Watercolor Inspired Non-Photorealistic Rendering for Augmented Reality

Jiajian Chen, Greg Turk and Blair MacIntyre
School of Interactive Computing, GVU Center
Georgia Institute of Technology

Abstract

Non-photorealistic rendering (NPR) is an attractive approach for seamlessly blending virtual and physical content in Augmented Reality (AR) applications. Simple NRP techniques, that use information from a single rendered image, have been demonstrated in real-time AR systems. More complex NRP techniques require visual coherence across multiple frames of video, and typical offline algorithms are expensive and/or require global knowledge of the video sequence. To use such techniques in real-time AR, fast algorithms must be developed that do not require information past the currently rendered frame. This paper presents a watercolor-like NPR style for AR applications with some degree of visual coherence.

CR Categories: I.3.5 [Non Photorealistic Rendering]: Hardware Accelerated Rendering—Painterly Rendering

Keywords: Non-photorealistic rendering (NPR), Augmented Reality (AR), Voronoi diagrams.

1 Introduction

Augmented reality (AR) has been suggested for a range of applications, from medical to maintenance to tourism to games. For some of these applications (e.g., medical), an AR system should keep the virtual and physical content visually distinct. For others (e.g., games), the virtual and physical content may need to be seamlessly blended. One approach is to make the virtual content as photorealistic as possible, but perfect blending of virtual and physical content in real-time in a live AR system, possibly in an unconstrained environment, is extremely difficult. An alternative approach is to use non-photorealistic (NPR) rendering techniques to create a stylized blend of the virtual and physical worlds, such that the two are indistinguishable. A variety of simple NPR styles have been implemented in AR systems, including cartoon-like, brush stroke and illustrative stylization [Fischer05a, Fischer05b, Haller04]. Unfortunately, all of the work to date has been limited to techniques that can be applied independently to each rendered frame.

In contrast, a range of interesting and compelling NPR effects, such as painterly rendering [Hertzmann00], watercolor rendering [Bousseau06, Bousseau07] and mosaic rendering [Battiato07], have been demonstrated in offline systems. These techniques typically analyze the video sequence, leveraging temporal and visual coherence in the video sequence. Unfortunately, many of these effects are extremely computationally expensive, or leverage information from future video frames, both of which are impossible in a real-time AR system.

e-mail: {johnchen, turk, blair}@cc.gatech.edu

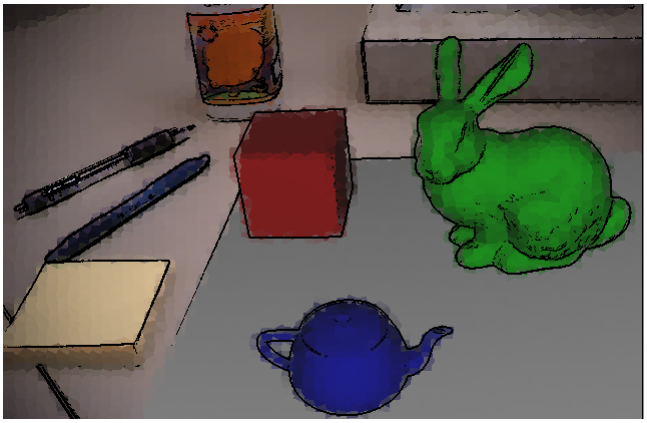

Figure 1: Watercolor-like AR. Three virtual objects (a teapot, a cube and a bunny) and several real objects (pens, notes, a bottle and a tissue box) are stylized.

The goal of our research is to create real-time AR NPR video effects of the sort that require visual and temporal coherence, such as water-colorization or mosaics. This paper presents our first step toward this goal, a water-colorization-like rendering technique for AR. Beyond describing this algorithm, we use it to illustrate the need for coherence and discuss our future plans in this area.

2 Watercolor-like Rendering for AR

Water-colorization for static images and videos is an interesting topic with a long history. [Curtis97] uses an ordered set of translucent glazes to model watercolor effects. A Kubelka-Munk compositing model is utilized to simulate the optical effect of the superimposed glazes. The method operates on each frame independently. [Bousseau07] proposes a method for maintaining temporal coherence in watercolor-like video. It employs texture advection along lines of optical flow to maintain texture coherence, and mathematical morphology to maintain abstraction coherence. All these past methods have focused on offline processing of a complete video sequence, but are too time-consuming for online rendering in a live AR video system.

In this paper we propose to use Voronoi diagrams to mimic the watercolor effects, adjusting the cells that lie along strong silhouette edges in the image in real-time to help maintain visual coherence at a relatively low temporal cost. A snapshot from our watercolor stylized AR video is shown in Figure 1.

