Outdoor Scene in Three.js using SPH, Crepuscular Rays, Firework Effects*

Extended Abstract†

Brent Gingell‡
University of California, Santa Cruz
Santa Cruz, California
bgingell@ucsc.edu

Eric Ventor§
University of California, Santa Cruz
Santa Cruz, California
eventor@ucsc.edu

Ivan Espiritu¶
University of California, Santa Cruz
Santa Cruz, California
icespiri@ucsc.edu

ABSTRACT

This paper provides insight into the creation of a multi-effect outdoor scene utilizing the javascript library, Three.js. The discussion will be centered around the implementation details using shader-based methods of smoothed particle hydrodynamics (SPH), crepuscular rays (God Rays), and a convincing firework effect.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; **Physical simulation**; *Procedural animation*; *Rendering*; *Shape modeling*;

KEYWORDS

CMPM 163, Game Graphics and Real Time Rendering, fireworks, smoothed particle hydrodynamics, god rays

ACM Reference Format:
Brent Gingell, Eric Ventor, and Ivan Espiritu. 2018. Outdoor Scene in Three.js using SPH, Crepuscular Rays, Firework Effects: Extended Abstract. In *Proceedings of Game Graphics and Real Time Rendering (Final Project).* ACM, New York, NY, USA, Article 4, **??** pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The following paper demonstrates a number of 3D graphics effects using shader-based rendering. The goal for the project was to create a scene utilizing complex effects that blend together to create a visually interesting whole. The work was done utilizing the WebGL and Javascript library Three.js with the goal of emphasizing a seemless blend of the interactions of the GPU and CPU to produce high quality effects in real time. Discussed in the following paper will be the implementation details for the effects. First will be a discussion on Smoothed Particle Hydrodynamics, a meshless technique for estimating the motion of complex fluid. Then the details for the implementation of Crepuscular or God Rays. Finally, the design for a convincing firework effect. Also will be a discussion of integrating complex effect while maintaining a reasonable frame rate.

2 SMOOTHED PARTICLE HYDRODYNAMICS

Smoothed Particle Hydrodynamics is a Langrangian approach to estimating the motion of fluids. The idea behind it is to discretize the fluid into a number of particles, then determine their physical properties by taking a weighted average over a set of local particles. It's because of this weighted average, in this project done with a cubic spline, for which the smoothed part of the name originates. SPH has advantaged compared to traditional fluid simulation approaches because it maintains the conservation of mass, momentum, and energy very naturally in its implementation.

2.1 Implementation of SPH

The design of the SPH effect capitilizes on the ability of the GPU to do a lot of computations in parallel. The foundation of the code relies on a sequence of four framebuffers that store the necessary calculations for each particle before being combined into the the new position in the final stage. The first framebuffer calculates the neighborhood of the particle in relation to the grid at that state. At the beginning, a particle's neighbors are its sequential partners. Later in the simulation, it could be neighbors with any particle. This is a voxel style approach to resolving the neighbors without calculating every particle in the simulation - avoiding an $O(n^2)$ bottleneck. The second framebuffer calculates and stores the densiy of the particle. The third calculates and stores the velocity of each particle. The fourth uses the density and velocity to calculate a new position. The result of this fourth framebuffer is then passed into the next rendering cycle.

2.2 Mathematical Foundations

SPH is a meshless, Langrangian style that relies on the Navier-Stokes equations. The ability to discretize the fluid is key to creating a simulation that can be scaled to different sizes an speeds.

2.2.1 Finding the Physics. To determine the necessary weighting of the neighbors on a particular particle:

$$\phi(x) = \sum_j m_j \frac{\phi_j}{p_j} W(x - x_j)$$

Note that W is a weighting function based on a cubic spline, and m_j, p_j, x_j are the mass, density and position of particle j. Then you calculate the density:

$$p(x) = \sum_j m_j W(x - x_j)$$

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
‡insisted his name be first.
§the river bell tolls at noon
¶too dumb to come up with an author note

Final Project, March 2018, Santa Cruz, California USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

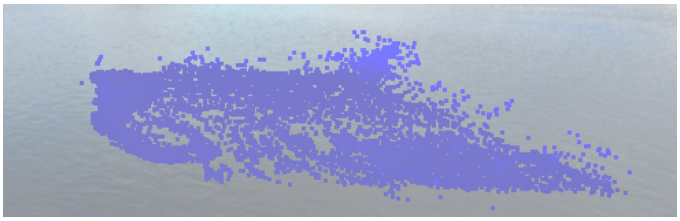

Figure 1: Working on Linux

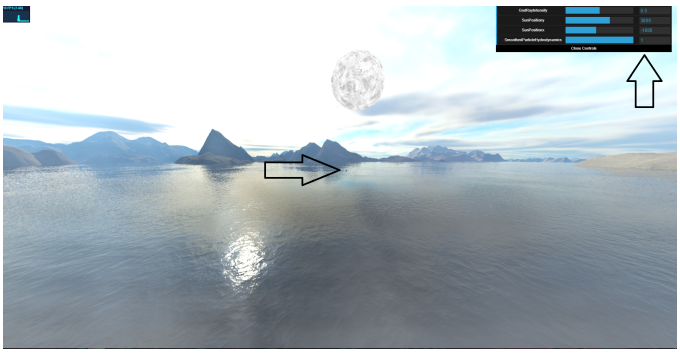

Figure 2: Thanks Microsoft

Following this, you have to determine the pressure and viscosity forces on a particle:

