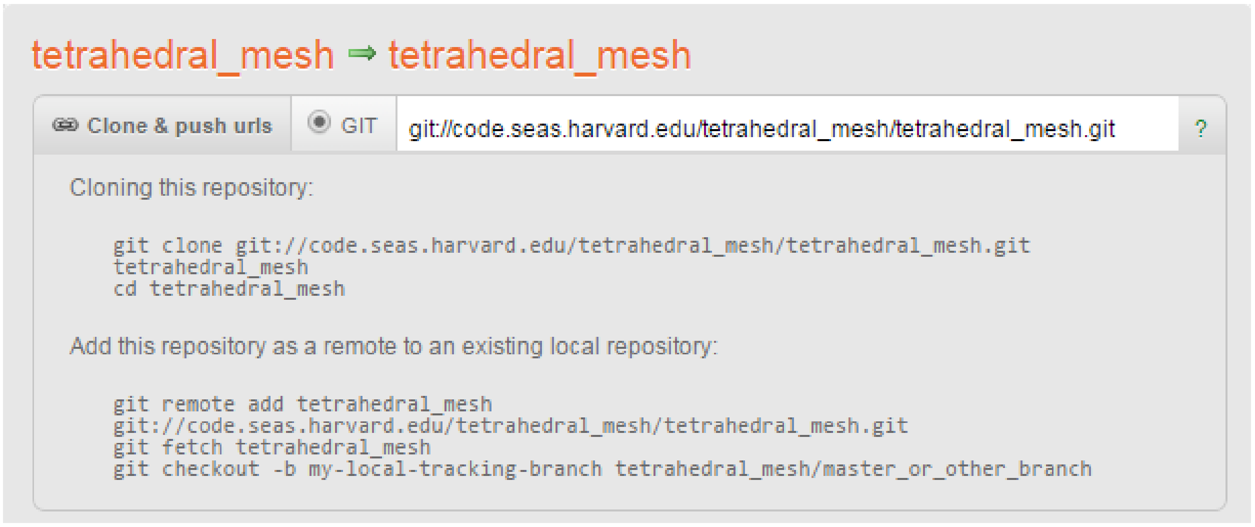
CS 207 Final Project

Prompt 3 - Tetrahedral Mesh

Yung-jen Cheng, Jeffrey Shen

Code:

The code can be found on
https://code.seas.harvard.edu/tetrahedral_mesh/tetrahedral_mesh/trees/master



tetrahedral_mesh ⇒ tetrahedral_mesh

git://code.seas.harvard.edu/tetrahedral_mesh/tetrahedral_mesh.git

Cloning this repository:

```
git clone git://code.seas.harvard.edu/tetrahedral_mesh/tetrahedral_mesh.git
tetrahedral_mesh
cd tetrahedral_mesh
```

Add this repository as a remote to an existing local repository:

```
git remote add tetrahedral_mesh
git://code.seas.harvard.edu/tetrahedral_mesh/tetrahedral_mesh.git
git fetch tetrahedral_mesh
git checkout -b my-local-tracking-branch tetrahedral_mesh/master_or_other_branch
```

Work Accomplished:

1. Provided tetrahedral mesh class to model 3D objects. (YC/JS)
2. Provided tetrahedral forces such as dashpot force and volume penalty force. (YC/JS)
3. Implemented Tian Lan's and Xide Xia's wind force. (JS)
4. Adopted Aaron's and Fil's listener structure into SDLViewer. Merged the function of Ruitao Du's and Yingzhuo Zhang's listener into this structure. (YC/JS)
5. Implemented wind listener. (YC/JS)
6. Implemented a box constraint to constrain the ball in a box and added friction force given from the box to simulate realistic bounces. (JS)
7. Wrote additional functions with the listener. Users can use the arrow keys to pan the camera and use the keyboard button "L" to reset the perspective to have the z axis as the vertical axis (YC)
8. Wrote additional interactive functions such as drag force and hold constraint to interact with objects. This provides users the ability to use the mouse to hold an object (ball) and apply a drag force to move the object. (YC)
9. **Simulated** a tetrahedral ball bouncing in a box. Users can increase, decrease, stop, and resume wind speed. They can also use the arrow keys to pan the camera to observe the simulation in different perspectives. Lastly, users can also use the mouse to click on the ball to hold it and drag the ball. (YC/JS)
10. Implemented Erik's and Lukas' collision detector with Morton code. (JS)

11. **Simulated** 2 balls colliding in a box. The simulation retains all the functionalities in the one ball simulation. Users can hold one of the balls, while the other bounces around. (YC/JS)

As seen above, both of us contributed equally to the project.

