A Simulated Watercolor Painting with Smooth Ink Flow

Michael Fotheringham
JixiPix Software, mikef@jixipix.com

Kristal Fotheringham
JixiPix Software, kristal@jixipix.com

***Abstract* – We are proposing a new algorithm based on a previous work at JixiPix Software, US patent numbered US10,008,011 titled "Methods for creating a simulated watercolor-painted image from a source image", [7] Effective Wet-in-Wet Flow Synthesis for Ink Diffusion, and [6] Expressive Rendering with Watercolor.**

Proposed Implementation

We are proposing a new algorithm based off the original *Balestrieri et al.*[1] while incorporating new techniques from [5][6][7]. The original algorithm processed most steps based on the created "edge buffer" with added noise. This added noise acted as a wobbling effect [5] for the abstracted areas as well as a bleed when smoothing the image. Secondary was the highlighted edge delineation map used to create the 'white areas' that artists sometimes leave in between colors . These two areas were the main difference between [6] Expressive Rendering with Watercolor and [7] Effective wet-in-wet flow synthesis for Ink Diffusion however the original authors saw the regions becoming separated as a negative whereas [1] decided to accentuate it. [7] took the implementation one step further and implemented a wet-in-wet flow synthesis based on ink diffusion rather than a 'random jitter'.

Image Preparation

Saturate original image. Watercolor paintings are usually very colorful and doing a saturate process gets us closer to a realistic painting. This implementation is as simple as dot(rgb, vec3(.2125, .7154, .0721)). [3] OpenGL Shading Language.

Texture Creation

Create a 'diffuse noise' image using techniques as described in [5]4.1.2, a 'pigment texture' based on high-frequency noise and combine with turbulent texture, and a 'paper grain' which can be a scanned image of a real watercolor paper.

Image Abstraction

Abstract the image using a graph cut image segmentation [4] or similar abstraction process. The goal is just to abstract the image and there are numerous techniques to accomplish this goal. Combine the small regions and use a means average of the regions to find average color inside each of the regions. Figure 1.

Find the most prominent colors in the original image based on pre-defined number of shades and group regions in the segmented image into similar colors. With a Watercolor painting there is a limited number of colors that an artist will use to paint with. After completion of this step there should be the same number of sets of regions as shades, i.e. there is a color associated with each set of regions. The regions do not necessarily need to be joined. The output of this stage is nothing more than an abstracted image with a list of colors used on the abstracted image. This list of colors will be used during the edge and rendering phase. Sort this color list from light to dark using a perceived value means (i.e. blue can be perceived as being as dark as black and red and not necessarily light to dark, in fact the perceived output looks closer to a real world watercolor painting when done this way). This version varies from [1] in that it describes a light to dark rendering process whereas we put emphasis on perceived color and can adjust our conversion on the fly. By default the equation is simply $P_c = 1 - (r*.241 + g*.691 + b*.068)$. Values with a darker value (higher P_c) will be rendered last and have less emphasis in the final painting.

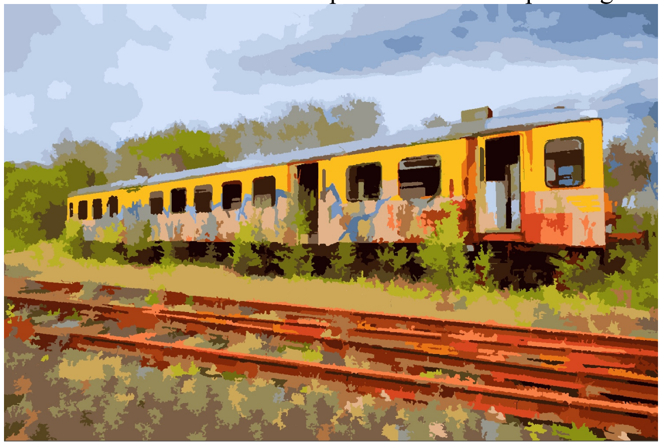


Figure 1 – Abstracted Image

July 3rd, 2018

OUTLINE

Next create an outline of this abstracted image using any of the technologies to find an edge, sobel, difference of gaussian, adaptive threshold, etc. In our example we use a difference of gaussian. Since we have already performed our abstraction we can use the abstraction layer as our basis for the outline which will not get small details typically associated with edge finding code. This new 'edge' layer will be used for a few processes including edge delineation and hard edges. The 'edge' layer can be a 1 channel image. Make a copy of the 'edge' layer and label it 'outline' layer for use later. This 'outline' layer will be used to make the 'edge darkening' at the end of the render phase.

