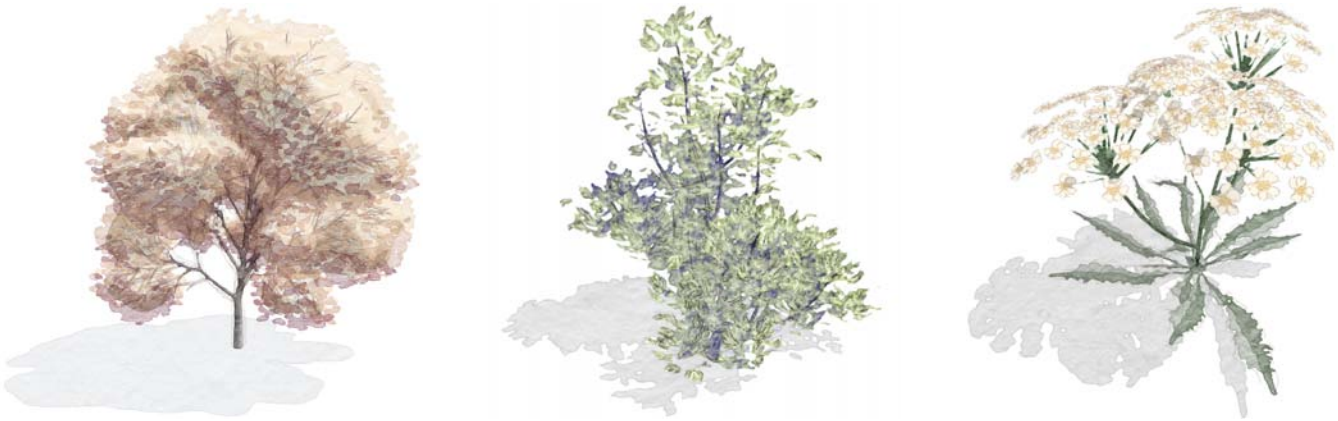
Real-Time Watercolor Illustrations of Plants Using a Blurred Depth Test

Thomas Luft*
University of Konstanz

Oliver Deussen†
University of Konstanz



Abstract

We present techniques to create convincing high-quality watercolor illustrations of plants. Mainly focusing on the real-time rendering, we introduce methods to abstract the visual and geometrical complexity of the original 3D plant models that are obtained from the photorealistic rendering. Additionally, we integrate detail layers to emphasize the structure, the texture, and the lighting. We distinguish between particle based, texture based, and line stroke based detail primitives, which provide a wide application range for the individual tree and plant models. To support smooth transitions and the frame-to-frame coherence during an animation, we introduce a so-called blurred depth test to perform depth comparisons. This blurred depth test is utilized for the visibility computations of the detail primitives and for the creation of a soft shadowing effect. With our techniques, we create an appropriate lighting and shadowing of the scene, and achieve a high frame-to-frame coherence. For the final watercolor appearance, we apply a formerly published approach that is implemented using hardware shaders.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; Bitmap and framebuffer operations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; Hidden line/surface removal

Keywords: artistic rendering, real-time rendering, watercolor, soft shadows, hidden surface removal, blurred depth test

*e-mail:luft@inf.uni-konstanz.de
†e-mail:deussen@inf.uni-konstanz.de

NPAR 2006, Annecy, France, 05–07 June 2006.
© 2006 ACM 1-59593-357-3/06/0006 $5.00

1 Introduction

The creation of watercolor illustrations has a long tradition in various arts, literature, fashion, advertisement posters, architectural sketches, and botanical figures. Thereby, the artist colorizes a sketchy or detailed drawing and exploits the very complex nature of watercolors in order to achieve a wide variety of artistic effects. The characteristic textures of watercolor are created by numerous natural effects such as diffusion, backruns, pigment granulation, and color graduation. For artistic purposes, the production of real-time watercolor illustrations is one of the most exciting, and promising processes in the field of non-photorealistic computer graphics, since a manual creation of animated watercolor is very intricate.

In contrast to already known real-time canvas simulations, e.g. [Baxter et al. 2004; Chu and Tai 2005; Van Laerhoven et al. 2004], our goal is to create convincing watercolor illustrations of 3D scenes in real-time. We focus on the rendering of trees, plants, and small landscapes, an area that has remained a challenge in the field of non-photorealistic rendering. Special consideration must here be given to the visual complexity, its simplification, and the modeling that differs strongly from that of the photorealistic rendering. For our 3D scenes we use complex plant models that are commonly used for photorealistic rendering. This has the advantage of large plant libraries being available [greenworks organic software 2005] that provide detailed and hierarchical classified geometry and thereby support the automatic simplification and segmentation of the objects, the frame-to-frame coherence of animated scenes, and the lighting and shadowing.

1.1 Related Work

There are several approaches that explicitly treat methods to generate watercolor paintings. A physically based approach is the work by Curtis et al. [1997], which is directly related to the cellular automaton approach by Small [1991]. Curtis et al. offer a detailed description of methods to automatically simulate the major effects that occur during the watercolor painting and drying process. The key idea here is to use a simulation model that consists of three physically based layers. While the simulation of these layers produce very convincing results, this method is computationally expensive,

and thus cannot be considered as real-time graphics. Another work based on this three-layer model is described by Van Laerhoven et al. [2004]. Their approach provides a real-time simulation of watercolor painting on the basis of a virtual canvas. Other works that simulate a virtual canvas are the IMPaSTo framework by Baxter et al. [2004] and the work of Chu and Tai [2005]. Here the user acts as the artist and draws the scene using special input devices. The work of Lum and Ma [2001], also inspired by watercolor paintings, describes a raytracing based rendering pipeline that creates procedural textures based on line integral convolution and produces a watercolor like appearance. Here the combination of several semi-transparent color layers forms the final result. Their work allows the rendering of 3D scenes, but do not provide real-time techniques. In contrast to these approaches, our work concentrates on the real-time rendering of 3D scenes with a watercolor painting appearance.

Stroke based approaches, such as [Strassmann 1986; Hertzmann 1998], generate a number of brush strokes using procedural geometry and modulate the color, the texture, and the transparency along the brush stroke according to user or system specified attributes. The approach of Hertzmann places strokes automatically in an iterative process on the basis of an input image. Thereby, large strokes are drawn first and smaller strokes are added iteratively to capture details. There are other approaches that follow the long line of particle based solutions, beginning with the work of Reeves and Blau [1985] and their particle forest. A milestone based on particle systems was the *painterly rendering for animation* introduced by Meier [1996]. Her work combines a brush stroke like appearance with stable, frame-to-frame coherent 3D particle sets. The work of Bastos et al. [2002] compares acceleration techniques for the non-photorealistic rendering of trees based on particle systems. While these results resemble more acrylic or oil paintings, we use 3D particles to create watercolor layers that reproduce additional details.

