Procedural Generation of Hand-drawn like Line Art

Tamás Umenhoffer, Milán Magdics, and Károly Zsolnai

BME IIT, Hungary
umenhoffer@iit.bme.hu

Abstract. This paper presents a high quality Non-Photorealistic Rendering (NPR) framework. Our system focuses on artistic line drawing generation and supports different contour and hatching line renderings. All these lines are represented as individual spline primitives and rendered as textured triangle strips. This paper mainly focuses on one powerful feature of our system, namely stroke style synthesis. To give the user a high degree of freedom in stylization, a sample stroke pattern can be defined for each artistic line type and the style of these samples will be transferred to all other individual lines. The samples can be given as images and will be converted into an offset list. We use random Markov fields and Q-learning to obtain new offset lists that can be used to perturb our line primitives.

1 Introduction

Photo-realism has been in the focus of rendering for decades. Photo-realistic rendering aims at creating images that are indistinguishable from real-world photographs, which is made possible by the precise simulation of physics laws — e.g. the Maxwell equations — during the rendering process [19]. The level of accuracy of the representation of physics in the rendering code determines the level of realism of the result.

Computer graphics also tries to mimic artistic expression and illustration styles [4, 3, 14, 18]. Such methods are usually vaguely classified as *non photo-realistic rendering* (NPR). While the fundamentals of photo-realistic rendering are in optics that are well understood, NPR systems simulate artistic behavior that is not mathematically founded and often seems to be unpredictable. Therefore, the first step of NPR is to model the artist establishing a mathematical model describing his style, and then solve this model with the computer. The result will be acceptable if our model is close to the not formally specified artistic behavior. During the history of NPR, many individual styles were simulated. In order to exploit their potential, these algorithms should be integrated in a complete system, and more importantly, their flexibility should be increased [22, 2, 11]. A critical problem of rendering is that while frames are calculated independently the image sequence should not present flickering or inconsistency [13, 9, 7]. The rendering process should resolve this contradiction while presenting natural randomness inherent in manual work [5, 1].

This paper presents an automatic method to generate strokes for line art illustration, e.g. for contour or hatch lines. In order to provide natural randomness, we build a Markov model from sample strokes, and use this model to generate automatic stokes. The generated strokes will be similar to the sample strokes in a statistical sense, but they are not periodic and might have arbitrary length.

2 Previous work

Painterly rendering covers the image by brush stokes to produce a painted look. Brush strokes may be fully random, sampled according to only the 2D image [9] or using also 3D geometry [13]. The size and the texture of the brushes can vary to simulate artist work who details only those objects that are put in the focus of the image.

Line art illustration or pen-and-ink rendering uses textured line primitives to describe the scene. These line primitives can emphasize object boundaries, called silhouettes, and places where there is an abrupt change in surface geometry, e.g. in ridges or contours.

Natural line stokes can be produced using hand made examples, as proposed in [8], which is inspired by video textures [17, 16, 15].

In this paper we generalize the Markov process method proposed for video textures and make it appropriate for animation generation. By simplifying the original approach we provide interactive control and by pre-generating curves, we eliminate flickering in animation sequences. The developed algorithm is built into a post-production sofware.

3 Contour and hatch line generation

Contours depict the boundary of the visible shape of an object. The silhouette separates the object from its environment, internal contours emphasize internal features. Contours have an important role in presenting the shape of the object. Contours show up in hand-made drawings, pen-and-ink illustrations, and may be added to paintings as well.

Contours can be created in the rendered image using edge detection filters. Since we wish to emphasize object boundaries, edge detection is worth executing on the depth image rather than the color buffer. A significant drawback of this approach is that the resulting contours are rather noisy and are made of independent points and not as a well defined curve. Thus, it is very difficult to assign a specific drawing style to the contour line. Fortunately, contours can also be generated from the 3D object geometry. Silhouette curves contain points where the surface normal is perpendicular to the viewing direction. We can also distinguish interior silhouettes from exterior silhouettes that give the outline of the objects seen from the camera. Ridges and valleys are defined by surface points where the surface normal changes abruptly.

