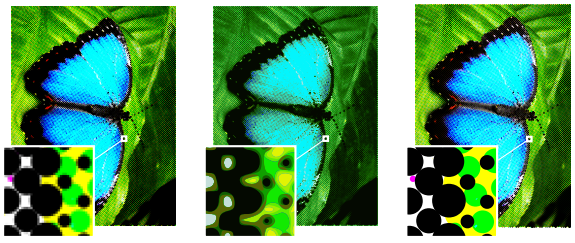
(b) Vectorization with intermediate representation

Figure 2: Comparison between a common vectorization approach (a) and the concept of stylization-specific intermediate representations (b).

To counterbalance this, vectorization would require high-resolution input data to convey fine features. Subsequent editing or animation of vectorization results, e.g., changing half-tone elements or its size and orientation, is hard or would require to perform stylization and vectorization repeatedly, which can be a tedious process. Thus, to support fast image stylization and high-quality vectorization results, it seems promising to research intermediate representations that encode specifics of the respective stylization approach and facilitate representation and manipulation of the vectorized results.

1.3 Approach and Contributions

Using the naive method, the raster image is read in with the custom parameters for a stylization technique, which creates a stylized image. The stylization output is again represented as a raster image and serves as the subsequent input image for vectorization that generates the final vector image. However, this vectorization is performed without any prior information describing the nature of the stylization. Instead of directly applying the stylization to an input image, our approach encodes a stylization-specific raster-based intermediate representation that is used to generate vector images, subsequently. Such representations acting as "data maps" are suitable for fast GPU-based processing and facilitate interactive applications.



(a) Input raster image (1440 × 1920 pixels) (b) Inkscape vectorization (12 colors) (c) Proposed approach (114 120 elements)

Figure 3: Comparison of vectorization results of a half-toned input image (a). Using the method of (b), the input image was vectorized with Inkscape yielding half-tone elements that can hardly be recognized. With the method of (c), the element's position, size, and color is defined by an intermediate representation that is used for its instancing during vectorization.

To summarize, this paper makes the following contributions:

(i) Concept for generating intermediate representation in order to create a vector image of a stylization effect.

(ii) Minimal adaptation of existing effect pipelines and their rendering passes with shader programs to use the raster-based outputs as basis for vectorization.

(iii) Provision of parameters for interactive modification of vectorization, which we demonstrate by stylization effects such as toon and half-tone filter.

The remainder of this paper is structured as follows. Section 2 reviews related work with respect to vectorization approaches in general and specific to stylized images. Section 3 presents the basic concept of our approach as well as implementation details specific to two exemplary image stylization techniques. Section 4 discusses the presented approach and Section 5 concludes this paper.

2 BACKGROUND

Antoniou *et al.* [1] has pointed out the inflexibility of raster image formats and describes a conversion of such into XML-based structure like Scalable Vector Graphics (SVG) files. While raster images are based on pixels, in an XML-based environment elements such as points, lines or polygons can be reused. However, this approach only describes how to convert each pixel directly into a rectangle and thus does not use other shapes.

Further, Olsen and Gooch introduced an edge-based image reconstruction method that extracts vector edges using tracing, which can be applied on photographs [8].

(a) Intermediate representation (adjusted) (b) Vector image using a dot element (c) Vector image using a flower element

Figure 4: Comparison of the intermediate representation (a) and two vector images (b) and (c). Each pixel of (a) encodes element color, size, and position.

The result is a vector image in grayscale, which is qualitatively well done, but this simplification does not include the color details from the original image. With respect to this, they showed that the vector-based image representation can be more memory efficient than raster images.

Noris *et al.* [7] present a vectorization method for raster images of line drawings with a focus on junction configurations. It produces a vectorization representation of the lines, but these have all the same width. Additionally, small details are not be captured by the topology extraction and the method needs solid color areas in the input image. Our approach allows the extraction of line widths from the input image and thus lines of variable width.

Simo-Serra *et al.* [13] focused on simplify sketch drawings and trained a convolutional neural network with a new data set of pairs of rough and simplified sketch drawings. The network provides a clean vector result out of a raster image with a rough pencil sketch. Their approach establishes the state of the art in sketch simplification. Additionally, the computation time using the GPU-based model achieves times under a second. Unfortunately, the approach depends on the quality and quantity of the training data. Furthermore, only line drawings are recognized, but no color areas or other compositions of an image. Therefore, this method is not suitable for vectorizing complex image stylization effects.

