Using Texture Synthesis for Non-Photorealistic Shading from Paint Samples

Christopher D. Kulla cdk1@cec.wustl.edu
James D. Tucek jdt1@cec.wustl.edu
Reynold J. Bailey rjb1@cs.wustl.edu
Cindy M. Grimm cmg@cs.wustl.edu
Washington University in St. Louis

Abstract

This paper presents several methods for shading meshes from scanned paint samples that represent dark to light transitions. Our techniques emphasize artistic control of brush stroke texture and color. We first demonstrate how the texture of the paint sample can be separated from its color gradient. We demonstrate three methods, two real-time and one off-line, for producing rendered, shaded images from the texture samples. All three techniques use texture synthesis to generate additional paint samples. Finally, we develop metrics for evaluating how well each method achieves our goal in terms of texture similarity, shading correctness and temporal coherence.

1. Introduction

Traditional artists convey shading using brush stroke texture and color variation. They work on a flat canvas, requiring good spatial sense to convey believable lighting. On the other hand, computer graphics programs can easily compute lighting, but cannot make artistic decisions. The algorithms and techniques we present in this paper address this disparity. Our goal is to retain artistic freedom while leveraging the computer's processing power to shade complex meshes.

In our system, an artist provides an example of a shading change from dark to light as an image strip. This paint sample can either be scanned in or created with a 2D paint program. We then apply this user-defined shading style to a mesh, making it appear to be painted with the same technique (see Figure 1).

A typical sample such as Figure 4(a) has two distinct properties that vary with shading: color and texture. Color transitions are typically smooth, albeit non-linear, making them easy to model. Extracting texture change is harder because texture transitions are coarser than lighting changes and because replicating the texture seamlessly is non-trivial. We rely on texture synthesis [3] to address this problem.

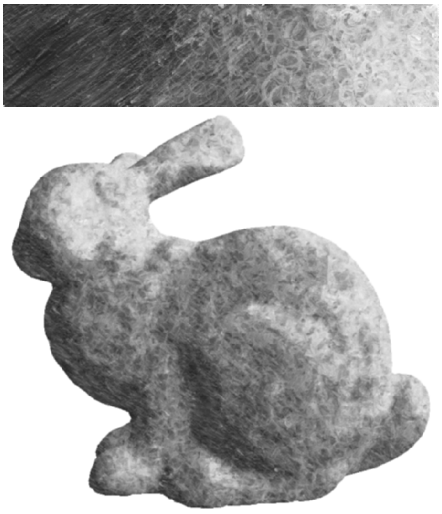


Figure 1: A typical paint sample, scanned by the user, used to shade a mesh.

We discuss previous work in section 2. In section 3 we show how to separate color transition from texture transition in paint samples. In section 4 we describe our three rendering methods to apply paint samples to a lit mesh. Section 5 introduces metrics that qualitatively capture common texture distortions, shading errors and temporal coherence in animation. Section 6 compares the quality of each rendering method using these metrics. We discuss future directions of our work in section 7.

2. Previous Work

Non-photorealistic shading is a well studied problem. Technical illustration shading [5] introduced the use of warm-cool color blends to enhance the perception of shape and orientation. The lit sphere approach [15] extracts artistic shading models from actual paintings. Unfortunately, much of the original brush texture is lost as the shading gradient is captured. Several papers have addressed the technique of hatching [14, 10] in which lighting is conveyed by vary-

Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG'03)
0-7695-2028-6/03 $17.00 © 2003 IEEE

ing stroke densities and orientations. Other rendering styles that rely purely on texture change include charcoal rendering [11], and half-toning [4]. All of these techniques are very stylized and can be captured procedurally or with minimal user input.

Stroke based techniques such as the WYSIWYG NPR system [8] attach paint strokes on the surface of the object. These strokes convey fixed features of the model or move over the surface in response to lighting changes. The idea of attaching paint strokes to the model is also used in painterly rendering [12]. This approach completely automates the painting process. The system also addresses frame-to-frame coherence by reusing strokes throughout the animation. Finally, Webb et al. [16] convey both texture and color change with shading by using the lapped texture method [13] to place several levels of texture onto a mesh and blend between them with 3D texture hardware.