Works that treat the abstract rendering of vegetation or similar delicate objects are those of Kowalski et al. [1999] and Deussen and Strothotte [2000]. Here the focus is the simplification and abstraction of the complex geometry during the rendering, while preserving details to emphasize structure and lighting. In contrast to Kowalski et al., who use procedural stroke based textures, so-called graftals, Deussen and Strothotte use depth differences in combination with simple drawing primitives to create abstracted pen-and-ink illustrations of trees. Fiore et al. [2003] propose a 2.5D system to create cartoon like animations of trees using a combination of abstract 2D drawing primitives and 3D information. Another approach considering abstraction is introduced by Wilson and Ma [2004]. This work incorporates image space techniques to simplify pen-and-ink style drawings and also offers applications of complex tree models. In contrast to these approaches, we introduce a modeling process that is especially adapted to watercolor animations of trees and plants. We use a combination of different techniques to abstract and simplify the visual and geometrical complexity, while at the same time enhance the final result with additional details in order to emphasize the lighting and the structure of the objects.

1.2 Contribution

We introduce techniques for the creation of convincing, detailed, and well lit watercolor illustrations of plants that preserve the frame-to-frame coherence, provide smooth visibility transitions, and that have strong depth cues. Our work focuses mainly on the following aspects.

To achieve smooth visibility transitions of objects, and thereby providing for a high frame-to-frame coherence, we introduce an alternative method to perform depth comparisons: we replace the traditional depth test by a so-called *blurred depth test*. We demonstrate the utilization of this technique for the rendering of soft shadows based on shadow maps and for the hidden line and surface removal in the domain of non-photorealistic rendering.

During the modeling process we utilize implicit surfaces to abstract and simplify our tree models, and enhance the final results with additional details based on 3D particle sets. For the detailed rendering of smaller plants we introduce intensity textures to reproduce the exact shape and the surface structure. Additionally created line strokes form the sketchy basis of the renderings.

Our techniques are mainly based on hardware shader implementations that facilitate the efficient generation of highly detailed watercolor illustrations of 3D vegetation scenes. We show that these techniques can easily be utilized for real-time applications. Finally, the appearance of the renderings can be adjusted with a minimum of user given attributes that provide a wide variety of possible results.

1.3 Overview

An overview of our system is given in Fig. 1. We distinguish between the rendering of tree models and smaller plant models. For trees, the modeling process is composed of an implicit surface computation and a 3D particle generation for the reproduction of details. Thus, the complex foliage geometry is completely replaced by a highly simplified and abstracted representation. The rendering of smaller plants is based on the original geometry in combination with specially prepared intensity textures to reproduce details. On the basis of these data, a number of *intensity maps* are rendered that describe the shape of the final watercolor layers while the colors of the watercolor layers are specified by the user. The lighting of the scene is incorporated in form of a shadow map and a lighting map. The shadow map is utilized to produce an additional watercolor layer representing the shadow. The lighting map modulates the color of the watercolor layers to emphasize dark and bright areas, thereby producing a more dynamic finish. In case of a tree model, both maps are based on the abstract implicit surface. Finally, we add line strokes as a sketchy basis of the renderings.

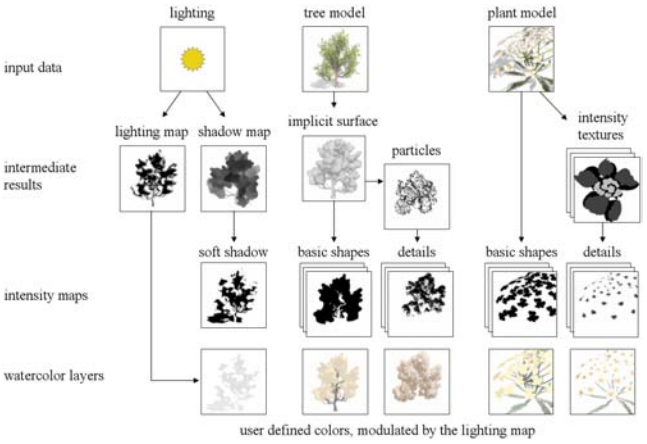


Figure 1: System overview. We distinguish between the modeling and rendering of tree models and smaller plant models.

The following sections describe our system in more detail: In Section 2 we introduce the *blurred depth test*, which is a key technique for the following sections. Here we provide a method for introducing smooth visibility changes using a non-binary depth comparison. Section 3 introduces our basic method to produce 3D render-

ings with a watercolor painting appearance. We briefly describe a formerly published approach for creating a number of watercolor layers and show how these results can be improved by incorporating an appropriate lighting and shadowing. Section 4 describes the modeling process that provides an abstraction and simplification of the given 3D models. Here we particularly adapt the given geometry and texture data obtained from the photorealistic rendering, whereby we distinuish between tree models and smaller plant models. In Section 5 we present some of our results, discuss the visual quality, and give a performance overview.

2 The Blurred Depth Test

This section describes a particular way to perform depth comparisons that is used in our system for the hidden line and surface removal as well as for the generation of soft shadows. The so-called *blurred depth test* is motivated by the requirement for a high frame-to-frame coherence and smooth visibility transitions in the field of non-photorealistic rendering. The traditional depth test is a binary test that denotes a fragment as being visible or not visible, and thus results in abrupt visibility changes. While this procedure is long proven and well suitable for the photorealistic rendering, we are looking for a way to achieve smooth visibility transitions for non-photorealistically rendered scenes. Therefore, we enhance the original metaphor of visible and none visible fragments with an alternative: a semi-visible fragment.

The implementation of the blurred depth test is realized by a two-pass rendering procedure. Firstly, we render the ordinary depth map $D : \mathbf{N^2} \rightarrow \mathbf{R}$ with $D(x,y) \in [0,1]$ and store it as an intermediary result. Secondly, we render the scene by replacing the original binary depth test through a smooth step function $\Delta step$. This function permits the existence of semi-visible fragments by considering the depth difference between the currently processed fragment and the stored depth map.

The smooth step function $\Delta step$ takes three input values: a lower bound l, an upper bound u, and an input value v. The input value v is normalized to the interval $[l,u]$ by $\hat{v} = \frac{v-l}{u-l}$. Then $\hat{v}$ is clamped to the interval $[0,1]$ and a cubic Hermite interpolation is applied:

$$\Delta step(l,u,v) = 3\hat{v}^2 - 2\hat{v}^3. \tag{1}$$

In our application, the input value v equals the difference between the depth value $f_d \in [0,1]$ of the fragment f being processed and the corresponding depth value $D(f_x,f_y)$ in the stored depth map: $v = D(f_x,f_y) - f_d$, with the near and far plane being located at 0 and 1. The lower and the upper bound of the smooth step function describe the depth range, in which semi-transparent fragments are produced (Fig. 2). Beyond this range, the fragments are either completely visible or completely invisible. Since we consider depth differences, this range equals a zero-centered interval $[-\delta,+\delta]$ that is specified by the user paramter $\delta \in [0,1]$ in camera coordinate space. Please note that we compute linearly distributed depth values throughout our rendering pipeline, in contrast to non-linear depth values usually resulting from a perspective projection matrix. Thus, the parameter δ describes a depth-independent band $[-\delta,+\delta]$ of constant width. For a proper hidden line and surface removal, we introduce a second user parameter $\beta \in [-1,1]$, which specifies a shift of the smooth step function. Thereby, a shift $\beta = \delta$ preserves the original visibility of an object A that is closely in front of another object B. Without such a shift, object A would become already semi-visible, if their depth difference is $B_d - A_d < \delta$.

