Interactive Example-based Hatching

Moritz Gerl, Tobias Isenberg

Interactive Example-based Hatching

Moritz Gerl[a,*], Tobias Isenberg[a,b]

[a] *Institute of Mathematics and Computer Science, University of Groningen, The Netherlands*
[b] *DIGITEO in collaboration with LIMSI–CNRS and AVIZ–INRIA, Orsay, France*

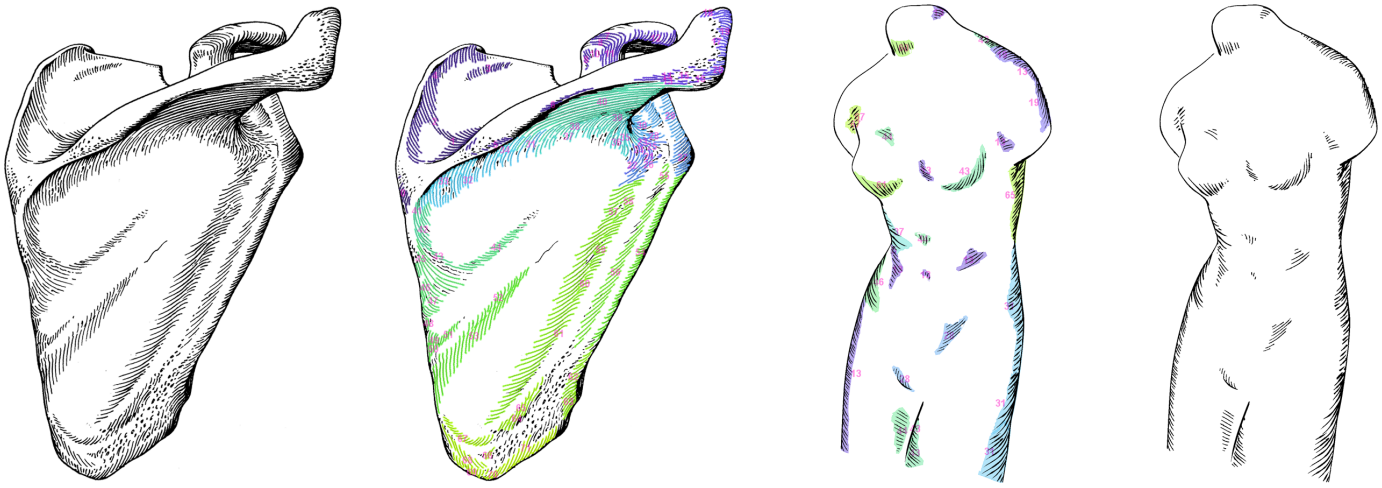


Figure 1: We learn the hatching style from a hand-drawn example (left) and use it to semi-automatically synthesize example-based hatching illustrations (right). The style transfer is based on patches of hatching strokes. The example illustration is manually decomposed into patches of similar strokes (indicated by the colors). Each patch of synthesized hatching strokes is assigned one patch of example strokes (indicated by the numbers) and recreates the corresponding hatching properties.

Abstract

We present an approach for interactively generating pen-and-ink hatching renderings based on hand-drawn examples. We aim to overcome the regular and synthetic appearance of the results of existing methods by incorporating human virtuosity and illustration skills in the computer generation of such imagery. To achieve this goal, we propose to integrate an automatic style transfer with user interactions. This approach leverages the potential of example-based hatching while giving users the control and creative freedom to enhance the aesthetic appearance of the results. Using a scanned-in hatching illustration as input, we use image processing and machine learning methods to learn a model of the drawing style in the example illustration. We then apply this model to semi-automatically synthesize hatching illustrations of 3D meshes in the learned drawing style. In the learning stage, we first establish an analytical description of the hand-drawn example illustration using image processing. A 3D scene registered with the example drawing allows us to infer object-space information related to the 2D drawing elements. We employ a hierarchical style transfer model that captures drawing characteristics on four levels of abstraction, which are global, patch, stroke, and pixel levels. In the synthesis stage, an explicit representation of hatching strokes and hatching patches enables us to synthesize the learned hierarchical drawing characteristics. Our representation makes it possible to directly and intuitively interact with the hatching illustration. Amongst other interactions, users of our system can brush with patches of hatching strokes onto a 3D mesh. This interaction capability allows illustrators who are working with our system to make use of their artistic skills. Furthermore, the proposed interactions allow people without a background in hatching to interactively generate visually appealing hatching illustrations.

Keywords: Illustrative rendering, non-photorealistic rendering, interactive illustrative rendering, style transfer, hatching, learning hatching, hatching by example, illustrations by example, example-based, interactive example-based, pen-and-ink

1. Introduction

The computer generation of pen-and-ink hatching illustrations has a long tradition in non-photorealistic and illustrative rendering. But although pen-and-ink rendering methods have been introduced quite some time ago, there seems to be little adoption outside the field. This contrasts the long tradition and widespread use of handcrafted pen-and-ink illustrations. Over centuries, those depictions have been proven to be an effective means of visually communicating information. And they are still widely used nowadays, e. g., in medical training. We be-

*Corresponding author. Tel +31 50 3634602, Fax +31 50 3633800. SVCG, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands.
Email address: gerl@cs.rug.nl (Moritz Gerl)
URL: http://tobias.isenberg.cc (Tobias Isenberg)

lieve that the lack of adoption of computer-generated hatching illustrations can at least in part be explained by their synthetic and overly regular visual appearance. A comparative study of Isenberg et al. [1] has shown that computer-generated illustrations are clearly distinguishable from hand-drawn illustrations due to such lack of 'character'. We aim at incorporating human virtuosity and illustration skills into the computer generation of pen-and-ink hatching illustrations to improve upon the aesthetic appeal and illustration effectiveness of such imagery. Recently, Kalogerakis et al. [2] presented an approach that shares our goals. In this paper we present an approach that uses an explicit representation of drawing elements, in contrast to the pixel-based approach chosen by Kalogerakis et al. [2]. This representation allows us to transfer drawing characteristics on higher levels than a pixel grid and to provide user interactions for adjusting the resulting illustrations. Our approach provides the following contributions:

Image analysis on hatching drawings: We apply image processing methods to the detection of the stroke trajectories in a hand-drawn pen-and-ink hatching illustration. This image analysis allows us to learn the drawing style from a given example image and to use the example image for texture mapping.

3D information for existing 2D drawings: We employ a 3D scene registered with a hand-drawn example illustration to infer 3D information related to the 2D elements in the example image. This enables us to include 3D measurements in learning a model of the drawing style in the example image.

Analytical representation of drawing elements: We represent patches of strokes and stroke trajectories analytically in object-space. This representation allows us to transfer drawing characteristics on a stroke level (as opposed to the transfer of hatching properties on a pixel level). The analytical representation of hatching patches and hatching strokes (see Fig. 1) also enables us to provide user interactions for adjusting the result.

Adaptive surface patches: We introduce the use of a real-time example-based mesh segmentation to create adaptive surface patches as the basis for creating patches of hatching strokes. This allows us to capture and reproduce global characteristics of an example illustration. While we use the surface patches for generating hatching strokes, the notion of patch-wise style transfer can be generalized to other depiction styles.

Stroke distance functions: We propose to learn the interrelationship of hatching strokes locally as a function of object-space mesh features. This allows us to model the stylistic variations of the target illustration style on a stroke level.

Interaction capabilities: We provide users of our system with the possibility to interact with the resulting illustration. Users of our system can alter the appearance and direction of hatching strokes within individual patches of strokes, can flexibly retouch the stroke trajectories, and brush with patches of hatching strokes. These direct interactions with the hatching illustration are an effective means of combining the benefits of automatic style transfer with human creativity and enable an application of our method in creative environments. Furthermore, the interaction capabilities of our approach allow users without a background in hatching to interactively create visually appealing hatching illustrations.

The remainder of this document is structured as follows. We first review related work in Section 2. Next, we give an overview of our approach in Section 3. We then detail how we capture a drawing style in Section 4 and explain how we reproduce the learned style in Section 5. We explain our user interactions in Section 6. In Section 7 we present some results of our method and discuss its limitations in Section 8. We conclude the paper and describe possibilities for future work in Section 9.

2. Related Work

Many different ways for creating hatching renderings have been explored in the past. Saito and Takahashi [3] introduce the usage of isoparametric lines to create hatching images. This idea was further developed by other researchers [4, 5]. Already in this first generation of hatching techniques, Winkenbach and Salesin [6] incorporate concepts derived from hand-drawn hatching illustrations, such as the notion of varying the width along a stroke in order to simulate marks which are created by applying ink to paper with a nib pen. Girshick et al. [7] introduce the creation of strokes based on principal curvature directions, a notion used by many following hatching techniques. Hertzmann and Zorin [8] perform an optimization of the curvature directions and use the resulting smooth direction field to create hatching strokes. In our work, we use this optimized direction field as a basis for learning the directions of example hatching strokes. Zander et al. [9] propose to create object-space hatching strokes by tracing the curvature directions in object space. In our approach, we also use a object-space representation of the stroke trajectories. In contrast to these explicit stroke descriptions, Praun et al. [10] introduce a hatching approach using textures. This allows for real-time rendering, while still achieving expressive results. Praun et al.'s [10] technique was extended by Kim et al. [11] to work for dynamic and specular surfaces. Despite the appealing properties of these texture-based methods, we decided to use an explicit stroke representation. This representation gives us the required control over singular strokes which we need for reproducing learned stroke properties. All the mentioned methods have in common that the hatching strokes depend on a mapping of a small set of lighting conditions and mesh features to stroke properties. It is thus very difficult for these methods to convincingly reproduce the stylistic properties and variations present in hand-drawn hatchings. For this reason, Kalogerakis et al. [2] recently proposed to learn hatching styles from example drawings. While Kalogerakis et al.'s [2] work serves as an inspiration of our own, we improve on it by using analytical representations of hatching patches and hatching strokes, by modeling stroke distance interrelationships in more detail, as well as by specifically permitting interaction. We explain how we deviate from Kalogerakis et al.'s [2] work in more detail below.

