Painterly Rendering for WebGL

Andy Hanson*
Scott Todd†

Figure 1: *The Utah teapot rendered using our system.*

Abstract

TODO.

Keywords: radiosity, global illumination, constant time

1 Introduction

TODO.

(Link to GitHub and hosted url)

1.1 Related Work

TODO [Meier 1996].

In [Hertzmann 1998], 2D input images are painted over by layers of spline brush strokes. Strokes are placed along areas of high gradient on a source image blurred proportionally to the current brush size. In this manner, progressive emphasis is placed on regions of high visual interest. An intuitive collection of parameters is chosen such that a user may easily select between a spectrum of visual styles. Parameter sets are presented for Impressionist, Expressionist, Colorist Wash, and Pointillist styles.

[Kalnins et al. 2002] explores interactive techniques that allow artists to create non-photorealistic renderings of 3D models. Artists are given control over brush styles, paper textures, basecoats, background styles, and the position and styling of each stroke. Their system uses information supplied from one or more viewpoints to render the models at any new viewpoint while preserving the desired aesthetic.

*email: hansoa2@rpi.edu
†email: todds@rpi.edu

2 Painterly Rendering System

2.1 Algorithm Overview

Our system takes as input a set of three.js geometries and a list of rendering parameters for each geometry, as well as a list of three.js directional lights. It outputs to a three.js WebGL renderer in any supporting browser.

TODO.

2.2 Stroke Selection

TODO.

3 Stroke Rendering

We represent each brush stroke as a particle within a particle system that is maintained by three.js. We provide a vertex shader and a fragment shader for three.js to use in rendering the particle system. Each layer of brush strokes on each object in the scene has its own particle system.

3.1 zQuality

We found that three.js could sort particles by depth for rendering, but this was too slow for our application and the number of particles we were using. Blending within the shaders was also unsuccessful.

We use a two-pass rendering approach to perform z-culling. First, we render only the original geometry into a floating point depth buffer texture. Then, we pass that texture into our brush stroke shaders and use that information to choose which strokes to render.

In the first iteration of our algorithm, we discarded fragments whose depth values differ from the value stored in the depth buffer at that position. This method performs a sharp cutoff around the silhouette of each object, so it conflicted with the painterly rendering focus of our project.

We refined this approach by expanding the silhouette of each object in the scene and fading strokes in based on the difference between

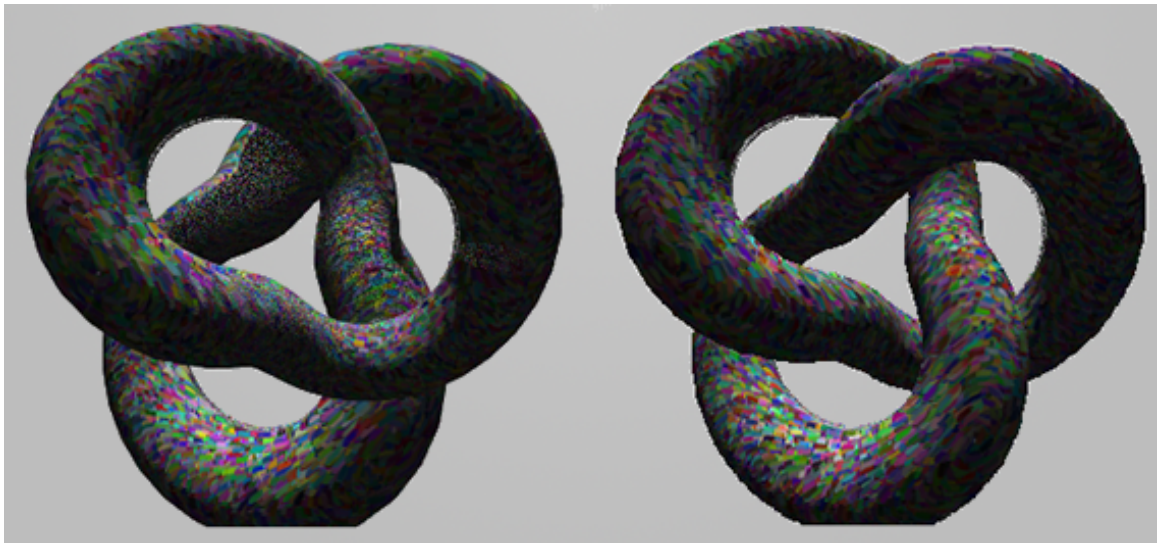

Figure 2: *The image on the right discards fragments whose depth values differ from the value stored in the depth buffer texture while the image on the left does not. Note the intersections and other artifacts on the left.*

their depth and the depth in the depth buffer texture. We expanded the silhouette of each object by pushing each vertex of their geometries by a factor of the normal at that vertex.