Figure 5: Half-tone image with multiple different elements ('W','S','C',G') at once.

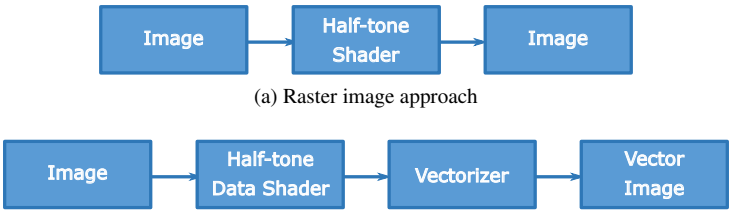
Xie *et al.* [15] treats vector images in an artistic way and promote the user to guide the vectorization interactively. For this purpose, the possible edges are calculated at first and the user then decides which ones should be used. In addition, the user can also draw edges or modify edges. The colors have to be applied manually, so there is no recognition of colored areas.

Favreau *et al.* [6] present a vectorization algorithm for photographs into cliparts based on stacking colored polygons. These layers are either opaque or semi-transparent and are faded to give the final result. But a segmented image is used as input, which must be created manually to generate a more stylized clipart.

Faraj *et al.* [5] explores the geometrical structure of an image to perform local operations like replacing or rotation. For it, a topographic map is used, which has a hierarchical order of color shapes in stacked layers [2]. Thus, this work can be considered as an image-abstraction method, which does not store the recognized mathematical forms in a vector image but in a tree. So this work is limited to the few operations that are offered. Our approach, however, saves all recognized components as SVG and can therefore be manipulated with common editing tools.

Several papers focus on image vectorization by detecting edges and shapes. Most of these works are limited to the color variety or only line vectorization. Both aspects are considered in our approach and are especially important for e.g., the toon effect. Moreover, our approach deals with the composition of stylization effects and examines the structure of effects in order to create the best possible generation of a vector image variant with the effect.

The previous works would try to generate a vector image from an input image with applied stylization effect. Thus the recognized areas and lines are vectorized independently of the applied style and the vector image is not sustainable. Possible later editing of this image



(a) Raster image approach

(b) Our vectorized approach

Figure 6: Comparison between a raster image half-toning pipeline (a) and our slightly modified vectorized approach (b).

e.g., replacing of elements by half-toning would be very complex or not possible (see Figure 3b).

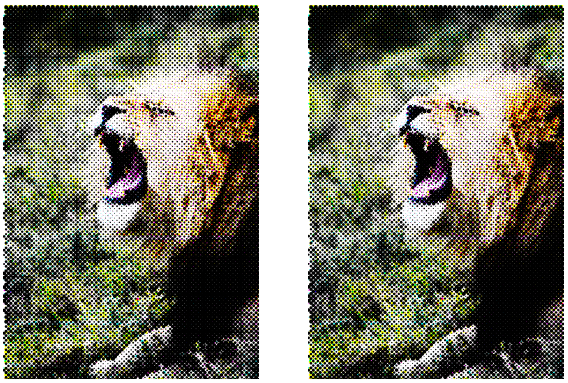
3 VECTORIZATION CONCEPT

3.1 GPU-based Image Stylization

In order to access different intermediate steps of a stylization effect to be used for vectorization, a system is required where a stylization effect is divided into individual sub-steps, which can be combined as desired. Our approach is based on the work of Semmo *et al*. [12], which provides a GPU-based framework for complex image stylization effects and was also used for ProsumerFX [4]. It consists of three components, which are explained below:

Effect pipeline: An *effect pipeline* defines the individual steps that are applied to an input image to produce an output image at the end.

Effect: A single step of the effect pipeline is an *effect* (e.g., difference-of-Gaussians filter [14]). This effect can have parameters that can be adjusted by the user.



(a) Half-tone vector image without sub pixel offset

(b) Half-tone vector image with sub pixel offset

Figure 7: Comparison of the half-tone stylization with and without sub pixel offset. Without it the rounding to whole pixel positions can create error patterns which are visible in the output (a). The use of the offset textures increase the positional resolution enough to eliminate the error patterns (b).

Rendering pass: For the execution of the effects the *rendering passes* are needed, which are available as shader programs or C++ code.