There has been some previous effort in style transfer and example-based illustrative rendering. Hamel and Strothotte [12] capture user-defined rendering parameters and re-use them for rendering other meshes. Mertens et al. [13] transfer texture variations between meshes by correlating texture properties with geometric mesh features, using similar fea-

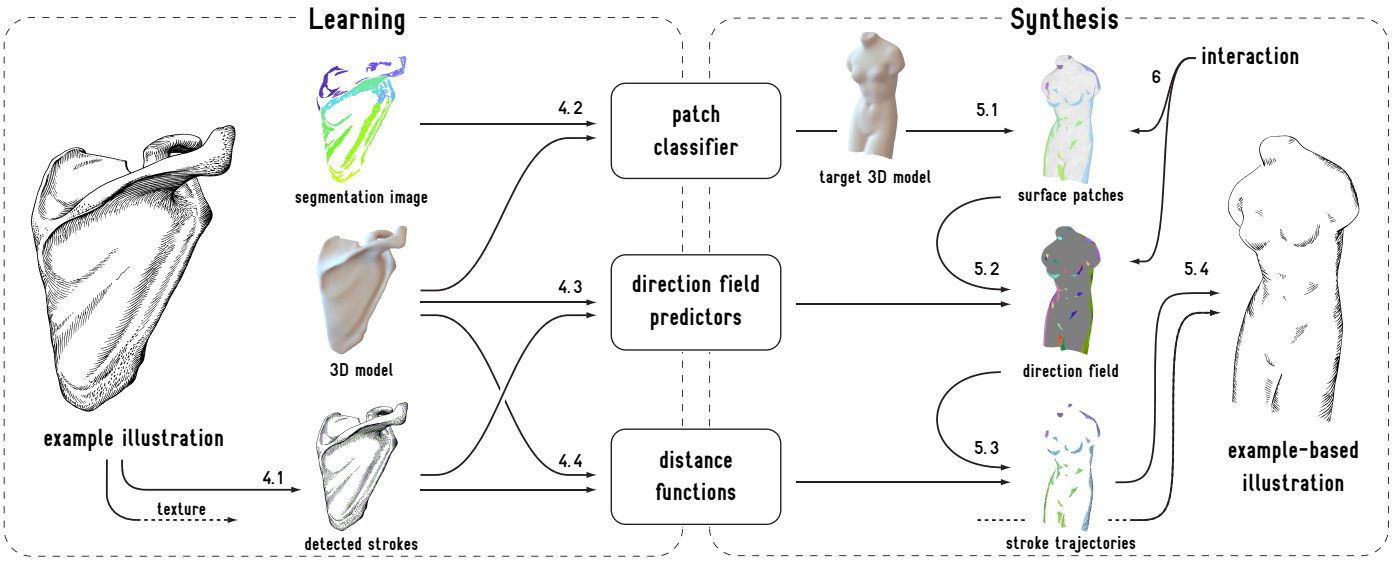
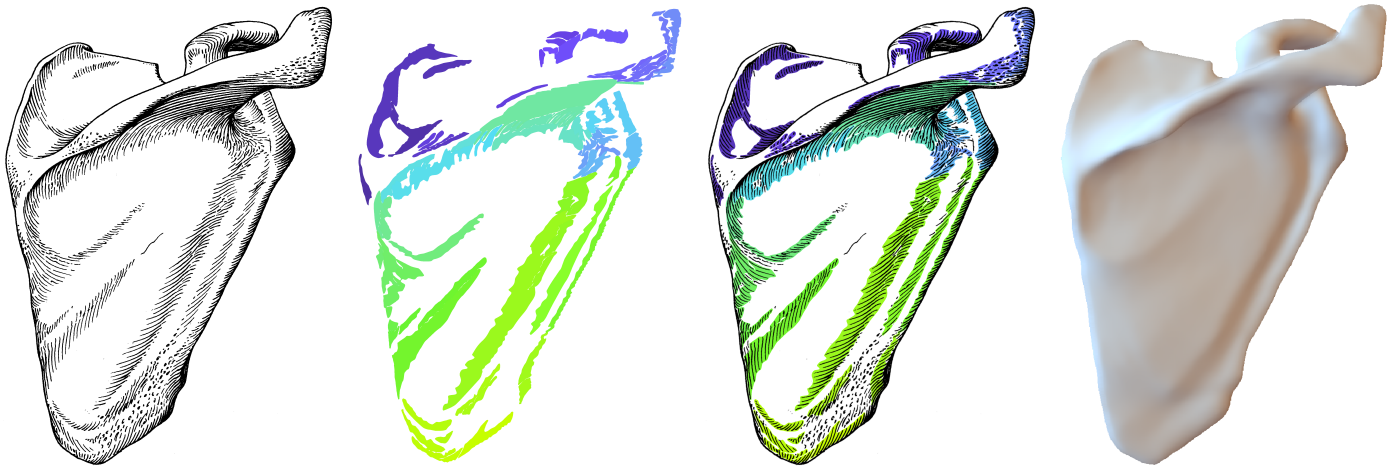


Figure 2: Overview of our approach to hatching by example. In a learning stage, we learn the characteristics of an illustrator's pen-and-ink hatching style. In a synthesis stage, we apply the learned style to a target 3D model to gain an example-based illustration. Capture and synthesis of the drawing style is performed using a four-level hierarchy of global, patch, stroke, and pixel levels. The arrows are annotated with the section numbers where the corresponding processes are explained.

tures as we do in our approach. Hertzmann et al. [14] present a framework capable of learning and reproducing image processing filters which mimic drawing and painting styles, taking 2D images as input. Although many different styles can be successfully transferred, the pixel-based nature of this technique makes it difficult to faithfully reproduce drawing styles which depend on long and individual strokes. Zhao and Zhu [15] generate example-based portrait paintings from photographs by transferring brush strokes from a collection of template paintings. Also working on 2D images, Kim et al. [16] adopt statistical texture transfer methods to transfer the stippling characteristics from example stippling illustrations to new images. Related to this, Martín et al. [17] present a method for scale-dependent and example-based stippling based on halftoning. Stippling by example is mostly concerned with calculating adequate stipple positions and shapes. Herein, the individual stipple points do not have a function on their own, but work as a conglomerate. In the drawing style which we aim to reproduce, in contrast, each drawing mark has an individual function. Furthermore, the hatching strokes are subject to interrelations along their entire extent, as opposed to the interrelations of only one 2D location per mark in the case of stippling. For these reasons, hatching by example requires more complex style transfer models.

Some approaches establish statistical models of stroke patterns. Jodoin et al. [18] as well as Barla et al. [19] synthesize stroke patterns by example based on statistics which they derive from given input stroke patterns. Other related work [20, 21, 22, 23] focuses on the style transfer between curves. These approaches transfer stroke patterns or line rendering styles merely in 2D. For our needs, however, it is necessary to involve 3D information in the style transfer process. For this reason, we condition our style transfer model on a 3D object, and also apply the model for creating renderings of 3D objects.

This is inspired by the work of Lum and Ma [24] as well as that of Cole et al. [25]. Both of these approaches use machine learning to correlate hand-drawn line drawings with computer-generated silhouettes and feature lines. Applying machine learning to learn hatching properties was only recently introduced by Kalogerakis et al. [2]. Their approach operates on pixels: both learning and inference of hatching properties are performed on a per-pixel basis. The final hatching illustrations are then synthesized by tracing streamlines in image space. The transfer of drawing characteristics based on pixels means that the transfer does not involve any explicit representation of drawing elements, such as strokes. The results of Kalogerakis et al. [2] prove that this strategy works very well for the illustration styles they work with. For learning the illustration styles that we aim to reproduce, however, we need different learning strategies, in particular a different drawing representation. Therefore, we operate on explicit analytical representations of hatching strokes and patches of strokes. This explicit representation of drawing elements enables us to capture drawing characteristics and stylistic properties present in four nested levels of abstraction. These levels are a global, patch, stroke, and pixel level. Representing the hatching strokes explicitly *during* the style transfer process allows us to transfer local stroke distance characteristics, which results in less uniform and less equidistant strokes than Kalogerakis et al.'s [2] global pixel-based approach to transferring stroke distances. Furthermore, the method proposed by Kalogerakis et al. [2] delivers a static result which cannot be modified after its generation. Our explicit description of drawing elements, in contrast, opens up the possibility to interactively modify the resulting illustration. We exploit this control by providing users of our system with the possibility to brush with patches of hatching strokes onto a 3D model, amongst other interactions. The interaction capabilities of the resulting semi-automatic hatching system allow



(a) (b) (c) (d)

Figure 3: The input to our approach for learning a hatching style. (a) A hand-drawn hatching illustration, (b) a manually created segmentation image of this illustration ((c) shows an overlay of the latter two images for reference), and (d) a 3D scene whose projection closely matches the example image.

users of our system to manually enhance the aesthetic quality of the results. This possibility for adjusting the result gives our method the potential of being employed in creative environments as well as by laymen in hatching illustration.

The concept of interactive illustrative rendering has been proven successful, not only since Seims [26] advocated to provide more user control for fully automatic non-photorealistic rendering methods. Salisbury et al. [27, 28] let users brush with stroke patterns to interactively create pen-and-ink illustrations. Deussen et al. [29] employ user-defined segmentation images and brushing interactions for the semi-automatic generation of stippling drawings. Rössl and Kobbelt [30] also use image-space segmentations in their interactive system for creating line-art renderings. Here, the segmentations are created automatically, but can be adjusted by the user. We also rely on a user-adjustable automatic segmentation to identify regions of different drawings characteristics. However, we propose to perform the segmentation on the mesh and to learn the segmentation using machine learning methods, similar to the approach of Kalogerakis et al. [31] for learning mesh segmentation and labeling. Furthermore, we provide users with the possibility to interactively modify the resulting illustration by allowing them to refine the mesh segmentation. This interaction is related in its intention to the direct tweaking of lighting and shading as proposed by Anjyo et al. [32] and extended by Todo et al. [33].

Furthermore, Breslav et al. [34] also use patches embedded on the surface for creating illustrative renderings. They use pre-defined 3D patches to transform 2D patterns in a way that the transformed 2D patterns match the underlying 3D transformation. In contrast to pre-defined patches, we use adaptive 3D patches that are dynamically predicted based on a learned function of lighting conditions and mesh features. And instead of transforming 2D patterns, we use the 3D patches to guide the generation of strokes trajectories in 3D.

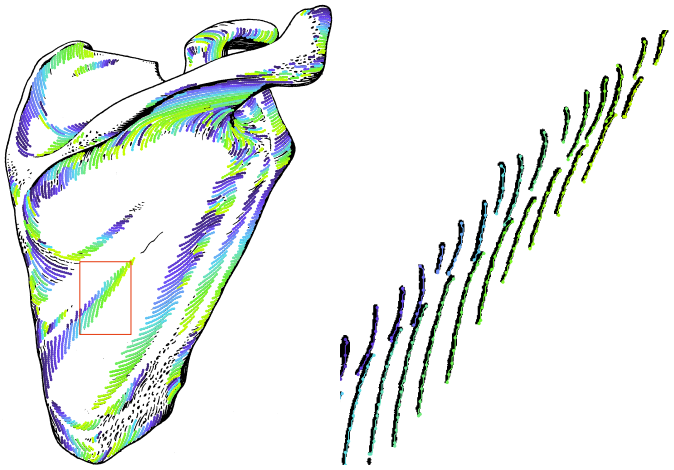
3. Overview

Our overall approach consists of two general stages (see Fig. 2). First, we learn a model of an illustrator's pen-and-ink hatching style (Section 4). Then, we apply this model to synthesize hatching illustrations of target 3D meshes (Section 5). The synthesis can be influenced by user interactions (Section 6).

As input to the learning stage (see Fig. 3) we use a hand-drawn hatching illustration, a manually created segmentation image of this illustration, and a 3D scene whose projection closely matches the example image. We refer to this 3D replication of the example drawing as 'registered 3D model'. We use it to infer 3D information related to the 2D example image. We obtain the registered 3D model manually by sculpting it from a 3D model similar to the object depicted in the illustration.

In a preprocessing step to the learning procedures, we use image processing to detect the trajectories of the strokes in the example image (Section 4.1). This stroke detection is facilitated by using the manually created segmentation image to separate groups of strokes from the remainder of the input image.

The segmentation image also defines patches of strokes in the example illustration which function as a group and which share common properties. We explicitly take these groups of similar strokes into account and attempt to learn the properties of the groups of strokes in the following way. Using the registered 3D model and the segmentation image, we train a classifier that maps from lighting and geometric mesh features to segment labels (Section 4.2). When we apply this classifier on a target 3D mesh in the synthesis stage, it yields a dynamic segmentation of the mesh into surface patches which incorporate the learned global drawing characteristics (Section 5.1). We use these surface patches as a basis to generate object-space stroke trajectories. The patch classifier operates on the whole mesh, and represents the first level of our four-level hierarchy of hatching style descriptors. To complement the described automatic inference of hatching regions, the surface patches can also be interactively modified by the user via brushing (Section 6).