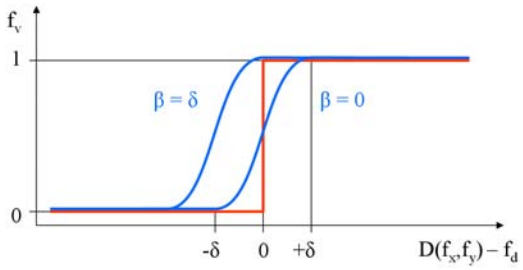


Figure 2: Blurred Depth Test. Transfer functions of the binary (red) and the blurred depth test (blue). δ specifies the depth range, in which semi-visible fragments are produced, and β describes a shift that preserves the original visibility of a fragment. The smooth step function is based on a cubic Hermite interpolation and corresponds to the implementation provided by the OpenGL Shading Language.

Using these parameters and the definition of the smooth step function, the visibility of the fragment is now expressed by $f_v \in [0,1]$ utilizing the blurred depth test:

$$f_v = \Delta step(-\delta,+\delta,D(f_x,f_y) - f_d + \beta). \tag{2}$$

After computing the fragment visibility f_v, another important issue is its visual represention. If we directly interpret the fragment visibility f_v as transparency, a depth ordered rendering of the fragments is necessary. We have implemented this version of the blurred depth test in combination with depth peeling [Everitt 2001], which provides a depth ordered rendering. We utilized this implementation in our line editing tool that is used for the creation of the detail strokes in our system. Here, the blurred depth test offers strong spatial relations of object parts and the already drawn line strokes (Fig. 3). In the following sections we show alternatives to interpret the fragment visibility f_v. Firstly, we utilize the blurred depth test for smooth shadow intensities. Secondly, we apply it as a smooth hidden line and surface removal for detail primitives, e.g. particles and line strokes. These applications are especially adapted to our rendering pipeline and thus, do not require a depth ordered rendering. Additionally, we apply a Gaussian low-pass filter to the depth map D and thereby produce smooth transitions between the depth values to further support the effect of the smooth step function and the smooth visibility changes. This procedure is especially useful for filigrane objects like plant models.

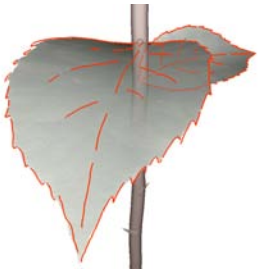


Figure 3: Application of the blurred depth test in our line editor.

In our implementation, we use a texture to store the depth map that is then utilized in the individual applications mentioned before. Obviously, the visual result strongly depends on the size and quality of this texture. When rendering the depth map D, a high ratio of noise can be realized. The reasons are the high signal frequency of filigrane objects and the missing antialiasing compared to the

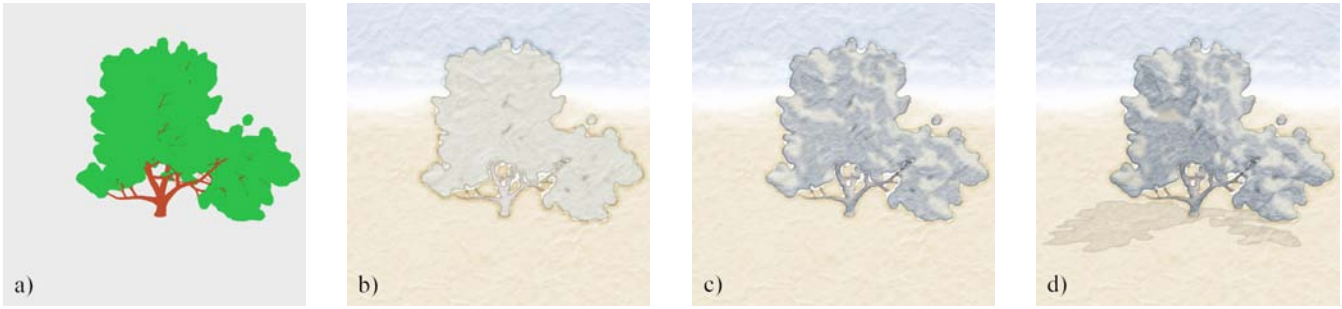


Figure 4: Watercolor Layers. (a) Segmentation of the scene, (b) plain watercolor, (c) diffuse lighting, and (d) soft shadowing effect.

color buffer. This noise propagates throughout subsequent rendering steps, e.g. the low-pass filtering and the blurred depth test itself, and finally disturbs the results. Consequently, we decided to explicitely render the depth information into a RGBA color texture using a very simple, but efficient coding scheme, which represents a compromise between visual quality and depth precision. Since we use linearly distributed depth values, the depth range $[0,1]$ is simply divided into four adjacent subranges with breaks at 0.25, 0.5, and 0.75. Each subrange is then coded into one of the RGBA color channel: Let $f_d \in [0,1]$ be the depth value of the fragment being processed, the coded color vector $c = [r,g,b,a]$ is computed by

$$c = \vec{c}_{scale} \cdot f_d + \vec{c}_{bias} \tag{3}$$

and clamped to the interval $[0,1]$. The scale and bias vectors are defined by $\vec{c}_{scale} = (4,4,4,4)^T$ and $\vec{c}_{bias} = (0,-1,-2,-3)^T$. The resulting color c can easily be decoded by

$$f_d = \frac{1}{4}\left(c_r + c_g + c_b + c_a\right). \tag{4}$$

Since we use four subranges of 8 Bit quantization, we achieve an overal depth precision of 10 Bit ($4 \times 2^8 = 2^{10}$), which is adequate for our application. Using this depth coding scheme, we achieve a high quality depth texture including antialiasing and bilinear texture access.

We conclude this section by discussing the blurred depth test. The introduced technique produces fragments of continuous visibility depending on a user defined depth range. This range defines the smooth transition from invisible fragments to visible fragments, which allows smooth occlusions and also a view into the inside of an object relative to its position. A drawback of our method is its computational expensiveness. Although, the major part can be implemented as hardware accelerated shader, multiple render passes are required. That means that the ordinary depth map must be rendered and stored. For the general case, a depth ordered rendering is required, which itself can be realized with a multipass rendering method like depth peeling. Since we replace the standard depth test with our own, we cannot benefit from the early occlusion culling, which finally results in an increased overdraw of the scene. Thus, the preferable applications of the blurred depth test are special rendering pipelines, mainly in the field of non-photorealistic rendering and technical illustrations.

3 Watercolor Layers

Our watercolor illustrations consist of several watercolor layers in combination with additional detail primitives. Hence, the user's vision of the final result strongly determines the overall watercolor appearance and the lighting and shadowing of the scene.

