# Interactive watercolor rendering with temporal coherence and abstraction

Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, François X. Sillion

▶ **To cite this version:**

Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, François X. Sillion. Interactive watercolor rendering with temporal coherence and abstraction. International Symposium on Non-Photorealistic Animation and Rendering (NPAR), 2006, Annecy, France. inria-00510223

**HAL Id: inria-00510223**
**https://inria.hal.science/inria-00510223**

Submitted on 13 Oct 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Interactive watercolor rendering with temporal coherence and abstraction

Adrien Bousseau, Matt Kaplan, Joëlle Thollot and François X. Sillion
ARTIS GRAVIR/IMAG INRIA* Grenoble France

(a) (b) (c)

Figure 1: Various watercolor-like images obtained either from a 3d model (a,b) or from a photograph (c) in the same pipeline.

## Abstract

This paper presents an interactive watercolor rendering technique that recreates the specific visual effects of lavis watercolor. Our method allows the user to easily process images and 3d models and is organized in two steps: an abstraction step that recreates the uniform color regions of watercolor and an effect step that filters the resulting abstracted image to obtain watercolor-like images. In the case of 3d environments we also propose two methods to produce temporally coherent animations that keep a uniform pigment repartition while avoiding the shower door effect.

**Keywords:** Non-photorealistic rendering, watercolor, temporal coherence, abstraction.

## 1 Introduction

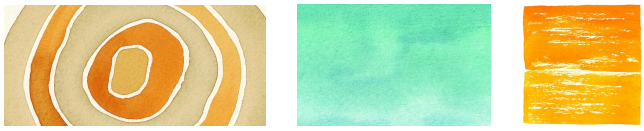
Watercolor offers a very rich medium for graphical expression. As such, it is used in a variety of applications including illustration, image processing and animation. The salient features of watercolor images, such as the brilliant colors, the subtle variation of color saturation and the visibility and texture of the underlying paper, are the result of a complex interaction between water, pigments and the support medium.

In this paper, we present a set of tools for allowing the creation of watercolor-like pictures and animations. Our emphasis is on the development of intuitive controls, placed in the hands of the artists, rather than on a physically-based simulation of the underlying processes. To this end, we focus on what we believe to be the most significant watercolor effects, and describe a pipeline where each of these effects can be controlled independently, intuitively and interactively.

Our goal is the production of watercolor renderings either from images or 3d models, static or animated. In the case of animation rendering, temporal coherence of the rendered effects must be ensured to avoid unwanted flickering and other annoyances. We describe two methods to address this well-known problem that differ in the compromises they make between 2d and 3d.

In the following, we will first review the visual effects that occur with traditional watercolor and present related work. Our pipeline is then described for images and fixed viewpoint 3d models, followed by our methods that address the temporal coherence of animated 3d models. Finally, we will show some results before concluding.

*ARTIS is a team of the GRAVIR/IMAG research lab (UMR C5527 between CNRS, INPG, INRIA and UJF), and a project of INRIA.



(a) (b) (c)

Figure 2: Real watercolor (©Pepin van Roojen). (a) Edge darkening and wobbling effect, (b) pigments density variation, (c) dry brush.

## 2 Watercolor effects

Curtis et al. [1997] listed the specific behaviors which make watercolor so distinct among painted media, that probably account for its popularity. We describe here the most significant and specific effects in watercolor, as illustrated in Figure 2 by pictures of real watercolor found in [van Roojen 2005].

First, the pigments are mixed with water but do not dissolve

totally. The result is that, after drying, even on a totally smooth paper, the pigment density is not constant (see Fig. 2-(b)). This manifests itself at two different scales: on the one hand there are low-frequency density variations due to a non homogeneous repartition of water on the canvas: the turbulence flow; on the other hand there are high-frequency variations due to non-homogeneous repartition of pigments in water: the pigment dispersion. In addition, the grain of the canvas can also introduce density variations since the pigments are deposited in priority in cavities of the paper.

Second, the fluid nature of watercolor creates two effects along the borders of colored regions: edge darkening due to pigment migration and a wobbling effect due to paper grain (see Fig. 2-(a)). In the case of a "dry brush" the paper grain also creates some white holes in the brush strokes when a nearly-dry brush only touches raised areas of the paper (see Fig. 2-(c)).

Finally, watercolor is usually created by a human, using a brush, often of a large size, and therefore represents a scene in a more abstracted way than a photograph. We believe that abstracting the shape, colors or illumination of the original scene is crucial to keeping the aesthetics of watercolor. Although this is true of most painting and drawing styles, the technical constraints of watercolor typically result in less precise details than other techniques.

We do not target the diffusion effect, also described by Curtis, for two main reasons. First this effect is often avoided by the artist when trying to depict "realistic" scenes. In such a case the lavis technique (wet on dry) is preferred. Second, this effect typically requires a fluid simulation, and would certainly not be temporally coherent. We thus leave this type of visual effect for future work and instead concentrate on uniform-color regions.

# 3 Related Work

Previous research in watercolor rendering can be grouped into two broad categories: Physical simulation of the media and image filters.

Physical simulations attempt to simulate all fluid, pigment and paper interactions. Curtis et al. [1997] described the critical effects created by watercolor media and presented a system that created convincing watercolor renderings. Their work was an extension of Small's [1991] connection machines. Curtis' method, while convincing, is extremely computationally expensive. Van Laerhoven et al. [2004] demonstrated an interactive method for rendering brush strokes painted by the user. Rendering eastern style inks is a related area that may be considered in this category [Chu and Tai 2005]. In this case the focus is more on the dispersion effects (wet on wet technique) than on lavis technique. Moreover, watercolor is a media consisting of a complex series of interactions between multiple layers of paint that diffuse pigment and water in ways that relate precisely to the method and area of their application. It is not clear that such methods are suitable for animation since the positions of painted regions in relation to one another necessarily change in animations. The interdependence of the qualities that make an image look like a watercolor painting may be too high to allow true temporal coherence.

