USTC-CG/2024 课程作业 实验报告

实验 8	Mass Spring
马天开	PB21000030 (ID: 08)
Due: 2024.04.29	Submitted: 2024.04.29

功能实现 Features Implemented

作业要求部分 Required Features

Semi-Implicit Euler

```
// Semi-implicit Euler
Eigen::MatrixXd acceleration = -computeGrad(stiffness) / mass_per_vertex;
acceleration.rowwise() += acceleration_ext.transpose();

if (enable_sphere_collision) {
    acceleration += acceleration_collision;
}

vel += h * acceleration * damping;

// set fixed points to zero velocity
for (unsigned i = 0; i < n_vertices; i++) {
    if (dirichlet_bc_mask[i]) {
        vel.row(i).setZero();
    }
}

X += h * vel;
```

`computeGrad`:

```
Eigen::MatrixXd g = Eigen::MatrixXd::Zero(X.rows(), X.cols());
unsigned i = 0;
for (const auto& e : E) {
    auto diff = X.row(e.first) - X.row(e.second);
    auto l = E_rest_length[i];
    auto grad = stiffness * (diff.norm() - l) * diff / diff.norm();
    g.row(e.first) += grad;
    g.row(e.second) -= grad;
    i++;
}
return g;
```

Implicit Euler

Hessian matrix:

```
unsigned n_vertices = X.rows();
Eigen::SparseMatrix<double> H(n_vertices * 3, n_vertices * 3);

unsigned i = 0;
const auto I = Eigen::MatrixXd::Identity(3, 3);
for (const auto& e : E) {
    Eigen::Vector3d diff = X.row(e.first) - X.row(e.second);
    auto l = E_rest_length[i];
    Eigen::MatrixXd H_e =
        stiffness * (1 - l / diff.norm()) * (I - diff * diff.transpose() /
diff.squaredNorm()) +
        stiffness * (diff * diff.transpose() / diff.squaredNorm());

    for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 3; k++) {
            H.coeffRef(3 * e.first + j, 3 * e.first + k) += H_e(j, k);
            H.coeffRef(3 * e.first + j, 3 * e.second + k) -= H_e(j, k);
            H.coeffRef(3 * e.second + j, 3 * e.first + k) -= H_e(j, k);
            H.coeffRef(3 * e.second + j, 3 * e.second + k) += H_e(j, k);
        }
    }

    // fix the fixed points
    if (dirichlet_bc_mask[e.first]) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                H.coeffRef(3 * e.first + j, 3 * e.first + k) = 0;
            }
            H.coeffRef(3 * e.first + j, 3 * e.first + j) = 1;
        }
    }

    i++;
}
```

Solve linear system:

```
auto H_elastic = computeHessianSparse(stiffness);  // size = [nx3, nx3]
Eigen::SparseMatrix<double> H =
    H_elastic +
    mass_per_vertex * Eigen::MatrixXd::Identity(n_vertices * 3, n_vertices * 3) /
h / h;


Eigen::MatrixXd fext = Eigen::MatrixXd(n_vertices, 3);
fext.rowwise() = acceleration_ext.transpose();
```

```
Eigen::MatrixXd Y = X + h * vel + h * h * fext;
Eigen::MatrixXd grad_g =  mass_per_vertex * (X - Y) / h / h -
computeGrad(stiffness);

// fix the fixed points
for (unsigned i = 0; i < n_vertices; i++) {
    if (dirichlet_bc_mask[i]) {
        for (int j = 0; j < 3; j++) {
            grad_g(i, j) = X(i, j);
        }
    }
}

Eigen::MatrixXd grad_g_ = flatten(grad_g);

Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>> solver;
solver.compute(H);
if (solver.info() != Eigen::Success) {
    std::cerr << "Decomposition failed!" << std::endl;
    return;
}

Eigen::MatrixXd X_new = solver.solve(grad_g_);
if (solver.info() != Eigen::Success) {
    std::cerr << "Solving failed!" << std::endl;
    return;
}

Eigen::MatrixXd X_new_ = unflatten(X_new);

X = X_new_;
vel = (X - init_X) / h;
```

运行截图 Screenshots

`demo.mkv`, sent with report.