Group Member: Yasheng Sun, Qichang Sun

Problem Restatement:

Apply implicit Laplacian smoothing to the model.

Methodology

Laplacian smoothing includes uniform smoothing and cotangent smoothing, which apply smoothing mechanism to a vertex using different weight of its neighbors.

$$L_c(\mathbf{v}_i) = \frac{1}{2A(\mathbf{v}_i)} \sum_{\mathbf{v}_j \in N_1(\mathbf{v}_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{v}_j - \mathbf{v}_i)$$

Fig. 1. Cotangent smoothing

$$L_u(\mathbf{v}_i) = \frac{1}{|N_1(\mathbf{v}_i)|} \sum_{\mathbf{v}_j \in N_1(\mathbf{v}_i)} (\mathbf{v}_j - \mathbf{v}_i) \qquad \text{where} \quad |N_1(\mathbf{v}_i)| = d_i$$

Fig. 2. Uniform smoothing

There are two common ways to execute Laplacian smoothing including implicit Laplacian smoothing and explicit Laplacian smoothing.

The explicit Laplacian smoothing computes the gradient of vertex position, which is added to the original position when updating the position of this vertex. (1) describe this procedure, μ indicate the step length of updating.

$$x_{n+1} = x_n + \Delta = x_n + \mu L(x_n) \qquad (1)$$

In implicit Laplacian smoothing, Laplacian operator is applied to new position for algorithm stability, as is shown in (2). After reorganize (2), this linear equation in (3) describe the mathematical procedure of smoothing.

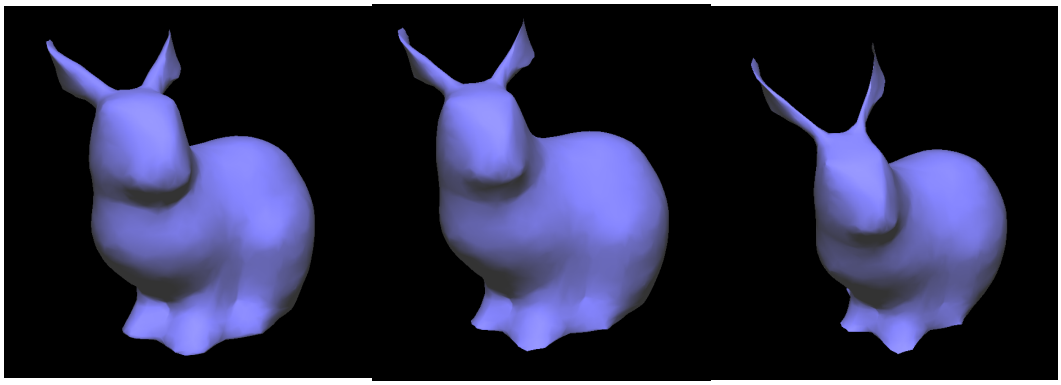
$$x_{n+1} = x_n + \Delta = x_n + \mu L(x_{n+1}) \qquad (2)$$

$$(I - \mu L)x_{n+1} = x_n \qquad (3)$$

When it comes to programing in practice, we use (4) to update the position of vertices to ensure the solution of this linear equation exists. At the same time, $(I - \mu L)^T (I - \mu L)$ is positive definite, which is required in cholesky decomposition if we would like to apply the solver which has already written well in our project.

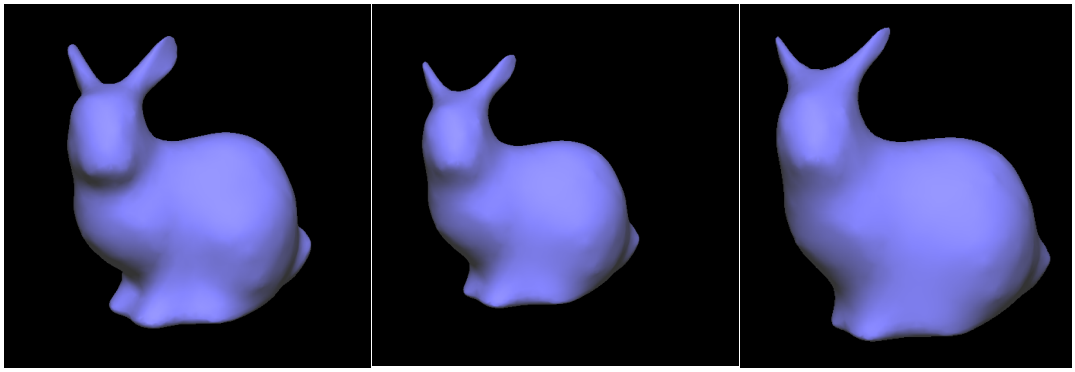
$$(I - \mu L)^T (I - \mu L) x_{n+1} = (I - \mu L)^T x_n \quad (4)$$

The results for uniform smoothing and cotangent smoothing is shown below.



10 iterations 20 iterations 30 iterations

Fig. 4 Shape of model in different iterations in cotangent smoothing



10 iterations 20 iterations 30 iterations

Fig. 5 Shape of model in different iterations in uniform smoothing

In both smoothing strategy, the model becomes smoother along with increasing iterations. However, the ear of rabbit in uniform smoothing becomes sleeker while the ear in cotangent smoothing becomes thinner, which was caused by the different choice of weight.

