Non-photo realistic Volume Rendering in Watercolor

by Anna Sokol

Abstract

In this project, is presented a watercolor inspired method for the rendering of surfaces. This approach mimics the watercolor process by building up an illuminated scene through the composting of several layers of semi-transparent paint. The key steps consist of creating textures for each layer using Perlin Noise, and then calculating the layer thickness distribution using an inverted subtractive lighting model. The resulting watercolor-style images have color coherence that results from the mixing of a limited palette of paints. The new lighting model helps to better convey large shape changes, while texture orientations give hints of less dominant features. The rendered images therefore possess perceptual clues to more effectively communicate shape and texture information.[1]

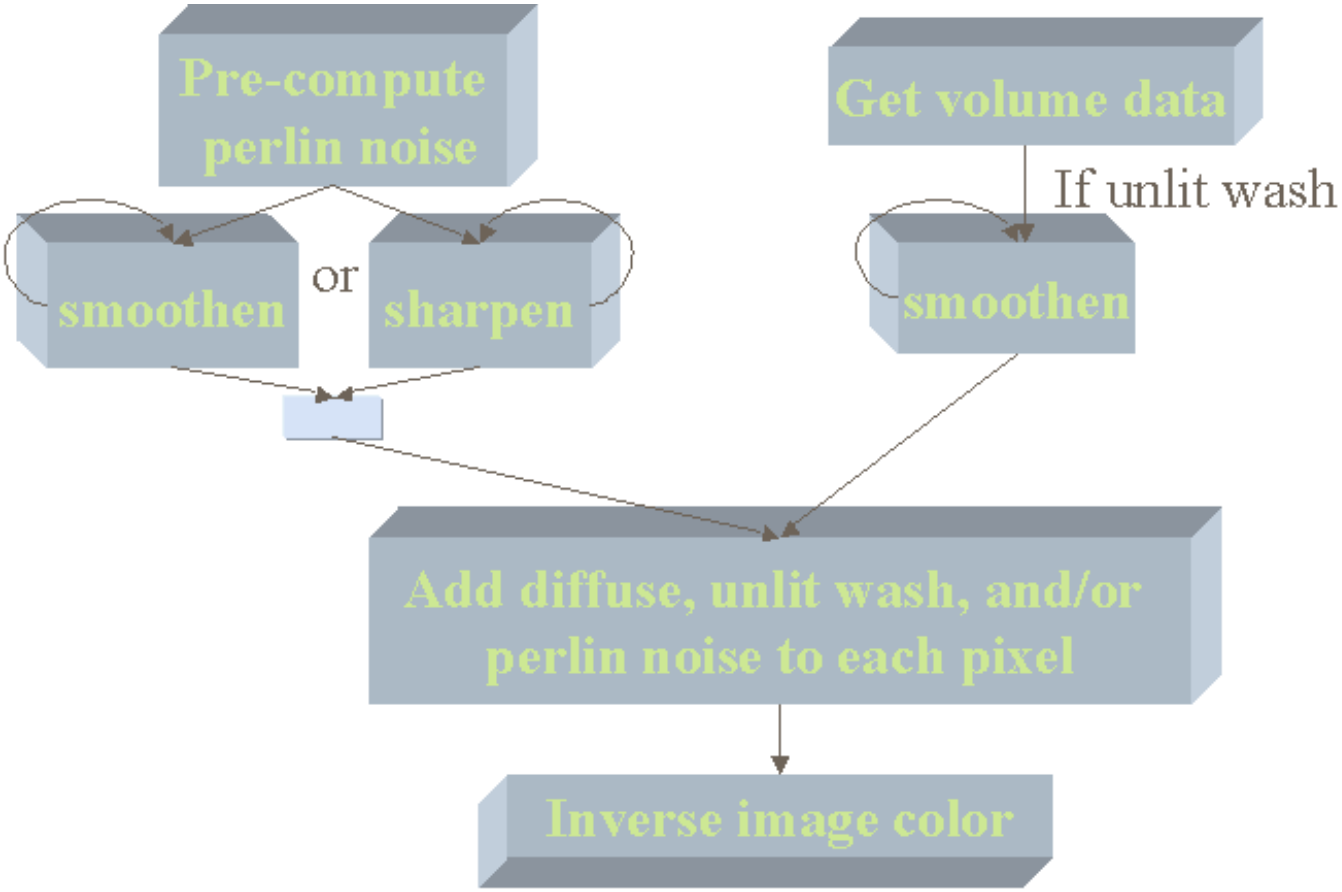
Introduction

Watercolor is created by building up an illuminated scene through the compositing of several layers of semi-transparent paint. In watercolor, color is added to white paper. When a new layer of color is added the underlying layers are still semi-visible.

Related Work

In the Lim and Ma paper, watercolor is generated by using LIC of perlin noise to composite several layers of semi-transparent paint.[1] In the Computer Generated Watercolor paper, watercolor is based on an ordered set of translucent glazes, which are created independently using a shallow-water fluid simulation. They used the Kubelka-Munk compositing model for simulating the optical effect of the superimposed glazes. They demonstrate how computer-generated watercolor can be used in three different applications: as part of an interactive watercolor paint system, as a method for automatic image "watercolorization", and as a mechanism for non-photorealistic rendering of three-dimensional scenes.[2]

Pipeline



In this pipeline, first the volume data is gathered. Every time a volume is rendered, perlin noise is computed (if perlinCheck is selected). If the perlin noise is computed it is smoothed or sharpened, based on which was selected and on the amount of perlin distortion selected. If unlitCheck is selected then the volume is smoothed based on how many unlit wash distortions are selected. The volume is then rendered with diffuse, unlit wash, and/or perlin noise color added to each pixel, based on which of the three were selected. Finally, the colors of the resulting image are inversed.