(a) (b)

Figure 4: Stroke detection. We detect trajectories of hatching strokes in a given example illustration using image processing ((b) shows a detail of (a)).

The second level of our hierarchy is concerned with the directions of the hatching strokes. We learn the properties of the stroke directions in the example illustration on a patch level. To be able to do this in 3D, we use the following heuristic: we reproject the detected 2D stroke trajectories onto the registered 3D model. In this way, we gain an object-space description of the trajectories of the hand-drawn strokes. We then use regression analysis to learn how the 3D stroke trajectories correlate with lighting and geometric features of the registered 3D model (Section 4.3). Note that the reprojected strokes allow us to learn the directions of the strokes *on the surface*, rather than learning the *image-space* directions of strokes in dependency of surface features. We train one regression function for each patch of strokes in the example image. In the synthesis stage, every surface patch is assigned one example stroke patch. We then apply one direction field function per surface patch to infer a direction field which incorporates the stroke directions learned from the corresponding example stroke patch (Section 5.2).

The third level of our model deals with the distance relationship of individual strokes with their neighboring strokes. Here we use the same reprojection setup as described above. The reprojection setup allows us to correlate 2D stroke distances with 3D lighting conditions and surface features (Section 4.4). In order to establish this correlation, we train a regression function for each individual stroke in the example illustration. In this way, we learn the 2D distances of the stroke to its neighboring stroke along its extent as a function of the surface features measured at the locations of the reprojected stroke control points. In the synthesis stage, we use these stroke distance functions during the tracing of stroke trajectories to reproduce the recorded patterns of stroke interrelationship (Section 5.3). This local approach to transferring stroke distances results in hatching patterns that exhibit controlled example-based irregularities.

The final step in the synthesis is to create 2D textured triangle strips from the 3D stroke trajectories (Section 5.4). This represents the fourth level of our hierarchy and involves two attempts of transferring low-level stroke properties.

The resulting hatching illustration can be manually refined by mapping and brushing operations (Section 6). This semi-automatic system effectively combines the advantages of automatic example-based hatching and human creativity. First, the illustration can be altered by overriding the learned mapping of surface patches to example stroke patches. By reassigning a different example stroke patch to a surface patch, the strokes within this patch can be changed. Second, the hatching angle of each individual patch can be controlled. Third, the direction field that we use as a reference for inferring the stroke direction field can be retouched with brushing tools. Fourth, we allow users of our system to brush with patches of example-based hatching strokes. All generated hatching strokes depend on the dynamic mesh segmentation (Section 5.1). Users can easily modify, add, or remove patches of strokes by refining this segmentation with a set of brushing interactions.

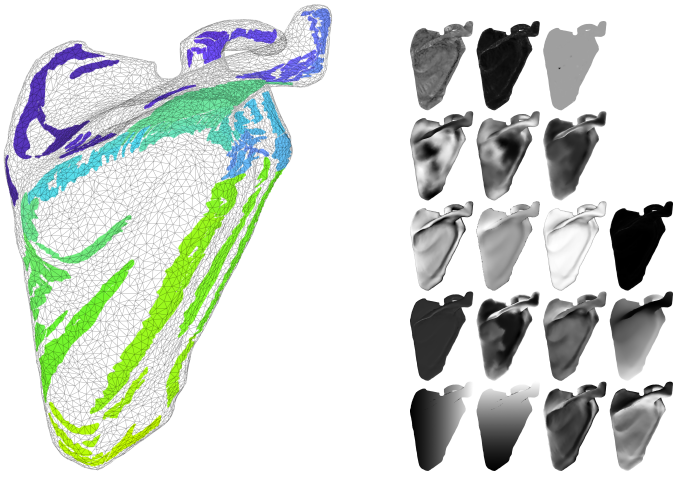
The interactions are facilitated by our object-space representation of drawing elements. Our object-space approach has several advantages over the image-space approach of Kalogerakis et al. [2] regarding user interactions. Users of our system can, e. g., zoom and rotate the 3D model to a view that permits an intended editing of hatching strokes, execute the interaction in this close-up view, and then go back to the original view.

4. Learning a Hatching Style

In this section we explain the learning part of our approach in more detail. The illustration style that we aim to learn is a specific type of traditional pen-and-ink hatching. Fig. 3(a) shows an example from an anatomy textbook [35]. A property of this hatching style is that the drawing is, for the most part, composed of separate individual strokes. Each stroke in the drawing has a particular function. This contrasts other, more areal, hatching styles which use many overlapping strokes (styles that are visually similar to, e. g., the real-time hatching images of Praun et al. [10]). Based on this property we can automatically detect the strokes in hand-drawn images that are drawn in our target illustration style. The detection of strokes would be harder to accomplish on an image consisting of many overlapping strokes.

4.1. Image Analysis

By establishing an explicit analytical representation of the strokes in an example illustration, we create the possibility to learn the properties of the strokes on higher levels than a mere pixel representation would allow us to do. As input to our stroke detection, we use a high-resolution black-and-white scan of an example drawing (Fig. 3(a)). As second input, a manually created segmentation image (Fig. 3(b)) allows us to separate patches of strokes from the rest of the drawing. For each patch, we run a series of standard morphological operations [36] to detect the trajectories of the hatching strokes. Our image processing pipeline starts with a morphological cleaning operation to remove scanning artifacts. Then we use connected component labeling to identify the strokes. Thinning the stroke regions yields skeletons of the strokes. A hit-or-miss transform identifies skeleton junctions and endpoints which we use to prune the



(a) (b)

Figure 5: Learning patch properties. We use (a) the segmentation image and the registered 3D model to learn the global properties of patches of strokes with respect to (b) lighting and geometric mesh features as described in the text.

skeletons and to separate the trajectories of overlapping strokes from each other. We then vectorize the strokes by creating equidistant control points along the stroke skeletons. We provide details on our image processing pipeline in Appendix A. Fig. 4(a) shows the result of the described stroke detection routines, Fig. 4(b) shows a detail section. By reprojecting the detected stroke trajectories onto the registered 3D model, we can relate object-space properties of the hand-drawn strokes to 3D measurements. This *reprojection* of drawing elements to object-space is a novelty compared to previous approaches that also use a 3D model registered with a 2D drawing, but merely *read out* 3D information related to 2D elements in an input drawing [2, 24, 25]. Our approach, in contrast, allows us to learn *object-space* properties of otherwise image-space elements, such as the directions of hatching strokes *on the surface*. Before we do that on a patch level and locally, however, we make an attempt to capture global properties of the drawing style as outlined next.

4.2. Patch Properties and Surface Features

We identify the grouping of strokes in patches as a central stylistic element of our target illustration style. It can be seen in Fig. 3(c) that the strokes in each patch share common attributes such as direction, width, and shape. In order to take this grouping of strokes into account, we explicitly involve it in our style transfer model. This explicit handling of stroke groups is an essential distinction between our approach and the global pixel-based approach of Kalogerakis et al. [2]. The manual segmentation of the example illustration allows us to faithfully distinguish the different patches of similar strokes. This would be difficult to realize automatically, as it involves complex perceptual and creative decisions. We use the segmentation image together with the registered 3D model (Fig. 5(a)), which we denote here as the input mesh, for capturing the properties of these groups of strokes. We project each vertex of the input mesh to image-space and read out the patch label found at this location in the segmentation image. Assigning a patch label to each vertex represents a segmentation of the input mesh that matches the segmentation given by the segmentation image. We take this mesh segmentation and a number of lighting and geometric features (as detailed below) measured at each vertex and train a classifier to learn a mapping from mesh features to segment labels. Applying this classifier in the synthesis stage assigns a segment label to each vertex of a target mesh (Section 5.1). The resulting dynamic segmentation of the target mesh into surface patches incorporates the global characteristics of the example illustration as defined by the segmentation image.

We use a voting multiclass classifier [37] with a one-vs.-one strategy for classification. We employ relevance vector machines [38] with radial basis function kernels as binary classifiers. We experimented with various classifiers and gained the most promising results with the named one.

As mentioned before, we use a relatively small set of surface features compared to the approach of Kalogerakis et al. [2]. We selected a set of 18 decisive features. We identified these by correspondence with professional artists and illustrators, by drawing conclusions from the literature on computer-generated hatching, and by experiment. We experimented with various features, and selected a set of features that lead to a robust classification of patch properties. We consider a classification as robust if it results in continuous patches which, assessed by subjective reasoning, match the areas used by the creator of the example image. We made a tradeoff of classification speed and accuracy for choosing the number of features. The classifier operates on feature vectors of scalar values. For including 2D and 3D measurements in our model, we either use their components or the dot product with the view vector as a view-dependent scalar of a 3D variable. Fig. 5(b) depicts renderings of the employed features in reading order and as listed below.

The six view-independent features we use are: the first and second principal curvature magnitudes $|\kappa_1|$ and $|\kappa_2|$, the 'parabolicalness' $|\kappa_1|/|\kappa_2|$, as well as the x-, y- and z-components of the first principal curvature direction after performing the curvature optimization procedure proposed by Hertzmann and Zorin [8] λ_{1_x}, λ_{1_y}, and λ_{1_z}, which we here denote as the first optimized curvature direction.

The 12 view-dependent features we use are: diffuse illumination I (Lambertian shading), approximated global illumination *SSDO* (screen-space directional occlusion as introduced by Ritschel et al. [39]), facing ratio $n \cdot v$ (where n is the normal and v is the viewing direction), facing ratio gradient magnitude $|\nabla(n \cdot v)|$, view-dependent facing ratio gradient direction $(\nabla(n \cdot v)) \cdot v$, view-dependent first optimized curvature direction $\lambda_1 \cdot v$ (where λ_1 is the first optimized curvature direction), view-dependent second optimized curvature direction $\lambda_2 \cdot v$, depth, the image-space coordinates x_i and y_i, as well as the x- and y-components of the normal projected to image space n_{i_x} and n_{i_y}.

We normalize and weight the features to control the influence of each of the features individually. The weighting is achieved by multiplying each feature with a user-controllable weight. Multiplying a feature that is normalized to the range of $[0, 1]$ with a weight greater than 1 causes the scaled feature to have greater impact during learning and inference.

We put most emphasis on the lighting features as we assume them to have the greatest impact on where the illustrator has drawn which kind of strokes. We see this assumption confirmed by the literature on illustration [40]. The assumption is also reflected by the ranking of features reported by Kalogerakis et al. [2]. When we learn a model of the stroke directions and distances, we adjust the feature weights accordingly. The feature weights we use are listed in Appendix B.

The described model of patch properties is prone to overfitting. Moreover, the sparse set of features makes it less accurate than the model of Kalogerakis et al. [2]. We discuss the resulting limitations in more detail in Section 8.

4.3. Stroke Directions

We learn and predict the locations of patches of example strokes on a global (mesh) level. We now descend one level in our style descriptor hierarchy and explain how we capture the directions of hatching strokes on a patch level. We do this by establishing a mapping of surface features to the directions of the example strokes reprojected onto the surface. In this way, we learn how the surface directions of the strokes drawn by the illustrator correspond with surface features. Applying this mapping in the synthesis stage yields an example-based direction field (Section 5.1) which incorporates the directional characteristics of the learned hatching style.

