CS40 Final Project

Zoe Junghans, Jess Karol
Swarthmore College

ABSTRACT

Our Computer Graphics Final Project is an implementation of a particle system that attempts to model water. In essence, we have many small particles that are interacting dynamically with each other, the environment, and are under the influence of gravity. Our environment consists of a cube with transparent sides. Our graphic can be rotated in two different manners: camera rotation and the rotation of the cube. When the cube is rotated, (in the x, y, or z direction),the particles reflect this change and continue to act as they would under normal gravity by continuing to fall downward. The camera rotation simply acts as though a camera were rotating around the entire system. In order to achieve particle interaction using a large number of particles without sacrificing real time performance, we utilized an octree to help with particle to particle collision detection. The octree subdivides the cube and checks for collisions only between particles within the same region. This reduces our collision detection run time significantly. Unfortunately, the physics that determines our particles interaction is more representative of small bouncing balls. While we began thinking about and implementing more complex particle interactions, like water pressure and viscosity, further modifications to this project could be improving these features. Our project was successful in that we implemented a large particle system with fast run time and complicated particle interactions.

1. INTRODUCTION

Our motivations for this project include the desire to model a non-linear physical system, implement a dynamic particle animation in real time, and extend some of the topics we covered in class. While we discussed particle systems in class, we never had the opportunity to work with one extensively, so we saw this project as a means of exploring this topic in greater depth. Furthermore, we wanted to gain experience working with particle interactions.

Our primary goal in this project is to develop a realistic simulation of water particle movement. We have developed and implemented functionality that handles collision detection so that water particles can interact with each other and the environment. Our original goals were as follows. First, we hoped to implement a particle system that had fast inter-particle and particle-wall collision interaction. After we achieved this, we aimed to work on the physics behind how water particles might interact in such a system. We then hoped to model the behavior of water, and our final goal (and quite a reach) was to abstract away the particles by applying rending techniques to blend the particles together.

Timeline of Short Term Goals:

- Nov 12 - Nov 19: Hand in project proposal and obtain started code from the particle fountain example done in class.
- Nov 20 - Nov 26: Implement particles interacting with scene and possibly each other for a few particles.
- Nov 27 - Dec 3: Start working toward adding more particles while keeping run time reasonable.
- Dec 4 - Dec 10: Finish having many particles interact with each other. Start thinking about how to model water. Implement more advanced models of water and start figuring out how to make the water look more aqueous.
- Dec 10 - Dec 19: Finish up report and continue to improve on simulation.

Some challenges we expected to face included handling a large amount of particles while maintaining efficient runtime, handling particle-wall and particle-particle collisions, efficiencly maintaing attributes of individual particles, and difficulty applying complex physics and mathematical equations to better approximate the movement of water.

Particle systems are often used to depict fluid motion or other chaotic behavior because complicated non-linear dynamics can be simplified to individual particle interactions. Systems like smoke and water can be simulated using other rendering techniques; however, this is often much more complicated. We were interested in using the knowlege we already had about graphics to implement a somewhat realistic physical system. Finally, this project gave us the opportunity to create a relatively simple and efficient way to model a real world particle system that is relevant and understandable to people with some amount of graphics knowledge.

Course paper for Swarthmore College course CS 40: Computer Graphics

2. RELATED WORK

We used the particle fountain in-class example as our starter code. We also incorporated the cube in-class example as the foundation for our static environment. [2]

In order to improve runtime and help implement collision detection, an octree implementation found on github [4] and [5] was incorperated into our design.

We reviewed the following various research papers and open source libraries to find informaiton about particle movement, collision interaction, water molecule physics, and particle simulation animation technieques. To learn about the math behind how water particles behave we looked at two research papers and borowed technieques from both "Real-time Fluid Simulation Using Height Fields" [1] and "Particle-based Viscoelastic Fluid Simulation" [6]. We also looked at a google library called LiquidFun which discribes some techniques they used to model water pressure and viscosity [3].

3. DESIGN

The fundemental base of this design started with the fountain and cube in-class examples. Both of these systems merged together yeild a particle system and a static environment that interact with each other.

The first major design concept in this project was drawing different aspects of the same scene. Since we first brought together the fountain and cube in-class examples into one project, we needed to design our code to handle using two different shaders for the same project. To render a scene with static and dynamic components more easily, we split up our two shaders to render moving components (ie particles) and static components (ie box walls). The particle shader uses the kinematic information (i.e. position, velocity) while the cube shader draws simply draws the verticles of walls of the cube. When painting the scene, these shaders must be applied consecutivly to draw both elements for each motion update.