To display contours we should extract them from the geometry, which is defined by a triangle mesh. Silhouette contours can be defined as the edges that

share a front and a back facing triangle. However this definition leads to a collection of line segments that should be combined with a set of heuristics into long smooth paths. Another solution is to reconstruct the surface normal of the original, not tessellated surface from the discrete surface representation. First, the normal vector is estimated at the triangle vertices as a weighted average of the triangle normals. We use the angle of the triangles at this vertex as weights. Then, the reconstructed normal inside the triangles is obtained as the bi-linear interpolation of the reconstructed vertex normals. Taking the reconstructed normal, first we search for silhouette points on the edges of the triangles by checking when the normal becomes perpendicular to the viewing direction (the zero crossing of the scalar product of these two vectors). As these lines will enter and exit at two of the edges of a triangle, continuous silhouette paths can be easily extracted. We can smooth these lines to better approximate the original surface. Ridge and valley curves are extracted similarly, they are defined by the zero crossing of the derivative of the surface curvatures.

Contours found so far must be processed further and cut those parts that are not visible from the camera due to occlusions or being outside of the camera frustum. To execute these operations, curves are vectorized and clipped onto the view frustum, then rasterized and the visibility is checked in every pixel. We can use depth buffer test or object and face ID test for visibility checking. If the curve turns out to be invisible in a pixel, the curve is cut here.

Hatch lines can be rendered similarly to contours. Seed points are generated with a density that mimics local shade [21]. Then hatch lines are extruded into the main curvature directions.

4 Procedural repetition of hand-drawn patterns

NPR produces the final image as a set of line or curve primitives. These primitives must look natural and similar to curves drawn by a human artist. To achieve this goal, we take curve samples drawn by humans, analyze them statistically, then use the statistical model to generate curves of arbitrary lengths that are similar to the samples in a statistical sense. Our underlying statistical model is the Markovian process [8]. During model building, this process is defined from the information of the sample curves. A Markovian process is unambiguously defined by the state transition probabilities, so in the first phase, we use Q-learning to build the state transition probability matrices from the examples. Then, in the generation phase, the probability matrices are exploited to generate curves that have similar statistical properties than the original example curves.

In order to interpret a curve as a finite state discrete time stochastic process, we sample it regularly and the index of the sequence is interpreted as the discrete time variable, while the sampled function value, i.e. the offset from a smooth curve, is the discrete state.

Denoting the function value at index i by y_i, the state space of the process is $y_1, \ldots, y_k$ where k is the length of the stroke. The original curve can be imagined as a Markovian process with probability 1 transitions (Figure 1).

