First publ. as: Paper / Pacific Conference on Computer Graphics and Applications (PG '05), Macao, China, 2005

Interactive Watercolor Animations

Thomas Luft Oliver Deussen

Department of Computer and Information Science
University of Konstanz, Germany
{luft, deussen}@inf.uni-konstanz.de

Abstract

We present algorithms that produce a natural watercolor appearance of 3D-scenes and render them in real-time. Our approach imitates the most important natural effects of watercolor and incorporates two essential painting techniques: the wet-in-wet and the wet-on-dry painting. We apply our technique to different scenes, and show how to segment and abstract the content.

Keywords: Non-Photorealistic Rendering, Watercolor, Real-time Rendering

1 Introduction

Watercolor paintings are characterized by diverse patterns and specific textures, which occur during the painting and drying process. Thus, in the field of non-photorealistic computer graphics, the production of watercolor paintings is one of the most intricate and complex processes.

In contrast to already known simulation algorithms, our goal is to create convincing watercolor renderings of 3D-scenes in real-time. We introduce an efficient approach that allows the imitation of the most essential drawing techniques and natural effects, and that give the appearance of natural watercolor paintings. Since the process of drawing is always an abstraction of the motif, special consideration must be given here to the overall scene complexity and its simplification (see Fig. 1).

In order to create real-time graphics, we especially rely on the potentials of hardware accelerated shaders.

1.1 Related Work

There are only a few works that explicitely treat the generation of watercolor paintings. A physically based approach is the work of Curtis et al. [1], which is directly related to the cellular automaton approach of Small [4]. Curtis et al. offer a detailed description of the most important effects that occur during the watercolor painting and drying process. A similar work is described by Van Laerhoven et al. [5]. Their approach allows an interactive simulation of watercolor painting that consists of a number of user given brush strokes. The work of Lum and Ma [3] was inspired by watercolor paintings. It describes a raytracing based rendering pipeline that creates procedural textures based on LIC (line integral convolution), which produce a watercolor like appearance. Their work allows the rendering of 3D-scenes, however, it does not allow the real-time rendering. Lei and Chang [2] propose a rendering pipeline that imitates several important watercolor effects and allows a real-time rendering of small scenes. Their work is similar to our approach, but uses different techniques.

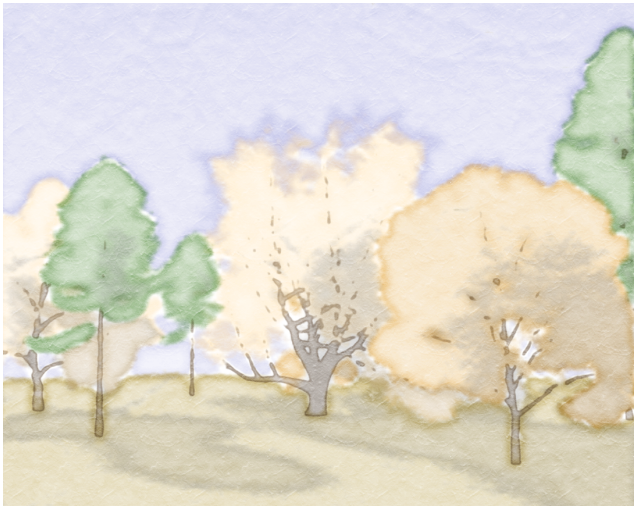

Figure 1: *Watercolored landscape scene.*

2 Imitating Watercolor Paintings

Curtis et al. [1] describe several effects that are essential for a watercolor appearance, such as the edge darkening,

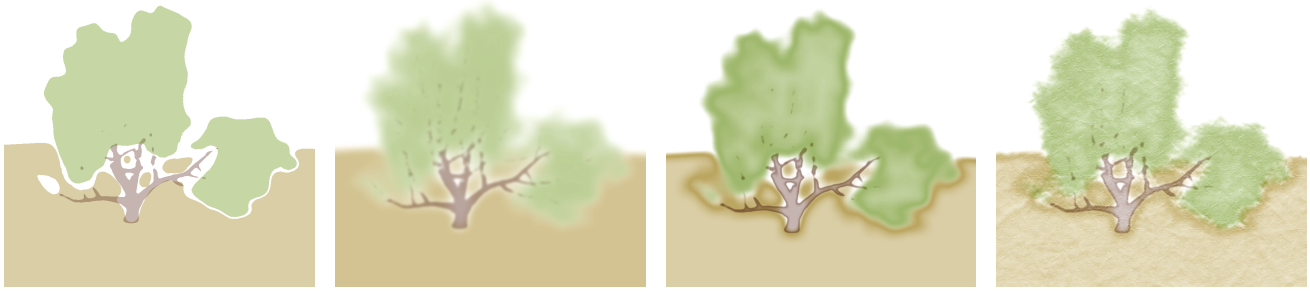

Figure 2: *Watercolor imitation: a) shape extraction, b) flow pattern, c) edge darkening, and d) paper structure*

the flow effect, and the pigment granulation. In this section we describe our approach to imitate these effects by using simple algorithms instead of an exact, but nevertheless complex simulation. Since our method is intended for real-time graphics, we implemented the effects using vertex and fragment shaders.

