# Expressive Rendering with Watercolor

Patrick J. Doran[*] Brown University

John Hughes[†] Brown University

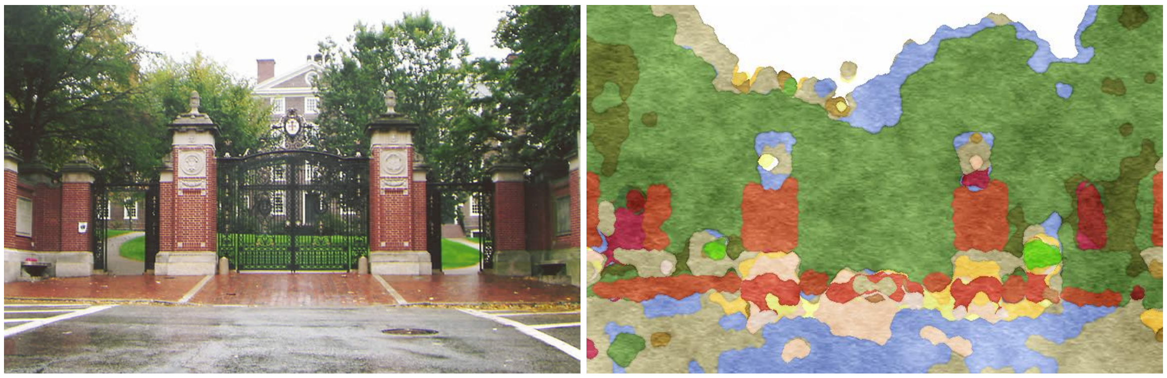


**Figure 1:** *Source and result of watercolor rendering*

## Abstract

We propose comprehensible rendering of Google Streetview Images using watercolor textures and line drawings. We discuss our experiments and results of watercolor rendering.

**Keywords:** watercolor, texture synthesis, non-photorealistic

## 1 Introduction

This work is part of a larger project, whose goal is to replace Google Streetview images with non-photorealistic renderings because it can be visually pleasing, reduce bandwidth requirements, and be more informative than the original photos. We do so using watercolor textures and line drawing of important features. This work describes our experiments with watercolor rendering.

In particular, we aim to render trees in watercolor texture like Michael Gage's watercolors, see Figure 2. Another student has worked on indentifying vegetation.

We took two approaches: texture synthesis and watercolor processing. We first tried to transfer an existing watercolor texture onto the photo. We alter the monchrome texture to be in the target's colorspace and then try existing texture tranfer algorithms. Next we tried applying image segmentation and filters to fake watercolor effects. In the following we discuss our experiments, report on the results and on future designs.

[*]e-mail: pdoran@cs.brown.com
[†]e-mail: jfh@cs.brown.edu

### 1.1 Related Work

There has been a lot of work in collecting images of cities at the street level for navigation, most notably Google's Street View 360° panoramas. These sort of images are helpful in determining what your destination will look like at eye level. This information has moved to mobile devices, but can use significant bandwidth.

Similar work to our own has been done for Large City Models [Quillet et al. 2006]. They focus on rendering edges and reconstructing the images as a 3d model of the city. We do not seek to reconstruct 3d models. We want only to process existing data in such a way that it can reduce bandwidth and still be as informative as the original photo.

## 2 Approaches

### 2.1 Computer Generated Watercolor

Simulating watercolor effects certainly would produce the desired effect. However, a proper simulation is too complex and slow for our purposes. We do not seek to be physically accurate like [Curtis et al. 1997]; we want near real-time processing of images.

### 2.2 Texture Synthesis

Texture synthesis appeared to be a reasonable approach to the problem. The idea is that a watercolor painting already has the texture we want, so we need only copy it.

The original intent of this work was to transform a monochrome region, specifically green representing trees, into a watercolor texture. The source watercolor texture may not have the same color as the the target region, e.g. we may want to use a blue watercolor texture and a green region as a target. This can be remedied by transforming the watercolor texture to the target's colorspace. In an example pipeline, an artist provides a monochrome watercolor painting with a light-to-dark variation. The algorithm performs PCA on the texture and target image in RGB colorspace. Change of basis is performed mapping the texture into the target image's colorspace. This only works well if both the target region and watercolor texture are