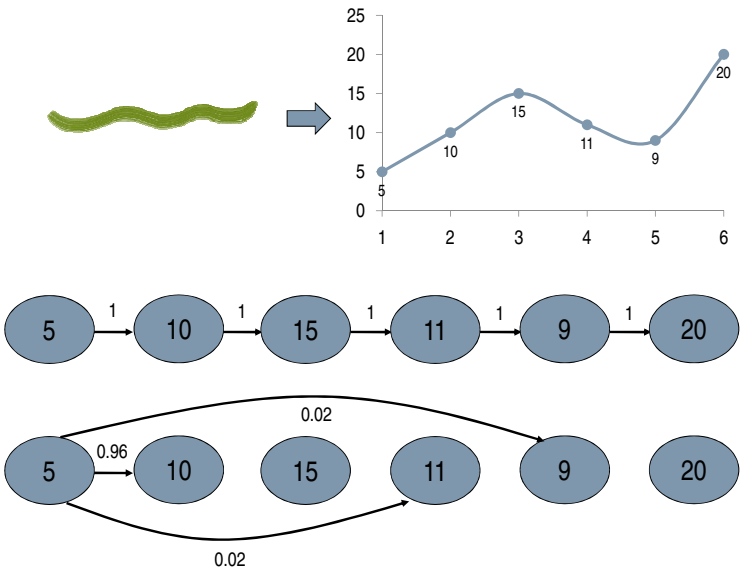

Fig. 1. The original curve as a Markov process with deterministic transitions, and randomizing the curve by allowing state transitions that to states that are close to the original next state. The original curve is a Markov process where state transitions are deterministic. By randomization, we allow transitions to all states based on the difference between the original next state and the displacement value. For example (the graph on the bottom), the next state of the first state in the original curve has displacement value 10. So we reduce the probability of the transition to this state from 1 and the difference is distributed in state transitions depending on how far other states are from value 10. In this example, values 9 and 11 are close, so we also allow transitions to these states as well, while other values are far, so the transition probabilities to those states are small (we depicted only higher probability transitions with arrows for clarity).

We apply a probabilistic approach to allow the generation of brush strokes that are similar to but also different from the specified curve. For a sample stroke with the length of k, we create a D_{ij} $[n \times n]$ *distance matrix* where $n = k + 1$ and ij denotes the distance between the offset of state i and j of the sample.

The main diagonal entries are always set to 0 as there is 0 distance between a state and itself:

$$D_{ij} = \begin{cases} 0, & \text{if } (i = j \vee i = n) \\ \infty, & \text{if } (i \neq n \wedge j = n) \\ |y_i - y_j|, & \text{otherwise.} \end{cases} \tag{1}$$

If we assigned probabilities to transitions in a way that the probability is one where the distance between two states is equal to the value of the distance matrix, then we would always generate the original curve (let us ignore the case when two states have the same value). So we relax this requirement and give non-zero probabilities to those transitions as well, where the distance is close to the distance matrix value.

The highest transition probabilities are set to the states that follow the trajectory of the reference curve, and lower ones are assigned to the others with an exponential characteristic inversely proportional to their distance. The σ parameter defines the weight of this characteristic — as lowering it directs the algorithm mostly towards ideal transitions, a higher value would favor paths that are less likely to occur in the given sample:

$$P_{ij} \propto \exp(-D_{i+1,j}/\sigma). \tag{2}$$

To be able to utilize a probabilistic approach, matrix P_{ij} must be a probability matrix, i.e. its row sums must be 1:

$$\forall i : \sum_{i=0}^{n} P_{ij} = 1.$$

This simple approach has two problems. On the one hand, it does not take dynamics into account, but according to the properties of Markovian processes, it considers only the current state when the state transition probabilities are determined. However, human hand has mass and momentum, so when the curve moves in a given direction, it is more likely that the curve continues into that direction than turning to the opposite one. To handle this, we use the concept introduced in video textures, and filter the distance matrix before computing the transition probabilities.

On the other hand, this process will quite quickly reach the last state from where there are no further state transitions, so our process is trapped and the curve is terminated. To avoid this, we set the transition probability to the terminating state to zero, and propagate this modification back to the Markovian model. The propagation is governed by Q-learning.

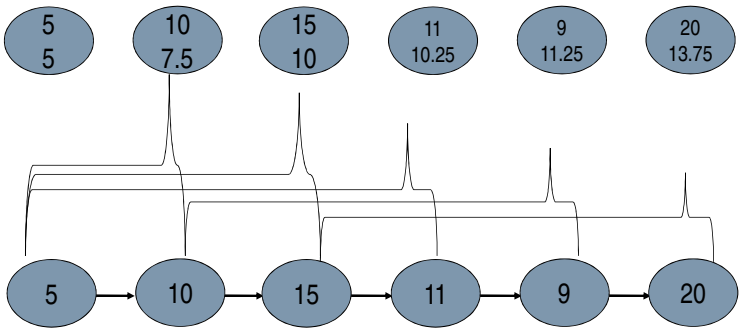

Fig. 2. Weighting the distance matrix resulting in a Markov process where each state of the new graph is characterized by two values, the original value which is used when the curve is drawn, and an average value that is taken when the distance from other states is calculated. Using the average rather than the original value mimics the dynamics of curve generation.