We use the same set of features as described in Section 4.2 for learning the stroke directions, while weighting the directional features significantly stronger. We use the optimized curvature direction field proposed by Hertzmann and Zorin [8] as a reference direction field. We gain an object-space representation of the example strokes by reprojecting the detected example strokes (Fig. 4(a)) onto the the registered 3D model (Fig. 3(d)). We then use regression analysis to learn a mapping from surface features to 3D stroke directions. We measure a scalar of the local stroke direction as the angle between the local stroke direction and the first optimized curvature direction in the tangent plane. With local direction we mean the direction of a segment of a stroke represented as 3D polyline. We train one regression function for each example stroke patch. For each vertex of a patch (see Fig. 5(a)) we gather and average the local stroke directions at the K nearest control points of the reprojected example strokes. For the examples presented in this paper, we used a value of $K = 5$. Eventually, the training data for learning the direction field function consist of one feature vector and one angle per vertex of a patch. We employ kernel ridge regression [41] using radial basis function kernels for learning. We selected this learning method also by experiment, comparing it to radial basis function networks and relevance vector machines.

4.4. Stroke Distances

While we perform the capture and reproduction of stroke directions on a patch level, we model the distances between neighboring strokes more locally on a stroke level. We here use the same reprojection setup as described in Section 4.3. For each example stroke, we learn the 2D distances from one of its neighboring strokes along its extent as a function of the surface features. In the synthesis, we use these distance functions to push strokes towards or away from their neighboring strokes (Section 5.3). In this way we can recreate the learned patterns of local stroke interrelationship.

For learning the stroke distance functions, we again use the same set of features as explained in Section 4.2 and put most emphasis on the image-space coordinates and on the diffuse and ambient lighting (see Appendix B for details). For a horizontal example stroke, we measure the 2D vertical distance to its lower neighbor stroke at every control point. We interpolate the surface feature vectors measured at the vertices to gain interpolated feature vectors at the reprojected control points. We use barycentric coordinates for a component-wise interpolation of the feature vectors. Using this data, we train a regression function for each stroke. We here also employ kernel ridge regression with radial basis function kernels.

4.5. Summary

After running the described learning procedures, we have a description of the drawing style stored in the following way. The coordinates and widths of the example strokes are stored in a text file, grouped in patches. We make use of the dlib library [42] for performing the described machine learning methods. The patch classifier, the direction field functions, as well as the stroke distance functions are stored as dlib decision functions. The example stroke data and learned functions can thus be loaded from disk and used in the synthesis stage of our method.

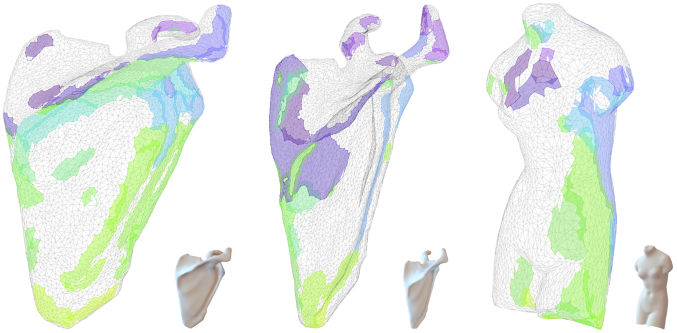
5. Hatching Synthesis

In this section we detail how the model described in the previous section is applied to a target 3D mesh in order to synthesize a hatching illustration by example.

5.1. Adaptive Patches

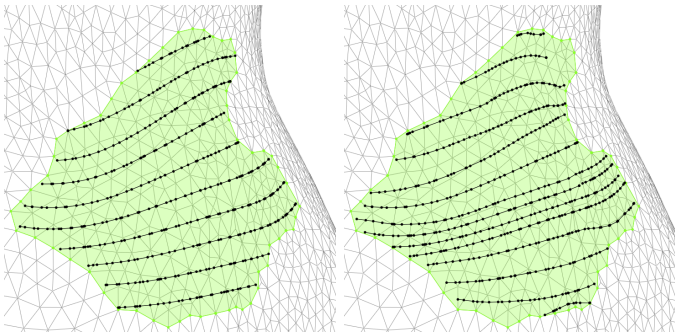
The first step in our synthesis pipeline is to apply the patch classifier described in Section 4.2 to predict a patch label at each vertex of the target mesh. This yields a real-time dynamic segmentation of the target mesh into surface patches which incorporate the recorded global properties of the hatching style we aim to reproduce. It is dynamic because the segmentation is regenerated for every frame, and gives a new result according to the new view direction and lighting conditions.

Based on this vertex labeling, we grow adaptive patches on the surface in order to gain an explicit geometric representation of the surface patches. We let surface snakes evolve on the mesh as proposed by Bischoff et al. [43] to gain the patches. We employ surface snakes following the predicted patch labels to collect the connected vertices and faces of every patch label, and to establish an explicit representation of the boundary of each patch. Our surface snakes do not evolve iteratively and do not move according to a velocity, in contrast to the snakes described by Bischoff et al. [43]. Our surface snakes evolve recursively and move the full length of an edge per recursion. We prevent the generation of too small patches by omitting all patches that are formed by a surface snake whose number of



(a) (b) (c)

Figure 6: Adaptive surface patches. We generate a dynamic example-based mesh segmentation and grow adaptive surface patches from the resulting labeling. The figure shows the patches generated for (a) the input mesh and view, (b) a different view, and (c) for a different mesh. We use these surface patches to generate hatching strokes. The surface patches can also be edited directly via brushing interactions (Section 6).



(a) (b)

Figure 7: Stroke distance control. The interrelationship of (a) strokes following the direction field is (b) enhanced by applying the learned distance functions. The stroke distance functions introduce controlled example-based irregularities.

snaxels is below a threshold. Fig. 6 shows the resulting adaptive surface patches for the input mesh, for a different view, and for a different mesh. We denote them as adaptive because they adjust to the current viewing and lighting conditions. When the object or the light sources are transformed, the patches move along the surface. These adaptive patches embedded on the surface are the basis for the following steps in our hatching synthesis pipeline. The explicit representation of hatching regions as surface patches also makes it possible to realize brushing interactions that allow users to directly adjust the resulting illustration (see Section 6). We manually adjust the surface patches for the results shown in this document to improve the results of our automatic prediction of hatching regions (see Section 8).

5.2. Example-based Direction Field

For each adaptive patch, we apply a direction field function as described in Section 4.3 to infer an example-based stroke direction field. Every adaptive patch is associated with an example stroke patch and uses its direction field function. Thus, a different direction field is inferred for each adaptive patch. The inferred direction field incorporates the directional characteristics learned from the example illustration. The inference takes place at the vertices of the target mesh. One angle is inferred for each vertex. The angle is obtained by evaluating the direction field function using the respective feature vector as argument. We rotate the optimized curvature direction by the inferred angle on the tangent plane to obtain the final stroke direction. We use the resulting example-based direction field for tracing the trajectories of hatching strokes on the surface. This patch-wise direction field inference is similar to the segment-wise direction inference in image space presented by Kalogerakis et al. [2]. Our object-space representation has the advantage, however, that it enables us to provide object-space brushing interactions for editing the reference direction field (see Section 6). This reference field retouching allows users to flexibly and directly adjust the trajectories of hatching strokes on the surface.

5.3. Stroke Tracing and Distances

We trace stroke trajectories on the surface by integrating the inferred stroke directions in object space. At the same time, we use the learned distance functions as explained in Section 4.3 to control the distances between strokes in image space. In this way we generate strokes that follow the example-based direction field and that recreate the learned patterns of stroke distance interrelationship. Each stroke is associated with an individual distance function. Our local approach to transferring stroke distances results in a more detailed transfer of stroke interrelationships as compared to the global pixel-based approach of Kalogerakis et al. [2]. This approach results in hatching strokes that are less regular and less equidistant, which enhances the hand-drawn character of our results.

The stroke control points are embedded on the surface, living within triangles of the mesh and on mesh edges. During the tracing within a triangle, we interpolate the stroke directions inferred at the triangle's vertices using barycentric coordinates. A trajectory is stopped when it reaches the boundary of an adaptive patch. During the tracing in object space, we control the stroke distances in image space, in contrast to previous object-space techniques [9]. While tracing a stroke, we repeatedly evaluate its associated distance function. We use the predicted distances from the neighboring stroke as a second component influencing the position of each new control point, complementing the directions from the stroke direction field. The contributions of the predicted directions and distances can be controlled with one user-tunable parameter. The effect of applying these stroke distance functions is demonstrated in Fig. 7. For creating the result images in this paper, we used a value of 0.3, meaning that each new position is calculated to 30 percent by distance from the neighbor and to 70 percent by direction.

We place strokes incrementally, according to a seeding strategy adopted from Jobard and Lefer's [44] streamline algorithm. We use the distances predicted by the learned stroke distance functions for seeding new strokes from existing ones as well as for terminating strokes which come too close to each other.

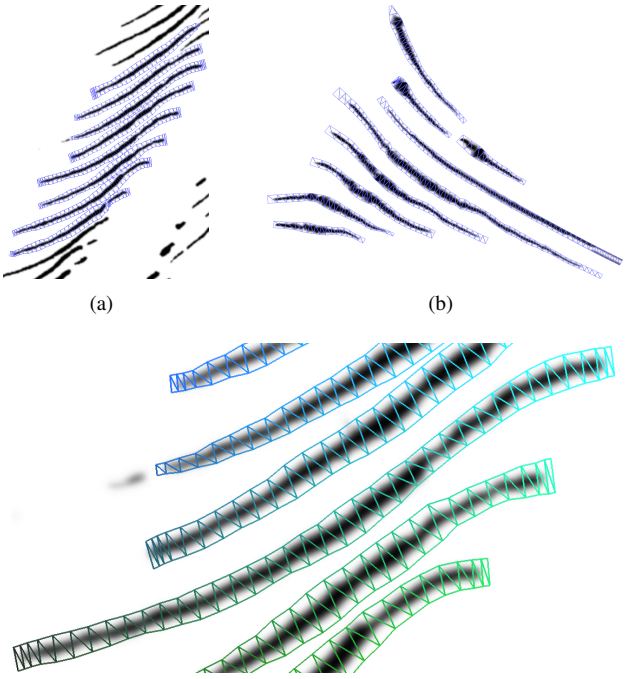


Figure 8: Example-based stroke widths and stroke texturing. (a) shows a section of the texture space (the texture image with superimposed texture coordinates) and (b) shows a set of rendered strokes in image space. We use the entire example image as texture and create (a) texture coordinates that tightly enclose strokes in the example image. This is done by mapping the trajectories of the detected strokes to texture space and inflating them according to the measured stroke widths. Each rendered stroke in (b) is assigned one texture-space stroke from (a). The measured stroke widths are used for creating both the geometry of the (a) texture-space strokes as well as of the (b) rendered strokes. (c) shows a detail section of (a) with a color-coded rendering of the texture coordinates. The u-coordinate is mapped to green color and the v-coordinate to blue color.

5.4. Stroke Rendering

We create 2D textured triangle strips from the 3D stroke trajectories for rendering the hatching strokes. This involves two attempts of transferring low-level stroke properties. With those attempts we try to reproduce the width and shape of the pen-and-ink strokes in the example image.

First, we use the stroke widths measured along the detected example strokes for creating the 2D stroke geometry. Our stroke detection allows us to measure the stroke widths along the extent of each detected stroke and to parameterize the measured stroke widths depending on the position on the trajectory. We use these parameterized stroke widths for creating triangle strips of varying width during stroke rendering. At each control point of a stroke to be rendered, we calculate the current width using the measured and parameterized stroke width.

Second, we texture the resulting strokes using the entire example image as texture. We realize this texturing by creating texture coordinates that tightly enclose individual strokes in the example image (see Fig. 8 for details and further explanation).

We perform antialiasing for on-screen display via blurring and mipmapping the example texture and via supersampling. We threshold and binarize the images for the result images in this document, which also helps to reduce texturing artifacts.

