Eurographics Symposium on Rendering 2013
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

Volume 32 (2013), Number 4

A Simple Method of Interactive Crayon Drawing Using Histogram Specification

paper1066

Abstract

We demonstrate a method for interactive non-photorealistic rendering of wax crayon drawing. Our method requires no artist input and generates plausible results at high framerates without any repeating patterns. Our method is based on empirical observations of the medium and uses sampled data from scans of the target paper with coloring applied. Minimal preprocessing that consists of basic image manipulations such as histogram matching is applied to the input samples in order to generate a non-repeating grain texture. At runtime, this grain texture is used to inform the alpha transparency of fragments written into a buffer using line rendering. We demonstrate believable effects with our method such as color mixing and drawing with various amounts of pressure. Our system is designed to be simple to implement and to operate efficiently on a low power device such a low end laptop or a tablet.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—

1. Introduction

We have designed a simple drawing and painting application which allows the user to select and use several different drawing tools on a number of backgrounds, and change the color. We have the goal of making this program work on low-powered devices like laptops and tablets. One of the materials we added to our program was a crayon rendering effect. Based on our application's requirements, we needed something that allowed real-time drawing, and could be added easily to the application. We also wanted this need for simplicity to be balanced with a realistic appearance when the user draws and colors repeatedly.

In this paper we demonstrate a method that meets our requirements of simplicity and speed. Our method uses histogram specification to mimic the distribution of wax on the material using scans of coloring on physical samples of the material. This gives us a realistic appearance, in addition to speed. This also means that unlike some other methods, we do not have to tweak parameters to get the desired appearance. An example of our results can be seen in Figure 1.

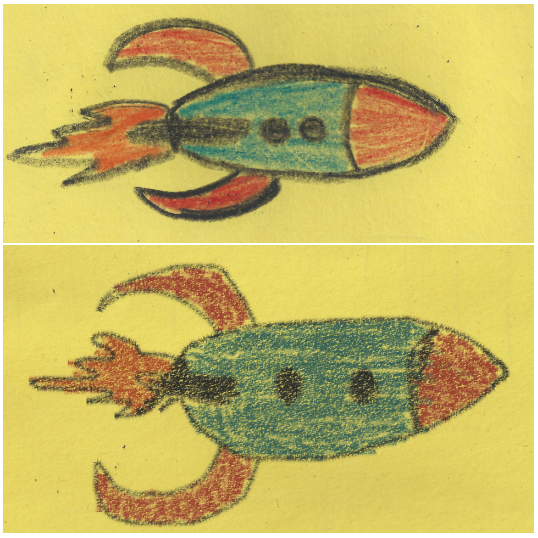

Figure 1: *Top: a drawing of a rocket with wax crayon on construction paper. Bottom: A similar drawing in our system with the same material and colors.*

2. Related Work

2.1. Image Processing

Our approach makes extensive use of common image processing techniques such as histogram specification, convo-

lution, and filtering. A overview of these techniques can be found in Gonzalez and Woods [GW06].

2.2. Non-Photorealistic Rendering

Non-photorealistic rendering (NPR) has two major schools of thought. One is concerned with extracting and presenting simplified, relevant information to a user, for instance, in the style of a technical drawing. The other is concerned with replicating the styles painterly or other natural artistic media in computer generated rendering. The latter type is the form with which we are concerned. A general overview of this subset of NPR for real-time rendering can be found in Akenine-Möller, et al. [AMHH08].

Our application requires real-time drawing, and rendering of the NPR effect from a fixed 2D viewpoint, rather than implementing the NPR effect as a post processing effect on a rendered 3D frame. This fact combined with the fact that we have direct access to the stroke data helps us simplify our approach and focus our review of prior work.

The effect of drawing on paper with a crayon, a pencil, or some other implement can be modeled using a height field of the paper. This height field is either scanned in from real samples, or is procedurally generated using procedural texturing methods. The amount of coloration imparted to the surface is based proportional to the height and is determined by some adhesion function that determines how much pigmentation the surface can receive. Such an approach is common in the literature [SB99] [MT02] [SG04] and is the basis of our approach.

