Meshed Mass Spring

CS207 Final Project

XIDE XIA, TIAN LAN

Chapter 1

Project Description

1.1 Description

In our project, we designed a system based on meshed mass-spring models. By using a triangular Mesh instead, we implemented forces applied to the surfaces (triangles) of the mesh. We combined our model with other cool designs such as collision detection, shallow water extension, and visualizer extension. Finally, our model were able to do several kinds of meshed mass-spring physics simulations such as a cloth flying in a wind, colliding inflated balls, and an inflatable ball falling into water.

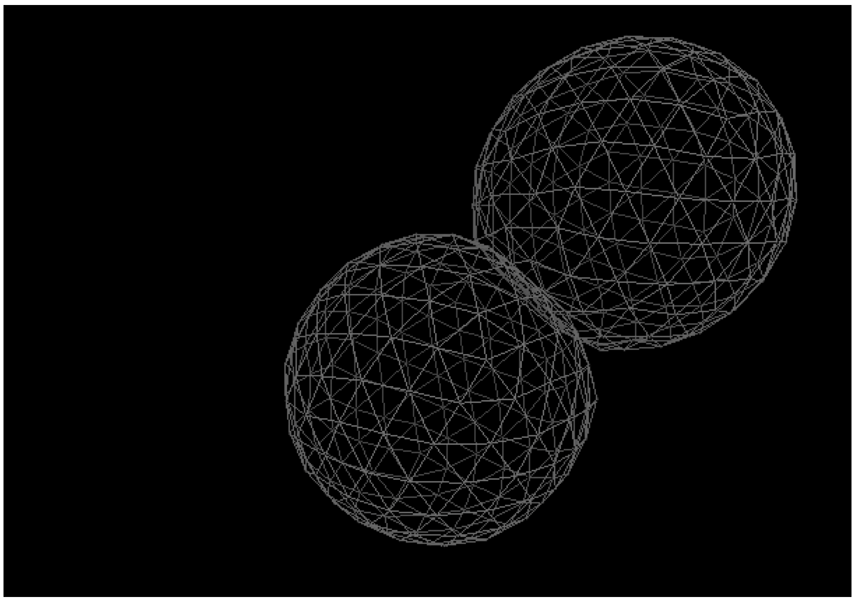

Figure 1.1: Collision of Two Spheres

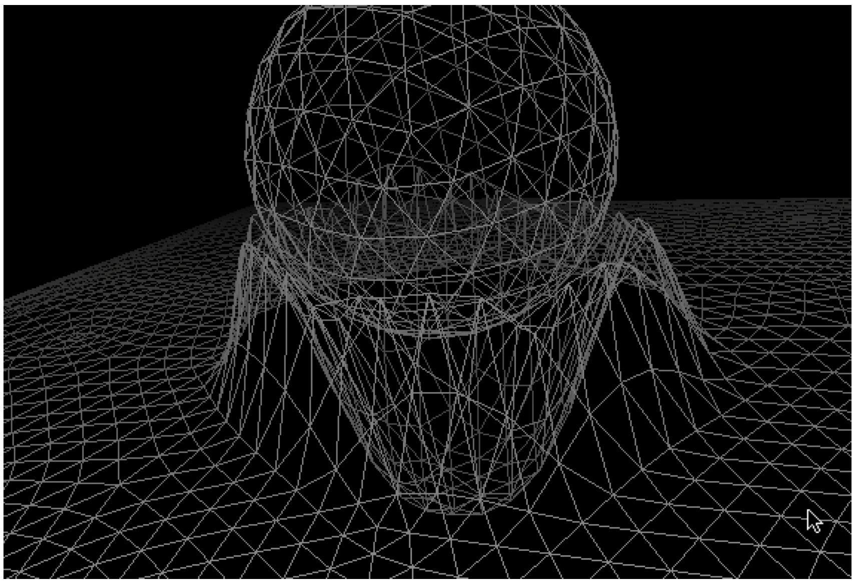

Figure 1.2: Collision of A Sphere and Shallow Water

1.2 Individual Team Breakdown

- Meshed Mass Spring
  - Designed 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.
  - A predicate function determining which node to show on screen
- Shallow Water Extensions
  - Designed by Peiheng Hu and Sail Wu.
  - Shallow water force and source functionality implementation.
- Collision Detector /Morton Coder
  - Designed by Lukas and Eric.
  - Using Morton Code to detect collision when a ball drops and hit shallow water.
- Visualizer Extensions
  - Designed by Zhenyang Pan and Yaxiong Cai.
  - Simulation interface implementation, i.e using keyboard to control the object movement and wind.
  - Pause/ accelerate/ decelerate/ recover the simulation by mouse and keyboard.
  - Adjust the objects' color with keyboard.

