Daniyal Latif
dal7782
Project A: Tornado of Squares, Fire, and Rope/Cloth Hybrid

User's Guide

Goals: My main objective was really just to complete the requirements with 4 enclosures and particle systems within them, but unfortunately I could not get flocking to work. Given that, I wanted to include a lot of optional features to compensate.

Instructions for interaction:
J and L to strafe left and right respectively, I and K to move forward and backward respectively, left and right arrow keys to turn the camera around the "glass cylinder," and up and down arrow keys to tilt the camera up and down. The on-screen buttons are the sole means of switching solvers, so please do use them. By pressing "c," you can turn on and off screen clearing, "s" will blow the spring system and make it resemble a sail or a cloth because of a rendering trick, and clicking and dragging on the screen while the "OldGood" solver is selected will allow you to manipulate the fire and tornado particle systems, although that feature only works from a certain angle and is flawed.

Code Guide

My structure probably shouldn't be too different from others since it is centered around the use of the VBObox code from Project C, which I imagine most used. PartSysBouncy14 includes the creation of the particle system objects and VBObox objects and their rendering on screen, which is essentially the same structure as the PartSysBouncy14 starter code. The biggest difference is that the shader program is not included in the file anymore, and has instead been moved to be a variable of the PartSys class in PartSys06. Basically, it's a default shader program now shared between two of my particle systems (tornado and spring rope) and includes a rendering trick. For Reeves fire, I overrided this program to get PART_AGE to change the color. The init() functions, applyForces(), and dotFinder()'s structure are the same as what they were. The differences that do exist are new boolean selectors like "this.isRope," and the shader strategy from Project C used within the init() functions. The dot2finder() function is new to calculate for the Verlet solver, though ultimately I couldn't get Verlet to work unfortunately. render() dynamically draws the particles on screen as expected, but it also includes a rendering trick for the spring rope that makes the rope look like a cloth or sail when it's blown. Finally, swap(), solver(), and doConstraints() aren't changed structurally either.

Results

In figures 1 and 2, you can see a rendering trick used to make little boxes inspired by Kelly Jiang's demo day video that I also ended up using with the tornado and spring rope/sail. Also, you can see the constraining box used on each particle system. Though it will be impossible to tell from stills, figure 1 is after just booting up the program with the default Euler explicit solver which is unable to keep the spring system (of 10 particles as per the instructions) stable, and figure two is after activating the implicit midpoint solver and waiting for a bit. There is also a rendering trick used to make the rope seem like a sail when you use a wind force on it but my keyboard is single-touch so I cannot activate it and use Snipping Tool on it at the same time. There's probably a way to do it, but to keep it simple please refer to my video for this second rendering trick. Furthermore, an anchoring constraint has been placed on the top of the rope, which is also inspired by Kelly Jiang's demo day video.

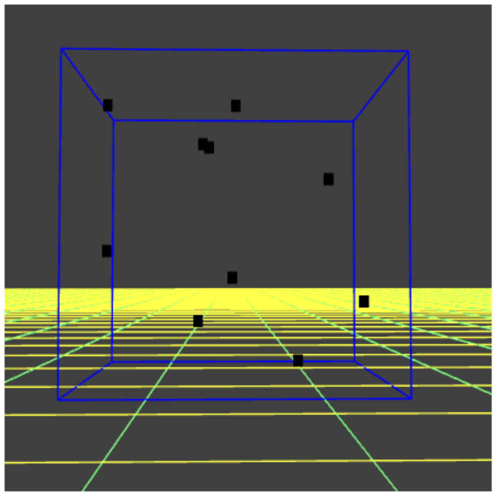


Figure 1: Spring system unstable with an explicit solver

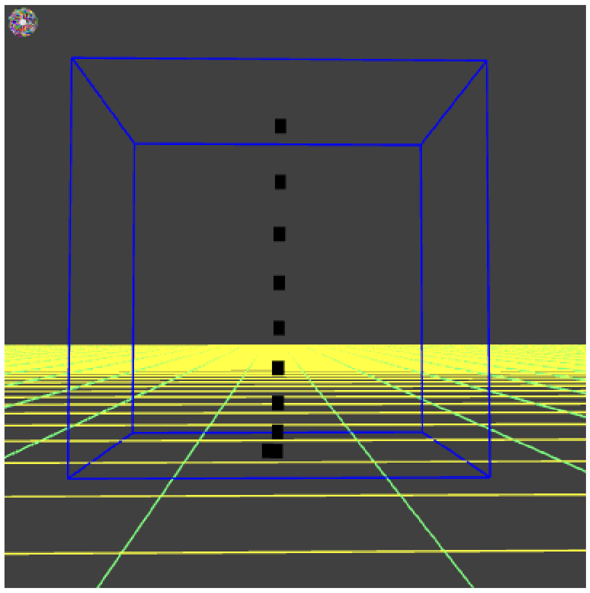


Figure 2: Spring system stable with implicit solver

In figure 3, you can see that I have turned the camera and moved to the Reeves fire particle system of 600 particles. Tricky to make out from a still, but it's a fountain that spews from around the center of the box (where the brightest red particles are) and with age it gets dimmer like ash, which you can see easily on the bottom. Below it are a sphere and box that have collision detection.

