DOI: 10.1111/cgf.13609

COMPUTER GRAPHICS *forum*
Volume 00 (2019), number 00 pp. 1–15

LinesLab: A Flexible Low-Cost Approach for the Generation of Physical Monochrome Art

S. Stoppel and S. Bruckner

University of Bergen, Bergen, Norway
sergejsto@googlemail.com, stefan.bruckner@uib.no

Abstract

The desire for the physical generation of computer art has seen a significant body of research that has resulted in sophisticated robots and painting machines, together with specialized algorithms mimicking particular artistic techniques. The resulting setups are often expensive and complex, making them unavailable for recreational and hobbyist use. In recent years, however, a new class of affordable low-cost plotters and cutting machines has reached the market. In this paper, we present a novel system for the physical generation of line and cut-out art based on digital images, targeted at such off-the-shelf devices. Our approach uses a meta-optimization process to generate results that represent the tonal content of a digital image while conforming to the physical and mechanical constraints of home-use devices. By flexibly combining basic sets of positional and shape encodings, we are able to recreate a wide range of artistic styles. Furthermore, our system optimizes the output in terms of visual perception based on the desired viewing distance, while remaining scalable with respect to the medium size.

Keywords: paint systems, image and video processing, halftoning and dithering, image and video processing

AMS CSS: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Early after the development of the computer, artists introduced computational art as a new discipline. Mathematicians like Georg Nees [Geo] and Frieder Nake [Fri] advanced computational art by generatively creating complex geometric figures and drawing them with an axial plotter on a paper canvas. As the technology advanced, the output devices for computational art became more complex, such as industrial devices equipped with a multi-axial arm in the case of *e-David* [LPD13] or sophisticated 3D printers that output multiple layers of paint as used in *The Next Rembrandt* [Rem].

While these advances in technology are capable of generating remarkable results, they rely on complex and expensive customized hardware setups. Several manufacturers addressed the needs of hobbyists and developed affordable axial plotters for home use. Although these devices are more limited than their high-end counterparts (e.g. with respect to medium support and process feedback), they are nonetheless capable of generating high-quality results at often only a small fraction of the cost.

This new generation of hobbyist devices has given rise to a renewed interest in customized computer art for recreational purposes that increasingly penetrate the mainstream. However, there is still a lack of capable software that addresses the needs of this market, providing a wide degree of stylistic choices, while simultaneously taking into account the limitations of the available devices.

We present LinesLab, a flexible automated system that transforms images into stylized drawings or cut-outs, by creating a set of commands that can be interpreted and performed by conventional hobby plotters and cutting machines, such as Silhouette Studio [Sil] or Cricut [Cri]. The main contribution of our work does not consist of the individual styles, but of the modular automated framework for the creation of monochrome line art and paper cut-outs. Since the LinesLab system is specifically targeted for hobbyist plotter setups, we identify the limitations and capabilities of home-use plotters and analyse the design space for line art and paper cuts. Based on this constraint, we present a system that is capable of creating a wide variety of art-inspired styles while being scalable with respect to the medium size. A selection of the possible style range is shown in Figure 1. The modular framework architecture

Figure 1: *Artistic examples generated with our method. From left to right: Paper cut-out, dashes drawing, single-line spiral drawing, stippling and triangulation drawing.*

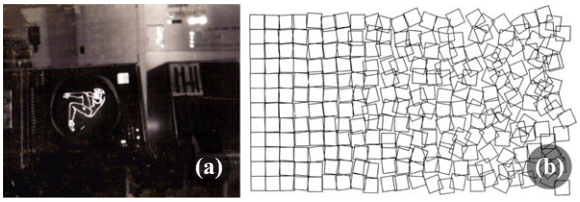

Figure 2: *(a) The pinup girl of SAGE, the first documented case of computer-generated art. (b) Schotter (1965), by Georg Nees—one of the first physical drawings made by a drawing machine.*

supports the synthesis and extension of new styles by combining sets of positional and shape encoding modules. Furthermore, being targeted at the generation of physical artwork, our approach can optimize its output based on the targeted viewing distance.

2. Related Work

The areas of computational art and computational aesthetics have been extensively studied over the years. Since we focus on physical artwork, we first cover drawing machines in art, and then continue to discuss related work on non-photorealistic rendering methods and image stylization.

Simple devices to support the drawing process were constructed already in the early 15th century. Later, automatic drawing machines that could produce complex geometric patterns with ease, such as the Harmonograph, were introduced. Jean Tinguely [Jea], for example, created a number of sophisticated drawing machines with complex repeating stroke patterns. While these machines could only generate fixed patterns, this changed with the development of computers. The roots of computer-generated art can be traced back to the late 1950s. The pin-up girl at the SAGE air defence system, as shown in Figure 2(a), was probably the first drawing to appear on a computer screen. The term computer art was first used by Edmund Berkeley in 1962, which inspired the first Computer Art Contest in 1963. This annual contest propelled the development of computer-generated art. Early pioneers of computer graphics like Georg Nees [Geo] or Frieder Nake [Fri] used plotters to create the first artworks that were constructed digitally and then physically drawn by a machine (see Figure 2b). Over the subsequent years, a number of artists used machines to create artworks, often of abstract nature.

In recent years, artists started to develop means for more realistic machine-generated paintings. Harold Cohen [Har] built sophisticated painting machines that could represent digital image content, but his main focus was still on abstract art. Pindar Van Arman [Pin] experimented with artificial intelligence based painting robots, collectively referred to as *cloudpainter*, to create original compositions. Tresset and Fol Leymarie [TFL13] described *Paul the Robot*, a robotic installation for face drawings. A more general setup was achieved with *e-David*, a feedback-guided painting robot, whose rendering techniques were discussed by Deussen *et al.* [DLPT12] and Lindemeier *et al.* [LPD13].

Galea *et al.* [GKAK16] presented a stippling method with flying quadrotor drones. Prévost *et al.* [PJJSH16] used an interactive optimization process to guide the user in the creation of large-scale paintings with spray paint. Jain *et al.* [JGKS15] developed a force-controlled robot that was able to draw on arbitrary surfaces. A similar approach was taken by Jun *et al.* [JJC*16], where a humanoid robot capable of drawing on a wall was developed. Calinon *et al.* [CEB05] used existing tools to create a humanoid robot to draw artistic portraits for entertainment purposes. Inspired by these results, many hobbyists and developers created algorithms and machines for drawing purposes. The project *Caravaggio* [Car] by Michele Della Ciana is particularly noteworthy, as it served as one of the inspirations for our work. *Caravaggio* is a custom-built polargraph drawing machine that produces artwork composed of a single continuous path. A single illustration created with this approach requires 12–24 h to complete. One of our aims was to reproduce the compelling results generated by this approach for arbitrary digital images on simple and affordable consumer devices.

Research in the areas of non-photorealistic rendering and computer-based image stylization is closely related to our work, as they typically aim to reproduce different types of artistic techniques or media. A common strategy to approximate an image is to use stroke-based rendering such as brush stroke techniques. Most of these techniques use low-level abstraction based on local image properties, such as done by Wen *et al.* [WLL*06] for colour sketch generation through a segmentation algorithm. In some approaches, the stroke placement is influenced by higher level

parameters. Shugrina *et al.* [SBC06] presented 'empathic painting', an interactive painterly rendering approach whose appearance adapts to the emotional state of the viewer in real time. A similar approach was taken by Colton *et al.* [CVP08], who developed a stroke-based rendering algorithm which heightens the emotions of the depicted person. Ostromoukhov and Hersch [OH99] used a multi-colour dithering approach to generate colour images made of artistic shapes. Many approaches convert a raster image into vector graphics, such as work done by Jeschke [Jes16], who introduced a generalized formulation for diffusion curve images. Durand *et al.* [DOM*01] introduced an interactive system that supports the user in the creation of drawings from photographs in a variety of styles.

