CS175: Sandbox: THREE.js Particle System

Michael Ge and Yuhan Tang

December 19, 2016

1 Abstract

WebGL is an up-and-coming technology that brings 3D graphics to the web. OpenGL for casual programmers is becoming a thing of the past with the advent of much more accessible web apps. For this project, we wanted to explore the aspects of WebGL to implement a particularly aesthetically pleasing effect: particles.

2 Research

To allow us to quickly start working with particles, we opted to bypass most of the clutter needed to set up a WebGL scene by using THREE.js. As of more recent builds of THREE.js, most of the built-in tools for particle systems have been deprecated. As a result, many of the projects you might find online are based on old THREE.js versions. Thus, over the course of this project, we had the chance to learn a graphical tool apart from OpenGL as well as explore some high-level ways of representing particles.

It is important to note that there is one GPU-based particle system created by Charlie Hoey that spawns and animates particles on the GPU. While clearly the better option, we couldn't find a reasonable particle system that runs on the CPU, so for pedagogical purposes, we proceeded with our implementation without looking at Hoey's implementation in great detail.

3 Attributing Code

First and foremost, we would like to note that we used the following libraries and references to speed up the extra baggage required to interact with our particle system:

- JQuery
- THREE.js
- dat-GUI: GUI for adjusting parameters
- OrbitControls: 3D camera motion
- Aerotwist Tutorials on getting a THREE.js scene started
- and lots and lots of documentation on THREE.js, HTML5 canvases, general Javascript know-how, and how to hook all this stuff up together.

Having sufficiently addressed the parts we didn't really do, let us now to turn what we actually did in this project.

4 Scene Setup

When creating the initial scene, we tried to mitigate the global namespace pollution by attaching all user-defined parameters to a single

`-`

. With this design in mind, the code is split into two scripts:

`main`

and

`particleSystem`

.

`main`

contains the general glue: it initializes default parameters, creates the 3D scene, starts the animation loop, and starts event listeners for user interface improvement.

`particleSystem`

contains our particleSystem class. The system is based on a geometry buffer that stores the positions of each particle and a

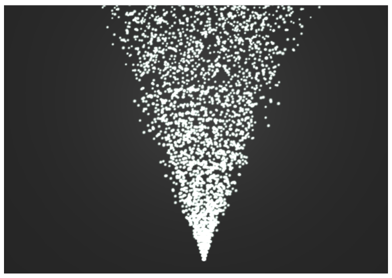
`THREE.PointsMaterial`

that texturizes each point.

5 Design

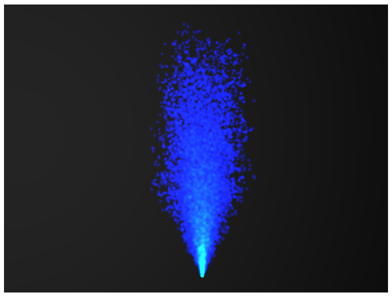
With the idea in mind, we now detail several iterations we went through while designing our code.

5.1 Basic System



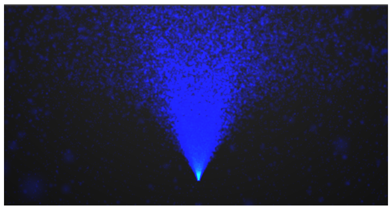
Our first iteration involved simply getting the particles to show up on the screen. Here, we used the simple Geometry object provided by THREE.js instead of the BufferGeometry since it lended itself so easily to the use of particles despite performance hits. Note that the system here involved naive particle displacement, sending each particle out along a vector whose coordinates are uniformly distributed in the unit cube. The result was a very blocky-looking particle system.

5.2 Customizable Color



We quickly grew bored of the black and white color scheme and tried to create THREE.js textures from HTML5 canvas elements. Many headaches later, the texture finally came out as a feathered particle that was fully customizable, but over time we began to notice a sizeable slowdown as the program continued to run. We are still not sure what causes the slowdown over time, but regardless we disabled this functionality in the final product.

5.3 Gravity

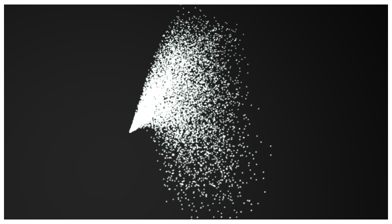


Next, we implemented basic Newtonian physics via a gravity simulation. Using the formula:

$$s(t) = s_0 + v_0 t + \frac{1}{2} a t^2$$

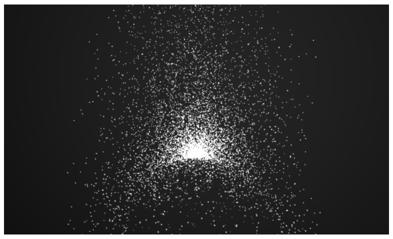
we were able to get a somewhat realistic arc motion for the particles. Still, many of the problems from the initial iteration remained.

5.4 Wind



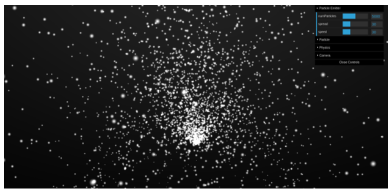
Implementing wind involved adding an extra $v_w t$ term to the formula.

5.5 Particle Life Randomness



Particle life was initially all identical, attributing to part of the awkward look. Once we randomized that, the project began to seem much more viable as a legitimate particle system.

5.6 GUI


numParticles 5000
spread 30
speed 30

Finally, we added the GUI interface to allow us to easily test and change the parameters.

5.7 Further Work

The project still can have more work done. Currently, we rely on the PointsMaterial which is extremely limiting, but due to the design of our program, using a material that allows shading would be a large overhaul in code design. And besides, Hoey's implementation probably already uses a shaderMaterial. Nevertheless, given more time, we'd like to look into using shaderMaterials for more flexible particles. Doing this would allow us to implement more algorithmically challenging topics like Perlin noise or multiple passes.