HW5 Question 4

Xu Yin 20205445

December 12, 2021

1 HKS calculation (see Fig. 1)

- Using the HKS as the mesh descriptor function, e.g F and G in the lecture 22.

```
// First get the function representation of the source (A) and target (B)

MatrixXd F = load_csv(root_path + "Results/" + source + "_hks.csv", 6890, 100);//dim of the des
MatrixXd G = load_csv(root_path + "Results/" + target + "_hks.csv", 6890, 100);
```

Figure 1: Compute the priority of a halfedge.

- Get the Pseudo inverse of bases (see Fig.2

```
MatrixXd S_eigval = load_csv(root_path + "Results/" + source + "_eigval.csv", 300, 1);
MatrixXd T_eigval = load_csv(root_path + "Results/" + target + "_eigval.csv", 300, 1);

Map<MatrixXd, 0, OuterStride<> > s_base(S.data(), S.rows(), K, OuterStride<>(S.outerStride()));//6890*40
Map<MatrixXd, 0, OuterStride<> > t_base(T.data(), T.rows(), K, OuterStride<>(T.outerStride()));//6890*40

// Get the pseudo inverse of the bases
MatrixXd trans_m_bases = s_base.completeOrthogonalDecomposition().pseudoInverse();//40*6890
MatrixXd trans_n_bases = t_base.completeOrthogonalDecomposition().pseudoInverse();//40*6890
```

Figure 2: Pseudo inverse of the bases.

- Get the A&B representation(see Fig.3

2 Least Square Fitting.

Implement the **Least Square Regression of two functions (See Fig. 4)**.

```
Map<VectorXd, 0, OuterStride<> > s_diag(S_eigval.data(), K, S_eigval.cols(), OuterStride<>(S.outerStride()));
Map<VectorXd, 0, OuterStride<> > t_diag(T_eigval.data(), K, T_eigval.rows(), OuterStride<>(T.outerStride()));

MatrixXd A = trans_m_bases * F;//40*100
MatrixXd B = trans_n_bases * G;//40*100
```

Figure 3: Get the representation of A and B.

```
tuple<MatrixXd, float> solver(MatrixXd p1, MatrixXd p2, VectorXd L1, VectorXd L2, float lambda)

{
    //implement the fitting of ||C*p1-p2||+lambda*||C*L1-L2*C||
    int n1 = p1.rows();
    int n2 = p2.rows();
    MatrixXd A_fixed = p1 * p1.transpose();
    MatrixXd B = p1 * p2.transpose();

    MatrixXd X = MatrixXd::Zero(n2, n1);

    for (int i = 0; i < n2; i++)
    {
        VectorXd test = lambda * (L1.array() - L2(i)).square();
        Eigen::MatrixXd diag = test.array().matrix().asDiagonal();
        diag += A_fixed;

        VectorXd sol(Map<VectorXd>(B.col(i).data(), B.rows()));
        X.row(i) = diag.bdcSvd(ComputeThinU | ComputeThinV).solve(sol);


    }
    float residual = ((X*p1) - p2).norm();
    residual *= residual;
    return make_tuple(X, residual);
}
```

Figure 4: Least Square Fitting of two terms.

3 Point to Point match (see Fig. 5)

.

4 Implementation and Results.

- You can implement all meshes' FM calculation and matching by running Report_total_error() function (in func_map.cpp).
- Evaluate the FM matching (**Achieve mean error:0.6950, see Fig. 6**).

```cpp
vector<int> Matching(MatrixXd FM, MatrixXd s_ps, MatrixXd t_ps)
{
    t_ps = t_ps.transpose();
    int nbr_s = s_ps.rows();
    int nbr_t = t_ps.rows();
    vector<int> result;
    vector<int> keypoint = read_keypoint();

    MatrixXd s_to_t = (FM * s_ps).transpose();//40*6890

    assert(s_to_t.size() == 40 * 6890);
    //cout<<keypoint.size()<<endl;
    for (int i = 0; i < keypoint.size(); i++)
    {
        int s_dice = keypoint[i];
        MatrixXd s_point = s_to_t.row(s_dice);
        float tmp_error = std::numeric_limits<float>::infinity();
        int indice = 0;

        for (int j = 0; j < nbr_t; j++)
        {

            float tmp = (s_point - t_ps.row(j)).squaredNorm();//calculate
            //cout<<tmp<<endl;
            if (tmp < tmp_error)
            {
                //cout<<tmp<<endl;
                tmp_error = tmp;
                indice = j;
            }
        }
        result.push_back(indice);
    }
    cout << result.size() << endl;
    return result;

}
```

Figure 5: FM matching with the projector.

```
Mean error of source mesh 1: 0.7395207031202279
Mean error of source mesh 2: 0.5369801875309134
Mean error of source mesh 3: 0.6241344359691187
Mean error of source mesh 4: 0.749525471690207
Mean error of source mesh 5: 0.7013338146651498
Mean error of source mesh 6: 0.6630037559967478
Mean error of source mesh 7: 0.6925111184599908
Mean error of source mesh 8: 0.761538572759552
Mean error of source mesh 9: 0.7869854944931645
Mean error of all source meshes: 0.6950592838538968

Process finished with exit code 0
```

Figure 6: Result of FM matching (**when setting** $\lambda = 5$).