# Image Stylization with a Painting Machine Using Semantic Hints

Thomas Lindemeier[a], Sören Pirk[a], Oliver Deussen[a]

[a]*Department of Computer and Information Science, University of Konstanz, Germany*

**Abstract**

In this paper we present and evaluate painterly rendering techniques that work within a visual feedback loop of eDavid, our painting robot. The machine aims at simulating the human painting process. Two such methods and their semantics-driven combination are compared for different objects. One uses a predefined set of stroke candidates, the other creates strokes directly using line integral convolution. The aesthetics of these methods are discussed and results are shown.

*Keywords:* Non-photorealistic rendering, painterly rendering, painting machine, visual feedback, image stylization

## 1. Introduction

In this paper, we present our painting machine, eDavid, which was built two years ago for creating artistic depiction with visual feedback. Using eDavid, we can obtain a wide variety of painting styles. In this paper, we compare and evaluate two of these styles based on their aesthetic properties.

eDavid is a one-arm industry robot that we modified and equipped for painting purposes (see [1] and figure 1). A camera observes the canvas, their images are compared with a given target function (usually an image, but could be a more complex representation such as a 3-d scene). New strokes are generated until the target function is sufficiently approximated by the paint on the canvas. Our rationale of building an industrial robot instead of using a pen plotter comes from the fact that with an industrial robot we are not restricted to use only one type of physical instrument to create paintings. Our current implementation of eDavid can successfully use any brush or pencil. It can paint with a number of physical colors ranging from ink to oil paint.

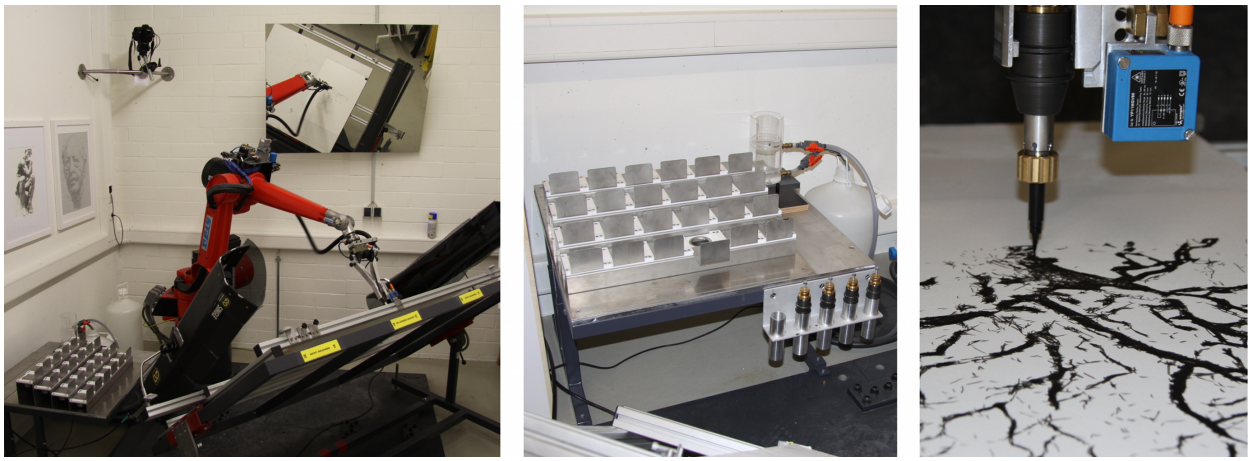
Furthermore, the feedback loop allows us to deal with the inaccuracies of brush-stroke rendering and the unpredictability of color interactions on a canvas. During our tests this behavior proved to be very important since it allows us not only to use quite inaccurate simulation techniques but can also easily be adapted to different strategies for placing strokes. This lets us realize various painting styles.

The whole eDavid experiment aims at approximating the manual painting processes by a machine. We want to find out to what extent we are able to produce artistically looking paintings. In art history it is also well known that physical limitations, e.g. interactions between ink and canvas, influence the formation of styles. We are looking for new forms of visual representations that are especially suited for painting machines; also we want to find out how to introduce high-level semantic information into the process. In recent years methods for image understanding developed a lot, so painting machines of the future could "know" what they draw and automatically adapt their painting strategy.

Since eDavid approximates the target by a sequence of strokes, we have to adapt methods for painterly rendering to work within a feedback environment: based on the target function and the currently distributed paint on the canvas a sequence of new brush strokes has to be computed that, if realized by the robot, creates a better approximation of the target by the canvas. In this sense greedy optimization is performed.

In this paper we compare two initial techniques for such stroke placements. In both cases we assume the target function to be a gray-scale image, thinned black ink serves as paint. Furthermore, the capability of each method, representing different objects is discussed. The result of the combination of both methods with respect to the semantic of objects is presented and compared to the results of each of the initial methods. An in depth exploration of color (and other media) is left for future

*Email addresses:* thomas.lindemeier@uni-konstanz.de (Thomas Lindemeier), soeren.pirk@uni-konstanz.de (Sören Pirk), oliver.deussen@uni-konstanz.de (Oliver Deussen)



(a) (b) (c)

Figure 1: System setup of the painting machine: a) robot with canvas and camera; b) repository for colors and brush tools (front row), in the back brush washing facility; c) brush tool and distance sensor.

works; however, an early result is shown in the last section. Using black ink simplifies the setup and the needed comparisons between target and canvas: color calibration can be simplified and we do not have to take care about the 3-d structure of the paint layers. Nevertheless, our framework is already able to work with color; which is why we mention color issues at some places.

The visual quality of the two initial algorithms depends on the target image. For some objects a set of short and straight strokes (we call them **stroxels**) is better suited due to their ability to create different visual appearances than a set of curved strokes and vice versa. After reviewing related works and describing our painting machine we compare their ability to represent different classes of objects. In addition, the semantics-driven and the region-based combination of both methods is demonstrated. Results are given and discussed, future works are outlined.

## 2. Related Work

There is vast majority of publications in the field of non-photorealistic rendering and computational aesthetics work on *simulating* the painting process. Since we deal with a real machine, we firstly cite methods for painterly rendering that are applicable within a feedback loop. Then we refer to drawing machines and robots that artists use for producing different kinds of paintings.

