CS230 Computer Graphics
Project 3 Write-Up

Yunshu Wang
Dept. of Computer Science
ywang1127@ucr.edu

Hsi-Min Chou
Dept. of Computer Science
hchou026@ucr.edu

1. Concrete Goals and Methods

Concrete Goals	Method	Accomplishment
Implement a line to be our target for later calculation.	In order to build and test our hair simulation model, we should first create an adapted framed curve (Discrete framed curve). OpenGL is able to plot lines with given starting and ending coordinates. We can use C++ and OpenGL to simulate an adapted framed curve by first creating a "curve" consisting of a centerline composed of $n + 2$ vertices where $n = 0$ at first.	Fully
Make the line be able to become an adapted framed curve (like hair).	Then increase n until the curve can be visualized as a curve.	Fully
Start the implementation of the bending energy.	The bending energy should be able to be delivered from one vertice to another while leaving the edges between them as straight lines. Since hairs are attached to the head, one side of the discrete framed curve should keep being static and not moving at all while the force should always be generated from the other side of the curve.	Fully
Apply formulas in the paper to deal with the result from the bending energy.	We then will use the equation (1) - (3) in the paper to realize the bending energy interaction with a "discrete naturally straight, isotropic hair".	Fully
Increase the number of the line to simulate "hair"	When one hair is able to illustrate success, we will try more hairs with them attaching on	Tried but Failed

along with the previous calculation.	the same spot and let the forces on the other side arbitrarily float each of the hair or let them do the same motion.	(Cannot add to one mesh)

2. **Programing Tools**
    a. C++ - using C++ as our programming language.
    b. Libigl - applying the library Libigl which includes OpenGL, GLFW, Eigen and so on to code more easily.

3. **Challenges and Implementation**

    There are mainly three challenges in our project:

    i. How to build a rigid body of a hair;
    ii. How to do animation (make the hair moveable);
    iii. Calculate the hair position (One step);

3.1 Build a Hair Rigid Body Prototype

To tackle the first challenge, we went from two different directions: WebGL and OpenGL interface. We first tried WebGL and found it needs a lot of resources and the example we found requires additional knowledge on html and it needs many lines to build an object. So we decided to use an interface of OpenGL to build our project.

We found Libigl to suit our goal, which is an open source C++ library for geometry processing research and development. On their tutorial website[3], the example to plot a parallelogram by two triangles only needs a 4 * 3 matrix for vertex and a 2 * 3 matrix for edges. It is fairly easy to start with. Inspired by the example, we started building our hair rigid body.

The first hair prototype was built by a 3 * 3 matrix for vertex with only first column $(0, 1, 2)^T$ and all other elements in the matrix zeros. The edges are a 1 * 3 matrix (1,2,3) which indicate the link to vertex 1,2 and 3. The result is shown in Figure 1.

Figure 1. First Hair Rigid Body Prototype

This prototype is later extended to a 10 vertex, 9 edges hair rigid body in section 3.3.

3.2 Animation

Since we are working on a simulation program, it is crucial that our hair is moveable. To enable the movement, we found an example that allows a ball to spin over a concrete box. [4] The implementation of this example is shown in Figure 2.