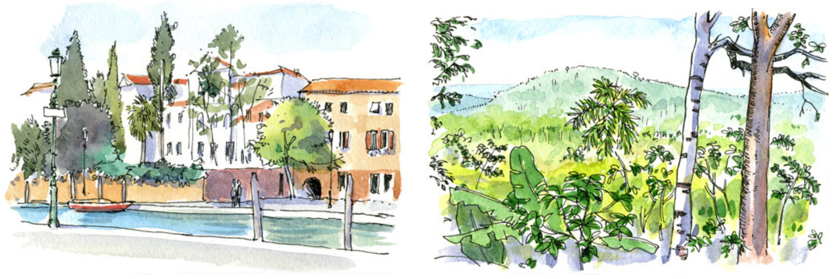
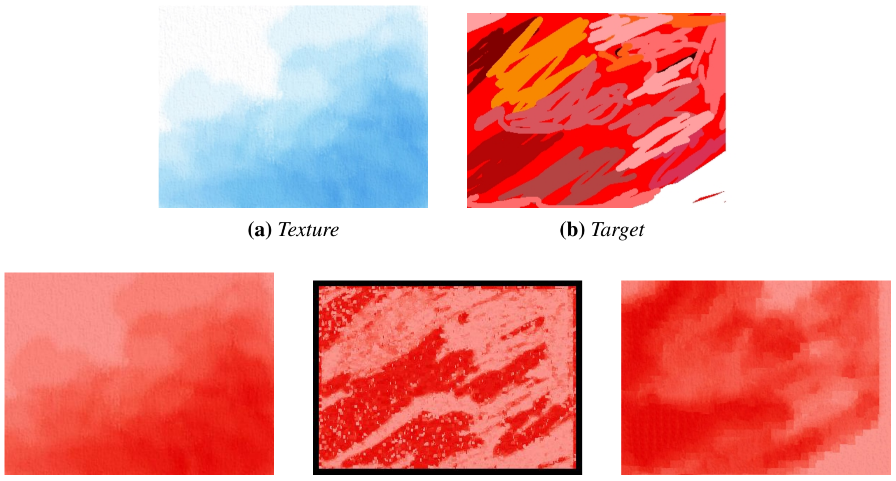

**(a)** *Piazza Roma* **(b)** *Las Terrazas*

**Figure 2:** *Goal paintings by Michael Gage*


**(a)** *Texture* **(b)** *Target*

**(c)** *New Texture* **(d)** *Ashikhmin Synthesis* **(e)** *Image Quilting*

**Figure 3:** *Texture Synthesis Approach*

monochrome. Once the texture has the correct color, texture synthesis (guided by an input image) can be performed.

### 2.2.1 Ashikmin's Texture Synthesis

Synthesizing Natural Textures [Ashikhmin 2001] was our first approach to render watercolor because it was fast and easy to implement. The algorithm works by initializing the output to random noise and then for each pixel in scanline order search the L-shaped neighborhood of the current pixel in the output image for a candidate whose neighborhood in the texture is most like the current pixel's; then copy the corresponding texture pixel to the output image. This is a very straightfoward way to render a texture. This tends to copy strips of the original texture to the output image and using larger neighborhoods increases the size of the strips.

This works well for what its intended purpose: quasi-repeating patterns of irregularly shaped objects. This did not work well for watercolor. The watercolor texture is not a quasi-repeating pattern. The underlying paper texture may be so, but the color is not. The result tended to be more pointillist than watercolor.

### 2.2.2 Image Quilting

Image Quilting [Efros and Freeman 2001] is a slighlty more complicated approach to texture synthesis. This algorithm differs from the above by copying patches of texture instead of single pixel rendering. The algorithm works by choosing a patch from the texture that minimizes the difference between the underlying target image and the overlap region of the patches that came before it and then performing a minimum seam cut in the overlap region.

Instead of using dynamic programming to solve for the cut, we used a GraphCut approach. This effectively turned the algorithm into GraphCut textures [Kwatra et al. 2003]. This worked well to copy sections of watercolor into the output image, but the seams were far too noticeable. To minimize the appearance of seams, Poisson image blending [Pérez et al. 2003] was used during the composite of each patch. Unfortunately, Poisson blending tended to flatten the color of the output image to the point where the algorithm no longer appeared to be guided by the input image. Following these texture synthesis experiments, we abandoned this approach.

## 2.3 Bousseau's Watercolor Rendering

In Bousseau's Interactive Watercolor Rendering [Bousseau et al. 2006] watercolor images are created from photos by first abstracting the photo by some form of color segmentation, such as mean shift or other clustering algorithms. Then the colors are modified by a darkening formula based on an intensity image input that fakes a simulated effect such as turbulent flow or edge darkening. This approach works better for our purposes as it was created specifically for rendering watercolor. Also, it is much faster than the texture synthesis. For this reason, we chose to build off Bousseau's work.

## 3 Watercolor Processing

### 3.1 Bousseau's Work

Bousseau's work on interactive watercolor rendering was for more than just non-photorealistic rendering of images, but also for rendering 3d scenes. We are modifying existing photos, so we built our algorithm off his photo processing.