In all, we successfully combined four groups' code to make our 2 simulations. In addition, we provided a new underlying structure (tetrahedral mesh), 4 new forces, 2 additional constraints, 2 mouse-click based interactions, and 9 keyboard based interactions.

Specifications and Documentation:

Documentation of our tetrahedral mesh can be found in **docs/Tetrahedral Mesh Documentation**.

Specifications and interface are found in the code in **tet_mesh.hpp** and **simulation.cpp**. They were too long to be copied into documentation and we thought it would be easier for people to directly access the code and read the specifications there.

Final Working Demonstration:

Copy pasted from README FINAL.txt:

Relevant Files:

Simulations are ran from **simulation.cpp**.

One ball simulation:

Simulation Overview:

In this simulation, we simulate a single ball bouncing inside an impenetrable box. Users can interact with the ball via increasing or decreasing wind force and by dragging or flinging the ball inside the box.

Make sure **#define MESH2** is commented out in simulation.cpp. Otherwise, this will simulate 2 balls.

Command Line Code Example:

```
make simulation

./simulation data/sphere87.*

./simulation data/sphere676.*
```

Functionalities:

1. Rotate camera using keyboard arrow keys.
    a. http://youtu.be/gSsR8oq4tdA
2. Reset z-axis as the vertical axis using keyboard button "L."
    a. http://youtu.be/gSsR8oq4tdA at 0:25-0:28
3. Wind force manipulations
    a. "D" to increase wind speed
    b. "A" to decrease wind speed

   c. "W" to resume wind force
   d. "S" to stop wind force
   e. http://youtu.be/O4mwjM8yCU8 (Look at the command line output to see commands)
4. Use mouse left-click to hold an object (make sure the camera is close to the object)
   a. http://youtu.be/rJStBwuFs_A (Look at the command line output when the ball is caught)
5. While holding an object, make a swipe action to drag or fling the object.
   a. http://youtu.be/rJStBwuFs_A

Two balls simulation:

Simulation Overview:

In this simulation, we simulate two balls bouncing inside an impenetrable box.

Make sure **#define MESH2** is **not** commented out in simulation.cpp. Otherwise, this will simulate 1 ball.

```
make simulation
```

```
./simulation data/sphere87.*
```

Functionalities:

Same as the above simulation of one ball.

http://youtu.be/ja5GfRd1OKI

Note: When the forces are too strong, there are some chance that balls become unstable and will get meshed up together. Also, we can only run the two ball simulation with sphere87 because we haven't found the correct values for spring constant, damping constant, and K in volume penalty to properly simulate bigger spheres. Please just restart the simulation when this occurs.

Collaboration Review:

Copy pasted from COLLAB EVAL.txt:

As previously mentioned, we used four groups' code for our final project. For all of the code, we had to make some minor adjustments to fit the code with our tetrahedral mesh. Most of the groups wrote their code with a triangle mesh in mind. Therefore, sometimes we had to go to the hpp files to modify the function calls.

1. We implemented Tian Lan's and Xide Xia's wind force so that we can utilize the wind force to blow the ball around in the box. In addition, we also utilized their collision force calculation to calculate the node's position and velocity after the node collides with another mesh. They were really a great resource for us and were helpful in answering questions about their code.
2. We merged Ruitao Du's and Yingzhuo Zhang's listener with Aaron's and Fil's listener. Ruitao and Yingzhuo were helpful in answering questions about the functionalities of their code and we were successful in using their wind force listener. Aaron's and Fil's listener was well designed and had flexible structure. Their documentation also clearly illustrated how to customize listeners based on their code. We were able to adopt their listener to implement our own ones successfully.

3. Lastly, we used Erik's and Lukas' collision detection. We had to go into the hpp files to make changes since their collision detector is triangle mesh based. Overall, it was a great module and worked really well. We were very impressed by their modules' functionalities and it was very easy to implement in our code.

Individual Review:

These have been emailed to the cs207-staff@seas.harvard.edu.