A system for real-time watercolour rendering

Conference Paper *in* Proceedings of Computer Graphics International Conference, CGI · July 2005

DOI: 10.1109/CGI.2005.1500426 · Source: IEEE Xplore

CITATIONS
9

READS
406

3 authors, including:

Geoff Wyvill
University of Otago
88 PUBLICATIONS **2,854** CITATIONS
SEE PROFILE

Scott King
Texas A&M University - Corpus Christi
77 PUBLICATIONS **528** CITATIONS
SEE PROFILE

Please see the color plate on page 286.

A System for Real-Time Watercolour Rendering

Jeremy Burgess*
Computer Graphics and Vision Research Group
Department of Computer Science
University of Otago

Geoff Wyvill†
Computer Graphics and Vision Research Group
Department of Computer Science
University of Otago

Scott A. King‡
Computing & Mathematical Sciences
Texas A&M-Corpus Christi

ABSTRACT

We present a system for the real-time rendering of scenes to produce images like watercolour paintings. Our system uses a hybrid of object space rendering techniques and image plane post processing to attain extremely good results very quickly. We achieve this through Graphics Processing Unit (GPU) programming, hardware accelerated shadow volumes, and the use of noise for various different tasks. We manage to circumvent problems found in both previous image space and object space methods by combining the two.

Keywords: NPR, Real-Time, Paint, Watercolour.

1. INTRODUCTION

Nonphotorealistic rendering (NPR) is the family of techniques that can be used to produce scenes that do not exactly resemble the real world. Standard, realistic graphics techniques do not produce image features that are necessary in many NPR fields. The field that we are interested in is painterly rendering – the rendering of scenes in the style of paintings. Some of the characteristics of paintings that are missing from standard approaches to rendering are:

- The sketchiness inherent in certain styles of painting.
- The presence of brushstrokes.
- Complex colour interactions caused by paint on canvas rather than by light in the real world.

NPR methods that attempt to model these characteristics have typically worked on finished images [1][2][3]. That is to say, the system would be provided with a rendered scene or photograph, and it would then process it so that it resembled a painting. Our method does much of its work in object space.

There are two key problems to painterly rendering in the image plane that we have dealt with through object space techniques. Firstly, it is usually very slow. The speed of such processes is not a necessary feature of image plane techniques. Such methods are typically slow because of the large amount of processing needed at every pixel. The second problem is that the post-processing step usually has only the information contained in a single frame. Because of this, generated features, such as brush strokes, are not coherent.

We present a hybrid system for rendering images in the style of watercolour paintings. Working in hardware and doing much of the 'hard work' in object space, our system is very fast. Object space techniques produce frame-to-frame coherency almost automatically.

Our system first produces various pieces of information about each pixel in the scene through object space methods, and then performs a post-processing step on this information to produce a final frame. In section 2 we look at the features of real watercolour paintings, examine previous work in painterly rendering, and discuss the limitations and benefits of working on the GPU. In section 3 we look at how we can rapidly produce shadows using a technique found primarily in video games, and in section 4 we examine our paint layers algorithm from beginning to end. In section 5 we examine our results, and in section 6 provide a brief discussion of our success up to this point with an eye towards future work.

2. BACKGROUND MATERIAL

2.1. Real Watercolours

To make clear what we are trying to achieve, we now present some real watercolour techniques (Figure 1).

- The Basic Wash: A wash is a layer of a single pigment of paint. An important variation is the graduated wash, in which one pigment is thinned as the wash progresses to create a gradient.
- Glazing: This is the process of painting a large number of very thin washes in order to produce the effect of colours 'glowing' through.
- Wet in Wet: This is when we paint onto a wet surface. Wet in wet painting causes the paint to spread somewhat unpredictably.
- Negative painting: In traditional watercolours, white is never used. If white is desired, the paper is left blank, hence the term negative painting.