4.1 Brush movement dynamics

According to the attributes of the Markov random field approach, the state transitions are taken independent of the preceding states. To preserve the dynamics of the artistic brush movement (i.e. to assume some momentum that cannot be changed instantly), when examining state i, we also consider the preceding states $[i-1, i-2, \ldots, i-m]$ by linearly weighting them (Figure 2):

$$D'_{ij} = \sum_{k=0}^{m} w_k \, D_{i-k,j-k}. \tag{3}$$

The weighting frame size m can be parameterized freely for different, interesting results. According to the experiences with our own reference implementation, $m = 4$ and $w_k = 1/m$ are well suitable for most cases.

So far, we did not consider what happens at the edge of the matrix where there are no m predecessors. Here, weight w should be increased to guarantee that the total sum of distance values is preserved. Generally, we use the following formula that handles edges as well:

$$w_k = \frac{1}{\min\{\min(i,j), m\} + 1}. \tag{4}$$

4.2 Model building by Q-learning

The described method is unable to produce brush samples with infinite length as there is a high probability that the algorithm follows the paths used in the reference curve, it will soon reach the end of the sample curve, remaining stuck in a "dead end".

To avoid these "dead ends", further examination of the reference curve is advised. The utilization of *Q-learning* (or reinforcement learning)[10] makes it

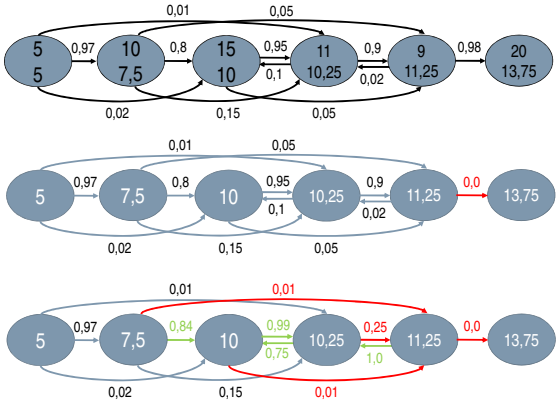

Fig. 3. Eliminating dead ends by setting the transition probability to the last state to zero and propagating this effect back with Q-learning. In the original process, the probability of transitioning to the last is 0.98. This is replaced by zero, but then the sum of exit probabilities of the last but one state is not unity. This is fixed by normalization, but we made an abrupt change in the model, so the probability of reaching this last but one state should also be decreased. Q-learning is an iterative approach for updating the whole graph.

possible to efficiently assign increased costs to the transitions that are expected to lead to unfavorable states. Originally, Q-learning works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter.

The cost of a transition is defined as a metric that is proportional to the absolute value of the displacement (offset) value between the two states. In order to keep the process from entering the final state, we assign very large cost to reaching the final state. The technique is proved to be convergent using the $\alpha < 1$ *learning rate* parameter, therefore we can solve the problem with iteratively applying

$$D''_{ij} \leftarrow D'_{ij} + \alpha \min_k D''_{jk} \tag{5}$$

after the $D''_{ij} \leftarrow D'_{ij}$ initialization until convergence is reached. Intuitively, Q-learning assigns a cost to a transition that is the sum of the cost of this step and the cost of the cheapest trajectory from the next state. In a more general form, we also allow the computation of some power p of the previous costs:

$$D''_{ij} \leftarrow (D'_{ij})^p + \alpha \min_k D''_{jk}. \tag{6}$$

After weighting and Q-learning, the state transition probabilities are computed from the modified transition distances (or costs):

$$P_{ij} \propto \exp(-D''_{i+1,j}/\sigma). \tag{7}$$

5 Analysis of the example stroke

The example stroke of the artist is stored in a bitmap image. Before analyzing his or her style to reuse it, the sample needs to be transformed to meet the definition of the mathematical functions. For any values of the variable x in the freeform brush sample, we have several y_i offsets — actually, as any kind of real stroke will have an own wideness, more y_i values are expected. Since the algorithm's task is to analyze the original path of the sample, we can solve the problem by the *skeleton* of the curve [6].