Image processing filters consist of using image, and other G-buffers (like depth or normal buffer) as input to create a watercolor-like image that is based on an empirical recreation of the relevant effects rather than a physical simulation. These methods are fast and can typically be performed at interactive speeds but lack the richness of physical simulations. Lum and Ma [2001] present a multi-layer approach inspired by watercolor painting. Their work is performed in 3d object space to ensure frame-to-frame coherence. Line integral convolution is applied along the principal curvature directions to render brush-like textures. Burgess et al [2005] created a similar system, using Wyvill noise to create stroke textures. Luft and Deussen [2005] abstract ID images at their boundaries to create flow patterns and edge darkening. These effects are typically isolated in real paintings and do not occur simultaneously as in their method. Furthermore, abstracting at the object level removes all detail about surface shape. Lei and Chang [2004] demonstrate a rendering pipeline using a GPU to create watercolor effects at interactive speeds. They require the user to "pre-paint" each object using a lit sphere interface. All flow and granulation effects are then encoded in a 1d texture and rendered in a manner similar to a toon rendering [Lake et al. 2000]. Encoding flow effects in a 1d texture is too abstract for adequate detail on a smoothly changing 3d surface. Furthermore, lit sphere lighting specification may be prohibitive for a large number of objects or changing light sources.

Johan et al. [2005] used watercolor principles to create brush strokes in image space but they do not present a method for watercolor rendering.

Much of this previous work makes use of the Kubelka-Munk model to simulate the composition of pigments. This model is physically based but limits the choices of colors to a predefined pigment library. In the case of an image input (like in [Curtis et al. 1997]) a color reconstruction has to be done to choose the right pigments for a given color. We propose in this work a more intuitive color treatment that stays close to the original color of the input.

On the other hand, temporal coherence of brush strokes has been an active area of research since there is an inherent conflict between a 3d motion field and marks created in a 2d plane. Various methods has been presented either in object space like [Meier 1996] or in image space like [Wang et al. 2004]. The shower door problem between the canvas and objects in the scene has been addressed in [Cunzi et al. 2003; Kaplan and Cohen 2005].

Watercolor is, in our opinion, not exactly a mark-based rendering technique because the pigments can not be considered as single brush strokes; on the other hand the pigments have to follow the objects in the scene so we cannot directly apply the dynamic canvas technique as in [Cunzi et al. 2003]. The temporal coherence methods we present are extensions of these previous techniques.

# 4 Single-image pipeline

Our work is directly inspired from previous techniques, however we do not restrict ourselves to physically realizable images. Instead, we purposely lift the constraints inherited from the physics of real watercolor since they can be avoided by the use of the computer. We prefer to offer a system that reproduces watercolors visual effects, and let the user control each of these effects independently, even if the particular combination that results would not make sense in the physical world. Thus, we do not need to use the Kubelka-Munk pigment model or a physical fluid simulation as in previous approaches.

In order to obtain the effects mentioned in Section 2 we propose a unified framework using the pipeline shown in Figure 3 and the accompanying videos, which takes as input either a 3d scene or a single image.

Our contributions are (a) to provide the user with various abstraction steps allowing the creation of the uniform color regions needed in watercolor, and (b) to provide a simple color combination model that ensures a plausible result while avoiding physical constraints.

We now describe each step of the pipeline. For clarity of exposition, we begin with the second stage (watercolor effects) before considering the various abstraction possibilities.

## 4.1 Watercolor effects

The input to this part of the pipeline is an abstracted image that can come either from a 3d rendering or from a processed photograph. Figure 4 illustrates the successive steps in the case of a 3d model.

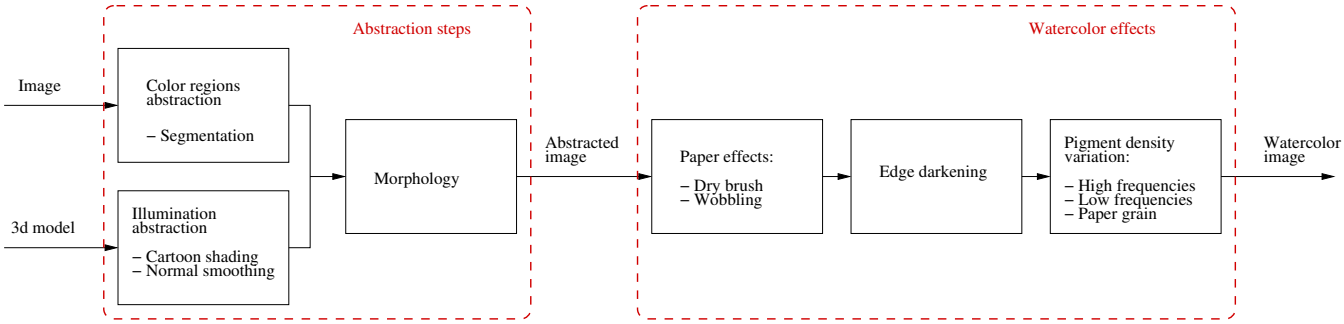



Figure 3: Static pipeline: Our pipeline takes as input either a 3d model or a photograph, the input is then processed in order to obtain an abstracted image and finally watercolor effects are applied to produce a watercolor-like image.

The main effect in watercolor is the color variation that occurs in uniformly painted regions. We first present the technique we use to reproduce these variations by modifying the color of the abstracted image and then we describe all the effects specific to watercolor.

#### 4.1.1 Color modification

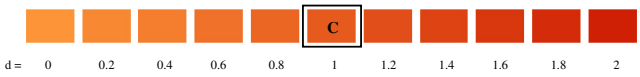



Figure 5: Darkening and lighting of a base color $C$ (here $(0.90, 0.35, 0.12)$) using a density parameter $d$.

As mentioned earlier we chose not to use a physical model to describe color combinations. The essential positive consequence of this choice is that the user can freely choose a base color (think of a virtual pigment), whose variation in the painted layer will consist of darkening and lightening, as shown in Figure 5.

