Tetrahedral Mesh

Yung-jen Cheng, Jeffrey Shen

CS207 Final Project

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

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 Struct Reference

Public Member Functions

- BoxConstraint (double constraint, double frictionCoef)
- void operator() (MeshType &m, double t)

Public Attributes

- double **constraint_**
- double **frictionCoef_**

4.1.1 Detailed Description

Box Constraint that constructs an impassable box

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BoxConstraint::BoxConstraint (double *constraint,* double *frictionCoef*) [inline]

BoxConstraint Constructor.

Parameters

in	*constraint*	Sets the coordinate to define the the box.
in	*frictionCoef*	friction coefficient for the friction force exerted by the box

4.1.3 Member Function Documentation

4.1.3.1 void BoxConstraint::operator() (MeshType & *m,* double *t*) [inline]

Horizontal Constraint Setter

Parameters

in	*m*	Valid mesh.
in	*t*	Valid time.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.2 CombineConstraints< Constraint1, Constraint2 > Struct Template Reference

Public Member Functions

- **CombineConstraints** (Constraint1 c1, Constraint2 c2)
- void **operator()** (MeshType &m, double t)

Public Attributes

- Constraint1 **c1_**
- Constraint2 **c2_**

4.2.1 Detailed Description

template<typename Constraint1, typename Constraint2>struct CombineConstraints< Constraint1, Constraint2 >

Combine Constraints Functor that returns a combination of constraints

Parameters

in	*Two*	valid constraints in c1 and c2.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.3 CombineForces< Force1, Force2 > Struct Template Reference

Public Member Functions

- CombineForces (Force1 f1, Force2 f2)
- Point operator() (Node n, double t)

Public Attributes

- Force1 **f1_**
- Force2 **f2_**

4.3.1 Detailed Description

template<typename Force1, typename Force2>struct CombineForces< Force1, Force2 >

Combine Force Functor that returns a combination of forces

Parameters

in	*Two*	valid forces in f1 and f2.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<typename Force1 , typename Force2 > **CombineForces**< Force1, Force2 >::**CombineForces** (Force1 *f1,* Force2 *f2*) [inline]

CombineForces Constructor.

Parameters

in	*f1*	First valid force.
in	*f2*	Second valid force.

4.3.3 Member Function Documentation

4.3.3.1 template<typename Force1 , typename Force2 > Point **CombineForces**< Force1, Force2 >::operator() (**Node** *n,* double *t*) [inline]

Calculates Combine Forces

Parameters

in	*n*	Valid node.
in	*t*	Valid time.

Returns

Point object that represents the combination of forces of *f1_* and *f2_*.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.4 DashpotForce Struct Reference

Public Member Functions

- DashpotForce (double K=100, double C=100)
- Point operator() (Node n, double t)

Public Attributes

- double **K_**
- double **C_**

4.4.1 Detailed Description

Dashpot Force Functor that returns the Dashpot Force

4.4.2 Constructor & Destructor Documentation

4.4.2.1 DashpotForce::DashpotForce (double *K* = `100`, double *C* = `100`) `[inline]`

DashpotForce Constructor.

Parameters

in	*K*	Spring constant in N/m
in	*K*	Damping constant in in N∗s/m

4.4.3 Member Function Documentation

4.4.3.1 Point DashpotForce::operator() (Node *n*, double *t*) [inline]

Calculates Dashpot Force

Parameters

in	*n*	Valid node.
in	*t*	Valid time.

Returns

Point object that represents the dashpot force.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.5 DragForce Struct Reference

Public Member Functions

- DragForce (double coeff)
- Point operator() (Node n, double t)

Public Attributes

- Point **dforce**
- double **coeff**

4.5.1 Detailed Description

Drag Force Functor that returns the force generated by mouse motion event

4.5.2 Constructor & Destructor Documentation

4.5.2.1 DragForce::DragForce (double *coeff*) [inline]

Default DragForce Constructor.

4.5.3 Member Function Documentation

4.5.3.1 Point DragForce::operator() (Node *n*, double *t*) [inline]

Calculates Gravity Force

Parameters

in	*n*	Valid node.
in	*t*	Valid time.

Returns

Point object that represents the drag force generated by mouse motion.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.6 Graph< V, E >::Edge Class Reference

Class representing the graph's edges.

```
#include <Graph.hpp>
```

Inherits totally_ordered< Edge >.