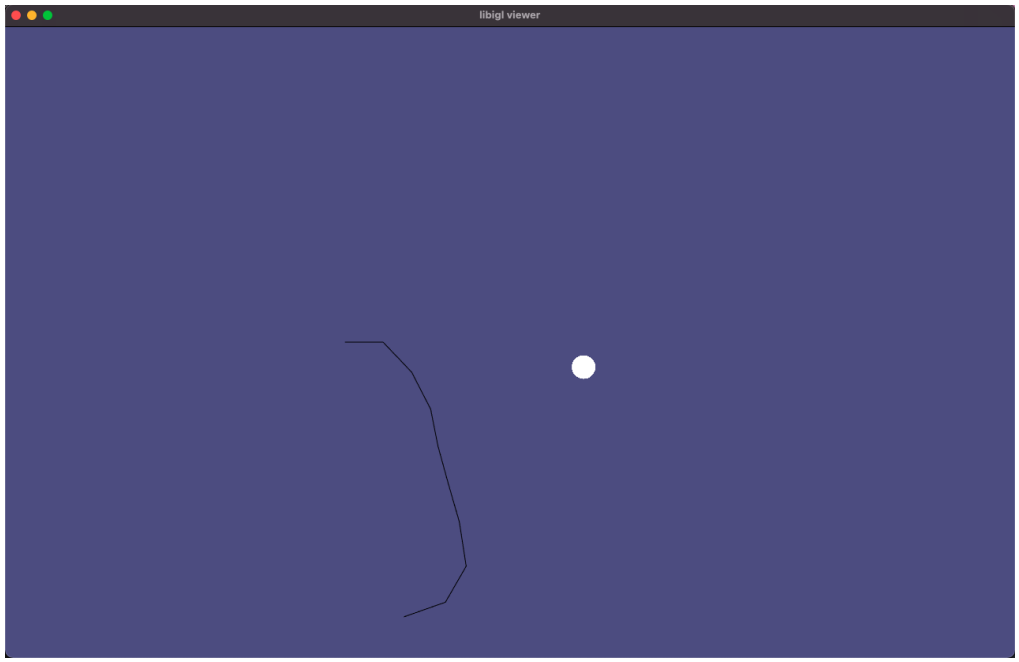

Figure 2. A Screenshot of the Animation Example

The animation is achieved by the libigl provided "*viewer.callback_pre_draw = [&](igl::opengl::glfw::Viewer &)->bool*" function. After achieving the animation of this example, as long as we are able to update our hair rigid body in this function body, we now are ready for the animation of the simulation.

3.3 Update the hair Rigid Body (with applying bending energy)

In this section, we are going to illustrate some notes about how we update the hair animation, the function calls and bodies can be found in the appendix.

To start the simulation, we build a hair with only 10 nodes and 9 edges. Then we push it to the function *"initVisualization()"* to make the hair immediately visualized, and this visualization helps the further programming a lot. In the Visualization function, we set the vertex of the the hair using a 10 * 3 matrix and at the same time update the edge of these nodes. The first column of the matrix is (-5, -4, ..., 4) and all other elements in the matrix are all zeros. This helps to plot a line lying on the x axis on the viewer. The edges simply link these vertices together and it only needs to be done once. The setting of edges is shown in Appendix 1.2.

In section 4. Kirchhoff rods in the "Discrete Elastic Rods" paper, the authors give the formula to calculate the bending energy.

$$E_{\text{bend}}(\Gamma) = \frac{1}{2}\int (\boldsymbol{\omega} - \overline{\boldsymbol{\omega}})^T \overline{B}(\boldsymbol{\omega} - \overline{\boldsymbol{\omega}}) \mathrm{d}s \,, \tag{a}$$

For example, in formula (a), B is a symmetric positive definite 2 * 2 matrix. However, in our program, we just set it to a very large number, so the curvature can be visualized rather than just straight lines. We also found that B can actually control the "softness" of the hair. If B is a very large number, the hair is very hard, but when B is a small number, the hair is "soft".

Applying the bending energy was basically followed by the equations on the paper:

$$(\kappa \mathbf{b})_i = \frac{2\mathbf{e}^{i-1} \times \mathbf{e}^i}{|\overline{\mathbf{e}}^{i-1}||\overline{\mathbf{e}}^i| + \mathbf{e}^{i-1} \cdot \mathbf{e}^i} \,. \tag{1}$$

$$\boldsymbol{\omega}_i^j = \left((\kappa \mathbf{b})_i \cdot \mathbf{m}_2^j, -(\kappa \mathbf{b})_i \cdot \mathbf{m}_1^j \right)^T \quad \text{for} \quad j \in \{i-1, i\} \,. \tag{2}$$

Equation (1) calculates the Discrete curvature binormal, our implementation is shown in Appendix 1.5. Equation (2) calculates the material curvatures, which is shown in Appendix 1.6. (Note the /2 can be deleted.)

$$E_{\text{bend}}(\Gamma) = \frac{1}{2}\sum_{i=1}^{n} \alpha \left(\frac{\kappa \mathbf{b}_i}{\bar{l}_i/2} \right)^2 \frac{\bar{l}_i}{2} = \sum_{i=1}^{n} \frac{\alpha (\kappa \mathbf{b}_i)^2}{\bar{l}_i} \,. \tag{3}$$

Equation (3) shows how to calculate the Discrete bending energy. We calculated this energy, which is shown in Appendix 1.7. However, to update our hair with visualization, we also need to calculate the gradient and apply the torsional force on each certerline node. Where the torsional force is a 3 * 1 matrix.

After applying the bending force and calculate each vertex's position, we update the visualization using the *updateVisualization()* function mentioned above to update the vertex and edges.

