Painterly Rendering for WebGL

Andy Hanson*
Scott Todd†

Figure 1: *Spring Training 2009, Peoria, AZ.*

Abstract

TODO.

Keywords: radiosity, global illumination, constant time

1 Introduction

TODO.

(Link to GitHub and hosted url)

1.1 Related Work

TODO [Meier 1996].

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.

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

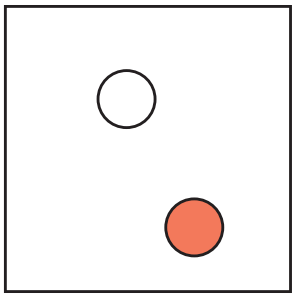

Figure 2: *Sample illustration.*

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 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.

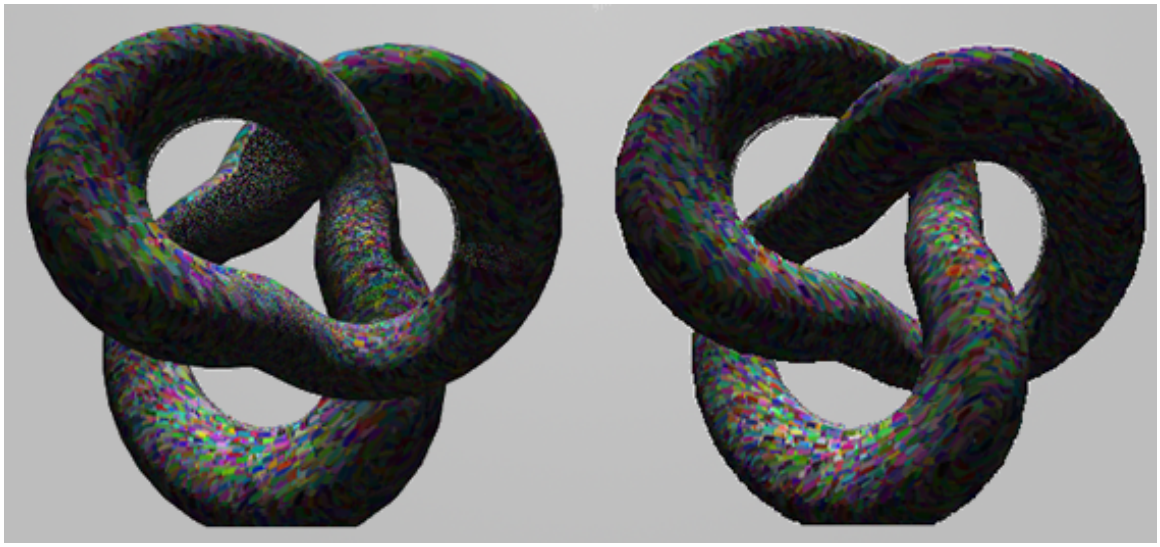

Figure 3: *The image on the right discards fragments whose depth values differ from the value stored in the depth buffer texture while the image on the left does not. Note the intersections and other artifacts on the left.*

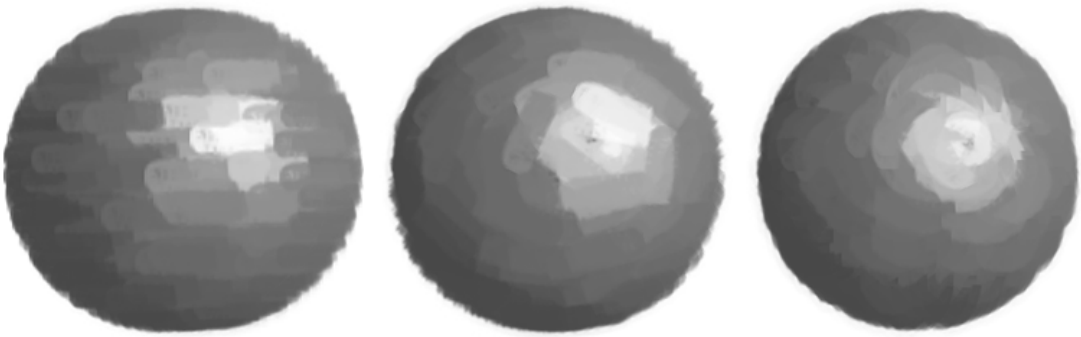

Figure 4: *From left to right: no orientation or curvature, orientation with no curvature, orientation and exaggerated curvature (2.0).*

3.2 Gradient Estimation

TODO.

3.3 Layering

TODO.

3.4 Parameters List

TODO.

4 Results

TODO.

5 Limitations

TODO.

6 Future Work

TODO.

7 Conclusion

TODO.

Acknowledgements

(TODO) Three.js

References

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.

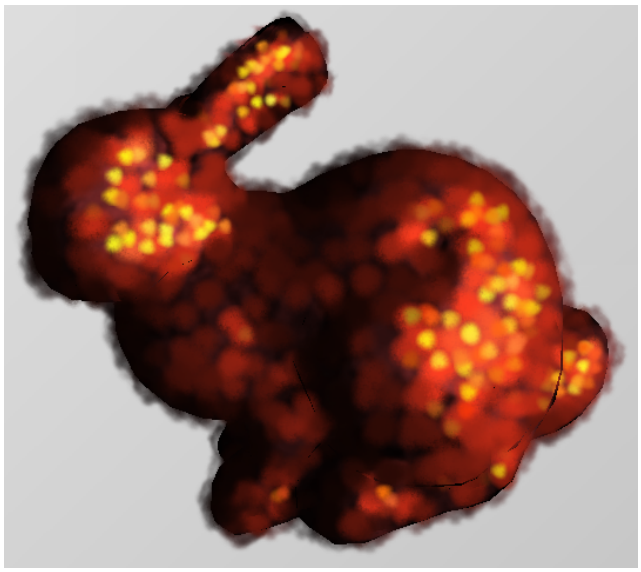

Figure 5: *The Stanford bunny rendered using three layers of brush strokes. The topmost layer uses a minimum specular cutoff of 0.95 and a specular fade-in of 0.63.*

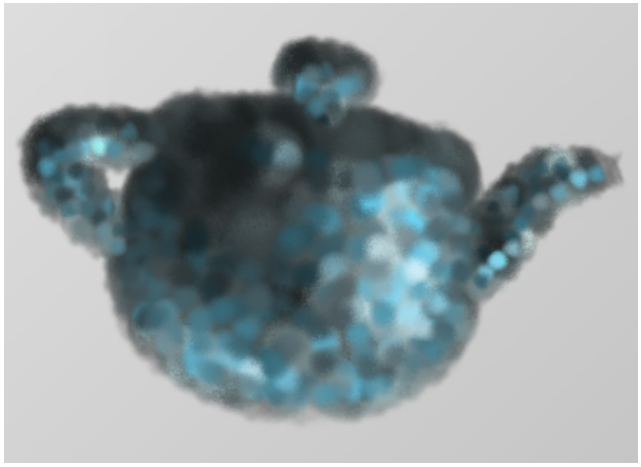

Figure 6: *The Utah teapot rendered using three layers of brush strokes. The topmost layer uses a specular fade-in of 0.03.*