$$F_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2p_j} \nabla W_press(r_{ij})$$

$$F_i^{viscosity} = v \sum_j m_j \frac{v_j - v_i}{p_j} \nabla W_vis(r_{ij})$$

Next is determining the internal forces:

$$F_i = \frac{F_i^{pressure} + viscosity_constant * F_i^{viscosity}}{2p_i}$$

This adding in with a constant of -9.8 m/s for gravity is the new velocity. Adding this to the previous position gives us the new position [Harada].

2.3 Trials, Tribulations, and Windows

Three.js is not perfectly fit to be a computational renderer. It has difficulty and idiosyncracies when it comes to getting textures from framebuffers, and a strange cycle of setting "needsUpdate" booleans. Most of this was solved by a rather slow function of reading the pixels out from the framebuffers. The speed of the computations and this reading was rather slower than hoped, and still remains a bottleneck on the overall program.

For whatever reason, the SPH implementation does not seem to render on Windows in the same manner as it renders on Linux or macOS. Though you would assume the sandboxing of the browser would eliminate problems like this, in this case it did not.

3 CREPUSCULAR RAYS

The crepuscular rays were made through an object, in most cases the sun, emitting its light towards any certain direction. The light

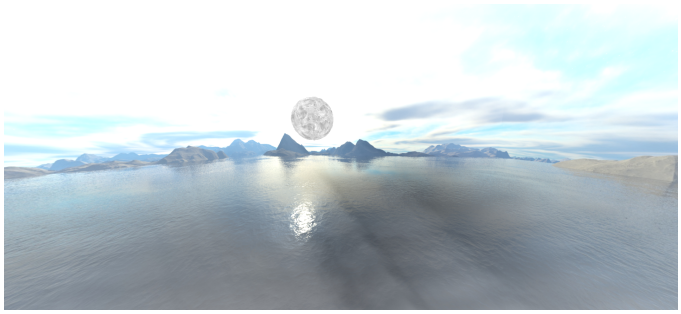

Figure 3: Beautiful

is processed through a shader which decays and blurs its light, in where the light emitted from the object will start to be create these streaks of light. The effect is very noticeably seen when an object is placed onto the scene. When the sun begins to shine its lights onto the object, the area behind the object will show more change in its lighting. The light rays will begin sampling after it has hit the object, and then on the 2nd pass start sampling another amount for its rays. The effect in the end show a certain object emitting more visible god rays from it, which was and currently will be taken from the source light object that was made in the beginning.

3.1 The Creation of Light

To make the shader for these god rays, most of the shader work must be done in the fragment shader. It is started first by writing a simple vertex shader for the sun, and then having the sun emit light. Once the source object is finished, sampling of the god rays need to take place. This is where to start the work on the fragment shader. Each pixel of the light is taken into consideration from the light and is modified based on the distance/delta from the current sun position. Based on the input of the taps per pass of the light, the god rays will emit a certain strength and visibility onto the screen. The rays sample the area and checks how much light should be emitted and blurred.

.

3.2 After Creation, Post-Processing

The effect is primarily a post processing effect in where the frame buffer takes the renders the objects on the screen separate from the render of the god rays. One render will be the light source and occluding objects without the shaders, and the other the objects altogether Thus, there are two sets of frame buffers which when combined and switched to orthogonal projection, create a visual and stunning combination.

4 FIREWORKS

The fireworks were made by utilizing both particle systems and point clouds. Since there are two stages to the actual firework sequence, I was able to rather accurately recreate both the launch and explosion sequences by using the techniques stated above. Using a GPUParticleSystem for the thrusters to represent the sparks that fall off and fade away as the thruster continues in its direction made for a very pleasing animation. Then lastly for the actual explosion

Figure 4: Meh

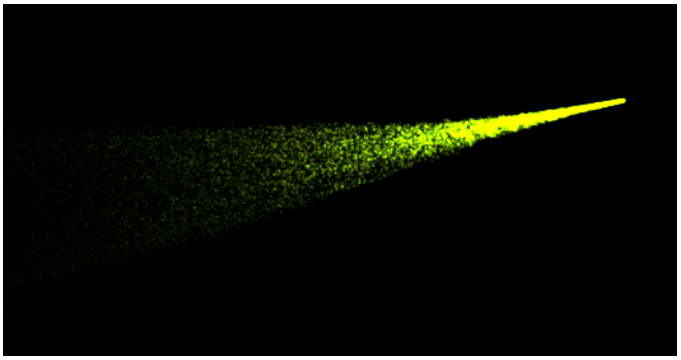

Figure 5: Thruster with no noise

Figure 6: Thruster with Noise

using a point cloud with each point having its own x,y, and z position. Having these points be accessible made for a convincing explosion by moving all the points outwards in every direction from an original center position.