In figure 4, you can see a tornado with 500 particles that has turned into a sort of conical shape as a result of a force field.

In both, you can see a small draggable cursor to manipulate each system but due to the single-touch issue that will be demonstrated in the video. Of course, camera controls and the high framerate of my program will also have to be shown in video form.

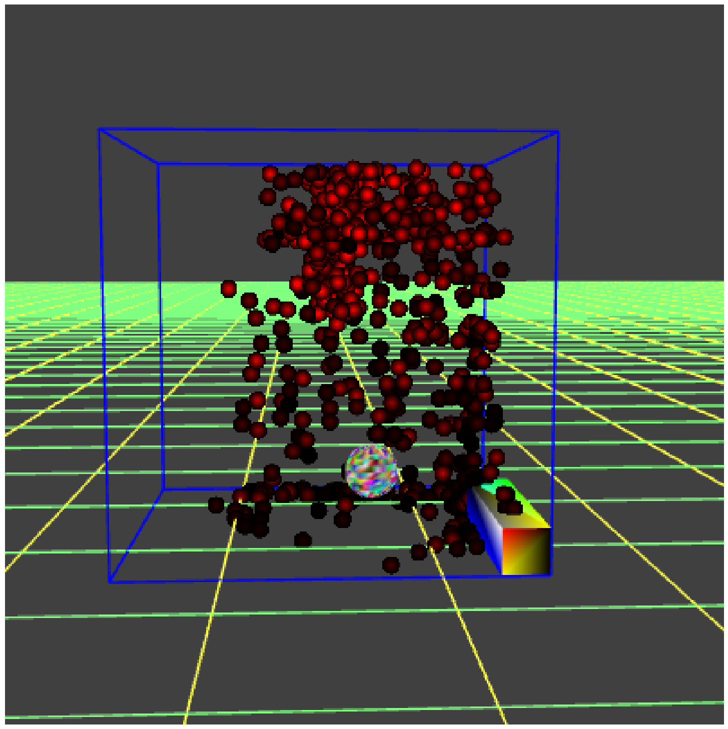

,

Figure 3: Reeves fire, two additional constraint shapes, and the spherical mouse cursor for manipulating it