1.3 Use Our Code

1.3.1 Get the Code

Our code is stored at: `https://github.com/lantian2012/CS207`

1.3.2 Sample

Take a look at meshed_mass_spring.cpp for a cool piece of sample code. We use the wind force, the air pressure force, and all forces in the old mass spring. You can find that the code for the old forces does not change at all.

1.3.3 Use the Code

Our code design is very easy to use, and it will be compatible with your old mass spring code if you change your graph to a mesh.

Before you start, check if your mesh supports the following interface.

```
//Iterator that iterates through all nodes of the mesh:
Mesh::node_begin();
Mesh::node_end();
//Iterator that iterates through all triangles of the mesh
Mesh::triangle_begin();
Mesh::triangle_end();
Mesh::node_type::degree();  // The number of incident edges
Mesh::edge_type::length();  //The length of an edge
//Incident Iterator that iterates through the incident edges of a node
Mesh::node_type::edge_begin();
Mesh::node_type::edge_end();
//Incident Iterator that iterates through the incident triangles of a node
Mesh::node_type::triangle_begin();
Mesh::node_type::triangle_end();
```

Then, make sure that all of your forces provide the following interface.

```
Point F(NODE n, double t)
```

All you need is to include Meshed_mass_spring.hpp. You may use our CombinedForce to make it easy to add many forces.

In your main function, use the following code to initialize WindForce.

```
WindForce wind_force(Point(x,y,z));
```

The direction of the wind is specified by (x, y, z)

In your main function, use the following code to initialize PressureForce.

```
PressureForce<NodeType, MeshType> pressure_force(p_out, C, &mesh)
```

p_out specifies the air pressure outside the ball. C specifies the amount of air you put into the ball.

After adding the force, template the mesh on the TriData, NodeData, and EdgeData we provide in meshed_mass_spring.cpp. You can add new members, but DO NOT delete any members.

1.3.4 Troubleshooting

1. The mesh explodes when it collides.

   Our code requires that the ball remains a convex shape. If there is very little air inside the ball (C is very small), the ball can become concave during collision. Try increase C.

2. The mesh explodes at the beginning.

   When K is large and dt is large, the mesh may be unstable. Try decrease dt. It is also possible that you forget to include the spring force. In that case, the ball will expand without constraint.

3. The ball is always shaking.

   As there is difference in air pressure inside and outside the ball at the begging, the volume of the ball will fluctuates due to the effect of spring force and air pressure. This is a physical phenomenon instead of a bug. If you do not like it, try increasing K. Also, you can try setting the L in spring force to be smaller than the length of edges.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 BoxConstraint$<$ G $>$ Struct Template Reference

Constraint that acts as a box.

```
#include <Meshed_mass_spring.hpp>
```

Public Member Functions

- BoxConstraint (double h1, double h2, double left, double right)

  Constructor Function.
- void operator() (G &g, double)

  Constrain nodes with box constraints.

Public Attributes

- double h_lower

  The height of the plane at the bottom.
- double h_upper

  The height of the plane at the top.
- double l

  The posistion of the plane on the left.
- double r

  The posistion of the plane on the right.

4.1.1 Detailed Description

template$<$typename G$>$struct BoxConstraint$<$ G $>$

Constraint that acts as a box.

This struct implements a constraint that acts like a box. This struct can define four planes at any posistion.

Template Parameters

G	A class that satisfies the graph concept.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 template<typename G > BoxConstraint< G >::BoxConstraint (double *h1,* double *h2,* double *left,* double *right*) [inline]

Constructor Function.