6 Curve generation

During the curve synthesis, new curve offset values are generated and used to deform previously made geometrical models. The original and new curves will both be represented with a Catmull-Rom spline.

We compute a new point with the help of the P_{ij} probability matrix: for any state k, the next state $k+1$ can be determined by generating a random number on the unit interval and picking the row k of the matrix: we are given a row vector $[P_{k1}, P_{k2}, \ldots, P_{kn}]$ where kn denotes the transition probability from state k to state n. Since the mentioned matrix is a stochastic probability matrix discussed in section 4.2, for any k, the sum of the elements of the row vector will be 1. For each next possible state we define an interval from the unit interval that is proportional to its probability, and accept it as the next state if the newly generated random number falls into its interval.

However, matching random numbers with the entries of the row vectors taken from P_{ij} can be rather troublesome. Consider a $[1500 \times 1500]$ matrix, where the main diagonal entries are between 0.99 and 0.999 for low σ parameters. Therefore the main diagonal takes the 99% of the interval, and all other entries are concentrated in the remaining 1%. The random number generator must be sufficiently accurate to produce satisfactory results. In our reference implementation, we used the Mersenne Twister algorithm with the accuracy of 53 bits [12].

The newly generated random offsets need to be used on previously drawn contours. However, the distortion of the curves is trivial for a straight line by simply adding the offset values to it, the same method cannot be used on bent lines. A dedicated coordinate system is used to follow the path of the contours. The solution is known as the TN (tangent-normal) coordinate system, where one axis points to the tangent direction ($\boldsymbol{T}$), one towards the normal ($\boldsymbol{N}$). Before any distortion the curves we must make sure that enough geometry detail is present that is the curve is finely and evenly tessellated in image space. The user can also set the image space length of the original sample pattern to be reconstructed.

7 Textured line drawing

Both contour lines and hatch lines arrive at the rendering phase as Catmull-Rom splines defined by their control points [20]. Here, these 2D curves are drawn

to provide a hand-crafted look while allowing interactive artist control. Hand-drawn curves have roughly constant width on the image independently of the object's distance from the virtual camera. The width and the image of a stroke depend on the artist's device used to draw this stroke. Hand-drawn strokes are not geometrically exact, they have inherent randomness. In order to present natural looking line art, curves are fattened to polygons and rendered as textured polygon strips where the texture simulates brush or pencil strokes. Textured line drawing consists of two basic tasks. We have to fit a triangle strip to the curve that represents the path of the stroke, and texture coordinates need to be calculated for the triangles. Both tasks are performed in image space.

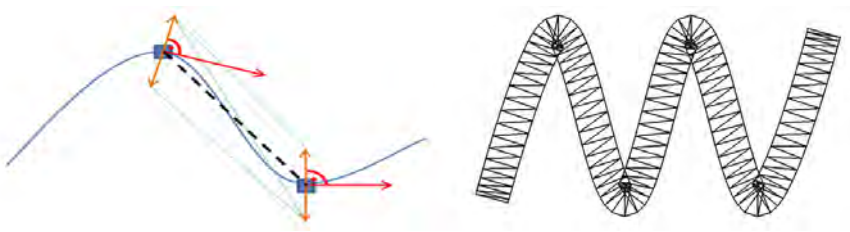

Fig. 4. Strip generation for a Catmull-Rom spline.