Public Member Functions

- Edge ()
- Node node1 () const
- Node node2 () const
- double length () const
- bool operator== (const Edge &e) const
- bool operator< (const Edge &e) const
- edge_value_type & value ()
- const edge_value_type & value () const

Friends

- class **Graph**

4.6.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::Edge

Class representing the graph's edges.

Edges are order-insensitive pairs of nodes. Two Edges with the same nodes are considered equal if they connect the same nodes, in either order.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 template<typename V, typename E> Graph< V, E >::Edge::Edge ( ) [inline]

Construct an invalid Edge.

4.6.3 Member Function Documentation

4.6.3.1 template<typename V, typename E> double Graph< V, E >::Edge::length ( ) const [inline]

Initial lenght

4.6.3.2 template<typename V, typename E> Node Graph< V, E >::Edge::node1 ( ) const [inline]

Return a node of this Edge

4.6.3.3 template<typename V, typename E> Node Graph< V, E >::Edge::node2 ( ) const [inline]

Return the other node of this Edge

4.6.3.4 template<typename V, typename E> bool Graph< V, E >::Edge::operator< (const Edge & *e*) const [inline]

Test whether this edge is less than *x* in the global order.

This ordering function is useful for STL containers such as std::map<>. It need not have any geometric meaning.

The edge ordering relation must obey trichotomy: For any two edges x and y, exactly one of x == y, x < y, and y < x is true.

4.6.3.5 template<typename V, typename E> bool Graph< V, E >::Edge::operator== (const Edge & *e*) const [inline]

Test whether this edge and *x* are equal.

Equal edges are from the same graph and have the same nodes.

4.6.3.6 template<typename V, typename E> edge_value_type& Graph< V, E >::Edge::value ( ) [inline]

Obtain the user defined type E stored in this edge.

Returns

the *edge_value* as a reference

Complexity: O(num_nodes()).

4.6.3.7 template<typename V, typename E> const edge_value_type& Graph< V, E >::Edge::value ( ) const [inline]

Obtain the user defined type E stored in this edge.

Returns

the *edge_value* as a const reference

Complexity: O(num_nodes()).

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.7 Mesh< N, E, T >::Edge Class Reference

Inherits totally_ordered< Edge >.

Public Member Functions

- Node node (size_type i) const
- Node node1 () const
- Node node2 () const
- edge_value_type & value ()
- const edge_value_type & value () const
- double length () const
- vector< Tetrahedral > edgeAdjTetrahedral () const
- bool operator== (const Edge &x) const
- bool operator< (const Edge &e) const

Friends

- class **Mesh**

4.7.1 Member Function Documentation

4.7.1.1 template<typename N , typename E , typename T > vector<Tetrahedral> Mesh< N, E, T >::Edge::edgeAdjTetrahedral ( ) const [inline]

Return a vector of tetrahedrals adjacent to the Edge

Precondition

Valid Edge.

Postcondition

return 1<= vector.size()

Returns

vector containing Tetrahedrals

Complexity: O(d) //From getEdgefrom2Nodes which uses the underlying graph's incident iterator

4.7.1.2 template<typename N , typename E , typename T > double Mesh< N, E, T >::Edge::length ( ) const [inline]

Return the length of the Edge

Precondition

Both nodes have valid positions.

Returns

Double length between the two nodes by Euclidean distance formula. Complexity: O(1).

4.7.1.3 template<typename N , typename E , typename T > Node Mesh< N, E, T >::Edge::node (size_type *i*) const [inline]

Return one of the two edge's nodes with uid with *i*.

Precondition

0 <= *i* < 2

Postcondition

result_node.index() == node_uid1_ if node_uid1_ < node_uid2_ else, result_node.index() == node_uid2_

Returns

Node such that if node_uid1_ < node_uid2_ returns node.index() == node_uid1_ else, returns node.index() == node_uid2_ Complexity: O(1).

4.7.1.4 template<typename N , typename E , typename T > Node Mesh< N, E, T >::Edge::node1 ( ) const [inline]

Return the node with the smaller uid of the edges 2 nodes.

Precondition

Valid Edge of the Mesh

Postcondition

result_node.index() == node_uid1_ if node_uid1_ < node_uid2_ else, result_node.index() == node_uid2_

Returns

Node such that if node_uid1_ < node_uid2_ returns node.index() == node_uid1_ else, returns node.index() == node_uid2_ Complexity: O(1).