Painterly rendering was introduced in 1990 by Haeberli [2] who created artistic images using the mouse as a virtual brush. Hertzmann [3] later published a general approximation scheme for painterly rendering that starts with a rough approximation of a given image that is refined by smaller and smaller brush strokes in subsequent steps. We use this idea for representing images with a set of predefined stroke directions and lengths (stroxels). In contrast to Hertzmann we therefore limit ourselves to a fixed set of lengths and widths.

While the above algorithm just represents the colors of the image, for many styles it is also important to conserve salient image features such as edges or object boundaries. Hays and Essa [4] introduce a set of layers that refine a painterly representation at important places. Collomosse and Hall [5] use a mapping of strokes to geometric elements of different width according to the image salience. Both algorithms work well for NPR in general but are not easy to implement within our sequential feedback loop since they reorder strokes.

Kang et al. [6] compute image gradients, smooth the resulting vector field and use it to track important image features by Line Integral Convolution (LIC). The method enables them to create smooth outlines even for low quality input. We use their method to produce image-guided strokes. In contrast to their approach we do not just draw outlines but try to represent the whole gray-scale content of the image by such strokes.

An important aspect, especially when colors are used, is the ordering of colors. Northam et al. [7] explore different strategies such as painting lighter colors over darker ones or vice versa to represent features optimally. Since we currently use colors only to a limited extent,

we do not alter the order but will take this into account in the future.

In the past, a number of machines were built to create drawings mechanically. After early attempts in the 19th century, Jean Tinguely (1925-1991, see [8]) created a number of sophisticated machines that were able to create complex stroke patterns. In contemporary art, Harold Cohens Aaron (see [9]) is the most famous robot painting project, ongoing for many decades now. However, the artist did not focus on faithfully representing image content but to realize abstract computer-generated artworks. In the early times he built and used a sophisticated painting machine, but recently he moved to ordinary printers.

Early pioneers of computer graphics used plotters (e.g. Frieder Naake, cf. [10]) and also robots (e.g. Ken Goldberg, [11]) to create art works, numerous others could be listed here. Today, a number of artists uses such machines, but typically their main purpose is to create abstract and artistic paintings. Ben Grosser [12] and Holger Baer [13] are typical representatives. Zanelle [14] by van Arman is a specialized plotter to create pop-art like paintings and also somewhat more realistic portraits. An interesting painting machine is Vangobot [15], also a specialized plotter, that uses a paint mixing hardware to create color variations. To our knowledge, none of the mentioned approaches uses a feedback loop for optimization.

The segmentation of images to acquire semantic information from a given image has been of interest for many years [16, 17]. The knowledge of objects within a scene allows overcoming the limitation of applying NPR algorithms to the entire image and allows decisions to be based on low-level characteristics like colors or gradients. Recently Zeng et al. [18] presented a semantics-driven approach for stroke-based painterly rendering. Their approach allows interactively decomposing an input image into a hierarchy of its components which is then used to apply different painterly rendering styles.

More recently Deussen et al. [19] presented two different visual feedback dependent painting styles to use with the eDavid System. They discovered, that each style produces different results on the same input image considering aesthetics. This leads to the conclusion, that the choice of the painting style should depend on the painted object. In addition to their approach this paper describes the step to combine these painting styles to increase the overall aesthetics. Two approaches of such combination are presented and results are shown.

## 3. System

As shown in Figure 1, eDavid consists of a number of components. The robot itself is equipped with a specialized picking device for grasping brush tools. Five different brushes can be used in parallel by the system (see Figure 1(b), front row).

The colors are stored in a repository, so far we are able to use 24 different colors. The robot accesses a color container by mechanically opening the cover plate and dipping the brush into the color. A cleaning facility (Figure 1(b), background right) is needed when changing the paint color of a brush.

Since painting with brushes needs precise interaction we measure the surface characteristics of the canvas using a distance sensor which is mounted on the robot arm. Besides working with (slightly) curved canvases this allows us also to compensate for mechanical tolerances of the robot while moving the brush over the canvas. Such tolerances are in the sub-millimeter range but still visible for image styles with long strokes and fine brushes. Due to restrictions in the sensor (color dependency of the precision) we so far measure the distance only before we start painting for the entire canvas.

The visual feedback of the system is created by a Canon EOS 5D Mark II SLR with a 21MPixel sensor and a fixed 50mm lens. This provides us with a resolution of about 1mm on the canvas. Two specialized fluorescent tubes with polarization filters are used to illuminate the canvas, also the SLR is equipped with a polarization filter. Selecting the polarization direction perpendicular to the canvas avoids specular highlights of the paint color.

### *3.1. Software Setup*

The robot is controlled by a assembler-like language, therefore we built a server application that controls the machine and accepts XML commands. Most commands are plotter-like instructions such as pen selection and drawing, but also specialized commands for measuring the canvas and the handling of brushes and colors have been implemented.

A second server is used for the camera. The Canon SDK allows us to control all necessary functions from the computer. Images are created in XYZ color space and are calibrated by using geometric calibration patterns and color sheets. The geometric calibration is within the range of a pixel, the color calibration is about 5%, which is sufficient for our current applications but has to be enhanced in the future.

## 4. Optimization via Visual Feedback

A typical application connects to the robot server and the camera server. Based on the target function and stroke placement strategy, the application creates a number of new strokes. The strokes are realized by sending them to the robot server; after painting, the camera server is invoked to obtain the canvas image.

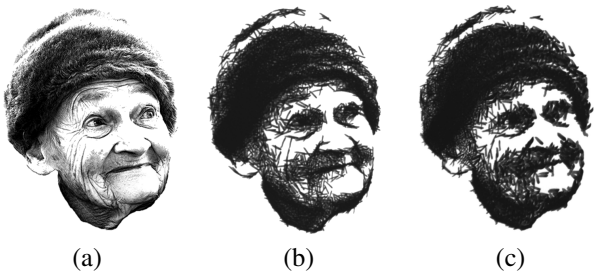
For producing paintings, a number of practical constraints have to be taken into account. Colors react differently to overdrawing when they are still wet. To let them dry we avoid painting on the same place within a given time interval. Colors such as inks are filled into specialized brush pens that allow the robot to draw continuously without dipping the brush into an ink container. For others, such as oil color, only short strokes can be realized after dipping. These characteristics are stored by the robot server in brush and paint color profiles.