* e-mail: jburgess@cs.otago.ac.nz
† e-mail: geoff@cs.otago.ac.nz
‡ e-mail: sking@sci.tamucc.edu

Proceedings of Computer Graphics International 2005 (CGI'05)
June 22-24, 2005, Stony Brook, NY, USA
0-7803-9330-9/05/$20.00 ©2005 IEEE

Figure 1: Three swatches produced with actual watercolour paints. The left shows a basic wash, the middle glazing, and the right wet in wet. See also colour plate.

2.2. Previous work

Painterly rendering has been concerned primarily with the image plane. We can attribute this to the fundamentally 'flat' nature of real paintings – all paintings must be created on a canvas or page of some sort. Many painterly effects, such as brush strokes and fluid simulation are easier to simulate and create in the image plane than in object space.

One exceedingly successful algorithm for producing watercolour images was designed by Curtis et al. [1]. Using a physically based colour model, and complex simulations of the interactions of paints, it could create any of the real world watercolour effects discussed in Section 2.1. The images produced by this system do resemble certain styles of watercolour painting.

However, Curtis et al.'s system is extremely slow and only semi-automatic – it requires user assistance to achieve very good results. Additionally, Curtis et al. provide very sparse discussion on how to extend their system for use with animation.

A system proposed by Barbara Meier is quite different in that it does not deal with creating watercolour scenes at all, but cunningly solves the problem of coherency in animation by moving brush strokes into object space [4]. Strokes are seeded on objects through the use of randomly spread particles. In Meier's system strokes always behave as they are expected to, and never move unpredictably.

Our own system takes much of its inspiration from another created by Eric Lum and Kwan-Liu Ma, which takes some of the good work from both Curtis et al. and Meier [6]. Their system provides brush stroke like textures in object space, and composites multiple layers of paint using the colour model found in Curtis et al.'s paper. Our method differs from that proposed by Lum and Ma in that we simplify brush stroke generation and add a post processing image plane step. The results of our simplified brush stroke generation resemble those in watercolours more closely. Our method produces a greater resemblance to a real watercolour wash and maintains the property of showing object contours. Our post-processing step allows us to gain specific effects present in real watercolours such as edge darkening and non-straight object edges. Lum and Ma use raytracing to draw the image to the screen, whereas we use rasterization, the OpenGL API, and programmable graphics hardware.

2.3. GPU programming.

We have implemented our system, where possible, on the GPU of a modern graphics card. To make some terminology clear, a brief outline of what we can do here is included.

Essentially, there are 2 types of GPU programs, called shaders. The first are vertex shaders, which operate on the vertices of polygonal models; and the second are the fragment shaders, which operate on rasterized pixels (Figure 2). Vertex shaders permit us to alter position, normal, and texture coordinate data for each vertex, and can output a limited amount of information to a following fragment shader. Fragment shaders allow us only to modify depth and colour, however it is not uncommon to store non-colour information in rasterized pixels [10].

In addition, graphics related functions have been simplified in some of the high level shading languages for GPUs, such as Cg, HLSL, and Sh [11]. In particular vectors and matrices are treated as first class types, which makes many operations in graphics easier to deal with.

The main benefit of working on the GPU is the performance that modern graphics cards offer.

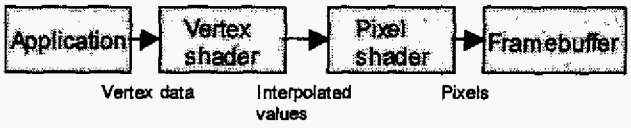

Figure 2: The Cg Graphics Pipeline

3. Hardware Accelerated Shadows.

One of the greatest benefits of working with raytracing, as Lum and Ma showed, is that shadowing is provided as a straightforward extension on the basic method. When working with an API based on rasterizing polygonal objects, this is not so, and generating shadows is more difficult. There are two commonly used methods for shadow generation, both of which can be accelerated in hardware – shadow mapping [12] and shadow volumes [5]. We decided that shadow volumes were preferable for our purposes. Although shadow-maps provide soft shadows, they also produce artifacts when the resolution of the maps is insufficient, or when the angle between the light and shadow-casting surface is too low. Additionally, the conceptual simplicity and generality of shadow volumes appealed to us.