3.2 Gradient Estimation

TODO.

3.3 Layering

As in [Hertzmann 1998], [Meier 1996], (and others?), we use multiple layers of brush strokes for each object. While the layer properties can be user-controller, we intend for the base layers to contain a smaller quantity of larger brush strokes and the top layers to contain a larger quantity of smaller brush strokes. The base layers capture rough details and cover the entirety of the original mesh, while the top layers show fine details and are expected to leave gaps between each other. We found that three layers of strokes was sufficient for the scenes that we tested.

We support using a specular threshold and specular fade in to control the visibility of strokes within a layer. If the specular lighting contribution at a stroke does not exceed the minimum value, it is discarded. Similarly, if a stroke has low specular, we change its alpha value based on the specular fade in. This allows us to approximate the visual interest sampling in [Hertzmann 1998] and (other reference).

3.4 Parameters List

TODO.

4 Results

TODO.

5 Limitations

TODO.

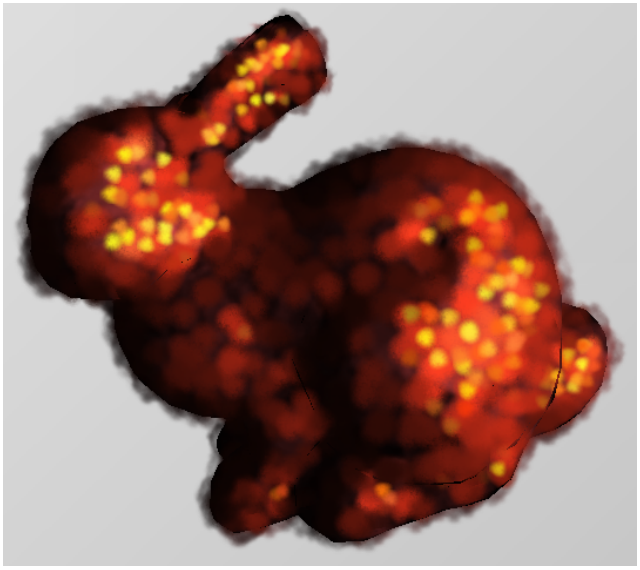

Figure 4: *The Stanford bunny rendered using three layers of brush strokes. The topmost layer uses a minimum specular cutoff of 0.95 and a specular fade-in of 0.63.*

6 Future Work

TODO.

7 Conclusion

TODO.

Acknowledgements

(TODO) Three.js

Figure 3: *From left to right: no orientation or curvature, orientation with no curvature, orientation and exaggerated curvature (2.0).*

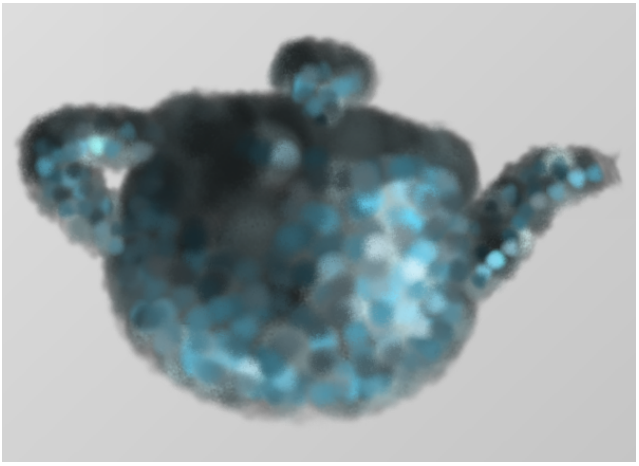

Figure 5: *The Utah teapot rendered using three layers of brush strokes. The topmost layer uses a specular fade-in of 0.03.*

References

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 453–460.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. Wysiwyg npr: Drawing strokes directly on 3d models. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '02, 755–762.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 477–484.