We present an alternative approach to non-photorealistic shading. When artists paint an image in real life, they often replicate identical brush strokes to completely cover the canvas. We model this process using texture synthesis. We were particularly inspired by the texture transfer algorithm [3] which is able to render an image in the texture of a provided sample.

3. Paint Sample Processing

A scanned paint sample has two distinct properties: texture and color. We call the global color change across the sample the color trajectory, as it defines a path through color space. We assume that our paint sample is stored in a wide horizontal 2D image and that the shading changes from left to right (Figure 4(a)). Plotting the colors of the paint sample in RGB space reveals the rough shape of its non-linear color trajectory (Figure 5).

To extract a smooth color trajectory from a given sample, we simply average the colors of each pixel column of the sample image. This effectively filters the 2D sample into a 1D image strip that represents a path through color space. Unfortunately, the result contains a fair amount of streaking due to local texture variations (Figure 4(b)). We run the following recursive algorithm on the trajectory's set of RGB points in order to sort the colors into a continuous gradient.

We seed the algorithm with the two endpoints of the unsorted trajectory. Given two sample colors A and B in RGB space, we let M be their midpoint. We search for the sample point C closest to M, but contained in the sphere of diameter AB (see Figure 2). If such a point is found, the algorithm runs recursively on A and C and on C and B. If no such point is found, A and B are close enough to be considered neighbors and the recursion stops by adding A and B into a linked list representing the sorted color trajectory. Since the algorithm is sensitive to noise, an initial low pass filtering step can be added if necessary. Figure 4(c) shows how the output of the algorithm has effectively removed the streaking effect while maintaining the shape of the color trajectory we wanted to extract.

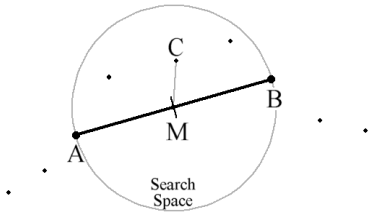


Figure 2: Finding the sample point halfway between A and B.

Texture change can be viewed as a local modulation of the color trajectory. We represent it separately by subtracting the color trajectory from each pixel column of the original paint sample. We can then add an arbitrary color trajectory back into this texture difference image to obtain a sample with different colors but similar texture. Figure 6 shows a red-to-yellow sample being modified into a much more creative color blend by a user-specified path through color space. The approach is not perfect but maintains most stroke features. This technique allows an artist to expand his palette from relatively few paint samples.

4. Rendering Methods

4.1. Image Based Texture Synthesis

This approach was inspired by the image quilting and texture transfer algorithm [3]. In two raster scan order passes, small blocks of the sample texture are cut and pasted, then "stitched" together to minimize visual discontinuity. Texture transfer is achieved by adding constraints to the initial block picking stage.

We use a lit grayscale image as a guide for the synthesis. We run image quilting with the constraint that blocks are only chosen from a narrow vertical strip around the ideal shading level in the paint sample. This added constraint introduces a very noticeable block structure because shading changes can occur on a much smaller scale than the block size. Effros and Freeman use additional passes with decreasing block sizes to solve this problem, but our knowledge of the data enables us to solve this problem more efficiently. We simply run the synthesis on the extracted texture difference image of the paint sample (see section 3), and add the color gradient back in on a per-pixel basis to create the final image.

Creating animations with this method by naively resynthesizing each frame from scratch produces a shower door

effect [7]. To improve temporal coherence we add an additional constraint: each block must match the previous frame as much as possible (computed as a squared pixel difference error). For small lighting or camera movements, this added constraint works very well. However, for paint samples that exhibit drastic texture variations this constraint makes it impossible for the synthesis to find suitable blocks after a few frames. In this case we rely on blending to improve coherence. We synthesize an entirely new set of texture every nth frame and blend texture values between these keyframes while recomputing the shading at every frame.

Rendering takes between 20 seconds to a minute depending on image resolution. The two following sections describe alternatives that run in real-time on commodity graphics hardware.

4.2. View Aligned 3D Texture Projection

Texture synthesis is performed as a preprocessing step only. We divide the input paint sample into 8 regions of roughly constant shade level. We synthesize larger versions of each section with image quilting. We found that generating 8 levels was adequate for the particular size of our paint samples given that this is about how often the texture changes. We experimented with generating more levels and with trying to keep strokes coherent from level to level, but observed no substantial gain.