Appendix

My code is on my Github https://github.com/sunyasheng/Computer-Graphics.

Core code

```
void Mesh::ImplicitUmbrellaSmooth(bool uniformWeights) {
    if (!uniformWeights) {
        Cotangent_Smoother cotangent_laplacian(this);
        cotangent_laplacian.Deform();
    }
    else {
        Uniform_Smoother uniform_laplacian(this);
        uniform_laplacian.Deform();
    }
}
```

```
class Cotangent_Smoother
{
public:
    Cotangent_Smoother(Mesh * mesh);
    ~Cotangent_Smoother();
    void Deform();
private:
    void BuildSystemMatrix();
    SparseLinearSystemSolver* solver;
    Mesh * mesh;
    Eigen::VectorXd vx, vy, vz;
    SparseMatrixBuilder* Lap = new SparseMatrixBuilder();
};
```

```
void Uniform_Smoother::BuildSystemMatrix() {
    VertexList vList = mesh->Vertices();

    for (int i = 0; i < vList.size(); i++) {
        int k = vList[i]->Valence();
        HEdge* nextHedge = vList[i]->HalfEdge()->Twin();
        for (int r = 0; r < k; r++) {
            Lap->AddEntry(i, nextHedge->Start()->Index(), 1.0/k);
            nextHedge = nextHedge->Next()->Twin();
        }
        Lap->AddEntry(i, i, -1.0);
    }
}
```

```
void Cotangent_Smoother::BuildSystemMatrix() {
    VertexList vList = mesh->Vertices();

    for (int i = 0; i < vList.size(); i++) {
        int k = vList[i]->Valence();
        HEdge* nextHedge = vList[i]->HalfEdge()->Twin();
        Eigen::Vector3d preVertex, currVertex, nexVertex;
        double cotAphla, cotBeta, sumWeight = 0.0;

        for (int r = 0; r < k; r++) {
            currVertex = nextHedge->Start()->Position();
            preVertex = nextHedge->Twin()->Prev()->Start()->Position();
            nexVertex = nextHedge->Next()->Twin()->Start()->Position();
            cotAphla = Mesh::Cot(vList[i]->Position(), nexVertex, currVertex);
            cotBeta = Mesh::Cot(vList[i]->Position(), preVertex, currVertex);
            sumWeight += cotAphla + cotBeta;
            nextHedge = nextHedge->Next()->Twin();
        }
        for (int r = 0; r < k; r++) {
            cotAphla = Mesh::Cot(vList[i]->Position(), nextHedge->Next()->Twin()->Start()->Position(), nextHedge->Start()->Po
            cotBeta = Mesh::Cot(vList[i]->Position(), nextHedge->Twin()->Prev()->Start()->Position(), nextHedge->Start()->Pos
            Lap->AddEntry(i, nextHedge->Start()->Index(), (cotAphla + cotBeta) / sumWeight);
            nextHedge = nextHedge->Next()->Twin();
        }
        Lap->AddEntry(i, i, -1.0);
    }

}
```

```cpp
void Cotangent_Smoother::Deform() {
    VertexList vList = mesh->Vertices();
    SparseMatrixBuilder* Identity = new SparseMatrixBuilder();
    for (int i = 0; i < vList.size(); i++) Identity->AddEntry(i, i, 1.0);

    Eigen::SparseMatrix<double> I(vList.size(), vList.size());
    I = Identity->ToSparseMatrix(vList.size(), vList.size());
    Eigen::SparseMatrix<double> Laplacian(vList.size(), vList.size());
    Laplacian = Lap->ToSparseMatrix(vList.size(), vList.size());
    Eigen::SparseMatrix<double> A(vList.size(), vList.size());
    double lamba = 0.8;
    A = I - lamba*Laplacian;

    vx = Eigen::VectorXd::Zero(vList.size());vy = Eigen::VectorXd::Zero(vList.size());vz = Eigen::VectorXd::Zero(vList.size(

    for (int i = 0; i < vList.size(); i++) {
        vx[i] = vList[i]->Position()(0);
        vy[i] = vList[i]->Position()(1);
        vz[i] = vList[i]->Position()(2);
    }
    Eigen::VectorXd bx = A.transpose()*vx;
    Eigen::VectorXd by = A.transpose()*vy;
    Eigen::VectorXd bz = A.transpose()*vz;
    Eigen::VectorXd vx2 = Eigen::VectorXd::Zero(vList.size());
    Eigen::VectorXd vy2 = Eigen::VectorXd::Zero(vList.size());
    Eigen::VectorXd vz2 = Eigen::VectorXd::Zero(vList.size());

    solver = new SparseLinearSystemSolver(A.transpose()*A);
    vx2 = solver->Solve(bx);
    vy2 = solver->Solve(by);
    vz2 = solver->Solve(bz);
    for (int i = 0; i < vList.size(); i++) {
        Eigen::Vector3d newPosition = Eigen::Vector3d(vx2[i], vy2[i] , vz2[i]);
        vList[i]->SetPosition(newPosition);
    }
}
```