#### 3.1.1 Color Modification

Bousseau provides a formula for color darkening where $C$ and $C'$ are colors and $d$ is the pigment density parameter. The parameter $d$ is determined by image intensity $T \in [0, 1]$ and a global darkness scale $\beta$. The color is repeatedly modified by the darkening formula using input $T$ that fakes a type of watercolor effect (turbulent flow, edge darkening, etc.).

$$\begin{aligned} C' &= C - (C - C^2)(d - 1) && (1) \\ d &= 1 + \beta(T - 0.5) && (2) \end{aligned}$$

#### 3.1.2 Pigment Density Variation

A watercolor image appears darker where the pigment is denser. Paper, turbulent flow and pigment dispersion all affect the density of watercolor in the image. The greyscale image of a paper, Perlin noise, and sum of gaussians can be used as input the pigment density function to simulate such effects. We chose to remove the sum of gaussians as the effect it produced was not what we wanted.

### 3.2 Our Work

Bousseau's work was designed for running on a GPU. We have no such constraint for speed right now, we only want a reasonable way to process images into a watercolor. We have expanded upon Bousseau's work without realtime constraints.

#### 3.2.1 Image Segmentation

Bousseau's work calls for using mean shift clustering to segment the image. For our work we have used mean shift and a $K$-Means based clustering approach. Both are equally viable for segmenting the image. For mean-shift we use a radius that is 10% of the RGB colorspace. We use $K$-means clustering for targeting specific objects in an image to get a color scale for the object, e.g. a green scale for trees where $K$ is 4.

After clustering, small color regions remain. We apply morpholical smoothing to further abstract the photo. The size of the morphological smoothing kernel can be viewed as the brush size. We choose a smoothing size based on the size of the image with the intent to reduce most of the fine details in favor of large color regions.

#### 3.2.2 Edge Darkening

Bousseau's paper creates darker edges by modifying the pigment density function using the gradient intensity of the original image over a very small window. This leads to very limited edge darkening. We found that this window should be adjusted to reflect the size of an image. In a large image it would be difficult to notice the 1-pixel width edge-darkening region. For this reason, we use a variable size gradient filter applied vertically and horizontally.

$$H_3 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad (3)$$

$$H_5 = \begin{pmatrix} -1 & -2 & -3 & -2 & -1 \\ -2 & -3 & -4 & -3 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 4 & 3 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix} \quad (4)$$

$$H_n = \begin{pmatrix} -1 & \dots & \lceil -n/2 \rceil & \dots & -1 \\ \vdots & & & & \\ -\lfloor n/2 \rfloor & \dots & -(n-1) & ... & -\lfloor n/2 \rfloor \\ 0 & 0 & 0 & 0 & 0 \\ \lfloor n/2 \rfloor & \dots & (n-1) & ... & \lfloor n/2 \rfloor \\ \vdots & & & & \\ 1 & \dots & \lceil n/2 \rceil & \dots & 1 \end{pmatrix} \quad (5)$$

We choose $n$ based on the size of the feature we would like to represent. In this regard, it should be related to the size of the morpholocial smoothing kernel. We have found setting $N$ to be 60% of the smoothing kernel size produces acceptable results.

#### 3.2.3 Wobbling

One of the effects described in Bousseau's paper is the wobbling effect of pigment along edges due to paper granularity. Wobbling is applied by offsetting pixels by the gradient of the paper. Exactly how this is done is not described. Basing our idea on the paper's general description we found that the pixels were only ever offset a small amount. We improve upon this by normalizing the gradient at each pixel location and then scaling it, where the scale is equivalent to the number of pixels moved if the gradient is 1 at that pixel. This resulted in noisy, jittered edges that are far from the desired wobbled edges. We solved this by applying morphological smoothing after jittering the pixels. By keeping the scale small, say 3, and repeating this step multiple times, the desired wobble effect can be achieved.

#### 3.2.4 Gamma Correction

One of the problems we came across in Bousseau's original work is that all of the images tended to come out very dark. We solved this by applying gamma correction to the saturation and value channels of the final image. We found that a saturation gamma of 0.6 and a value gamma of 0.45 produce good results for most input images. Unfortunately, this introduces more parameters to control.

$$\begin{aligned} h &= h && (6) \\ s &= s^{0.6} && (7) \\ v &= v^{0.45} && (8) \end{aligned}$$

## 4 Results

The results we achieved are promising, but not sufficient. If we just apply Bousseau's rendering technique to the photo the resulting image is quite dark. By applying gamma correction, we can control how light and saturated an image appears. These additional controls add more complexity to controlling the watercolor processing, but still do so without the overhead of fluid simulation. As can be seen in Figure 4b, gamma correction makes the resulting image brighter and more contrasted; it is arguably a more pleasing image.