We build an empirical model based on the intuitive notion that the effects due to pigment density can be thought of as resulting from light interaction with a varying number of semi-transparent layers. Consider first a color $C$ ($0 < C < 1$) obtained by a uniform layer of pigment over a white paper, we can interpret this as a subtractive process where the transmittance of the layer is $\tau = \sqrt{C}$ (the observed color is the transmittance squared since light must travel through the layer twice). Adding a second layer of the same pigment therefore amounts to squaring the transmittance, and hence also the observed color.

We need to be able to continuously modify the color, rather than proceeding with discrete layers or squaring operations. A full, physically based analysis could be based on the description of a continuous pigment layer thickness and the resulting exponential attenuation. Rather we obtain similar phenomenological effects by introducing a pigment density parameter $d$, which provides the ability to continuously reinforce or attenuate the pigment density, where $d = 1$ is the nominal density corresponding to the base color $C$ chosen by the user. We then modify the color based on $d$ by removing a value proportional to the relative increase (respectively decrease) in density and the original light attenuation $1 - C$. The modified color $C'$ for density $d$ is thus given by

$$\begin{aligned} C' &= C(1-(1-C)(d-1)) \\ &= C-(C-C^2)(d-1) \end{aligned} \quad (1)$$

Note that the value $d = 2$ corresponds to the two-layer case discussed above and to a squared color.

#### 4.1.2 Pigment density variation

As described in section 2 the density of pigments varies in several ways. We propose to add three layers, one for each effect: turbulent flow, pigment dispersion and paper variations. No physical simulation is performed. Instead, each layer is a gray-level image whose intensity gives the pigment density as follows: An image intensity of $T \in [0,1]$ yields a density $d = 1 + \beta(T - 0.5)$, which is used to modify the original color using formula 1. $\beta$ is a global scaling factor used to scale the image texture values and allow arbitrary density modifications. The three effects are applied in sequence and each can be controlled independently (see Figure 4-(g,h,i)). The computation is done per pixel using a pixel shader. The paper layer is applied on the whole image whereas the other layers are applied only on the object projection.

The user is free to choose any kind of gray-level textures for each layer. We found it convenient to use Perlin noise textures [Perlin 1985] for the turbulent flow, a sum of gaussian noises at various scales for pigment dispersion and scanned papers for the paper layer.

#### 4.1.3 Other watercolor effects

The pigment density treatment recreates the traditional texture of real watercolor. We then add, as in most previous techniques, several typical watercolor effects:

**Edge darkening:** Edges are darkened using the gradient of the abstracted image (see Figure 4-(f)). The gradient is computed on the GPU using a fast, symmetric kernel:

$$\Delta(p_{x,y}) = |p_{x-1,y} - p_{x+1,y}| + |p_{x,y-1} - p_{x,y+1}|$$

The gradient intensity (used to modify the pigment density using formula 1) is computed by averaging the gradient of each color channel.

Any other gradient computation could be used depending on the compromise between efficiency and realism needed by the user. Our method is clearly on the efficiency side.

**Wobbling:** The abstracted image is distorted to mimic the wobbling effect along edges due to the paper granularity. The $x$ offset is computed with the horizontal paper gradient, and the $y$ offset with the vertical paper gradient (see Figure 4-(e)). The user can change the paper texture resolution: Indeed, the real process involves complex interactions and using the paper texture directly may produce details at too fine a scale. By decreasing the resolution we keep the overall canvas structure while decreasing the wobbling frequency. We could have also used another texture for this effect as in [Chu

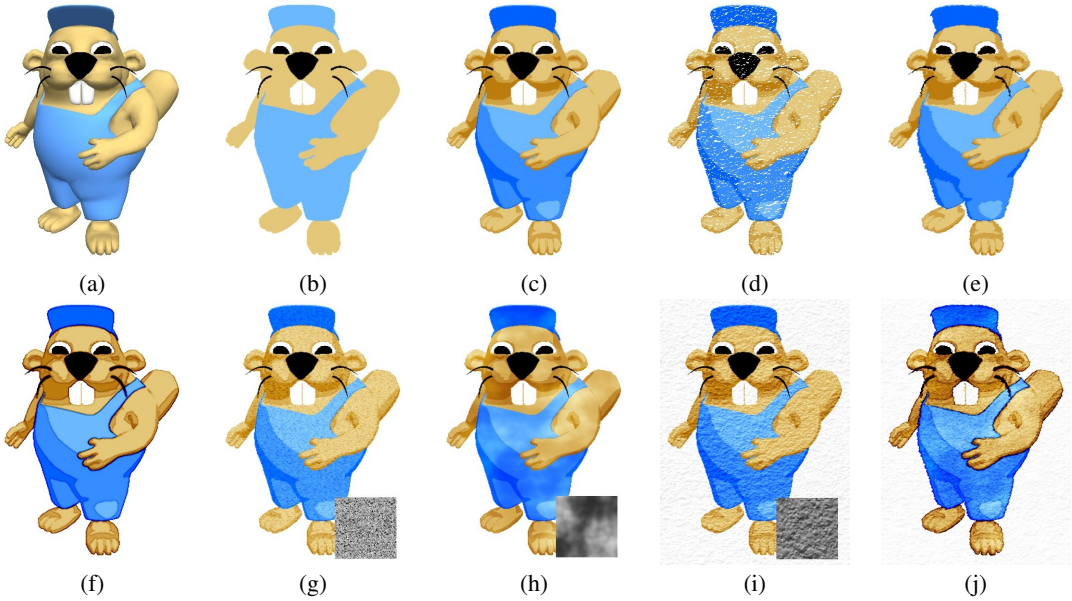


Figure 4: Static pipeline illustrated for a 3d model: (a) 3d model, (b) original color chosen by the user, (c) toon shading, (d) dry brush (not kept for this example), (e) wobbling, (f) edge darkening, (g) pigment dispersion layer, (h) turbulence flow layer, (i) paper, (j) final result.

and Tai 2005]. As a side effect, the paper offset adds noise along edges, which decreases the aliasing effect.