### *4.1. General Optimization Strategy*

In each iteration, a number of brush strokes is generated to minimize the difference between canvas and target image. For the computation of the best approximation, the application has to anticipate the effect of painting a new stroke on the canvas. In our optimization program we implement this based on OpenGL graphics using a brush texture and opacity values that have been determined for each color. A stroke path is generated and the color application is simulated for all pixels under the stroke. Based on this simulation the quality of the stroke is computed, it determines how effective the stroke minimizes the difference between canvas and target. For a large number of stroke candidates the quality is computed and the best candidates are realized by the machine.

**Predefined stroke candidates:** A simple implementation of this strategy is to predefine a set of stroke candidates. Such candidates are typically short stroke paths in different orientations. For a position within the image all candidates are translated to that position, the quality is computed and the stroke with the highest local quality is selected and stored. For a large number of random positions this is repeated, the strokes with the globally highest quality are drawn within an iteration step.

Figure 2 shows the application of a static stroke set for the portrait of a woman. A set of 180 predefined stroke paths of the same length and width but different orientations was given. After optimization, the selected strokes approximate the given gray scale image quite well. In subfigure (c) the paper was purposely crimped to introduce an error to the realization by the robot.



(a) (b) (c)

Figure 2: Painting with predefined stroke candidates: a) input image (original: Dominik Fusina, www.flickr.com) ; b) approximation with a predefined set of 180 strokes in different directions that are applied to positions in the painting that have the highest quality. c) approximation with introduced realization error (thicker strokes are painted on the right side of the face), the process adapts to this and reduces the stroke density within this area.

Strokes on the right side of the portrait are drawn with larger line width since the brush is closer to the canvas here. Due to the visual feedback loop the machine adapts automatically to this error and the final gray scale values are similar to the undisturbed version.

**Dynamically generated strokes:** For a given position in the image, a stroke can also be generated dynamically from the image content. As suggested by works on painterly rendering (cf. [3]) a strategy for directing strokes is to draw them perpendicular to the image gradient. For each position a path is created, the effect on the canvas is simulated and the corresponding quality is computed, candidates with highest quality are realized. Both methods will be described below.

### *4.2. Computing Stroke Quality*

A number of factors influence the quality for a stroke candidate. These factors are not only responsible for the style of the result, but also for the painting strategy, i.e., the order in which strokes are painted to form the final image.

In general, strokes can only be applied at places where the color difference between target image and canvas can be lowered by applying a brush stroke with the selected paint. For gray-scale input this is equivalent to the gray-scale difference of all pixels under the stroke being larger than the opacity of an additional layer of paint, in the chromatic case this has to be measured for all color channels.

We compute the difference $d(T, C)$ between canvas image $C$ and target image $T$ and select the regions with sufficiently high image differences for the given color $i$. These regions, we call them $d_i(T, C)$, are our candidate regions for adding new brush strokes.

As mentioned above, for the practical realization and for implementing both styles, some factors have to be considered:

**Overdrawing:** Often it is not intended to paint wet-on-wet, the colors may mix in an unpredictable way or the paper might even crimp due to too much ink. Therefore regions where strokes have been applied are avoided for a predefined and color-specific time duration. This is done by intersecting $d_i(T, C)$ with a map $R$ that shows recently painted strokes. The resulting map $M_i = R \cap d_i(T, C)$ specifies the regions where new strokes can be placed.

**Brush size:** The width of the brush is a property, that influences the granularity of the results. Each brush width can be provided with a weight. This makes it possible to make specific brush widths more dominant to increase the usage of a brush width. Detailed input pictures are better represented with thin strokes, while pictures with large uniform areas are painted more visually appealing with thicker strokes. Each brush stroke $sc$ is provided with a brush weight $b(sc)$.

**Proximity:** The color (the amount of ink) that is needed to alter the pixels under a stroke candidate towards the intended color in $T$ should be approximated optimally by one of the given paint colors. So we compute the average color of those pixels and determine the closest paint color. The comparison is done on the basis of the color hue, the average difference in brightness $c_i(sc)$ is an additional factor for the quality of the stroke.

The difference weight $c_i(sc)$ for an estimated color $i$ is then defined as the normalized sum of the differences of all pixels, that are contained in $sc$:

$$c_i(sc) = \frac{1}{n} \sum_{p=0}^{n} ca_p - t_p, \tag{1}$$

with the number of pixels in the stroke $n$, the target image $t$ and the canvas $ca$.

#### *4.2.1. Additional Factors for Predefined Strokes*

Since this method only generates strokes with random directions, some further factors are needed to evaluate the orientation of stroke candidates.

**Homogeneity:** The pixels under a stroke candidate $sc$ within $M$ should have a large homogeneity $h(sc)$ or a small color variance, rsp., to be represented effectively by a single color. This automatically selects strokes perpendicular to the image gradient since in this direction there is a higher probability of having small color variances. Stroke directions of many painting styles are set according to this rule.

**Orientation:** Many painting styles prefer a uniform orientation of strokes, at least in areas where the image gradient is not too large. Examples are crosshatchings as well as impressionist or expressionist styles. The orientation of the strokes can therefore be weighted by a function $o(sc)$ that, e.g., prefers strokes in horizontal and vertical direction.

The orientation weight $o(sc)$ is calculated as follows:

$$o(sc) = 2(|cos(\alpha - \beta)^{100}| - 0.5), \tag{2}$$

with the initial angle of the brush from the texture $\alpha$ and the preferred angle $\beta$.

Values for orientation and proximity are typically in conflict and have to be balanced against each other. For some styles a direction field could be given that lets strokes follow a predefined pattern.