Many commercially released applications including popular computer games [Pur09] get a lot of mileage out of a simple stamp-based system. One of a number of pre-created brushes is selected as the user draws, perhaps with a rotation to hide repetitions. This method is simple, fast, and produces good looking results.

The thesis by Halstead [Hal04] uses a method that models the tip of the crayon as a polygon where each vertex has pressure distribution coefficients. This allows changing the contact profile of the crayon by simply adjusting some vertex weights. When this polygon is rasterized, linear blending across the polygon gives a pressure coefficient for each pixel that is colored. The paper is modeled as a heightmap. User specified values give the heights accessible by the crayon and maximum wax deposition amounts at low and high pressure. Linear interpolation between these two values based on the simulated pressure of the crayon at each pixel determines the amount of wax deposited at a pixel. The amount of material deposited at each pixel is tracked, and exponential falloff is applied to new strokes as more and more wax is deposited to simulate the lessening of friction causing less wax to be deposited in areas that have already been colored. Colors are stored in a single layer in a subtractive color space. When coloring over a previously colored area, the output color is a simple weighted average of the new color and the previously stored color based on the amount of material deposited.

Rudolf, et al [RMN03] [RMN05] present a physically inspired model for wax crayon rendering. Their method models the paper as a heightmap, and the crayon tip as a height mask. The height mask changes as it interacts with the surface. This allows their method to simulate the crayon tip eroding as the user colors the page and wax being carried from one part of the page to another. They use frictional interaction between the crayon and the paper (including wax that has already been placed) to model wax deposition and smearing. A Kubelka-Monk color model approximates light transmittance and scattering effects for rendering. Their method did not run real-time at the time of their publication, and has steep memory requirements since it stores multiple layers of wax with their own material properties including height, transmittance, color scattering. Nonetheless, their model is notable for its physical plausibility and detail.

3. Overview

For our application, we favored simplicity and visual plausibility over physical accuracy. We wanted something that could be implemented quickly, look credible, and run efficiently enough using OpenGL 2.1 features to consider a port to a mobile device or WebGL. With that in mind, we had a list of visual phenomena we wanted to capture with our approach, and designed our system around those restrictions.

3.1. Goals

We wanted to design a system to simulate drawing on a variety of materials and do it in real time. Drawing should look different based on the material it is drawn on. We were willing to allow some preprocessing, but never wanted to compromise on the speed or require long wait times to add and test a new material in the system. In this system we focused on our effort on crayon drawing.

Crayon drawing on paper has a number of observable properties that we wish to reproduce. First, the manner in which pigment is distributed is dependent on the surface roughness of the material on which the drawing is applied. This means that coloring on different materials should have a different appearance.

Additionally, drawing with more pressure applied to the crayon should cause "darker" lines to be drawn as more pigment is applied to the page. Since our target systems may not have the ability to detect the pressure the user is applying to a mouse button or stylus, we assume that drawing speed is proportional to pressure.

We also want the drawing of new lines to interact in a believable manner with the coloring already on the page. This means that when the user selects different colors, the new colors should interact with the existing colors on the page in

a similar manner to real crayons. When a real piece of paper is colored repeatedly, it picks up less and less coloration each time. The amount of friction between the crayon and the page reduces as more and more wax is deposited on the page. This means that it should be difficult to color over areas that have already been colored in with some other color.

Finally, we want to avoid obvious repeating patterns in the final rendering.

3.2. Approach

A simple collection of bitmap stamps to apply to the screen seems to satisfy most of our criteria. We did actually try this approach initially, but were unsatisfied with the results. Generating good looking stamps that don't obviously repeat is not theoretically difficult, but artistic ineptitude and the desire to be able to add new materials without creating new content led us to explore an approach that could be more automated and software driven.

We create a simplified model that has two inputs; a heightmap of the paper surface the user draws on, and a function that converts heights into an alpha value that determines how much wax gets deposited on a particular pixel the user draws over. We apply this function to the heightmap as a preprocess to create a per-pixel alpha mask or grain texture.