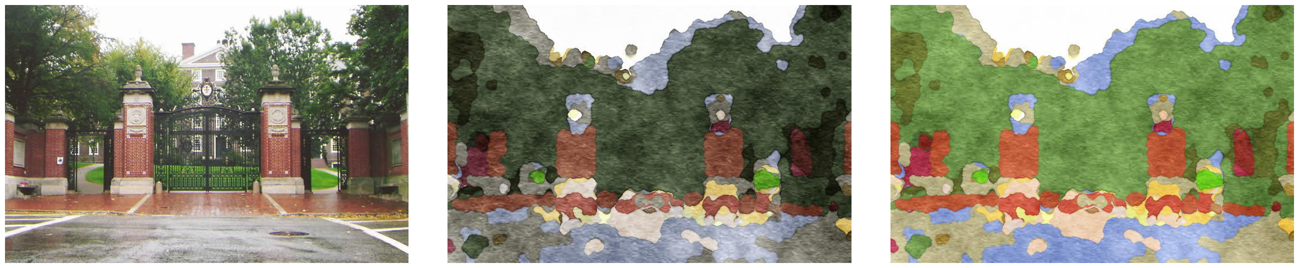

(a) *Source* (b) *Processed* (c) *Gamma Corrected*

**Figure 4:** *Watercolor Renderings*

Processing time was typically under 10 seconds in Matlab software. Most of the processing time was spent segmenting the image using mean shift and performing image wobbling, as that involves repeated convolution using a large filter.

### 4.1 Problems

Wobble size and morphological smoothing kernel size are parameters that need to be configured based on the image scale. Thus, using multiple scale images can become difficult to configure. Fortunately, Google Streetview images appear to have little variation in the scale of features.

The wobbling processing step can result in segmented regions becoming separated. In Figure 5, this happens between the blue and orange region. This problem can be reduced by choosing a smaller wobbling size, but then using a larger image becomes difficult. It may also be useful to reevaluate how the wobbling effect is generated.

Meanshift and morplogical smoothing can result in speckling and noisy regions that do not make sense in a watercolor. Figure 8 exemplifies this. Since segmentation is based on color it can remove sense of structure in the image as seen in Figures 16 and 15. In addition to this, semi-obstructed objects can become more obstructed as in Figure 18.

## 5 Conclusion

We have presented the effectiveness of different algorithms for expressive rendering in watercolor. Fluid simulation is far too complex and time consuming to run. Texture Synthesis is also slow and produces inadequate results. Faking watercolor effects is a better approach than either full simulation or texture transfer.

### 5.1 Future Work

There are potential improvements to this process. One such improvement would be to find an improved segmentation method based on structure of features. Another improvement would be to find a way to automatically configure parameters for wobbling, morphological smoothing, and gamma correction. Alternatively, finding an image-scale independent method to apply wobbling and morphological smoothing.

## Acknowledgements

Thanks to John "Spike" Hughes for his ideas and advice, Milagro Feijoo for collaborating on line drawings and watercolor and Robert "Moustache Man" Mustacchi for his support.

## References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, ACM, 217–226.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 141–149.

BOYKOV, Y., AND KOLMOGOROV, V. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 9, 1124–1137.

COMANICIU, D., AND MEER, P. 1999. Mean shift analysis and applications. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, IEEE Computer Society, Washington, DC, USA, 1197.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 421–430.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 341–346.

GAGE, M. Michael gage: Artist. `http://www.michaelgage.co.uk/`.

GOOGLE. Google street view. `maps.google.com`.

HUANG, J., BUE, B., PATTATH, A., EBERT, D. S., AND THOMAS, K. M. 2007. Interactive illustrative rendering on mobile devices. *IEEE Comput. Graph. Appl. 27*, 3, 48–56.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3 (July), 277–286.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. 22*, 3, 313–318.

PERLIN, K. 1985. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3, 287–296.

QUILLET, J.-C., THOMAS, G., GRANIER, X., GUITTON, P., AND MARVIE, J.-E. 2006. Using expressive rendering for remote visualization of large city models. In *Web3D '06: Proceedings of the eleventh international conference on 3D web technology*, ACM, New York, NY, USA, 27–35.