Wobble the. 'edge' layer created above using techniques similar to [5] and [6]. This step does require a paper texture to be used as the wobble structure, however we have a much simpler approach in that you can perform a simple diffuse of the edge then apply a morphological filter [8]. Future reference to this technique will be just called 'wobble'. There are many approaches that can be taken to wobble the image, diffuse/morphological is just one example. The outcome is a roughened edge that is similar to paper fibers in a typical watercolor paintings. Figure 2

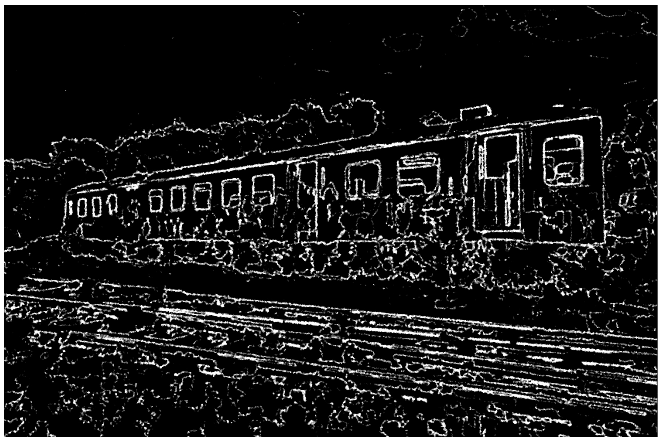


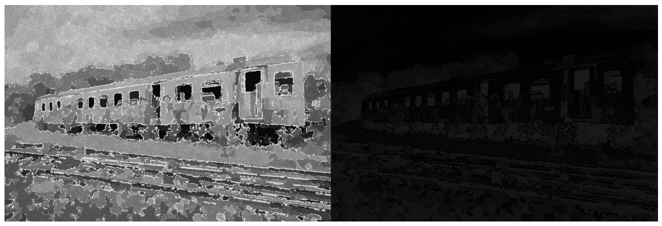
Figure 2 – outline generated from abstracted image

CONTROL IMAGE CREATION

The 'region' adjusting stage now begins as well as the creation of the 'control' and 'mark' image used later during the rendering phase. Go through each abstracted region and 'wobble' said region. Wherever the 'wobbled' region is active and where the 'edge' layer is NOT active you will now put a number corresponding to the color list for the current region into the 'mark' image and the P_c into the 'control' image. Repeat this for all regions.

Add the diffuse noise image to the 'control' image. This will be used in the propagation render phase to 'bleed' the regions together. Figure 3. During the rendering phase we are using a form of a bilateral filter that will not cross edge boundaries. By adding in a little noise to this control image we are giving the bilateral filter the ability to go outside of those strict edge boundaries.

You will notice in figure 3 that the mark image is very dark. This is because this image stores the index to the region in the color list that is 0 based, simply a way to optimize the rendering phase.


(a) Control Image (b) Mark Image
Figure 3

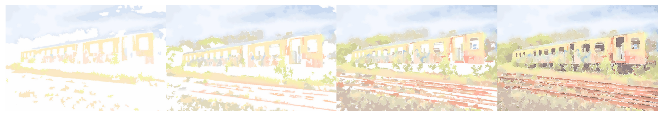
RENDER PHASE

Create a 'paint' image that will be used to store the rendering and initialize it to white. For each region in the 'mark' image render into the 'paint' image using a darken blend mode while using an opacity on a sliding scale based on the P_c with the most prominent colors getting a max of 50% transparency and lighter P_c getting a max of 80% transparency. The transparency gives a perception of transparent ink when applied over top of the previous rendered layers. After each compositing mode run a simple cross bilateral filter[10][11] on the 'paint' image with the 'control' image used for the contours or range function and the sigma range varying based on the P_c with sigma range growing for lighter colors which will give the perception of smoothing/growing ink on lighter colors.

$$W_p = \sum_{q \in S} \mathrm{G}_{\sigma s}\left(\left\|p-q\right\|\right) G_{\sigma r}\left(F_p - F_q\right)$$

Number of passes and/or distance of spatial kernel will give the perception of more or less 'liquid' being applied to the 'paint' image. However it is important to note that in a regular bilateral implementation there is a point of diminishing returns and more 'cartoon' like side-effects associated with more passes rather than a larger spatial kernel parameter. Figure 4

After completion of all rendering phases the 'edge darkening' [6] is applied. For each color in the color list render the region into the 'paint' image using a 'darken' blend mode only where the 'control' image and 'outline' image are active. This will render in the outline but only where the 'wobbled' 'control' region is at which gives the perception of un-even darkened edges. Figure 4



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

Figure 4: Rendering pipeline. (a) rendering 10 colors complete. (b) rendering 18 colors complete. (c) rendering 24 colors complete. (d) edge darkening added to final image.