Our standard method to determine stroke quality for predefined strokes in a color $i$ is to combine $h(sc)$, $o(sc)$, $b(sc)$ and $c_i(sc)$ and find a path within $M_i$ that maximizes:

$$q(sc) = w \cdot h(sc) + (1-w) \cdot o(sc) + c_i(sc) + bw \cdot b(sc), \tag{3}$$

with $w, bw \in [0..1]$ being weight factors. More complex functions could perform a non-linear balancing between $h(sc)$ and $o(sc)$.

#### *4.2.2. Additional Factors for Dynamically Generated Strokes*

**Geometry:** The dynamic version of the drawing algorithms as described in [19] generates whirl strokes. This results from whirls in the vector field. A bigger smoothing kernel for the vector field suppresses most of these artifact, but does not remove all of them. A fast approach to reduce the whirls is to compute the ratio of the bounding rectangle $R$ of a stroke $sc$ and compare it to its current length $l(sc)$. The whirls and the improved results can be seen in figure 3.

The geometric weight $g(sc)$ is then defined as:

$$g(sc) = 0.5 \frac{max}{l(sc)} + 0.5 \left(1 - \frac{min}{max}\right), \tag{4}$$

with:

$$\begin{aligned} min &= min(R_{width}, R_{height}) \\ max &= max(R_{width}, R_{height}). \end{aligned}$$

Our standard method to determine stroke quality for dynamically generated strokes in a color $i$ is to combine $b(sc)$, $c_i(sc)$ and $g(sc)$ and find a path that maximizes:

$$q(sc) = w \cdot g(sc) + (1 - w) \cdot b(sc) + c_i(sc), \tag{5}$$

with $w \in [0..1]$ being a weight factor.

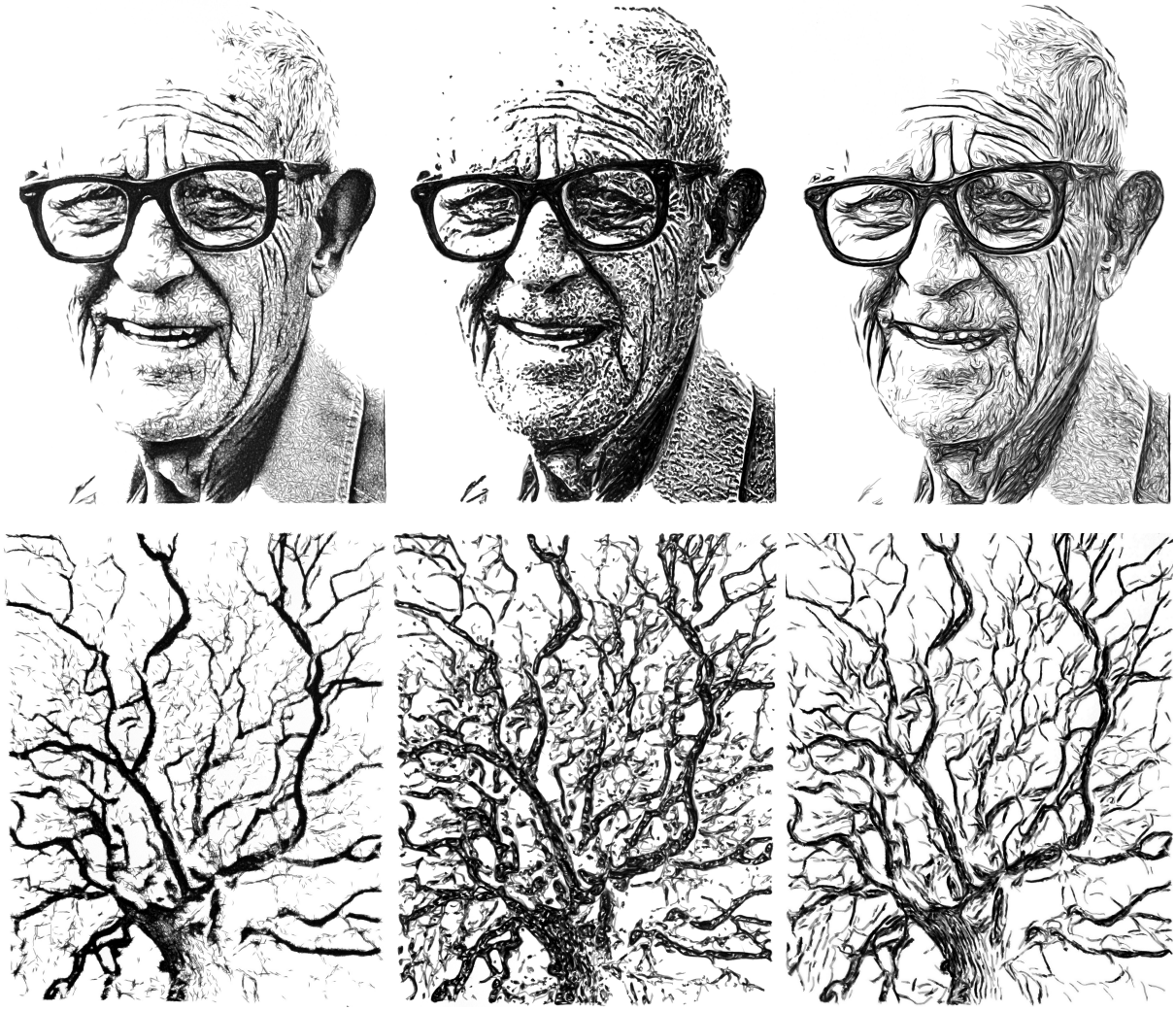


Figure 3: Predefined (left), dynamically generated (middle) and the improved version of dynamically generated strokes (right).

## 5. Realization of two Painting Styles

As mentioned above, in this paper we compare two styles to represent a target image: predefined stroxels and dynamically generated longer strokes. After we introduced the two styles in the previous section we now describe their practical realization to produce robot paintings. The two styles can be implemented as a general drawing algorithm shown in algorithm (1). The algorithms have some steps in common and only differ in the generation of the strokes in line 5 of the algorithm. Both have different representational abilities and are well suited for different objects. We will discuss this in the next section.