3.2 General Approach

Figure 2 shows a comparison between a common vectorization approach (a) and the proposed one (b). While common vectorization of stylized images is performed by applying a stylization technique to an input image, resulting in a stylized raster image that is converted to a vector image using tracing [11]. In contrast thereto, our approach comprises the following main components:

Data Encoder: The encoder basically implements the image stylization technique and generates the intermediate data specific to this technique. Therefore, it reads in the image data and user parameters and executes the GPU-based stylization.

Intermediate Representation: The intermediate data represents the major components of the stylization Gestalt (e.g., colored areas edges, half-toning elements) and is represented using textures. These different textures can have different channel counts, do not necessarily have to have the same resolution as the input image, and can be obtained using render-to-texture in combination with multiple render-targets.

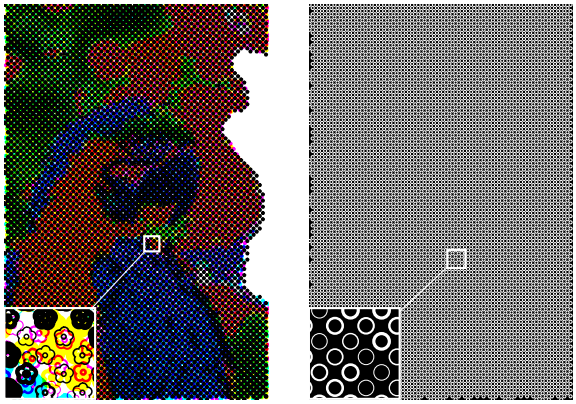
Vectorizer: A stylization-specific vectorizer receives the intermediate representation with user parameters and generates a final vector image. This makes it possible to separate the vectorization into substeps: thus, different components of the effect can be vectorized separately and then merged for the result. This approach enables to generate an specifically structured vector image for the respective stylization. To allow easy reuse of data encoders and vectorizers we integrated these as image processing passes into a modular system. These components can be combined with different image preprocessing steps to create new image stylizations.

In the following, we describe how these three components can be adapted to different stylization techniques.

(a) Half-tone vector image with no rotation applied to the elements

(b) Half-tone vector image with random rotation applied to the elements

Figure 8: Comparison of aligned and randomly rotated halftone elements. Aligned half-tone elements with an recognizable orientation introduce noticeable patterns (a). This can be avoided by randomly rotating the elements (b).



(a) Line thickness modulated multi channel image

(b) Line thickness modulated single channel image[1]

Figure 9: Color intensity modulated using line thickness instead of shape size.

3.3 Half-toning Vectorization

Half-toning is a common reproduction technique for photography in printing, where the continuous tones of the images are represented by full-tone dots of varying size, shape, and density [3]. For the printing medium, usually the Cyan-Magenta-Yellow-Key (CMYK) colors are used for the point-data, which are mixed subtractively. This image style can be recreated perfectly with SVGs since it requires only single colored shapes, which can easily be represented in the SVG format.

Effect Pipeline. The original pipeline consists only out of one step. This is a GLSL shader getting a raster image and some parameters as input and outputting a raster image with the half-tone effect applied (Figure 6a). In our pipeline the shader is slightly modified and becomes the data encoder and its output is the intermediate representation. Furthermore, as a new step, a vectorizer is added, that reads the intermediate representation and generates the SVG output (Figure 6b). Vectorization for half-toning stylization can be achieved by implementing these components as follows.

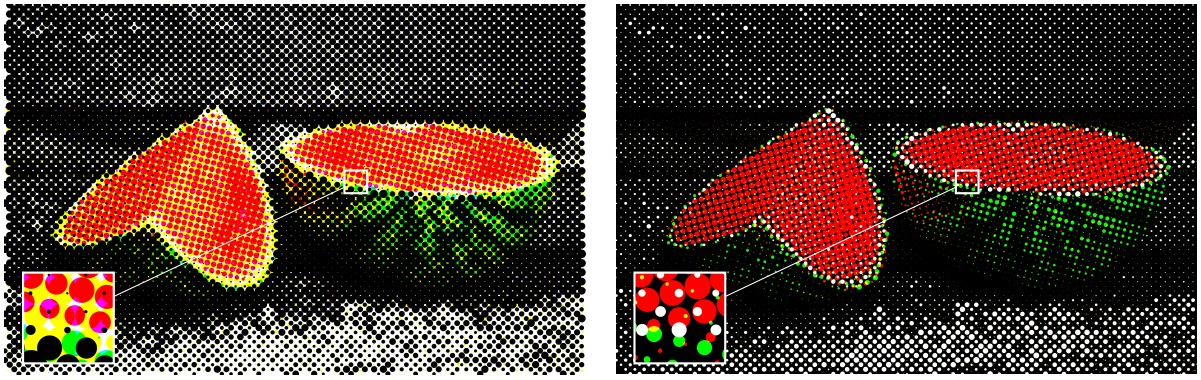
Data Encoder. We use a variation of a single-pass GLSL shader-based half-tone stylization. Here, the individual elements constituting a half-tone stylization are not rendered directly into a raster image, but the determined element positions, colors, and sizes are encoded in three textures.

