**Real-Time Balloon Simulation** 🎈

Annie Pa Michael Kelly

**Abstract**

We developed a real-time balloon simulation application using a mass-spring simulation and explicit Euler integrator. Our application reads in a user-specified quad mesh obj file and generates a balloon with structural, shear, and flexion springs between the vertices of the balloon. Our application also allows the user to inflate and deflate the balloon in real time. Optionally, the user can apply a vertex position correction algorithm to the vertices to constrain the balloon’s shape. Additionally, we have implemented simple sphere-balloon collision detection so that the user can throw spheres at the balloon.

**Introduction**

Who the fuck knows lol

**Previous Work**

Some bullshit

**Technical Challenges**

Since we wanted our application to generate balloons out of arbitrary quad meshes, we could not make any assumptions about the topology of our balloons, specifically the valence of each vertex. As a result, we needed to write a .obj parser that would generate springs between an arbitrary number of faces.

Additionally, we had starter code that simulated cloth with Provot correction. However, this code assumed that cloth particles were laid out in a 2D grid and that the number of springs attached to each particle was the same for each particle. This meant we needed to do a bit of code refactoring to get the starter code to work.

Next, we had to determine how to simulate balloon inflation forces and what (if any) particle position correction should be applied to our balloon particles at each timestep.

Finally, we wanted the ability to throw spheres at our balloons, so we needed to implement collision detection and resolution.

**The Balloon Data Structure**

**Spring Generation**

**Particle Simulation**

**Particle Correction**

**Collision Detection and Resolution**

After new particle positions, velocities, and accelerations are calculated, and particle correction is optionally applied, we detect and resolve collisions. Our application currently only handles balloon-sphere collisions, since this collision type is relatively inexpensive. Thus, our collision detection algorithm is straightforward.

for each sphere s:

for each balloon particle p:

if distance(p.position, s.center) < s.radius:

Move p outside the sphere

Apply a penalty force to s

Update acceleration, velocity, and position for s

When moving balloon particles outside of a sphere, we move it along the direction from the sphere’s center to the particles position. The penalty force is applied in the opposite direction of the particle’s movement and is proportional to how inflated the balloon is. While this penalty force calculation is not physically accurate, it does give convincing results, especially for a real-time application. By varying the mass of a collision sphere and the inflation of a balloon, we can achieve different collision scenarios.

**INSERT BOWL BALLOON FIGURES**

**Examples**

**Discussion**

**Conclusion**

**Special Thanks**

We would like to thank Professor Barb Cutler for her mentorship and for providing starter code for our application.