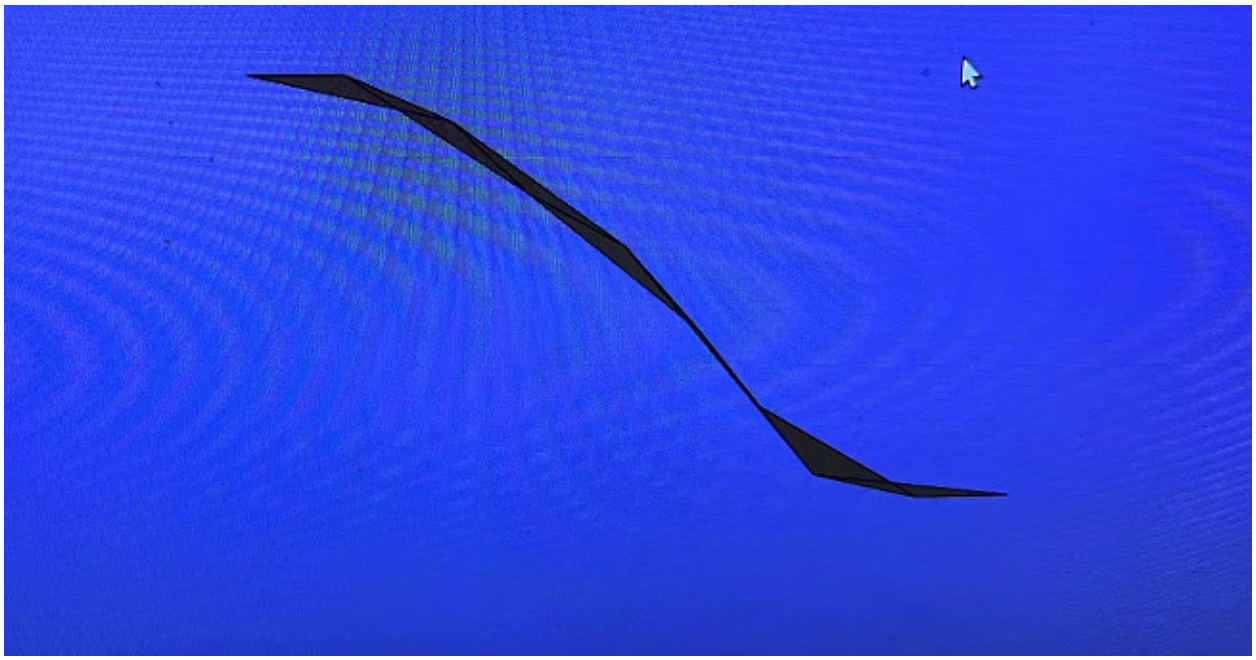

Figure 3. Hair Simulation but with Triangles

4. Discussion and Conclusion

If we could do it over again, we would go find a tutorial from step to step, so we do not need to waste time trying to figure out how to start the project. There are numerous ways to visualize a line, but to find a specific method to build a line by giving only nodes and edges is challenging us. What's more, it's our first time touching computer graphics. However, through this project, we get to know more about how to build a hair simulation. From understanding the matrix calculation and realizing energy is delivered from the first node to the next point in order to animate it. In addition, the equations in the paper cited clearly with numbering to let us understand the procedure of implementing. We also learn about how to simulate a hair, only bending energy is not enough. It is important to add gravity without stretching energy on the line to make it more vivid. The reason why we do not need to add stretching force is we plan to simulate an inextensible rod.

Although we did not implement the twisting energy, we read the paper carefully so that we are able to catch the concept. To add the twisting force, it is necessary to update the material frame by considering the angle of rotation. Furthermore, the key of

the rotation is to define parallel transport, which is also mentioned in Chapter 4.2.2. Above all, We believe it should be another interesting challenge and an attractive animation if we have finished implementing the twisting force and stretching force.

There is also one important goal that we did not fulfill, which is increasing the number of the rods to make them look like “hair”. We did try to increase the amount, however, it is not as easy as we think. Though we raised the number of hairs, the initiation of the line's position might be a problem.

Reference

[1] K. Ward, F. Bertails, T. Kim, S. R. Marschner, M. Cani and M. C. Lin, "A Survey on Hair Modeling: Styling, Simulation, and Rendering," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 213-234, March-April 2007, doi: 10.1109/TVCG.2007.30.

[2] Miklós Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. 2008. Discrete elastic rods. In *ACM SIGGRAPH 2008 papers* (*SIGGRAPH '08*). Association for Computing Machinery, New York, NY, USA, Article 63, 1–12. DOI:https://doi.org/10.1145/1399504.1360662

[3] “License¶.” *Libigl*, https://libigl.github.io/license/.

[4] BrianFruitBrianFruit 4388 bronze badges, and Alec JacobsonAlec Jacobson 5. “Clear, Move, Animate a Point on Libigl Viewer?” *Stack Overflow*, 1 Mar. 1967, https://stackoverflow.com/questions/55878584/clear-move-animate-a-point-on-libigl-viewer.