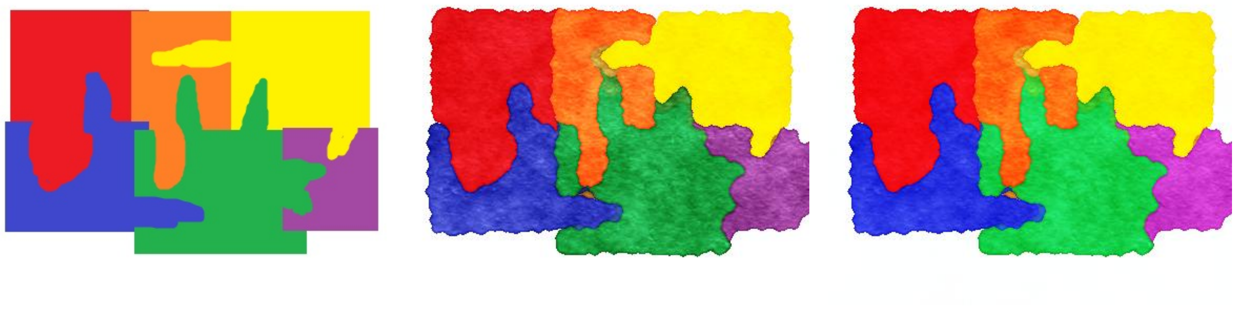


**Figure 5:** *Test image using fully saturated colors*

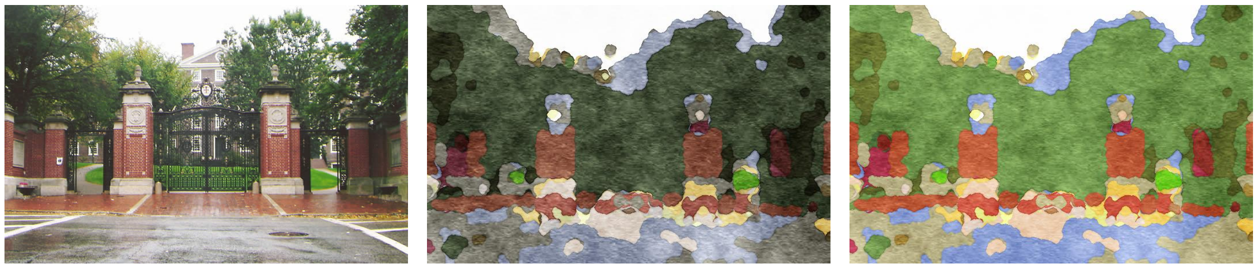


**Figure 6:** *Initially too dark, gamma correction brightens this image*

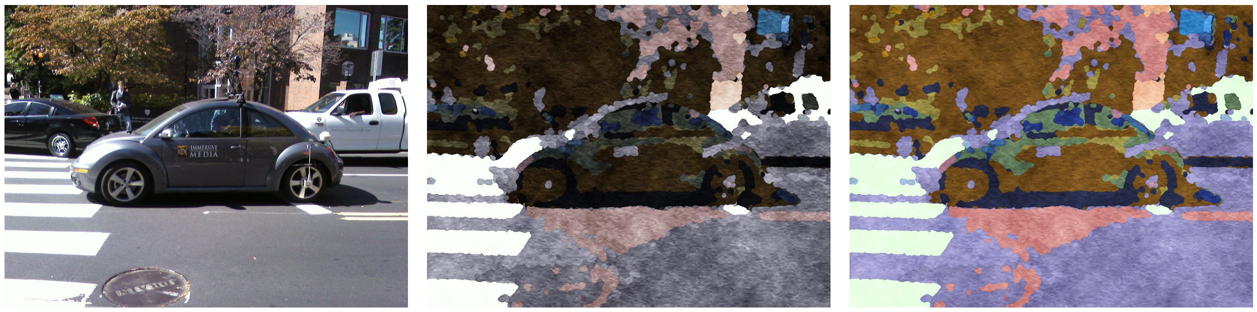


**Figure 7:** *Initially quite grey, gamma correction brings out pinks and blues*

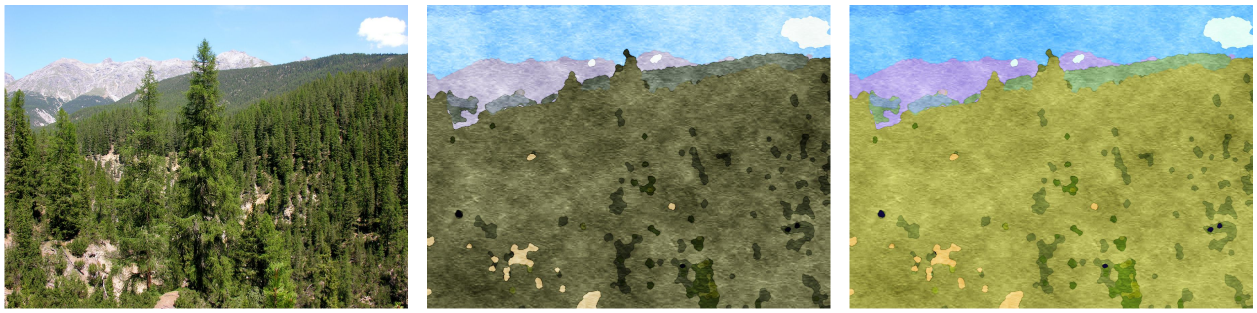