Parameters

in	*h1*	The height of the plane at the bottom
in	*h2*	The height of the plane at the top
in	*left*	The posistion of the plane on the left
in	*right*	The posistion of the plane on the right

4.1.3 Member Function Documentation

4.1.3.1 template<typename G > void BoxConstraint< G >::operator() (G & *g,* double) [inline]

Constrain nodes with box constraints.

Parameters

in,out	*g*	An object that satisfies the graph concept.
in	*t*	Time.(Not used here)

This function sets the velocity and position of nodes, so nodes will act as they reach a box and bounce back.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.2 CollisionConstraint< G > Struct Template Reference

Constraint for collision between spheres.

```
#include <Meshed_mass_spring.hpp>
```

Public Member Functions

- void operator() (G &g1, G &g2, std::vector< unsigned > list1, std::vector< unsigned > list2)

  Resets the velocity and position of nodes during collision.

4.2.1 Detailed Description

template<typename G>struct CollisionConstraint< G >

Constraint for collision between spheres.

This struct implements a constraint that sets the velocity and position of nodes so that spheres can collide with each other and bounce back.

Remarks

As this constraint needs to take in two graphs, this constriant does not conform the general "Constraint concept."

Template Parameters

G	A class that satisfies the graph concept.

4.2.2 Member Function Documentation

4.2.2.1 template< typename G > void CollisionConstraint< G >::operator() (G & *g1*, G & *g2*, std::vector< unsigned > *list1*, std::vector< unsigned > *list2*) `[inline]`

Resets the velocity and position of nodes during collision.

Parameters

`in,out`	*g1*	An object that satisfies the graph concept.
`in,out`	*g2*	An object that satisfies the graph concept.
`in`	*list1*	Nodes in g1 that are inside g2
`in`	*list2*	Nodes in g2 that are inside g1

This function sets the velocity and position of nodes in two graphs, so the two spheres can collide with each other and bounce back.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.3 CombinedConstraint< C1, C2, G > Struct Template Reference

Constraint that combines two constraints.

```
#include <Meshed_mass_spring.hpp>
```

Public Member Functions

- CombinedConstraint (C1 c1=C1(), C2 c2=C2())

  Constructor Function.
- void operator() (G &g, double)

  Constrain the node with the effect of two constraints.

Public Attributes

- C1 cons1

  The first constraint.
- C2 cons2

  The second constraint.

4.3.1 Detailed Description

template<typename C1, typename C2, typename G>struct CombinedConstraint< C1, C2, G >

Constraint that combines two constraints.

This struct implements a constraint that combines the effects of two constraints.

Template Parameters

G	A class that satisfies the graph concept.
C1	A class that satisfies the constraint concept.
C2	A class that satisfies the constraint concept.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<typename C1, typename C2, typename G> CombinedConstraint< C1, C2, G >::CombinedConstraint (C1 *c1* = `C1()`, C2 *c2* = `C2()`) `[inline]`

Constructor Function.

Parameters

`in`	*c1*	The first constraint
`in`	*c2*	The first constraint

4.3.3 Member Function Documentation

4.3.3.1 template<typename C1, typename C2, typename G> void CombinedConstraint< C1, C2, G >::operator() (G & *g*, double) `[inline]`

Constrain the node with the effect of two constraints.

Parameters

`in,out`	*g*	An object that satisfies the graph concept.
`in`	*t*	Time.(Not used here)

This function sets the velocity and position nodes, so nodes will act according to the combined effect of two constraints.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.4 CombinedForce< F1, F2 > Struct Template Reference

Constraint that combines two forces.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- CombinedForce (F1 f1=F1(), F2 f2=F2())

    Constructor Function.
- template<typename NODE >
  Point operator() (NODE n, double t)

    Get the force on the node with the effect of two force.

Public Attributes

- F1 force1

    The first force.
- F2 force2

    The second force.

4.4.1 Detailed Description

template<typename F1, typename F2>struct CombinedForce< F1, F2 >

Constraint that combines two forces.

This struct implements a force that combines the effects of two forces.

Template Parameters

F1	A class that satisfies the force concept.
F2	A class that satisfies the force concept.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 template<typename F1, typename F2> CombinedForce< F1, F2 >::CombinedForce (F1 *f1* = `F1()`, F2 *f2* = `F2()`) `[inline]`

Constructor Function.