Shadow volumes use the principle that every area that is occluded by a certain object belongs to a volume in space where light is not cast. Creating these volumes is reasonably easy, but how do we harness the raster pipeline to figure out which pixels are in a given shadow, and which are not?

To create shadow volumes for a given light source, all we do is extrude the silhouette edges of each object from a light source to infinity, and cap each end with a polygon. Because we have to rasterize these shadow polygons in the process of using them, it is easier to do this on the fly in a vertex shader than to calculate silhouette edges in software. The vertex shader checks to see if a vertex is 'front facing' relative to the light source, and then does one of two things. If the vertex faces forwards, it is left where it is. If it does not, it is extruded to infinity. A sample shader that does just this can be found in the Cg Handbook [10].

At the end of the shadow volumes process, we know which pixels are in shadow and which are not. Our implementation provides for only one light source. We could adapt our system to use multiple light sources by storing each light's stencil buffer in a texture. This is not too useful though, as the vast majority of watercolours include only one light source.

Note that normally in shadow volume rendering, we do one pass for each light source and add the contribution of that light to places where there is no shadow. As we need to know where shadows are (Section 4), we want the reverse of this.

For a more detailed discussion of shadow volumes, see Lengyel [5].

4. Our Method

Our method is based on the idea that a real watercolour painter rarely paints exactly the colour he or she wants at a given point, and instead paints several layers to get the desired effect. Our system is a very simplified version of this idea that consists of only three layers of paint for any given point.

To start with we have a scene determined by a set of polygonal models and a scene description. Each object must have a pigment and a transformation matrix to position, scale, and rotate it in space. All the user needs to do is specify each of these features in a scene description file, and the system does the rest. Ideally the objects are fairly simple. The cups in the two cups scene in section 5 for example consist of 10 000 faces each, and the scene can be rendered 25 times per second. Increasing the polygon count of the scene results in reduced performance, but otherwise does not affect the functioning of the system. To render the scene our system consists of four key steps:

1. Determines shadows (see section 3).
2. Determines paint thickness and colour.
3. Performs image plane post-processing.
4. Composites layers.

4.1 Paint layer calculation

Like Lum and Ma's system, our system has three layers of paint: a diffuse layer, a shadow layer, and a texture layer [6]. Each layer is ideally supposed to represent a different kind of layer that could occur in a real watercolour painting. It is important to note that colours here are pigments (see Section 4.3), rather than RGB triples.

For a given pixel, the diffuse layer is a uniform thickness of the pigment assigned to that object (Figure 3), and is merely supposed to represent the actual colour of the object. For example, if we were looking at pixel (30, 40), and at that point a pink vase was to be drawn, the diffuse layer would be a uniform thickness of a pink pigment. Thickness is a relative value where '1.0' is the base thickness for a single layer of paint for use with the Kubelka-Munk model (Section 4.3). Although a unit thickness is convenient, it is inappropriately dark for watercolours based on a small number of washes. We have found that using 0.5 for the base thickness provides the most pleasing results.

The shadow layer is the second layer, and consists of a uniform pigment for a scene, with varying thickness (Figure 3). Wherever there is complete shadow, this layer's thickness is 0.1. In places where the phong lighting equation yields some darkness, but not total shadow, we allow the shadow thickness to vary between 0 and 0.1.

The final layer is the most interesting, and is the 'texture layer'. This is the layer that is supposed to give the illusion of the texture in a watercolour wash. At any given point it has the same pigment as the diffuse layer, but its thickness is free to vary. The thickness of this layer is found by the following procedure:

1. Find sample point, x, of this fragment in the image plane.

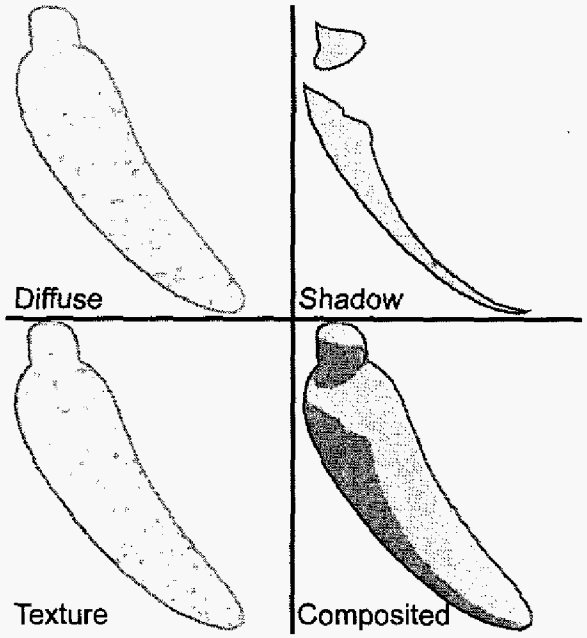

Figure 3: The three paint layers on a banana, and all three layers composited. Notice how the strokes follow the surface in the texture layer. See also colour plate.

2. Sample a two component noise texture using x, and add the result to the x – this gives a 'wobbly' effect. In our implementation we use Wyvill noise [9].

3. Find the projection of the surface normal in the image plane at this point, and add this also to x. This will cause the sampled noise to follow the surface (Figure 3).

Note that in Lum and Ma's system, a Line Integral Convolution (LIC, see Interrante [8]) was used to get the effect of strokes stretched around objects. This, we believe, is both overly expensive and obtains unrealistic results. The LIC also appears too artificial – the perfection of the strokes following the surface has no analogue in real painting.

All of this information is calculated in a preliminary shader. It outputs an image that preserves all of this information perfectly per pixel without doing any processing on it. Each pixel then has the following format:

Red channel = normalised object identifier (id). Each object is assigned a unique id when the scene is read in. To convert an object id to a value between 1 and 0, we divide it by the total number of objects.

Green channel = shadow thickness.

Blue channel = texture thickness.

Alpha channel = depth.

We don't need the diffuse thickness, as it is uniform anyway.

4.2. Image plane post-processing

The original images produced with our system were reasonable, however it was clear that adding some postprocessing with a specific watercolour goal in mind would yield better results. In particular two key features could add a lot to the images produced. Firstly, in real watercolours there is darkening around edges. This is because pigment tends to move toward the edges of

wet areas. The second point is that real watercolours are not perfect. Object shape is rarely precise.

For the edge effect, the first thing we do is take a gaussian average of 25 sparsely sampled pixels from the previous frame across all channels (Figure 4). Then, we find out three differences that indicate what to do next: the respective differences between the average and original object id (i), the shadow thickness (s), and the depth (d). Note that the difference between object ids is treated abnormally. The difference here is only between the decimal component of the two values. So, if the original object id was 10.0, and the averaged id was 7.5, we would find the absolute difference between 0.0 and 0.5.

We thicken the diffuse pigment if the depth is lower than the average depth, and we thin the diffuse pigment if the depth is higher than the average depth, but only if the absolute value of the difference exceeds a threshold value. Regardless of the threshold value, if the object id varies, we thicken or thin the diffuse pigment proportional to both depth and object id difference. Finally, we thicken the shadow layer proportionally to the difference between the shadow average and the original shadow pixel.

The difference between doing this edge darkening and not doing it is striking, and adds significantly to the effect of the image (Figure 5).

All sample points are offset by x and y noise values, and this leads to the imperfect appearance of the objects, characteristic of real watercolours.

4.3 Compositing

Our system uses the Kubelka-Munk colour model [7]. In this model each colour is actually a pigment with light scattering and absorption 'values' across all relevant wavelengths (red, green and blue). The idea behind this model is that a given pigment of a real paint will look different depending on how thick it is, and what colour is underneath. For example, if a colour should be very strong over white, but almost transparent over black, the Kubelka-Munk colour model will accurately show this [7].

The Kubelka-Munk model has been shown to be very accurate for oils, and although it is not perfectly designed for watercolours, it has been shown to be very successful. Pigments can be specified as two RGB triples; one over white, and one over black. Equations for compositing and calculating relevant values using pairs of RGB triples rather than scattering and absorption values across all wavelengths can be found in Curtis et al. [1].