Our work instead focuses on monochrome styles such as halftoning, stippling or hatching. Pnueli and Bruckstein [PB96] were among the first to discuss gridless halftoning techniques. Ahmed *et al.* described two line-based halftoning techniques via recursive division [Ahm14] and line amplitude modulation [AD16]. Ahmed [Ahm15] also addressed the general brightness/contrast problem of line-based halftoning methods and proposed to use error diffusion as pre-processing step. Pang *et al.* [PQW*08] presented a structure-aware halftoning technique that aims to preserve the structure and tone similarities between the original and the halftone images. Later, Chang *et al.* [CAO09] used an error-diffusion approach to create visually similar results as in the method by Pang *et al.* but with significantly decreased computation time. Pedersen and Singh [PS06] addressed the synthesis of organic maze structures, and Xu and Kaplan [XK07] discussed a set of algorithms for designing mazes based on images. Kaplan and Bosch [KB05] described how to construct drawings with a continuous line by solving the traveling salesman problem. Chiu *et al.* [CLLC15] extended the traveling salesman problem (TSP) algorithm to a tone- and feature-aware path creation method for circular scribble art. Hiller *et al.* [HHD03] discussed generalized methods for computer-generated stippling by exploring primitives other than points. Bartesaghi *et al.* [BSMG05] proposed an approach for generating hatching drawings based on multiple images with fixed positions and angles to the camera. The work of Son *et al.* [SLKL11] introduced the notion of feature-oriented structure grids for directional stippling to determine the position and orientation of rendering primitives. Xu *et al.* [XKM07] presented a technique for procedurally generated two-tone papercut designs with guaranteed connectivity. For an extensive overview of artistic stylization techniques, we refer to the state-of-the-art report by Kyprianidis *et al.* [KCWI13].

Several studies have been performed to evaluate the aesthetics of stippling algorithms compared to human artists, as well as to each other. Maciejewski *et al.* [MIA*08] compared hand-drawn and computer-generated illustrations using image processing techniques. Spicker *et al.* [SHL*17] investigated if the perceived abstraction quality of stipple illustrations is related to the number of stipples using a crowd-sourced user study. A comprehensive study of digital stippling was done by Martin *et al.* [MARI17].

While our approach is able to generate similar results to some of the work above (e.g. line-based halftoning), we specifically focus on the physical realization of the stylized image, and hence must account for technical restrictions of a plotter that were not considered in previous work. Furthermore, our system is based on a unifying framework that allows for the creation of a wide range of styles as well as the easy integration of new abstractions.

3. LinesLab System Overview

The LinesLab system was designed for users with the intent of creating computational artworks with little programming knowledge. Computational artists typically use dedicated software such as Processing [Pro] to create their works. While such tools are very flexible, they usually require the user to write the code from scratch, and offer no support in finding suitable parameters for the visual primitives. This can lead to a long trial-and-error process where the user learns how to create appealing art through experimentation. Our goal is to create a system that offers comparable flexibility for the creation of monochrome line art or cut-out styles, while simultaneously reducing the user workload of finding suitable parameters for the visual representations. Furthermore, we aim for a system that is well suited for novice users, by allowing for the synthesis of a wide range of artistic styles from a relatively small set of simple basic modules that can be easily extended by the user.

To achieve these goals, we constructed the LinesLab system as a set of exchangeable plug-ins for style selection. An overview of our system is depicted in Figure 3, with arrows annotating the information flow in the system. The orange boxes denote user input or user-defined processing steps. To create an artwork in a particular style, the user has to select the target medium, as well as the medium size and encodings for the shape and positions of the visual primitives. The LinesLab system automatically optimizes the visual primitive parameters in a nested optimization loop to create an artwork perceptually close to the input image, while following constraints set by the plotter hardware.

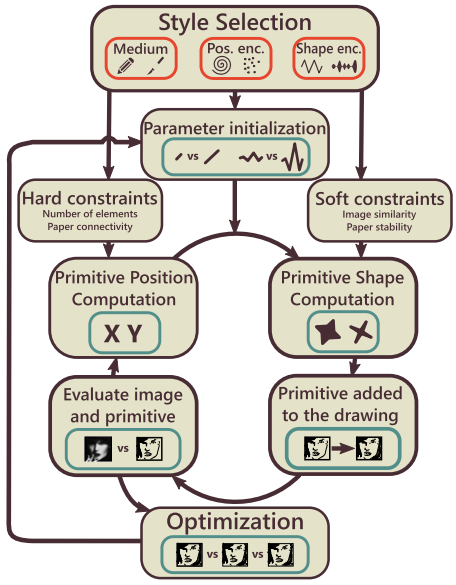

Figure 3: *Overview of our system. The user input is annotated with orange boxes, and turquoise boxes illustrate the algorithm results.*

The first step in our system is style selection. This is the only part of the system that actively involves the user. First, the user is required to select the medium (drawing or cut-out), which defines a basic set of global constraints for the optimization process. Next, the user can define the style through a choice of positional encodings, such as pre-defined positions on a regular grid or iteratively computed positions, and shape encodings, such as circles, points or lines. Entirely novel encodings can be added by creating new plug-in modules. If a new encoding is introduced, the user has to denote which parameters are to be optimized in the optimization loop and the range of the parameter space. For example, parameters that define the density of the samples or the size and shape of the primitive are natural choices for the optimization. Our system is able to handle any function as long as the output can be described by a set of curves. In addition, the user is able to modify global settings such as the paper size, viewing distance, pen tip width or maximum number of primitives. As default values, the system uses a paper size of A4, with a viewing distance of 1 m, a pen tip width of 0.5 mm and no upper limit on the number of primitives.

The optimization process then proceeds as the execution of the two nested optimization loops depicted in Figure 3. The outer loop samples the style parameters and simulates the drawing process for each of the samples. The inner loop then seeks to place primitives and adjust their shapes such that they maximize image similarity (and paper stability, in the case of cut-outs). During drawing simulation, in the first step of the loop, a suitable position for a primitive is found. In the case of pre-defined positions (e.g. primitive placement on a fixed grid), the system computes the positions only once, based on the current parameter settings. If the positions are computed iteratively, the system evaluates if the new position would violate the hard constraints, and computes a different position if it does. In the next step, the shape of a primitive is computed according to the local properties of the input image and the soft constraints. After the position and shape of a primitive have been computed, the system evaluates the resulting primitive for violations of the constraints. If the primitive fulfils all the constraints, it is added to the drawing. If an upper limit for the complexity was specified by the user, our system checks if this limit has been reached and terminates the loop accordingly. If the limit has not been reached, our system evaluates the coverage by the visual primitives. If a sufficient coverage has been achieved, the drawing is considered to be finished and the drawing simulation loop ends. The resulting drawing is then evaluated against all previously generated samples in the outer loop.

In the following sections, we discuss the concept of the visual encoding utilized in LinesLab and review the individual parts of our system in detail. We provide examples of different styles and explain their formulation as a pseudo-code in the results section.

4. Device Restrictions

Before focusing on the individual parts of the LineLab system in detail, we want to briefly outline the possibilities and limitations of a hobby plotter for visual representations, as these restrictions directly affect the design choices of our system. A hobby plotter is conceptually very similar to an *XY*-plotter with an on/off state for the *Z*-axis. In contrast to common *XY*-plotters, where the drawing instrument is moved freely in *X*- and *Y*-directions, most hobby plotters move the blade or the pen along the *X*-axis only. The processed medium is moved along the *Y*-axis. The working area of hobby plotters is typically limited to the size of approximately 30 cm × 300 cm, where 30 cm is a natural constraint by the machine dimensions and 300 cm is a common artificial constraint of the plotter software. Some techniques, such as those used for *Paul the Robot* [TFL13], utilize process feedback of the current drawing state. Because the processed medium is constantly moved in a hobby plotter, it is not easily possible to track this process with a camera and to use this information in a feedback loop. Therefore, all commands for the plotter must be pre-computed.

Figure 4: *Connected lines are processed differently by a plotter depending on the tool used. While the pen preserves the shape of the input lines (left), the blade tool rounds up the edges to ensure that the blade can rotate smoothly (right).*

Naturally, an *XY*-plotter processes its input as a collection of paths. Many hobby plotters allow for two different processing tools: a pen and a small freely rotating blade. While the path is unchanged for the pen, the transitions between individual line segments are rounded for the blade, as illustrated in Figure 4. This ensures that the blade has enough space to rotate appropriately. Many methods from computational aesthetics, such as hatching, employ varying line thickness to emphasize darker regions. Such techniques are not possible with a plotter, since a pen can only create a line with constant width. Even sophisticated pens with line thickness varying under pressure would not allow for varying lines, since the pressure cannot be changed during the drawing process. Furthermore, a hobby plotter is not able to draw filled forms. Instead, a collection of narrowly spaced lines has to be created in order to approximate filled regions. As a hobby plotter can only use one pen, or in some cases two, the plotter is not able to use a palette of colours, and only monochrome images are possible. The software used by conventional plotters is targeted at designs with limited complexity. While this is not a fundamental restriction of the hardware, for practical reasons, our system can optimize the appearance of the drawings with an upper limit on the complexity. In Table 1, we compare existing approaches against the plotter restrictions and our requirements. In addition to the restrictions in the table, the high price and limited availability of the hardware of existing custom solutions are the primary reasons for targeting conventional home-use plotters.