Our heightmap is derived from scans of the material. The mapping function is also based on scans of the material samples, but this time with coloring applied rather than a physical model or a software heuristic. We create a histogram of intensity values from these samples. We know that the intensity distribution of colored areas for a particular material should be similar in the synthetic crayon rendering for that material. We also know that the amount of coloration should be proportional to the height in the heightmap. Therefore we use histogram specification [GW06] to match the color distributions of the heightmap to the target histogram generated from the samples.

At runtime, the alpha mask is used to modulate the intensity of fragments written into a color buffer over areas where the user draws. Simple blending and shader effects mimic the other effects we require.

Since both the heightmap and the histogram of the wax samples will vary between different materials, we have an automated way of introducing new materials to the system. This allows each material to have a different characteristic as it is colored in.

4. Preprocessing Step

The purpose of the preprocessing step is to take in samples of coloring on the material and output the grain texture that will be used as an alpha mask at runtime.

4.1. Manual Labor

The first requirement of our approach is to obtain samples of the desired paper material. First the material is scanned without any coloring applied.

Then straight lines are drawn onto the material with a wax crayon and the material is scanned again. A black crayon is used during this step to more easily distinguish where darker coloration has been applied. Once the sample coloring is scanned in, portions of the straight lines are manually selected in an image program to be provided to the preprocessing step.

4.2. Initial Preprocessing

First, the scan of the blank paper is converted from a color image into a luminance image. We use this directly as the heightmap. This is somewhat justified; lower areas will be more occluded and receive more shadow in the scan. Assuming that the coloration of the paper is fairly consistent then we can use the luminance map of the scan as a rough heuristic of relative heights.

Next we high-pass filter the heightmap. Due to variations in lighting from the scan there will be low frequency changes in brightness across the material. What we really care about is the high frequency "grain" of the material and not any global lighting or coloring effects. The high-pass filter gives us this data and prevents the average wax deposition from varying drastically from one location to another on the material as the user draws.

4.3. Histogram Matching

For our model, we assume that higher areas of the heightmap will have more contact with the crayon and thus have more coloration. In our system, this means that we want high areas in the heightmap to have high alpha values in the grain map we output.

The heightmap we have at this point meets these criteria, but the intensity distribution does not necessarily have any relation to an alpha map that reflects actual crayon coloring on the material. This means we need to rearrange the distributions of the colors in the heightmap. However, there is an ordering in our heightmap that must be preserved; if the value of some pixel in the input heightmap is greater than the value of some other pixel in the input heightmap, this relationship must also hold in the output heightmap. This is because of our previous statement that high points in the heightmap should accumulate wax easier than lower areas.

Histogram specification [GW06] is a method that takes an input image and a target CDF for the image and maps values from the input image such that the distribution of values in the resulting image resembles the target CDF. Thus we can transform one image to match the color distribution of another.

This method is based on creating and evaluating cumulative distribution functions (CDFs) of the source and target images. A CDF is by definition monotonically increasing, meaning that the operations described in the following paragraphs will cause the ordering of the input intensities to be preserved. Thus, the histogram specification approach preserves the relative orderings of intensity from the heightmap while giving us an intensity distribution that approximates that of real samples of coloring on the material.

4.3.1. Converting Crayon Samples to Alpha Maps

We read in the samples of lines drawn with crayon on the page. We do not use these directly; we convert them to a single channel alpha map. To do this we assume a linear alpha blending between the average paper color and the crayon color.

This gives the vector equation with average paper color $\vec{P}$, crayon color $\vec{C}$, sample color $\vec{S}$, and unknown alpha blending weight α:

$$\alpha * \vec{C} + (1.0 - \alpha) * \vec{P} = \vec{S} \tag{1}$$

Rearranging we get:

$$\alpha * (\vec{C} - \vec{P}) = \vec{S} - \vec{P} \tag{2}$$

Since we used a black crayon, we estimate all the components of $\vec{C}$ as 0.0. This gives:

$$-\alpha * \vec{P} = \vec{S} - \vec{P} \tag{3}$$

Since this is a three-dimensional vector equation, we have three equations for one unknown. This system is overdetermined. We can write the error as:

$$\varepsilon^2 = ((\vec{S} - \vec{P}) + (\alpha * \vec{P}))^2 \tag{4}$$

The squared error ε^2 is minimized with respect to α either through least squares or by setting the derivative with respect to α to zero and solving. This gives us:

$$\alpha = \frac{-\vec{P} \cdot (\vec{S} - \vec{P})}{\vec{P} \cdot \vec{P}} \tag{5}$$

Using this equation, we convert each of the three-channel wax sample images into single-channel alpha maps. An example can be seen in Figure 2.