**Dry brush:** The dry brush effect occurs in watercolor painting when a nearly-dry brush applies paint only to the raised areas of a rough paper. We simulate this effect with a simple threshold of the paper height (represented by the paper texture intensity) as illustrated in Figure 4-(d).

This effect has not proven to be very useful in practice because it is applied to the whole scene. However one can easily add color conditions or masks in the pipeline to selectively apply such an effect.

## 4.2 Abstraction

The watercolor rendering obtained by our pipeline generally seems too detailed compared to a painting done by hand. Indeed, the small details produced by a 3d rendering are often "too perfect", which is a common issue in computer graphics. Similarly, if we take a photograph as input, the result appears too precise.

We propose to use various abstraction steps to simplify the shapes and to abstract the color and the illumination of our input data before adding the watercolor effects already described. The first abstraction step aims at reproducing uniform color regions by abstracting the illumination in the case of a 3d model (see Section 4.2.1) or segmenting the original image (see Section 4.2.2) and the second abstraction step aims at discarding the small regions that remains after the first step (see Section 4.2.3).

### 4.2.1 Abstraction of the illumination of a 3d model

In watercolor painting, shading is usually performed by compositing pigment layers. In wet-in-wet painting, this superposition results in a smooth shading, whereas in wet-on-dry painting, it results in sharp color areas. To obtain these shaded areas, a toon shader as described in [Lake et al. 2000] is used. A smooth transition between areas can be obtained by smoothing the toon texture (Figure 6). We apply toon shading using our color modification method so that it stays consistent with the rest of the pipeline. The user thus simply has to provide a gray-level 1d texture.

Figure 6: Toon shading and the corresponding toon texture

Depending on the original 3d model and even with a toon shading there are still sometimes lots of details. Therefore, we propose to smooth the normals of the model before computing the illumination. The normal smoothing is applied by replacing a vertex normal by the average of its neighbors normals. This process can be repeated to obtain a higher smoothing. Normal smoothing removes shading details but preserves silhouettes (Figure 7).

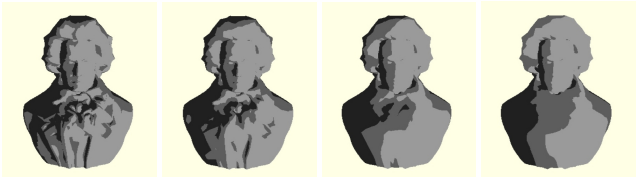
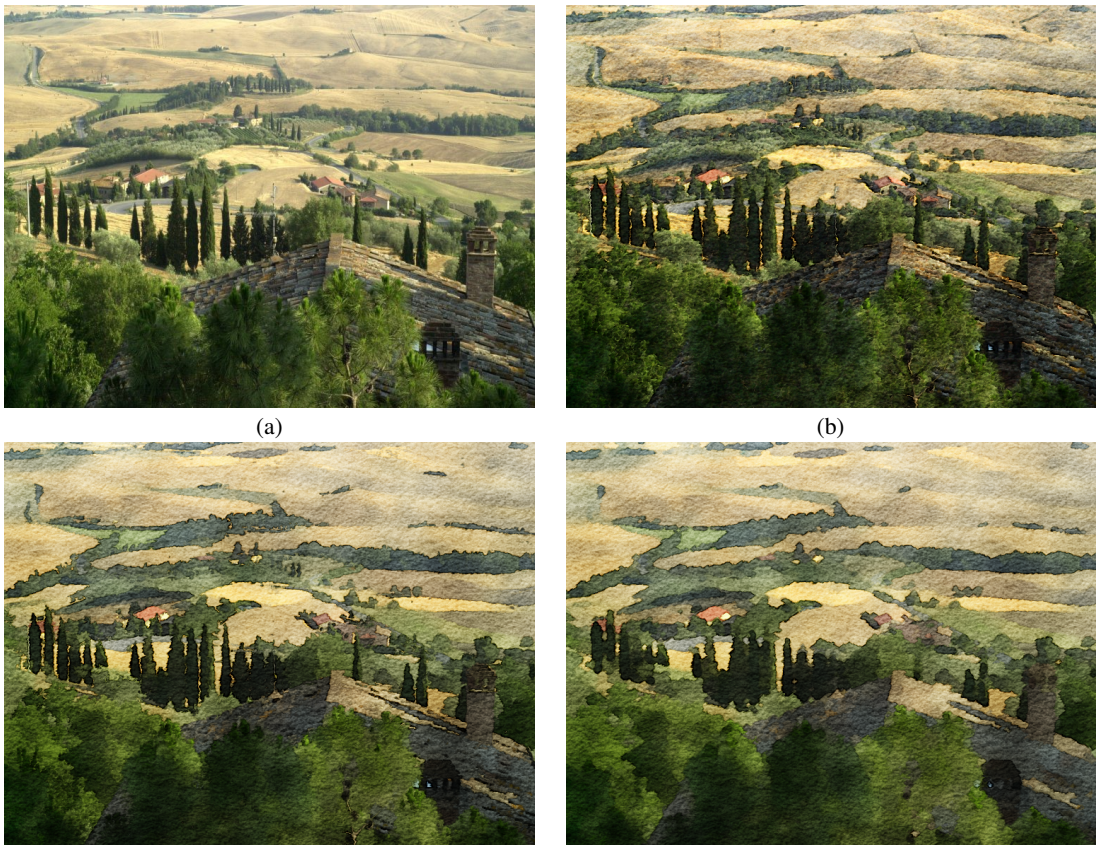


Figure 7: Normal smoothing: by averaging the normals the user simplifies the shading progressively to the desired level.

### 4.2.2 Abstraction of the color of an image

When dealing with a photograph, we no longer have uniform color regions as given by the toon shader. To obtain such regions, we



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

Figure 8: Image abstraction: (a) original image, (b) without abstraction: direct application of watercolor effects, the image is too detailed, (c) result after segmentation, (d) final result with segmentation and morphological smoothing.

propose a color segmentation step. We use a mean shift algorithm [Comaniciu and Meer 1999] to perform such a segmentation. This gives a less detailed image as presented Figure 8-(c) showing the uniform color regions typical in watercolor.

