Outline

1. Project Summary
2. Building Instructions
3. Technical Accomplishments & Challenges
4. Further Extensions
5. Sources & Works Cited

Erik Godard & Jessica Wang

Project Summary

For our CS175 final project, we extended the CS175 Assignment framework to support a particle system simulation with textured particles. The particle data is encapsulated in a Particle class, and an emitter recycles these Particle objects while generating them. We designed a physical model of the particles that contains their position, velocity, and acceleration, and simulated the effect of gravity on the particles after they were shot upward from a given point. We used a preallocated vector of Particle objects to hold our data, and we used a Queue data structure to determine which particle object should be recycled to initialize new particles. The exact particle system created was aimed to look like a fountain. To achieve this we implemented the Material class to use a 3D normal texture of water. In addition to the texture, alpha blending was used to reflect the transparent nature of water.

Build Instructions

We developed our code on mac OS. To build our code, navigate to the directory with our files and type

```
make && ./project
```

Technical Accomplishments & Challenges

Particle System:
We implemented several key features to build the particle system. The first was a game loop. Previously, all of the assignments simple reacted to user input, so there was no need to keep a running loop that updated the screen at a set frequency. However, since we are now running a simulation, we need a loop that continually updates the screen. We chose to implement this using the glIdleFunc function. From what we read online, this was not an ideal solution for several reasons, but it worked well enough to fit our needs.

Before we ran the game loop, we allocated memory for all of the Particles and stored them in a vector. During each iteration of the game loop, we calculated each particles updated position based on the amount of time that had elapsed while setting the particle's alive property to false if it had fallen off the screen. In this loop, we also generated a number of new particles based on the amount of time that had elapsed. Once this update step was complete, we rendered the results to the screen by calling the draw function.

Material Integration:
After creating the initial particle system, we moved towards make the particle system look more aesthetically like a fountain. In order to integrate a water texture on the sphere particles, we implemented Material similar to Assignment 8. The first step to incorporating the 3D texture, key functionality of the particle system built off Assignment 3 had to be adapted, including adapting to RigTForm, SgRbtNode, and the Material framework itself. The texture was implemented through a pair of normal textures, both water images converted to ImageTextures in the initMaterials() function. The large and constantly changing number of particles makes it very difficult for it to be incorporated into a SgRbtNode format. Additionally there is no need to implement relational characteristics between particles similar to the robots. Thus the material

cannot be integrated within the initScene() function. Rather the material is called from the drawStuff() function as each particle is create.

While the texture added some aesthetic of water, it did not capture the transparent aspect of water. Thus we enabled the blending function of the Material type. After testing many options, a variation of alpha blending was determined to be the best.