5. System Details

Having outlined the limitations of hobby plotters, we can discuss the functionality of our system within those limits. As noted before, our goal is a flexible system for the creation of a wide range of monochrome line art and cut-outs. To reproduce a wide variety of artistic styles, our approach aims to represent the tonal content of a digital image using a set of generalized primitives. The basic strategy for approximating the tonal content of the source image is then

Table 1: *A brief comparison of existing algorithm requirements and plotter restrictions.*

Related work	Visual feedback	Complexity	Line width	Similarity option	Scalability
E-David	Optional	No constraint as each stroke is computed separately	Varying width possible	Via visual feedback after a set of strokes	Limited
Paul the Robot	Required	No constraint the pen movement is updated on the fly	Constant line width	No tonal optimization	Limited
Conventional plotter solutions	Optional	No constraint due to successive feed	Constant line width	Varies with solution	Limited

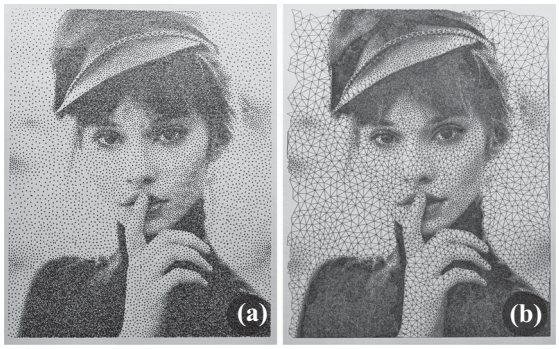

Figure 5: *Two examples of positional encoding: (a) Stippled image and (b) triangulation of the stipple positions.*

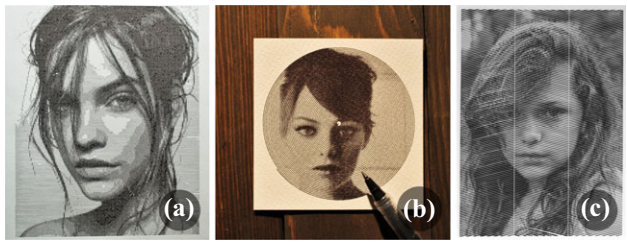

Figure 6: *Three examples of shape encodings: (a) Halftone image, (b) amplitude-modulated halftoning and (c) width-modulated halftoning realized as a paper cut-out. Note that the positions of the graphical elements are fixed. The image is encoded through shape difference only. Note that these images are best to be inspected on a computer screen.*

determined by a user-specified choice between a set of encodings that control the subsequent global optimization process.

Many visual encodings exhibit a significant degree of overlap. Grid halftone images, for example, are usually generated by placing differently sized circles on a fixed grid. Often, only the grid will vary for different styles while the encoding as circles stays fixed. To avoid repetitive definition for the visual primitives and to further reduce the encoding complexity, we split the definition of visual primitives into two parts. Conceptually, our approach is a stroke-based method, and we draw our inspiration from traditional line art. Most drawing techniques consist of two main abstractions: the position of a pen stroke and the drawn patterns. In fact, art students often train their skills by restricting themselves to one specific abstraction. Following this concept, we distinguish between two basic types of encodings for visual primitives: *positional encoding* and *shape encoding*.

Positional encoding *Positional encoding* denotes all techniques that recreate the original image with geometric shapes with varying positions, while the essential shape of the drawing primitive stays fixed and is not dependent on the image intensity values. The density of the primitives conveys the grey values of the image. A stippled image, as shown in Figure 5(a), is the most basic example of positional encoding. However, positional encoding can have more complex forms, as shown in Figure 5(b), where primitive positions are triangulated to create a new style.

Shape encoding By *shape encoding*, we denote techniques that use a direct mapping of the image intensity in a local neighbourhood to a geometric property, such as size or orientation, on a fixed position. Grid halftone images, as shown in Figure 6(a), are simple examples of such direct encodings. However, shape encodings can have various forms and the position of the encoded object does not have to be aligned to a regular grid. In Figure 6(b), the input image is encoded through an amplitude-modulated cosine function on an Archimedean spiral. Another examples for shape encodings are paper cut-out images, where the illustration is created by cutting out segments of varying thickness from the paper, as shown in Figure 6(c). The image intensity values are directly encoded as thickness of the cut-out.

Design combinations Based on these two types of encodings, we can proceed to generate new drawing styles by combining positional and shape encodings. This allows us to express a large number of different artistic styles as combinations of positional and shape encoding with varying degrees of freedom. As illustrated in Figure 7, most monochrome line art can be represented as a combination of position and shape encoding with varying degrees of freedom for each encoding. For instance, the images in Figure 5 can be seen as combinations with a high degree of freedom for the positional encoding and a low degree of freedom for the shape encoding. In contrast, the images shown in Figure 6 have a high degree of freedom in the shape encoding and a low degree of freedom in the positional encoding. Likewise, it is possible to generate styles that have a high degree of freedom in both positional and shape encoding, and their exact weighting results in countless possibilities. For example, combining positional encoding in the form of stippling and shape encoding in the form of circles sizes yields a drawing as in

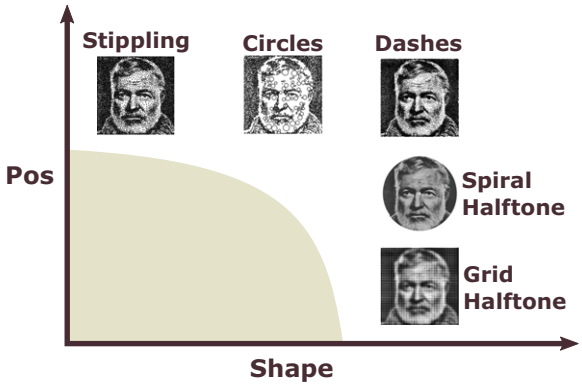

Figure 7: *Most drawing styles can be represented as a combination of positional and shape encoding with varying degrees of freedom. Designs with low degree of freedom in both encodings (grey area) are not suitable for closely representing input images.*

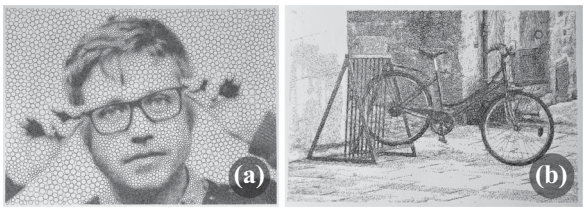

Figure 8: *Examples of a design combination: (a) The image is encoded through the position and the size of circles. (b) The image is encoded through position, orientation and length of straight lines.*

Figure 8(a). Even expressive paintings, such as the works of Vincent van Gogh, can be seen as combinations of positional encodings and shape encodings, where the position, orientation and the size of the brush stroke carry information about the colour and structure of the depicted object. Inspired by Van Gogh's style, we create monochrome drawings as depicted in Figure 8(b). In this example, the spatial encoding of stipples was combined with a shape encoding of lines, where the orientation of the line is dependent on the local variance in the input image.

5.1. Drawing simulation

As outlined before, we deconstruct the visual primitives into their positional and shape encoding. In practice, the visual primitives are computed in two stages. Our approach first generates a set of primitive positions, which are subsequently assembled into shapes. As a result, each visual primitive consists of a backbone, defined by a sampled centreline of the primitive, and the primitive shape. The backbone of each primitive is stored as a point list with arbitrary length. When multiple primitives are present, a collection of lists is utilized. Because the shape module always expects point lists as input, the system allows to combine the position and shape encodings arbitrarily. We illustrate how two different shape encodings use the same backbone in Figure 9(a). Furthermore, point lists are highly suitable for self-intersection tests and concavity estimation. In the

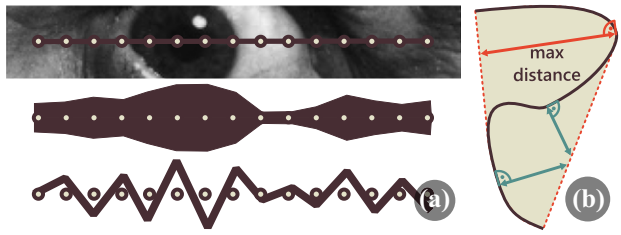

Figure 9: *(a) The primitive backbone always consists of a list of points, but the actual shape of the primitive can be defined freely, for example, as a cut-out or amplitude modulation. (b) The concavity is measured by computing the distance from the primitive backbone to its convex hull along a ray perpendicular to the backbone.*

remainder of this section, we discuss the generation strategies for the positions and shapes, respectively.