#### 4.2.3 Morphological Smoothing

After the previous step, either segmentation or illumination smoothing, some small color regions remain. We allow the user to apply a morphological smoothing filter (sequence of one opening and one closing) to reduce color areas details. The size of the morphological kernel defines the abstraction level of color areas and silhouettes (Figure 8-(d)). It can be viewed as the brush size. The same kind of 2d approach is used by [Luft and Deussen 2005], except that they use a Gaussian filter and thresholds to abstract their color layers.

## 5 Temporal coherence

In the static version of our watercolor pipeline, we have shown that a convincing watercolor simulation may be created by using texture maps that represent both low frequency turbulent flow and high frequency pigment dispersion. Using this method in animation leads to the "shower door" effect, where the objects slide over the pigments textures. Ideally, we would like the pigment effects to move coherently with each object, rather than existing independently.

While it is tempting to simply map pigments textures onto each object (as in [Lum and Ma 2001] for example), this leads to problems with ensuring the scale and distribution of noise features. For example, the features of a pigment dispersion texture that appears correct from a specific camera position may blur if the camera is zoomed out. Therefore, we seek a compromise between the 2d location of pigments and the 3d movements of objects.

This question has already been addressed in the case of the canvas, leading to two approaches of a dynamic canvas [Cunzi et al. 2003; Kaplan and Cohen 2005]. The problem there was to match the canvas motion to the camera motion. In the watercolor case, we want the pigment textures to follow each objects motion, not just the camera. We thus propose two different methods that each extend one of the previous dynamic canvas approaches.

As far as the abstraction steps are concerned, the toon shader and normal smoothing are intrinsically coherent since they are computed in object space. On the other hand, the morphological filter is not coherent from frame to frame, since it is computed in image space. Solving this question would probably need to follow color regions along the animation as in "video tooning" [Wang et al. 2004]. Such a method would imply the loss of interactivity and thus we have preferred to concentrate here on the watercolor effects part of the pipeline and leave the incoherency of the morphological smoothing for future work.

### 5.1 Attaching multiple pigment textures in object space

Our first method takes inspiration both from Meier's work on painterly rendering [Meier 1996] and Cunzi et al.'s work on a dynamic canvas [Cunzi et al. 2003].

(a) (b) (c)

Figure 9: (a,b) Our particle method illustrated with a chessboard texture. (a) Rotation: 2d textures follow 3d particles (black dots), (b) z-translation: An infinite zoom of the texture maintains a constant frequency. (c) Result when using our pigment and paper textures.

We decompose the possible movements of the object between translation along the $z-$axis and the other transformations. The idea is that, when an object is moving towards (or away from) the camera, we have to maintain a constant frequency of pigments while the size of the object projection increases or decreases in image space. For the other motions the pigments have to follow the object without being distorted by the perspective projection.

Like Meier, we use particles (points on the 3d mesh) to attach our drawing primitives to the object. Our primitives are the pigment density textures. They are much larger than brush strokes, therefore we need far fewer particles.

The particles are distributed on the mesh by projecting onto the mesh regularly sampled points of the object bounding box. In practice, for a simple object that stays totally in the camera field of view, we use 6 particles (the middles of the 6 faces of the bounding box), shown by the black dots on the right hand side. Pigment textures are then drawn in image space centered on each particle projection, and blended while rendering the mesh. The blending amount of each texture is defined for a vertex according to the 3d distance between the vertex and each particle. The closer a vertex is to a particle, the more the corresponding texture will be taken into account (see the blending in false colors). Such a blending avoids the popping effect that occurs in painterly renderings when strokes appear due to visibility. Indeed, a pigment texture attached to a non visible particle can continue to influence the final rendering, allowing for a continuous evolution. The result is that pigments are drawn in 2d space but move according to the 3d object motion (fig.9-(a)).

This does not define how to scale the texture according to a $z-$translation. As pigments are defined in screen space, we would like to keep their frequency constant. But if we directly keep the texture scale unchanged, the object will seem to slide on the texture during a zoom. To avoid this effect, an infinite zoom in the texture is used, as described in [Cunzi et al. 2003][1]. Such an infinite zoom is produced by cycling continuously between successive octaves (that is successive scales) of the pigment textures. Here we use four octaves. The observer seems to zoom into the pigment textures, while the frequency of the pigments stays constant (fig.9-(b)).

## 5.2 3D Animation of 2D Flow Effects

As pointed out in [Kaplan and Cohen 2005], no transformation on a 2d texture map can exactly match motion fields produced by arbitrary movements in a 3d environment. An alternative solution is to move the constituent elements of the 2d map in 3d and then to reproject those elements back to 2d to create a new texture map. We associate noise features with 3d particles on the surface of the objects. At runtime, we project those particles into screen space and draw features into a 2d texture, reconstructing new noise and turbulence textures for every frame (see Figure 10).

[1]The source code is available at
`artis.imag.fr/Members/Joelle.Thollot/dynamic_canvas/`