Technique

The diffuse thickness layer is calculated for every pixel in the volume by

$$\text{diffuse thickness} = (1 - (V \cdot R)^n k_{specular}) k_{diffuse}$$ [1]. The unlit wash thickness layer is calculated by

$$\text{unlit layer thickness} = (1 - (L \cdot N)) k_{ambient}$$ [1]. The perlin noise layer is calculated by multiplying the unlit layer thickness by the gain and the diffuse thickness by the bias.

Perlin noise is a texturing primitive you can use to create a very wide variety of natural looking textures. Combining noise into various mathematical expressions produces procedural texture. Unlike traditional texture mapping, procedural texture doesn't require a source texture image. As a result, the bandwidth requirements for transmitting or storing procedural textures are essentially zero. Also, procedural texture can be applied directly onto a three dimensional object. This avoids the

"mapping problem" of traditional texture mapping. Instead of trying to figure out how to wrap a two dimensional texture image around a complex object, you can just dip the object into a soup of procedural texture material. Essentially, the virtual object is carved out of a virtual solid material defined by the procedural texture. For this reason, procedural textures are sometimes called solid texture. [3]

Algorithm

When renderWaterColor is selected the following algorithm is called:

```
If (perlin)
    vol2 = CreatePerlinNoise3D(alpha, beta, harm, wrap);
    For (p=0…perlin_distortions)
       If (perlinType==SMOOTH)
          SmoothVolume(vol2);
       Else SharpVolume(vol2);

If (unlit)
    For (u=0…unlit_distortions)
        SmoothVolume(vol);

RenderVolume(vol);
InverseImageColor();
```

This algorithm first checks to see, if the perlin noise has been selected. If it has, then perlin noise is created for the 3D volume data set, based on the pre-selected perlin alpha, beta, harm, and wrap. Then for the amount of perlin distortions selected, the perlin noise is either smoothed or sharpened, based on pre-selection. This actually creates interesting brush effects. If the perlin noise is smoothed, it is like that layer has been painted with a big brush. The brush seems to increase in size as the amount of distortion increase. On the other hand, if the perlin noise is sharpened, it is like that layer has been painted with a thin brush. The brush sees to decrease in size as the amount of distortion increases.

