Amittai

COSC 89.18 Final Project: Position Based Dynamics

For my final project, I implemented the position-based dynamics simulator discussed by Muller et al. in their paper "Position-Based Dynamics for Rigid Bodies".

In the paper, the researchers describe a method for simulating the dynamics of particle-based objects using positions and constraints instead of the mass-spring model.

An advantage of this position-based model is better controllability — insdeed, while the mass-spring model we implemented in A1 earlier in the term required significant parameter tuning, the position-based model worked almost out-of-the-box.

Model Discussion

1. Mathematical Model We represent a dynamic object by a set of N vertices and M constraints.

A vertex $i \in [1, ..., N]$ has a mass m_i. a position x_i and a velocity v_i.

A constraint $j \in j[1, ..., M]$ consists of:

- a cardinality n_j
- a function $C_j : \mathbb{R}_{3_{nj}} \to \mathbb{R}$
- a set of indices $\{i_1, \ldots i_{nj}$, $i_k \in [1, ...N]$
- a stiffness parameter $k_j \in [0...1]$
- a **type** of either **equality** or **inequality**

Constraint j with type **equality** is satisfied if $C_j(x_{i_1}, \ldots, x_{i_{n_j}}) = 0$.

Constraint j with type **inequality** is satisfied if $C_j(x_{i_1}, \ldots, x_{i_{n_j}}) \geq 0$.

The stiffness parameter k_j defines the strength of the constraint j in a range from zero to one.

2. Numerical Algorithm Based on the above model, at timestep Δt, the dynamic object is simulated as follows:

```
forall vertices i
  initialize xi, vi, wi
endfor
loop
  forall vertices i do update vi with external Forces
  dampVelocities(v1,...,vN)
  forall vertices i do pi = xi +timestep * vi
  forall vertices i do
    generateCollisionConstraints(xi to pi)
  loop solverIterations times
    projectConstraints(C1,...,CM+Mcoll ,p1,...,pN)
  endloop
  forall vertices i
    vi = (pi - xi) / timestep
    xi = pi
  endfor
  velocityUpdate(v1,...,vN)
endloop
```

3. Code Implementation

- **Main simulation loop:**

```cpp
void AdvancePBD(double dt) {
  ApplyForce(dt);
  DampVelocities();
  Project(next_positions, dt);
  auto collision_constraints = GenerateCollisionConstraints();
  ProjectCollisionConstraints2(collision_constraints, next_positions, dt);
  ProjectDistanceConstraints(next_positions, dt);
  EnforceBoundaryConditions();
  UpdateParticles(next_positions, dt);
  ResetForces();
}
```

- **Constraints:**

My simulation represents constraints as two arrays, the first containing an `std::pair<int, int>` of the two indices in the constraint, and the second containing an `double` representing the weight of the constraint.

A shortfall with this representation is that the constraint is not able to have a cardinality greater than 2.

- **Solver:**

As discussed in the paper, I implemented these formulae for constraint projection:

$$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

$$\Delta p_2 = +\frac{w_2}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

- **Damping:**

As discussed in the paper, I implemented my damping function as follows:

$\mathbf{x}_{cm} = (\sum_i \mathbf{x}_i m_i)/(\sum_i m_i)$
$\mathbf{v}_{cm} = (\sum_i \mathbf{v}_i m_i)/(\sum_i m_i)$
$\mathbf{L} = \sum_i \mathbf{r}_i \times (m_i \mathbf{v}_i)$
$\mathbf{I} = \sum_i \tilde{\mathbf{r}}_i \tilde{\mathbf{r}}_i^T m_i$
$\boldsymbol{\omega} = \mathbf{I}^{-1}\mathbf{L}$
forall vertices i
  $\Delta\mathbf{v}_i = \mathbf{v}_{cm} + \boldsymbol{\omega} \times \mathbf{r}_i - \mathbf{v}_i$
  $\mathbf{v}_i \leftarrow \mathbf{v}_i + k_{\text{damping}}\Delta\mathbf{v}_i$
endfor

Figure 1: Damping Velocities Routine

4. Examples Since my project was based on the priorly done assignment 1, A sanity check was to certify correct behavior on the A1 tests.

You can control which scenario runs;

```
1 = single strand / pendulum, demonstrate correct dynamics.
2 = multiple strands with collisions, demonstrate correct collision handling.
3 = 2D cloth, demonstrate correct dynamics.
4 = 3D beam, demonstrate correct dynamics.
5 = 3D curly hair strand
6 = An array of falling particles bouncing off a surface.
7 = Particle-Sand set-up with Position-based Dynamics.
```

How To Run

I implemented my project on top of A1, so it should be runnable by running `a1_mass_spring` through

```
scripts/run_assignment.sh a1_mass_spring [1-7]
```

Or, if on windows:

```
scripts\run_assignment.bat a1_mass_spring [1-7]
```

I also changed some of the provided source-code (`ImplicitDriver`, `Common.h`, `Particles.h`). To avoid conflicts, I added them in the `a1_mass_spring/src` directory and changed the `#include` statements to try to pick up those specific versions over the global versions.