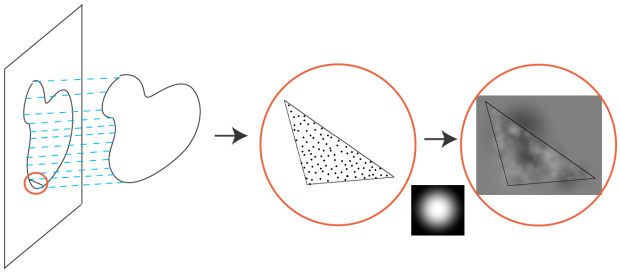


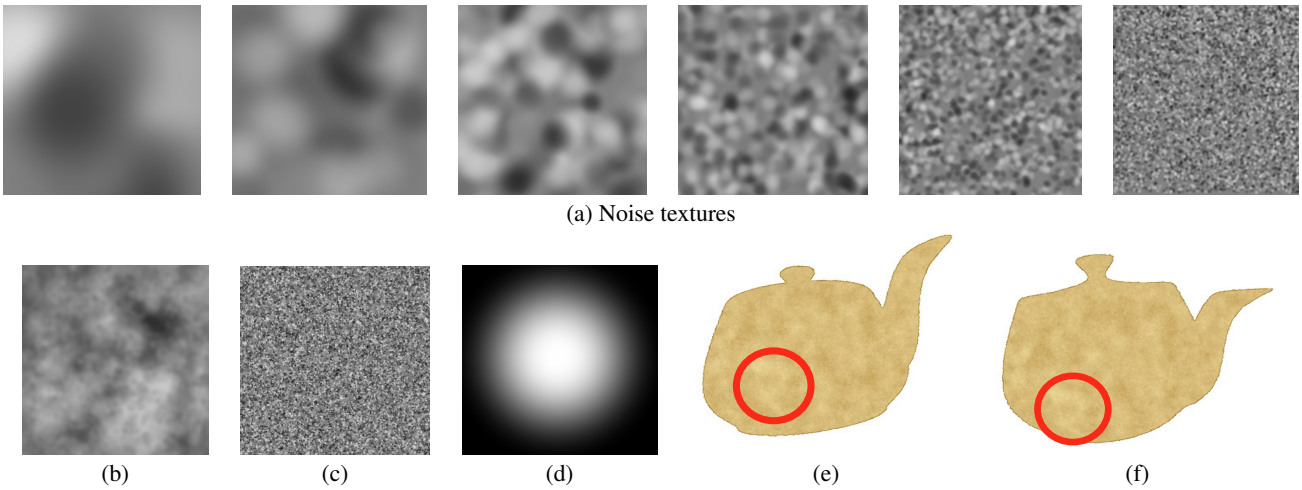
Figure 10: An overview of the interactive noise generation system. First, the models vertices are projected to screen space. Next, the system determines the amount of screen space occupied by each visible triangle. This determines the number of randomly positioned features to draw to create a noise texture of the appropriate frequency. Randomly colored, alpha blended quadrilaterals of that frequency are drawn at feature locations, creating the texture. The alpha texture is shown inset.

Turbulent flow can be represented as a harmonic summation of noise textures [Perlin 1985]. Noise is characterized by randomly valued (a color value typically) changes in a field of arbitrary dimension where the changes in value occur with a constant spatial separation, referred to as frequency $f$. *Features* are the locations at which changes in the field occur. Therefore, following the Perlin noise, we desire a continuously generateable 2d noise texture with the following properties, 1) features occur at a specific frequency, 2) feature values are pseudo-random, 3) frequency range is band-limited, 4) features move with the surfaces. Note that properties 1 and 4 are contradictory and may not be simultaneously achievable for arbitrary transformations.

An approximation of the 2d Perlin noise may be obtained by interpolating between random grayscale values located at grid points, representing features, placed at distance $f$ apart, where $f$ is the desired frequency of the noise [Ebert 1999]. Though we cannot use a grid, since our feature points move in 3d, this is the basic approach we adopt.

A noise feature is represented in our program with a structure consisting of random barycentric coordinates, $\beta = b_0, b_1, b_2$ and a random grayscale color, $c$. A feature of frequency $f$ may be drawn using a quadrilateral in screen space centered at $\beta$ with width and height $f$, using an alpha texture defined by the function $6t^5 - 15t^4 + 10t^3$ [Perlin 2002] (shown in Figure 11d) and color $c$. The alpha texture is used to emulate interpolation between adjacent features.

A noise texture $N_f$ is computed as follows. First, for every visible triangle (surface face), $t_i$, we compute the amount of image space, $\lambda_i$ it occupies. Then, we draw $\lambda_i/(f*f)$ features for every $t_i$. The location of each feature is calculated by using each $t_i$'s vertices (projected into screen space) as control points for the features



(a) Noise textures

(b) (c) (d) (e) (f)

Figure 11: (a) Progressive levels of noise generated interactively with increasing frequencies. (b) Compositing noise levels from (a) yields a turbulence texture or (c) a pigment texture - using only high frequency noise textures. (d) The alpha texture used to draw the features, (e,f) Notice how the pigments and turbulence features follow the surface. No paper texture is used for these images.

barycentric coordinates. This creates a noise texture with average feature distribution of frequency $f$.

A turbulence texture may be created by summing a series of such noise textures - a typical series of frequencies might be, for base frequency $F_0 = f, F_1 = f/2, F_2 = f/4 \cdots F_K = f/2_K$ - by blending them with the contribution of each $N_j$ being defined by the harmonic function $1/f$. Examples are shown in Figure 11.

### 5.2.1 Minimizing Artifacts

This method yields an average frequency distribution of $f$ rather than an exact distribution. Two problems may occur because of this random placement. First, we are not guaranteed to completely cover the image space. Attempting to order the spacing of the particle placement may not be useful since we cannot predict what type of perspective deformation the triangle will undergo. Therefore, the screen color is initially cleared to a flat middle gray; features then color the screen as an offset from a strict average rather than defining the actual screen color themselves as in standard noise. This may overly weight the screen to gray, but if our random distribution of feature locations is good (see [Turk 1990] for a discussion), then this problem will be minimal and in practice, this seems to be true. Second, we may create hotspots of feature placement when features overlap. Again, a good initial distribution of features will minimize this problem, but in practice, the blending function is smooth enough that this issue is not apparent.