3 NPR Algorithm Descriptions

The idea of using Voronoi diagrams is inspired by several NPR papers [Hausner01, Battiato07]. Those papers use Voronoi diagrams to create various non-rigid rendering styles, such as the digital mosaic style for images and videos. We found that the 2D Voronoi diagrams can be effectively used to produce a watercolor inspired effect for live AR video in real time. It creates a non-rigid color bleeding along the edges of objects in the video. Also, by re-

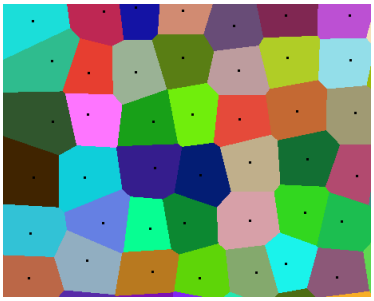

Figure 2: A Voronoi pattern in the frame buffer. We have chosen different brighter color for each region in this image to make the regions easy to see. In the actual algorithm the color of each region is determined by its region index k.

tiling Voronoi cells along edges detected in the video frame, it keeps some degree of visual coherence in the output video.

The algorithms can be divided into two parts: creating a watercolor-like style using a 2D Voronoi diagram, and keeping visual coherence by detecting strong edges and re-tiling the Voronoi cells. In the first part, an AR video frame is processed using a Voronoi pattern to produce the color bleeding effect. In the second part, Voronoi cells are re-tiled to keep visual coherence in each independent video frame.

3.1 Using Voronoi Diagrams to Tile the Image

Given an AR image that is a blend of video and computer-generated objects, our first task is to create a tiled version of this scene. We begin by creating a tiling of the plane with a Voronoi diagram, and then we color these tiles based on the original image. We use a jittered sampling of the plane in order to create the centers of the Voronoi cells for our tiling. Assume the size of the video frame is *WxH*, and that we will create *m*n* tiles. The jittered center of the Voronoi cell at the *i*-th row and *j*-th column is then chosen as follows:

$$V(i,j)=\left(\frac{W}{n}\left(j+\frac{1}{2}+random\right),\frac{H}{m}\left(i+\frac{1}{2}+random\right)\right),$$

$$0\le i<m,\ 0\le j<n,\ -\frac{1}{2}<random<\frac{1}{2}$$

To create the Voronoi cells, we use the observation that a Voronoi diagram can be created by placing a cone at each of the cell centers [Hausner01]. The portion of a given cone that is closer than all other cones delineates one of the Voronoi cells. Using graphics hardware, these cones are approximated by collections of polygons, and the closest portions of the cones are determined by depth buffering. We rasterize m*n cones at the centers V(i, j) with depth buffering enabled. During rasterization, each cone is given a unique RGB color so that the region of each cone can be identified by examining the pixel colors from the framebuffer. The framebuffer that we read back is divided into m*n Voronoi regions by the Voronoi cells. Figure 2 shows a low resolution version of what the frame buffer looks like after the cones have been rasterized. The screen is divided into many irregular Voronoi regions.

The index of the Voronoi region for pixel (i, j) is decoded from its RGB color. These regions form a Voronoi pattern that we then use to create a watercolor inspired style for AR video frame. To produce the final tiled image, we color each Voronoi cell according to the average color of the original image within the cell. Figure 3 is an original AR video frame, and Figure 4 is the frame stylized using such a Voronoi tiling. Note that others have used Voronoi tilings to create mosaic rendering styles for single image off-line non-photorealistic rendering [Hausner01, Battiato07].

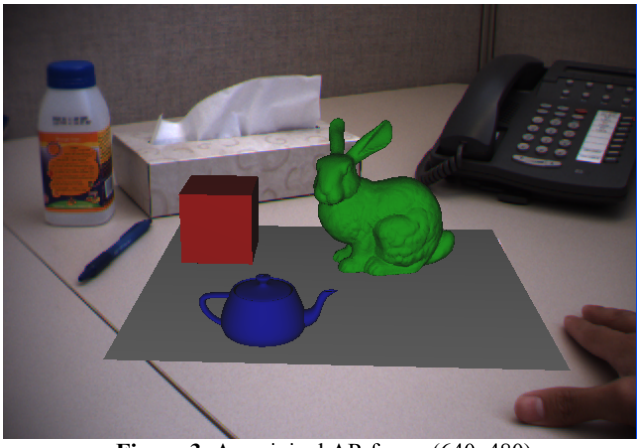

Figure 3: An original AR frame (640x480).