The goal of stripification is to create a triangle strip that fits to a curve. First, the curve is decomposed to shorter segments of similar arc length. Computing the derivative, i.e. the tangent vector at the end points of the segment and rotating them by 90 degrees, we obtain the normal vectors of the curve. Translating the end points in the direction of the normals, we can calculate the vertices of the enclosing polygon strip (Figure 4).

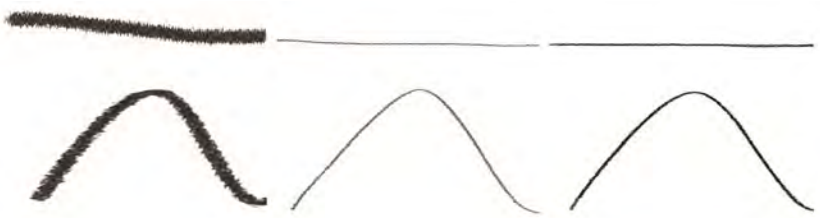

Fig. 5. Three different textures simulating brushes and a curve textured with them.

The u texture coordinate for the resulting strip is computed from the arc length. The v coordinate is 0 or 1 depending on whether the vertex is the translation of the curve point by the normal or in an opposite direction.

8 Results

We integrated the contour and hatch line extraction, the artistic pattern regeneration and the line stripification and rendering into a movie production pipeline. Two main commercial software packages were used: 3Delight (a RenderMan compatible renderer) and Nuke (a postprocessing application with advanced 3D capabilities). Line extraction was implemented in the renderer while all other tasks — which require user interaction- were moved to the postprocessing application. The user can freely draw the desired pattern with the built — in capabilities of Nuke and the final deformed contour and/or hatching curves are displayed interactively.

Figure 6 shows a torus where sample strokes modify the style of the silhouette strokes.

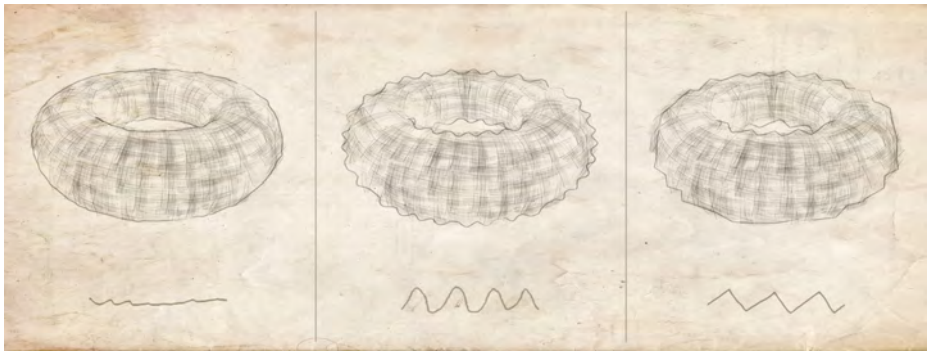

Fig. 6. Silhouettes on a torus model.

Fig. 7. Torus rendered with different hatching styles.

Figure 7 demonstrates that a wide variety of artistic styles can be simulated by using different brush strokes. Figure 8 depicts a pawn rendered with strokes following the main curvature directions and with two different randomizations based on two sample strokes. Figure 9 presents a Venus where both silhouettes and hatch lines are generated by the proposed model.