Position generation For the generation of primitive positions, our system offers two basic strategies: they can either be pre-computed or iteratively generated. For *pre-computed positions*, the user can either choose from a set of predefined functions or define a new function that covers the canvas with a set of points or curves defining the backbones of the primitives. A positional encoding module can either output a set of positions directly, or generate curves that are then automatically sampled by our system, following the restriction on maximum number of elements (if such as restriction has been specified). The final point positions are used in the shape generation either as centre points of the primitives or in a user-defined way. The positions are stored in a list or in a collection of lists. For instance, a single list of points on a spiral as the centre of amplitude modulated line would create an image, as shown in Figure 6(b), while as a collection of lists can be used for a cut-out as in Figure 16(b), where each list stores the points along the primitive centreline. Our system evaluates each list with respect to intersection and aborts the computation if the backbones of the primitives already intersect each other. This ensures paper stability for cut-outs and prevents overlapping elements during the drawing process, which lead to dark spots in the drawing. In the case of paper cut-outs, our system additionally determines how concave the backbone of the visual primitive is. A highly concave primitive can lead to unstable results that cannot be hanged freely. To measure the concavity, we compute the maximum distance between the primitive's backbone and its convex hull, as shown in Figure 9(b). If the system detects elements that exceed the degree of concavity that can be handled by the output device, they are discarded to prevent the generation of mechanically unstable results.

Many artistic techniques are best modelled with an *iterative computation of primitive positions*. This design is not only a natural choice, but also allows us to effectively simulate the drawing process on the computer and thus to optimize the drawing parameters. To allow for efficient operation while providing a simple interface, we utilize a simple strategy for modules that use iterative position computation, as illustrated in Figure 10. Essentially, we create a copy of the input image that functions as canvas. In this copy, the system finds the position of the darkest pixel, computes a primitive at that position and adds the primitive in white colour to the input

Figure 10: *An example of incremental position encoding. The system creates a copy of the input image (top left). The copy is blurred, a primitive is computed at the darkest pixel of the blurred image (top middle). A bigger primitive with the same orientation is added to the initial copy (top right). This process is iterated (lower row).*

image copy. To account for the perceptual error, the image copy is blurred with a Gaussian kernel. We discuss the kernel size computation in Section 5.2. Each computed primitive is evaluated for constraint violation, such as intersections for the paper cut-outs. If the primitive violates any constraints, it is discarded and not added to the drawing simulation. However, the centre position of the primitive is still marked in the image copy, to ensure that this position is not selected again. If the total value of the remaining darkest pixel lies above a user-specified threshold (we use the value of 0.95 as a default and for all the results presented in this paper) or the maximum number of primitives is reached, the drawing is considered to be finished and the drawing loop ends.

Shape generation The shape of the primitives can either be encoded through a set of predefined functions or via a user-defined function. Typical encodings consist of a shape variation around the generated backbones, such as a variation of circle size or amplitude variation, which directly encodes the image intensity, but it is equally possible to use derivative information, e.g. to steer the direction of strokes. In Figure 16(b), we show a typical shape encoding for a cut-out style. The cut-out is represented by a polygon that is symmetric along the backbone axis with the width encoding the grey values. In the case of cut-outs, the system also computes the distance between the visual primitives—based on a material simulation, we discard all results that have a distance below 1.4 mm between the primitives, as those are likely to tear.

5.2. Optimization

The goal of our system is the generation of line art or cut-outs from any image input and we provide a flexible plug-in-based architecture that allows for the combination of different types of positional and shape encodings. However, finding good parameters for these encodings can be a challenging task that typically results in a trial-and-error approach. While greedy algorithms for tonal reproduction can generally achieve good results for the generation of drawings based on a given set of fine-tuned input parameters, their results are heavily affected by these inputs (e.g. thresholds or parameter ranges). This task becomes even harder when the artistic style has

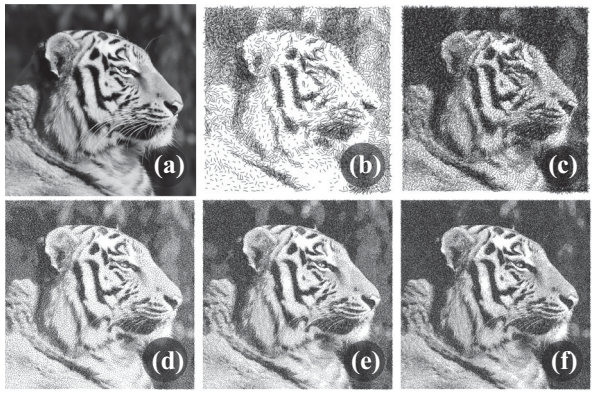

Figure 11: *Intermediate results of the optimization process for the dashes style. In this particular style, the length of the dashes and the thickness of control primitive, as shown in Figure 10, are optimized. (a) shows the input image. (b)–(f) depict intermediate results of the optimization process.*

to scale with the medium, as the parameters do not necessarily scale linearly with the medium size. Often, artists create drawings with an intended viewing distance, and following this idea, we also want to be able to globally indicate a desired distance instead of individually controlling the various parameters of the individual encodings. We therefore set up our approach as a meta-optimization procedure, where the process of creating an individual drawing is steered by a second optimization loop that seeks to optimize its parameters.

Typically, the visual primitives parameters affect one another; hence, the similarity between the input image and the simulation result tends to be highly non-convex over the parameter space. This imposes a challenge for many gradient-based algorithms, as they tend to get stuck in local minima. To overcome this challenge, we use a greedy algorithm that iteratively samples the parameter space. To reduce the number of simulations but still sample the space uniformly, we use Latin hypercube sampling in our system, and simulate the drawing for each parameter setting. In each iteration, the optimizer finds the parameter setting that generates a drawing closest to the input image and then refines the sampling in a local neighbourhood reduced by the common factor of $\sqrt{2}$ around the current best sample. Per default, we stop the refinement after $N = 8$ iterations or if

$$\frac{\delta_{n+1}}{\delta_n} > 0.99, \tag{1}$$

where δ_n is the minimal difference between the simulated image and the input image in the nth iteration. This means that we stop the optimization either after the search space was reduced to approximately 6% of the initial parameter space or the optimization improves by less than 1%. While these values can, of course, be adjusted, we found that they provide robust results and all the images in this paper were generated with these settings. In Figure 11, we show the intermediate results during an optimization process. One can see that the initial guess in Figure 11(b) does not represent the input image well. The second iteration in Figure 11(c) overcorrects the image and makes it too dark. In the following

Figure 12: *Results of drawing simulations optimized for a sheet of A3 format viewed from a distance of (a) 1 m, (b) 2 m and (c) 4 m.*

iterations in Figures 11(d)–(f), the input image is depicted more faithfully.

We compute the differences between the simulated drawing and the input image by rasterizing both and computing the normalized mean square difference between them. We choose this rather simple metric because it is less sensitive to the structural differences between the input image and the generated primitives. Furthermore, our system does not create the typical failure cases for the mean square error metric, such as transformed images or colour shift. As a drawing can only be an approximation of the input image, it is infeasible to compute the differences directly. Instead, both images are scaled to be of the same size and filtered with a Gaussian filter. By adapting the kernel size of the Gaussian filter, we can optimize the similarity of the drawing and the input image based on the desired viewing distance. The visual acuity of a normally sighted person is defined through a visual angle of 1 arc minute[CK15]. This means that a person with normal sight would be able to identify an isolated object of the size of approximately 1 mm from a distance of 3 m. However, this is the smallest object an average human could recognize with the naked eye. A study performed by Crété-Roffet *et al.* [CRDLN07] indicates that humans hardly perceive blur generated with a Gaussian kernel with the size of 1 mm from a 1 m distance. Using this information, we can optimize the similarity of the drawings dependent on the distance by using an appropriate kernel size. More precisely, we can directly compute the kernel size k of a Gaussian filter to optimize the visual similarity from a fixed distance with the following formula:

$$k = ceil\left(\frac{h_I(px)}{h_P(mm)} \cdot d_{P,V}(m)\right), \tag{2}$$

where h_I is the height of the scaled input image and simulated image in pixels, h_P is the height of the physical drawing in millimetres and $d_{P,V}$ is the distance between the viewer and the physical drawing in metres. Using this formula, we compute the appropriate kernels for drawings of size A4 and A3, for an image scaled to 1500 px height during the computation, viewed from 1 m distance as 5 and 7, respectively. In Figure 12, we compare the results of our simulation for a drawing of size A3 optimized for the viewing distance of 1, 2 and 4 m, respectively. In Figure 13, we show close-ups of the lower right corner of these drawings. One can notice that high-frequency details become less and less preserved as the distance increases.