All compositing is performed in a pixel shader in our system. Figure 3 shows a fully composited object.

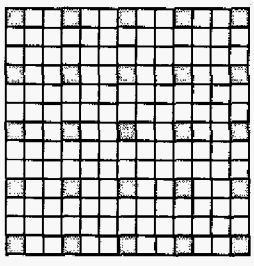

Figure 4: Sampling grid. All samples have their positions offset by noise. The center square is the pixel to be drawn, where the other gray squares are other samples.

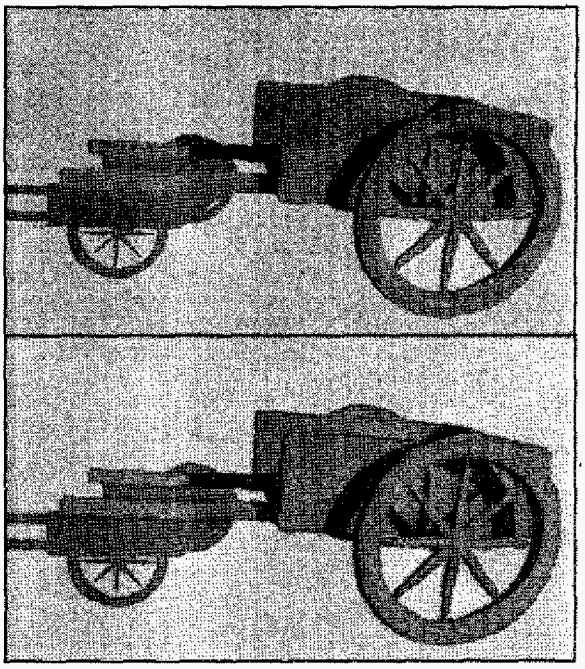

Figure 5: A wagon with (bottom) and without (top) edge darkening.

5. Results

Our results have been excellent. Our system was based on one proposed by Eric Lum and Kwan-Liu Ma. Their system renders single objects at 128 x 128 pixels in less than one second. In contrast, we can render single objects at 512 x 512 pixels at approximately 50 frames per second on an nVIDIA GeForce 6800. Our two cups scene (Figure 9) is rendered at 512 x 512 pixels at approximate 25 frames per second.

Our system also produces images that are an improvement on some previous results, looking more like real watercolours. Although our images have fewer features of real watercolours than those of Curtis et al. [1], they are produced much faster. The only way to assess a system like ours is to examine our images – see figures 7, 8 and 9.

Note also that we have maintained a good degree of coherency in animation. Only two features detract from this. Firstly, the noise texture used to reduce the perfection of the image never moves. This is a very slight defect and it could be reduced by aligning the noise sample plane with the original scene camera, rather than the post-processing camera. This would remove any incoherency when moving through the world, although rotating the camera would still cause minor artefacts.

The second, and more noticeable, feature that detracts from coherency in animation is that the noise used for brush stroke textures is always aligned with the camera. Camera or scene rotation causes objects to 'move through texture space'. This could be repaired through the use of a cube-mapped noise texture. We can still use the normal offset to get object contours to show up by projecting normals in the planes of the cube map. Even without this, in our system the brush strokes are always predictable, and do not look bad at all. Figure 6 demonstrates a short animation of a single object. Notice how brush strokes are for the most part coherent, and always follow object contours.