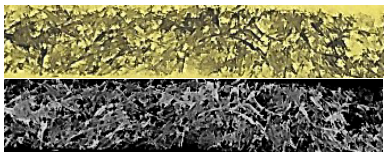

Figure 2: *Top: a sample line drawn in black crayon on our material. Bottom: The alpha texture that this sample is converted to in order to populate the histogram.*

4.3.2. Generating Grain Textures

We take the single channel alpha maps generated in the previous step and create an intensity histogram with the combined contributions of all the samples. Each pixel's contribution is weighted by a gaussian falloff function based on distance to the edge of the line, perpendicular to the drawing direction. This is because of the observation that when a line is drawn with a crayon, the center of the line is drawn the darkest and the edges of the line further away from the main point of contact with the crayon are not drawn as dark. Since we are using a single histogram to represent the entire drawing, we do not want the pixels at the edge of the line to falsely weight our histogram towards zero alpha values. We are concerned primarily with the center of the line drawn, and will mimic the edge-falloff as a shader effect at runtime.

Once the histogram is populated, a normalized CDF is generated from it. This is our target CDF, CDF_t A normalized CDF is also generated for the high pass filtered paper texture. We call this source-image CDF CDF_s. Histogram specification [GW06] is used to take the paper texture which is acting as our height map, convert it into an texture whose color distribution closely matches that of the wax samples for the material.

Given the target CDF of the samples CDF_t and the CDF of the source heightmap CDF_s, we map the intensity of a heightmap pixel i to an output intensity i_{out} as follows:

$$i_{out} = CDF_t^{-1}(CDF_s(i)) \tag{6}$$

A lookup table for each of the 256 possible values of i is generated ahead of time for efficiency.

The nonlinear mapping of the histogram specification causes some low frequency variations to appear in the output grain images for some materials. Therefore we take the optional step of using the high-pass filter again for these materials.

4.4. Allowing for Different Crayon Pressures

We know from observation that drawing with more pressure on the paper will cause the more material to be transferred to the paper and the coloring to be darker. Rather than perform some trick at runtime to modify the alpha value read from the alpha mask based on pressure, we stick to our data-driven approach. We produce a second set of drawing samples on the material, this time with more pressure applied in the drawing. We repeat all of the steps above to generate a grain texture for this second set of samples. This gives us two grain textures: one for low pressure drawing and one for high pressure drawing. At runtime, we can interpolate between them.

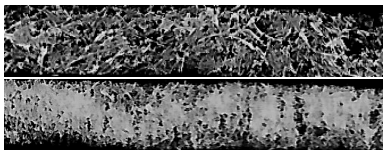

Figure 3: *Top: an alpha mask generated from a sample of a line at low pressure. Bottom: an alpha mask generated from a line drawn with high pressure. The difference in the material transference is quite apparent.*

5. OpenGL Runtime

Our basic approach is to draw a line connecting where the cursor was last frame to where it is on the current frame. This line uses a shader that outputs a color and alpha value that takes into account the values stored from previous crayon strokes, stored as a texture. This line is rendered into a texture using offscreen rendering. Our final image is generated by blending the crayon buffer with the background paper using the alpha values stored in the crayon buffer.

5.1. details

At the beginning of each frame, we estimate the pressure from the cursor speed. The user specifies a speed corresponding to low pressure, and a larger speed corresponding to high pressure. From the current speed and these two thresholds, a normalized floating point value is generated corresponding to the pressure.

