Real-Time Rendering of Watercolor Effects for Virtual Environments

Eugene Lei and Chun-Fa Chang

Department of Computer Science
National Tsing Hua University
{zenith,chang}@ibr.cs.nthu.edu.tw

In this paper, we present an empirical approach for rendering realistic watercolor effects in real-time. While watercolor being a versatile media, several characteristics of its effects have been categorized in the past. We describe an approach to recreate these effects using the Kubelka-Munk compositing model and the Sobel filter. Using modern per-pixel shading hardware, we present a method to render these effects in an interactive frame-rate.

Keywords: Non-photorealistic rendering, graphics hardware, image processing, Kubelka-Munk, watercolor.

1. Introduction

While non-photorealistic rendering (NPR) is a relatively new field, many different approaches for variant styles have been proposed through the past decade. With increasing computational power and improving graphics hardware architecture, a number of approaches on implementing those effects in real-time has been proposed. So far, the real-time automation scene of the NPR field mainly focuses on simulating stroke-based effects, such as pencil hatching [14], pen-and-ink [15], engraving [6] etc. Medium that requires extensive cell-to-cell interaction of pigments, such as watercolor, is difficult to render in real-time realistically.

In this paper, we followed the work of Curtis et al. [2], which characterizes several defining effect of watercolor. We propose a system that can render these effects in real-time, with the assistance of per-pixel shading hardware. We also demonstrate the use of Sobel filter for simulating edge darkening and granulation characteristics of watercolor.

1.1. Related Work

The pioneering work of Small [1] used a Connection Machine to simulate watercolor. Curtis et al. [2] used a similar approach, which used an improved physical simulation model to achieve more realistic effects. Their method, while looks realistic

and physically sound, requires too much computation to achieve an interactive frame-rate. Inspired by the work of Lake et al. [5], we use a "color-band" approach instead of simulating fluid and pigment interaction for each frame. Such color-band is defined by using a simplified Lit-sphere [4] interface. We use a pigment-blending model based on the Kubelka-Munk [3] (KM) model to simulate the composition of different pigments.

One of the most obvious effects of watercolor is edge darkening, i.e. a dark deposit at the edge of a wet-on-dry stroke. Intuitively, recreating this effect as a post-process is similar to the silhouette-finding problem. While silhouette sketching is a well-understood field [9, 13], conventional methods focus on finding the silhouette of a geometric model. Instead we use the Sobel filter [18] to find the edge of every painted region efficiently. The work of Nienhaus et al. [8] describes a possibility to use 2D image processing process to enhance result in real-time, and we implement our Sobel filter in a similar approach.

Recently some commercial applications and games used the Kuwahara filter [16] to create a watercolor style NPR. While this technique is fast, it failed to recreate the rich appearance of watercolor paintings, which depends on the KM model.

1.2. Organization

The rest of this paper is organized as follows. The next section reviews the characteristics of watercolor, as described by Curtis et al., and provides an overview of our approach. In section 3 we describe the system details of our implementation. We present the results and performance evaluation in Section 4.

2. Overview

Curtis et al. categorized several distinctive effects of watercolor: edge darkening, backruns, granulation, flow effects and glazing. Since our implementation is an automated system for creating artistic imagery, there are a few assumptions on when and where to apply such effects.

2.1. Watercolor Effects

Edge darkening is the key effect that most artists rely upon, and one of the most defining characteristics of watercolor. It is created when the pigment migrates from the interior of a wet region towards its edges as the paint begins to dry, leaving a dark deposit at the edge. We treat the entire painted area as the wet region, and add the darken edge by applying the Sobel filter.

Granulation of pigment creates a grainy texture concentrated on the peaks and valleys in the paper. The amount of granulation varies from paper to paper. We

simulate the granulation effect by specifying a *granulation constant* for each kind of pigment, and use the paper texture and the Sobel filter to emphasize this effect.

Backrun is the effect when water is added to a wet painted area, carrying pigment along outwards, leaving a darkened, branched shape. Flow effect is observed when wet paint is applied on wet paper. The wet surface allows the pigment to spread freely, leaving a soft branching pattern. Color glazing is the process of adding thin layers of watercolor, causing pigments to be mixed optically.

We treat the backrun, flow effect and glazing as a universal effect. We simulate these effects by using a "color-band" approach. The color-band is a one-dimensional texture, which is created by mixing different amount of pigments in user-specified position. The color-band is then mapped onto the geometry using Lake et al.'s cartoon shading [5]. Additional flow effect is simulated in real-time using pixel-shader hardware.