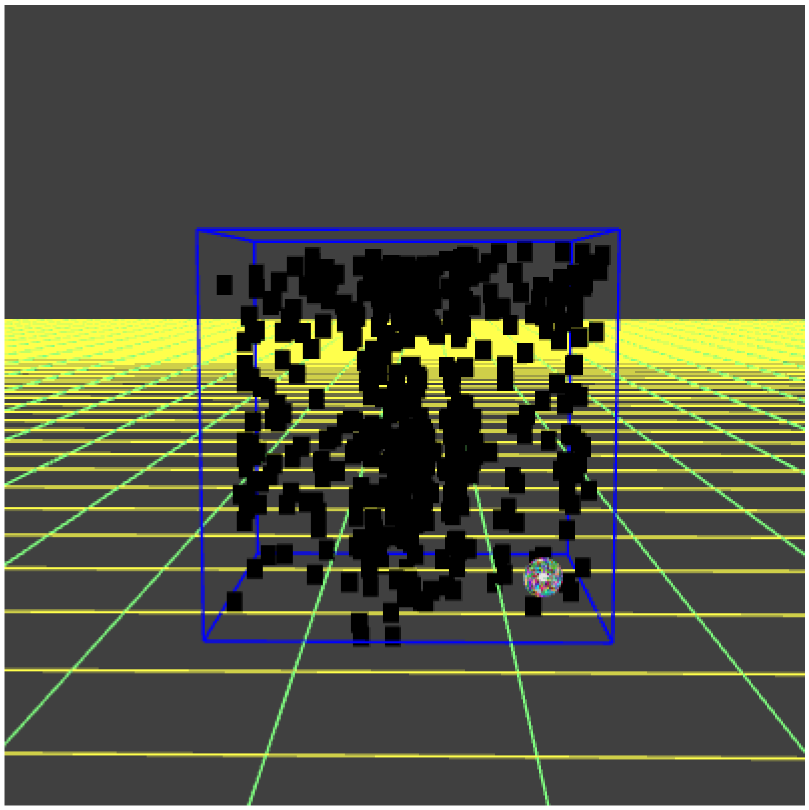
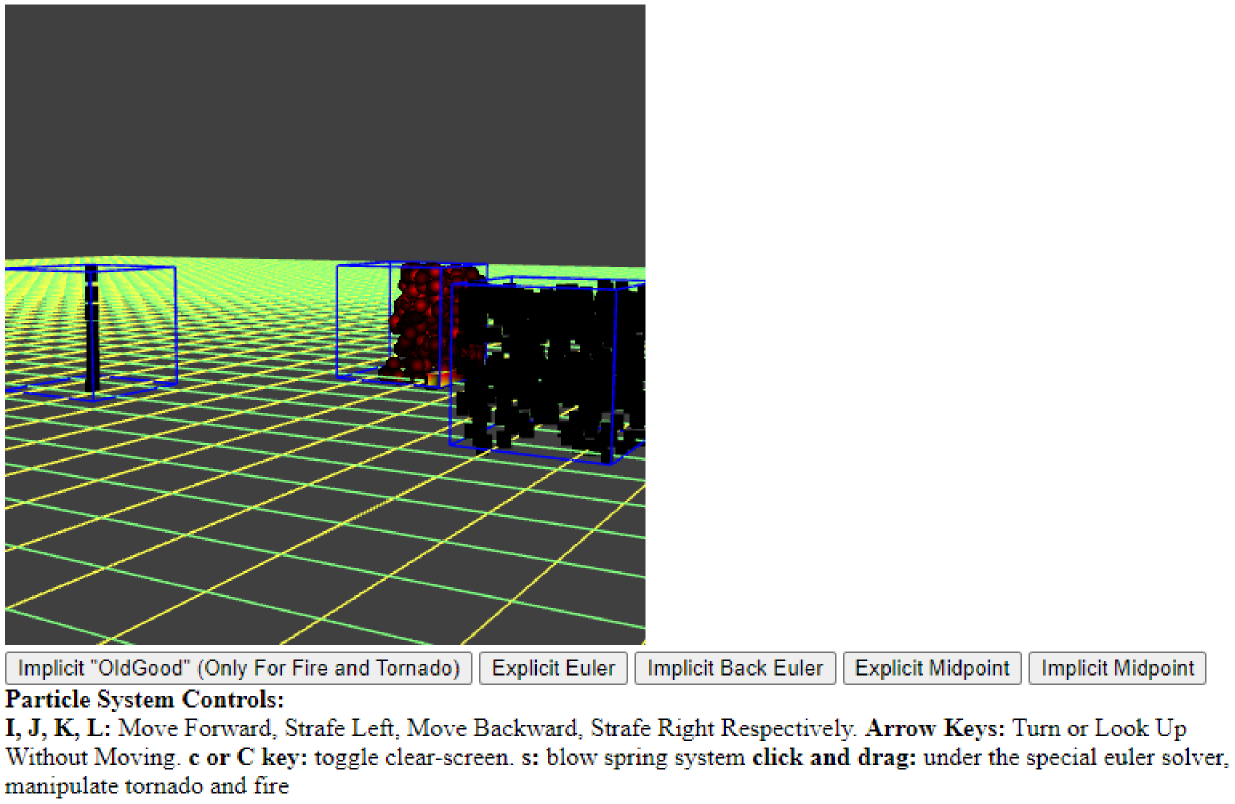


Figure 4: Tornado and the spherical mouse cursor for manipulating it and Reeves fire



Particle System Controls:
I, J, K, L: Move Forward, Strafe Left, Move Backward, Strafe Right Respectively. **Arrow Keys:** Turn or Look Up Without Moving. **c or C key:** toggle clear-screen. **s:** blow spring system **click and drag:** under the special euler solver, manipulate tornado and fire

Figure 5: Full program shot with instructions and user controls included.

In figure 5, you can see sensible on screen instructions as well as the three additional shaders I was able to add.I tried to implement the Verlet solvers as well but unfortunately those didn't work out.

OPTIONALS LIST:
All requirements except for the flocking system are completed.
As for optionals, I have:

- A rendering trick that turns particles into a square shape in the tornado and spring rope systems
- A rendering trick to make the rope look like a cloth or sail when blown
- A new wind force
- An anchoring constraint on the spring rope
- Explicit Midpoint solver added
- Implicit Midpoint solver added
- Implicit Back Euler solver added

- Potentially my flawed cursor manipulations of the fire and tornado systems, though I mainly used them to test my constraints and I understand if they don’t suffice