3.1 Imitation of Watercolor

For the watercolor appearance in our illustrations, we use our formerly published approach [Luft and Deussen 2005]. In the following, we briefly outline the proposed techniques and discuss the visual results that are achieved so far. In contrast to simulation based algorithms, this approach imitates the important natural effects of watercolor using simple and efficient algorithms that are implemented as hardware shaders. Thus, this method is especially applicable for real-time renderings of 3D scenes or other source material that can be easily segmented.

The algorithm starts with the segmentation of the scene, which is based on unique identifiers that are assigned to objects or groups of objects (Fig. 4(a)). Objects having the same identifier are rendered into a separate color channel of the RGBA color space, meaning that one or more ID-images are created each containing up to four individual ID-layers. These ID-layers are then processed to create the shape of the watercolor layer, to imitate a wet-on-wet and a wet-on-dry painting technique, and to incorporate the characteristic edge darkening effect. A paper structure is integrated to simulate the drawing medium and its influence on the watercolor pigment transportation. Finally, each watercolor layer obtains a static color specified by the user.

While the application of these algorithms provides a convincing impression of watercolor, the rendering of 3D objects suffers from its flat appearance and its weak depth cue (Fig. 4(b)). This is caused by the static color of the segmented areas and the absent influence of the lighting condition. The occlusions of the objects provide the only depth cue. Unfortunately, this depth cue can be distorted due to a merging of objects that belong to the same watercolor layer. As a consequence, the rendering of objects using the original watercolor shading algorithms leads to smooth results with a natural watercolor painting appearance, but with few details.

3.2 Diffuse Lighting

In order to improve the depth cue by integrating a diffuse lighting condition, we use the cool-to-warm shading approach proposed by Gooch et al. [1998]. This shading algorithm combines two color ramps: one for the hue shifting and another for the object color. For our application, we create these color ramps manually considering the object color and the nature of real watercolor (Fig. 4(c)).

The overall tone of the final rendering is produced by the hue shifting color ramp. Here, the user may choose a classic cool-to-warm

shading, sepia tones or any other color combination depending on the planned result (Fig. 5). The second color ramp, the object color ramp, depends on the type of the object and is also selected by the user. Here, usually a transition from the object color to a slightly darker tone is produced. In our system, the color ramps additionally include an alpha channel describing the transparency. By changing this transparency, the user can choose the saturation and the intensity of the final result.

Having these color information, we now compute the final color ramp as defined in the work by Gooch et al. In order to access this color ramp, we also evaluate the standard diffuse lighting term. For optimization, this lighting term is normalized to [0,1] and rendered into a texture, the so-called *lighting map*, that is then used for all watercolor layers that have to be processed. Consequently, the user specifies one hue shifting color ramp for the overall scene and individual object color ramps instead of one static color for each watercolor layer. This procedure results in slight color shadings resembling results that are achieved with a wet-on-wet painting technique.

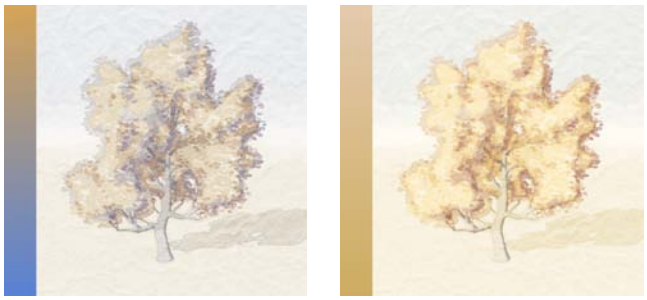


Figure 5: Diffuse Lighting. Comparison of two different hue shifting color ramps and their final results.

3.3 Soft Shadowing Effects

For the impression of soft shadows, we create another watercolor layer that is semi-transparently painted onto the existing watercolor layers (Fig. 4(d)). The creation of smooth and stable shadows that are based on a shadow map strongly depends on the resolution of this shadow map [Reeves et al. 1987]. We can circumvent this problem by computing physically incorrect, but visually convincing soft shadows using the blurred depth test: we first create the shadow map, apply a Gaussian filter, and then utilize the blurred depth test during the rendering of the shadow intensity map (Fig. 6(a)). During the rendering of the soft shadows, the parameter δ of the blurred depth test determines the shadow appearance, meaning that we can adjust the softness of the shadows continuously. Here, the application of a Gaussian filter before utilizing the blurred depth test is especially important for creating the smooth edges of the shadows. The parameter β is used to compensate quantization errors of the shadow map, and thus to avoid false self shadowing in the scene.

More specific, let $f_s \in [0,1]$ be the resulting shadow intensity of the fragment f being processed, $D(f_x, f_y) \in [0,1]$ the according depth value in the shadow map, and $f_d \in [0,1]$ the according depth in light space. We replace the original binary shadow computation by the already mentioned smooth step function of the blurred depth test:

$$f_s = \Delta step(-\delta, +\delta, D(f_x, f_y) - f_d + \beta). \quad (5)$$

The resulting intensity map is then processed equivalently to the existing watercolor layers using the watercolor shading algorithm.

This procedure of rendering soft shadows also produces an alternative illumination of complex plant models for the photorealistic rendering (Fig. 6(b)). Here we achieve a smooth lighting with smooth shadowed regions that significantly decreases the aliasing artifacts of low resolution shadow maps, and thereby supports the frame-to-frame coherence during an animation.

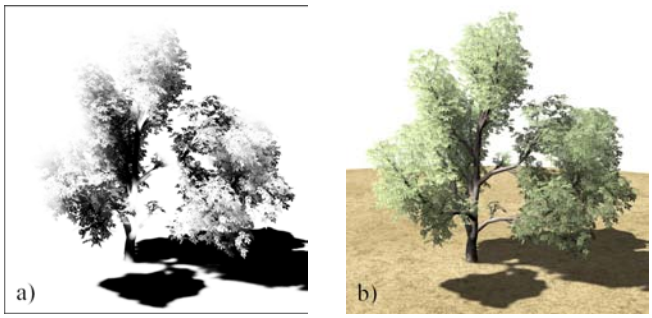


Figure 6: Soft Shadows. Comparison of (a) the shadow intensity map that is used for the creation of the shadow watercolor layer, and (b) a photorealistic rendering using our soft shadow algorithm.

4 Scene Modeling

A drawing process is always an abstraction and simplification of the motif. This is especially true for watercolor paintings of vegetation, since here the details are very complex and delicate. Through the use of different brushes, and the careful selection of watercolors and techniques, the artist is able to create a convincing simplification that is rich of information. We imitate this procedure by a simplification and abstraction of the model geometry and texture, while at the same time enhancing the final result with additional details.

4.1 Abstraction and Simplification

While modeling a scene, special consideration must be given to the color and shape of the resulting watercolor layers. The color is chosen by the user while the shape of the watercolor layers is described by the corresponding intensity map that is based on the already described segmentation scheme. Thus, the granularity and subtlety of the watercolor layers is given directly by the complexity of the modeled geometry.