Usually the half-tone shader would determine for each pixel the position of the closest half-tone element center and the size of this element. This information is than used to determine if the examined pixel is inside the element radius and colored accordingly. This is done for each of the color layers (usually cyan, magenta, yellow and black). In our modification we only set the value if the pixel is less then half a pixel away from the center, therefore only modifying the pixel at the center of the element. Also instead of choosing the color dependent on the half-tone color that is processed, this just determines which texture channel is written to, as described in the next paragraph.

Intermediate Representation. This is a set of three textures with pixels of different colors (Figure 4a). The color of the half-tone element is determined by the color channel where the red, green, blue and alpha channels represent the cyan, yellow, magenta and black channels respectively. The half-tone element size is determined by the intensity of each pixel in the corresponding channel in the first texture. Pixels with a value of zero do not represent an element. The position of the half-tone element is determined by the position of the pixel. To improve the positional accuracy and therefore prevent artifacts introduced by rounding to a full pixel position, we use the other two textures. These encode a sub-pixel offset in the x- and y-direction respectively (Figure 7). This increase in resolution allows us to decrease the actual resolution of the three intermediate textures. The resolution of the three textures is not dependent on the input image but on the half-tone frequency. The resolution must be big enough that two elements of the same color layer do not correspond to the same pixel position.

Vectorizer. This outputs an SVG for further editing. First, half-tone elements are generated using a template (cf. Line 5 in Listing 1). Following to that, a group element (`<g>`) is created for each of the CMYK color channels. Inside these, instances of the template are created for each of the non zero pixels in the main tex-

[1] Depending on the used PDF viewer it might be necessary to zoom this figure to full screen size.



(a) Half-tone vector image with the color space CMYK for the half-tone elements.

(b) Half-tone vector image with the color space RGB for the half-tone elements.

Figure 10: Comparison of the CMYK and RGB color spaces for the halftone elements. Images with a high black content would have to generate a lot of black halftone elements (a). It is easier to change the color space, since black is the basis for the RGB color space (b).

ture (`<use>`). The position and intensity of the pixel is used for the transformation, which consists of a scaling and a shift. The x- and y-offset textures are used for fine adjusting the position. To achieve the characteristic subtractive color mixing, the SVG blending mode is set to multiplication (Line 2) and applied to the respective color group. For wider support of display and editing tools the blend mode is specified using the `comp-op` attribute and Cascading Style Sheet (CSS) styling.

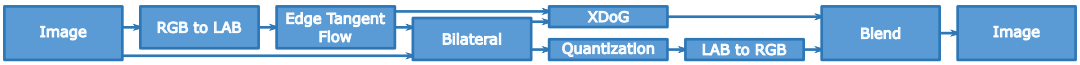
Figure 3b shows the vectorization result of a stylized half-tone image (Figure 3a) in a common way using Inkscape. For comparison, Figure 3c shows that the appearance of half-tone elements can be conveyed, yielding high-quality visual output. The common vectorization achieves only colored areas that do not represent the half-tone elements sufficiently. Using the blend mode for creating the mixed color areas instead of creating separate shapes for the overlapping areas, we can save calculation time and decrease the file size. This also allows users to modify the resulting SVG more intuitively because the shapes of the half-tone elements are still intact and can be manipulated as a whole.

```
1  <svg width="1280" height="484" ...>
2  <style>.blend{mix-blend-mode: multiply;}</style>
3  <g>
4   <g opacity="0">
5     <g id="element"> <!-- Graphical element to re-use -->
6       <path transform="..." d="m 0.0,-0.5 ..."/>
7     </g>
8   </g>
9   <!-- Cyan color group -->
10  <g fill="#0ff" comp-op="multiply" class="blend">
11    <use xlink:href="#element" transform="..."></use>
12    <use xlink:href="#element" transform="..."></use>
13    ...
14  </g>
15  <g fill="#f0f" comp-op="multiply" class="blend">...</g>
16  <g fill="#ff0" comp-op="multiply" class="blend">...</g>
17  <g fill="#000" comp-op="multiply" class="blend">...</g>
18 </g>
19 </svg>
```

Listing 1: Exemplary structure of a SVG re-using elements to represent the results of an half-tone stylization.