### *5.1. Predefined Strokes*

Our predefined set of paths for this style has three different lengths (5,10 and 20mm) and three widths (2mm, 3mm, 4mm) in 60 different directions each. This results in 540 paths that we test against thousands of random positions in $M_i$. The stroke candidates with the highest quality are selected and painted. We take care that they do not overlap by using a paint map $R$.

The upper and lower left paintings of figure 3 show objects painted with this method. Within each iteration 300 strokes were painted. This number is a good compromise between painting speed and quality. The quality will get better if fewer strokes are painted in each iteration, but the painting time will increase enormously.

**Algorithm 1:** Drawing algorithm

**Input** : input picture
**Output**: picture on the canvas
**Data**: target ← input picture

```
1 while enough strokes generated in previous step do
2     feedback ← picture from camera;
3     seeds ← random seed points;
4     foreach Point p in seeds do
5         generateStroke(p, feedback);
6     end
7     sort strokes according to their scores;
8     draw k-best strokes;
9 end
```

Increasing the number of strokes per iteration, results in too much false prediction and overpainting. This leads to a lower painting quality.

The strokes are determined in sets of five strokes each. These sets are the top stroke candidates from 1000 random positions according to Eq. (3) with $w = 0.5$ and $o(cs) = 0$ (no orientation preference) and algorithm (2). Sixty of these sets are painted in a single iteration. If a stroke was selected, its simulation is added to the maps $R$ and $M_i$ to prevent the ongoing simulation from overdrawing strokes within the current iteration.

**Algorithm 2:** generateStroke - Pre Defined Strokes

**Input** : point, feedback
**Output**: evaluated stroke
**Data**: sw ← brush size weight, ow ← orientation weight, pw ← proximity weight, hw ← homogeneity score

```
1 sc ← get random brush from pre defined textures;
2 h(sc) ← homogeneityScore(stroke);
3 b(sc) ← brushScore(stroke);
4 o(sc) ← orientationScore(stroke);
5 c_i(sc) ← differenceScore(stroke);
6 q_sc ← hw·h(sc) + sw·b(sc) + ow·o(sc) + pw·c_i(sc);
```

### 5.2. Dynamically Generated Strokes

In a second method we compute strokes dynamically on the basis of the image information and its gradient. The image gradient is computed using a Sobel operator, subsequently it is filtered to allow larger and smoother strokes (cf. Kang et al. [6]). This gradient is used to produce stroke paths that later guide the brush strokes.

In each iteration of the visual feedback loop, a large number of initial random positions is generated. For each position a path is generated using Line Integral Convolution (LIC) of the gradient vector field (see Cabral et al. [20]). This lets the strokes follow what Kang et al. call "edge tangent flow". We use a Runge-Kutta integration method of fourth order. Uniform regions can be filled with predefined directions or by computing a distance transform [21].

Since with this method many paths are directed towards the edges of the image, we store the already drawn paths in a separate file. We test if a new path overlaps existing ones and if the difference at the overlap position of the canvas and the target image brightness is less than zero, the path will be cut (see algorithm (3), line 1-16). If the remaining stroke is shorter than the minimum length *minLen* it will be deleted (see algorithm (3), line 17).

The minimum length is set to 5 pixels. Strokes with a length below 5 pixels tend to look similar to strokes produced with the previous method. The maximum length *maxLen* used for the results in this paper is set to 80 pixels. Strokes longer than this have the characteristics to overpaint themselves. In addition, this maximum stroke length fits best the size of the painted objects. Within each iteration 100 strokes instead of 300 were painted, since this method generates larger strokes than the previous one to counteract false prediction and overpainting as mentioned in section 5.1.

In a second step (see algorithm (3), line 18-27) the remaining part of the path is tested for rendering according to Eq. (5). We take $M_i$ and compute the stroke quality for a stroke with maximum width. Typically the stroke will overlap with already drawn parts and therefore possibly extend outside of $M_i$, which is penalized in the quality function. The stroke width is gradually reduced and the quality is determined again. The width with maximum quality is added to a candidate list. The robot realizes the candidates with the highest quality. The method automatically prevents the machine from overdrawing, therefore no additional overlap test is applied here.

### 5.3. Practical Considerations

Both styles were drawn with a specialized fiberglass brush pen, a mixture between a brush and a pen. This brush is very robust and thus convenient for our application. We extended its ink cartridge to be able to draw many thousand strokes without refilling it. This allows us to paint 2000-3000 strokes per hour. Unfortunately, higher speeds are prohibitive since the acceleration may cause unintended spurting of ink on the canvas and reduces the accuracy.

**Algorithm 3:** generateStroke - Dynamically Generated Strokes

**Input** : point, feedback
**Output**: evaluated stroke
**Data**: v ← edge tangent flow, minLen ← minimum stroke length, minScore ← minimum score, sw ← brush size weight, gw ← geometry weight, pw ← proximity weight

```
1  lp ← point;
2  for i ← 0 to maxLen/2 do
3  |   lp ← RungeKutta4Forward(v, lp);
4  |   if feedback_lp − target_lp > 0 then
5  |   |   append copy of lp to stroke;
6  |   end
7  |   else break;
8  end
9  lp ← point;
10 for i ← 0 to maxLen/2 do
11 |   lp ← RungeKutta4Backward(v, lp);
12 |   if feedback_lp − target_lp > 0 then
13 |   |   prepend copy of lp to stroke;
14 |   end
15 |   else break;
16 end
17 if length of stroke < minLen then return;
18 foreach brush size bsize do
19 |   sc ← createBrushStroke(bsize, stroke);
20 |   b(sc) ← brushScore(s);
21 |   g(sc) ← geometryScore(s);
22 |   c_i(sc) ← differenceScore(s);
23 |   q_sc ← sw · b(sc) + gw · g(sc) + pw · c_i(sc);
24 end
25 choose brush stroke with highest score q;
26 if best q < minScore then reject stroke;
27 else accept stroke
```

The ink was thinned by a factor of 1:15 to have the ability of gradually darkening the canvas. Interestingly, we had many problems with highly thinned ink because it has the tendency to sublimate and block the pen brush, which then dries out. All images have a size of 40-60cm and were drawn with approx. 30.000-40.000 brush strokes. Painting took between 10 and 15 hours depending on the type of brush strokes and the overall speed of the machine.