2.2. Approach

Our system is divided into two phases: *Color-band specifying* and *watercolor shader*. The *color-band specifying* phase lets the user create a color-band for each object in the scene, using an isotropic lit-sphere interface [4]. The lit-sphere interface is set on top of a watercolor simulation engine, created using Curtis et al.'s water-flowing model [2]. In this stage, the flowing effect, backrunning and glazing are simulated as the user literally paints in the lit-sphere interface.

The *watercolor shader* is the main phase in our automated system. Using per-pixel shading hardware, we simulate edge darkening, granulation, paper texture and further flowing effects using shader programs. Details of the hardware-shader are discussed in section 3. A diagram is shown in Figure 1 to illustrate the various stages in our system.

2.2.1. Color-band Specifying

As mentioned in section 1, we use the Kubelka-Munk (KM) model [3] to perform the optical mixing of pigments. Each pigment is assigned a set of absorption coefficients and scattering coefficients, and a granulation constant. We use the KM model to compute the resulting color in RGB space.

User can choose from a range of pigments in the *pigment library*, and the mixture amount of each pigment. The pigment is then applied to the lit-sphere via a virtual paintbrush, and the flowing and mixing of different pigments is simulated. Flowing effect is isotropic since the paper texture is not simulated in this stage.

After the user finished painting on the lit-sphere, a one-dimensional color-band will be calculated by sampling an arbitrary angle along the sphere's radius. Along with the resulting color in RGB, a granulation variable is also stored in the color-band as the alpha value. The granulation variable defines how visible the granulation effect

will be. It is the sum of the granulation constant for each pigment multiplied by the intensity (‘thickness”) of said pigment in each cell.

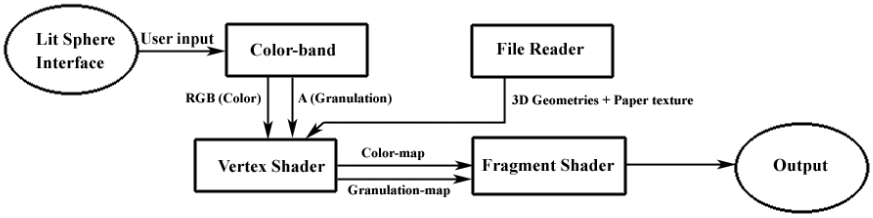

Fig. 1. System Diagram

3. System Details

The automatic rendering system takes the original 3D geometric models as the input, and apply the watercolor stylization to the 3D scene using vertex and fragment shaders.

The shaders we used are written in the NVIDIA Cg language [7]. Two shader scripts are used in generating our watercolor effect. First we take the 3D objects and the color-band to generate a *color-map* and a *granulation-map* using a vertex shader script. Then we process the *color-map* using a fragment shader. The shader takes the *color-map* and a *paper-texture* as the inputs. It combines them with a *Sobel edge map*, which is generated by the shader, to create various watercolor effects.

3.1 Vertex Shader

This part is similar to Lake et al.’s cartoon shading [5]. We return the result of the Phong shading function as a texture coordinate.

```
TexCoord0 = (Diffuse*(N·L) + Specular*(N·H));
```

Where N is the normal vector, L is the light vector and H is the halfway vector between the eye vector and light vector. *Diffuse* and *Specular* are the amount of diffuse and specular reflection. In our experiment both are set to 1.

The script is run twice: first pass takes the RGB element of the color-band as the input, and returns a *color-map*. Second pass takes the alpha value of the color-band (which stores the granulation variable) as the input, and returns a *granulation-map*. Both maps are created using OpenGL’s render-to-texture capability, and used as the input for the fragment shader.

3.2 Fragment Shader

In this stage, we take the *color-map*, *granulation-map* and the paper texture as input. This shader processes the *color-map* and returns a watercolor styled image, which is our final result. The paper texture is a 2D texture and is treated as a height field. It can be specified by the user or generated using Worley's cellular texturing process [17].

When wet watercolor paint is applied on dry paper, there is a wobbling effect along the edge of the stroke, due to the raggedness of the paper. We simulate this effect first by distorting the *color-map* using the *paper-texture*.

```
colormap_coord = coord+((tex2D(paper_map,coord)-M/2)*D)
```

Where *coord* is the current texture coordinate, *M* is the mean luminance of the paper texture, *D* is the amount of distortion. In our experiment *M*=0.8, *D*=0.0125.