Appendix

```
6   //create a line for visualization
7   static void initVisualization(int nv, VectorXd x_, vector< Vector3d>* visual_nodes)
8   {
9       for (int i = 0; i<nv; i++) {
10        Vector3d node(x_(3*i), x_(3*i+1), x_(3*i+2));
11        visual_nodes->push_back(node);
12      }
13  }
14
15  //update the line's position after applying energy
16  static void updateVisualization(int nv, VectorXd x_, vector< Vector3d>* visual_nodes)
17  {
18      for (int i = 0; i<nv; i++) {
19        Vector3d node(x_(3*i), x_(3*i+1), x_(3*i+2));
20        visual_nodes->at(i)= node;
21      }
22  }
```

Appendix 1.1 Visualize Nodes on the Showing Window

```
const Eigen::MatrixXi Edg = (Eigen::MatrixXi(9,3)<<
1,1,2,
2,2,3,
3,3,4,
4,4,5,
5,5,6,
6,6,7,
7,7,8,
8,8,9,
9,9,10).finished().array()-1;
std::cout <<"Edg.cols()" << Edg.cols() << std::endl;
```

Appendix 1.2 Edges Plotting Into the Mesh

```
    //initiate basic values
    int nv = 0;
    VectorXd x_;
    vector<bool> is_fixed;
    VectorXd theta;

    nv = 10; //we set 10 nodes as our default value. when setting to nv=5, errors exist
    x_.resize(nv*3);
    x_.setZero();
    theta.resize(nv-1);
    theta.setZero();

    //decide to stick the first two nodes still and compute rest of the nodes
    for (int i = 0; i<nv; i++) {
      x_(3*i) = (-(nv >> 1)+i);
      is_fixed.push_back(false);
    }
    for (int i = 0; i<2; i++) {
      is_fixed[i] = true;
    }
```

Appendix 1.3 Initialize the Number of the Nodes to Build a Line and Stick the First Two Nodes Still Just Like a Hair Stick on the Scalp

```
MatrixX3d DiscreteElasticRods::unitTangents(VectorXd& x_)
{
    MatrixX3d unit_tangents(nv-1, 3);
    for (int i = 0; i<nv-1; i++) {
        unit_tangents.row(i) = ((x_.segment<3>(3*(i+1))-x_.segment<3>(3*i)).normalized()).transpose(); //assign frames to edges, rather than to vertices
        //cout << unit_tangents.row(i) << endl;
    }
    return unit_tangents;
}
```

Appendix 1.4 In Paper 4.2, We Consider Adapted Frames, Where *ti=ei/|ei|* is the Unit Tangent Vector Per Edge

```
242    void DiscreteElasticRods::curvatureBinormal(MatrixX3d t)
243    {
244        for (int i = 0; i<nv-2; i++) {
245            Vector3d t_i = t.row(i).transpose();
246            //cout << t_i << endl;
247            Vector3d t_ip1 = t.row(i+1).transpose();
248            //cout << t_ip1 << endl;
249            Vector3d kb_i = 2*(t_i).cross(t_ip1)/(1+t_i.dot(t_ip1));
250            kb.row(i) = kb_i.transpose();
251        }
252    }
```

Appendix 1.5 Equation (1) Curvature Binormal

```
254    void DiscreteElasticRods::materialCurvature(MatrixX2d& kappa)
255    {
256        kappa.resize(nv-2, 2);
257        for (int i = 0; i<nv-2; i++) {
258            VectorXd kb_i = kb.row(i).transpose();
259            //kappa
260            kappa(i, 0) = kb_i.dot((m2.row(i)+m2.row(i+1)).transpose())/2;
261            kappa(i, 1) = -kb_i.dot((m1.row(i)+m1.row(i+1)).transpose())/2;
262        }
263    }
```

Appendix 1.6 Equation (2) Material Curvature

```
296  double DiscreteElasticRods::bendingForce(VectorXd& gradient, vector<Triplet<double>>& hessian, MatrixX3d& t, VectorXd& bending_force)
297  {
298      vector< Triplet<double>> hess_bending_triplets;
299      const double r = 0.1; //segment radius
300      double E_b = 0.0;
301      MatrixX2d kappa;
302      materialCurvature(kappa);
303
304      for (int i = 1; i<nv-1; i++) {
305          const double area = M_PI*r*r; //area = pi*r*r
306          const double B_half = 1000000*area*r*r/4; //bending modulus = 1000000
307          const double l_i = l_temp(i-1);
308          const double kappa1_i = kappa(i-1, 0);
309          const double kappa2_i = kappa(i-1, 1);
310
311          //compute bending energy
312          E_b += (B_half*(kappa1_i-kappa_temp(i-1, 0)) * (kappa1_i-kappa_temp(i-1, 0)) + B_half*(kappa2_i-kappa_temp(i-1, 1)) * (kappa2_i-kappa_temp(i-1, 1)))/l_i;
```

Appendix 1.7 Bending Energy

```
196        //adding gravity into calculation
197        const double g = -9.8;
198        for (int i = 0; i<nv; i++) {
199            gradient(3*i+1) -= g;
200        }
```

Appendix 1.8 Gravity