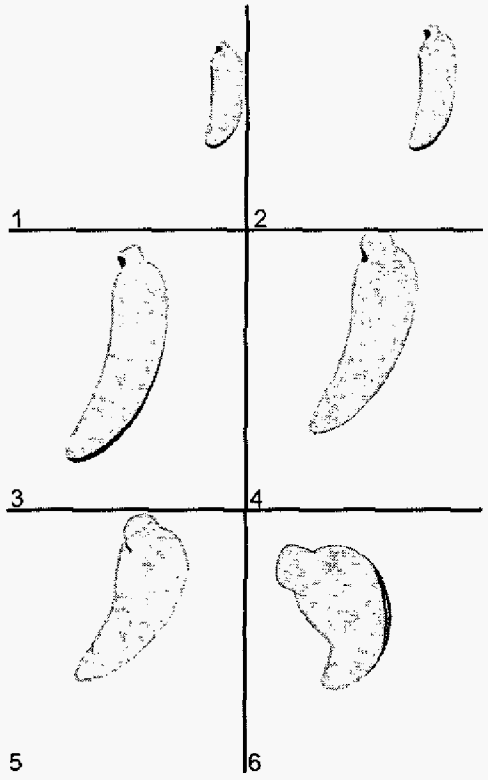

Figure 6: A small animation of one object. Notice how brush strokes remain coherent. Contrast has been increased to make strokes more obvious.

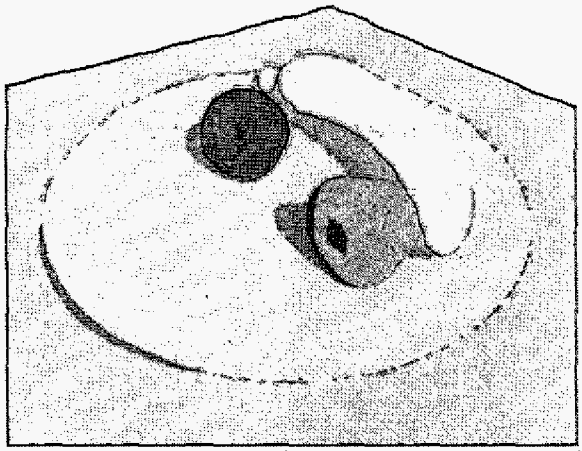

Figure 7: A still life rendered with our system. See also colour plate.

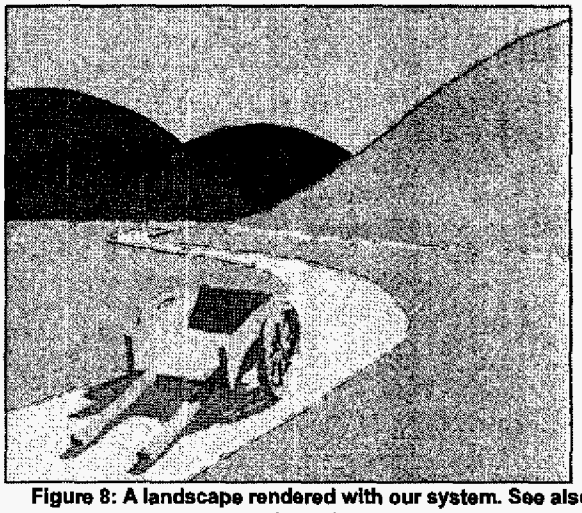

Figure 8: A landscape rendered with our system. See also colour plate.

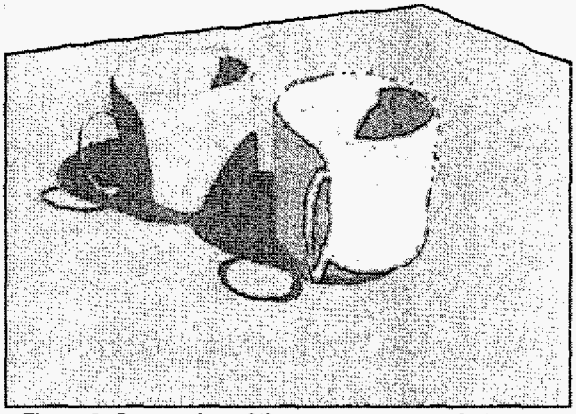

Figure 9: Our version of the two cups scene that appears in Lum and Ma [6]. See also colour plate.

6. Conclusions and Future Work.

Our system is that it is both an improvement on similar systems in terms of visual results, and is much faster. The power of the modern GPU has allowed us to create a reasonably complete system that still allows for real time navigation of a scene. With a small number of modifications, it could also allow for the real-time alteration and creation of scenes for use with this style of rendering.

To be fair we do need to ask ourselves whether or not our results actually look like real watercolours. We would suggest that in many ways, they do look a lot like some kinds of watercolour. That said, the images we produce still lack something essential to real watercolours. This is not to say that we have failed, merely that aspects of certain real watercolours are lacking from the images we produce.