The usage of an instanced template decreases the overall file size for half-tone elements more complex than a circle. This also allows the user to modify the shape of all the elements at once by just editing the template. Actually instancing the group containing the shape even allows swapping the shape for an entirely new one easily. Also, the reference for all the elements does not always have to refer to the same template and thus displaying different elements at once is possible. Figure 5 shows how four templates are displayed at once, which have the four letters "W", "S", "C" and "G" defined as path. Not only the reference can be changed, also the transformation matrix. Thus each element can be rotated as desired. This has the benefit that the grid structure is not immediately visible. Figure 8a shows that the half-tone element of a heart reveals the direction of the grid through its tip, but by random rotation the grid is no longer recognizable. Furthermore, the scaling in the transformation matrix can also be adjusted. Thus, the size can be easily changed for all elements and allows the user to define the size according to his own perception.

It is also possible to make all elements the same size and allow intensity modulation through the outlines of the elements. Therefore, the fill is changed to transparent and the stroke width is scaled differently. Figure 9a shows how even a complex half-tone element can express the intensity through the line thickness and Figure 9b shows this with a circle as half-tone element. Currently, the color space CMYK is used with a subtractive blend mode. This can also be changed to additive and the color space Red-Green-Blue (RGB). For this purpose, the SVG blend mode must be changed from *multiply* to *lighten* and the shader of the rendering pass must use the color space of RGB instead of CMYK, accordingly. Also only certain color channels can be displayed. For this purpose, the grayscale rep-



(a) Raster image approach as abstract overview of the 29 rendering passes based on Semmo *et al.* [12]

(b) Our vectorized approach

Figure 11: Comparison between a raster image toon pipeline (a) and our slightly modified vectorized approach (b).

resentation of the image is used for the intensity value. There are many ways to generate a grayscale representation, e.g., using the brightness value per pixel. Figure 10 shows how the respective color components can also be displayed individually. The frequency can be adjusted by the user for the level of detail of the display. A higher value means that the grid for the color channels has a higher resolution and therefore more elements are displayed.

In addition to the many interaction possibilities, the SVG structure allows easy access to a representative element. Thus, further modifications, e.g., using JavaScript, are possible to animate the elements.

3.4 Toon Vectorization

Toon stylization [14] is a typical representative for image abstraction techniques, e.g., for expressing caricature drawings. This image style is a good candidate for vector image representation because it is composed of mostly uniform colored shapes as well as lines, which both are geometric primitives of SVG.

Effect Pipeline. The original pipeline already consists of several steps. The underlying GPU-based effect pipeline based on Semmo *et al.* [12] for toon stylization calculates two parts from an input image. On the one hand an edge detection is performed with the extended difference-of-Gaussians and on the other hand a color quantization based on local luminance in the LAB color space. These two partial results are merged again for the output (Figure 11a). In our pipeline the result of the edge detection is passed to a thinning step. The thinned edges are then used for the edge vectorization. Additionally, the quantized color data is used for an area vectorization. Instead of blending the intermediate results pixel based, the two vectorization results are merged to create the final vector image (Figure 11b).

(a) Input raster image (1280 × 1920 pixels) (b) Inkscape vectorization (11 colors) (c) Proposed approach

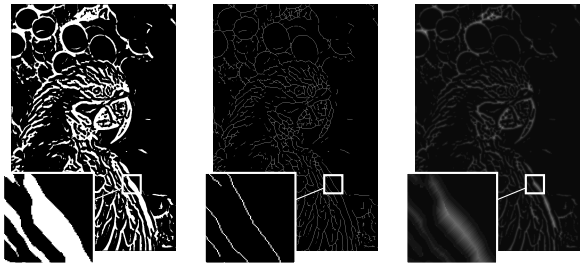
Figure 12: Comparison of the vectorized results for a toon effect (a). In method (b), the stylized image was vectorized with Inkscape. In method (c) the intermediate representation was used to generate a vectorized result, which has a typical toon color area and the edges are paths with individual line width.