The amount of screen space occupied by each surface face $\lambda_i$ changes with camera motion. Because the number of features drawn is dependent on this value, new features may come into and out of existence very quickly which will cause popping artifacts if not handled with care. We minimize this artifact by taking advantage of the fact that our turbulence texture is the sum of a series of noise textures. Continuity is maintained by moving features between each noise texture $N_j$ and textures $N_{j+1}$ and $N_{j-1}$. When $\lambda_i$ grows larger, features are moved from $N_{j-1}$ to $N_j$ proportional to the amount of screen space gained. Conversely, when $\lambda_i$ shrinks, features are moved from $N_{j+1}$ to $N_j$ and from $N_j$ to $N_{j-1}$. Because features are transitioned between noise textures at different frequencies an ordering between noise texture levels is created. Therefore, the turbulence appears consistent at different resolutions, i.e. if an object is zoomed away from the viewer, the low frequency noise is transitioned to high frequency noise so the visual characteristics of the turbulence appears consistent. In order to maintain the frequency distribution, one feature is moved between each noise texture levels at a time, linearly scaling the width between levels. With this method, popping artifacts occur only at the highest and lowest frequencies. At high frequencies, the artifacts were very tiny and so, not very noticeable. Low frequency errors can be avoided by making the lowest frequency larger than the screen.

As a final consideration, the surface meshes in the scene may be composed of triangles that occupy screen space of less than $f * f$ pixels, in which case *no* face will draw any features of the desired frequency! We create a hierarchy of faces, grouping faces into superfaces as an octree. We then analyze the screen space occupied by the superfaces in order to determine the number of features to draw at low frequencies. Only normal similarity is considered when creating our hierarchies, and though better schemes may exist for grouping faces, this method works well in practice.

## 6 Results and Discussion

We show some results obtained by our method using as input either a photograph or a 3d model Figure 12. More watercolor results and some videos can be found in `artis.imag.fr/Membres/Joelle.Thollot/Watercolor/`.
As most of the work is done by the GPU, the static pipeline runs at a speed similar to a Gouraud shading (between 25 frames per second for 25k triangles downto 3 frames per second for 160k triangles) for a $750 \times 570$ viewport on a Pentium IV 3GHz with a GeForce FX 6800.

Evaluating the visual quality of watercolor renderings is not an easy task. Figure 13 shows a comparison we have made with Curtis original method [Curtis et al. 1997] that was a full physical simulation. Our method allows us to produce images that stay closer to the original photo while allowing the user to vary the obtained style. On the other hand Curtis' result benefits from his simulation by showing interesting dispersion effects. However, it is precisely these dispersion effects that make that method unsuitable for animation. In light of this, it is unclear whether we lose anything by not doing a full fluid flow simulation in an animated or interactive environment.

The temporal coherence effects of our two methods yield distinct results with several tradeoffs. The texture attachment method yields good results for a single object viewed with a static background.

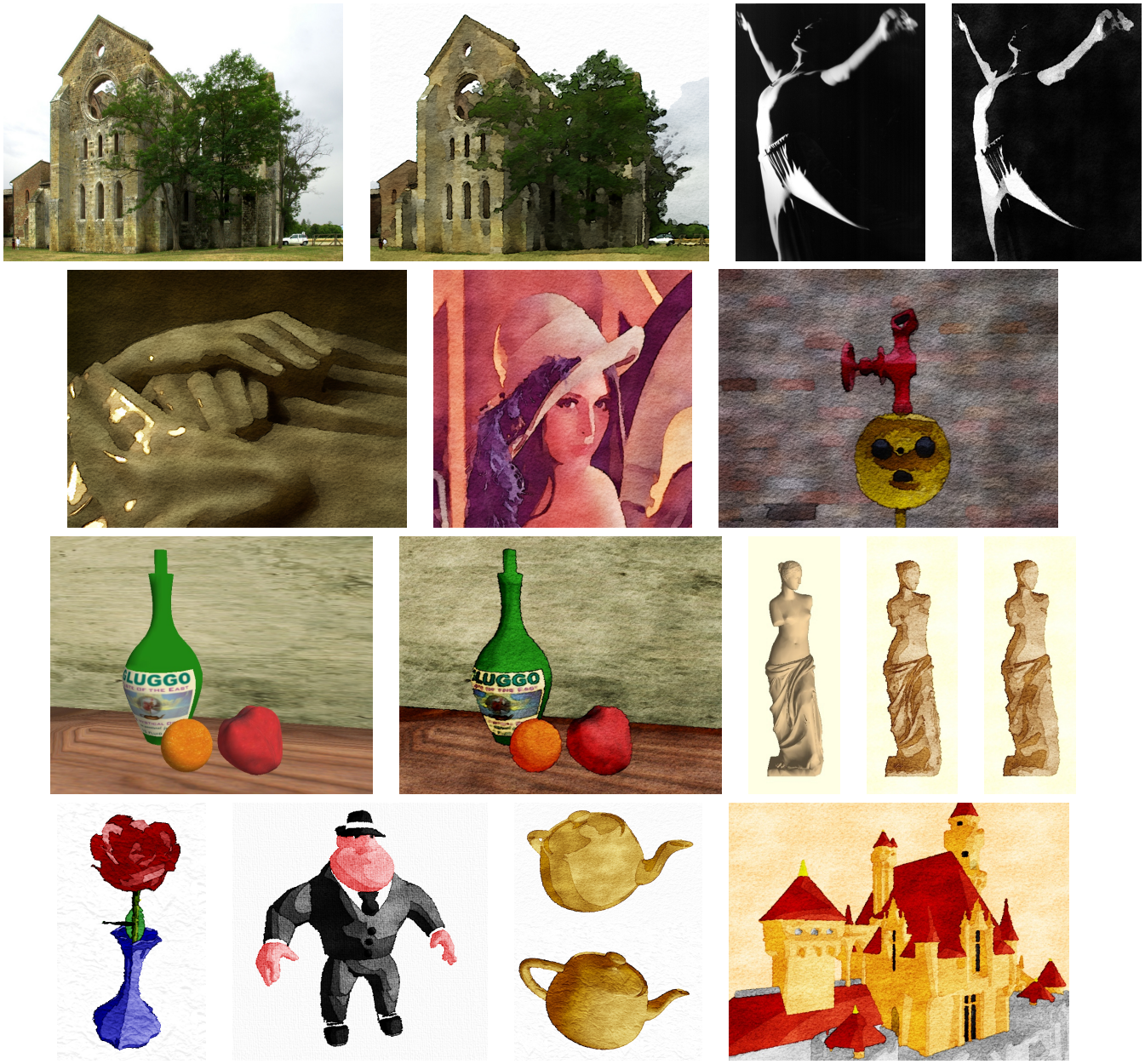


Figure 12: Watercolor-like results. The first line shows the original photograph and the resulting watercolor-like image. The second line shows other filtered photographs. The third line shows a gouraud shaded 3d scene and the resulting watercolor rendering with two versions of the Venus model without and with morphological and normal smoothing. The fourth line shows more watercolor-like 3d models.

