Advanced Dynamic Simulations - Proposal

Student Names: Anbang Hu, Yang Yu

Overview

We will implement solvers more advanced than forward Euler and simplectic Euler based on Scotty3D.

Advanced Solvers

We have implemented forward Euler and simplectic Euler in Assignment 4. We will explore another two stable integration methods - [improved Euler and backward Euler (https://math.la.asu.edu/~dajones/class/275/ch2.pdf)](https://math.la.asu.edu/~dajones/class/275/ch2.pdf).

Improved Euler

Forward Euler can be improved by considering

```
u(t+1) = u(t) + 0.5 * tau * (f(u(t)) + f(u(t+1)))
```

In order to get rid of implicit term `f(u(t+1))`, we can apply forward Euler to `u(t+1)` inside `f(u(t+1))`. Then the improved Euler update rule is:

```
u(t+1) = u(t) + 0.5 * tau * (f(u(t)) + f(u(t) + tau * f(u(t))))
```

We will implement improved Euler in `Mesh::improved_euler()` in `mesh.cpp`.

We expect more stable result from improved Euler than from forward Euler.

Backward Euler

Backward Euler update rule

```
u(t+1) = u(t) + tau * f(u(t+1))
```

has `u(t+1)` on both sides of the equation. To solve this algebraic equation for unknown `u(t+1)`, we will try the following two options.

Option I Fixed-point iteration (First attempt)

We will implement the following iterative algorithm in function `Mesh::backward_euler_iterative()` in `mesh.cpp`:

```
# Initialization
u(t+1) = u(t)

# Loop until convergence (absolute error between
# consecutive u(t+1) is less than tolerance)
u(t+1) = u(t) + tau * f(u(t+1))
```

We anticipate that this method may requires a large number of iterations before `u(t+1)` eventually converges.

Option II Solving linear systems (Power of matrix multiplication)

We will follow the section titled "Meshes and Matrices" in the course notes (http://www.cs.cmu.edu/~kmcrane/Projects/DGPDEC/paper.pdf) and use linear algebra package Eigen (http://eigen.tuxfamily.org/index.php?title=Main_Page) to implement backward Euler in `Mesh::backward_euler_matrix()` in `mesh.cpp` for triangle mesh:

1. Assign unique index to the vertices in the mesh;
2. Build the Laplace and mass matrice according to the indices of the vertices;
3. Solve the linear system that describes the backward Euler update rule.

The linear system we need to solve is

```
u(i,t+1) = u(i,t) + 0.5 * tau * sum_j (cot(alpha(i,j)) + cot(beta(i,j))) *
(u(j,t+1) - u(i,t+1))
```

where `u(i,t)` is the configuration of the vertex with index `i` at time `t`, `sum_j` means summation over all neighbors of `i`. The system can be rewritten as

```
0 = 0.5 * tau * sum_j (cot(alpha(i,j)) + cot(beta(i,j))) * u(j,t+1) -
    (0.5 * tau * sum_j (cot(alpha(i,j)) + cot(beta(i,j))) - 1) * u(i,t+1) +
    u(i,t)
```

according to which we can build a matrix `B` of size (V+1)-by-(V+1), where V is the number of vertices in the mesh. Each row of `B` corresponds to the coefficients of one linear equation. The coefficients are arranged in correspondence with `u = (u(0,t+1), u(1,t+1),...,u(V-1,t+1), 1)`. We can then solve the linear system `Bu = 0` to get each `u(i,t+1)`.

We expect that solving linear system can potentially improve the performance of the program.