Data Encoder. We use the output of the multi-pass stylization technique to generate on the one hand a stylized color quantized raster image and on the other hand an edge raster image from the input. Usually these two images would be blended on a per-pixel basis to generate the stylized output image. The edge image is additionally passed through a thinning pass to create lines of one pixel thickness.

Intermediate Representation. Two textures are used to represent the stylization outputs for subsequent vectorization: (1) a RGB texture stores quantized colored areas and (2) a luminance texture stores edge data. The edge data contains a Boolean (Figure 13b), whether an edge exists at the pixel and another value, how thick the edge would be at the location. To calculate these values the rendering pass is used, which creates the edge image for the toon stylization. This is used as the input image to perform a thinning [9, 10]. Thereby the edge widths in the image are always reduced by one pixel from out-

```
<defs>
  <inkscape:path-effect
    end_linecap_type="round"
    scale_width="1"
    miter_limit="4"
    linejoin_type="round"
    start_linecap_type="round"
    interpolator_beta="0.2"
    interpolator_type="CentripetalCatmullRom"
    sort_points="true"
    offset_points="0,9.5| 1,2.5 | 2,9.5"
    lpeversion="1"
    is_visible="true"
    id="path-effect998"
    effect="powerstroke" />
</defs>
<path
  id="CPUVectorizeEdgePassProcessor998"
  style="fill-rule:evenodd"
  inkscape:path-effect="#path-effect998"
  fill="rgb(0,0,0)"
  stroke="none"
  stroke-linecap="round"
  inkscape:original-d="M 79 9.5 Q 74.1 1.8 68.5 1 L 67 4.5"
  d="m 87.014,4.399 c 0,0 0.411,0.350 0,0 C 85.193,2.849
        81.295,0.691 77.096,-0.392 73.863,-1.227 70.964,-1.173
        68.853,-1.474 a 2.5,2.5 15.554 0 0 -2.651,1.490 c
        -0.5,1.166 -6.611,0.618 -7.934,0.742 -0.180,0.016 0,0 0,0 A
         9.5,9.5 90 0 0 75.731,8.242 c 0,0 0.112,0.142 0,0 C
        74.909,7.199 70.297,3.151 70.797,1.984 1 -2.651,1.490 c
        1.338,0.191 2.260,1.885 2.607,3.967 0.428,2.568 0.093,5.734
         0.231,7.157 0.033,0.349 0,0 0,0 A 9.5,9.5 90 0 0
        87.014,4.399 Z" />
```

Listing 2: Definition of offset points in Inkscape to apply this effect to a path via the Id.



(a) Thresholded edge-pass texture (720 × 1080 pixels) (b) Edge texture after thinning (c) Distance data used for local line thickness

Figure 13: Edge preprocessing for the toon stylization. The edge pass from the toon effect is thresholded to create a binary image (a). This is than thinned iteratively to yield the edge skeletons (b) necessary for the vectorization. During the thinning the shortest distance from outside the to the middle is determined (c).

(a) Combined color and edge layer of the raster sylization (720 × 1080 pixels) (b) Vector output with average line width per path (c) Vector output with dynamic line width along paths

Figure 14: Different line width representations. In the raster based stylization the line thickness can vary along the line (a). The SVG standard (1.1) does not include a way for varying line thickness along a path. Therefore the width information can only be used to set the thickness to e.g., the average (b). Vector graphic editors like Inkscape support dynamic stroke widths with there own extinctions and therefore allow for a higher degree of detail (c).

side to inside during an iteration until only one-pixel thick skeleton results (Figure 13b). Thus, the number of times a pixel has been deleted is saved as distance data in the resulted skeleton and this value can be used later to reconstruct the actual width of the line (Figure 13c).

Vectorizer. This consists of three components: a area vectorizer for colors, a edge vectorizer, and a merger. The area vectorizer generates colored polygons from separate quantized color-areas using a standard vectorization approach[2]. The edge vectorizer generates paths based on the edge image. This is performed by first creating paths with one node per pixel and than simplifying the path using line and curve segments where possible. The resulting nodes read the actual width from the luminance texture. This can then be used for the varying line thickness of a path. The SVG standard does not support varying line widths, but there is a proposal for this[3]. However, the most common vector graphics editors e.g., Inkscape and Adobe Illustrator support the display of this. In our approach, we have exported the results for display in Inkscape. For it, we set the path effect *powerstroke*. This requires offset-points (Listing 2), where the first value is the position on the path (e.g., 0 for the first node, 1 for the second node, etc.) and the second value is the distance from the node, the value we got during the thinning process.