There are various avenues that we could follow in attempting to rectify this. Probably more than anything else, a detailed examination of the quality and character of a real wash would be helpful for improving our method of noise texturing, such that it would produce even more of the characteristics of a real watercolour wash. Another important aspect that we should

investigate is a way to reduce the very 'flat colour' appearance of flat objects. The use of depth to reduce thickness may improve results here.

Secondly, it would help to add some kind of texture processor that makes a 'best guess' as to the pigment that each colour in a texture should be assigned. With this, we could produce objects painted in more than one pigment, which would add significantly to the quality of scenes we could produce. An extension to this would be to try to reproduce wet in wet colour mixing within objects. We believe that this could be achieved with a noisy blur that increases in length proportional to the distance from the center of a pigmented area on an object. Obviously we would need a way to get this information in our post-processing step.

Finally, and in an attempt to get closer to real watercolours, it may add a lot if we were to actually composite objects on top of one another. Of course within layers this should not happen, but it is not uncommon in real watercolours to find a background, with a middle and foreground painted on top. To get this information, we could render to multiple textures, with each containing fragments only within a certain range of depth.

Our results are very promising. They are both fast, and look very good. Additionally, with the increasing power of GPUs, we can expect even more available power. All additions we make will have some performance overhead, however, we believe that this expense is worth it.

REFERENCES

[1]. Cassidy J. Curtis, Sean Anderson, Josh Seims, Kurt Fleischer, and David Salesin. Computer Generated Watercolor. In SIGGRAPH '97 Conference Proceedings, pages. 421–430, August 1997.

[2]. Aaron Hertzmann. Paint by Relaxation. In CGI 2001, pages 47-52, 2001.

[3]. Doug DeCarlo and Anthony Santella. Stylization and Abstraction of Photographs. ACM SIGGRAPH 2002, pages 769-776, 2002.

[4]. Barbara J. Meier. Painterly rendering for animation. In SIGGRAPH '96 Conference Proceedings, pages 477-484, 1996.

[5]. Eric Lengyel. The Mechanics of Robust Stencil Shadows. Gamasutra.com, http://www.gamasutra.com/features/20021011/lengyel_pfv.htm, visited on 7/8/2004.

[6]. Eric B. Lum and Kwan-Liu Ma. Non-Photorealistic Rendering using Watercolor Inspired Textures and Illumination. In Proceedings of the 9th Pacific Conference on Computer Graphics and Applications (PG'01), IEEE, pages 322-331 2001.

[7]. Chet S. Haase and Gary W. Meyer. Modelling Pigmented Materials for Realistic Image Synthesis. In ACM Transactions on Graphics 11, 4, pages 305-335, 1992.

[8]. Victoria Interrante. Illustrating surface shape in volume data via principal direction driven 3d line integral convolution. In SIGGRAPH '97 Conference Proceedings, pages 109-116, 1997.

[9]. Geoff Wyvill. "The Nature of Noise", http://www.cs.otago.ac.nz/graphics/Geoff/NoisePages/Nature.html, visited on 17/7/2004.

[10]. *The Cg Toolkit Users Manual Release 1.2.* From developer.nvidia.com, nVIDIA Corporation, 2004.

[11]. William R. Mark, R. Steven Glanville, Kurt Akeley, and Mark J. Kilgard. Cg: A system for programming graphics hardware in a C-like language. In SIGGRAPH 2003 Conference Proceedings, pages 896-907, 2003.

[12]. Lance Williams. Casting Curved Shadows on Curved Surfaces. In SIGGRAPH 1978 Conference Proceedings, pages 270-274, 1978.

Figure 1: Three swatches produced with actual watercolour paints. The left shows a basic wash, the middle glazing, and the right wet in wet.