**Figure 8:** *Landscape segmented into very simple regions*

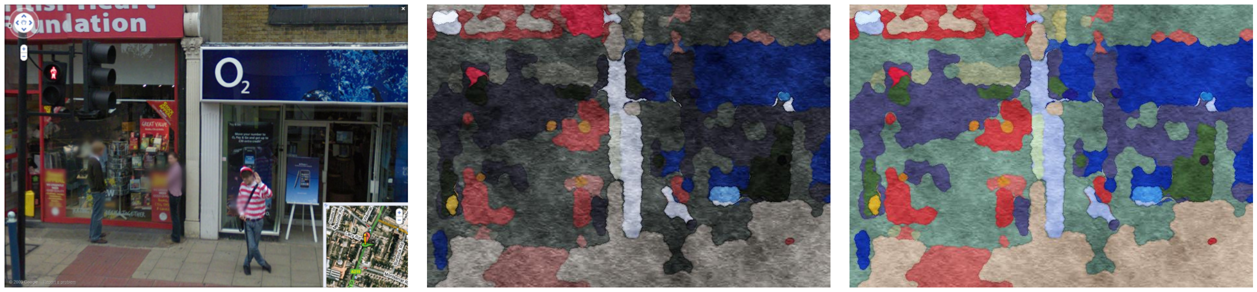


**Figure 9:** *Too cluttered to get detail out with just color*

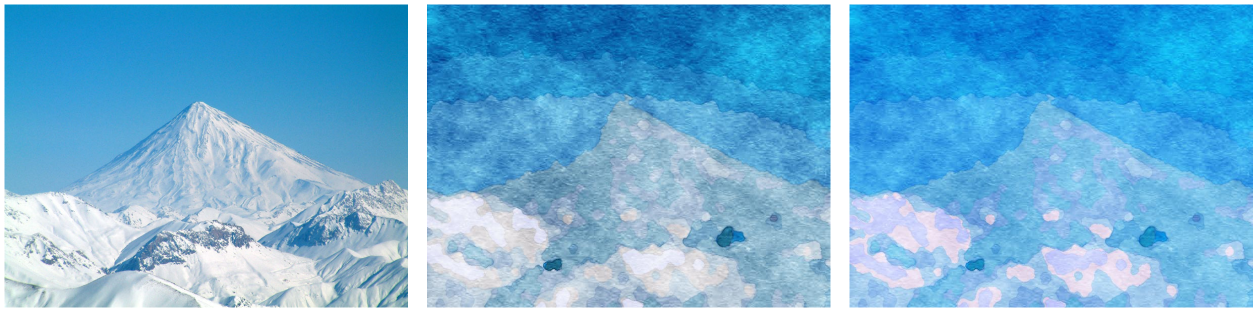


**Figure 10:** *Landscape that is not very affected by our processing*

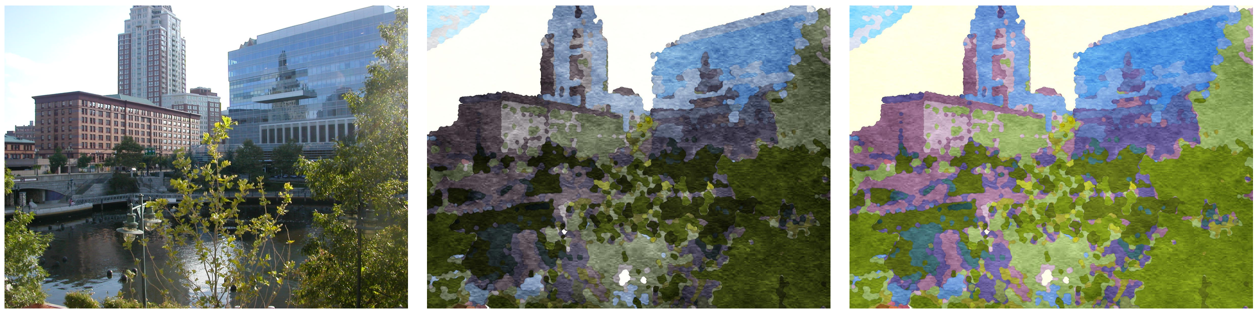