Using the set-up described so far, a number of practical constraints are imposed on the simulation and feedback loop of our active visual control:

**Termination:** For each iteration we record the number of unsuccessful stroke placements (no stroke with sufficiently high quality was found within 1000 random positions). If too many failures occur in a sequence we stop the iteration (see algorithm 1, line 1). This kind of stopping criterion seems to be much more stable than other methods such as root mean squared pixel errors (RMS) between target image and canvas. This is due to the fact that a lot of visual noise is created during our approximation leading to unstable image-based termination criteria.

**Color Calibration:** Since color calibration never works perfectly we have to take into account that the saturation on the canvas will possibly not reach the exact amount of our prediction. Therefore tolerances have to be added, otherwise the process would not terminate and the robot would repeatedly overdraw already saturated regions.

**Camera Lens:** Though we have a relatively high resolution camera, the feedback image is blurred. Small strokes are sometimes not shown exactly at the right position and thus can also introduce repeated overdraw. On the other hand the blur seems to stabilize the iteration. In his approximation scheme Hertzmann [3] also uses blur for distributing the introduced error. Maybe a similar effect is introduced here.

**Paint Interaction:** On the canvas, colors mix in a much more complex way than we can simulate on the computer. Therefore we use highly thinned colors to correct our prediction errors by repeated overdraw during the feedback iterations. Even in the gray-scale case such complex interactions have to be taken into account.

### *5.4. Visual Analysis*

Both algorithms terminate with a valid visual representation of the object. It is hard to determine their quality by means of absolute measurements since their characteristics are too different and the approximations are too rough to use quantitative pixel-based error metrics such as RMS.

In Figure 4 we compare a simulation of stroxels and their practical realization. Typically different strokes are drawn in our simulation and by the robot within the feedback loop; in general, however, the gray-scale values are simulated quite faithfully. Some differences can be seen though: strokes are darker than assumed, their width is larger and also their shapes do not really match

our simulation. As mentioned above, our feedback loop allows the process to adapt to the so far created gray-scale values on the canvas. This limits the overall error.

A very informal investigation of members of the university did not yield a clear preference for one of these styles (please refer to [22] for a review of evaluation methods based on user studies). While the stroxels look more "technical", the dynamically generated strokes have more curly and "expressionistic" look that might be better suited for organic forms.

Both styles incorporate enough "artistic" attributes to be considered "beautiful". If predefined paths are not restricted in their direction, the textures of technical objects such as the building are sometimes rendered too arbitrary, orientation preferences could help here.

Figure 4: Simulation of stroxels (left) and their practical realization.

## 6. Semantics

Both introduced algorithms handle the whole picture uniformly. Each painting technique described in section 4 generates strokes that only depend on the brightness and gradients. There is no differentiation between objects. A common technique in the art of painting is to change the painting style according to objects. Background is painted with less detail, while the foreground is painted with thinner strokes to enhance the details and to let the foreground appear sharper [18]. Humans recognize semantics of objects and adapt their painting style. Grass for example is painted with short thin strokes, which are aligned perpendicular to the ground. Skin is represented by strokes, which are painted into one another to construct a uniform area. There are only a few painterly rendering techniques that take semantics and the differentiation of objects into account. Most of the algorithms like Hertzmann's painterly rendering method [3] just use region information to choose the size of the brushes used. Since the use of the additional information about objects could improve painterly rendering results, this suggests a fruitful direction for future work, which would be to explore painting different styles.

### *6.1. Combination of two Painting Styles*

The two styles described in section 4 create different representations of the objects from the input image. The short static strokes tend to represent details with more precision and fill areas in a more natural way. It performs better with technical, clearly defined objects. The dynamic method produces larger curvier strokes, which follow the flow of the objects. It creates nice results on using it with organic objects like trees or hair. The combination of both styles can produce results with a more appealing overall look.

There are two different approaches to combine the two methods:

**Region based approach:** Since the dynamic method uses the image gradients to generate strokes it comes to mind to draw only lines, that follow strong gradients. Only lines, whose total gradient magnitude lies above a fix threshold are drawn. Noisy uniform regions are rejected and later filled with predefined strokes. The results are shown in figure 6. The strong edges at the beak and feathers are drawn with dynamically generated strokes. The dark areas and details are later filled with predefined strokes.

**Object based approach:** A more semantics-driven approach is to manually select objects and to split the input image by this objects. Each object is now painted alone with each corresponding algorithm. It could also be possible to select objects with the techniques mentioned in [18], but at the moment it is sufficient to select the objects manually. The results are shown in figure 5. The tree is painted with dynamically generated strokes. The more flowing style creates the illusion of wind. The grass is painted with predefined strokes. The short strokes represent the grass stalks more realistic and appealing.

