

Hardware Accelerated Pigment-Based Rendering System

Gabe Smedresman
Yale University
GabeSmed@yahoo.com

Abstract

In this paper, I present a real-time implementation of a watercolor-inspired rendering system outlined by Eric Lum and Kwan-Liu Ma in Non-Photorealistic Rendering using Watercolor inspired Textures and Illumination [1]. The major innovation is translating and optimizing the Kubelka-Munk pigment model for hardware shaders.

Introduction

Emulating the style of a watercolor painting in computer graphics is a problem that has been tackled in many ways, from simple aesthetic approximations to full fluid flow analysis [2]. I choose to follow in the steps of Lum and Ma, in creating a rendering system loosely based on the watercolor media, but not to the point of physical accuracy. However, I update the implementation of this method to utilize current vertex and pixel shaders available in commercial graphics hardware, bringing a previously non-interactive process to real-time speeds.

Shading Model

Most conventional computer rendering systems use a realistic approach: starting with a completely dark scene and adding lights, which illuminate surfaces. I, however, pursue a different approach emulating that of the watercolor artist. I start with a white or off-white page and add two layers of colored pigment.

The first layer, the *wash layer*, indicates the general color of the object. The thickness of this surface follows a general practice by watercolor artists: covering an object with an initial coat of a basic color, but leaving highlights and bright areas without pigment,



rather than adding light pigment later. The thickness of the wash layer is determined by the following equation. Note that it is essentially the inverse of a specular lighting equation, where H denotes the half vector of the viewing angle and lighting angle, and N denotes the surface normal.

$$\text{wash thickness} = (1 - (H \cdot N)^n) k_{\text{specular}} k_{\text{wash}}$$

The other layer, the *unlit layer*, describes the side of the object facing away from the light, and the shadow of the object. The base thickness of this layer is determined as the inverse of a typical diffuse lighting model, with L as the lighting angle and N the surface normal:

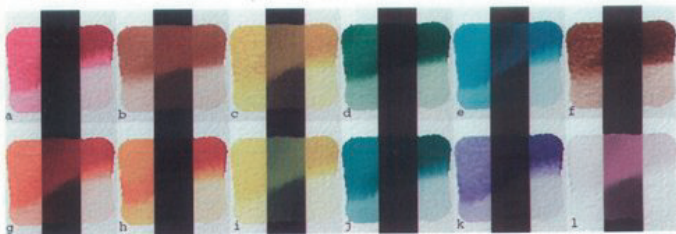
$$\text{unlit thickness} = (1 - (L \cdot N)) k_{\text{unlit}}$$

This pigment layer is typically blended with the wash layer, as to allow the general color of the object to show through. Furthermore, this thickness is modified by a gain and bias: the bias, or shift, adjusts the prominence of the unlit layer, and the gain, or scaling, tightens the gradient between fully lit and fully unlit, as to more accurately represent the gradient an artist might choose.

If it is desired, the thickness of the wash layer can be decreased in the presence of unlit pigment. This results in a purer color for the shadowed parts of a model, and less pigment mixing, a more artistically realistic result.

KM Pigment Layering Model

These layers are then assigned pigments, which are blended according to the well-known Kubelka-Munk model for compositing layers of pigment.



	PIGMENT	K_r	K_g	K_b	S_r	S_g	S_b	ρ	ω	γ
a	"Quinacridone Rose"	0.22	1.47	0.57	0.05	0.003	0.03	0.02	5.5	0.81
b	"Indian Red"	0.46	1.07	1.50	1.28	0.38	0.21	0.05	7.0	0.40
c	"Cadmium Yellow"	0.10	0.36	3.45	0.97	0.65	0.007	0.05	3.4	0.81
d	"Hookers Green"	1.62	0.61	1.64	0.01	0.012	0.003	0.09	1.0	0.41
e	"Cerulean Blue"	1.52	0.32	0.25	0.06	0.26	0.40	0.01	1.0	0.31
f	"Burnt Umber"	0.74	1.54	2.10	0.09	0.09	0.004	0.09	9.3	0.90
g	"Cadmium Red"	0.14	1.08	1.68	0.77	0.015	0.018	0.02	1.0	0.63
h	"Brilliant Orange"	0.13	0.81	3.45	0.005	0.009	0.007	0.01	1.0	0.14
i	"Hansa Yellow"	0.06	0.21	1.78	0.50	0.88	0.009	0.06	1.0	0.08
j	"Phthalo Green"	1.55	0.47	0.63	0.01	0.05	0.035	0.02	1.0	0.12
k	"French Ultramarine"	0.86	0.86	0.06	0.005	0.005	0.09	0.01	3.1	0.91
l	"Interference Lilac"	0.08	0.11	0.07	1.25	0.42	1.43	0.06	1.0	0.08

measured pigment data

Every pigment has absorbancy and scattering coefficients defined for the red, green, and blue wavelengths. With this data, along with the thickness of the layer, we can determine the reflectivity and transmittance of each layer of pigment. The KM formulas used by this process can be seen in [2]. The specific implementation is thanks to Julie Dorsey, as used in a project on metallic patinas [4].