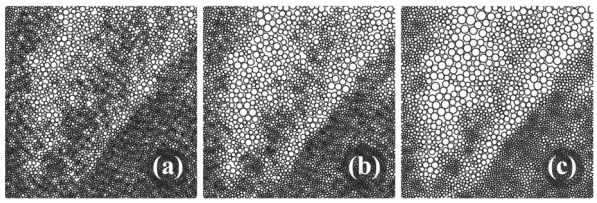

Figure 13: *Close-up on an image depicting drawing of a fabric. The drawing was optimized for a viewing distance of (a) 1 m, (b) 2 m and (c) 4 m. One can see that the high-frequency changes disappear with the distance.*

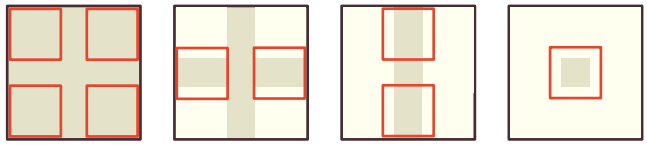

Figure 14: *We parallelize the primitive generation by subdividing the image into a set of tiles, where the primitives are computed.*

6. Implementation and Performance

Our system was implemented in Python 3.6 using CUDA, OpenCV and DXFWrite. CUDA was used to parallelize the drawing simulation, OpenCV is a library mainly aimed at real-time computer vision that was used for efficient image operations and DXFWrite was used to export the drawing paths as dxf files that can be read by conventional plotter software. We minimize the computation time by computing the primitives in parallel on the graphics processing unit (GPU). For shape encodings where the positions of all primitives are fixed, we compute the shape of all primitives in parallel. Hence, the computation time for purely shape-encoded styles lies within a few milliseconds. To reduce the synchronization overhead required for the parallelization of iterative positional encodings, we subdivide the image into overlapping tiles (Figure 14) which are then computed in parallel. This process is performed in the following manner: First, we subdivide the image into tiles that have at least a pairwise distance of twice the size of the largest primitive. This ensures that no primitives are in conflict (Figure 14, left). Then, the tiles are shifted as shown in the subsequent images in Figure 14 and the computation is repeated in each step.

The subdivision into tiles not only allows for parallel computation, but also results in an overall speedup as only a limited area has to be evaluated. This simple strategy leads to a performance increase by a factor of approximately 380 compared to the naive approach. In our experiments, only the circle and the single-line style exceeded a computation time of 2 s due to their serial structure. On average, the remaining styles take around 0.5–1 s to generate an image. Table 2 lists the computation times for the results presented in the paper. For completeness, we also include the production times as measured on a Silhouette Studio plotter.

Because the primitive positions are computed by evaluating the image row by row, the primitives tend to accumulate at the right and the lower border of the tiles. This can potentially introduce undesired artefacts. To minimize these artefacts, the computation is performed

Table 2: *A brief comparison of existing algorithm requirements and the plotter restrictions.*

Style	Figure	Computation time	Production time
Stippling	5(a)	0.67 s	96 min
Triangulation	5(b)	0.72 s	84 min
Spiral	6(b)	0.01 s	47 min
Cut-out	16(b)	0.09 s	56 min
Dashes	17	1.70 s	92 min
Circles	18(b)	4.76 s	87 min

Figure 15: *A comparison of a drawing computed as whole (a) and with tiles (b). Illustration (a) took 741 s to compute, while (b) took 1.43 s to compute. Qualitatively, the results are almost undistinguished.*

for a small range of grey values, i.e. we subdivide the computation into n steps and in the kth step compute the primitives up to $\frac{k}{n}$ image intensity. This leads to much more evenly distributed primitives as the primitives of the neighbouring tiles cover parts of the right and lower tiles borders in each iteration. In practice, we found that $n = 5$ provides results with no visible artefacts. In Figure 15, we show a comparison between an illustration computed as whole (a) and one computed with tile subdivision (b). The illustration in Figure 15(a) took 741 s to compute, while the one in (b) took 1.43 s. The resulting images are indistinguishable to the naked eye.

7. Results

To generate a drawing, the user has to select a module for the positional and shape encoding. Conceptually, these modules are realized as functions in a catalogue, to which the users can add own functions as well. To define a style, the user needs to select at least one positional and one shape encoding. Currently, the LinesLab system provides 11 different positional encodings, such as fixed patterns or iterative encodings for the stippling and dashed style, and 17 shape encodings, such amplitude modulation or triangulation of points. If desired, the user can add further encodings by adding the function calls for positional and shape encodings. We provide pseudo-code of the used modules to illustrate their functionality for the following examples. When two modules are selected, our system loads two stored lists with the upper and lower bounds of the used parameters. Therefore, if users create a new module, they also need to define a possible range for the used parameters. However, the parameter ranges can also be overridden prior to the optimization process. This information is then sent to the optimizer function, which samples the parameters and calls the drawing simulation function for each

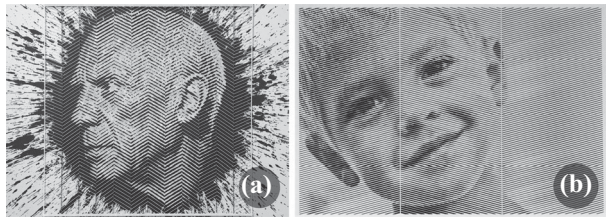

Figure 16: *(a) Cut-out portrait by Yoo Hyun. (b) Cut-out generated with our system.*

parameter setup. The drawing simulation function calls the positional encoding and shape encoding functions. After each primitive computation, LinesLab evaluates the result for constraint violation. To generate the upcoming result, the setup of our system requires the following Python code:

```
img = cv2.imread('input.jpg')
drawing = dxf.drawing('output.dxf')

min_dimension = 1200.0 # set canvas size

# set tonal coverage threshold
# for iterative generation
max_darkness_threshold = 0.95

# define tile_size for parallelization
tile_size = 200

# define limit of primitives, set to inf if not
  given num_prim = 10000

#test for intersection concavity
intersec = True
concavity = False

# Add positional and shape encodings.
pos_enc = ["hering_bone"]
shape_enc = ["cut_out"]

# Define upper and lower bounds of the
# optimized parameters.
# When left empty a preset is loaded.
UB = []
LB = []

LinesLab_opt(pos_enc,shape_enc,UB,LB,intersec,
  concavity)
```

Paper cut-outs The Korean artist Yoo Hyun[Yoo] creates breathtaking portraits of celebrities by carefully slicing away thin slivers of paper. In this way, he forms a woven zigzag pattern contained inside a solid frame, as shown in Figure 16(a). With our system, we are able to recreate the same style by choosing a similar positional encoding as Hyun, a regular tiled grid, and a shape encoding that translates the darkness values into line thickness. The pseudo-code of the encodings is shown in Algorithms 1 and 2.

Algorithm 1. Positional encoding for the cut-out style

```
Cutout positional encoding
input : number of divisions: nx, number of cut lines per
        division: ny, angle of the cutout: α
for cx in range (0, nx) do
    for cy in range (0, ny) do
        compute the start and endpoints pS, pE for
         alternating primitive backbones
    end
    primitives.add([pS, pE])
end
return primitives
```

Algorithm 2. Shape encoding for the cut-out style

```
Cutout shape encoding
input : thickness of the cutout: th, sampling distance: d,
        primitive backbone containing pS and pE: p
compute primitive length pL
ns = pL/d
for s in range (0, ns) do
    pos = pE * s/ns + pS * (ns-s)/ns
    br = (1-img(pos))
    primitiveThickness = br * th
end
```

The positional encoding essentially creates a herringbone pattern. LinesLab automatically optimizes the encoding parameters to create a result with the highest tonal similarity to the input image while ensuring paper stability by restricting the distance between the cut-outs to at least 1.4 mm. The result of our cut-out is shown in Figure 16(b). In contrast to Yoo Hyun, our system leaves small strips of paper between the segments to ensure the stability of the paper while cutting. Hyun spends tens of hours creating such portraits. Using our system, we are able to create the cut-out in just over 45 min (including the production time).