The second thing this algorithm does is to check if the unlit wash has been selected. If it has, then for the amount of unlit distortions selected, the original volume is smoothed. The more smoothing the more feathered the edges of the volume become. Then the this volume is rendered. The colors, of the resulting 2D image from the rendering, are then inversed.

$$\text{diffuse thickness} = (1 - (V \cdot R)^n k_{specular}) k_{diffuse}$$ [1]

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

```
if (diffuse)
    color = transFunc[pixval]*diffThick*(1-de);
if (unlit)
    color += transFunc[pixval]*unlitThick*(1-am);
```

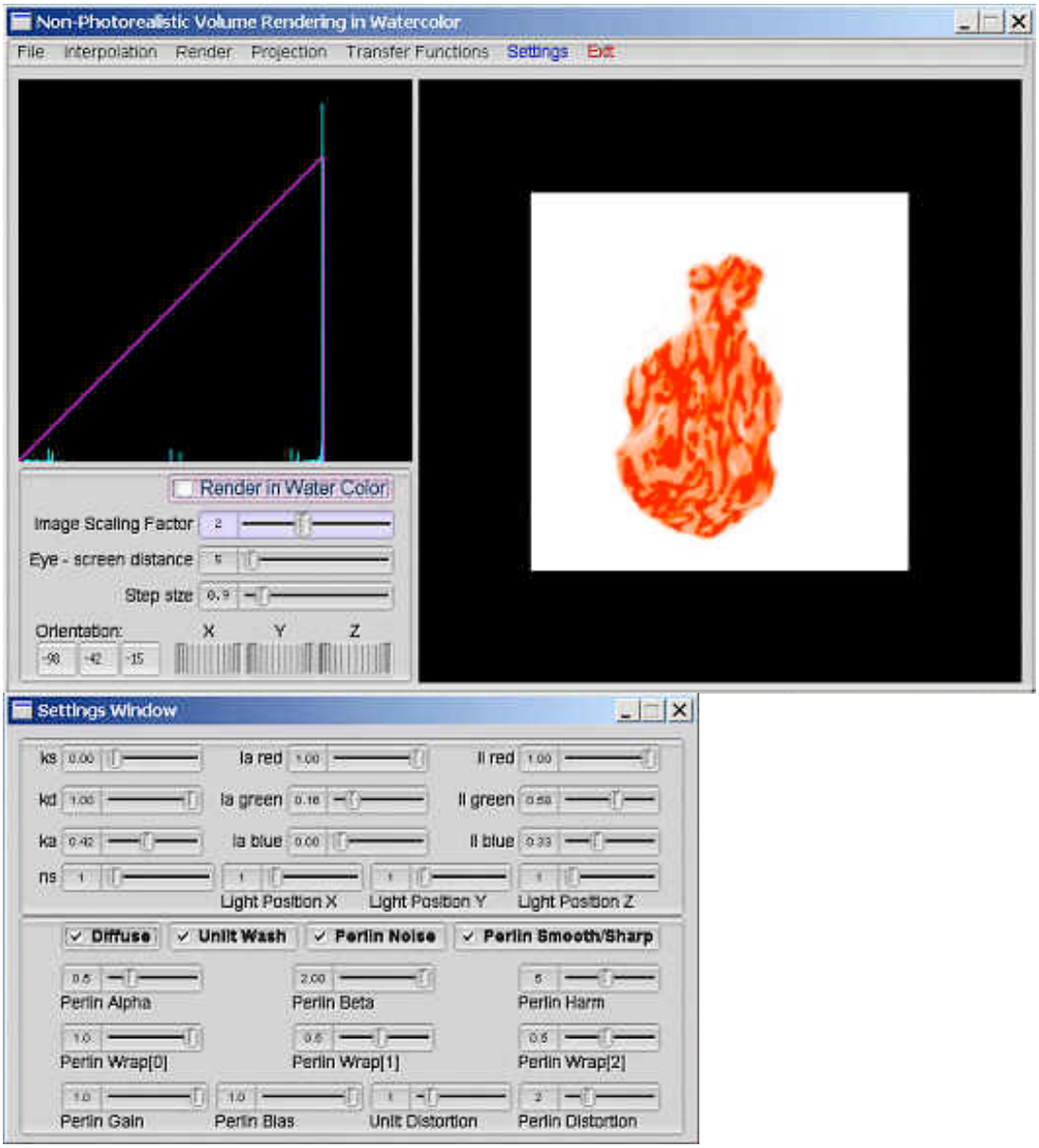
```
if (perlin)
   color += transFunc[(vol2.data[pos])]*(unlitThick*(1-am)*gain + diffThick*(1-
de)*bias);
```