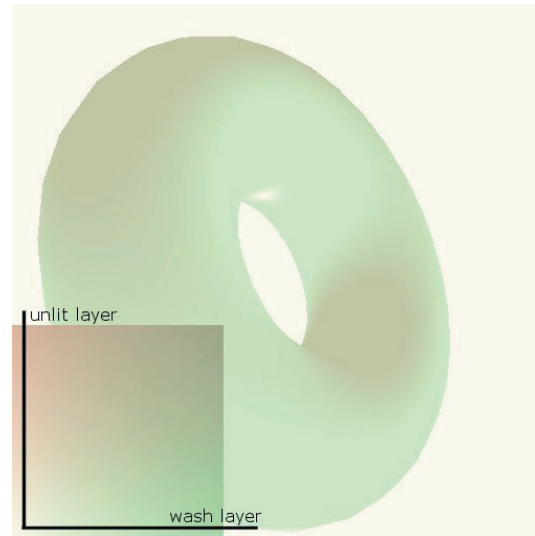
These values can be then overlapped on top of a sheet of paper (with its own R and T values) to give final data for all layers combined. This is then turned into the final pixel color.

Hardware Implementation

The pigment shading model was developed in nVidia's CG high-level shader language, with a demo application written in C++ and OpenGL.

The KM model, though relatively simple, is too slow and computationally complex to calculate on hardware. Therefore, using the coefficient data from the wash and unlit pigments, a 2-dimensional function of the thicknesses of both these pigments is pre-calculated and encoded into a texture.

This procedure obviously induces some limitations to the range of rendering possibilities: objects can have only one shade of color over their entire surface (or if there is more than one surface, the edge will be sharp and unnatural). Furthermore it limits texture-mapping on these objects to pigment thickness maps.



final pixel color as a function of wash and unlit thickness

The hardware shader then calculates the thickness of the wash and unlit layer, either per-vertex or per-pixel (both versions are available depending on hardware) and uses this as texture coordinates to lookup the texture. Currently the texture yields a final pixel color, but this could be easily modified to yield an R and T value to be transformed by the shader into a pixel color.

Contribution

This watercolor rendering method, as a merely interpretive measure with no attempt at physical accuracy, does not really add any new knowledge to the behavior of watercolor ink. My work in particular, then, as the creative re-iteration of another project, in this case does not advance the current understanding of watercolor rendering.

However, in making a real-time implementation I have expanded the possible range of uses of this method to include interactive watercolor rendering previews and computer games, and shown that this method is ripe for optimization. Furthermore, as just a foray into the world of real-time NPR rendering, this implementation shows the surface of the promise that hardware shaders show in expressive NPR rendering at interactive speeds.

Future Work

Obviously future work remains to be done on this project. In order to bring the real-time implementation up to the level of the offline implementation, a pre-computed brush-stroke texture layer needs to be added to the calculation. This certainly need not be calculated per frame as Lum and Ma do, and would provide much-needed hints to the geometry of the shaded object.

Another direction to take this would be to possibly incorporate the work of Hugue Hoppe et. al. in real-time hatching and tonal art maps [3] to generate brush stroke or wash patterns on the fly.

Additionally, pigment application is never perfectly even, especially with the watercolor medium. Some allowance for paint unevenness should be allowed.

Self-shadowing is also an extremely important cue for interpreting geometry, and is notably missing from my implementation.

References

- [1] LUM., E., AND MA, K. Non-Photorealistic Rendering using Watercolor Inspired Textures and Illumination. *Ninth Pacific Conference on Computer Graphics and Applications, October 2001*
- [2] CURTIS, C., ANDERSON, S., FLEISCHER, K., AND SALESIN, D. Computer-Generated Watercolor. *SIGGRAPH 1997 Conference Proceedings, August 1997.*
- [3] PRAUN, E., HOPPE, H. Real-time Hatching. *SIGGRAPH 2001.*
- [4] DORSEY, J., HANRAHAN, P. Modeling and Rendering of Metallic Patinas. *SIGGRAPH 1996.*