**Figure 11:** *Cityscape*

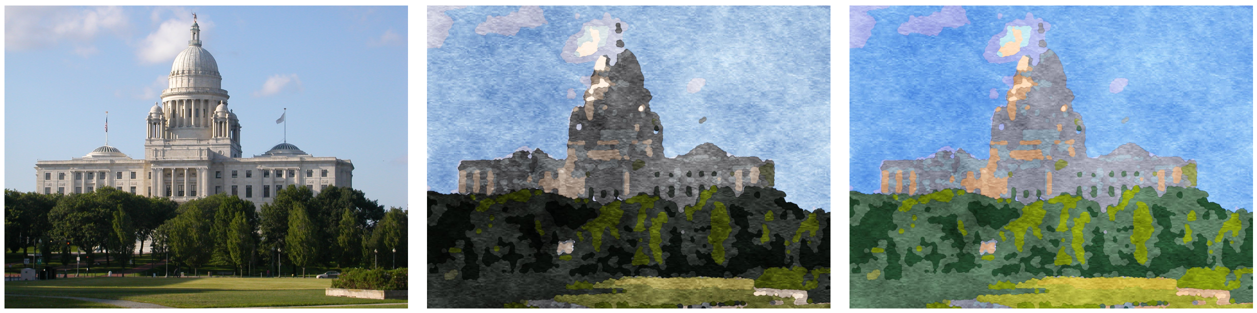


**Figure 12:** *Cityscape*

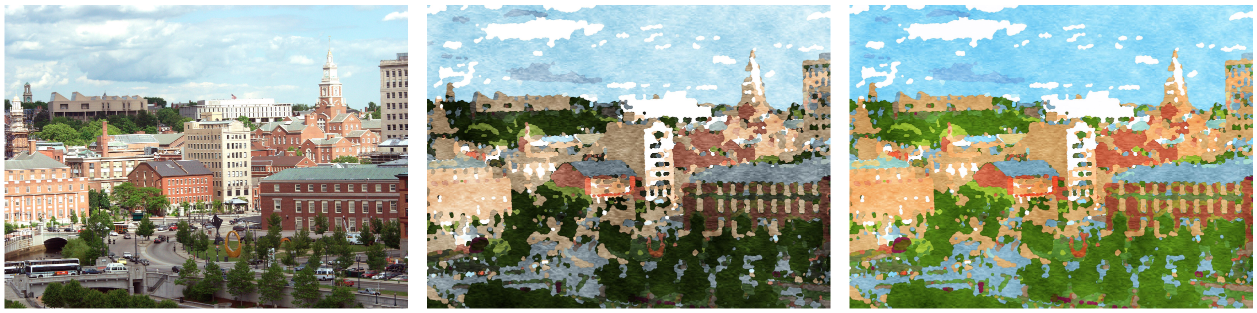


**Figure 13:** *Cityscape with arguably too much detail for a watercolor painting*

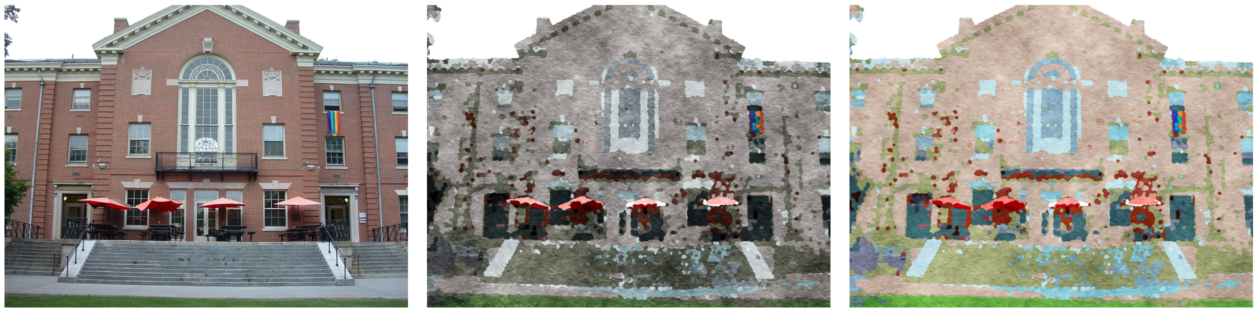


**Figure 14:** *Building clearly visible, but color is not pleasant*

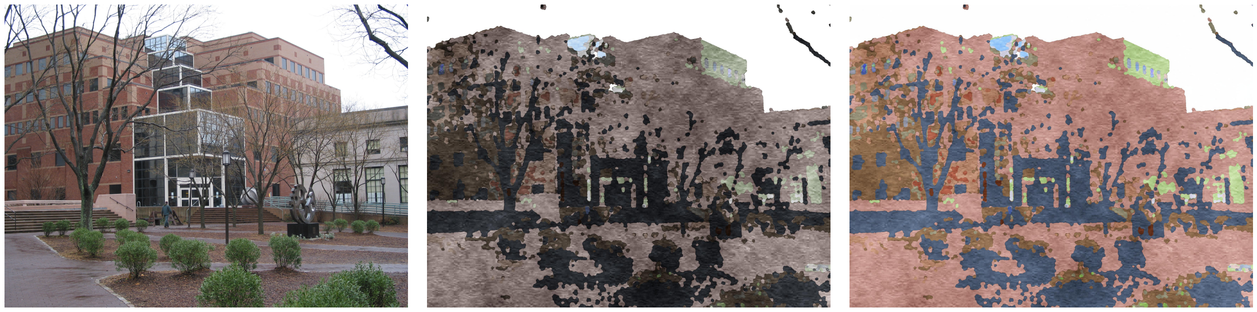