Simplification and Segmentation In order to simplify and segment the scene, we separate it into different areas with more or less uniform content. This segmentation is based on an unique identifier that we assign to an object or a group of objects. As a result, we achieve a more precise segmentation in comparison to image segmentation techniques that are based on color or texture. This advantage provides a very high degree of frame-to-frame coherence during the animation of our watercolored scenes.

The first step is to create an ID-image $S : \mathbf{N^2} \rightarrow \mathbf{R^4}$ in the RGBA color space, which contains individual ID-layers $\rho : \mathbf{N^2} \rightarrow \mathbf{R}$. Each layer is coded by one individual color channel of the ID-image. If more layers are preferred, several ID-images have to be created. This coding scheme allows us to apply anti-aliasing without crosstalk artifacts between the ID-layers. Thus, the quality of the ID-layers is increased significantly, and a high noise ratio can be avoided that would otherwise propagate throughout the next rendering steps and disturb the results.

For the simplification step we apply a Gaussian filter to each ID-layer. The result of this simplification is an abstract smooth shape, which is described by the intensity values of the ID-layers. The intensity values are used directly to create the different washes of watercolor (in the following referred to as color layers). Each wash has a specific color and transparency depending on the ratio between water and color pigments. Consequently, our watercolor model consists of several color layers $\lambda : \mathbf{N^2} \rightarrow \mathbf{R^4}$ that are specified by a RGB color vector c_{rgb} and an overall transparency c_a. In the following, we outline the imitation of the watercolor effects due to additionally modulating the alpha channel $\lambda_a(x, y)$ of each color layer.

Shape extraction The shape of the color layer arises from the intensity values of the corresponding ID-layer $\rho(x, y)$. To extract the intensity, we use the dot product of the ID-image $S(x, y) = [r, g, b, a]$ and a mask $m = [r, g, b, a]$ that specifies the color channel of the desired ID-layer. Then a step function is applied to produce a first, hard edged shape of the color layer:

$$\rho(x, y) = S(x, y) \cdot m$$

$$\lambda_a(x, y) = c_a \cdot step(\kappa_\rho, \rho(x, y))$$

Here the threshold κ_ρ defines the dilation of the color layer. A high threshold results in a smaller area, while a small threshold increases the color area (see Fig. 2(a)). The size of the ID-layers are adjustable and do not have to be equal to the screen size. By choosing a particular resolution for the ID-image, we can control the level of abstraction. Consequently, the smaller the ID-image, the higher is the degree of the abstraction.

Wet-in-Wet and Wet-on-Dry Painting We distinguish between the two most important watercolor painting techniques: the wet-in-wet and the wet-on-dry painting. These techniques mainly affect the border of the color layer. While the wet-on-dry painting causes a hard edged border, the wet-in-wet painting causes smooth, feathery patterns at the border that may overlap with the underlying color layers and simultaneously interact with them.

To adjust the border behavior, we add another parameter κ_δ and replace the step function with a smooth step function (cubic Hermite interpolation):

$$\lambda_a(x, y) = c_a \cdot \Delta step(\kappa_\rho - \kappa_\delta, \kappa_\rho + \kappa_\delta, \rho(x, y))$$

Here the parameter κ_δ specifies the smoothness of the border and reproduces the wetness of the underlying color layer. A small κ_δ simulates a harder border, and thus a dry underground, while a large κ_δ produces a smooth color blending, and consequently imitates a wet-in-wet painting (see Fig. 2(b)).

Edge Darkening During the drying process, the watercolor pigments are transported towards the border due to water flow. This effect is easily imitated using our filtered ID-layer. The applied Gaussian filter causes a smooth intensity transition following the border of the color layers. To achieve a darkened border, we additionally modulate the alpha channel $\lambda_a(x, y)$ by the smooth intensity values $\rho(x, y)$:

$$\lambda_a(x, y) = \lambda_a(x, y) \cdot \rho(x, y) \cdot \kappa_\omega$$

Here the user can specify the intensity of the edge darkening by changing the attribute κ_ω, which influences the final result drastically. This attribute represents a particular quality, which is usually determined by the manufacturer of natural watercolors (see Fig. 2(c)).