Since the original plant models are normally used for a photorealistic rendering, they must be simplified and adapted to our purpose. This preprocessing depends on the size and type of the plant. Smaller plants are generally used without modification, because here we want to reproduce individual leaves or blossoms. Thus, we use the transparency information that is contained in the photorealistic textures to preserve special shape details. In contrast, the preprocessing of the trees produces an overall shape of the foliage without individual leaves. Here, we define implicit surfaces to abstract and simplify the foliage of our tree and shrub models (Fig. 7). The trunks and branches are used unmodified.

The implicit surfaces are described by a number of constraint points c_j that each has a radius of influence r_j. The constraint points correspond to the original vertices of the foliage geometry. The influence of one single constraint point c_j at a point p is described by a density function $D_j(p)$ that is defined as

$$D_j(p) = \left(1 - \left(\frac{\| p - c_j \|}{r_j}\right)^2\right)^2 \quad (6)$$

for $\| p - c_j \| \leq r_j$, otherwise as $D_j(p) = 0$. This density function was proposed by Murakami and Ichihara [1987], but with an additional wheight. The implicit surface arises from a summation of the density functions for all constraint points:

$$F(p) = \sum_j D_j(p) - T. \quad (7)$$

Thus, the implicit surface $F(p) = 0$ is defined as those points p where the summation of density functions equals the threshold T. The influence radius r_j of the constraint points and the threshold T are empirically chosen, because they strongly depend on the number and density of the constraint points, and also on the height h of a tree model. Finally, we triangulate the implicit surface and optimize the resulting mesh. Thus, a flexible and highly simplified geometry can be created. Furthermore, this approach significantly supports the lighting of the trees by providing smooth surface normals and a very vivid appearance. Table 1 provides a comparison between different levels of simplification.

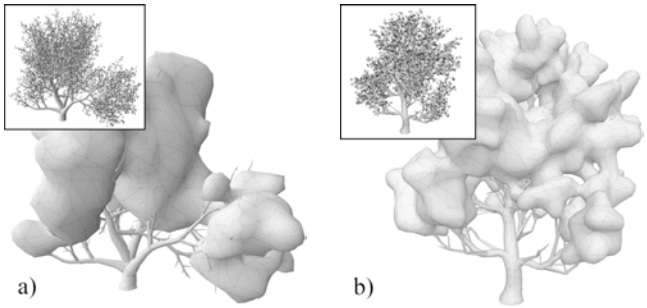


Figure 7: Abstraction and Simplificiation. Comparison of two tree models and its simplification based on implicit surfaces with a different degree of abstraction.

Figure	#tri foliage	#tri implicit surface	T	r_j	h
7 (a)	56952	3538 (≙6.2%)	83.3	2.8	37.1
7 (b)	56250	9494 (≙16.9%)	20.0	1.2	23.8

Table 1: Abstraction und simplification of tree models. Comparison of the triangle count and the used parameters for the creation of the implicit surfaces. The abstraction, and thus the reduction of the triangle count is given as percentage.

4.2 Detail Layers for the Tree Foliage

For the reproduction of the delicate structure of the foliage, we integrate additional watercolor layers that give the impression of individual leaves. Similar to the paint strokes of Meier [1996], these watercolor layers are based on a 3D particle set providing for a high frame-to-frame coherence. However, instead of rendering the particles directly as semi-transparent primitives representing individual leaves, we use them to create a number of intensity maps. These intensity maps are then processed equivalently to the existing watercolor layers using the watercolor shader. This approach is inspired by the characteristic of natural watercolor, where small details finally form larger areas of watercolor because of merging before drying. Our particles form specific areas that are rich of detail, and that have a homogenous structure. In contrast, an instantaneous rendering of the particles, e.g. by applying precomputed watercolor-like textures, leads to unnatural, cluttered results.

The lighting of the detail layers is incorporated in several ways. First, the resulting watercolor layers are modulated equally to the existing watercolor layers using the lighting map. To further support the lighting and to emphasize the structure of the tree, we try to imitate another important characteristic that constitutes the dynamics of watercolor paintings: the glazing effect. Glazing describes the composition of several thin washes that are painted on top of each other after each layer has dried thoroughly. As a consequence, the color pigments are blended optically instead through a physical, blurred mixing. Related to our detail primitves, we distiguish three individual intensity maps representing bright, medium, and dark areas of the foliage (Fig. 8). Each of these intensity maps is used to create an individual watercolor layer, which are then semi-transparently composed resembling the glazing effect. By using three individual intensity maps, we follow the often applied approach of toon-shading that also uses only a few discrete colors to represent dark and highlighted areas.

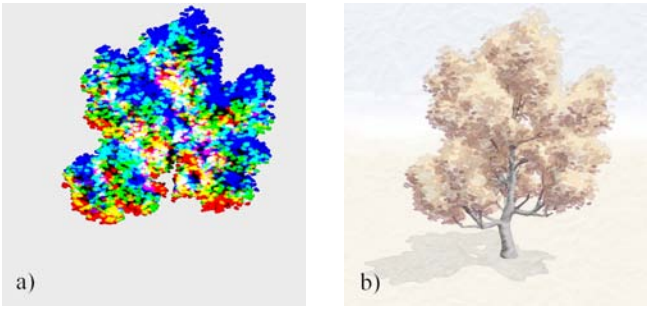


Figure 8: Detail Layers for the Tree Foliage. Comparison of (a) the intensity maps rendered in the RGB color space representing bright (blue), medium (green), and dark (red) leaves, and (b) the finally watercolored intensity maps. This example tree contains 17000 particles. We defined $\omega_{quantity} = 30.0$ and $\omega_{spread} = 0.07$, while the tree height is given as $h = 23.8$.

Particle Creation: The creation of the particles is performed during a preprocessing step. First, we compute the number of particles N proportional to the surface area of the corresponding bounding sphere:

$$N = \omega_{quantity} \cdot h^2. \quad (8)$$

Thus, it depends on the tree height h and a user defined attribute $\omega_{quantity}$. It is an import characteristic that the smaller the individual particles and the fewer particles are created, the coarser and looser are the resulting watercolor layers.

Each particle is described by a position p, a normal vector $\hat{n}$, a size s, and a 2D texture representing the shape of the particle. The position of a particle p is based on a corresponding vertex v of the implicit surface. An additional random influence $\hat{r}$ is integrated to place the particles:

$$p = v + \hat{r} \cdot \omega_{spread} \cdot \sqrt{h}. \quad (9)$$

The vector $\hat{r}$ is a normalized vector of arbitrary direction that is weighted by the user specified parameter ω_{spread}, and an additional,

height-dependent scale factor that is empirically defined as $\sqrt{h}$. As a consequence, the particles are no longer placed directly onto the foliage surface, but they are placed rather loosely and in a diversified manner around the foliage surface. The normal vector $\hat{n}$ is the second essential attribute of a particle, since it defines its affiliation to one of the three intensity maps. The normal vector is directly obtained from the normal vector $\hat{n}_v$ of the corresponding vertex v of the implicit surface. Thus, the smooth and vivid lighting of the implicit surface is transferred to the detail primitives providing a strong cue for the structure of the foliage. The size s of the particles is determined during rendering, since it depends on the distance to the camera. There are a lots of possibilities to compute this dependency, e.g. linear or exponential. We use a simple linear equation depending on the particles distance to the viewer $d \in [0,1]$:

$$s = (1-d) \cdot \omega_{scale} + \omega_{bias}. \tag{10}$$

The user defined parameters ω_{scale} and ω_{bias} represent the increase of the size and the minimum size of the particles. In our implementation, s is a resolution dependant parameter and gives the particle width and height in pixels. For the results rendered at a resolution of 720×720 pixels, we defined $\omega_{scale} = 200$ and $\omega_{bias} = 5$.