Figure 9: Mapping interaction. A patch of hatching strokes is assigned a different example stroke patch.

6. Interaction with the Hatching Illustration

The processes described in the previous section automatically synthesize a hatching illustration of a 3D model. To complement this automatic generation, we allow users to interact with the illustration in four different ways. Users can modify which type of strokes are generated within a surface patch, adjust the hatching angle, retouch the reference direction field with brushing tools, and brush with patches of hatching strokes. This set of interactions allows users to adjust the illustrations according to their requirements and aesthetic judgment and, thus, to enhance the aesthetic appearance of the resulting illustrations.

First among the interactions, users can modify which type of strokes are created within a particular region. We realize this modification by assigning a different example stroke patch to a surface patch. The reassignment leads to the usage of a different direction field function, different stroke distance functions as well as different stroke widths and textures. This interaction, thus, results in a different appearance of hatching strokes within an adaptive patch. Fig. 9 shows the effect of such an interaction.

Second, users can control the hatching angle of each individual patch of hatching strokes. We realize this interaction by adding a user-controlled angle to the stroke direction angle that is inferred during the direction field inference. This interaction allows users to adjust the general hatching direction of all strokes within one patch, while the strokes still incorporate the learned and reproduced directional characteristics.

Third, users can modify the stroke trajectories by retouching the reference direction field (the optimized curvature direction field) with brushing interactions. Modifying the reference direction field results in a modified inferred direction field and, thus, in modified stroke trajectories. We provide three different direction field editing tools. These three radial brushing tools operate with a user-controllable brush size, strength, and hardness (a Gaussian attenuation of modification intensity dependent on the distance from the brush center). The first of the three tools rotates the reference directions by a user-defined angle on the tangent plane. This tool allows users to freely adjust the direction of stroke trajectories. As a second operator, a blur tool averages the reference directions in the brushing area and allows users to smooth stroke trajectories. Finally, a clone stamp tool transfers the reference directions from a source area to the brushing area. Equally to the clone stamp in Adobe Photoshop®, the

source area is selected initially and moves relatively to the cursor location. This clone stamp tool allows users to conveniently retouch singularities and discontinuities of the reference direction field. Together, the described brushing interactions on the reference direction field permit users to freely adjust the stroke trajectories to their needs. The tools can be used for coarse adjustments, such as modifying the general hatching direction of an entire patch, or fine-grained modifications, such as bending the tip of an individual stroke. This flexible control over object-space stroke trajectories is a novelty of our approach that improves upon the static nature of existing hatching methods.

Fourth, users can brush with patches of hatching strokes. This is realized by interactively altering the automatic mesh segmentation (see Section 5.1) via brushing. This brushing interaction is implemented as the assignment of a particular patch label to mesh vertices within the brushing area. Changing the mesh segmentation in this way effectively results in adding, modifying, or removing adaptive patches. The hatching strokes within the modified patches are re-generated on the fly. This gives users the possibility to interactively modify the hatching illustration to achieve the desired result. In some cases, the automatically generated hatching patches are suboptimal because they cover unwanted areas and are not existent in other areas where strokes are desired. Users can then adjust the illustration with the described brushing interaction to achieve aesthetically more pleasing and more effective results. Users can as well disable the automatic prediction of hatching regions and start from scratch to freely brush hatching patches onto the surface according to their requirements. The stroke directions and distances, however, are always inferred automatically.

The described interactions override the automatic prediction of hatching properties. On the one hand, these interactions serve for dealing with limitations of our automatic style transfer mechanisms (see Section 8). On the other hand, the interactions effectively combine the advantages of automatic hatching and human creativity. Our interactions, therefore, represent novel tools for the semi-automatic generation of hatching illustrations in the spirit of existing semi-automatic methods for non-photorealistic rendering [6, 27, 29]. The interactions provide users with a means to directly and intuitively specify or modify the regions where strokes are placed, which type of strokes are generated in which region and at which direction on the surface. These interaction capabilities make an employment of our method in a creative environment more likely than the usage of a fully automatic and static method. Furthermore, the proposed integration of example-based hatching with interaction capabilities permits users without a background in hatching to create illustrations they would otherwise not be able to create, as demonstrated in Section 7.

7. Results and Discussion

In this section we discuss some results generated with our method. We first present some results of applying the drawing style learned from the shoulder blade illustration shown in Fig. 3(a). We then show the input and results of transferring the hatching style of a second example illustration. All the result images are created semi-automatically. Manual adjustments include the mapping of example stroke patches, the modification of the general hatching directions of individual patches, the retouching of the reference direction field to refine stroke trajectories, and the adjustment of the hatching areas via brushing.

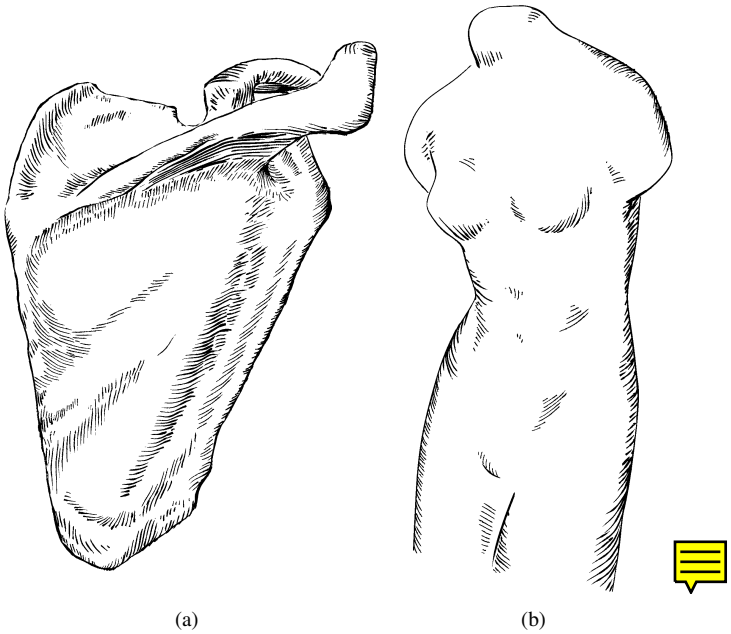
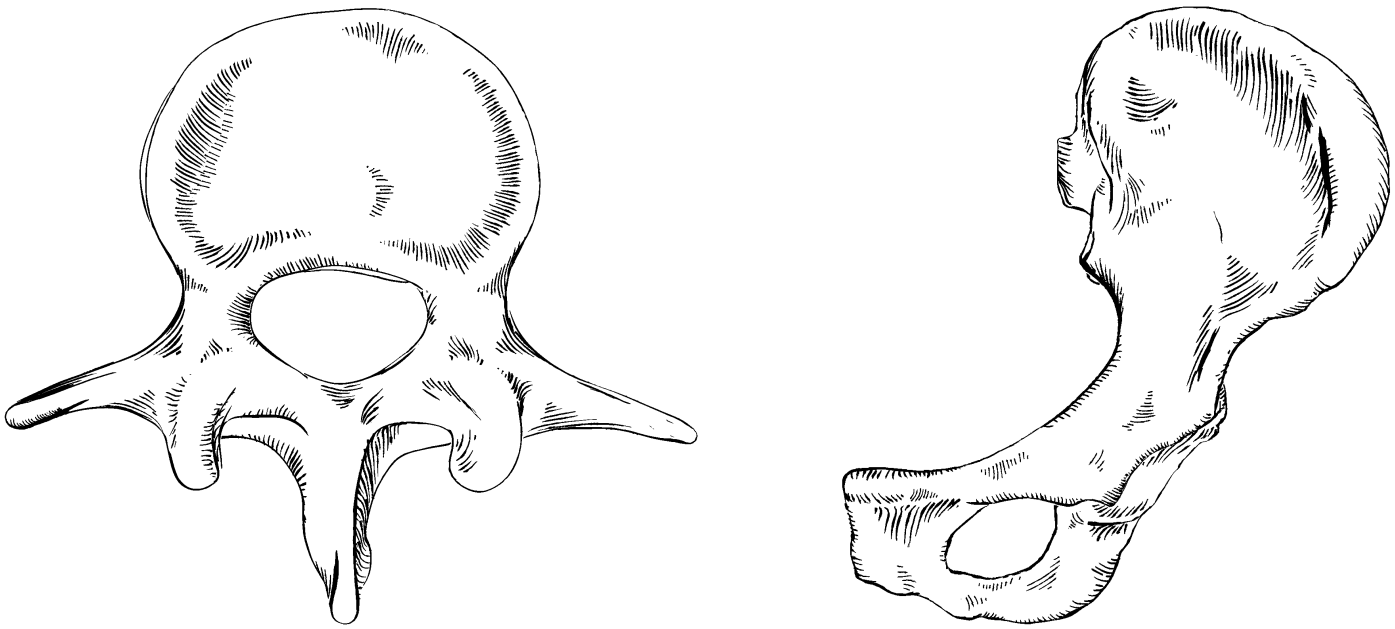


Figure 10: Illustrations using the hatching style learned from the shoulder blade illustration in Fig. 3(a). The depicted objects are (a) the shoulder blade mesh we use for learning and (b) the venus mesh. Both illustrations are created semi-automatically by applying the interaction methods described in Section 6. Contours are curvature-controlled image-space contours as proposed by Bruckner and Gröller [45], but any other silhouette technique [46] could be chosen.

Both Fig. 10 and Fig. 11 show results of transferring the hatching style learned from the shoulder blade illustration in Fig. 3(a). While creating Fig. 10(a), we aimed to match the example illustration (Fig. 3(a)). We also transfer the learned hatching style to new objects. The venus illustration in Fig. 10(b), the vertebra illustration Fig. 11(a), and the hip bone illustration in Fig. 11(b) demonstrate that we can successfully transfer the learned hatching style to different target meshes.

For comparison, we also apply our method to the hatching style of a different illustrator. Fig. 12 shows results of transferring the hatching style learned from Fig. 13(a) to various objects. Fig. 13(a) shows this second example illustration. It is a hand-drawn pen-and-ink illustration of of a carnivorous pitcher plant which was created for a previous study of Isenberg et al. [1]. Together with the segmentation image in Fig. 13(b) and the registered 3D model in Fig. 13(c), we use the pitcher plant illustration as input for learning a second hatching style.

The results show that many characteristics of the example illustration styles are successfully reproduced. For example, patterns of hatching strokes within the patches of strokes are reproduced. These patterns emerge from both the directions and the distances of strokes. Regarding the directions, patterns emerge from the quasi-parallel trajectories of the strokes and the way individual strokes deviate from these common directions. Pat-



(a) (b)

Figure 11: Illustrations using the hatching style learned from the shoulder blade illustration in Fig. 3(a). The depicted objects are (a) a vertebra and (b) a hip bone.

terns with respect to the distances emerge from the sequences of distances between the strokes. This applies to both the sequences of distances between strokes within a patch and the sequences of distances along the extents of neighboring strokes. The latter successfully transfer patterns of the relationship between neighboring strokes, meaning the way in which neighboring strokes approach and deviate from each other. Furthermore, the directions of the resulting hatching strokes on the surface appear to be similar to the strokes' surface directions in the example image. The example-based direction field inference does successfully reproduce the way in which strokes are following the surface. Using this direction field to trace strokes on the surface thus results in hatching strokes that visually model the depicted surface in similar ways as the strokes in the example illustration. Furthermore, the interplay of groups of different stroke types helps reproducing the visual appearance of the example illustrations. This effect is supported by our stroke rendering approach, which simulates real pen-and-ink marks to a certain extent. However, not all of the characteristics can be faithfully reproduced. The overall appearance of our results still shows differences between the originals and the synthesized illustrations. We elaborate on this in Section 8.

