COS426 SPRING 2019

CLOTH SIMULATION

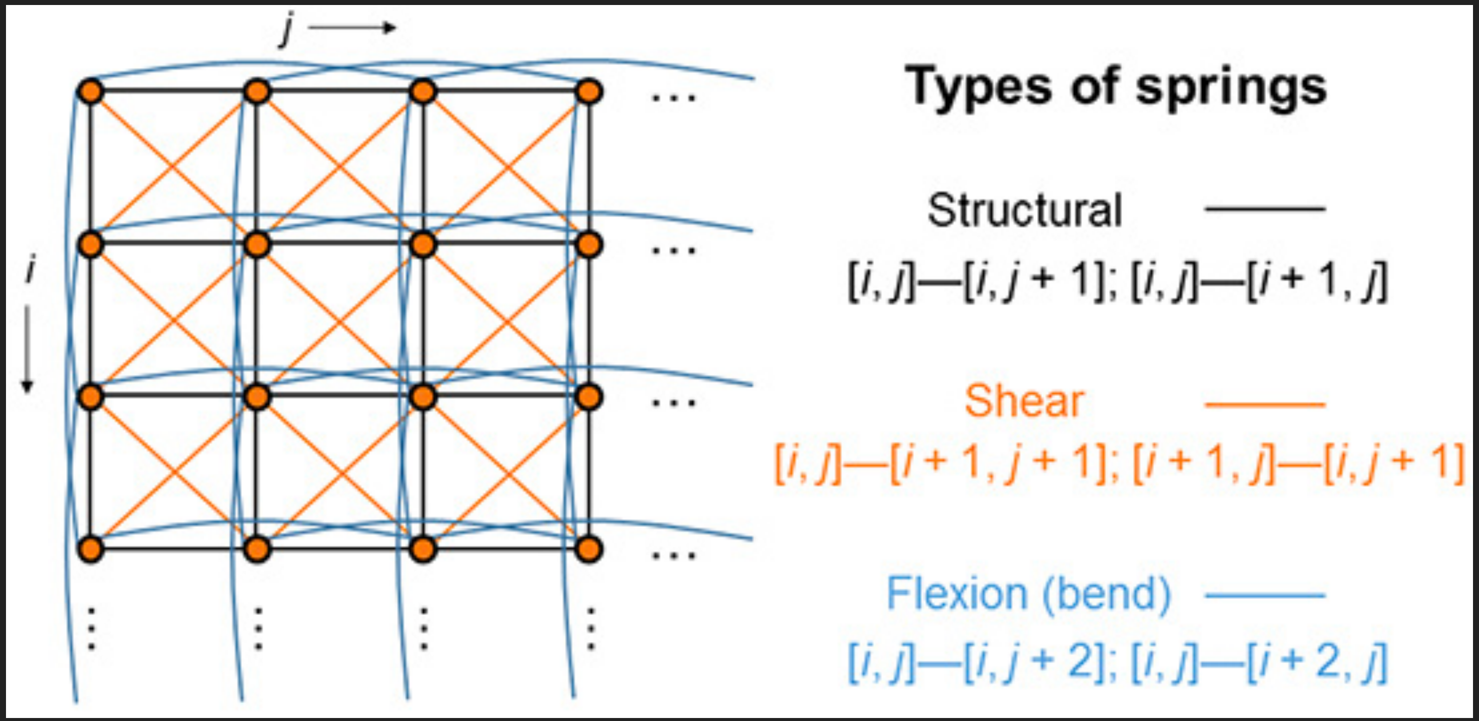
Austin Le & Jiaqi Su

PHYSICALLY-BASED CLOTH SIMULATION

- **MAIN IDEA**
  - Represent cloth discretely as a grid of point masses connected by springs
- In A5, each point mass is a single particle in the particle system
  - Each point mass is affected by forces in the system
- In A5, each spring is a constraint on our particle system that holds the point masses together

3 TYPES OF SPRINGS (CONSTRAINTS)

- Structural
  - 1-away neighbors in row and column
- Shear
  - 1-away diagonal neighbors
- Bending
  - 2-away neighbors in row and column


j
i
Types of springs
Structural
[i, j]—[i, j + 1]; [i, j]—[i + 1, j]
Shear
[i, j]—[i + 1, j + 1]; [i + 1, j]—[i, j + 1]
Flexion (bend)
[i, j]—[i, j + 2]; [i, j]—[i + 2, j]

SIMULATION LOOP — AT A HIGH LEVEL

1. Accumulate forces acting on each particle (e.g. gravity)
2. Solve Newton's equations of motion (by numerical integration) to compute new positions for each particle
3. Handle collisions
4. Enforce constraints
5. Repeat from Step 1

1. ACCUMULATE FORCES

- Each particle experiences some net force at every instant in time
- In A5, we have gravity and possibly wind forces
  - For each particle, simply add up all force vectors acting on it into a single net force
- Each particle can also be affected by spring forces (Hooke's law) from nearby particles, but we omit this in A5

2. SOLVE EQUATIONS OF MOTION — NUMERICAL INTEGRATION

- Many choices available:
  - Explicit Euler
  - Implicit Euler
  - **Verlet — good numerical stability, simple to implement**
  - Midpoint
  - Runge-Kutta
  - And more!