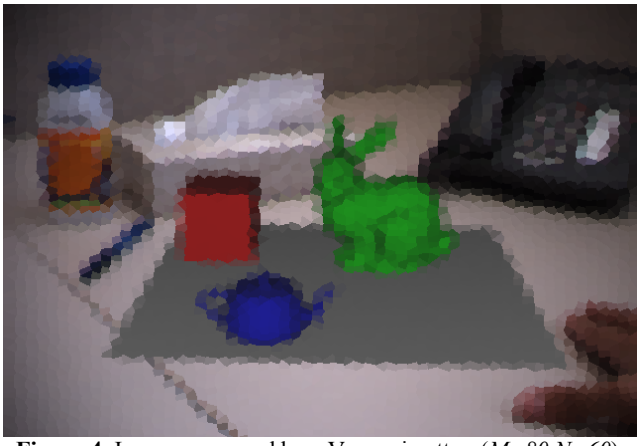

Figure 4: Image processed by a Voronoi pattern(*M=80,N=60*).

Many watercolor paintings include dark strokes that highlight object silhouettes. In the next step we mimic this style by detecting and drawing strong edges in the video frame. By combining these two steps (tiling and edge drawing) we produce a watercolor inspired NPR style for augmented reality.

3.2 Edge Detection

Edge detection is a common technique for NPR rendering, since many NPR styles highlight silhouettes and strong edges. For example, [Fisher05a] uses YUV space to detect strong edges in the AR video frame.

In our algorithm, we first render the virtual objects using a simple toon shader, and later detect edges in this AR frame. The reason that we use a simple toon shading instead of default OpenGL Gouraud shading is that we want to exaggerate the color difference between so we can get better edge detection results.

Our toon shader performs intensity thresholding to create just three discrete suface intensities instead of using continuous shades for an object's surface. The code of our fragment shader for toon shading is given below.

```
varying vec3 lightDir,normal;
uniform vec3 myColor;
void main()
{
        vec4 color;
        vec3 n = normalize(normal);
        float intensity = dot(lightDir,n);
        if (intensity > 0.7)      color = vec4(myColor, 1.0);
        else if (intensity > 0.2) color = vec4(0.4 * myColor, 1.0);
        else                      color = vec4(0.1 * myColor, 1.0);
        gl_FragColor = color;
}
```

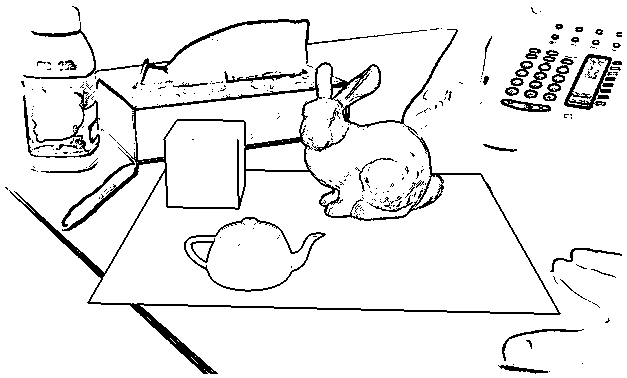

Figure 5: Detected edges in an AR frame.

The virtual objects and the video background are rendered and combined to a texture. Edges are detected in a fragment shader in the GPU so we don't need to copy the pixel data of this texture to the CPU. The pixel information in both of RGB and YUV color space are used to detect edges. The pixel conversion between these two color spaces can be performed in a shader as shown below.

$$mat4\ RGBtoYUV;$$
$$vec4\ rgb = texture2D(tex, texcoord);$$
$$vec4\ yuv = RGBtoYUV * rgb$$

The equation for edge detection is shown below. ∇ is the color difference along the x-axis and the y-axis. If this value is larger than a threshold then this pixel is considered to be an edge pixel.

$$Edge_{YUV} = (1-\alpha)\cdot|\nabla_Y| + \alpha\cdot\frac{|\nabla_U|+|\nabla_V|}{2}$$
$$Edge_{RGB} = \frac{|\nabla_R|+|\nabla_G|+|\nabla_B|}{3}$$
$$Edge? = \begin{cases} Yes, & if\ \beta\cdot Edge_{YUV} + (1-\beta)\cdot Edge_{RGB} > threshold \\ No \end{cases}$$

An AR video frame with edge detection is shown in Figure 5. The silhouettes are detected for three virtual objects (the teapot, the cube and the bunny) and for the objects in the real world.

3.3 Re-tiling Voronoi Cells

Visual coherence is a key challenge in video NPR processing. Many papers have studied on this topic. [Bousseau07] uses advection texture to keep temporal coherence in watercolor rendering for a general video. Some NPR papers use geometry information to maintain temporal coherence. However, those methods are too expensive to use for real-time AR video.