Rendering: As already mentioned, we create three individual intensity maps that are each based on the 3D particle set. Each intensity map represents an individual watercolor layer showing bright, medium or dark areas of the foliage. The creation of these three intensity maps is performed during one rendering pass by rendering each particle into one of the RGB color channels (Fig. 8(a)). Consequently, the assignment of a specific particle to one of the three intensity maps is handled by the particle color $c = [r,g,b]$, and it depends on the alignment of the particle normal $\hat{n}$ to the light direction $\hat{l}$. Thus, we use the standard diffuse lighting term $d = \hat{n} \cdot \hat{l}$ to perform a classification:

$$c = [r,g,b] = \begin{cases} (1,0,0) & d \le t_0 \\ (0,1,0) & t_0 < d \le t_1 \\ (0,0,1) & t_1 < d \end{cases} . \tag{11}$$

The user can change two thresholds t_0 and $t_1 \in [-1,1]$ that specify the crossover of the dark-to-medium transition, and that of the medium-to-bright transition. Our examples are created by writing $t_0 = -0.5$ and $t_1 = 0.1$.

The visibility of the particles is determined using the blurred depth test. However, the resulting fragment visibility f_v as described in Sec. 2 is not directly treated as the fragment transparency. We rather have to consider two important issues while rendering the particles. First, the particles do not represent the final rendered details, but they are rather used to produce a number of intensity maps, which describe the shape of the final watercolor layers. Second, the color of the particles represents a classification to one of the intensity maps. Consequently, we can not utilize a standard blending function naturally used for semi-transparent objects, since then the crosstalking between different particles having different colors would disturb the result. We use an additive blending function to avoid this effect: Let f_v be the visibility of the fragment f being processed, c the particle color, and $b(f_x, f_y)$ the existing background color. The resulting background color $b'(f_x, f_y)$ is calculated by

$$b'(f_x, f_y) = f_v \cdot c + b(f_x, f_y) \tag{12}$$

and subsequently clamped to the interval [0,1]. This blending function actually treats the semi-visibility of the fragments as intensities and accumulates them, while not mixing the color channels. Thus, a depth ordered rendering of the particles is not necessary.

The application of the blurred depth test allows the particles to be blended very smoothly as they are occluded by other objects. As a result, we achieve a smooth, controllable hidden surface removal, which avoids popping artifacts. Futhermore, the blurred depth test allows a view into the inside of an object. This feature is essential for our particles, since they are not located directly on the foliage surface, but also inside the geometry. At the same time, the large differences in the visual density of areas that are viewed tangentially and areas that are viewed in direction of the surface normal, are avoided by the blurred depth test (Fig. 9). Here, the parameter δ of the blurred depth test specifies, up to which depth the particles are still visible. To achieve proper results, the blurred depth test has to be shifted using the parameter $\beta = \delta$. The resulting visibility of the particles provides strong depth cues and especially motion cues during a camera movement.

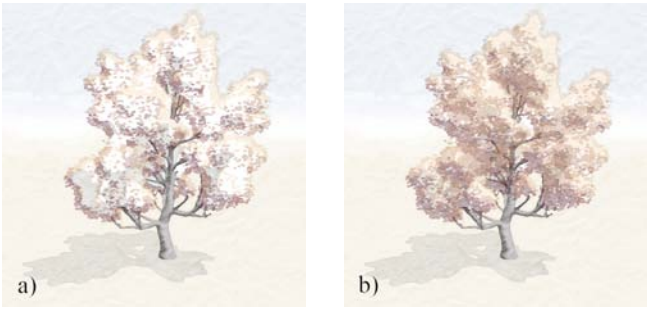


Figure 9: Visibility Computation. Comparison of the visual density by (a) using the binary depth test, and (b) using the blurred depth test for the visibility computation of the particles.

After rendering the intensity maps, we apply the watercolor shader and achieve controllable detail layers that represent the complex structure of the foliage. The colors of the watercolor layers are specified by the user and additionally modulated by the lighting map.

4.3 Intensity Textures for Smaller Plants

The identification of smaller plants strongly depends on the exact shape, color, and texture. A segmentation solely based on the object geometry as performed for the trees is not sufficient. By using the alpha channel that is commonly used for the photorealistic textures, the exact shape of the leaves and blossoms can be preserved. Thus, basic watercolor layers can be created that are based on a per object segmentation, and that already equal the desired shapes of the individual plant parts (Fig. 10(a)). To further increase the granularity of the segmentation and to achieve a diversity of color and surface structure, we introduce *intensity textures*.

Creation: Intensity textures store a number of detail maps to create additional watercolor layers representing specific surface structures of a plant. Here, we distinguish between plant parts, e.g. blossoms or leaves, and utilize individual intensity textures for them. The alpha channel of an intensity texture is copied from the photorealistic texture and used for the basic watercolor shape of an object.

By storing one detail map into one color channel of the RGB color space, an intensity texture contains up to three different intensity maps, which provide an adequate modeling freedom for all plant

Figure 10: Intensity Textures for Small Plants and Flowers. (a) Rendering based on the geometry combined with the alpha channel, (b) the manually created intensity textures (bottom), and (c) the final rendering using intensity textures.

models. If more than three watercolor layers are required, additional intensity textures can be used. This simple coding scheme allows for an easy creation of the intensity textures using standard painting applications and provides the possibility of overlapping watercolor layers. As a basis for the intensity textures, we use the structure information given by the photorealistic textures. Here, specific areas are traced and simplified, and subsequently stored in the RGB color channels. This abstraction can be performed manually or automatically by a quantization of the photorealistic textures using only few discrete colors. However, the manual creation of the intensity textures provides full control over the final result, and allows for a higher abstraction, and a specific adaption to natural watercolor paintings (Fig. 10(b)).

Rendering: The plants are rendered by applying the intensity textures and using the standard depth test. For each intensity texture one rendering pass is required. The results contain a number of intensity maps that are extracted and subsequently processed by the watercolor shader. The color of the individual color layers again is specified by the user and modulated by the lighting map. By using intensity textures, we achieve a wide diversity of watercolor layers for the plant leaves and blossoms, while preserving their original shape (Fig. 10(c)). This technique can also be utilized to achieve a more detailed appearance of the trees' bark.