Dashes style Inspired by Van Gogh, we created an image composed of short dashes in the direction of lowest variance. In this style, the positional encoding is performed iteratively as described in Section 5.1 and can be formulated as shown in Algorithm 3.

Algorithm 3. Pseudo-code of positional encoding for the dashes style

```
Dashes positional encoding
input : input image with all existing primitives marked
        white: img
p = minLoc(img)
mark primitive in img
return p
```

The shape encoding is done by sampling a line at each generated position and choosing the direction of the sample with the lowest variance along the line. We describe the shape computation with the pseudo-code shown in Algorithm 4. For this style, only the maximum length of the primitives was declared as an optimization parameter and is determined in the outer optimization loop. For instance, for an image of size A4 drawn with a pen of 0.5 mm thickness and a viewing distance of 1 m, the resulting value of the maximal line length is 4 mm. Three results generated in this style are shown in Figure 17. The respective photographs were taken at the optimized distance for each image. The leftmost drawing has a size of 3.5×5 cm and is optimized for a viewing distance of 20 cm, the middle image is 21×29.7 cm big and is optimized for a viewing distance of 120 cm, and the rightmost image measures 29.7×42 cm and is optimized for a viewing distance of 170 cm.

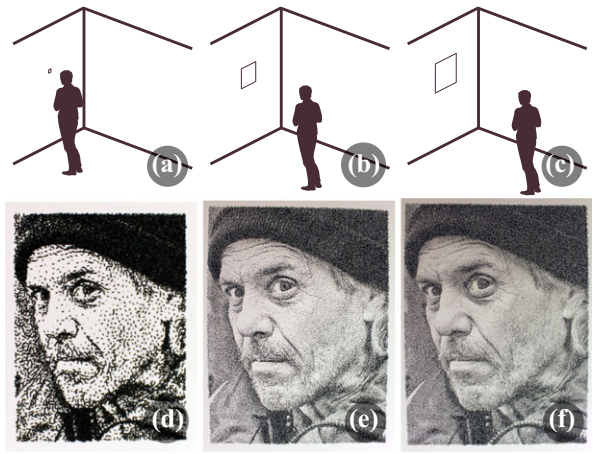

Figure 17: *Examples of a drawing on different medium sizes and viewing distances. The first row shows the medium size and viewing distance in proportion to an 1.8 m high man, while the lower row depicts photographs of the corresponding drawings. The drawing in (d) is 35×50 mm big, while the drawings in (e) and (f) have the size 210 × 297 mm and 297 × 420 mm, respectively.*

Algorithm 4. Shape encoding for the dashes style. For each primitive, the system finds the direction of the lowest variance around that point and draws a line in that direction encoding the image brightness with the line length

```
Dashes shape encoding
input : input image: img, primitive pos: p, maximal
        primitve length: mL, number of radial samples: ns
lowVar = +inf
for s in range (0, ns) do
    a = s * pi/ns
    pS = [p.x + mL/2 * sin(a),p.y + mL/2 * cos(a)]
    pE = [p.x - mL/2 * sin(a),p.y - mL/2 * cos(a)]
    var = measure variance in img from pS to pE
    if var <lowVar then
        lowVar = var
        aa = a
    end
end
pL = (1-img[p]) * mL
pS = [p.x + pL/2 * sin(aa),p.y + pL/2 * cos(aa)]
pE = [p.x - pL/2 * sin(aa),p.y - pL/2 * cos(aa)]
return (pS,pE)
```

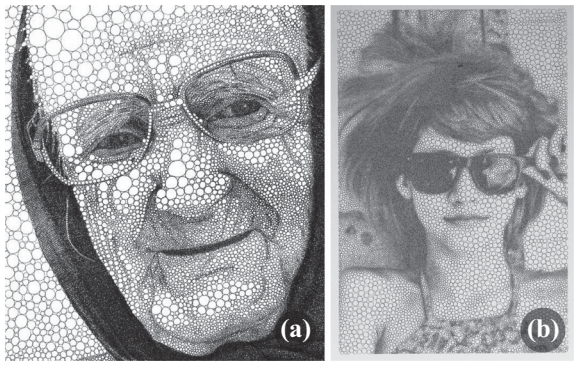

Figure 18: *(a) Circles drawing by Màrton Jancsò and (b) drawing generated by our system.*

Figure 19: *Crosshatching drawing made with the LinesLab system. (a) The illustration is drawn on paper. (b) The same illustration etched on Medium-density fibreboard (MDF) with a laser cutter.*

Circle style Many artists use familiar shapes to create complex portraits. Màrton Jancsò, for example, created a portrait composed of only circles, as shown in Figure 18(a). Our system is able to recreate this style with the above-mentioned iteratively computed positional encoding in Algorithm 3 with circles as primitives. The marked circles are computed using the image brightness and a minimum and maximum radius as input parameters. Our system stores the centre points of the circles as the primitive backbones. Next, to draw the circles in a space filling manner, the backbones are send to a build in function that computes a Voronoi diagram of the primitive positions and finds the largest fitting circle for each cell. A drawing made by our system is depicted in Figure 18(b). As can be seen, our approach tends to leave blank spaces between the circles in very bright areas and at areas close to the border. This is due to the elongated shape of the Voronoi cells that occurs more often at the image border.

Crosshatching style Creating drawings with crosshatching techniques originated in the middle ages and was perfected by Albrecht Dürer. It is still a popular technique for artists in modern days. The main concept of crosshatching is that the quantity, thickness and spacing of lines depicts the brightness of the image. As mentioned in Section 4, we cannot change the line thickness during the drawing process. Therefore, to recreate a convincing crosshatching style, only the line quantity and spacing can be adjusted in LinesLab. A crosshatching drawing made using our system can be seen in Figure 19(a). This illustration was created with two function calls for the positional encoding and two function calls for the shape encoding. The shape encoding function consisted of straight lines connecting the backbones. The first positional encoding function computes the prominent contours in the image with a Canny edge detection algorithm, which is part of our function library. In the second positional encoding, the image is quantized to five intensity levels. For each of the levels, the algorithm then fills in the corresponding areas with crosshatching lines. To create a natural appearance, we add noise to the start- and endpoints of each line. The system then optimizes the length of the lines, the distance between the lines for each tone and the noise level. The pseudo-code for the positional encoding is shown in Algorithm 5. Our system can be directly applied to other media as well. In Figure 19(b), we show the same illustration engraved with a laser cutter on a medium-density fibreboard.

Algorithm 5. Positional encoding for the crosshatching style

```
Crosshatching positional encoding
input  : input image: img, line angle: a, distance between
         lines: d, maximal line length: l
postImages = posterize(img,5) // returns 5 binary images
  encoding the posterized are as 1 and background as 0
canStartLine = True
canEndLine = False
for pImg in postImages: do
    for x in range (0,img.w) do
        for y in range (0,img.h) do
            if canStartLine and pImg(x,y) == 1 then
                begin line
                canStartLine = False
                canEndLine = True
            end
            if canEndLine and (pImg(x,y) == 0 or
              lineLenght > l) then
                end line
                draw line
                canStartLine = True
                canEndLine = False
            end
        end
    end
end
```

Single-line style Artwork created with a single line ranges from abstract minimalist drawings to highly detailed portraits. We can realize a single-line style in our system by using the same positional encoding as for the stippling style (Algorithm 3). Then, in the shape encoding, we choose a random point and declare this point as the line ending. We compute the five nearest points to the endpoint and randomly choose one as the next point. This process is repeated until all points have been processed and the line can be drawn. We show a result of this style in Figure 20. The only optimized parameter for this style is the size of the marker in the positional encoding. We increase the performance of finding the closest primitives by considering only primitives in the adjacent cells. Due to the iterative structure, this style is rather slow to compute and one simulation for a drawing on a A3 canvas can take several minutes. Therefore, the optimization process for this style was performed overnight. However, once the optimization process is performed, the