Inside the render volume function, the shading function is called for every pixel. The shading color of each pixel is calculated based on the above algorithm. First the diffuse thickness and unlit layer thickness are calculated on the fly, then the resulting amounts are used to calculated the shaded color of that pixel. First, if diffuse has been selected, then the original pixel value of that position in the volume is transformed by the transfer functions and multiplied by the diffuse thickness and the inverse of the diffuse (Il) color selections. Second, if unlit has been selected, then the original pixel value of that position in the volume is transformed by the transfer functions and multiplied by the unlit layer thickness and the inverse of the ambient (Ia) color selections and then this color is added to the already accumulated color. Finally, if perlin has been selected, then the pixel value of the perlin noise volume in that current position is transformed by the transfer functions and multiplied by the unlit layer thickness, the inverse of the ambient (Ia) color selections, and the gain and then added to the diffuse thickness, the inverse of the diffuse (Il) color selections, and the bias, and then that resulting color is added to the accumulated color.

```
dd = 255/volparam;
dx = 1/vol2.nx;
dy = 1/vol2.ny;
dz = 1/vol2.nz;
for (i=0...nz)
    for (j=0...ny)
        for (k=0...nx)
            po[3] = {k*dx, j*dy, i*dz};
           r=abs(PerlinNoise3D(po[0]*pwrap[0],po[1]*pwrap[1],
                                    po[2]*pwrap[2], palpha, pbeta,
pharm));
            val = r*volparam;
            vol2.data[i*vol2.nx*vol2.ny + j*vol2.nx + k] = (3-
val)*dd;[4]
```

Implementation

This project has been implemented in openGL and C++. The GUI used here is FLTK.



Results

Diffuse only:

All images have been rendered twice their original size. The paint appears light because it is semi-transparent.

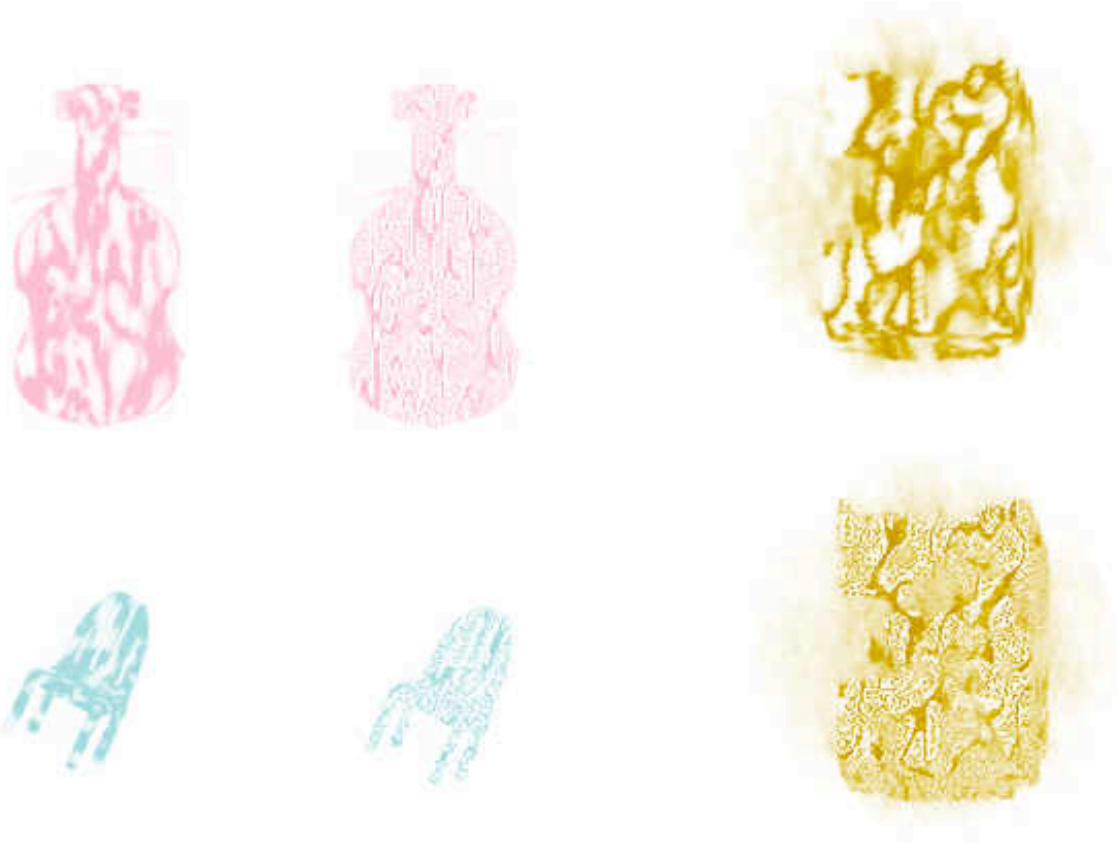
Unlit wash only:

The volume smoothing distortion in each of these images is four. This makes the edges of the volume appear more feathered. The paint appears light because it is semi-transparent

Diffuse and unlit wash:

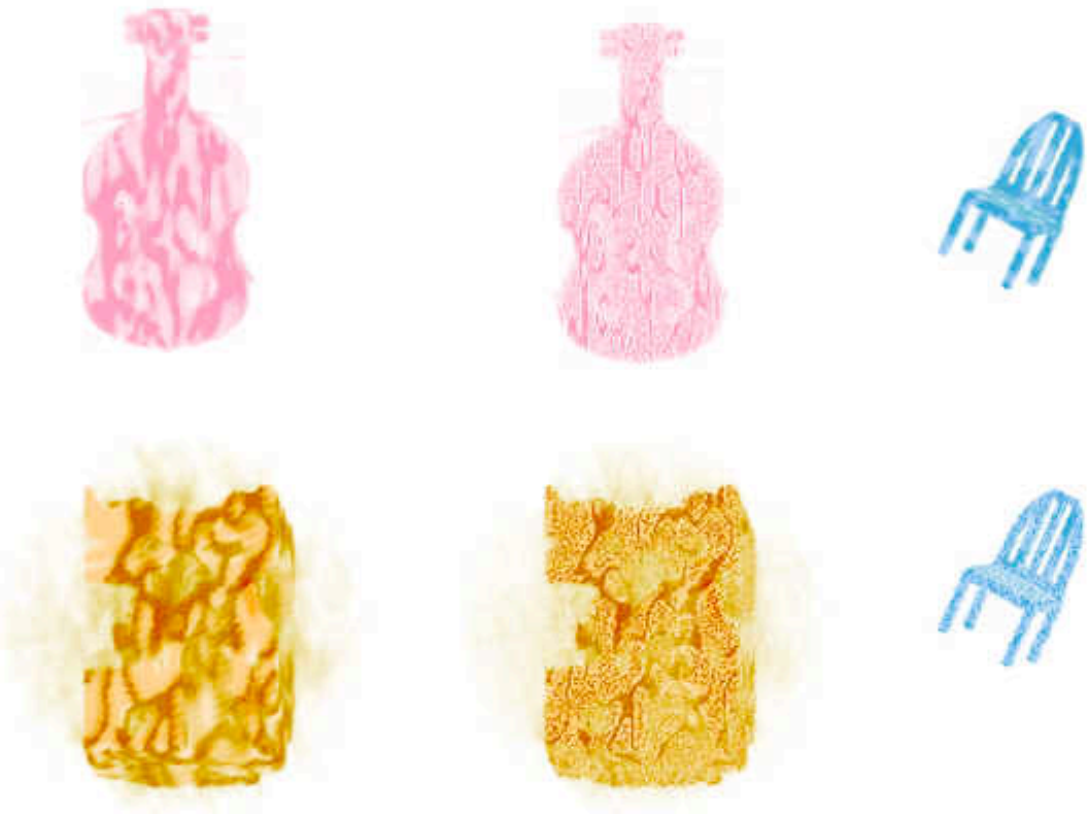
Here the paint appears darker because 2 layers of paint have been used.