Texture Composition

Composite 'paint' buffer with the 'pigment texture' using a Burn Color Mode. Composite the 'paint' image with the 'paper grain' using Overlay Color Mode. Figure 5(a-d)

FUTURE IDEAS

The ideas behind this proposal adds onto previous work [1][5][6][7] while still retaining most of the previous work benefits and issues. Issues with [1] is mainly that there is limited control of what areas become smooth however the fluid nature of the algorithm is very pleasing. Issues with [5][6][7] is the lack of fluid smoothing and blending with [7] becoming the closest. There is lots of room for future improvement while keeping the nearly exact algorithm intact.

We could provide color correction/augmentation before or after the algorithm. Changing color by using a LUT, overlay of real painting colors, contrast etc. would also change the location of the abstraction and ultimately the regions and how they are rendered which would ultimately cause the smoothing effect to happen sooner in the rendering phase on a given area. Adding LUT, overlay, contrast, etc. afterwards to leave those color changes more intact and that a LUT change would be more gradual and smoother than pre-abstraction.

Along these same lines you could convert the pre-abstraction to duo-tone, tri-tones, monotones, etc. Again before or after pre-abstraction would drastically change the way the painting processes.

Other ideas include varying the perceived calculation and base it on real world instances. This could include rendering a scene based on a Chinese ink paintings, rendering a sky and giving more importance to light or blue colors.

Other possibilities include not combining similar colored regions and use a salient technique to define what regions should be rendered first. This would require adjusting the 'perceived' system for rendering order but not necessarily transparency by having 2 different functions (although it could easily be optimized to handle both). Salient techniques are quite common and picking the right option based on usage. One of those is described by *Min-Ming Cheng, et.al* [9].

One could also remove the 'diffuse' noise image from the 'control' image and use it during the rendering phase. This affords numerous advantages. One of which is the ability to offset your 'diffuse' texture for every pass or every color during the bilateral phase. This will give more of a perception of ink flow, especially the further out the liquid progresses. This would also mitigate the issues with multiple passes on a bilateral filter 'cartooning' the image.

Varying where the 'diffuse' noise is added into the rendering pipeline also varies the overall look of the bleed and wetness in the image. Adding noise to the original image before abstraction gives more disjointed areas, while adding noise to the regions will give more of a rougher painting edges.

Adjust the P_c and transparency during the render phase to take into account how many regions and unique colors there are in the image. This will keep the 'wetness' of the painting nearly identical across more shades or more complex regions.

REFERENCES

[1] Methods for creating a simulated watercolor-painted image from a source image. US 10,008, 011 – John Balestrieri – 11/25/2015

[2] Provisional Patent Application 62,084,790 – John Balestrieri – 11/26/2014

[3] OpenGL Shading Language 3rd Edition – Randi J. Rost, Bill Licea-Kane

[4] Efficient Graph-Based Image Segmentation – Pedro F. Felzenszwalb, Daniel P. Huttenlocher

[5] Interactive Watercolor rendering with temporal coherence and abstraction – Adrien Bousseau, Matt Kaplan, Joell Thollot, Francois X. Sillion – 2006

[6] Expressive Rendering with Watercolor – Patrick J. Doran, John Hughes – 2010

[7] Effective Wet-in-Wet Flow Synthesis for Ink Diffusion – Ren-Jie Want, Chung-Ming Wang – 2010

[8] www.mif.vu.it/atpazinimas/dip/FIP/fip-Morpholo.html

[9] Global Contrast based Salient Region Detection – Ming-Ming Cheng, Guo-Xin Zhang, Niloy J. Mitra, Xiaolei Huang, Shi-Min Hu – 2011

[10] Digital Photography with Flash and No-Flash Image Pairs – Georg Petschnigg, Maneesh Agrawala, Hugues Hoppe - 2004

[11] Flash Photography Enhancement via Intrinsic Relighting – Elmar Eisemann, Fredo Durand - 2004

AUTHOR INFORMATION

Michael Fotheringham is currently an effect engineer for JixiPix Software. Michael has a long history of doing special effect work dating back to 1987 and has worked for many pioneering effect and 3D software companies.

Kristal Fotheringham is currently an artist for JixiPix Software. Kristal has a long history in the art field and has been artineering work in many fields from advertising agencies to fine arts, to interior design, to special effects. Kristal is not directly an effect engineer but works closely with the engineering team when designing and implementing algorithms and processes.