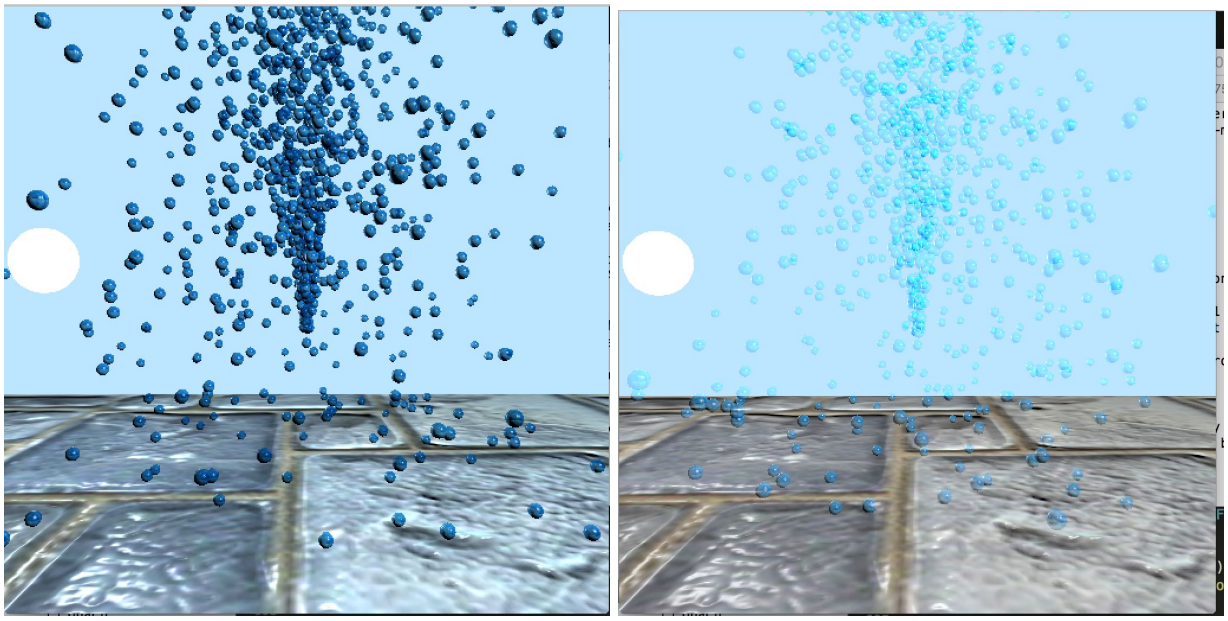

Figure: Particles with texture and particles with texture and alpha blending

Additional methods were attempted to make the particles look more like water including antialiasing, Gaussian blur and billboard but were unsuccessful as discussed later in this paper.

Optimizations:

We made several optimizations that made our code perform better. First, instead of allocating and freeing Particles throughout the lifetime of the program, we began by preallocating a certain number of particles (determined by MAX_PARTICLES) and storing them in a global vector. Each particle had a property to determine if it was alive or dead, and when we went to initialize a new particle, we would simply obtain a particle from this vector that was no longer alive and recycle it. This was initially a linear search through the vector to find suitable Particle to reuse, but another optimization that we made was to store the list of available indices in a queue. Pushing and popping onto this data structure is an O(1), which noticeably improved the performance of our code.

Challenges:

One of the biggest challenges in this project was simulating the particles as water. One of the biggest attempts was to modify the vertex and fragment shaders themselves to implement

Gaussian blur on the edge of each particle. While we found material online on how to implement Gaussian blur within a 2D texture we faced many difficulties in modifying the equation into our 3D simulation. Additionally, the large majority of resources we found had Gaussian blur within a single texture whereas we required Gaussian blur of an object with the background.

Another attempt we made to blur the edges of each particle was antialiasing. While antialiasing was implemented to make the edges of each particle softer and not as pixelated, it was unable to create a strong blurring effect that we desired (we hoped to achieve a "fuzzy" edge that can be seen from afar, similar to fire particles).

Further Extensions

There are several possible directions we would extend our project if we were given more time. For one, we would like to be able to increase the number of particles our system supports. Currently we can support around 3000-5000 before the frame rate begins to noticeable decline, likely because we are reusing the sphere drawing code from the previous assignments. There are likely better methods of drawing this many points than drawing each individually.

We would also like to experiment with different properties of the physical simulation. Right now we start each particle off with a random velocity vector and let gravity bring it to the ground. It might be interesting to experiment with other physical phenomenon such as fire rising or more involved mechanics.

Another area in which to experiment would be with our textures and blending. Right now we are attempting to simulate something resembling water, but there is certainly room for improving the accuracy of the simulation. We would also like to learn more about and improve our blending function. We are currently using one of the options of glBlendFunc, but we would like to experiment with writing a new fragment shader, such as one with Gaussian blur, for this. Other methods like billboards of a 2D faded out circle could also be effective.

Sources & Works Cited

---

http://buildnewgames.com/particle-systems/ - We read this link to get a basic overflow of how particle systems work, including the many components that make them up and can be

configured. The actual code from this tutorial wasn't useful, since it was targeted at 2D web demos, but it was useful for getting an introduction to the concepts.

http://www.bfilipek.com/2014/04/flexible-particle-system-start.html - This link was helpful for desrcribing how to structure the code in C++, namely how to ensure that each component only has one responsibility and that properties can be easily tweaked. The describes similar concepts such as Particle classes, separating Particle Data, Generators, and Emitters, but it does so in a C++ context. Much of the concepts discussed would be helpful for a large scale particle system but weren't immediately relevant to our smaller one.

http://www.glprogramming.com/red/chapter06.html - This link has documation and explaination about the mathmatical functions behind alpha blending. The chapter also includes a small portion on antialiasing. It additionally has many other variation of blending that could be utilized in other projects including depth of vision and polygon offset.

http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/ - This link was helpful about learning about Gaussian blur with recommendations on how to implement it.

http://gamedev.stackexchange.com/questions/8623/a-good-way-to-build-a-game-loop-in-opengl - This link was helpful in learning how game loops work in OpenGL. We ultimately went with the glIdleFunc option.