An advantageous effect when using a basic watercolor layer in combination with additional watercolor layers based on intensity textures is the automatic level of detail for distant views. Details that were added using intensity textures are smoothly faded when moving away from the camera, whereby the basic color shape still remains visible (Fig. 11). This is caused by the watercolor shading process, which applies a Gaussian filter to the intensity maps. Since this Gaussian filter is a low pass filter, small areas are smoothly faded out when their dimensions drop below the filter kernel size.

4.4 Line Strokes as a Sketchy Basis

Similar to other drawings, watercolor illustrations are based on a sketchy or detailed line drawing of the scene. In our system we use line strokes to emphasize the shape and structure of leaves, blossoms or branches (Fig. 12). All line strokes are 3D primitives that are integrated into the modeled scene. The line strokes can be created manually or automatically using well known techniques, such as [Hertzmann and Zorin 2000; DeCarlo et al. 2003]. For the manual creation of line strokes, we have integrated an editor that allows the user to draw details directly onto the 3D object surface. In order to automate this procedure, we use the existing texture parameterization to duplicate the line strokes on all objects that have the same

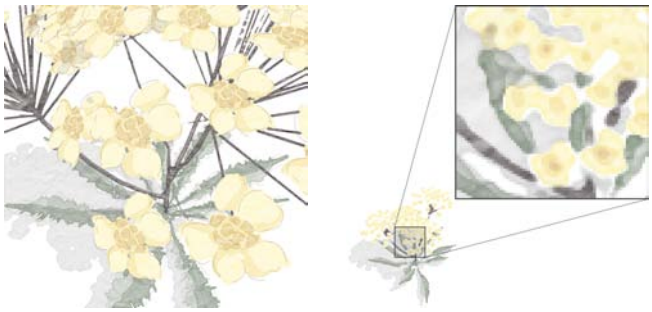


Figure 11: Intensity Textures for Small Plants and Flowers. Comparison of two different view distances that show the automatic level of detail effect.

texture. Thus, the user has to sketch only one leaf or blossom, and all other primitives of the same type are sketched by the editor accordingly. Because of the individually modeled shapes of the leaves and blossoms, the duplicating is hardly noticable. The automatic creation of line strokes is performed for contours of trunks, stalks or branches.

Figure 12: Line Strokes As Sketchy Basis. Comparison of two sketchy drawings showing the different visibility conditions and the application of the blurred depth test.

During rendering, we differentiate several conditions for the line stroke visibility depending on the occlusion, the camera position, and the light direction. For a smooth hidden line removal of the line strokes, we utilize the blurred depth test. Here, we directly interpret the fragment visibility f_v as the fragment transparency allowing us to render these semi-transparent line strokes using a standard blending function. The issue of a depth ordered rendering for semi-transparent objects is easily solved: While the spatial relation of surfaces is an important depth cue of a scene, the depth order of line strokes in a line drawing can not be visually reconstructed

	figure	#triangles	#details	#strokeVertices	#watercolor layers	light/shadow/depth map	intensity maps	watercolor	strokes	fps
plants	13(a)	55556	7	12687	7	18.0ms	10.0ms	8.9ms	1.4ms	26.1
	13(b)	1147	5	27513	7	7.7ms	7.6ms	9.0ms	1.8ms	38.3
	13(c)	3686	7	13340	6	8.6ms	7.8ms	8.9ms	1.4ms	37.4
trees	13(d)	45470	64209	46544	5	9.8ms	41.9ms	7.1ms	3.3ms	16.1
	13(e)	41190	26358	30144	5	8.9ms	21.4ms	6.7ms	3.1ms	24.9

Table 2: Comparison of the scene complexity and the rendering times. The column *#detail* gives the number of particles (in case of a tree) or the number of intenstiy textures (in case of a smaller plant). The column *#strokeVertices* gives the overall number of control points that form the detail strokes.

by the viewer. We exploit this characteristic by rendering the line strokes on top of the existing scene in the order they are created, which is comparable to the painter's algorithm. Consequently, we do not require a depth ordering when applying the blurred depth test. As a result, the line strokes are blended very smoothly, when they are covered by parts of the plant or by other objects. As an additional visibility condition, we fade out the line strokes when moving away from the camera. The manually created line strokes are subjected to other visibility conditions. Special consideration must here be given to the viewing direction, since a set of line strokes tend to cluster when viewed from a narrow angle. There are known techniques to prevent this disadvantage for non-real-time rendering, e.g. [Grabli et al. 2004; Winkenbach and Salesin 1994]. In our case, we simply fade out the line strokes according to the dot product of the view vector and the normal vector of the associated surface. Additionally, line strokes representing a hatching pattern can be specified as being light dependent. Then, we fade out the line strokes according to the dot product of the light direction and the normal vector of the associated surface.

5 Results

The results (Fig. 13) show a convincing imitation of natural watercolor illustrations, although our approach is completely different from a physically based simulation. Naturally, there are still significant differences in comparison to real watercolor paintings due to a missing pigment diffusion that produces spontaneous and subtle textures. Although we concentrated on real-time renderings with a limited resolution, it is also possible to produce high-resolution renderings for printing. Then special care must be given to the level of abstraction, which scales with the resolution of the intensity maps.

We tested our implementation on a 3.0 GHz Pentium IV with a GeForce 6800 graphics board. We use a multi-pass rendering procedure to render the lighting map, the shadow map, the depth map, and the intensity maps that are used for the watercolor color shading. These intermediate maps are rendered at a resolution of 520×520 pixels, which provides an appropriate balance between abstraction level, visual quality, and performance. The resolution of the final rendering is 720×720 pixels. Table 2 gives an overview of the scene complexity and the rendering times. The system performance depends on several factors: The computation time of the watercolor shading is almost constant. However, it scales with the number of watercolor layers and especially the resolution of the final rendering, since the watercolor shader is mainly based on fragment processing. For trees the number of particles strongly influence the rendering time, because here we applied the blurred depth test for the hidden surface removal. The comparison of *light/shadow/depth* shows that the rendering of small plant models is generally more costly, since a here multi-texture mapping is applied. The rendering time of the line strokes represents only a minor influence. While the line strokes of small plants are mainly created manually, the line stroke of the trees are contours and thereby computationally more expensive.

6 Conclusion

We presented techniques to create convincing high-quality watercolor illustrations of plants. In contrast to existing canvas simulations of watercolor, we focused on the real-time rendering of 3D scenes. We introduced methods that allow for abstracting the visual and geometrical complexity of the original, photorealistic plant models. To emphasize the structure, the texture, and the lighting of the scene, we additionally integrated detail layers that provide a wide application range for the individual tree and plant models. The blurred depth test was utilized to perform depth comparisons that are used for the visibility computations of the detail primitives and for the creation of a soft shadowing effect. As a result, we achieved smooth visibility transitions and a high frame-to-frame coherence during an animation.

References

BASTOS, C. C., GURREA, E. C., BAUSET, R. Q., GUIJARRO, J. H., AND QUINTANA, I. R. 2002. Acceleration techniques for non-photorealistic rendering of trees. In *SIACG '02: Proc. of the 1st Ibero-American Symposium in Computer Graphics*.