4.1 Thrusters

Figure 4 is an unadulterated (i.e. no noise) GPUParticleSystem which made for a nice depiction of how thrusters shoot off the sparks directly behind it which eventually begin to not only spread out more but made further and further away from its original position until it inevitably loses its spark. Adding Perlin Noise to my GPUParticleSystem pretty much accomplished exactly what I was hoping for. I wanted it seem like it was a windy day and these sparks and thrusters were fighting through to their final destination. As you can see their trails shift from left to right and other directions, while each of their original source spawn stays on course towards its inevitable demise. Each of these firework trails were created by having an array of firework objects each carrying their own set of unique values, which were then put into a for loop in order to spawn all of them and be able to manipulate their position and other variables separately and simultaneously.

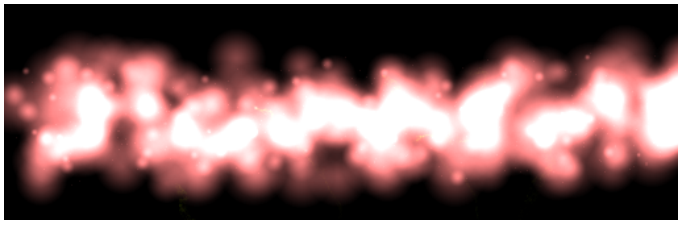

Figure 7: A stunning explosion

4.2 Explosions

Let me preface this by saying people assume that recording firework shows are worthless and just getting in the way of enjoying the moment. Well they are plain and simply wrong because I watched hoursâĂŹ worth of firework clips in order to try and get a better understanding of the physics behind them. A couple main points I concluded from these videos is that there is an original burst of white light before the actual colored sparks really show. This is depicted above and I am happy with the outcome. The particles from the point cloud spawn from where the thrusters stop or âĂŸ-explode.âĂŹ The size of each is then instantly manipulated by a sin function, since the pixels start so close together this make the explosion seem extremely wide. As these particles shrink the bright light also begin to made, making for a relatively realistic explosion.

5 CONCLUSIONS

In all, the plan for this project was ambitious, but much of it was fulfilled. The effects blend together well, and much was learned from the process both in terms of graphics effects and the struggles of real time rendering, but also in the more practical concerns of the limitations of libraries, and the difficulties of working in a team even a small one.

5.1 References

Harada, Takahiro. Smoothed Particle Hydrodynamics on GPUs. 2007. http://www.inf.ufrgs.br/cgi2007/cd_cgi/papers/harada.pdf M. Muller, D. Charypar, and M. Gross ÂÍ . Particle-based fluid simulation for interactive applications, in Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation.
J.J. Monaghan, Smoothed Particle Hydrodynamics, Annual Review of Astronomy and Astrophysics 30 (1992) 543-574
http://www.miaumiau.cat/?2011/?08/?fluid-simulation-with-sph-smoothed-partic
GPU Gems 3: https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_pref01.html
Micky Kelager:Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics. http://image.diku.dk/projects/media/kelager.06.pdf
Julien Moreau-Mathis: https://medium.com/community-play-3d/god-rays-whats-that-5a67f26aeac2
Fabian Sanglard: http://fabiensanglard.net/lightScattering/ Paul Twist: https://aerotwist.com/tutorials/creating-particles-with-three-js/

Des Holmes: https://codepen.io/desholmes/pen/KzRgEE?editors=0010

Ricardo Cabello: Threejs.org

Jerome Etienne: https://jeromeetienne.github.io/fireworks.js/

Steve Jenkins: https://www.gadgetdaily.xyz/code-animated-fireworks-with-three-js/

Rainner Lins: https://codepen.io/rainner/pen/LREdXd

Seb Lee-Delisle: http://creativejs.com/tutorials/creating-fireworks/index.html

Charlie Hoey: urlhttps://github.com/flimshaw/THREE.GPUParticleSystem

Angus Forbes: week4/w4_5_pointShader

ACKNOWLEDGMENTS

The authors would like to thank everyone who helped make this project almost a reality. We'd like to thank mr.doobs, the guy who made Three.js without whom all of this, while still possible, would have been a lot more work. Finally, we'd like to thank Angus Forbes and Lucas Ferreira for their excellent course and guidance throughout. Additionally, we'd like to continue to wonder why sph doesn't work on windows, and hope that that isn't too big a deal.