**Figure 15:** *Overly simplified segmentation*

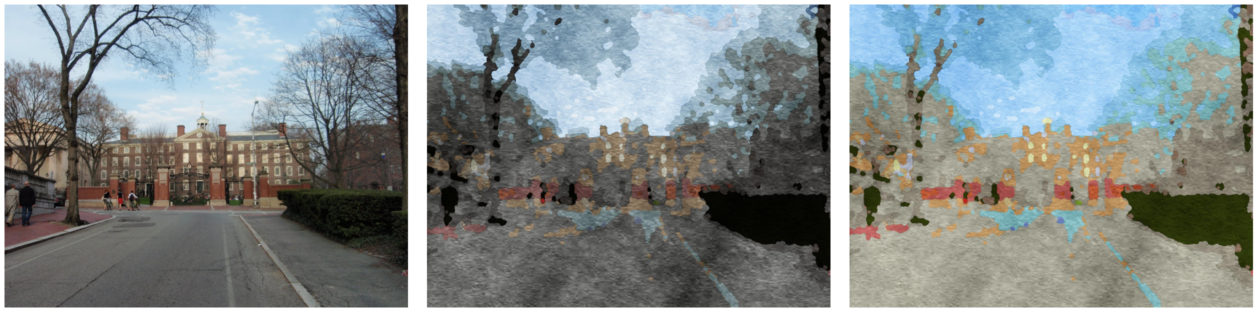


**Figure 16:** *Overly simplified segmentation*

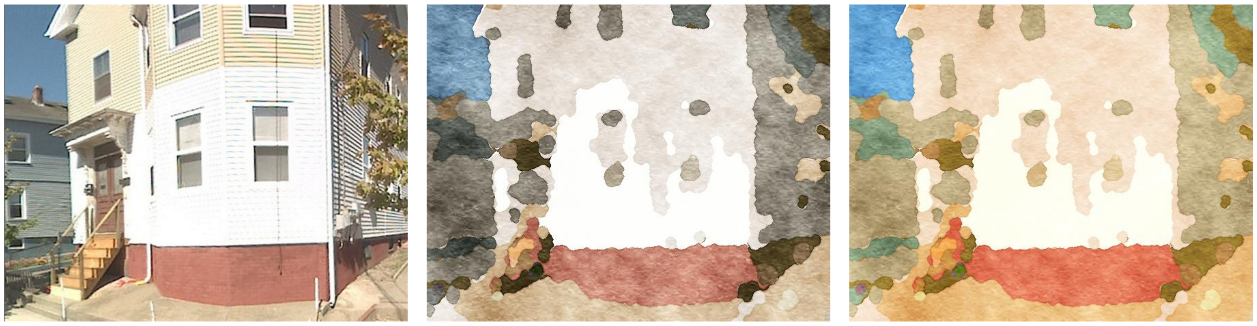


**Figure 17:** *Structure clearly visible*

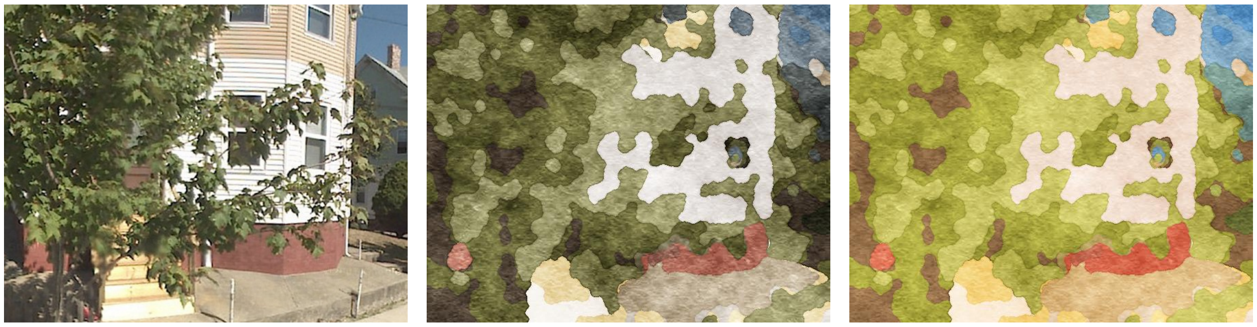


**Figure 18:** *Same structure obstructed by trees*