Our implementation runs on a GeForce 4 class graphics card with each level set to 512×512 pixels. In order to keep texture information separate from the color gradient, we subtract the average color of each section and only synthesize a texture difference image. We store this difference image in a regular bitmap by mapping the interval $[-1, 1]$ linearly to $[0, 1]$. Additionally, we guarantee that the texture be tileable using a simple masking technique [2].

We create a 3D texture from each of the synthesized levels by stacking them in order of increasing shade level [16]. A simple pixel shader is used to access the 3D texture, expand the value back to the interval $[-1, 1]$ and add in a color gradient indexed by lighting value. We index the 3D texture using the screen coordinates of the pixel for s and t, and the lighting value for r, the depth texture coordinate. Stroke density can be adjusted by scaling s and t. Since the texture is tileable, no seams are visible when the texture repeats over the image.

To avoid the impression that the texture is fixed to the screen and that the mesh is "sliding" through it, we keep track of an offset in s and t that we adjust when moving the model. We increment this offset by the average screen space displacement of the vertices most directly facing the camera. This gives the illusion that the texture follows the movement of the object, at least for the polygons that occupy most of the screen space. It is impossible to perfectly move the texture along with the mesh since it is attached to the view plane, but this approximation improves coherence.

4.3. View Dependent Interpolation

The basic idea of this method is to assign specific textures to the important vies of the model and perform blending between these textures for all other views. Determining which views are important is left to the discretion of the user. The only restriction we impose is that every face in the mesh of the model must appear in at least one of these views. This is necessary to ensure that there are no gaps in the resulting image.

To create the texture maps we chose a small number of views (typically 12-15) which surround the object. We center the object in the view then use projection to generate texture map coordinates. We create an alpha mask using the dot product of the viewing direction and the face normals. We then use texture synthesis to create 3D textures, as was described in section 4.2 and create a set of projected 3D texture maps for each view for interpolation. We address self-occlusion with a scan-line algorithm as detailed in [6].

5. Metrics

In this section we outline our choice of metrics. Our error metric has three components: texture fidelity, shading error, and frame to frame coherency.

We develop a metric that is capable of measuring the types of texture distortion we expect to be present. These distortions can be categorized as rotation, stretch or shearing effects, and discontinuities from poor texture sampling.

The image similarity measures we use are common building blocks in image database retrieval algorithms. The first measure is the difference in the color histograms. Next, we filter the image to locate edges in the horizontal, vertical, and diagonal directions. The second measure is the difference between the edge image histograms. Together, these two measures capture the distribution of color and edge directions within the image. To compare two pixels we first find the $s \times s$ block surrounding the pixel, then build the histograms using that block. We then measure the Euclidean distance between each pair of histograms. To compare a pixel to the source texture we find the best pixel match. To speed up this process, we pre-process the data and store it in a $k - d$ tree. This allows us to find the k nearest pixels in $O(\log^3 n)$ time [1]. To check that this metric captures texture distortion we evaluated it on three test cases: rotation, scale, and synthesis distortion (image quilting algorithm with varying block sizes). The results confirm the validity of the metric, as detailed in [9].

The shading error metric calculates how close the texture at a pixel is to the desired texture for that shade value. We first find the k pixels in the source texture that are the closest to the test pixel, using the metric outlined above. We then average the shade values corresponding to those k source pixels and compare with the real shade value. By using k matched pixels ($k \approx$ 3-5) instead of a single pixel we get a better average shade measure, since texture can be fairly similar across a range of shade values.

We measure frame-to-frame coherence in image space by comparing the histogram difference between the same pixel location in frame i and frame $i + 1$.

6. Results

Each of the rendering techniques presented in this paper has its distinct set of advantages and drawbacks. The image based texture synthesis method is the best for individual frame quality, but takes a long time to render. The view aligned 3D texture method is very attractive because it can almost match the quality of the image based technique, but runs in real time. The hardware does, however, introduce error when interpolating across levels in the 3D texture. The view dependent technique also runs in real time, but keeps the texture attached to the object's surface.

	Similarity	Shading	Temporal
Texture Synth.	0.07539	0.00536	0.00297
3D Texture	0.08048	0.00521	0.00016
View dep.	0.10197	0.00582	0.00547

Figure 3: Evaluating each rendering method with our metrics on the green-yellow texture of Figure 7.

