Journal Report 1

9/2/19-9/6/19
Mikail Khan
Computer Systems Research Lab
Period 5, White

Daily Log

Monday September 2

I tried to directly apply gravity to my rotational/translational simulator by adding a downward velocity (as opposed to a force). It didn't work very well and made energy increase a lot between collisions, making it pretty funny to watch. I read a wikipedia article about modeling collisions through impulses, which is what I'm doing, but it has a slightly different implementation which I think will make things a lot easier.

Wednesday September 4

I tried implementing impulse based collisions as described by the wikipedia article, but I was getting some weird results and I didn't completely understand the math. I looked at two of the sources of the article though and they explained it a bit better so I got it to work. There were still some problems with applying gravity but having a distinction between using impulses as instantaneous forces and forces that are actually affected by the timestep helped. Something I noticed is that lower elasticities don't decrease the energy of the system, so I added a simple velocity *= elasticity after each collision but based purely on my own expectations it seems to have too strong of an effect.

Friday September 6

I worked a bit more with how impulses are applied vs how forces are applied. I got gravity to mostly work except that objects resting on the ground jitter around a lot, so I decided to make a distinction between collisions and resting forces. It seems to work pretty well but how I'm deciding between an impulse collision and a resting force is kind of arbitrary.

Timeline

Date	Goal	Met
Today minus 2 weeks	Get elastic translational collisions working	Yes, but it turns out that I can't really reuse much code for the rotational part of it
Today minus 1 weeks	Rotational elastic collisions	Mostly, I got it working that weekend
Today	Persistent collisions and gravity/friction	Persistent collisions and gravity, but not friction
Today plus 1 week	Functional 2D mechanics with basic input	Should be pretty fun
Today plus 2 weeks	More complete GUI, easy to use, but not with FBDs, graphs, etc.	It looks the milestone's I've set in these reports are way ahead of the ones I put in the proposal. I'd like to refactor/clean up a bit.

Reflection

In narrative style, talk about your work this week. Successes, failures, changes to timeline, goals. This should also include concrete data, e.g. snippets of code, screenshots, output, analysis, graphs, etc.

Even though I didn't quite meet my goal of adding friction, I think I was pretty successful. Honestly I had to rewrite a lot of the collisions stuff I did last week but it still helped a lot. I didn't realize how far ahead I was compared to the timeline I had made before, so I think I'll spend some time refactoring and generally cleaning up my code since it's pretty messy right now. I also have a lot of time to work on the UI and graphs. I would like to spend some time optimizing it, but I'm not sure what improvements I can realistically make.

The simulation itself has become presentable I think. It's hard to know if it's 100% accurate, but everything looks fine and I think I'll try to make it into more modular framework outside of class so that I can use it for games.

I don't think screenshots would be very informative and I don't have any graphs yet, but here's my code that calculates the magnitude of the impulse. It's in Rust.

```
let jr_mag: f64 = {
      let res_vel = -(1.0 + elasticity) * vr;
      let numerator = na::Matrix::dot(&res_vel, &normal);

      let mut denominator = {
          let m_inv_sum = 1.0/m1 + 1.0/m2;

          // radius cross normal z component
          // r x n
          // dbg!(r1, r2);
```

```
            let ang_z_comp1 = (r1.x * normal.y - r1.y * normal.x) / I1;
            let ang_z_comp2 = (r2.x * normal.y - r2.y * normal.x) / I2;
            // dbg!(ang_z_comp1, ang_z_comp2);

            //the above crossed with r again
            // (r x n) x r
            let z_comp_moment_1 = Vector::new(-r1.y * ang_z_comp1, r1.x * ang_z
            let z_comp_moment_2 = Vector::new(-r2.y * ang_z_comp2, r2.x * ang_z
            // dbg!(z_comp_moment_1, z_comp_moment_2);

            let ang_sum = z_comp_moment_1 + z_comp_moment_2;

            let ang_dot_n = na::Matrix::dot(&ang_sum, &normal);
            ang_dot_n + m_inv_sum
        };

        numerator/denominator
    }.abs();
```

This is how I initialize objects:

```
  let obj_data = [
     ObjectData::Rectangle([10.0, 0.], [0., 0.], 19.5, 0.2).with_mass(INFINITY),
     ObjectData::Rectangle([10.0, 20.], [0., 0.], 19.5, 0.2).with_mass(INFINITY)
     ObjectData::Rectangle([0.0, 10.], [0., 0.], 0.2, 19.5).with_mass(INFINITY),
     ObjectData::Rectangle([20.0, 10.], [0., 0.], 0.2, 19.5).with_mass(INFINITY)
     // ObjectData::Circle([0.0, 0.0], [0., 0.], 1.),
     // ObjectData::Circle([18., 10.], [-2., 0.], 1.),
     // ObjectData::Square([0., 10.], [0.5, 0.], 1.5).rotate(PI/4.),
     // ObjectData::Square([18., 13.], [-0.5, 0.], 1.).rotate(PI/4.),
     // ObjectData::Circle([18., 18.], [-0.5, -0.2], 1.0),
     // ObjectData::Circle([0., 12.], [0.2, 0.1], 0.5),
     ObjectData::Circle([2.5, 15.], [0.12, 0.0], 1.0).with_mass(15.0),
     ObjectData::Rectangle([5., 12.0], [0.1, 0.2], 1.0, 1.0).with_mass(6.0).with
     ObjectData::Rectangle([15.0, 15.0], [-0.25, -0.0], 2.0, 2.0).with_mass(25.0
     // ObjectData::Rectangle([5.0, 5.], [-0.5, 0.], 1.0, 2.25).rotate(PI / 5.).
     // ObjectData::Rectangle([15.0, 11.], [-0.5, 0.0], 1.0, 1.25).with_mass(12.
     ObjectData::Rectangle([3.0, 5.], [0.0, -0.1], 1.0, 2.25).with_mass(15.0).ro
     // ObjectData::Circle([18., 11.0], [-0.85, 0.0], 0.5),
     // ObjectData::Convex([10.0, 10.], [-2., 0.], &points).rotate(PI / 3.),
 ];
```