In order to have more control over the particles, we created our own particle class. This class handles updating the location of the particle based on the the particle's velocity and gravitational force. This class is influetial in handling particle-partilce collisions and is a helpfull layer of abstraction between the particle system and the individual particles. It is also a derived class of the octree library, which allows the octree implementation to use the particles in its own methods.

The next design component involved how to get the particles to be interactive rather than simply following the basic path set forth for them by basic influence of gravitation force without interaction with the environment. Equations of motion shown below:

$$v_f = v_o + \mu * g$$

$$p_f = p_o + V_f * dt$$

The imported octree data structure was influential at this point because it allowed the particles to be quickly located with respect to their relative locations to each other and within the static environment. The octree does this by subdividing the cube recursivly into eight octants and keeps track of which subdivisions a particle is located within.

The next design feature was implementing particle collision. At first, we implemented a means for the each particle to run collision detection with every other particle. This code was terribly inefficient and ran in $O(n^2)$ time. Once we learned about ocrees and incorperated one into our design, collision detection became much more efficient. The octree can triavially check for collisions between particles by only checking particles within each subdivision. The time complexity is now much less, but also much more difficult to define numerically.

Finally, wall collision is a major design concept we achieved. Wall collision is handled through a method that checks if a point is close to and moving toward a plane. Wall collisions must also preserve energy. In other words, when a ball drops from a certain height, the ball could bounce back to the same height if friction was not present. However, to make our design more realistic, a coefficient of friction for both ball collisions and wall collisions was added so that after some time the particles reach a steady state. By designing wal l collisions, particles can interact with any kind of flat surface (not just the walls of a cube). While the vertices of the cube are hard-coded into this program, one could specify the vertices of any three dimential shape and the system should adjust accordingly.

The design of wall collisions also made it quite simple for particles to react with a changing (rotating) environment. As mentioned above, the wall collisions are simply determined by the surfaces that make up the static environment. Thus, by applying rotation matrices to each of the vertices of the cube, the particles are able to interact with a changing angle and positoin. This design choice makes the particle interactions in this program applicable to many different environments.

Design outline:

- Obtain or implement a particle system and a static environment
- Paint the static environment and particle system using two different shaders
- Implement a particle class that maintains position, velocity and other necessary information
- Implement an octree and add all particles into this data structure
- Implement wall collision detection
- Modify the octree and the particle class for particle-particle collision detection
- For every time step, adjust each particle's position, check wall collision, update the octree which in turn checks for particle collisions, and update velocities and positions accordingly
- Have particles respond to rotations of the static environment

4. IMPLEMENTATION

Both Qt and OpenGL were essential to the implementation of this project. QT was used to create the GUI and animation, and all programming was completed in C++.

As mentioned in the design secion, two different shaders were needed in the implementation of this simulation. The first shader is used for the particle system and the second shader is used for the cube. While the vertex shaders for

each are very similar and include simple information about the position of points and the camera, the fragment shaders implementation are different. The fragment shader for the cube is a simple shader that only colors applies color, while the fragment shader for the particles is more complex and includes ambient, specular, and diffuse lighting. Two different paint functions were also implemented: one to deal with the static scene and one to deal with the particles.

Writing information to the VBO was another key element of our implementation that we were previously inexperienced with. Specifically, we had to store the position information for all the particles for a given time step to a temporary array and then write this informaiton to the VBO at once. Each time the positions were updated, the information sent to the VBO had to be overwritten. The shaders were then responsible for interpretting this information and drawing the scene.

The octree was a key element of our implementation because it greatly improved the runtime and performance. Octrees work by recursively subdividing the space into eight regions untill a specified depth isreached. We use a depth of five because we experimentally found this worked well with the number of particles we use. Octrees can be used in various graphics applications for spatial representaiton. Our application, and a common one for octrees, is collision detection. When we want to detect if a particle is in collision with any other particle, we check the boundaries of that particle with all other particles that exist within the same octree node because particles that exist within an octree node are close together. This allows us to dramatically reduce the amount of particles that are in possible collision with our current particle.

To find out if two particles within the same octree node are in collision, the following check is run:

$$if\ (P_1 - P_2)^2 < (4 * radius * radius)$$
$$and\ \Delta V \cdot \Delta R < 0\ then\ collision$$

If indeed two particles are in collision then the updated velocities are as follow:

$$V_1 = \mu[V_1 - 2 * \Delta R * (V_1 \cdot \Delta R)]$$
$$V_2 = \mu[V_2 - 2 * \Delta R * (V_2 \cdot \Delta R)]$$

$$P_1 = Particle\ 1$$
$$P_2 = Particle\ 2$$
$$\Delta R = Displacement$$
$$\mu = Friction$$

This velocity update is enegy perserving. A coefficient of friction is added to achieve a steady state if no rotation is applied to the cube as time goes to infinity.