We use the metrics outlined in section 5 to compare our renderings (Figure 3). Our metric shows that the texture synthesis provides the greatest amount of texture fidelity. All methods capture shading with the same amount of error, which is the most important result since our goal is to convey shading. Temporal coherence was measured in image space. In this context, 3D texturing works best because the texture is only translated from one frame to the next, whereas texture synthesis must do blending to provide coherence. The view dependent method, while coherent in object space, is not at all coherent when measured in image space as the texture may be distorted by the curvature of the mesh.

7. Conclusion and Future Work

We have demonstrated three rendering algorithms based on texture synthesis for shading a mesh using a provided paint sample. One possible direction for future work is to constrain the synthesis to capture silhouettes in styles provided by the user.

References

[1] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 271–280, 1993.

[2] P. Bourke. Tiling textures on the plane (part 2) http://astronomy.swin.edu.au/ pbourke/texture/tiling2/.

[3] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, pages 341–346, 2001.

[4] B. Freudenberg, M. Masuch, and T. Strothotte. Real-time halftoning: a primitive for non-photorealistic shading. In *Proceedings of EUROGRAPHICS 2002*, pages 227–232, 2002.

[5] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of SIGGRAPH 98*, pages 447–452, 1998.

[6] C. Grimm. Wucse-2003-53: Painting lighting and viewing effects. Technical report, Washington University in St. Louis, 2003.

[7] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *NPAR 2000*, pages 7–12, 2000.

[8] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. Wysiwyg npr: drawing strokes directly on 3d models. In *Proceedings of SIGGRAPH 2002*, pages 755–762, 2002.

[9] C. Kulla, J. Tuceck, R. Bailey, and C. Grimm. Wucse-2003-54: Using texture synthesis for non-photorealistic shading from paint samples. Technical report, Washington University in St. Louis, 2003.

[10] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR 2000*, pages 13–20, 2000.

[11] A. Majumder and M. Gopi. Hardware accelerated real time charcoal rendering. In *NPAR 2002*, pages 59–66, 2002.

[12] B. J. Meier. Painterly rendering for animation. In *Proceedings of SIGGRAPH 96*, pages 477–484, 1996.

[13] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Proceedings of SIGGRAPH 2000*, pages 465–470, 2000.

[14] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proceedings of SIGGRAPH 2001*, page 581, 2001.

[15] P.-P. Sloan, W. Martin, A. Gooch, and B. Gooch. The lit sphere: A model for capturing npr shading from art. In *GI 2001*, pages 143–150, June 2001.

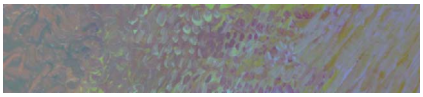
[16] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. Fine tone control in hardware hatching. In *NPAR 2002*, pages 53–ff, 2002.

0-7695-2028-6/03 $17.00 © 2003 **IEEE**

(a) Red to yellow paint sample

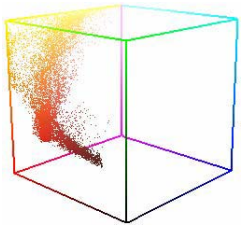
(b) Color gradient before sorting

(c) Color gradient after sorting

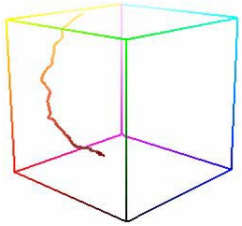


(d) Extracted texture

Figure 4: Extracting color gradient and texture from a typical paint sample.



(a) Color distribution

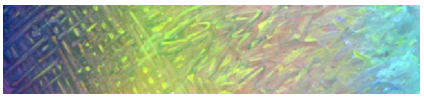


(b) Color trajectory

Figure 5: RGB Path for Figure 4(a).

(a) A scanned paint sample

(b) User specified color gradient



(c) Applying the new gradient to the paint sample

Figure 6: Changing color transitions without affecting texture.

(a) Paint samples

(b) Image Based Texture Synthesis

(c) View Aligned 3D Texture Projection

(d) View Dependent Interpolation

Figure 7: Rendering a skull mesh with various paint samples.