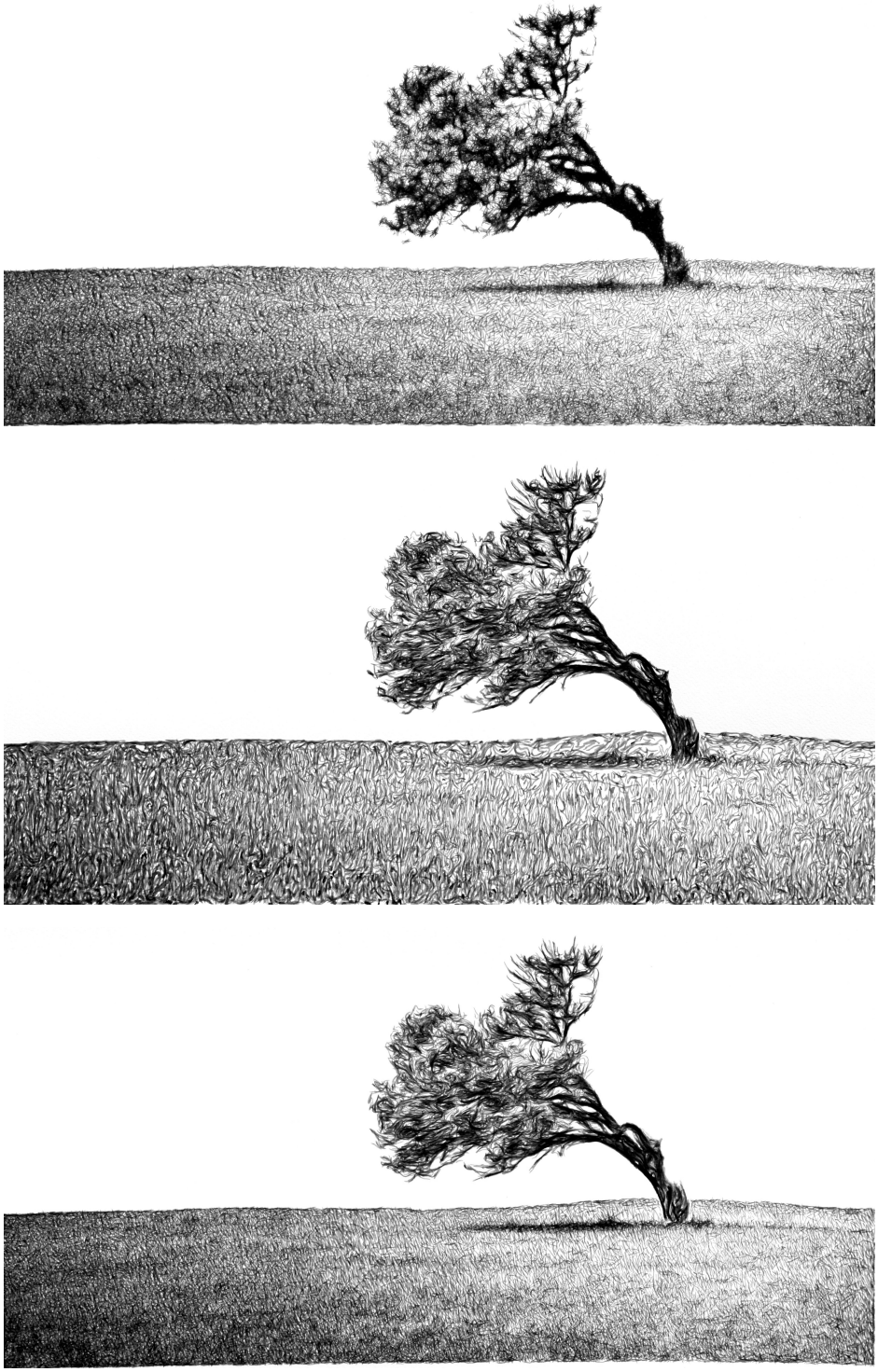


Figure 5: Predefined (top), dynamically generated (center) and the semantic driven combination (bottom).

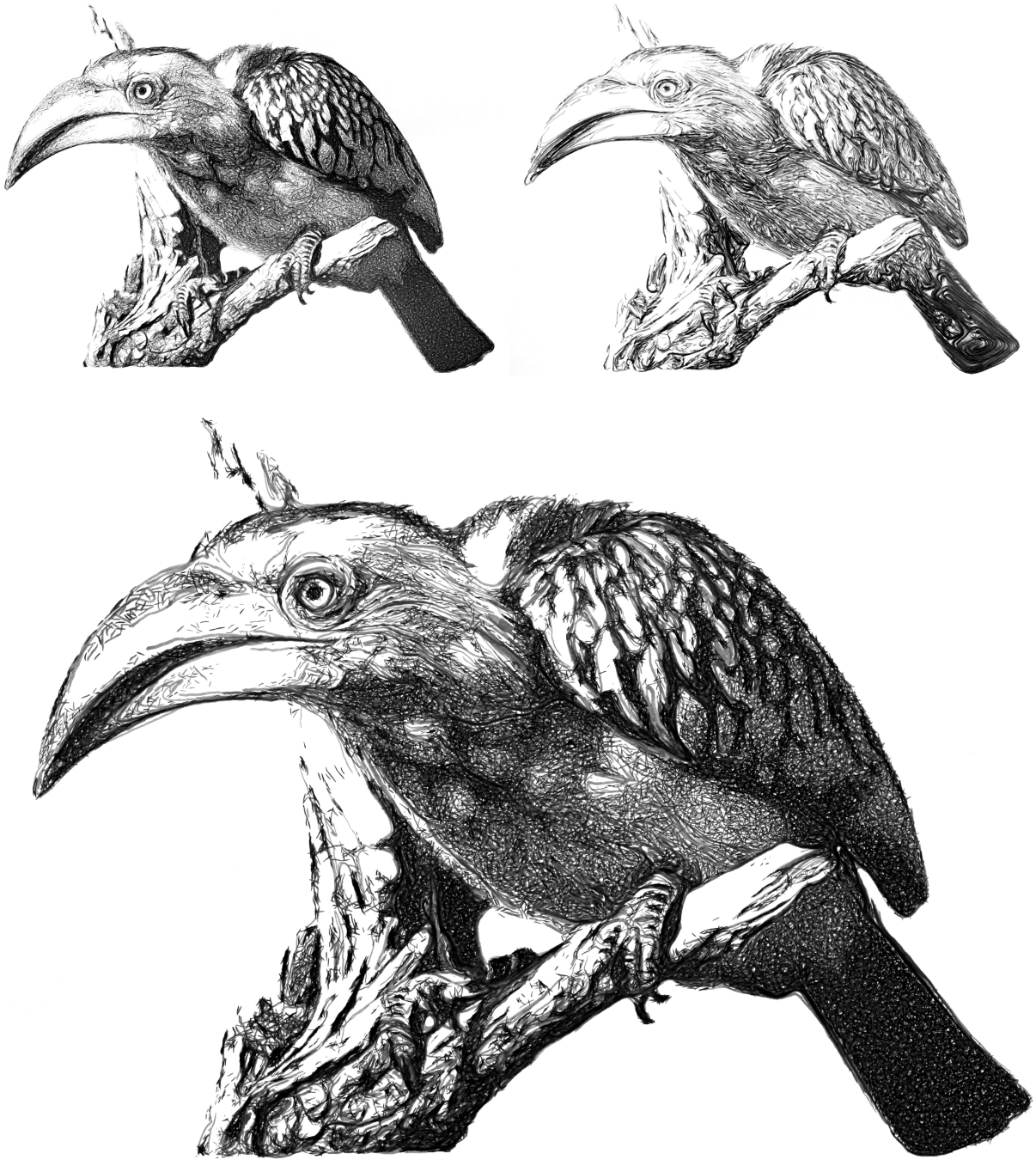


Figure 6: Predefined (top left), dynamically generated (top right) and the region based combination (bottom).