When comparing our results to the results of Kalogerakis et al. [2], we make the following observation. Three of their example styles use quite regular styles (Fig. 7–9 in [2]), while two example styles use more irregular styles (Fig. 6 and 10 in [2]). We observe that the irregular styles are not transferred as faithfully as the uniform styles. In particular local characteristics of the irregular styles, such as the relationship of neighboring strokes, cannot be reproduced that well. The uniform hatching styles, however, are reproduced very accurately. The overall appearance of the synthesized hatchings visually match the example illustrations impressively well. We do not achieve the same transfer accuracy with our automatic method. The described observation suggests that uniform and regular drawing styles can more easily be reproduced than irregular and complex styles. We now observe that the drawing styles that Kalogerakis et al. [2] employ are much more uniform and regular than the drawing styles we try to reproduce. The strokes in our example illustrations are much more complex and irregular, which makes it all the more difficult to faithfully transfer the drawing styles. Although we might not reach the same transfer accuracy as Kalogerakis et al. [2], we can successfully transfer some characteristics of the complex illustration style. In particular, local characteristics , such as the distance relationship between neighboring strokes, can be transferred more successfully with our model. Furthermore, the explicit handling of groups of strokes allows us to transfer drawing characteristics that are embodied in patches of strokes and in the relationships of these patches.

Apart from that, Kalogerakis et al. [2] use a vast set of geometric features, which is another reason that their style transfer model can more accurately reproduce the overall appearance of the example styles. Using such many features, however, severely affects the performance of their algorithm. Kalogerakis et al. [2] name 5 to 10 hours learning time and 30 to 60 minutes synthesis time on an Intel Core i7 processor. Our method takes 1 to 2 minutes for learning and 0.4 to 30 seconds for synthesis on an Intel Core 2 Duo processor. As a pixel-based approach, the performance of their method depends on the resolution of the result image, while the performance of our method is virtually independent of the output resolution.

Another advantage of our approach over that by Kalogerakis et al. [2] is the possibility for interaction. Our interactions permit users to adjust the illustration in order to achieve more aes-

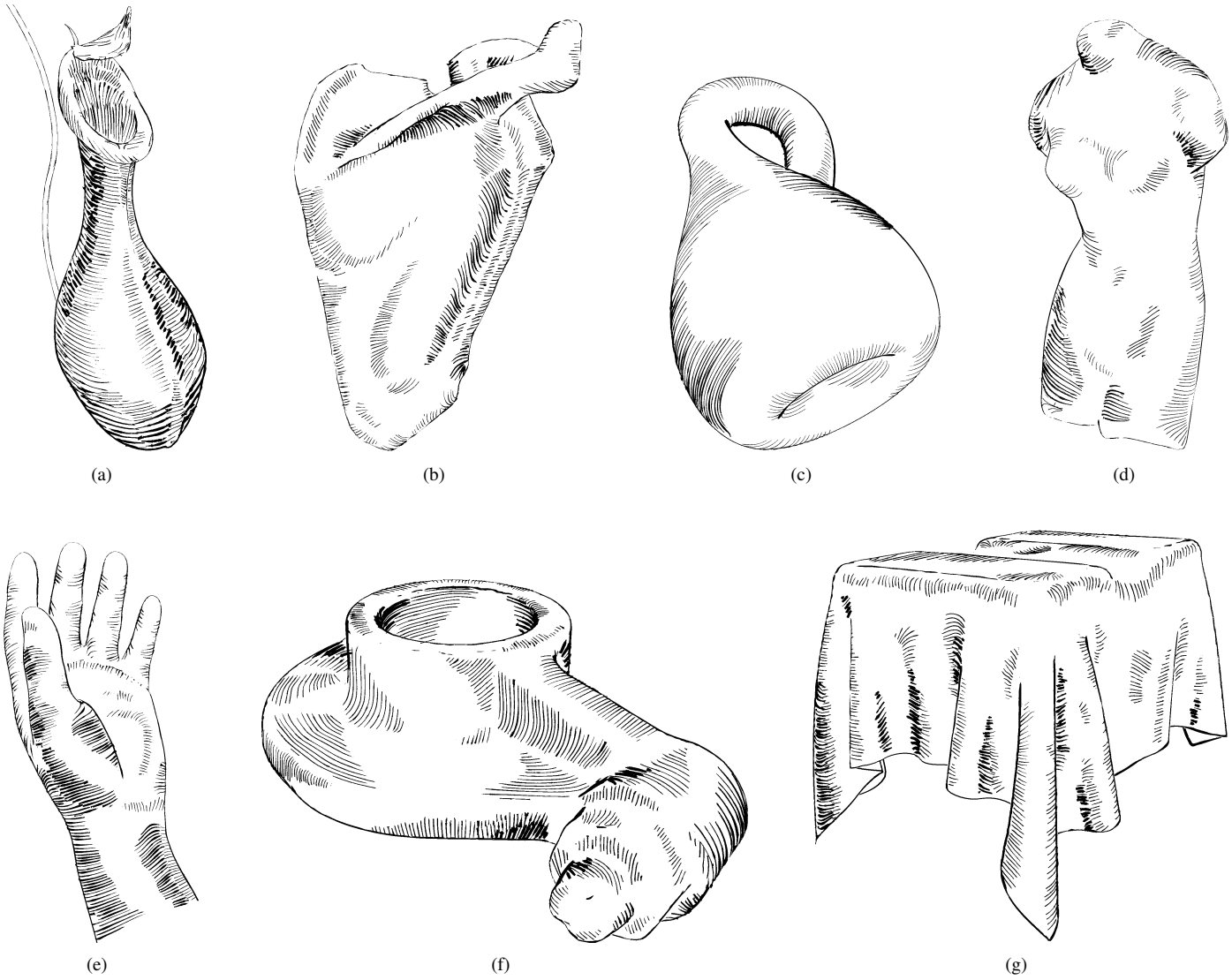


Figure 12: Illustrations using the hatching style learned from the pitcher plant illustration shown in Fig. 13(a). The depicted objects are (a) a pitcher plant (intended to match Fig. 13(a)), (b) a shoulder blade, (c) a Klein bottle, (d) a venus statue, (e) a hand, (f) a rocker arm, and (g) a two box cloth.

thetically pleasing results. Such adjustments are used, e. g., to reshape large continuous hatching regions, to add strokes to blank regions, or to make use of the aesthetics of combining different patches of strokes. The fully automatic approach of Kalogerakis et al. [2] does not facilitate such adjustments. Our method, in contrast, can serve as the basis for a tool for artists and illustrators due to its interaction capabilities.

Together, the differences between our approach and that of Kalogerakis et al. [2] can be summarized as the following: Kalogerakis et al. [2] present a fully automatic approach with a high style transfer accuracy for uniform hatching styles. We, in contrast, present a semi-automatic approach with a lower transfer accuracy for complex hatching styles which has the capability to enhance the results via user interactions.

We make a step towards the generation of hatching images that incorporate human virtuosity and illustration skills. Even if our approach cannot capture and reproduce all the stylistic properties of the example illustrations, it does reproduce many of their characteristics. The interaction capabilities of our approach facilitate a further enhancement of the hand-drawn appearance and allow the results to be influenced by human creativity and virtuosity. For these reasons, we gain result images that look less synthetic and less uniform, and arguably exhibit more 'character' than the results of previous methods, which are either not example-based or not interactive.

Our patch-based approach to hatching can also easily be extended to achieve crosshatching. We realize this by adding another layer of hatching patches which use stroke directions at a user-controllable angle to the inferred stroke directions. Although our target illustration style does not use crosshatching, the illustrations resulting from this extension still have a certain aesthetic appeal. Fig. 14 shows an example of a crosshatching illustration achieved in this way.

We showed our results to two professional medical illustrators to gain informal user feedback. Both illustrators were impressed by the aesthetic quality of our illustrations. One of

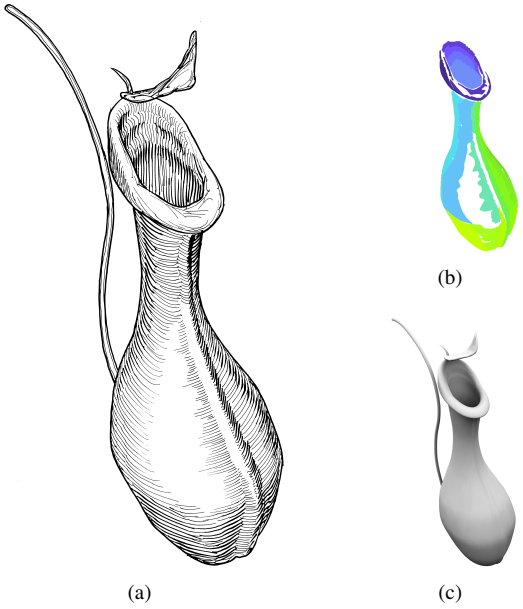


Figure 13: (a) A second example illustration and the corresponding (b) segmentation image and (c) registered 3D model. It shows a pitcher plant's trap.

the illustrators described our results to have a *"lively appearance and not stiff like a digital feeling."* The other illustrator commented positively that our *"method can produce good 'pen&ink' illustrations in far less time than a hand drawn illustration."* One of the illustrators informed us that she would be interested in working with a system such as ours and that she lacked comparable functionality in the software she is using. Furthermore, both illustrators stated that the manual creation of pen-and-ink hatchings similar to ours is so tedious and time-consuming that the required time prohibits them to create such illustrations for customers. The illustrators appreciated the possibility to create hatching illustrations with the computer while still having interactive control over the result.

8. Limitations

The results presented in the previous section show that we can successfully capture and reproduce characteristics learned from a hand-drawn illustration. In particular, we can faithfully reproduce local characteristics such as the distance relationship between neighboring strokes along their extent. In combination with the interaction methods, this style transfer offers new possibilities for creating pen-and-ink hatching renderings. The results also show, however, that our style transfer model suffers from certain accuracy issues. One reason that our learning methods do not fully capture the example style is that we use a limited number of surface features, as discussed in the previous section. Another problem is that our learning approach suffers from overfitting. We use just a single example illustration for learning a hatching style. All our drawing style descriptors are thus conditioned to one specific setup of viewing, lighting, and geometry which holds for this single illustration. This overfitting results in the problem that the hatching properties we infer for viewing and lighting situations other than the training setup as well as for other shapes do not match the hatching properties found in the example drawing. The described overfitting problem has most negative impact on the globally learned patch properties, which is apparent in Fig. 6. The surface patches inferred for the training setup match the regions of the segmentation image very well. For other viewing and lighting situations, however, the patches do not always match the regions which we assume the creator of the example illustration would have used. This is one of the major reasons that our approach is not as highly accurate as the approach of Kalogerakis et al. [2] with respect to a fully automatic style transfer. We handle this limitation regarding the patches with the brushing interaction presented in Section 6. The described overfitting problem could be tackled by using more extensive training data, i. e., to learn an illustrator's hatching style from a multitude of illustrations of different objects. The preparation of the segmentation image and the registered 3D model, however, is quite labor-intensive. This requirement for a manual creation of prerequisites of the learning procedures is another drawback of our method.

Furthermore, our stroke rendering method exhibits certain problems. First, for some strokes it is inevitable to erroneously sample black color from a neighboring stroke or from the contour, resulting in unwanted artifacts. To avoid this, we simply omit these strokes for rendering. Second, distortion effects can appear when an example stroke is used for texturing a stroke whose trajectory strongly deviates from the trajectory of the example stroke. Our stroke rendering, therefore, does not always yield satisfactory quality. Better results could be achieved with extracting a set of representative stroke textures from the example illustration, where each texture contains one separate stroke.