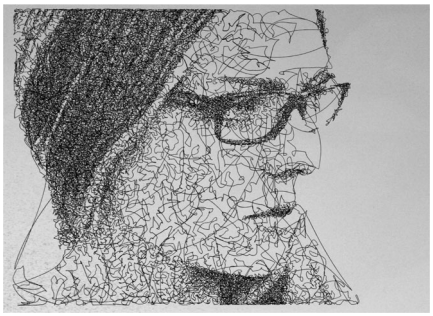

Figure 20: *Single-line drawing, created by connecting the positions of a stippled image with a single line.*

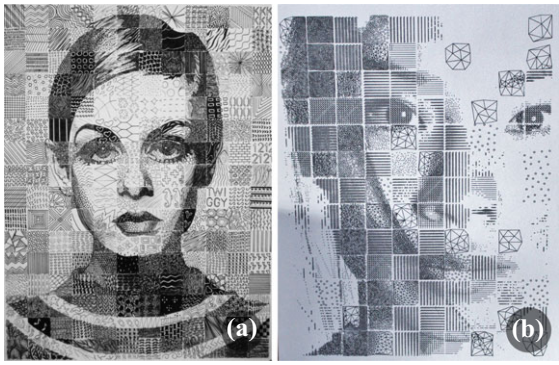

Figure 21: *(a) Grid portrait by Katherine Cahoon. (b) Grid portrait generated with our system.*

parameter settings can be stored and retrieved again for a different drawing.

Mosaic style A popular task for art students is to create portraits on a regular grid. The portrait by Katherine Cahoon in Figure 21(a) is an example of this style. With our method, we are able to create such a grid portrait by simply subdividing the input image into regular tiles and processing the tiles individually with a random style. The result of this approach can be seen in Figure 21.

8. User Feedback

To gain feedback on the usability and utility of our approach, we presented LinesLab to two computational artists with different levels of programming experience. Both artists use Processing to create their artworks, but the first artist has used Processing for several years, while the second user has only experimented with it for three months. We explained LinesLab to them and let them freely explore the system with the possibility to ask questions at any point. After the exploration phase, we asked both artists to add a new style to the system. Finally, we asked them to comment on LinesLab and its usability.

Experience with programming languages was beneficial for the first artist in terms of required learning time and in implementing new encodings. The first artist spent 34 min in the exploration phase, while the second artist needed 78 min to fully understand

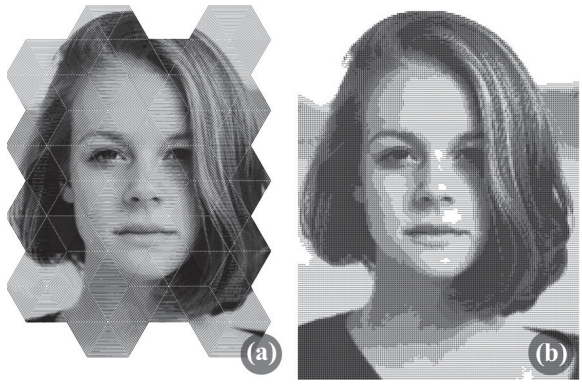

Figure 22: *(a) A new hexagonal style created by an artist with long strong programming skills. (b) A halftone style create by an artist with limited programming experience.*

the entire functionality of the system. During the exploration phase, both artists simulated different styles by exchanging the positional and shape encodings, while investigating the code of the used encodings. Both artists were able to create a new style in our system. However, the first artist was able to implement a new style within just under 20 min, while the artist with less programming experience needed 37 min. We show the results of the two styles in Figure 22. In Figure 22(a), the artist added a new positional encoding in form of lines aligned in a hexagonal pattern, which were superimposed with the existing amplitude modulation function. LinesLab automatically optimizes the spacing inside the hexagon as well as the sampling rate and the amplitude of the modulation function. In Figure 22(b), the artist used an existing positional encoding (regular grid), and then created a star pattern for each primitive. For this style, our system optimizes the spacing between the primitives and the length of the drawn line in the star pattern.

After the hands-on phase, we asked the two artists to comment on the usability of our system, features they liked and disliked, and if they would consider using our system in the future. With respect to usability, both artists said that the functionality of the system can be understood fairly easily through experimentation with the different encoding functions. When asked to comment on features they liked, the first artist replied: 'I especially like the separation between shape and positional encodings. This really makes it easy for me to experiment with different style combinations. Furthermore, it forces me to mentally decompose my ideas for a style, which makes it easier to translate them into code later on'. The second artist noted: 'The automatic parameter tuning is really helpful. I would still change the parameters a bit to get the result I had in mind, but the automatic tuning drastically reduces the testing time'. When asked about any annoyances or features of dislike, the second artist replied: 'It confused me that some styles were computed almost instantly while some took a bit longer. Maybe having a progress bar would be good'. Both artists mentioned that a drag and drop interface for the encodings would make the exploration of the system easier. Nonetheless, both artists said they would appreciate to use our tool in the future. The first artist said: 'I would absolutely use this tool, but mostly to create a basis on which I can work in other programs such as Illustrator or Inkscape'. The second artist replied: 'Yes, I would use this tool. I think it would save me a lot of time.

For me it is much faster to create good results in this tool than in Processing, because I can easily reuse old styles and I get to see good results without tuning all the parameters'.

9. Discussion

The creation of physical drawings or cut-outs can have unexpected and unwanted artefacts that we normally do not encounter when only dealing with digital images. For instance, the pen may not distribute the ink evenly resulting in too light or too dark areas. Similarly, a blade can wear off and tear the paper instead of producing a clean cut. In our experience, such issues can be minimized by reducing the speed of the plotter, allowing for a more even distribution of ink and lower the forces on the paper. Another way to avoid excessive ink leakage of a pen is to use strongly absorbing paper, such as Aquarelle paper.

To ensure paper stability, we simulated the stress of a paper sheet during the cutting process and the deformation of the paper after the cutting and incorporated the results as global parameters of our system. The simulation was performed in ANSYS, where the paper was approximated as a thin sheet of custom material based on wood with its tensile strength and density changed to the values of paper. The result of the simulation was that the stress during cutting is negligible, if the cutting paths have at least 1.4 mm distance between them. Furthermore, we simulated the deformation of hanging paper cut-outs fixed on the upper corners. The simulation showed that the deformation is minimized when the cutting pattern follows vertical lines or lies within an angle of 14 and 46 degrees. While such a simulation can only provide guidelines for the cutting process, we still see such results as valuable for the design of our system and hope to incorporate further results in the future. In fact, we could confirm that the paper often tore when the distance between cutting paths fell below 1.4 mm.

At present, we use a rather simple optimization process of refining around the best result of the current iteration. While this approach has proven to be effective, more complex optimizers could be easily integrated, as our system is completely modular in this respect. While we deliberately targeted off-the-shelf hobbyist plotting and cutting devices in the development of our system, there are no fundamental restrictions that would prevent its use for professional devices. In fact, we tested our system on industrial laser cutters, as can be seen in Figure 19. In the future, we plan to experiment with professional vinyl cutters that can be used for large-scale drawings with only moderate expenses. The current implementation of LineLab provides a function-based interface. We plan to extend the usability by providing a simple drag and drop functionality to further simplify experimentation. Currently, if a user aims to add a new style to the system, he needs to define the corresponding positional and shape encoding functions. Inspired by the advances of deep neural networks, we see a promising direction for future work in automatic style extraction from exemplar images.

Despite its modularity and the simplicity of our optimization process, our system is still able to closely recreate the results of specialized approaches. For instance, we were able to create similar results to Chiu *et al.* [CLLC15], by computing the positions as stippling points first and constructing a backbone path through them with a TSP solver. The shape encoding was done through a trochoid

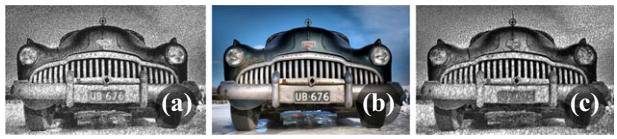

Figure 23: *(a) Scribble art generated by a method proposed by Chiu et al., (b) input image and (c) scribble art generated with our system.*

function, which is a curve defined by a fixed point on a rotating circle travelling along a line. The optimized parameters were the mapping parameters that computed the rotation speed and circle radius from the local image intensity. The comparison of our result and the result by Chiu *et al.* can be seen in Figure 23. Clearly, our formulation cannot exactly reproduce the tonal quality of a highly specialized approach. However, this example shows the strength of our system in being able to produce high-quality results with a much simpler setup.