Paper Structure The underlying paper structure affects the water flowing process significantly, and thereby also the visual results. To emphasize this process, we additionally modify the color layers alpha channel according to an intensity texture $T : \mathbf{N^2} \rightarrow \mathbf{R}$ that represents the paper structure:

$$\lambda_a(x, y) = \lambda_a(x, y) \cdot T(x, y) \cdot \kappa_\tau$$

Since the paper structure influences the pigment transportation fundamentally, it also changes the appearance of the border of the color layers. Thus, we integrate the paper structure into the intensity values of the ID-layers:

$$\rho(x, y) = S(x, y) \cdot m + T(x, y) \cdot \kappa_\theta$$

The intensity of the paper structure and its influence on the border of the color layers is specified by the parameters κ_τ and κ_θ.

Finally, the color layers are composed on the screen using the standard OpenGL blending function. In order to emphasize structure and lighting, additional layers can be integrated and processed similarly, e.g. the shadow in figure 1 and 3.

3 Conclusion

We present algorithms that produce a natural watercolor appearance of 3D-scenes and render them in real-time. Our approach imitates the most important natural effects of watercolor and incorporates two essential painting techniques. We applied our technique to different scenes, and show how to segment and abstract the content.

We tested our implementation on a 3.0 GHz Pentium IV with an GeForce 6800 graphics board. A multi-pass rendering procedure was used to render one or more ID-images, and the shadowing effect that is based on a shadow map. In Table 1 an overview of the scene complexity and the performance is given: *#triang* is the number of triangles of the original scene, *#render passes* is the number of required render passes, *#color layers* gives the number of color layers, and *fps* gives the average number of frames per second during an animation. The scenes are rendered at 720×720 pixels, the ID-images at 360×360 pixels.

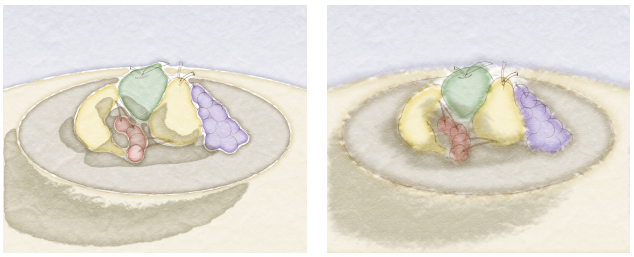

Figure 3: *Watercolored still life showing different effect settings, combined with silhouette strokes.*

Future work aims at the lighting and shading of the scenes. Currently, the color layers have a unique color and thus they produce a flat impression. Here the incorporation of a shading algorithm will be helpfull as well as the integration of additional color layers that emphasize highlighted or dark areas.

Table 1: *Scene complexity and performance.*

Figure	#triang	#render passes	#color layers	fps
1	420614	3	7	26.5
2	62624	1	3	134.3
3	62560	4	8	44.7

References

[1] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin. Computer-generated watercolor. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 421–430, 1997.

[2] E. Lei and C.-F. Chang. Real-time rendering of watercolor effects for virtual environments. In *PCM '04: Proceedings of the 5th Pacific Rim Conference on Multimedia*, pages 474–481, 2004.

[3] E. B. Lum and K.-L. Ma. Non-photorealistic rendering using watercolor inspired textures and illumination. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, pages 322–330, oct 2001.

[4] D. Small. Simulating watercolor by modeling diffusion, pigment, and paper fibers. In *Proceedings of SPIE 91*, feb 1991.

[5] T. Van Laerhoven, J. Liesenborgs, and F. Van Reeth. Real-time watercolor painting on a distributed paper model. In *Proceedings of Computer Graphics International 2004*, pages 640–643, jun 2004.