We keep visual coherence by re-tiling Voronoi cells along the strong edges in the scene. If we kept the same Voronoi pattern and apply it to each video frame all the time, the output video would have an undesired "shower door" look. To avoid this appearance we re-tile the Voronoi cells along the strong edges in each video frame, thus generating a new Voronoi pattern. Our approach is to pull each Voronoi cell *towards* strong edges in the image. This gives the appearance of color bleeding between regions since the cells will straddle two regions that are separated by an edge. Note that this is in some sense the reverse of Hausner's technique of having Voronoi cells aviod strong edges [Hausner01]. Our procedure is as follows:

- Go through each Voronoi region to compute the average center of all edge pixels in that region. This average edge pixel location becomes the new center for a cell. If the number of edge pixels is smaller than a threshold in a region, then keep the previous center for that region.

Figure 6: A final rendering, combining tiling and edges.

- Move the Voronoi cells to the new centers and rasterize the cones again.
- Read back the frame buffer and use it as a new Voronoi pattern.

Ideally we want to generate a new Voronoi pattern for each video frame, but it's an expensive procedure since we need to read back the frame buffer from GPU to CPU by using `glReadPixels`. To balance the speed and quality of the output AR video, we do this re-tiling periodically, re-tiling the Voronoi cells and generating a new pattern every 10 frames. We are investigating the possibility of performing the tiling entirely on the GPU, which should allow us to generate a new tiling pattern every frame.

4 Implementation Details

The workflow of our NPR renderer is shown in Figure 7.

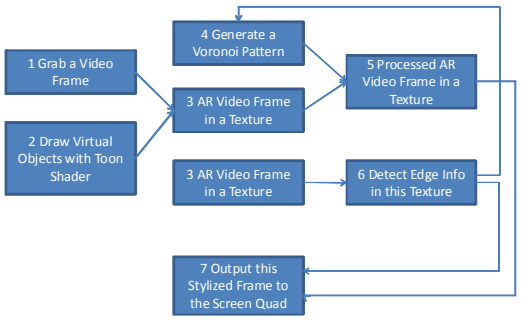

Figure 7: Workflow of the watercolor style rendering.

We used OpenGL extensions (ARB functions) and the GLSL shading language to accelerate the rendering in the GPU. All intermediate results are rendered and manipulated as textures. By directly manipulating the texture in shaders we eliminated the expensive data copy between the GPU and the CPU.

We created a Frame Buffer Object (FBO) with two frame buffers. The content of each buffer is linked with a texture ID and it can be directly used as a texture. As shown in Figure 7, most steps are done by manipulating these two textures. Shaders are used to render the virtual objects and to detect edges. The steps in the rendering procedure are listed below. The numbers given in brackets are shown in the workflow graph.

- Generate an initial Voronoi pattern (4).
- Fetch a frame from the video camera and render the video background and virtual objects (1,2) to the first frame buffer in FBO to generate the original AR frame as a texture (3).
- Copy this texture from the first frame buffer to the second frame buffer in FBO (3).

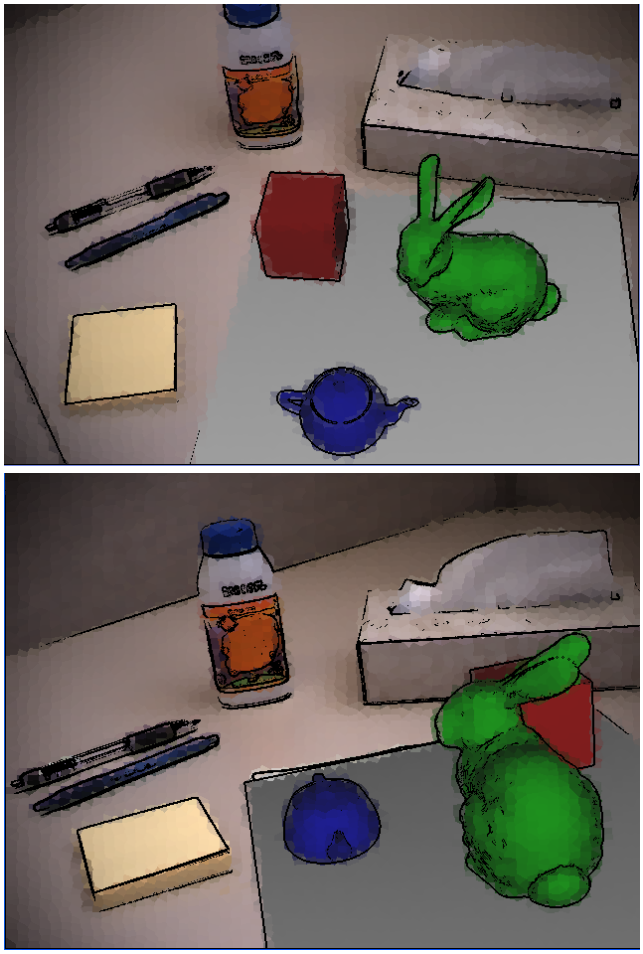

Figure 8: Two screenshots from a watercolor stylized AR video.