Finally, the merger combines the two partial results to an SVG (Figure 12c). Figure 14b and Figure 14c show the difference and added value achieved by varying path thicknesses. This option has advantages over previous work, where only paths of constant thickness could be created.

Figure 12a shows an exemplary raster-image output of the toon stylization technique. By using the common method, both color areas and edges are represented as colorized polygons in the final vector image (Figure 12b). In contrast thereto, our approach handles colored areas and edges separately by creating line paths on top of polygons (Figure 12c), which can be edited individually. This technique is closely related to how vector artists work, namely creating different layers for different elements of the image like the line and color layer. This also allows for easy manipulation of all the lines or shapes at once (e.g., changing the thickness or color). Also by including the outlines as paths in the SVG they can be edited as such e.g., path patterns or stylizations can be applied.

4 DISCUSSION

4.1 Performance Evaluation

We tested the run-time performance of prototypical applications of our approach to two stylization techniques with respect to the following resolutions: 1280 × 720 (HD), 1920 × 1080 (FHD), 3840 × 2160 (4K), and 7680 × 4320 (8K-UHD) pixels. The performance tests were conducted using a NVIDIA GeForce GTX1080Ti GPU with 12 GB VRAM on a Intel i7 CPU with 3.7 GHz and 32 GB RAM. We used two different input images with different level of detail in the form of existing edges and number of colors (Figure 15). Measurements are averaged over 100 runs per input image (Table 1).

[2] https://github.com/jankovicsandras/imagetracerandroid/blob/master/process_overview.md

[3] https://www.w3.org/Graphics/SVG/WG/wiki/Proposals/Variable_width_stroke

(a) Low detail raster image used for performance test.

(b) Split view of toon and half-tone vector outputs of the low detail image.

(c) High detail raster image used for performance test.

(d) Split view of toon and half-tone vector outputs of the high detail image.

Figure 15: Images used for the performance test. Image (a) has very little detail. Image (c) on the other hand has a lot of details.

One can observe an increase of run-time with respect to the (1) input image resolution, (2) the complexity of the stylization techniques, (3) the number of vectorization passes, and (4) level of detail. For simple single-pass stylization techniques such as half-toning, which yield only a single vectorization pass, interactive frames rates could be achieved. Additionally, this image stylization technique is independent from the level of detail in the input image.

However, for more complex stylization techniques that yield multiple vectorization passes, the resulting performance is below interactive constraints for input resolutions greater than High Definition (HD). Furthermore, the level of detail affects the running time of the complex toon effect. An input image with more edges and colored areas leads to slower performance especially for the tracing step. Thus, this effect takes almost double the time with a more detailed image as with a less complex one. This is mainly due to the higher number of steps for thinning and also a higher quantity of path effects that have to be generated for the individual line thicknesses.

4.2 Image Stylization Representation

Our approach to use intermediate representations to create stylized vector images was successfully demonstrated using two stylization effects. Here we use the output of the existing rendering passes and extend them only with calculations like thinning. Furthermore, we enable interaction with parameters to change the vectorization. We could show that our result of the stylized vector image is more suitable for further use than a vector image that is only based on a stylized image. This is due to our representation in the SVG structure, which facilitates editing. For example, the use of cloning in the half-tone effect allows the shape of the half-tone element to be easily replaced in the final SVG. Only the one template element needs to be adjusted. Besides the exchange of elements, these can also be animated by accessing the certain SVG elements and their attributes, e.g., line thickness, displacement or rotation.

4.3 Limitations

Besides the many advantages of an SVG, the presented implementation has a few limitations.

Currently, the variable line thickness of the toon effect can only be displayed in an appropriate editor (e.g., Inkscape) that supports variable line thickness. For each effect we have considered individually how this effect can be logically broken down into vectorization steps. Now vectorization steps exist to trace edges and surfaces. There are also other pipelines for the placing of clones and the thinning step. All these vectorization steps can be reused for future effects.

Another limit is e.g., if the user edits the shape of the template object in a half-tone SVG, then all clones would be edited as well. This leads to high update times in vector image editors like Inkscape. But this is rather due to the software than the vector file.