2. VERLET INTEGRATION

- If we use a very small timestep **`dt`**, we can assume constant acceleration and velocity for the equations of motion
- Then, new position at time **`t + dt`** can be calculated from the old previous position at time **`t`** using the following equation, where **`v_t * dt`** is approximated by the change in position relative to the last timestep

$$x_{t+dt} = x_t + (1 - D) * v_t * dt + a_t * dt^2$$

2. TIMESTEP TRADEOFFS

- Small timesteps provide greater stability and accuracy, but require more steps of the simulation (i.e. your simulation can be very slow) to achieve the same end results
- Large timesteps will require less work and fewer steps of the simulation (i.e. your simulation will just run faster), but are prone to error
  - Timesteps that are too large may never find a “resting state”

3. HANDLE COLLISIONS

- Particles may collide with other objects (or even other particles in the same cloth!)
- Detect collisions in 3D space and apply a correction or repelling force (more physically accurate)
    - In A5, we will apply a correction and simulate some friction to still get visually plausible results

3. HANDLE COLLISIONS — FLOOR

- Assume infinite plane with cloth above it
  - Perform simple “hack” of pushing particle back to surface of the floor if it goes under

3. HANDLE COLLISIONS — SPHERE

- Suppose that at time **`t`**, the particle is just barely outside of the sphere
- Suppose also that at time **`t + dt`**, the particle is now just barely inside the sphere
  - Collision!
- <u>If there is no friction</u>, then project the particle's position back to the closest point on the sphere's surface, called **`posNoFriction`**.

3. HANDLE COLLISIONS — SPHERE

- If there is friction **F**, then we want to simulate the particle "clinging onto" the sphere that it is in contact with, especially when it is moving.
  - Adjust the particle's previous position by the same motion that the sphere made in the last timestep to get a new **`posFriction`**.
- New particle position is weighted sum:
  - **`posFriction * F + posNoFriction * (1 - F)`**

3. HANDLE COLLISIONS — BOX

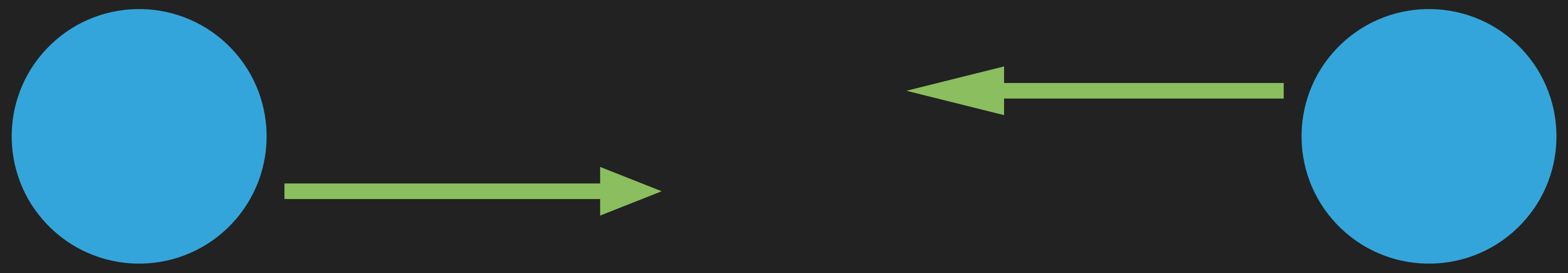
- Same idea as sphere!
- Main difference is in finding the closest point on the surface of the box to the particle's position.
- In A5, a **`boundingBox`** variable is given, which is a Three.js **`Box3`** object.
  - You can get its **`min`** and **`max`** to help you in finding the closest point through a series of if/else logic.
  - No need for complicated math!

3. HANDLE COLLISIONS — SELF

- In A5, you can handle self-collisions as a non-required extension
- Core idea:
  - For each pair of particles in the cloth…
    - If they are too close (closer than rest distance), apply a correction that pulls/pushes them both back towards to the desired rest distance.
  - Very similar to how you enforce the constraints!

4. ENFORCE CONSTRAINTS

- Each constraint (spring) tries to keep the particles (point masses) on either end together at roughly their natural rest distance.
- At each timestep in the simulation, apply a "correction" to both particles to bring them closer to their rest distance.



EXTENSIONS

- Time-varying, sinusoidal wind
  - `A[cos/sin](wt) + C`
  - Wind = `s(t) * <f(t), g(t), h(t)>`
- Handle self-intersections
  - Naive `O(n^2)` solution: apply correction to every pair of particles that are too close to each other
  - More optimized solutions include **spatial hashing** for faster search of which particles might be too close to each other

EXTENSIONS

- General Plane Collisions
  - Floor collisions are a bit of a "hack" and not entirely robust to clipping
  - Consider general plane equation **`dot(P, N) + D`**
  - Implement collisions with plane and account for friction
    - Should be very similar to interacting with one side of a box

EXTENSIONS

- New Objects
  - Add support for collisions with something other than a sphere, box, or plane (floor)
- Custom Scene
  - Put together an interesting scene in which a cloth interacts with multiple other objects (sphere, box, etc.)

EXTENSIONS

- Textures
  - Use Three.js's libraries to apply some textures to the objects in the scene to make it more exciting
- Your own idea!

QUESTIONS?