- Use `glMapBuffer` to access (3) and modify pixels based on the Voronoi pattern in (4), and place the processed AR video frame into a texture (5).
- Output the stylized frame to the quad. This process is done in a fragment shader. The input to this shader are two textures, one containing the tiled image and another with the edges. The shader draws the edges on top of the tiled image.
- Re-tile the Voronoi cells based on the detected edges.

The most expensive step is (5). This step computes the average color of all Voronoi regions in the current pattern and assigns the color to each pixel in the current frame. Also, this is the only step that can stall the rendering pipeline because it calls `glMapBuffer`. It would be possible to accelerate this by reading only a portion of the pixel buffer, processing the pixel data in a separate thread, and then reading another portion of the pixel buffer. We did not resort to this in our system since our video resolution is relatively small (640x480).

Another possible way to eliminate this bottleneck for higher resolutions is to perform the non-regular color averaging on the GPU. The algorithm would have the following steps:

- Generate the Voronoi pattern.
- Use two input textures in the fragment shader: the current AR frame texture and the Voronoi pattern. For each pixel *p*, search its 16x16 neighbors in the Voronoi texture and record the positions (texture coordinates offsets) of the neighbors that have the same color as the current pixel *p*.
- Compute the average color of the pixels in this list of positions in the AR frame texture and assign this color to *p* in the final output image.

This approach is similar to performing convolution on the GPU. The difference between this and the typical method for convolution is that in our case each pixel has a unique kernel value and size, and kernels for a pixel may change over time.

Our system runs on a Windows PC with Xeon 2.2GHz CPU and an nVidia G7950 graphics card. This graphics card supports OpenGL extensions (e.g., Pixel Buffer Object and Frame Buffer Object) and GLSL. We use a PointGrey flea camera, which provides a crisp and bright video that is superior to Webcam videos. Pictures in this paper are captured from a live AR video. The video resolution is 640x480 and the frame rate is 15fps. Our algorithm uses the GPU to effectively process the image in shaders, so the method is applicable for higher resolution AR videos with a fast graphics card.

5 Conclusion and Future Work

We have demonstrated the use of Voronoi diagrams to create a real-time watercolor inspired style for live AR video. We achieve visual coherence by detecting strong edges and re-tiling the Voronoi cells along these edges. A near-term goal is to see if the Voronoi regions can be constructed entirely on the GPU. Currently this method of keeping visual coherence considers only independent video frames. In future work, we hope to use information from multiple frames to improve between-frame coherence. We also plan to investigate using simulated paper and canvas textures, as done by [Cunzi03].

References

BATTIATO, S., BLASI G., FARINELLA G. AND GALLO G. 2007. Digital Mosaic Frameworks - An Overview. In *Computer Graphics Forum Vol. 26(4)* 794-812.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 141-149.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video Watercolorization using Bidirectional Texture Advection. In *ACM Transactions on Graphics,* 26(3): 104:1-104:7.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J-D., DURAND, F. 2003. Dynamic canvas for non-photorealistic walkthroughs. In *Graphics Interface 2003*.

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

FISCHER, J., AND BARTZ, D. 2005a. Real-time cartoon-like stylization of AR video stream on the GPU. *Technical Report WSI-2005-18,* Wilhelm Schickard Institute, 2005

FISCHER, J., BARTZ, D., AND STRASSER, W. 2005b. Artistic reality: Fast brush stroke stylization for augmented reality. In *Proc. of ACM VRST 2005*. 155-158.

HALLER, M. Photorealism or/and Non-photorealism in Augmented Reality. 2004. In *ACM International Conference on Virtual Reality Continuum and its Applications in Industry*, 189-196.

HAUSNER, A. Simulating Decorative Mosaics. 2001. In *SIGGRAPH 2001,* 573-580

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 7-12.

LUM, E. B., AND MA, K.-L. 2001. Non-photorealistic rendering using watercolor inspired textures and illumination. In *Pacific Graphics*, 322-331.