We bind the texture that will store the results for the current frame as the current render target and then render a full screen quad containing the results texture from the previous frame. Then we draw a quad in the shape of a line that connects the cursor point from last frame to the cursor point in the current frame. This quad is drawn with a shader that will output a color and alpha value. The shader samples the low-pressure and high-pressure alpha mask grain textures at the coordinates corresponding to the screen location covered by the fragment. The low-pressure value and high-pressure value are interpolated using the pressure value calculated by the program, which is passed to the shader as a uniform value.

This interpolated grain value is used as an initial estimate of the alpha value α_{new} that will be accumulated into the crayon buffer. However, we apply some transformations first. We apply a Gaussian falloff to this value based on the distance to the edge of the line the user drew. This is to make the edges of the line less dark than the center. Second, we scale by $1.0 - \alpha_{prev}$ where α_{prev} is the alpha value stored in the crayon buffer from last frame. This makes sure the previous coloring is taken into account; if 1.0 is defined as the maximum saturation of material at each pixel, this step scales the value from the grain texture to be a portion of the quantity the pixel can still take on. Thus, as more and more material accumulates on a pixel, this scaling value will become smaller and smaller, meaning subsequent colorations will have less impact on that pixel.

The output color is a weighted average of the color of the currently selected crayon $Color_{selected}$ and the color stored in the previous results buffer $Color_{prev}$. This average is given by the equation:

$$\frac{\alpha_{new} * Color_{selected} + \alpha prev * Color_{prev}}{\alpha_{prev} + \alpha_{new}} \quad (7)$$

The output alpha is the sum of the previous alpha α_{prev} and the current alpha α_{new}.

No depth testing or blending stages are applied to this quad, so the framebuffer will store the output of the shader on whatever fragments are covered by the quad. Everywhere else will be the results from previous frames.

To render the final output, we bind the screen as the render target, and render a full screen quad with a shader than blends the crayon result buffer with the paper texture using the stored alpha values for each pixel for linear interpolation.

6. Results

For our implementation, we wrote both the runtime and the preprocessing steps as C++ applications on a 2008 Macbook Pro notebook. This system features a 2.2 gHz Intel Core 2 Duo processor, 4 GB of system memory, and an nVidia GeForce 9600m GT graphics card. Based on 3DMark 06 benchmarks, this system is roughly on par with a 2012 MacBook Air notebook. The graphics portions of our program was written against the OpenGL 2.1 specification.

Aside from a custom C++11 image library (to be released as open source) both the application and the preprocessing steps combined total around 1,000 lines of C++ source code, including code to create a window, handle user input, and output debug information. This highlights the simplicity of our implementation.

For our samples, we scanned 6 by 9 inch rectangles of our material at 600 dpi using an off the shelf HP scanner. These samples were down sampled to 1024x768 for purposes of our demo. We were able to implement cardboard, coloring book paper, construction paper, and tissue paper into our demo.

6.1. Preprocessing

Our preprocessing step took approximately 1.15 seconds on average per dataset for the downsampled materials at the resolution of 1024x768. For the full size scans at 4629x3366 pixels, the average time was 39.18 seconds. While these

times will vary based on how many sample lines are provided for each material, the dominating factor of the preprocessing step is high pass filtering the paper image. This is evidenced by the fact that when we cached the results of this step, the total time taken for *all* materials to process (4 full size, 4 down sampled) totaled 19.467 seconds.

The high pass filter can be made significantly faster; we had a MATLAB script for the high pass filtering step that operated in the frequency domain. This script was able to high pass filter even the full 4629x3366 images in under 10 seconds. Our custom C++ image library is not able to perform this optimization at present and performed the high pass filter by subtracting a Gaussian-blurred version of the image with a large spatial filter. This Gaussian blurring was done using a discrete convolution with a separable filter. However, to generate the data used for our demo we opted to use the C++ version to allow easier automation as a single C++ program for all datasets, with no external dependencies.

6.2. Runtime

Using this approach, our demo averaged between 700 and 1,000 frames per second during coloring. This is expected given the extremely simple operations performed at runtime. This speed, combined with the broad availability of the OpenGL features required, makes this method ideal for implementation in a mobile or WebGL application.