## 7. Conclusion and Future Work

We presented a mechanical painting system that works with visual feedback and an optimization loop. An industry robot is used to move a brush over the canvas and a set of specialized tools was developed to enable experiments with different brushes and paint colors. Two painting styles and a semantics-driven combination were investigated and discussed together with a number of results.

A methodologically interesting question for the future would be a numerical evaluation of the painting results. Appropriate statistical means have to be found to compute the quality of different artistic representations.

In the future we want to further extend our experiments to colors, an early result is shown in Figure 7. First trials with oil colors showed that this medium is quite difficult to handle, different colors have different viscosity, the handling of brushes is much more complex. Furthermore, the mixing of colors has to be improved, a solution similar to VangoBot [15] could help here. Nevertheless, we consider our set-up as a good test bed for many experiments, and in the future we plan to invite artists to explore the space of creative possibilities opened by eDavid.

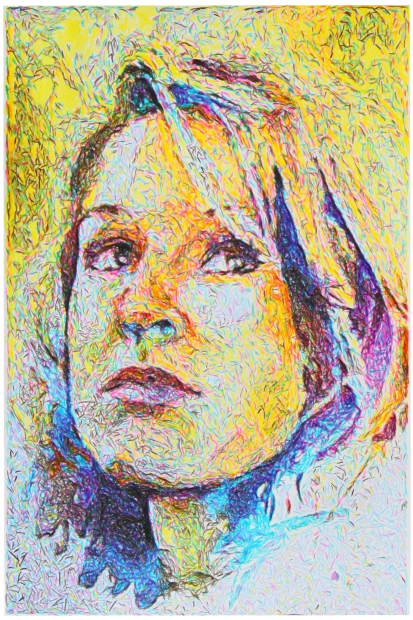


Figure 7: Color Result. Created after a painting of David Lobenberg.

## References


[1] O. Deussen, eDavid, a Painting Robot, http://graphics. uni-konstanz.de/, 2012.

[2] P. Haeberli, Paint by numbers: abstract image representations, in: Proceedings of the 17th annual conference on Computer graphics and interactive techniques, SIGGRAPH '90, ACM, New York, NY, USA, 1990, pp. 207–214.

[3] A. Hertzmann, Painterly rendering with curved brush strokes of multiple sizes, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, SIGGRAPH '98, ACM, New York, NY, USA, 1998, pp. 453–460.

[4] J. Hays, I. Essa, Image and video based painterly animation, in: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering, NPAR '04, ACM, New York, NY, USA, 2004, pp. 113–120.

[5] J. P. Collomosse, P. M. Hall, Painterly rendering using image salience, in: 20th Eurographics UK Conference, Eurographics Assoc., 2002, pp. 122—128.

[6] H. Kang, S. Lee, C. K. Chui, Coherent line drawing, in: ACM Symposium on Non-Photorealistic Animation and Rendering (NPAR), pp. 43–50.

[7] L. Northam, J. Istead, C. S. Kaplan, Brush stroke ordering techniques for painterly rendering, in: Computational Aesthetics 2010 Eurographics Workshop on Computational Aesthetics in Graphics Visualization and Imaging Victoria British Columbia Canada May 2830 2009, Eurographics Association, 2010, pp. 59–66.

[8] Wikipedia, Tinguely art machines, http://en. wikipedia.org/wiki/Tinguely, 2012. March 13th, 2012.

[9] H. Cohen, Aaron, http://crca.ucsd.edu/ hcohen/, 2012. March 13th, 2012.

[10] Wikipedia, Frieder Naake, http://de.wikipedia. org/wiki/Frieder_Nake, 2012. March 13th, 2012.

[11] Wikipedia, Ken Goldberg, http://en.wikipedia. org/wiki/Ken_Goldberg, 2012. March 13th, 2012.

[12] B. Grosser, http://bengrosser.com/, 2012. March 13th, 2012.

[13] H. Baer, http://www.holgerbaer.com/, 2012. March 13th, 2012.

[14] P. van Arman, Zanelle, http://www.vanarman.com/, 2012. March 13th, 2012.

[15] L. Kelly, D. Marx, The vangobot project, http://vangobot.com, 2012. April 30th, 2012.

[16] Z. Tu, X. Chen, A. L. Yuille, S. C. Zhu, Image parsing: Unifying segmentation, detection, and recognition, in: Toward Category-Level Object Recognition'06, pp. 545–576.

[17] Z. Tu, S.-C. Zhu, Parsing images into regions, curves, and curve groups, Int. J. Comput. Vision 69 (2006) 223–249.

[18] K. Zeng, M. Zhao, C. Xiong, S. C. Zhu, From image parsing to painterly rendering, ACM Trans. Graph. 29 (2009).

[19] O. Deussen, T. Lindemeier, S. Pirk, M. Tautzenberger, Feedback-guided stroke placement for a painting machine, in: Computational Aesthetics, pp. 25–33.

[20] B. Cabral, L. C. Leedom, Imaging vector fields using line integral convolution, in: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, SIGGRAPH '93, pp. 263–270.

[21] T. Saito, J.-I. Toriwaki, New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications, Pattern Recognition 27 (1994) 1551 – 1565.

[22] A. Hertzmann, Non-photorealistic rendering and the science of art, in: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, NPAR '10, ACM, New York, NY, USA, 2010, pp. 147–157.