4.4 Future Work

For future work, we plan to test our approach with additional image stylization techniques and explore possibilities of further intermediate representation variants. With respect to run-time performance improvements, a more compact intermediate representation could speed-up the vectorizer by avoiding scan-lining during tracing.

5 CONCLUSIONS

This paper shows how the extension of image vectorization by means of intermediate data can be used to increase the output quality of vector images for certain types of image stylization techniques. Our approach, which is to perform stylization as part of vectorization rather than as a pre-process, is actually close to how vector artists work, since they commonly create complex artworks as multiple layers (e.g., one layer for color regions and another layer for outlines), and fill-in regions with vector patterns. We enable the combination of fast GPU-based image stylization techniques and high-quality visual output by means of vector images. However, this requires minimal changes for encoding the output of each technique to yield the intermediate representation that drives the particular vectorization. We implemented and tested the presented

Table 1: Run-time performance comparison in total (*Total*), for the stylization (*Style*), and tracing stages (*Trace*) (in milliseconds) for input images of different spatial input resolutions (in pixels).

	Half-tone (H) [3] Single GPU-pass 1 Vectorization pass						**Toon (T) [14]** 29 GPU-passes 2 Vectorization passes					
	Low Detail Image			High Detail Image			Low Detail Image			High Detail Image		
Input Resolution	H_{Style}	H_{Trace}	H_{Total}	H_{Style}	H_{Trace}	H_{Total}	T_{Style}	T_{Trace}	T_{Total}	T_{Style}	T_{Trace}	T_{Total}
1280×720	26.5	35.0	61.5	24.7	37.7	62.4	24.8	792.2	817.0	20.0	1389.4	1409.4
1920×1080	23.6	57.2	80.8	20.6	57.3	77.9	26.5	1590.7	1617.2	30.4	2822.5	2852.9
3840×2160	33.1	154.9	188.0	34.7	147.3	182.0	37.2	6037.0	6074.2	35.3	10655.1	10690.4
7680×4320	52.3	545.9	598.2	47.6	544.1	591.7	91.0	28167.6	28258.6	102.3	53362.9	53465.2

concept using GPU-based half-tone and toon stylization techniques.

ACKNOWLEDGMENTS

We thank the reviewers for their insightful comments. The project was supported by the Federal Ministry of Education and Research (BMBF), Germany (mdViPro, 01IS18092).

REFERENCES

[1] Vyron Antoniou and Lysandros Tsoulos. Converting raster images to xml and svg. In *SVG Open*, 01 2004.

[2] Vicent Caselles, Bartomeu Coll, and Jean-Michel Morel. Topographic maps and local contrast changes in natural images. *International Journal of Computer Vision*, 33:5–27, 1999.

[3] Oliver Deussen and Tobias Isenberg. Halftoning and stippling. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, pages 45–61. Springer London, London, 2013.

[4] Tobias Dürschmid, Maximilian Söchting, Amir Semmo, Matthias Trapp, and Jürgen Döllner. Prosumerfx: Mobile design of image stylization components. In *SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications*, SA '17, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Noura Faraj, Gui-Song Xia, Julie Delon, and Yann Gousseau. A generic framework for the structured abstraction of images. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '17, New York, NY, USA, 2017. Association for Computing Machinery.

[6] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Trans. Graph.*, 36(6):180:1–180:11, November 2017.

[7] Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Trans. Graph.*, 32(1):4:1–4:11, February 2013.

[8] Sven Olsen and Bruce Gooch. Image simplification and vectorization. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 65–74, New York, NY, USA, 2011. ACM.

[9] T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, C-23(8):860–870, Aug 1974.

[10] Theo Pavlidis. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing*, 13(2):142 – 157, 1980.

[11] Philip J. Schneider. An algorithm for automatically fitting digitized curves. In Andrew S. Glassner, editor, *Graphics Gems*, pages 612–626. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[12] Amir Semmo, Tobias Dürschmid, Matthias Trapp, Mandy Klingbeil, Jürgen Döllner, and Sebastian Pasewaldt. Interactive image filtering with multiple levels-of-control on mobile devices. In *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications*, SA '16, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Trans. Graph.*, 35(4), July 2016.

[14] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Trans. Graph.*, 25(3):1221–1226, July 2006.

[15] Jun Xie, Holger Winnemöller, Wilmot Li, and Stephen Schiller. Interactive vectorization. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6695–6705, New York, NY, USA, 2017. ACM.