CSCE 645 Research Project Update 1

"Interesting Interactions with Granular Material"

Gabriel Stella

1 Summary

The single goal I had for the first update was the following:

> implement a real-time 2D sand simulation environment using a rigid-body polysphere system, with support for some form of user interaction with the simulation

Technically, I have *almost* met this. My polysphere implementation has some... bugs... so I decided to simplify and start with a basic sphere simulator. Unfortunately, it ended up being really interesting and fun to play with, so I got a bit distracted. It really is fascinating how such interesting behavior can arise just from solving force equations (in this case, normal forces for bouncy repulsion and tangent forces for friction). This simulation is up on my website, under `Toys::Circles`. Particles are rendered such that their rotation is visible, and clicking spawns a new particle with random rotation under the cursor. There is also color-based particle energy visualization, which allows the user to watch energy dissipate and diffuse during collisions. Fascinating[1]! The simulator follows rules from the paper "Particle-based simulation of granular material", which I had originally decided would be the main starting point for this project. In particular, the parameters (which can be modified in real-time in my simulation via the GUI at the top right of the screen) are:

- k_d: a coefficient that determines the dissipation of kinetic energy during collisions
- k_r: a coefficient that determines the rigidity of the particles
- u: coefficient of friction (slows rolling particles)
- α and β: control how particle overlap affects the normal force formula

Though theoretically $u \in [0, 1]$, and other parameters should be positive, I allow the user to set them slightly negative *just for fun*. It's interesting to watch the energy of the system increase if you set $u < 0$; any slight spin on a particle quickly spreads to its neighbors. Of course, be cautious with these settings; with great power comes great responsibility, or something like that.

This may seem like a very basic fact, but it's also absolutely amazing to me that from just a few simple formulas:

$$f_n = -(k_d \zeta^\alpha * \zeta' + k_r \zeta^\beta) \qquad \zeta \text{ is particle overlap} \tag{1}$$

$$F_n = f_n \vec{N} \qquad \vec{N} \text{ is line between particle centers} \tag{2}$$

$$F_t = u f_n \hat{V}_t \qquad \hat{V}_t \text{ is normalized relative tangent velocity} \tag{3}$$

we get complex behavior like particles rolling over each other.

[1] It's even better if you set ColorCold=#000000.

I'm glad to have started with a little web demo, especially since it can be easily shared and experienced by others, but ideally I would like to create a C++ desktop application. That was the original plan, and I was hoping my new computer parts would arrive in time to run some beefy sand simulations; unfortunately, there were some delays. Luckily, though, the last part finally came in today, so I'll be able to put it together soon, and then we'll see what *real* sand looks like. I mean, real *fake* sand. It'll be cool. Plus, when using C++ rather than *Javascript*, I'll be able to write a much more sophisticated program. I have the following features planned:

1. Polysphere particles with various grain types (triangle, square, 2x1, 3x1, ...) and sizes
2. Space grid to accelerate computations
3. Using the space grid to do compression
    - When a grid square is "stable", i.e. particles aren't moving in/out, *and all of its neighbors are stable*, it can become "inactive" - to save computation, all of the particles inside "disappear" and it simply remembers what was there
        - The average energy of constituent particles can be stored as "temperature"
    - Grid squares can also combine dynamically (like an octree)
    - Some rule must be added to prevent over-compression; e.g., when a stable pile forms, the system shouldn't compress and combine all grid squares into one
        - This could have something to do with the quantity (absolute or relative) of particles in each grid cell
4. Better numerical integration using Runge-Kutta methods or something (I don't know much about this, so it'll take time to learn)

2 Planning

In the original plan, the next step was to integrate fluid coupling into the simulation. I think I'd rather focus on these other elements:

1. Implementation efficiency
2. User interaction support
3. Representation compression

This is essentially swapping the place of the last two update plans. I'm also not sure if I'll have fluids be the last goal; there are other interesting options, such as rigid-body coupling, that I could pursue. On the other hand, the reason I originally put fluid coupling at the second update was because I *really* wanted to be able to build sandcastle-like structures. I think we'll see when we get there. Reimplementation in C++ will likely take some time by itself, since I'll now be using raw OpenGL (so the GUI, text rendering, interaction, etc will be a bit of a pain; luckily I already have my own basic GUI library that I can use to get started), so it's entirely possible that I won't finish experimenting with compression by the next update. Even if that is the case, if the project ends up being just a good implementation with *some sort* of method for computation reduction, I would be satisfied; the real key to me is that it should be a real-time simulation that allows user interaction.