Then we need to refine the image for processing with the Sobel filter. As mentioned in section 2, we use the Sobel filter to create the edge-darkening and granulation effects. Therefore we subtract an amount of *paper-texture* from the original color. The amount of subtraction depends on the *granulation-map*. The higher the granulation value, the more we subtract from the original color. We call the result in this stage a *Sobel color-map*.

```
Sobel_colormap = org_color –
    (tex2D(paper_map, coord)*tex2D(gran_map, coord)*G)
```

Where *org_color* is the original color in *color_map* distorted by the paper texture. *G* is the weight of granulation effect. In our experiment *G*=0.5. The Sobel filter detects the discontinuities of the *Sobel color-map*. We intentionally create discontinuities where granulation value is high. When the Sobel filter is applied on the *Sobel color-map*, the resulting edge map will help us create the edge-darkening and granulation effects simultaneously.

After processing the *Sobel color-map* with the Sobel filter (code of which is included in Appendix), finally we put all the results together.

```
out_color = (1-W) + org_color*W – paper_map*P –
    sobel_edge_map*S
```

The result is a weighed sum of the original color (after distortion), the paper color, and the Sobel edge map, weighted *W*, *P* and *S* respectively. In our experiment, we set $W = 0.6$, $P = 0.1$, $S = 0.3$.

4. Results and Discussion

We used three sample scenes as our example, shown in Figure 2 and 3. In the "Teapot" scene, we can see that by using the KM model, we can create rich watercolor-style mixing of colors. In the "Bamboos" scene, we can see that even with

a simple light-to-dark color-band, our shader can still produce stylish results similar to hand-drawn paintings.

Fig. 2. (Left) Fruit and Vase, 7880 polygons. Notice the different amount of granulation in different paints. (Right) Bamboos, 11080 polygons. Notice how the 2D texture distortion creates the wobbling effect.

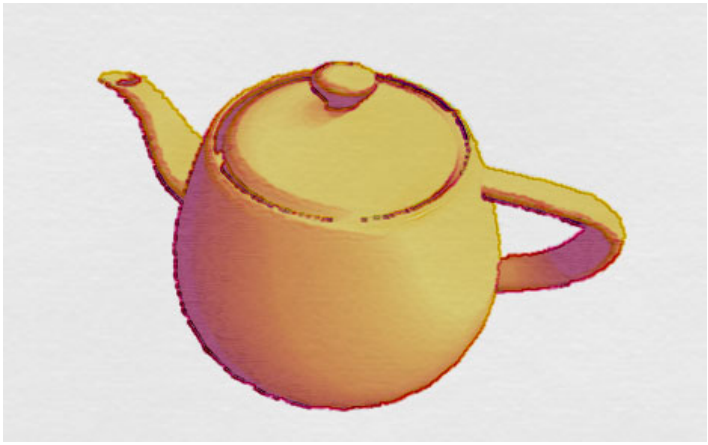

Fig. 3. Teapot, 4096 polygons.

4.1 Performance

We have tested our application on a Pentium 4 3.0GHz machine with an NVIDIA GeForce 5900FX graphics board. All scenes run at about 20 frames-per-second in resolution of 512*512 pixels. The performance depends more on the rendering

resolution than the number of polygons, since the fragment shader must perform its operation for every pixel.

4.2 Conclusions and Future Work

We have introduced a relatively simple approach to render realistic watercolor effects in real-time. We demonstrated by using a combination of color-band shading and the effective use of Sobel filter and 2D scene distortion, we can give the scene a stylish appearance.

For now the flowing effect is simulated only by the color-band and texture distortion. In the future we will try to create a more realistic effect using texture splats [11].

