Authors: Yaxiong Cai, Zhenyang Pan

1. Project Description

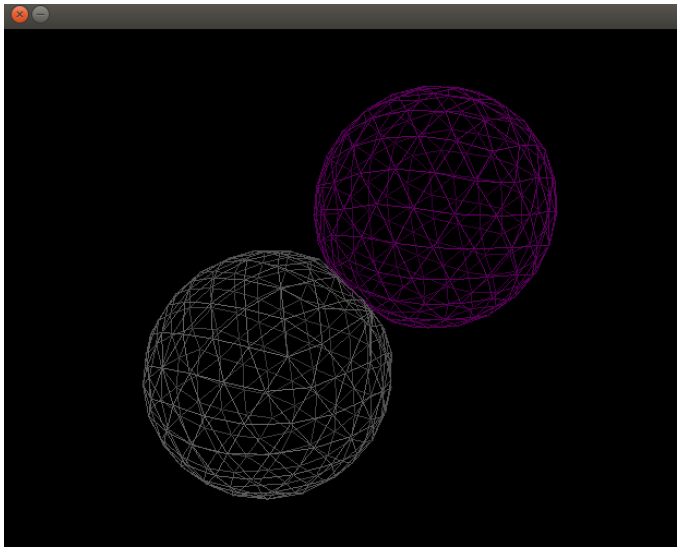

Figure 1: Collision Demo

In our project, we designed a robust SDLViewer that can provide user a simple and very cool GUI interacted by mouse and keyboard. After consulting various needs from different groups, our SDLViewer extension have achieved features that are able to:

- Pause/ accelerate/ decelerate/ recover your visualizer by mouse and keyboard at any time
- Move your objects in 3 dimensions by keyboard at any time
- Change/ Alter/ Hide balls with different color choices
- Initiate/ Increase external wind force to one ball
- Propel one ball directly toward the other ball

Moreover, users can add whatever other specific customized functions they like by very simple steps!

Specifically in the demonstration, we combined our model with Meshed Mass - spring group and Collision Detection Extension group to conduct ball collision physics simulations in various ways. In Meshed Mass - spring, we can alter the property of these two balls. Through our SDLViewer, we can move one ball onto the other ball; hit one ball to the other ball through external wind force; and propel one ball to the other ball through internal speed - up.

2. Collaboration Breakdown

∗ Visualizer extensions: By our group

- Pause/ accelerate/ decelerate/ recover your visualizer by mouse and keyboard at any time
- Move your objects in 3 dimensions by keyboard at any time
- Change/ Alter/ Hide balls with different color choices
- Initiate/ Increase external wind force to one ball

- Propel one ball directly toward the other ball

∗ Meshed Mass-Spring: By Tian Lan and Xide Xia
- Forces (wind force, gravity, pressure, etc) implementation applying to triangle meshed surface
- An inflated ball based on meshed mass-springs

∗ Collision Detection Extension and Morton Code: By Lukas and Erik
- Detect collision when the two balls collide with each other
- Use Morton Code to a accelerate the searching speed

3. Specifications and Documentation

3.1. Class and Struct

Pause_listener: a listener to pause the screen when the right key of the mouse is clicked. When constructed, we store a copy value of the original dt. When this listener is triggered, the dt is set to 0. When clicking again, it will recover back to original dt value.

```
/** @class Pause_Listener
 * @brief Pause or unpause the visualizer
 * @right click: pause; right click again: unpause
 */
struct Pause_listener : public CS207::SDL_Listener {

  void handle(SDL_Event e) {
    switch (e.type) {
      case SDL_MOUSEBUTTONDOWN: {
        if (e.button.button == SDL_BUTTON_RIGHT ) {
          if (dt_ == original_dt_)
            dt_ = 0;
          else
            dt_ = original_dt_;
        }
      } break;
    }
  }
  // constructor
  Pause_listener(double& dt)
    : dt_(dt), original_dt_(dt) {}
  private:
   double& dt_;
   double original_dt_;
};
```