The implementation for the octree imported from github uses a series of ISpatial classes. The octree itself inherits from an ISpatialStructure class. In addition, the octree contains ISpatialObjects. It was then necessary to designed the particle class to fit the specification for the ISpatialObject abstract class provided in the octree library. [4] Therefore, we implemented our particle class to inherit from the ISpatialObject class so that particles could be compatible with the octree implementation. The advantage of these ISpatial classes is that through the use of polymorphism , our particles could easily be adapted to the octree and ISpatial method of collision detection . In addition, this type of Octree implemtation provides modularity. For example, if one were to try to simulate a different type of physical system that perhaps has a different approach to velocity, position, and collisions a new type of particle class that also inherits from ISpatialObject could easily be switched in without having to change any of the octree methods.

The use of ISpatial objects also introduced an inconvienence in our implementation. Instead of using vec3's the ISpatial classes use another class called Vector3. Since it would have been tedious to either change all of the ISpatial implementation to use vec3s or to change all of the qt code to use Vector3s we ended up having to use some combination of both. Therefore in our particle class, we have representations of the particle's velocity and position in both the vec3 version and the Vector3 version. This implementation choice is not ideal but the best solution due to the time constraint.

The implementation of collision detection had to be adjusted within the octree and the particle class. By using the mathematical methods from the sources already mentioned, the VCheckCollision method in the particle class simulates an equal and opposite recation if any two particles collide. Some other mathematical models were incorporated into the changes in velocity to approximate certain aspects of water viscosity and pressure. The particle viscosity attempts to simulate the cohesivness of water by bleding two particle velocities upon collision. This detail can be futher explored in order to create a simulation in which all of the particles act more like a viscus fluid. The water pressure is a different approach to particle collisions in which a given particle's velocity is adjusted based on the number of particles it is colliding with, or in other words, the amount of pressure it is under. An instance of the octree was then created and the particle objects were added to the octree to keep track of their positions.

Wall collisions were implemented with simple vector math using the normal to the wall being checked, the particle's velocity, and position. The method onPlane determines whether a particle has hit a specified wall and if so, that particle's velocity is reversed to simulate a ball's deflection off of a surface.

The implementation of the cube rotation involved simple rotation matricies. This rotation matrix was then applied to all of the vertices of the cube. Since the vertices of each cube determine the norma l of each surface involved in wall collisions, the particles respond accordinly.

One limitation of our implementation is evident in the particle's response to the rotation of the cube. In order to keep the particles from all sinking down to the same level once they had fallen, collided, and almost come to rest, we chose to disable the force of gravity if the particle is in continuous collision with other particles. This creates the stacking or piling up effect. However, when the cube is rotated and the particles are expected to respond accordingly, this implementation choice makes them act "sticky". While in the real world, all of the particles would simply fall together, our implementation choice causes the particles on the top of the pile to be lifted off the others before the ones below can move again, created a rather sluggish or sticky effect. A better solution to the stacking effect may have been reached given more time.

Another limitation to our implementation is the realisticness of the simulation. We have implemented basic physics

equations so that the particles interact appropriatly when a collision occurs but this behavor is more analogous to a group of bouncing balls than water. To take this project further we would more accuratly measure how water molucules beneath and on the surface act. We could implement a feature that simulates surface tension and rippling effect when the water is disturbed.

A final limitation involves the wall collisions. Since the particles do not move continuously, but rather at certain time steps, it is unlikely that any given particle will exactly intersect with the plane of the static environment's walls. This presented a challenge and caused us to create a sort of buffer around the walls of the cube so that we could accurately detect wall-particle collisions. However, since each particle's velocity decreases exponentially due to friction, the particles that did not have enough time to escape this buffer within one time step would fall out of the cube. We then had to incorporate previous positions into our implementation to solve this problem and other glitches in collision detection. Our implementation now considers the direction the particles are moving. However, we believe a more accurate method could be found with more time.