5. References

1. David Small. “Simulating watercolor by modeling diffusion, pigment, and paper fibers.” In Proceedings of SPIE ’91. February 1991.
2. Curtis, C. j., Anderson, S.E., Seims, J. E., Fleischer, K. W. and Salesin, D. H. “Computer-Generated Watercolor.” In Proceedings of SIGGRAPH 87, Computer Graphics Proceedings, Annual Conference Series, edited by Turner Whitted, pp. 421-430, Reading, MA: Addison-Wesley, 1997.
3. P. Kubelka. “New contributions to the optics of intensely light-scattering material, part ii: Non-homogeneous layers.” J. Optical Society, 44:330, 1954.
4. Peter-Pike J. Sloan, William Martin, Amy Gooch, Bruce Gooch. "The Lit Sphere: A Model for Capturing NPR Shading from Art." Proceedings of Graphics Interface 2001.
5. Lake, A., Marshall, C., Harris, M., and Blackstein, M. “Stylized Rendering techniques for scalable real-time 3D animation.” In NPAR 2000: First InternationalSymposium on Non-Photorealistic Animation and Rendering, edited by Jean-Daniel Fekete and David H. Salesin, pp. 13-20, New York: ACM SIGGRAPH, 2000.
6. Freudenberg, B. “A Non-Photorealistic Fragment Shader in OpenGL 2.0.” SIGGRAPH 2002 Talk, San Antonio, July 2002.
7. Fernando, R., Kligard, M.J. “The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics” Nvidia Corporation, Addison-Wesley, 2003.
8. Nienhaus, M., Doellner, J. “Edge-Enhancement – An Algorithm for Real-Time Non-Photorealistic Rendering” WSCG '03 - The 11-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2003, Plzen, Czech Republic, February 3 - 7, 2003, pp. 346-353.
9. Markosian, L., Kowalski, M. A., Trychin, S. J., Bourdev, L. D., Goldstein, D., and Hughes, J. F. “Real-Time nonphotorealistic rendering.” In Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series, edited by Turner Whitted, pp. 415-420, Reading, MA: Addison-Weslsy, 1997.
10. Willians, L.”Shading in Two Dimensions.” In Graphics Interface ’91, pp. 143-151, San Francisco: Morgan Kaufmann Publishers, 1991.
11. Roger A. Crawfis, Nelson Max. "Texture Splats for 3D Scalar and Vector Field Visualization." IEEE Visualization '93 Proceedings.

12. Gooch, B., Sloan, P. S., Gooch, A., Shirley, P., and Riesenfeld, R. “Interactive Technical Illustration.” ACM Symposium on Interactive 3D Graphics 1999, Atlanta, GA, pp.31-38, April 1999.
13. Raskar, R., and Cohen, M. “Image Precision Silhouette Edges.” ACM Symposium on Interactive 3D Graphics 1999, Atlanta, pp. 135-140. 1999.
14. E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Realtime hatching," in Proceedings Computer Graphics (ACM SIGGRAPH) , 2001, pp. 581--586.
15. Freudenberg, B., Masuch, M., and Strothotte, T. "Real-Time Halftoning: A Primitive For Non-Photorealistic Shading." 13 Eurographics Workshop on Rendering. Pisa, Italy, pp.1-4, June 2002.
16. Jason L. Mitchell. "Real-Time 3D Scene Post-Processing", Game Developers Conference 2003 Talk.
17. Steven P. Worley. “A cellular texturing basis function.” In SIGGRAPH ‘9 Proceedings, pages 291-294, 1996.
18. Rafael C. Gonzales, Richard E. Woods. “Digital Image Processing.” Addison Wesley, 1993.

Appendix – Sobel filter implemented by fragment shader

```
#define SOBEL_COLOR ORG_COLOR-(tex2D(paper_map,
coord)*tex2D(gran_map, coord)*G)

float2 coord = tCoords;
coord.x = tCoords.x-offset; coord.y = tCoords.y-offset;
float3 color00 = SOBEL_COLOR;
coord.x = tCoords.x; coord.y = tCoords.y-offset;
float3 color01 = SOBEL_COLOR;
coord.x = tCoords.x+offset; coord.y = tCoords.y-offset;
float3 color02 = SOBEL_COLOR;
coord.x = tCoords.x-offset; coord.y = tCoords.y;
float3 color10 = SOBEL_COLOR;
coord.x = tCoords.x; coord.y = tCoords.y;
float3 color11 = SOBEL_COLOR;
coord.x = tCoords.x+offset; coord.y = tCoords.y;
float3 color12 = SOBEL_COLOR;
coord.x = tCoords.x-offset; coord.y = tCoords.y+offset;
float3 color20 = SOBEL_COLOR;
coord.x = tCoords.x; coord.y = tCoords.y+offset;
float3 color21 = SOBEL_COLOR;
coord.x = tCoords.x+offset; coord.y = tCoords.y+offset;
float3 color22 = SOBEL_COLOR;
float3 Sobel_x = (-1)*color00+(-2)*color01+(-1)*color02
 +(1)*color20 + (2)*color21 + (1)*color22;
float3 Sobel_y = (-1)*color00+(-2)*color10+(-1)*color20
 +(1)*color02 + (2)*color12 + (1)*color22;
float3 Sobel_edge_map = abs(Sobel_x)+abs(Sobel_y);
```

Where offset is the size of a cell in the Sobel filter, which is 1/(color_map.width).