Speed_listener: a listener to adjust the dt value. When constructed, we store a copy value of the original dt. When this listener is triggered, the dt added by 3e-4 if down arrow key is pressed to

slow down, or decreased by 3e-4 if up arrow key is pressed to speed up. When clicking left arrow key, it will recover back to original dt value.

```
/** @class Speed_Listener
 * @brief Adjusts the simulation speed by keyboard.
 * @ Up: accelerate; Down: decelerate; Left: Recover the original speed
 */
struct Speed_listener : public CS207::SDL_Listener {
  private:
    double& dt_;
    double original_dt_;

/** functions to trigger the change of dt
 * @pre @dt_ is the main program's dt, @original_dt_ is assigned to be the
 * initial dt in the program
 * @post @a_ is increased by 3e-4 or decreased by 3e-4 or restore to
 * original dt value
 */
  public:
    Speed_listener(double& dt, double  odt)
      : dt_(dt), original_dt_(odt) {}

    virtual void handle(SDL_Event event) {
      if (event.type == SDL_KEYDOWN) { // keyboard key pressed
        switch (event.key.keysym.sym) {
          case SDLK_DOWN: // down arrow pressed
            dt_ -= 3e-4;
            break;
          case SDLK_UP: // up arrow pressed
            dt_ += 3e-4;
            break;
          case SDLK_LEFT: //recover dt
            dt_ = original_dt_;
            break;

          default: break;
        }
      }
  }

};
```

XYZ_listener: a listener to adjust the x,y,z value of all the node by iterating through the mesh node. Every single click increase or decrease the position value by 0.5.

```
/** @class XYZ_Listener
 * @brief Changes the position of the graph by keyboard
 * @ w, s: moving along x-axis; a, d: moving along y-axis; z, x: moving along z-axis;
 */
template <typename M>
struct XYZ_listener : public CS207::SDL_Listener {

```

```
8
9   /** functions to trigger the change of node position
10   * @pre mesh is valid and has a node iterator
11   * @post for all node.x is increased by 0.5 or decreased by 0.5,
12           for all node.y is increased by 0.5 or decreased by 0.5,
13           for all node.z is increased by 0.5 or decreased by 0.5.
14   */
15    virtual void handle(SDL_Event event) {
16
17      if (event.type == SDL_KEYDOWN) {
18        switch (event.key.keysym.sym) {
19          case SDLK_w:
20            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
21              (*b).position().x += 0.5;
22            }
23            break;
24          case SDLK_d:
25            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
26              (*b).position().y += 0.5;
27            }
28            break;
29          case SDLK_s:
30            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
31              (*b).position().x -= 0.5;
32            }
33            break;
34          case SDLK_a:
35            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
36              (*b).position().y -= 0.5;
37            }
38            break;
39          case SDLK_z:
40            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
41              (*b).position().z += 0.5;
42            }
43            break;
44          case SDLK_x:
45            for(auto b=(*mesh_).node_begin();b!=(*mesh_).node_end();++b){
46              (*b).position().z -= 0.5;
47            }
48            break;
49
50          default: break;
51        }
52      }
53    }
54
55    XYZ_listener(M* mesh)
56      : mesh_(mesh){}
57
58    private:
59     M* mesh_;
60  };
```

Color_listener: a listener to change the color of the graph by changing the integer values of the color functor. Number key 1,2,3 will change the color.

```
/** @class Color_Listener
 * @brief Adjusts the color by keyboard.
 * @ Number key 1 2 3 4 5 6 7: pink, blue, red, cyan, yellow, green
 */
struct Color_listener : public CS207::SDL_Listener {

/** functions to trigger the change of color
 * @pre @a_ is either 0 or 1, @b_ is either 0 or 1, @c_ is either 0 or 1
 * @post @a_ is either 0 or 1, @b_ is either 0 or 1, @c_ is either 0 or 1
 */
virtual void handle(SDL_Event event) {

    if (event.type == SDL_KEYDOWN) {
      switch (event.key.keysym.sym) {
        case SDLK_1:
          *a_ = 1-*a_;
          break;
        case SDLK_2:
          *b_ = 1-*b_;
          break;
        case SDLK_3:
          *c_ = 1-*c_;
          break;
        default: break;
      }
    }
  }

  Color_listener(int* a, int* b, int* c)
    : a_(a), b_(b), c_(c){}

  private:
   int* a_;
   int* b_;
   int* c_;
};

/** @class color
 * @brief a color functor
 */
struct color{
  int a_;
  int b_;
  int c_;
  color(int a, int b, int c): a_(a), b_(b), c_(c){}

  template <typename NODE>
  CS207::Color operator()(const NODE& n) {
    return CS207::Color(a_, b_, c_);
  }
```

```
51 };
```