Below are some screen shots of our simulation:

Figure 1: 1000 particles before collisions

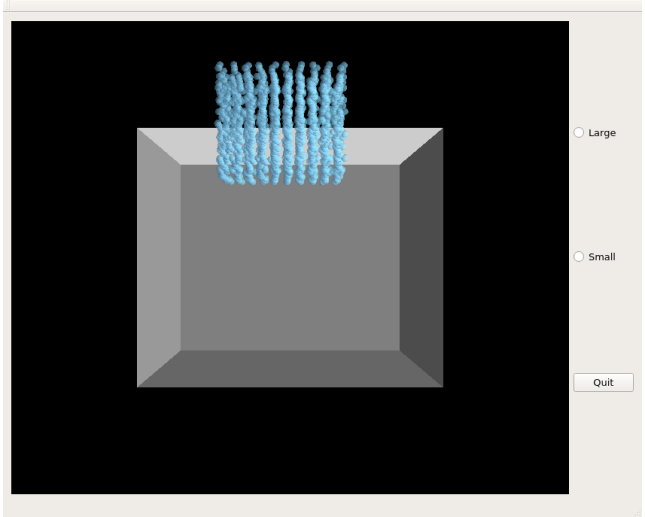


Figure 2: 1000 particles after collisions

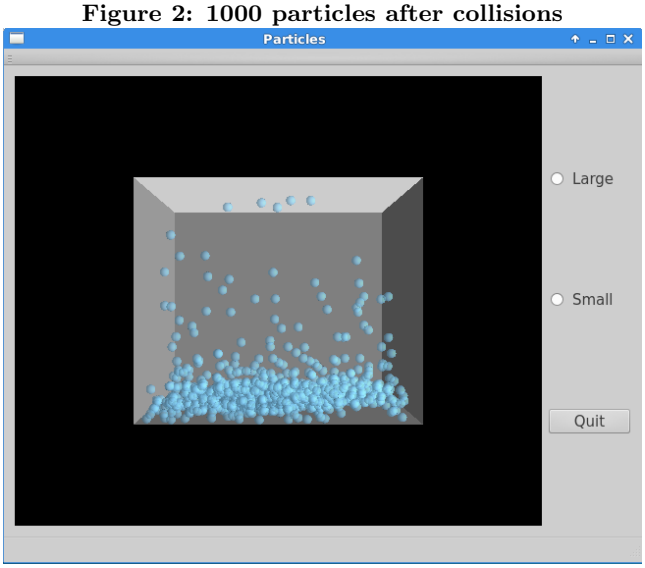
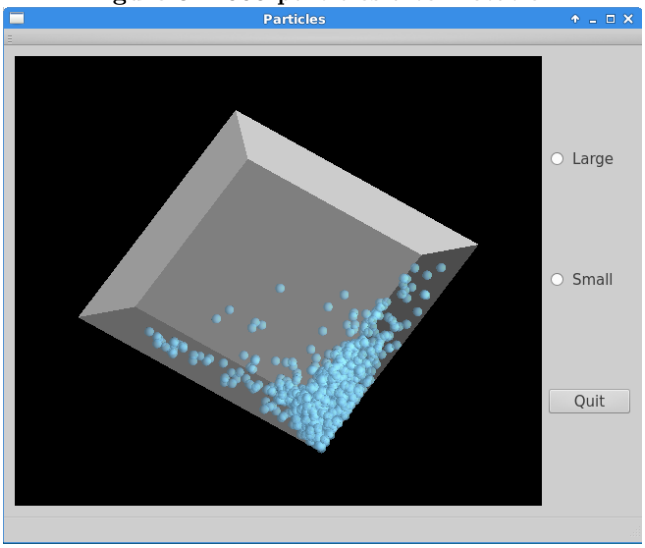


Figure 3: 1000 particles after rotation



5. EVALUATION

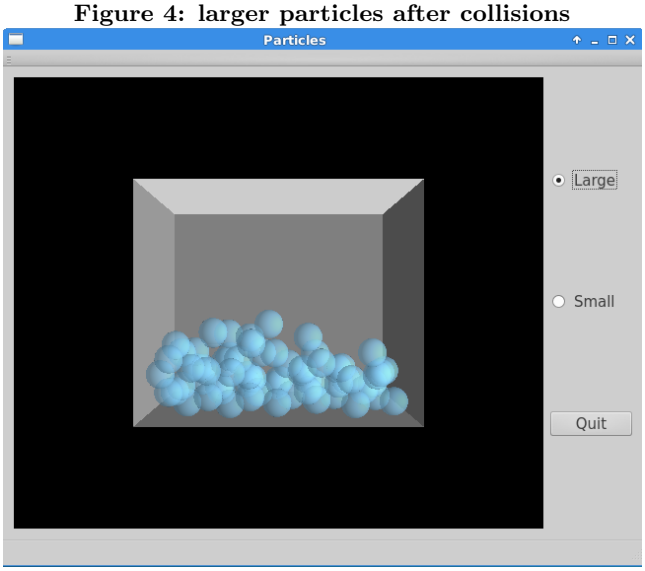
This project was an overall success.