10. Conclusion

We presented a novel system capable of creating line art and cut-outs with conventional hobby plotters and cutting machines. Our approach is flexible and modular, supporting a wide range of artistic styles generated from a basic set of position and shape encodings. By employing a nested optimization process, we are able to automatically generate artistic results that conform to the physical and mechanical constraints of the output devices. To the best of our knowledge, our system is the first one that optimizes the generated drawings based on the distance of the viewer to the artwork itself, while maintaining scalability with respect to the medium size. We achieve results comparable to existing works of art that can be easily reproduced with off-the-shelf hardware. Furthermore, our architecture features a simple plug-in mechanism, which allows the easy addition of new styles.

Acknowledgements

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway.

References

[AD16] Ahmed A. G. M., Deussen O.: Amplitude modulated line-based halftoning. In *Proceedings of Computer Graphics and Visual Computing* (2016), pp. 41–43.

[Ahm14] Ahmed A. G. M.: Modular line-based halftoning via recursive division. In *Proceedings of Workshop on Non-Photorealistic Animation and Rendering* (2014), pp. 41–48.

[Ahm15] Ahmed A. G. M.: From stippling to scribbling. In *Proceedings of Bridges: Mathematics, Music, Art, Architecture, Culture* (2015), pp. 267–274.

[BSMG05] Bartesaghi A., Sapiro G., Malzbender T., Gelb D.: Three-dimensional shape rendering from multiple images. *Graphical Models 67*, 4 (2005), 332–346.

[CAO09] Chang J., Alain B., Ostromoukhov V.: Structure-aware error diffusion. In *SIGGRAPH Asia* (2009), pp. 162:1–162:8.

[Car] Caravaggio. http://caravaggio-a-drawing-machine.webnode.it/. Accessed: 27 July 2017.

[CEB05] Calinon S., Epiney J., Billard A.: A humanoid robot drawing human portraits. In *Proceedings of International Conference on Humanoid Robots* (2005), pp. 161–166.

[CK15] Carlson N. B., Kurtz D.: *Clinical Procedures for Ocular Examination*. McGraw-Hill Education, New York, 2015.

[CLLC15] Chiu C.-C., Lo Y.-H., Lee R.-R., Chu H.-K.: Tone- and feature-aware circular scribble art. In *Proceedings of Pacific Graphics* (2015), vol. 34, pp. 225–234.

[CRDLN07] Crété-Roffet F., Dolmiere T., Ladret P., Nicolas M.: The blur effect: Perception and estimation with a new no-reference perceptual blur metric. In *Proceedings of Electronic Imaging Symposium* (2007), pp. 6492–6503.

[Cri] Cricut. https://home.cricut.com/. Accessed: 22 May 2017.

[CVP08] Colton S., Valstar M. F., Pantic M.: Emotionally aware automated portrait painting. In *Proceedings of Digital Interactive Media in Entertainment and Arts* (2008), pp. 304–311.

[DLPT12] Deussen O., Lindemeier T., Pirk S., Tautzenberger M.: Feedback-guided stroke placement for a painting machine. In *Proceedings of Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (2012), pp. 25–33.

[DOM*01] Durand F., Ostromoukhov V., Miller M., Duranleau F., Dorsey J.: Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proceedings of the 12th Eurographics Conference on Rendering* (2001), pp. 71–82.

[Fri] Frieder Nake. https://en.wikipedia.org/wiki/Frieder_Nake. Accessed: 22 July 2017.

[Geo] Georg Nees. https://en.wikipedia.org/wiki/Georg_Nees. Accessed: 22 July 2017.

[GKAK16] Galea B., Kia E., Aird N., Kry P. G.: Stippling with aerial robots. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering* (2016), pp. 125–134.

[Har] Harold Cohen Aaron. http://www.aaronshome.com/aaron/aaron/index.html. Accessed: 22 July 2017.

[HHD03] Hiller S., Hellwig H., Deussen O.: Beyond stippling - Methods for distributing objects on the plane. *Computer Graphics Forum 23*, 3 (2003), 515–522.

[Jea] Jean Tinguely. https://en.wikipedia.org/wiki/Jean_Tinguely. Accessed: 22 July 2017.

[Jes16] Jeschke S.: Generalized diffusion curves: An improved vector representation for smooth-shaded images. *Computer Graphics Forum 35*, 2 (2016), 71–79.

[JGKS15] Jain S., Gupta P., Kumar V., Sharma K.: A force-controlled portrait drawing robot. In *Proceedings of Industrial Technology* (2015), pp. 3160–3165.

[JJC*16] Jun Y., Jang G., Cho B., Trubatch J., Kim I., Seo S., Oh P. Y.: A humanoid doing an artistic work - Graffiti on the wall. In *Proceedings of Intelligent Robots and Systems* (2016), pp. 1538–1543.

[KB05] Kaplan C. S., Bosch R.: Tsp art. In *Renaissance Banff: Mathematics, Music, Art, Culture* (2005), pp. 301–308.

[KCWI13] Kyprianidis J. E., Collomosse J., Wang T., Isenberg T.: State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics 19*, 5 (2013), 866–885.

[LPD13] Lindemeier T., Pirk S., Deussen O.: Image stylization with a painting machine using semantic hints. *Computer Graphics 37*, 5 (2013), 293–301.

[MARI17] Martín D., Arroyo G., Rodríguez A., Isenberg T.: A survey of digital stippling. *Computers & Graphics 67* (2017), 24–44.

[MIA*08] Maciejewski R., Isenberg T., Andrews W., Ebert D., Sousa M., Chen W.: Measuring stipple aesthetics in hand-drawn and computer-generated images. *IEEE Computer Graphics and Applications 28*, 2 (2008), 62–74.

[OH99] Ostromoukhov V., Hersch R. D.: Multi-color and artistic dithering. In *ACM SIGGRAPH* (1999), pp. 425–432.

[PB96] Pnueli Y., Bruckstein A. M.: Gridless halftoning: A reincarnation of the old method. *Graphical Models and Image Processing 58*, 1 (1996), 38–64.

[Pin] Pindar Van Arman. http://www.cloudpainter.com/. Accessed: 22 July 2017.

[PJJSH16] Prévost R., Jacobson A., Jarosz W., Sorkine-Hornung O.: Large-scale painting of photographs by interactive optimization. *Computers & Graphics 55* (2016), 108–117.

[PQW*08] Pang W.-M., Qu Y., Wong T.-T., Cohen-Or D., Heng P.-A.: Structure-aware halftoning. *ACM Transactions on Graphics 27*, 3 (2008), 1–8.

[Pro] Processing. https://processing.org/. Accessed: 20 February 2018.

[PS06] Pedersen H., Singh K.: Organic labyrinths and mazes. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering* (2006), pp. 79–86.

[Rem] The next Rembrandt, can the great master be brought back to create one more painting? https://www.nextrembrandt.com/. Accessed: 22 May 2017.

[SBC06] Shugrina M., Betke M., Collomosse J.: Empathic painting: Interactive stylization through observed emotional state. In *Proceedings of Symposium on Non-Photorealistic Animation and Rendering* (2006), pp. 87–96.

[SHL*17] Spicker M., Hahn F., Lindemeier T., Saupe D., Deussen O.: Quantifying visual abstraction for stipple drawings. In *Proceedings of NPAR* (2017), pp. 1–10.

[Sil] Silhouette America. https://www.silhouetteamerica.com/. Accessed: 22 May 2017.

[SLKL11] Son M., Lee Y., Kang H., Lee S.: Structure grid for directional stippling. *Graphical Models 73*, 3 (2011), 74–87.

[TFL13] Tresset P., Fol Leymarie F.: Portrait drawing by Paul the robot. *Computer Graphics 37*, 5 (2013), 348–363.

[WLL*06] Wen F., Luan Q., Liang L., Xu Y.-Q., Shum H.-Y.: Color sketch generation. In *Proceedings of Symposium on Non-Photorealistic Animation and Rendering* (2006), pp. 47–54.

[XK07] Xu J., Kaplan C. S.: Image-guided maze construction. In *ACM SIGGRAPH* (2007).

[XKM07] Xu J., Kaplan C. S., Mi X.: Computer-generated papercutting. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (2007), pp. 343–350.

[Yoo] Hyun Yoo. https://www.instagram.com/yoo.hyun/. Accessed: 22 August 2017.