A limitation of our image processing procedure is that it is restricted to example images with separated individual strokes. It fails in detecting the strokes in hatching images with many overlapping strokes. More elaborate image analysis would be necessary to detect the strokes in such imagery.

Another limitation of our approach is that both the speed of the automatic style transfer as well as the interaction granularity depend on the mesh resolution. The performance of inferring hatching patches and directions at the vertices is linearly proportional to the number of vertices. We thus have to work with low-resolution meshes to facilitate a reasonably fast inference of these two properties when using our system for real-time animation. Using a mesh of 20k triangles, we can infer the two properties at 3 fps on an Intel Core 2 Duo. This low polygon count affects the quality of the result with respect to depicting surface detail. A benefit of hatching low-polygon models, however, is that the interpolations during the generation of strokes create an impression of smooth shapes for rather blocky meshes. While the speed of the fully automatic synthesis benefits from a low mesh resolution, the brushing interactions benefit from a high resolution. A higher resolution enables the user to adjust hatching patches and directions with a finer granularity. This fine-grained control improves the users' creative freedom and editing possibilities. In an interactive setting, a higher mesh resolution is thus desirable, and we here worked with resolutions from 35k to 90k triangles. This mesh resolution does not hinder

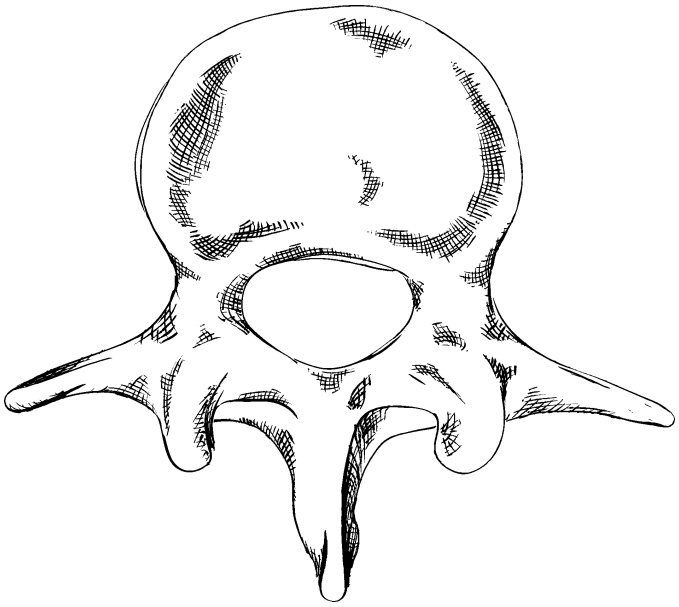


Figure 14: Crosshatching illustration of a vertebra using the hatching style learned from the shoulder blade illustration in Fig. 3(a).

a responsive editing because the inference of hatching patches is turned off in this interactive setting and because the stroke directions are only inferred at the currently edited vertices.

In the process of developing our approach, we learned that we can automatically transfer local characteristics, such as varying distances along a stroke or the local stroke directions, more reliably than global characteristics, such as regions where to place strokes. While experimenting with different ways of capturing the local characteristics, we found out that simple measuring and re-applying the measured values was not sufficient and that we need machine learning techniques to capture these properties. We also learned about the granularity at which specific characteristics can be transferred and which data is required. For example, we first measured the stroke directions only locally at each reprojected stroke control point and tried to capture it only as the measured angle in which it deviates from the curvature direction. We learned that this strategy was not robust enough. First, it was not robust enough because the granularity was too fine: we could not robustly map the direction measured at only one location on the surface to another location. Second, it was not robust enough because the data involved was too little: we could not robustly map the deviation from the curvature at one location to the deviation at another location using only the curvature direction. For these reasons, we now transfer the stroke directions on a coarser granularity (as a patch-wise direction field) and involve more data (all surface features that we also use for the stroke regions).

9. Conclusions and Future Work

In summary, we propose a novel approach for the interactive example-based generation of pen-and-ink hatching illustrations from 3D meshes. We present a new learning setup that makes it possible to learn the depiction style of a hand-drawn example image, including a way to infer 3D information related to the 2D example image. We propose an analytical representation of hatching patches and hatching strokes. This representation of drawing elements is coupled with an hierarchical style transfer model that captures rendering properties on four levels of abstraction. We introduce adaptive surface patches that incorporate global drawing characteristics and which can be used for the creation of hatching strokes in object space. We present ways to capture and reproduce the directional characteristics on a patch level and the distance characteristics of hatching strokes locally on a stroke level. Finally, we provide novel interaction methods that allow users to directly and intuitively modify the resulting illustration. This interaction capability improves upon the static nature of previous methods.

We can successfully reproduce some of the characteristics of the example illustrations. Our method can transfer directional characteristics with the help of an example-based patch-wise stroke direction field. Our method can also reproduce patterns of patch-wise and local stroke distance relationships, which results in hatching strokes that are less uniform and less regular than the hatching strokes generated by existing methods. The transfer of global characteristics incorporated in the surface patches has some limitations. Therefore, our method does not produce as highly accurate results as the method of Kalogerakis et al. [2] with respect to a fully automatic style transfer. We propose possible ways of improving upon this limitation. We also provide a brushing interaction that copes with this problem and that gives direct and intuitive control over hatching regions to the users of our system. Together, the proposed methods allow us to generate computer hatchings that arguably exhibit more 'character' than the results of previous techniques.

So far, we judge upon the quality of the results only by subjective reasoning. It would require more extensive evaluation to be able to judge upon the results less subjectively. One could envision a statistical assessment of the hatching properties as done for stippling by example [16]. But it would be even more interesting to conduct a field experiment designed as sort of a visual Turing test as proposed by Salesin [47], reviewed by Gooch et al. [48], and performed to a certain extent by Isenberg et al. [1]. Showing the participants a set of hand-drawn and computer-generated illustrations, one would ask the participants whether the images were drawn by hand or generated by an algorithm. In this way one could examine how successfully the hand-drawing characteristics can be reproduced, i. e., how well the human rendering process can be simulated, although the validity of such a visual Turing test is debatable [49, 50]. Furthermore, it would be interesting to quantitatively evaluate the accuracy of our automatic style transfer approach by applying our method to an example illustration and 3D model used by Kalogerakis et al. [2] and comparing the results statistically.

The possibility to brush with patches of hatching strokes onto a 3D model is a novel way of interacting with hatching renderings. It allows users to directly specify the desired hatching regions. The brushing interaction is very effective and pleasant to work with. The possibility for adjusting the illustration contrasts the static results of many comparable techniques. This creative freedom offered by our semi-automatic example-based

approach makes it more likely that it is employed in a creative environment than it is likely for a fully automatic and static approach. We believe the proposed brushing interaction has the potential to serve as the basis for a tool for artists and illustrators. For future work, it would be interesting to further explore the interactive creation of hatching illustrations. The brushing of hatching patches can be extended with brushing metaphors to modify stroke properties such as distance, randomness, width, shape, etc. High-level brushes can be used for accentuation and abstraction as well as for modifying material properties. Integrating a layer support for multiple layers of hatching strokes can allow users to achieve various hatching effects. A design gallery showing previews for the result of using different example stroke patches can assist users to conveniently select the type of hatching patch to brush with.

Furthermore, we believe that the notion of explicitly represented dynamic patches embedded on the surface can be generalized to other illustrative rendering styles. Many methods for stylized rendering create such patches implicitly in image space, e. g., regions of homogeneous shading in cartoon rendering. An explicit patch representation, however, provides control over the type and location of patches. This control can be used by automatic methods, such as our automatic prediction of patches. And the control can also be employed for realizing user interactions with the illustration, such as our brushing tools. We are convinced that the notion of adaptive surface patches has great potential for future developments within illustrative rendering.

10. Acknowledgments

We thank the artists and illustrators for their valuable feedback on our project. We also thank the Aim@Shape, VAKHUN, Google, and Polhemus repositories as well as the Princeton Graphics Group for the 3D models we used. We also thank the developers of dlib, CGAL, and VolumeShop for their software.

References

[1] Isenberg, T., Neumann, P., Carpendale, S., Sousa, M.C., Jorge, J.A.. Non-Photorealistic Rendering in Context: An Observational Study. In: Proc. NPAR. New York: ACM; 2006, p. 115–126. doi> 10.1145/1124728.1124747

[2] Kalogerakis, E., Nowrouzezahrai, D., Breslav, S., Hertzmann, A.. Learning Hatching for Pen-and-Ink Illustration of Surfaces. ACM Transactions on Graphics 2012;31(1):Article No. 1. doi> 10.1145/2077341.2077342

[3] Saito, T., Takahashi, T.. Comprehensible Rendering of 3-D Shapes. In: Proc. SIGGRAPH. New York: ACM; 1990, p. 197–206. doi> 10.1145/97880.97901

[4] Elber, G.. Line Art Rendering via a Coverage of Isoparametric Curves. IEEE Transactions on Visualization and Computer Graphics 1995;1(3):231–239. doi> 10.1109/2945.466718

[5] Winkenbach, G.A., Salesin, D.H.. Rendering Parametric Surfaces in Pen and Ink. In: Proc. SIGGRAPH. New York: ACM; 1996, p. 469–476. doi> 10.1145/237170.237287

[6] Winkenbach, G.A., Salesin, D.H.. Computer-Generated Pen-and-Ink Illustration. In: Proc. SIGGRAPH. New York: ACM; 1994, p. 91–100. doi> 10.1145/192161.192184

[7] Girshick, A., Interrante, V., Haker, S., Lemoine, T.. Line Direction Matters: An Argument for the Use of Principal Directions in 3D Line Drawings. In: Proc. NPAR. New York: ACM; 2000, p. 43–52. doi> 10.1145/340916.340922

[8] Hertzmann, A., Zorin, D.. Illustrating Smooth Surfaces. In: Proc. SIGGRAPH. New York: ACM; 2000, p. 517–526. doi> 10.1145/344779.345074

[9] Zander, J., Isenberg, T., Schlechtweg, S., Strothotte, T.. High Quality Hatching. Computer Graphics Forum 2004;23(3):421–430. doi> 10.1111/j.1467-8659.2004.00773.x

[10] Praun, E., Hoppe, H., Webb, M., Finkelstein, A.. Real-Time Hatching. In: Proc. SIGGRAPH. New York: ACM; 2001, p. 581–586. doi> 10.1145/383259.383328

[11] Kim, Y., Yu, J., Yu, X., Lee, S.. Line-art Illustration of Dynamic and Specular Surfaces. ACM Transactions on Graphics 2008;27(5):Article No. 156. doi> 10.1145/1457515.1409109

[12] Hamel, J., Strothotte, T.. Capturing and Re-Using Rendition Styles for Non-Photorealistic Rendering. Computer Graphics Forum 1999;18(3):173–182. doi> 10.1111/1467-8659.00338

[13] Mertens, T., Kautz, J., Chen, J., Bekaert, P., Durand, F.. Texture Transfer using Geometry Correlation. In: Proc. EGSR. Goslar, Germany: Eurographics Association; 2006, p. 273–284. doi> 10.2312/EGWR/EGSR06/273-284

[14] Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D.H.. Image Analogies. In: Proc. SIGGRAPH. New York: ACM; 2001, p. 327–340. doi> 10.1145/383259.383295

[15] Zhao, M., Zhu, S.C.. Portrait Painting using Active Templates. In: Proc. NPAR. New York: ACM; 2011, p. 117–124. doi> 10.1145/2024676.2024696

[16] Kim, S., Maciejewski, R., Isenberg, T., Andrews, W.M., Chen, W., Sousa, M.C., et al. Stippling by Example. In: Proc. NPAR. New York: ACM; 2009, p. 41–50. doi> 10.1145/1572614.1572622

[17] Martín, D., Arroyo, G., Luzón, M.V., Isenberg, T.. Scale-Dependent and Example-Based Stippling. Computers & Graphics 2011;35(1):160–174. doi> 10.1016/j.cag.2010.11.006

[18] Jodoin, P.M., Epstein, E., Granger-Piché, M., Ostromoukhov, V.. Hatching by Example: a Statistical Approach. In: Proc. NPAR. New York: ACM; 2002, p. 29–36. doi> 10.1145/508530.508536

[19] Barla, P., Breslav, S., Thollot, J., Sillion, F.X., Markosian, L.. Stroke Pattern Analysis and Synthesis. Computer Graphics Forum 2006;25(3):663–671. doi> 10.1111/j.1467-8659.2006.00986.x

[20] Freeman, W.T., Tenenbaum, J.B., Pasztor, E.. An Example-Based Approach to Style Translation for Line Drawings. Tech. Rep. TR-99-11; MERL – A Mitsubishi Electric Research Laboratory; 1999.