Perlin noise only:



The first cello, chair, and engine are rendered with a smoothing distortion of four. The second cello, chair, and engine are rendered with a sharpening distortion of four.

Diffuse and perlin:

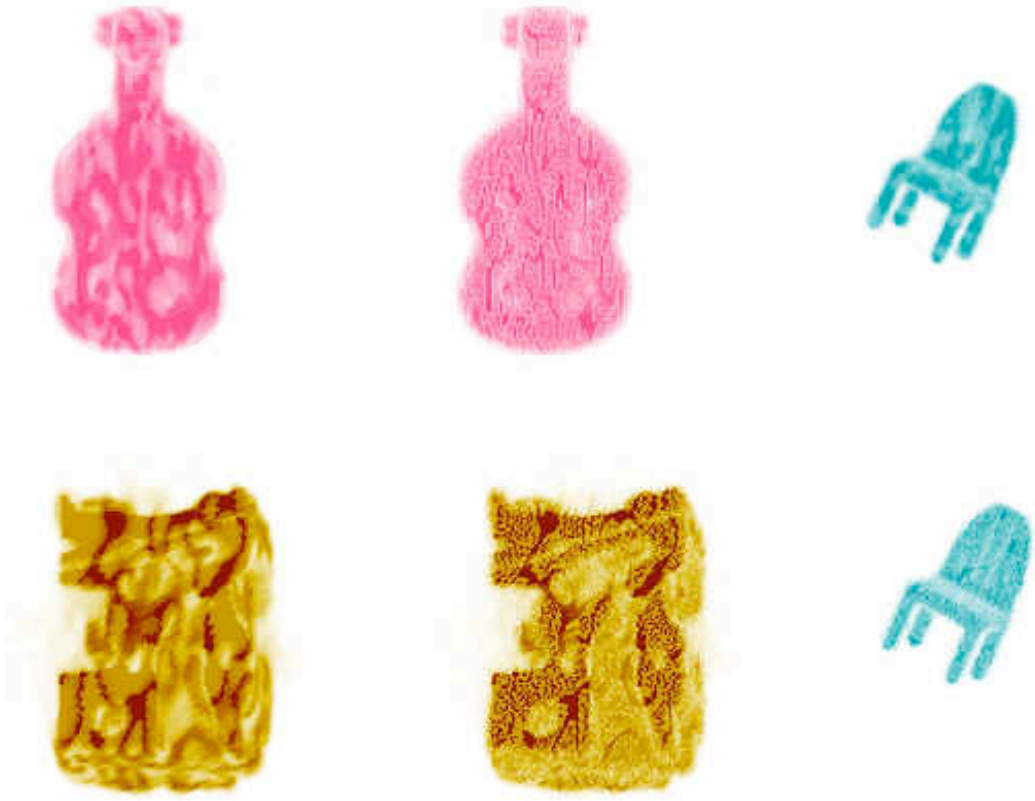


Here the paint appears more solidified because two layers have been used

Unlit and perlin:

Here the paint appears more solidified because two layers of paint have been used.

Diffuse, unlit, and perlin:



Here all three layers of paint have been used and the volumes appear very solidified.

References

- [1]Lum and Ma, “Non-photorealistic rendering using watercolor inspired textures and illumination” Pacific Graphics 2001
- [2]Curtis, Anderson, Seims, Fleischer,Salesin, “Computer Generated Watercolor” SIGGRAPH 1997
- [3]http://www.mrl.nyu.edu/~perlin/
- [4]http://www.utah.edu/~jmk/