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

Figure 13: Comparison with Curtis result: (a) original image, (b) Curtis watercolor result, (c,d) our results.

The number of particles used to render the animation has to be a compromise between two effects: Too many particles would tend to blur the high frequencies due to texture blending whereas not enough particles creates scrolling effects due to independent movements of each patch of texture. Some distortion occurs at pixels far from attachment points, but in practice, using at least 6 attachment points minimized this effect. Ideally the number of particles should be adapted to the viewpoint. For example after a zoom, the particles may become too far from each other and some new particles should be added.

Calculating flow textures interactively matched the movement of the turbulent flow and pigment dispersion exactly with the motion of the objects in the scene, making it useful for complex animations in immersive environments. Unfortunately, this method is much more expensive to compute since between 1-10 noise textures and 1-2 turbulence textures must be interactively constructed for every frame, yielding as few as 1 frame a second for 60k triangles. Finally, small scale popping artifacts may be visible yet we found this was not very noticeable for complex turbulence textures possibly due to the low alpha value assigned to such high frequency features. We allowed the user to control several variables (such as which frequencies to use) in order to construct the pigment and turbulence textures. This yielded a wide range of visual styles yet those styles did not always exactly correspond with those produced by the static method. Reasonable fidelity to the static method was achievable as long as the frequencies we used corresponded well with the static pigment repartition texture.

## 7 Conclusions

We have presented a watercolor rendering technique that is fully controllable by the user, allowing the production of either coherent animations or images starting from a 3d model or a photograph. Our framework is interactive and intuitive, recreating the abstract quality and visual effects of real watercolors.

Each step of the pipeline can still be improved, especially by offering the user a choice between slower but better methods for each effect. This can be suitable for producing movies when high quality is most important. Ideally our idea would be to keep our pipeline as a WYSIWYG interface for preparing an offline full computation of more precise effects.

As mentioned in the paper, there are several issues that we want to address in the future. The diffusion effect would be interesting to recreate. We can think of using work like [Chu and Tai 2005] to perform diffusion computation but it opens the question of how to control such an effect when dealing with an already given scene. To stay in our philosophy we have to think of simple and intuitive ways of deciding which part of the image must be concerned with the diffusion.

A lot of questions remain concerning the temporal coherence. The methods we propose do not target the morphological smoothing step or any image processing. Taking inspiration from "video tooning" [Wang et al. 2004] we would like to address this problem in our pipeline. Another possibility would be to use image filters like in [Luft and Deussen 2005] to decrease the popping effects without adding too much computation.

## Acknowledgments

This research was funded in part by the INRIA Action de Recherche Cooprative MIRO (`www.labri.fr/perso/granier/MIRO/`).

Thanks to Laurence Boissieux for the 3d models Fig 1-(a,b), 4 (©Laurence Boissieux INRIA 2005). Thanks to Gilles Debunne for the reviewing and the web site.

## References

BURGESS, J., WYVILL, G., AND KING, S. A. 2005. A system for real-time watercolour rendering. In *CGI: Computer Graphics International*, 234–240.

CHU, N. S.-H., AND TAI, C.-L. 2005. Moxi: real-time ink dispersion in absorbent paper. In *Siggraph 05*, ACM Press, 504–511.

COMANICIU, D., AND MEER, P. 1999. Mean shift analysis and applications. In *ICCV (2)*, 1197–1203.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Graphics Interface*, A K Peters, LTD., 121–129.

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

EBERT, D. 1999. Simulating nature: From theory to application. In *Siggraph Course*. ACM Press.

JOHAN, H., HASHIMOTA, R., AND NISHITA, T. 2005. Creating watercolor style images taking into account painting techniques. *Journal of Artsci*, 207–215.

KAPLAN, M., AND COHEN, E. 2005. A generative model for dynamic canvas motion. In *Computational Aesthetics*, 49–56.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR: International symposium on Non-photorealistic animation and rendering*, ACM Press, 13–20.

LEI, E., AND CHANG, C.-F. 2004. Real-time rendering of watercolor effects for virtual environments. In *PCM: Pacific Rim Conference on Multimedia*, 474–481.

LUFT, T., AND DEUSSEN, O. 2005. Interactive watercolor animations. In *PG: Pacific Conference on Computer Graphics and Applications*, 7–9.

LUM, E. B., AND MA, K.-L. 2001. Non-photorealistic rendering using watercolor inspired textures and illumination. In *PG: Pacific Conference on Computer Graphics and Applications*, 322–331.

MEIER, B. J. 1996. Painterly rendering for animation. In *Siggraph 96*, ACM Press, 477–484.

PERLIN, K. 1985. An image synthesizer. In *Siggraph 85*, ACM Press, 287–296.

PERLIN, K. 2002. Improving noise. In *Siggraph 02*, ACM Press, 681–682.

SMALL, D. 1991. Modeling watercolor by simulating diffusion, pigment, and paper fibers. In *SPIE*, vol. 1460, 140–146.

TURK, G. 1990. Generating random points in triangles. In *Graphics Gems I*, A. Glassner, Ed. Academic Press.

VAN LAERHOVEN, T., LIESENBORGS, J., AND VAN REETH, F. 2004. Real-time watercolor painting on a distributed paper model. In *CGI: Computer Graphics International*, 640–643.

VAN ROOJEN, J. 2005. *Watercolor Patterns*. Pepin Press / Agle Rabbit.

WANG, J., XU, Y., SHUM, H.-Y., AND COHEN, M. F. 2004. Video tooning. In *Siggraph 04*, ACM Press, 574–583.