[21] Freeman, W.T., Tenenbaum, J.B., Pasztor, E.C.. Learning Style Translation for the Lines of a Drawing. ACM Transactions on Graphics 2003;22(1):Article No. 2. doi> 10.1145/588272.588277

[22] Hertzmann, A., Oliver, N., Curles, B., Seitz, S.M.. Curve Analogies. In: Proc. EGWR. Goslar, Germany: Eurographics Association; 2002, p. 233–246. doi> 10.1145/581896.581926

[23] Kalnins, R.D., Markosian, L., Meier, B.J., Kowalski, M.A., Lee, J.C., Davidson, P.L., et al. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In: Proc. SIGGRAPH. New York: ACM; 2002, p. 755–762. doi> 10.1145/566654.566648

[24] Lum, E.B., Ma, K.L.. Expressive Line Selection by Example. The Visual Computer 2005;21(8–10):811–820. doi> 10.1007/s00371-005-0342-y

[25] Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H.S., Finkelstein, A., Funkhouser, T., et al. Where Do People Draw Lines? ACM Transactions on Graphics 2008;27(3):Article No. 88. doi> 10.1145/1360612.1360687

[26] Seims, J.. Putting the Artist in the Loop. ACM SIGGRAPH Computer Graphics 1999;33(1):52–53. doi> 10.1145/563666.563685

[27] Salisbury, M.P., Anderson, S.E., Barzel, R., Salesin, D.H.. Interactive Pen-and-Ink Illustration. In: Proc. SIGGRAPH. New York: ACM; 1994, p. 101–108. doi> 10.1145/192161.192185

[28] Salisbury, M.P., Wong, M.T., Hughes, J.F., Salesin, D.H.. Orientable Textures for Image-Based Pen-and-Ink Illustration. In: Proc. SIGGRAPH. New York: ACM; 1997, p. 401–406. doi> 10.1145/258734.258890

[29] Deussen, O., Hiller, S., van Overveld, C., Strothotte, T.. Floating Points: A Method for Computing Stipple Drawings. Computer Graphics Forum 2000;19(3):40–51. doi> 10.1111/1467-8659.00396

[30] Rössl, C., Kobbelt, L.. Line Art Rendering of 3D-Models. In: Proc. Pacific Graphics. Los Alamitos: IEEE Computer Society; 2000, p. 87–96. doi> 10.1109/PCCGA.2000.883890
[31] Kalogerakis, E., Hertzmann, A., Singh, K.. Learning 3D Mesh Segmentation and Labeling. ACM Transactions on Graphics 2010;29(4):Article No. 102. doi> 10.1145/1778765.1778839
[32] Anjyo, K.i., Wemler, S., Baxter, W.. Tweakable Light and Shade for Cartoon Animation. In: Proc. NPAR. New York: ACM; 2006, p. 133–139. doi> 10.1145/1124728.1124750
[33] Todo, H., Anjyo, K.i., Baxter, W., Igarashi, T.. Locally Controllable Stylized Shading. ACM Transactions on Graphics 2007;26(3):Article No. 17. doi> 10.1145/1275808.1276399
[34] Breslav, S., Szerszen, K., Markosian, L., Barla, P., Thollot, J.. Dynamic 2D Patterns for Shading 3D Scenes. ACM Transactions on Graphics 2007;26(3):Article No. 20. doi> 10.1145/1275808.1276402
[35] Dauber, W., Spitzer, G., Feneis, H.. Feneis' Bild-Lexikon der Anatomie. Georg Thieme Verlag; 9th ed.; 2005. ISBN 3-13-330109-8.
[36] Soille, P.. Morphological Image Analysis: Principles and Applications. Berlin/Heidelberg: Springer-Verlag; 2nd ed.; 2003. ISBN 3540429883.
[37] Hastie, T., Tibshirani, R.. Classification by Pairwise Coupling. In: Proc. NIPS. Cambridge, MA, USA: MIT Press; 1998, p. 507–513. doi> 10.1214/aos/1028144844
[38] Tipping, M.E., Faul, A., Avenue, J.J.T., Avenue, J.J.T.. Fast Marginal Likelihood Maximisation for Sparse Bayesian Models. In: Proc. Artificial Intelligence and Statistics. Key West, Florida: Society for Artificial Intelligence and Statistics; 2003, p. 3–6.
[39] Ritschel, T., Grosch, T., Seidel, H.P.. Approximating Dynamic Global Illumination in Image Space. In: Proc. I3D. New York: ACM; 2009, p. 75–82. doi> 10.1145/1507149.1507161
[40] Hodges, E.R.S.. The Guild Handbook of Scientific Illustration. John Wiley; 2nd ed.; 2003. ISBN 9780471360117.
[41] Hoerl, A.E., Kennard, R.W.. Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics 1970;12(1):69–82. doi> 10.2307/1271436
[42] King, D.E.. Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research 2009;10:1755–1758.
[43] Bischoff, S., Wey, T., Kobbelt, L.. Snakes on Triangle Meshes. In: Bildverarbeitung für die Medizin. Berlin/Heidelberg: Springer-Verlag; 2005, p. 208–212. doi> 10.1007/3-540-26431-0_43
[44] Jobard, B., Lefer, W.. Creating Evenly-Spaced Streamlines of Arbitrary Density. In: Proc. VisSci. Berlin/Heidelberg: Springer-Verlag; 1997, p. 45–55.
[45] Bruckner, S., Gröller, M.E.. Style Transfer Functions for Illustrative Volume Rendering. Computer Graphics Forum 2007;26(3):715–724. doi> 10.1111/j.1467-8659.2007.01095.x
[46] Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., Strothotte, T.. A Developer's Guide to Silhouette Algorithms for Polygonal Models. IEEE Computer Graphics and Applications 2003;23(4):28–37. doi> 10.1109/MCG.2003.1210862
[47] Salesin, D.H.. Non-Photorealistic Animation & Rendering: 7 Grand Challenges. Keynote talk at NPAR; 2002.
[48] Gooch, A.A., Long, J., Ji, L., Estey, A., Gooch, B.S.. Viewing Progress in Non-Photorealistic Rendering Through Heinlein's Lens. In: Proc. NPAR. New York: ACM; 2010, p. 165–171. doi> 10.1145/1809939.1809959
[49] Isenberg, T.. Evaluating and Validating Non-Photorealistic and Illustrative Rendering. In: Rosin, P., Collomosse, J., editors. Image and Video based Artistic Stylisation, volume 42 of Computational Imaging and Vision; chap. 15. London/Heidelberg: Springer-Verlag; 2013, p. 311–331. doi> 10.1007/978-1-4471-4519-6_15
[50] Hall, P., Lehmann, A.S.. Don't Measure—Appreciate! NPR Seen through the Prism of Art History. In: Rosin, P., Collomosse, J., editors. Image and Video based Artistic Stylisation, volume 42 of Computational Imaging and Vision; chap. 16. London/Heidelberg: Springer-Verlag; 2013, p. 333–351. doi> 10.1007/978-1-4471-4519-6_16
[51] Ruberto, C.D.. Recognition of Shapes by Attributed Skeletal Graphs. Pattern Recognition 2004;37(1):21–31. doi> 10.1016/j.patcog.2003.07.004

Appendix A. Stroke Detection

We used Matlab® to implement the stroke detection described in Section 4.1. We employ the following commands for morphological cleaning, connected component labeling, thinning, and detection of skeleton endpoints and junctions [51]:

```
Source = bwmorph(Source,'clean',Inf);
[Boundaries, Labels] = bwboundaries(Source,'noholes');
Skeletons = bwmorph(Source,'thin',Inf);
Endpoints = BOHitOrMiss(Skeletons,'end');
Junctions = BOHitOrMiss(Skeletons,'triple');
```

After running these commands for each separated patch of example strokes, we use the number of endpoints and junctions to differentiate between different stroke types. For stroke types consisting of overlapping strokes, we discriminate the overlapping strokes from each other by starting at the longest skeleton segments and searching for adjacent shorter skeleton segments that are oriented in the same direction as the long segments. We omit small strokes whose area is below 5% of the average stroke region within a patch or whose skeleton length is below 50 pixels. During vectorizing the stroke trajectories, we generate control points in a distance of 25 pixels along the stroke skeletons for an input image of 3670×7360 pixels (Fig. 13(a)).

Appendix B. Learning Parameters

We use a stopping epsilon of $\varepsilon = 0.001$ for the multiclass classifier in our learning procedures. In all of our learning routines, we use a gamma of $\gamma = 0.08$ for the radial basis function kernels. Table B.1 shows the feature weights we use for learning, listed in the same order as the features are named in Section 4.2.

Feature	Regions	Directions	Distances
$\lvert\kappa_1\rvert$	1.0	2.0	1.0
$\lvert\kappa_2\rvert$	1.0	2.0	1.0
$\lvert\kappa_1\rvert/\lvert\kappa_2\rvert$	1.0	2.0	1.0
λ_{1_x}	2.0	7.0	2.5
λ_{1_y}	2.0	7.0	2.5
λ_{1_z}	2.0	7.0	2.5
I	5.0	1.5	4.0
$SSDO$	4.0	1.5	4.0
$n \cdot v$	2.5	2.0	3.0
$\lvert\nabla(n \cdot v)\rvert$	1.5	2.0	3.0
$(\nabla(n \cdot v)) \cdot v$	1.5	2.0	2.0
$\lambda_1 \cdot v$	2.0	5.0	2.0
$\lambda_2 \cdot v$	2.0	5.0	2.0
z	2.3	1.0	4.0
x_i	2.3	1.0	4.0
y_i	1.5	1.0	4.0
n_{i_x}	2.0	4.0	2.5
n_{i_y}	2.0	4.0	2.5

Table B.1: Feature weights. κ_1 and κ_2 are the first and second principal curvature directions, λ_1 and λ_2 are the first and second optimized principal curvature directions according to Hertzmann and Zorin [8] (λ_{1_x} is the x-component of λ_1), I is the diffuse illumination (Lambertian shading), $SSDO$ is the screen-space directional occlusion [39], n is the surface normal, v is the viewing direction, ∇ is the gradient, z is the depth, x_i and y_i are the image-space coordinates, n_{i_x} and n_{i_y} are the x- and y-components of the image-space normal.