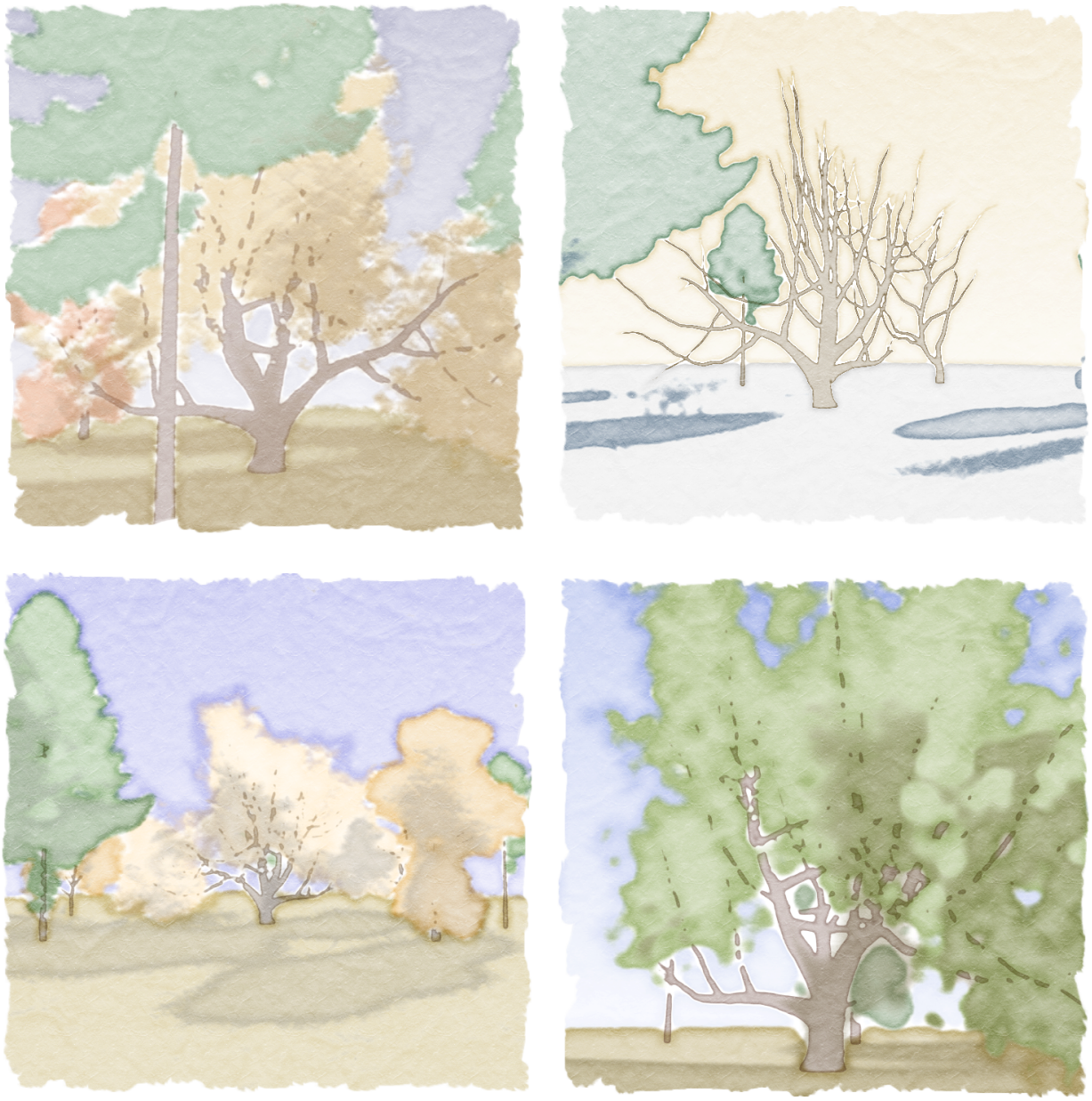

Figure 4: *Images from "The four seasons" showing different effect settings.*

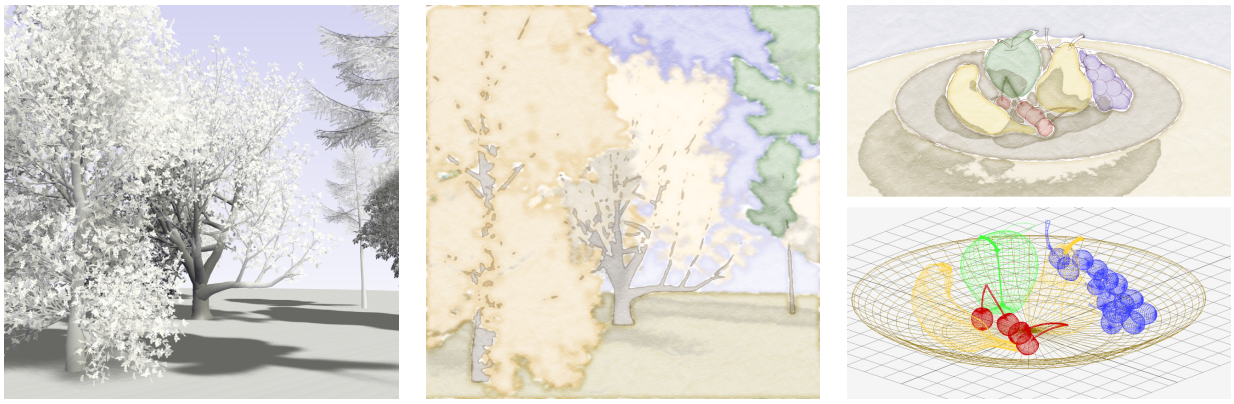

Figure 5: *A comparison with the original scenes.*