Our final output consists of two different scenes. The first scene begins with a group of one thousand particles in the formation of a cube dropping into the tank. (see Figure 1) The particles then respond accordingly to particle-particle collisions and wall-particle collisions. If the 'r' key is pressed, one thousand more particles enter the scene from another dropped cube. The second scene shows fewer but much larger particles coming from a single source at a certain time displacement. This second scene makes the accuracy of the particle's behavior clearer to the viewer. In either scene the camera can be rotated using the 'x', 'y', and 'z' keys and the cube can be rotated using the 'a', 's', and 'd' keys. These various scenes and interactions provide evidence of a high level of understanding of different shaders, writing to VBOs, and particle interactions.

The metrics used to evaluate our success include the following:

- A simulation of a particle system that can run in real-time
- The ability to handle a large number of particles without sacrificing runtime
- Accurate particle interations and collisions
- Having a particle system interact with both a static and changing environment
- Increased understanding of particle systems, octrees, and collision detection
- Accurately modeling the movement of water

Figure 4: larger particles after collisions



A large measure of success was the ability to work with hundreds of particles in realtime. Because of the implementation of the octree described above, instead of having to check every particle against every other particle to determine collisions, any given particle had to be check against less than ten other particles. Specifically, our octree was implemented so that it could subdivide up to twenty (however depth 5 seems optimal) and a maximum of five particles can be stored in every node (octant). While checking every particle against every other particle would take *bigO* from $O(n^2)$ with linear particle collision detection to much less with octree collision detection.

However, there are still clear limitations to the number of particles that can interact without seeing the performance suffer. Our current implementation can handle approximately two thousand particles before it begins to get sluggish and get hung up. Given more time , it may be beneficial to try either different implementations of an octree or a different data structure altogether in order to obtain maximum performance.

Another clear measure of success was the accuracy of the particle and wall collisions. (The "large" mode of our final project shows these interactions most clearly.) These collisions model correct real-world physical behavior. In addition, the particles reacting to the rotating cube provide further evidence of a robust implementation of particle interations with the environment.

While originally, we were hoping to implement many more features of water dynamics, this goal proved to be too much given the time constraint. However, we managed to do some research on the properties of both pressure and viscosity of water and although they are not fully or correctly functioning, we have applied them to our particle interations. Although our goal of acurate water movement was not fully realized, the research done and knowledge gained is a sign of success. It is also a measure of our success that these more acurate models of water can easily be incorporated into our design by simply adding methods to the particle class or changing the particle collision equation found in the particle-particle collision method.

Thus, while more work and reasearch could be done, this project provides a solid, robust, and well designed basis on which to build a more complex water simulation.

6. CONCLUSION

In this project we successfully modeled a particle system which begins to approximate the movement of water. The particle system allows for particle-particle collisions as well as particle-wall collisions with the environment we created. The use of an octree allowed for large performance benefits and the ability to introduce a much larger number of particles into the system. Finally, the particles were made to respond correctly to transformations applied to the static environment.

This project greatly expanded our knowlege in the areas of both graphics and physical systems in general. First of all, we gained a lot of knowedge about vertex and fragment shaders. We learned how to send information to these shaders as well as get information out of them and used them in more complex ways than we had in previous projects. We also obtained a lot of knowlege about octrees, their structure, benefits, and implementation . Although we imported the octree implementation from github, we had to incorporate it into our design and change the implementation to better work with our particle collisions . The properties of water that were researched also gave us some understanding of the physics involved in this type of physical system. Finally, we have a solid understanding of particle systems. We have learned about their basic features such as intialization and drawing as well as more complex ideas such as particles interacting with each other and with the environment. By efficiently applying changes to single particles based on collisions we were able to model a much larger system.

Given the current status of the project, some next steps include:

- Changing the color of particles depending on their depth and collisions
- Applying more equations that model water-like movement (i.e. surface water movement, waves, etc.)- particularly look into the Navier-Stokes and Euler equations of fluid dynamics
- Making the particles react more fluidly to changes in their environment
- Further improving runtime with an increased number of particles
- Apply advanced rendering techniques to blend the individual particles together in order to make the visual output more similar to water

7. REFERENCES

[1] Balint Miklos. Real-time fluid simulation using height fields. 2004.
[2] A. Danner. Provided Starter Code and Helped During Office Hours.
[3] Google. Liquidfun. http://google.github.io/liquidfun.
[4] Ioan A Sucan. Flexible-collision-library. https://github.com/flexible-collision-library.

[5] Mykola Konyk. Spatial-collision-datastructures. https://github.com/ttvd/spatial-collision-datastructures.

[6] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based Viscoelastic Fluid Simulation. 2005.