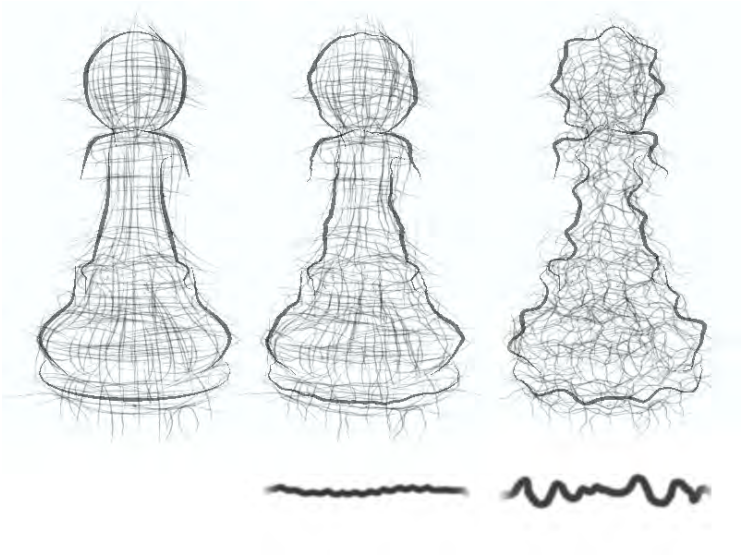

Fig. 8. Pawns rendered with strokes following the main curvature directions (left) and with two different randomizations based on two sample strokes.

Fig. 9. Hatching and silhouettes of the Venus model.

9 Conclusions

This paper presented an NPR system and its application in the movie rendering pipeline. Our system implements NPR effects like contouring and hatching. This project also proves that NPR effects can be made flexible enough to allow the artist to express his own ideas and produce images meeting his expectations.

Acknowledgements

This work has been supported by OTKA K-719922, and by the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" (TMOP-4.2.1/B-09/1/KMR-2010-0002).

References

1. Zainab AlMeraj, Brian Wyvill, Tobias Isenberg, Amy A. Gooch, and Richard Guy. Computational aesthetics 2008: Automatically mimicking unique hand-drawn pencil lines. *Comput. Graph.*, 33(4):496–508, 2009.
2. Liviu Coconu, Oliver Deussen, and Hans-Christian Hege. Real-time pen-and-ink illustration of landscapes. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 27–35, 2006.
3. Balázs Csébfalvi, Lukas Mroz, Helwig Hauser, Andreas Knig, and Eduard Gröller. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum*, 20(3):452–460, 2001.
4. Paul Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90*, pages 207–214, 1990.
5. Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Piché, and Victor Ostromoukhov. Hatching by example: a statistical approach. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 29–36, 2002.
6. Palágyi Kálmán:. *Vékonyitó algoritmusok 3D képekre.* Doktori értekezés, Szegedi Tudományegyetem, 2000.
7. Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. *ACM Trans. Graph.*, 22(3):856–861, 2003.
8. Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. In *SIGGRAPH '02*, pages 755–762, 2002.
9. Levente Kovács and Tamás Szirányi. Creating video animations combining stochastic paintbrush transformation and motion detection. In *16th ICPR*, pages 1090–1093. IEEE and IAPR, 2002.
10. Andrew Moore Leslie Pack Kaelbling, Michael Littman. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
11. Thomas Luft, Frank Kobs, Walter Zinser, and Oliver Deussen. Watercolor illustrations of cad data. In *International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, pages 57–63. Eurographics Association, jun 2008.

12. Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.
13. Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH '96*, pages 477–484, 1996.
14. Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *SIGGRAPH '01*, page 581, 2001.
15. Arno Schödl and Irfan Essa. Machine learning for video-based rendering. In *In Advances in Neural Information Processing Systems*, pages 1002–1008. MIT Press, 2000.
16. Arno Schödl and Irfan A. Essa. Controlled animation of video sprites. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 121–127, New York, NY, USA, 2002. ACM.
17. Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 489–498, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
18. T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufman, 2002.
19. László Szirmay-Kalos. *Monte-Carlo Methods in Global Illumination — Photorealistic Rendering with Randomization*. VDM, Verlag Dr. Müller, Saarbrücken, 2008.
20. László Szirmay-Kalos, György Antal, and Ferenc Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés*. ComputerBooks, Budapest, 2003.
21. László Szirmay-Kalos and László Szécsi. Deterministic importance sampling with error diffusion. *Computer Graphics Forum (EG Symposium on Rendering)*, 28(4):1056–1064, 2009.
22. Hui Xu and Baoquan Chen. Stylized rendering of 3d scanned real world environments. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 25–34, 2004.