Parameters

`in`	*c1*	The first force
`in`	*c2*	The first force

4.4.3 Member Function Documentation

4.4.3.1 template<typename F1, typename F2> template<typename NODE > Point CombinedForce< F1, F2 >::operator() (NODE *n*, double *t*) `[inline]`

Get the force on the node with the effect of two force.

Parameters

`in`	*n*	The node to calculate the gravity
`in`	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

the combined force of the node

This function calculates the force on a node. The force is the same calculating two forces separately and adding them together

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.5 ConstantConstraint< G > Struct Template Reference

Constraint that pins two points of a graph.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- ConstantConstraint (Point P1, Point P2)

  Constructor Function.
- void operator() (G &g, double)

  Pins two nodes.

Public Attributes

- Point p1

  The first point to be pinned.
- Point p2

  The second point to be pinned.

4.5.1 Detailed Description

template<typename G>struct ConstantConstraint< G >

Constraint that pins two points of a graph.

This struct implements a constraint that pins two nodes of a graph. This struct can define two points at any posistion.

Template Parameters

G	A class that satisfies the graph concept.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 template<typename G > ConstantConstraint< G >::ConstantConstraint (Point *P1*, Point *P2*) [inline]

Constructor Function.

Parameters

in	*P1*	The first point to be pinned
in	*P2*	The second point to be pinned

4.5.3 Member Function Documentation

4.5.3.1 template<typename G > void ConstantConstraint< G >::operator() (G & *g*, double) [inline]

Pins two nodes.

Parameters

in,out	*g*	An object that satisfies the graph concept.
in	*t*	Time.(Not used here)

This function sets the velocity and position of two nodes, so the two nodes will not move.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.6 DampingForce Struct Reference

Force Function to calculate damp force.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- DampingForce (double coef)
  Constructor Function.
- template<typename NODE >
  Point operator() (NODE n, double t)
  Calculate the damping force on a node.

Public Attributes

- double c
  The damping coefficient.

4.6.1 Detailed Description

Force Function to calculate damp force.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 DampingForce::DampingForce (double *coef*) `[inline]`

Constructor Function.

Parameters

in	*coef*	The damping coefficient

4.6.3 Member Function Documentation

4.6.3.1 template<typename NODE > Point DampingForce::operator() (NODE *n*, double *t*) `[inline]`

Calculate the damping force on a node.

Parameters

in	*n*	The node to calculate the gravity
in	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

the damping force of the node

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.7 EdgeData Struct Reference

Custom structure of data to store with Edges.

```
#include <Meshed_mass_spring.hpp>
```

Public Attributes

- double L

  Edge length.
- double K

  Edge spring constant (stiffness)

4.7.1 Detailed Description

Custom structure of data to store with Edges.

This struct stores data associated with each edge. Users can store their own data as members, and add new member functions. However, users should NOT delete the *L* and *K* defined here.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.8 GravityForce Struct Reference

Force Function to calculate gravity.

```
#include <Meshed_mass_spring.hpp>
```

Public Member Functions

- template<typename NODE >
  Point operator() (NODE n, double t)

  Calculate the gravity of a node.

4.8.1 Detailed Description

Force Function to calculate gravity.

4.8.2 Member Function Documentation

4.8.2.1 template<typename NODE > Point GravityForce::operator() (NODE *n*, double *t*) `[inline]`

Calculate the gravity of a node.

Parameters

in	*n*	The node to calculate the gravity

in	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

The gravity of the node

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.9 MassSpringForce Struct Reference

Force Function to the spring force on a node.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- template<typename NODE >
  Point operator() (NODE n, double t)

  Calculate the spring force on a node.

4.9.1 Detailed Description

Force Function to the spring force on a node.

4.9.2 Member Function Documentation

4.9.2.1 template<typename NODE > Point MassSpringForce::operator() (NODE *n*, double *t*) [inline]

Calculate the spring force on a node.

Parameters

in	*n*	The node to calculate the gravity
in	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

the spring force of the node

This function calculates the spring force of a node The force is calcuated by Hook's law. The initial length of the spring can be set by the user.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.10 NodeData Struct Reference

Custom structure of data to store with Nodes.

```
#include <Meshed_mass_spring.hpp>
```

Public Attributes

- Point velocity

  Node velocity.
- double mass

  Node mass.