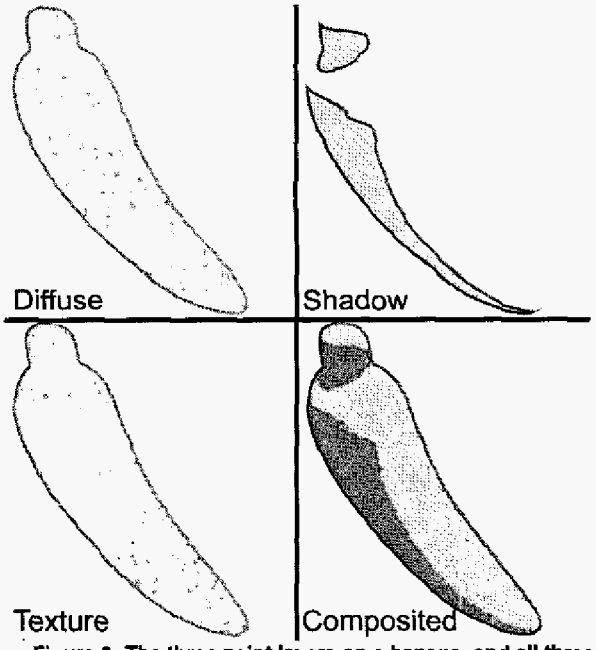

Figure 3: The three paint layers on a banana, and all three layers composited. Notice how the strokes follow the surface in the texture layer.

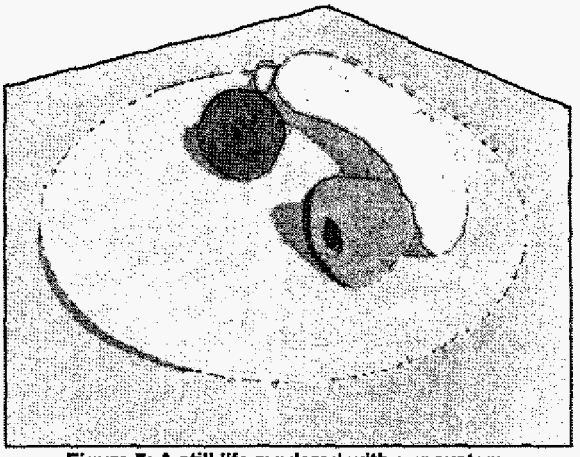

Figure 7: A still life rendered with our system.

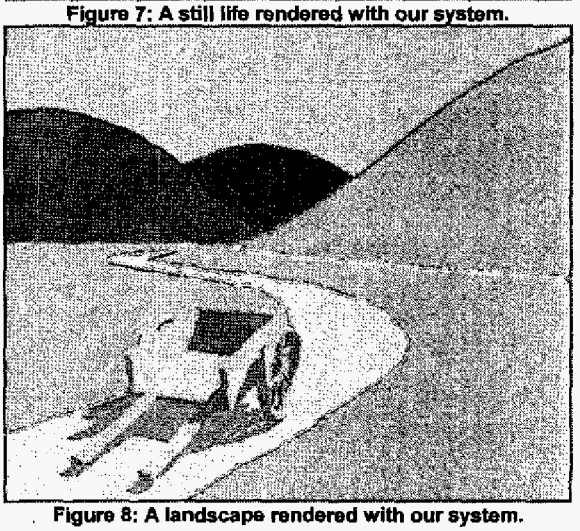

Figure 8: A landscape rendered with our system.

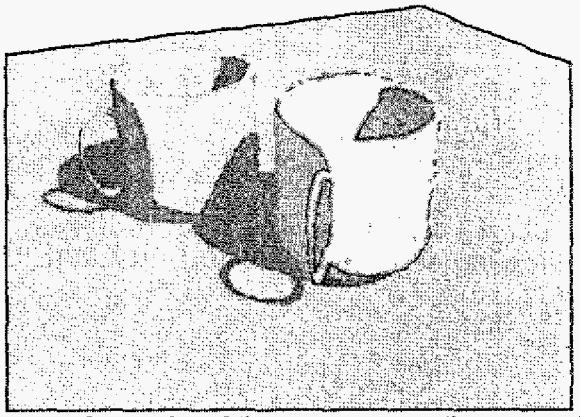

Figure 9: Our version of the two cups scene that appears in Lum and Ma [6].