Force_listener: a listener to apply the force to the graph. By pressing key f, a force will apply to the graph. The force is controlled by a point with (0,0,0) as the empty force and others as the position of the force.

```
1  /** @class Force_listener
2   * @brief add force
3   * @key f will add a force
4   */
5  struct Force_listener : public CS207::SDL_Listener {
6    private:
7      int* a_;
8      int* b_;
9      int* c_;
10
11   public:
12     Force_listener(int* a, int* b, int* c): a_(a), b_(b), c_(c){}
13
14 /** functions to trigger the force
15  * @pre @a_ is 0, @b_ is 0, @c_ is 0
16  * @post @a_ is increased by k times number of 50
17  */
18     virtual void handle(SDL_Event event) {
19       if (event.type == SDL_KEYDOWN) { // keyboard key pressed
20         switch (event.key.keysym.sym) {
21           case SDLK_f:
22             *a_ += 50;
23             *b_ += 50;
24             *c_ += 50;
25             break;
26           case SDLK_r:
27             *a_ = 0;
28             *b_ = 0;
29             *c_ = 0;
30             break;
31
32           default: break;
33         }
34       }
35   }
36
37 };
```

Move_listener: a listener to move one sphere towards another sphere. By pressing key m, both of the center of the two sphere will be calculated. The difference of the two centers times 10 will be set as the new velocity for the first sphere in order to move towards the second one.

```
1  /** @class Move_listener
2   * @brief Move one sphere towards another shpere
3   * @ key: m
4   */
```

```
5  template <typename M>
6  struct Move_listener : public CS207::SDL_Listener {
7
8    /**functions to move the sphere: find the center of two sphere and then
9     * set the velocity of the sphere to be moved to the vector of the two center
10    * @post for all n belongs to @mesh1_, n.value().velocity
11    * is changed toward to @mesh2_ center
12    */
13   virtual void handle(SDL_Event event) {
14       if (event.type == SDL_KEYDOWN) { // keyboard key pressed
15         switch (event.key.keysym.sym) {
16           case SDLK_m:{
17             int obj1_x = 0;
18             int obj1_y = 0;
19             int obj1_z = 0;
20             int obj2_x = 0;
21             int obj2_y = 0;
22             int obj2_z = 0;
23             int obj1_n = 0;
24             int obj2_n = 0;
25
26
27             //compute the center point for mesh1
28             for(auto b=(*mesh1_).node_begin();b!=(*mesh1_).node_end();++b){
29               obj1_x += (*b).position().x;
30               obj1_y += (*b).position().y;
31               obj1_z += (*b).position().z;
32               obj1_n += 1;
33             }
34
35             obj1_x /= obj1_n;
36             obj1_y /= obj1_n;
37             obj1_z /= obj1_n;
38
39
40            //compute the center point for mesh2
41            for(auto b=(*mesh2_).node_begin();b!=(*mesh2_).node_end();++b){
42               obj2_x += (*b).position().x;
43               obj2_y += (*b).position().y;
44               obj2_z += (*b).position().z;
45               obj2_n += 1;
46             }
47
48             obj2_x /= obj2_n;
49             obj2_y /= obj2_n;
50             obj2_z /= obj2_n;
51
52             obj1_x = obj2_x - obj1_x;
53             obj1_y = obj2_y - obj1_y;
54             obj1_z = obj2_z - obj1_z;
55
56             obj1_x *= 10;
57             obj1_y *= 10;
```

```
58              obj1_z *= 10;
59
60              //set the velocity of mesh1
61              for(auto b=(*mesh1_).node_begin();b!=(*mesh1_).node_end();++b){
62                (*b).value().velocity = Point(obj1_x, obj1_y, obj1_z);
63              }
64              }
65             break;
66
67           default: break;
68         }
69       }
70   }
71
72
73   Move_listener(M* mesh1, M* mesh2)
74   : mesh1_(mesh1), mesh2_(mesh2){}
75
76
77   private:
78    //the object to move
79    M* mesh1_;
80    M* mesh2_;
81
82 };
```