4.10.1 Detailed Description

Custom structure of data to store with Nodes.

This struct stores data associated with each node. Users can store their own data as members, and add new member functions. However, users should NOT delete the *velocity* and *mass* defined here.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.11 PlaneConstraint< G > Struct Template Reference

Constraint that acts as a horizontal plane.

```
#include <Meshed_mass_spring.hpp>
```

Public Member Functions

- PlaneConstraint (double h)

  Constructor Function.
- void operator() (G &g, double)

  Constrain nodes with plane constraints.

Public Attributes

- double height

  The height of the plane.

4.11.1 Detailed Description

template<typename G>struct PlaneConstraint< G >

Constraint that acts as a horizontal plane.

This struct implements a constraint that acts like a plane. This struct can define a plane at any height.

Template Parameters

G	A class that satisfies the graph concept.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 template<typename G > PlaneConstraint< G >::PlaneConstraint (double *h*) `[inline]`

Constructor Function.

Parameters

`in`	*h*	the height of the plane

4.11.3 Member Function Documentation

4.11.3.1 template<typename G > void PlaneConstraint< G >::operator() (G & *g*, double) `[inline]`

Constrain nodes with plane constraints.

Parameters

`in,out`	*g*	An object that satisfies the graph concept.
`in`	*t*	Time.(Not used here)

This function sets the velocity and position of nodes, so nodes will act as they reach a plane and bounce back.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.12 PressureForce< NODE, G > Struct Template Reference

Force Function to the air pressure force.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- PressureForce (double p_out, double c, G ∗graph)

    Constructor Function.
- Point operator() (NODE n, double t)

    Calculate the pressure force on a node.

4.12.1 Detailed Description

template<typename NODE, typename G>struct PressureForce< NODE, G >

Force Function to the air pressure force.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 template<typename NODE , typename G > PressureForce< NODE, G >::PressureForce (double *p_out*, double *c*, G ∗ *graph*) `[inline]`

Constructor Function.

Parameters

`in`	*p_out*	The airpressure outside the sphere
`in`	*c*	The amout of air inside the sphere
`in`	*graph*	Pointer to the sphere mesh to calculate air pressure on

4.12.3 Member Function Documentation

4.12.3.1 template<typename NODE , typename G > Point PressureForce< NODE, G >::operator() (NODE *n*, double *t*) `[inline]`

Calculate the pressure force on a node.

Parameters

`in`	*n*	The node to calculate the gravity
`in`	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

the pressure force of the node

The pressure force of a node is calculated by aggreggating the pressure force of the adjacent triangles of the node. The force on one triangle is: F = (c/V - p_out)∗area. V is the volume of the sphere, and area is the area of the triangle.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.13 TriData Struct Reference

Custom structure of data to store with Triangles.

`#include <Meshed_mass_spring.hpp>`

Public Attributes

- Point n

  the outward surface normal vector of the triangle

4.13.1 Detailed Description

Custom structure of data to store with Triangles.

This struct stores data associated with each triangle. Users can store their own data as members, and add new member functions. However, users should NOT delete the *n* defined here.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

4.14 WindForce Struct Reference

Force Function to calculate wind force.

`#include <Meshed_mass_spring.hpp>`

Public Member Functions

- WindForce (Point wind)

  Constructor Function.
- template<typename NODE >
  Point operator() (NODE n, double t)

  Calculate the wind force on a node.

Public Attributes

- Point w

  The strength and direction of the wind.

4.14.1 Detailed Description

Force Function to calculate wind force.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 WindForce::WindForce (Point *wind*) `[inline]`

Constructor Function.

Parameters

`in`	*wind*	The direction and strength of the wind force

4.14.3 Member Function Documentation

4.14.3.1 template<typename NODE > Point WindForce::operator() (NODE *n*, double *t*) `[inline]`

Calculate the wind force on a node.

Parameters

`in`	*n*	The node to calculate the gravity
`in`	*t*	Time.(Not used here)

Template Parameters

NODE	a class that satisfies the Node concept

Returns

the wind force of the node

The wind force of a node is calculated to simulate the aggregated effect of the wind force on adjacent triangles of the node.