Examining images generated with this approach, we see that the general look of crayon drawing is captured. We see that the choice of material makes a very noticeable difference in rendering. For instance, compare the appearance of coloring on tissue paper in Figure 5 to the appearance of coloring on construction paper 4. We see in figure 7 that changes in pressure give variance in texture to lines drawn. Finally, we see colors blend together realistically in Figure 6.

Figure 4: *Construction paper with yellow crayon layered over top of black crayon. We see that the effect of the yellow crayon is greatly diminished in areas where the coloration from the black crayon was darkest.*

The appearance of the drawing can be ruined if color selection is not done carefully, however. Figure 9 shows us oversaturated colors that do not resemble colors that would

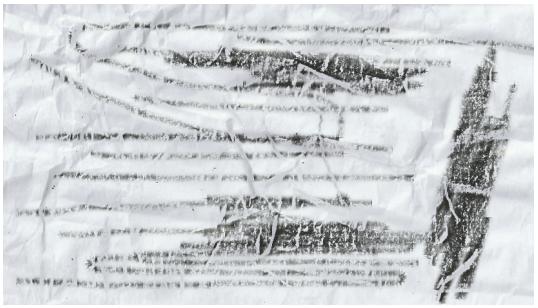

Figure 5: *Tissue paper with black crayon applied. We see that the texture of the crayon strokes is drastically different than the other materials. We also see that areas of shadow in the scan are not colored in at all.*

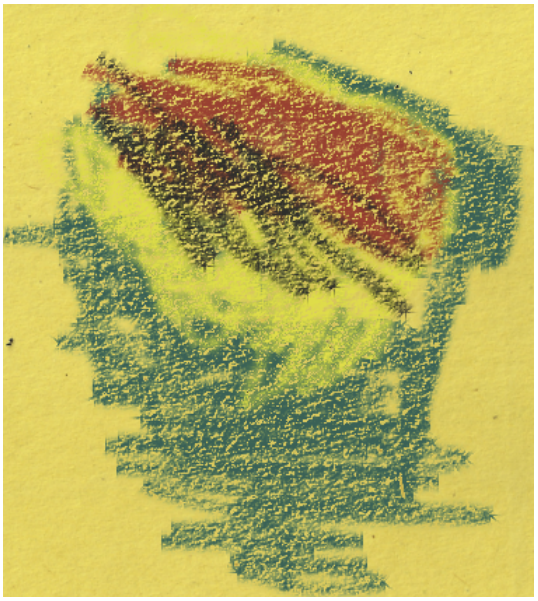

Figure 6: *Regions of high-pressure scribbling with a number of colors. We see believable color interactions; notably, when the blue and the yellow interact, a blue-green hue is produced.*

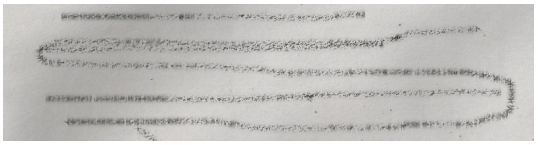

Figure 7: *Varying the pressure (approximated using drawing speed) as the line is drawn changes the density of the wax deposited on the page.*

be produced from real crayons. In keeping with our data-driven approach, we scanned in a sample of the material with a number of colors applied. Regions of this image containing different colors were averaged to get an appropriate color. You can see these colors in Figure 8

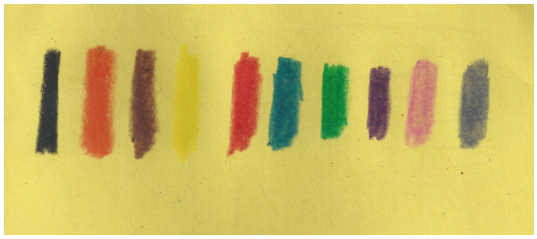

Figure 8: *Construction paper with regions of coloring with different colors. Used to derive RGB crayon colors used in the final program.*