3.2. Creating the Objects

For pause listener and speed listener, the dt needs to be plug in to create the object. For XYZ listener, we need to plug in the mesh which should have an iterator of all the nodes. For the color listener, we need to plug in three integers as the reference for the color. For the force, we need to plug in three integers as the reference for the point. For move, we need to plug in the two mesh.

```
1  //three color parameters
2   int color1 = 1;
3   int color2 = 1;
4   int color3 = 1;
5   int color4 = 1;
6   int color5 = 0;
7   int color6 = 1;
8
9   // wind force parameters
10  int a=0;
11  int b=0;
12  int c=0;
13
14
15  //Create listener
16  Pause_listener* pause = new Pause_listener(dt);
17  Speed_listener* speed = new Speed_listener(dt, dt);
18  XYZ_listener<MeshType>* xyz = new XYZ_listener<MeshType>(&mesh);
19  Color_listener* col = new Color_listener(&color1, &color2, &color3);
```

```
20    Color_listener* col2 = new Color_listener(&color4, &color5, &color6);
21    Force_listener* wind = new Force_listener(&a, &b, &c);
22    Move_listener<MeshType>* mov = new Move_listener<MeshType>(&mesh, &mesh2);
```

3.3. Adding listeners

Using the object created, just add the listener in the viewer.

```
1    //add listener
2    viewer.add_listener(pause);
3    viewer.add_listener(speed);
4    viewer.add_listener(xyz);
5    viewer.add_listener(col);
6    viewer.add_listener(col2);
7    viewer.add_listener(wind);
8    viewer.add_listener(mov);
```

4. Simple Steps to Get and Use The Code

∗ For CS207 Staffs for Project Demonstration:

- Getting the package:

$https://github.com/JasonCyx/Final-Project-Deliverables$ and unzip the CS207FinalProject.zip file

∗ For CS207 Students who want to use this extension:

- Getting the code: Download the SDLViewer.hpp, Viewer_Extension.hpp files from:

$https://github.com/JasonCyx/Project-Code$

- Using the code:

  ▷ Replace your SDLViewer.hpp with our SDLViewer.hpp in your CS207 folder, and place the Viewer_Extension.hpp in your working directory

  ▷ Create the customized listener in your main function and add the reference of the listener objects into the viewer

  ▷ Run the program and press the keys and mouse button

- Extending the code:

  ▷ Create your own listener class or any functors in Viewer_Extension.hpp

  ▷ Create the customized listener in your main function and add the reference of the listener object into the viewer

  ▷ Run the program with the customized keys and mouse button

5. Pre-Requirement

∗ Your code should have the following objects

- Mesh Class with nodes iterator
- Graph Class
- Point Class

6. Working Demonstration

∗ To run the code in terminal:

- In terminal change the directory to "CS207FinalProject"
- In terminal $ make two_balls_collision
- In terminal $ ./two_balls_collision data/sphere200* data/sphere200

∗ Interesting features to try:

- RightClick: Pause/ Unpause the simulation
- Keyboard f: Activate/ Increase external wind force to the pass-in object
- Keyboard m: Propel internal speed to the pass-in object
- Keyboard UP: Accelerate the simulation (Cannot accelerate too much due to limitation of meshed mass spring)
- Keyboard DOWN: Decelerate the simulation
- Keyboard LEFT: Recover the simulation to original simulation setting
- Keyboard w (s): Moving the pass-in object along x-axis increase (decrease)
- Keyboard d (a): Moving the pass-in object along y-axis increase (decrease)
- Keyboard z (x): Moving the pass-in object along z-axis increase (decrease)
- Keyboard number key 1 2 3: Changing different colors for the pass-in object

7. Collaboration Evaluation This demonstration contains three (or four) different prompts:

- Visualizer extensions by Zhenyang Pan and Yaxiong Cai:

Basically our code is robust and very easy to use as reflected by other groups. Details contained in the group evaluations

- Meshed Mass-Spring developed by Tian Lan and Xide Xia:

Tian Lan and Xide Xia developed an excellent and cool Meshed Mass-Spring. We obtained their code from Piazza and communicated with them frequently. Their code is very easy to use and well adapted to our code. The wind force and the ball both work quite awesome. We collaborated with Tian Lan and Xide Xia a lot too and they helped us understanding their code as well as with merging their code with other groups.

Some minor limitations:

The properties of the two balls are hard to determine. That is, we need to try different combinations in the main function of value "K", amount of air inside the ball, and value "b", the pressure outside the ball, to determine the softness of the two balls almost iteratively.

- Collision Detection Extension + Morton Code by Lukas and Erik:

Lukas and Erik developed a robust Collision Detector. We obtained the code through Piazza and

we have not communicated with Lukas or Erik in person. It is really awesome that their code integrates the Morton Code, for we previously tried collision detector of another group and the one developed by Lukas and Erik is much faster.

However, there is a bug that both my group and other groups found: when the object is very small or very large in terms of distance between two nodes, either the collision cannot be detected or the program fails.