The documentation for this struct was generated from the following file:

- /Users/tianlan/Documents/code/Meshed_mass_spring.hpp

Chapter 5

File Documentation

5.1 /Users/tianlan/Documents/code/Meshed_mass_spring.hpp File Reference

Add pressure and wind force to a mesh.

```
#include <fstream>
#include "CS207/SDLViewer.hpp"
#include "CS207/Util.hpp"
#include "CS207/Color.hpp"
#include "Point.hpp"
```

Classes

- struct NodeData

  Custom structure of data to store with Nodes.
- struct EdgeData

  Custom structure of data to store with Edges.
- struct TriData

  Custom structure of data to store with Triangles.
- struct PlaneConstraint< G >

  Constraint that acts as a horizontal plane.
- struct BoxConstraint< G >

  Constraint that acts as a box.
- struct ConstantConstraint< G >

  Constraint that pins two points of a graph.
- struct CollisionConstraint< G >

  Constraint for collision between spheres.
- struct CombinedConstraint< C1, C2, G >

  Constraint that combines two constraints.
- struct GravityForce

  Force Function to calculate gravity.
- struct MassSpringForce

  Force Function to the spring force on a node.
- struct DampingForce

  Force Function to calculate damp force.
- struct WindForce

  Force Function to calculate wind force.

- struct PressureForce< NODE, G >

  Force Function to the air pressure force.
- struct CombinedForce< F1, F2 >

  Constraint that combines two forces.

Functions

- template<typename C1 , typename C2 , typename G >
  CombinedConstraint< C1, C2, G > make_combined_constraint (C1 c1, C2 c2, G &g)

  Make a combined constraint from two constraints.
- template<typename G , typename F >
  double symp_euler_step (G &g, double t, double dt, F force)

  Change nodes according to the symplectic Euler method.
- template<typename F1 , typename F2 >
  CombinedForce< F1, F2 > make_combined_force (F1 f1=F1(), F2 f2=F2())

  Make a combined force from two forces.
- template<typename F1 , typename F2 , typename F3 >
  CombinedForce< CombinedForce
  < F1, F2 >, F3 > make_combined_force (F1 force1, F2 force2, F3 force3)

  Make a combined force from three forces.

5.1.1 Detailed Description

Add pressure and wind force to a mesh.

5.1.2 Function Documentation

5.1.2.1 template<typename C1 , typename C2 , typename G > CombinedConstraint<C1, C2, G> make_combined_constraint (C1 *c1*, C2 *c2*, G & *g*)

Make a combined constraint from two constraints.

Parameters

`in`	*g*	An object that satisfies the graph concept.
`in`	*c1*	The first constraint
`in`	*c2*	The second constraint

Returns

A constraint that acts as the combined effect of two constraints.

5.1.2.2 template<typename F1 , typename F2 > CombinedForce< F1, F2 > make_combined_force (F1 *f1* = `F1()`, F2 *f2* = `F2()`)

Make a combined force from two forces.

Parameters

`in`	*f1*	The first force

`in`	*f2*	The second force

Returns

A force that acts as the combined effect of two forces.

5.1.2.3 template<typename F1 , typename F2 , typename F3 > CombinedForce<CombinedForce<F1, F2>, F3> make_combined_force (F1 *force1,* F2 *force2,* F3 *force3*)

Make a combined force from three forces.

Parameters

`in`	*f1*	The first force
`in`	*f2*	The second force
`in`	*f3*	The third force

Returns

A force that acts as the combined effect of three forces.

5.1.2.4 template<typename G , typename F > double symp_euler_step (G & *g,* double *t,* double *dt,* F *force*)

Change nodes according to the symplectic Euler method.

Parameters

`in,out`	*g*	Graph
`in`	*t*	The current time (useful for time-dependent forces)
`in`	*dt*	The time step
`in`	*force*	Function object defining the force per node

Returns

the next time step (usually *t* + *dt*)

Template Parameters

G	A class that satisfies the graph concept.
F	is a function object called as *force*(n, *t*), where n is a node of the graph and *t* is the current time. *force* must return a Point representing the force vector on Node at time *t*.

Change a graph's nodes according to a step of the symplectic Euler method with the given node force.