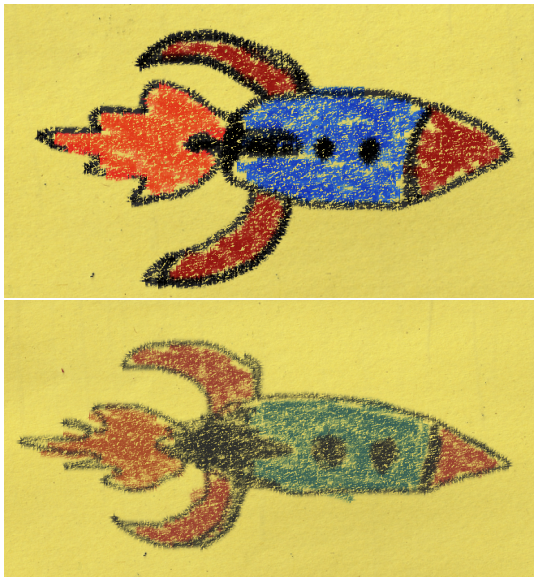

Figure 9: *Top: Lack of care in color selection for the crayons can lead to an artificial, over saturated appearance. Bottom: With sampled colors, the appearance is much more natural.*

However, there are some limitations. There is no obvious directionality to the strokes applied to the page. This could be fixed by a wax smearing effect. Some areas are zeroed out in both grain textures. This causes them to never be colored in regardless of the number of times the user colors over them. A wax smearing system would hide this as well, and cause there to be less "holes" in the final rendering. Additionally, the material properties of translucent layers of wax are not represented or captured. Finally, we see that some on the very crumpled tissue paper sample, some shadowing made its way into the scan and coloring in those areas appears flat. Our system is entirely dependent on the quality of scans that can be obtained.

7. Conclusions and Future Work

We feel that our approach satisfies our design criteria and is ideal for implementation due to its simplicity. We can automatically add plausible looking crayon effects for a number of materials to our system. No parameters need to be tweaked or set except for the speed thresholds associated with low and high pressure drawing. The data needed by the renderer is generated using basic off the shelf image processing techniques, and the runtime works in the subset of OpenGL 2.1 features making it compatible with a number of systems. For future extensions, we would like to add a wax smearing effect similar to the one described in Rudolf, et al [RMN05] to make the appearance more realistic and give a more directional look to the strokes.

We would also like to try using a number of different histograms, such as histograms for line drawings at different speeds, different directions, different rotations of the crayon, and for different distances to the center of the stroke rather than using only two alpha masks to blend between strokes of different pressure.

We find that using the CDFs from actual samples of a material as a transfer function is an interesting approach to generating an NPR effect that requires little manual tweaking and would like to explore using this approach for a broader diversity of materials and artistic media.

References

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. 2

[GW06] GONZALEZ R. C., WOODS R. E.: *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. 2, 3, 4

[Hal04] HALSTEAD H.: Interactive crayon rendering for animation, 2004. doi:http://handle.tamu.edu/1969.1/3155. 2

[MT02] MURAKAMI K., TSURUNO R.: Pastel-like rendering considering the properties of pigments and support medium. In *ACM SIGGRAPH 2002 conference abstracts and applications* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 227–227. URL: http://doi.acm.org/10.1145/1242073.1242240, doi:10.1145/1242073.1242240. 2

[Pur09] PURHO P.: Crayon physics deluxe. http://crayonphysics.com/, 2009. 2

[RMN03] RUDOLF D., MOULD D., NEUFELD E.: Simulating wax crayons. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2003), PG '03, IEEE Computer Society, pp. 163–. URL: http://dl.acm.org/citation.cfm?id=946250.946956. 2

[RMN05] RUDOLF D., MOULD D., NEUFELD E.: A bidirectional deposition model of wax crayons. *COMPUTER GRAPHICS FORUM 24* (2005). 2, 7

[SB99] SOUSA M. C., BUCHANAN J. W.: Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum 18* (1999), 195–208. 2

[SG04] SEHGAL P., GROVER P. S.: A novel approach to cartoon style rendering of an image with an approximated crayon texture. In *Proceedings of the International Conference on Computer Graphics, Imaging and Visualization* (Washington, DC, USA, 2004), CGIV '04, IEEE Computer Society, pp. 82–88. URL: http://dx.doi.org/10.1109/CGIV.2004.8, doi:10.1109/CGIV.2004.8. 2