BAXTER, W. V., WENDT, J., AND LIN, M. C. 2004. IMPaSTo: A realistic model for paint. In *Proceedings of NPAR '04*, 45–56.

CHU, N. S.-H., AND TAI, C.-L. 2005. Moxi: real-time ink dispersion in absorbent paper. *ACM Transactions on Graphics 24*, 3, 504–511.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of ACM SIGGRAPH 97*, 421–430.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3, 848–855.

DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *Proceedings of ACM SIGGRAPH 2000*, 13–18.

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., NVIDIA. http://developer.nvidia.com.

FIORE, F. D., HAEVRE, W. V., AND REETH, F. V. 2003. Rendering artistic and believable trees for cartoon animation. In *Proceedings of Computer Graphics International '03*, 144–151.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of ACM SIGGRAPH 98*, 447–452.

GRABLI, S., DURAND, F., AND SILLION, F. 2004. Density measure for line-drawing simplification. In *Proceedings of Pacific Graphics '04*, 309–318.

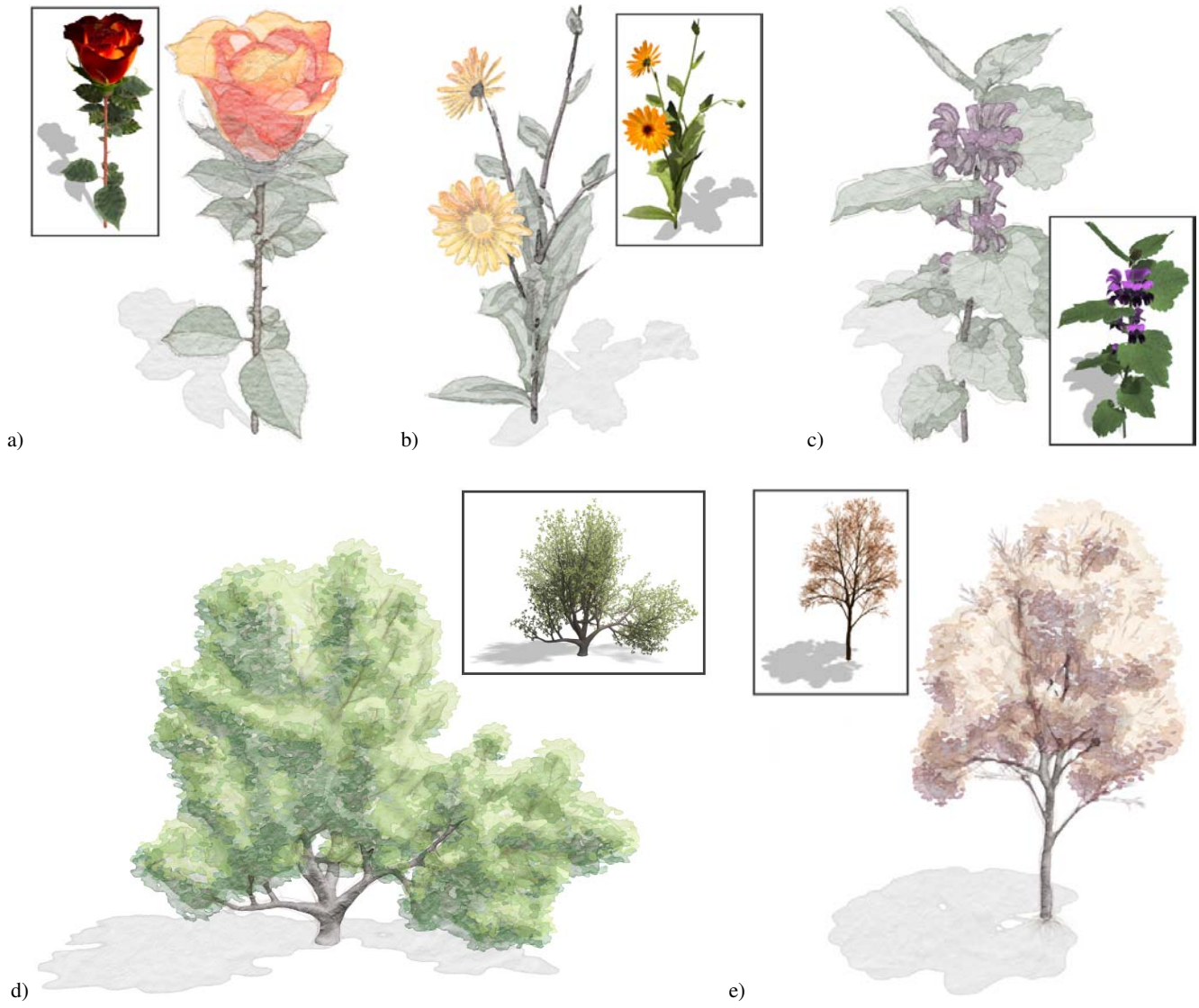


Figure 13: Results. Comparison of the original, photorealistic plant model and the final rendering using the introduced techniques.

GREENWORKS ORGANIC SOFTWARE, 2005. Homepage of the Xfrog modeling software. http://www.greenworks.de.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proceedings of ACM SIGGRAPH 2000*, 517–526.

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of ACM SIGGRAPH 98*, 453–460.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of ACM SIGGRAPH 99*, 433–438.

LUFT, T., AND DEUSSEN, O. 2005. Interactive watercolor animations. In *Proceedings of Pacific Graphics '05*, 7–9.

LUM, E. B., AND MA, K.-L. 2001. Non-photorealistic rendering using watercolor inspired textures and illumination. In *Proceedings of Pacific Graphics '01*, 322–330.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of ACM SIGGRAPH 96*, 477–484.

MURAKAMI, S., AND ICHIHARA, H. 1987. On a 3d display method by metaball technique. *Transactions of the Institute of Electronics, Information and Communication Engineers J70-D*, 8, 1607–1615. In Japanese.

REEVES, W. T., AND BLAU, R. 1985. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Proceedings of ACM SIGGRAPH 85*, 313–322.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *Proceedings of ACM SIGGRAPH 87*, 283–291.

SMALL, D. 1991. Simulating watercolor by modeling diffusion, pigment, and paper fibers. In *Proc. of SPIE '91: Image Handling and Reproduction Systems Integration*, vol. 1460, 140–146.

STRASSMANN, S. 1986. Hairy brushes. In *Proceedings of ACM SIGGRAPH 86*, 225–232.

VAN LAERHOVEN, T., LIESENBORGS, J., AND VAN REETH, F. 2004. Real-time watercolor painting on a distributed paper model. In *Proceedings of Computer Graphics International '04*, 640–643.

WILSON, B., AND MA, K.-L. 2004. Rendering complexity in computer-generated pen-and-ink illustrations. In *Proceedings of NPAR '04*, 129–137.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proceedings of ACM SIGGRAPH 94*, 91–100.

Real-Time Watercolor Illustrations of Plants Using a Blurred Depth Test

Thomas Luft Oliver Deussen