4.7.1.5 template<typename N , typename E , typename T > Node Mesh< N, E, T >::Edge::node2 ( ) const [inline]

Return the node with the greater uid of the edges 2 nodes.

Precondition

Valid Edge of the Mesh

Postcondition

result_node.index() == node_uid2_ if node_uid1_ < node_uid2_ else, result_node.index() == node_uid1_

Returns

Node such that if node_uid1_ < node_uid2_ returns node.index() == node_uid2_ else, returns node.index() == node_uid1_ Complexity: O(1).

4.7.1.6 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Edge::operator< (const Edge & *e*) const [inline]

Test whether this edge is less than *x* in the global order. This ordering function is useful for STL containers such as std::map<>. It need not have any geometric meaning.

4.7.1.7 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Edge::operator== (const Edge & *x*) const [inline]

Test whether this edge and *x* are equal.

Parameters

in	*x*	Edge in a mesh

Returns

True if this Edge's mesh pointer is the same as *x's* mesh pointer && both nodes' uids match.

Equal edges are from the same mesh and have the same nodes.

Complexity: O(1).

4.7.1.8 template<typename N , typename E , typename T > edge_value_type& Mesh< N, E, T >::Edge::value ( ) [inline]

Retrieve the Edge's value (Modifiable)

Precondition

Valid Edge.

Returns

reference to this Edge's value.

Complexity: same as g_real_.edge.value()

4.7.1.9 template<typename N , typename E , typename T > const edge_value_type& Mesh< N, E, T >::Edge::value ( ) const [inline]

Retrieve the Edge's value (Cannot be modified)

Precondition

Valid Edge.

Returns

reference to this Edge's value.

Complexity: same as g_real_.edge.value()

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.8 EdgeData Struct Reference

Public Attributes

- double **length**

4.8.1 Detailed Description

Custom structure of data to store with Edges

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.9 Mesh< N, E, T >::EdgeIterator Class Reference

Iterator class for edges. A forward iterator.

`#include <tet_mesh.hpp>`

Inherits equality_comparable< EdgeIterator >.

Public Types

- typedef Edge value_type
- typedef Edge ∗ pointer
- typedef Edge & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- EdgeIterator ()
- Edge operator∗ () const
- EdgeIterator & operator++ ()
- bool operator== (const EdgeIterator &target) const

Friends

- class **Mesh**

4.9.1 Detailed Description

template<typename N, typename E, typename T>class Mesh< N, E, T >::EdgeIterator

Iterator class for edges. A forward iterator.

4.9.2 Member Typedef Documentation

4.9.2.1 template<typename N , typename E , typename T > typedef std::ptrdiff_t Mesh< N, E, T >::EdgeIterator::difference_type

Difference between iterators

4.9.2.2 template<typename N , typename E , typename T > typedef std::input_iterator_tag Mesh< N, E, T >::EdgeIterator::iterator_category

Iterator category.

4.9.2.3 template<typename N , typename E , typename T > typedef Edge∗ Mesh< N, E, T >::EdgeIterator::pointer

Type of pointers to elements.

4.9.2.4 template<typename N , typename E , typename T > typedef Edge& Mesh< N, E, T >::EdgeIterator::reference

Type of references to elements.

4.9.2.5 template<typename N , typename E , typename T > typedef Edge Mesh< N, E, T >::EdgeIterator::value_type

Element type.

4.9.3 Constructor & Destructor Documentation

4.9.3.1 template<typename N , typename E , typename T > Mesh< N, E, T >::EdgeIterator::EdgeIterator ( ) [inline]

Construct an invalid EdgeIterator.

4.9.4 Member Function Documentation

4.9.4.1 template<typename N , typename E , typename T > Edge Mesh< N, E, T >::EdgeIterator::operator* ( ) const [inline]

Dereference the edge iterator

Returns

the Edge corresponding to the edge in g_real_.

Complexity: same as g_real_type::EdgeIterator operator*(), probably O(num_nodes()).

4.9.4.2 template<typename N , typename E , typename T > EdgeIterator& Mesh< N, E, T >::EdgeIterator::operator++ ( ) [inline]

Increase the edge iterator

Postcondition

the *eit_* increase by 1, may point to an invalid position.

Returns

the modified EdgeIterator.

Complexity: same as g_real_type::EdgeIterator operator++(), probably O(num_nodes()).

4.9.4.3 template<typename N , typename E , typename T > bool Mesh< N, E, T >::EdgeIterator::operator== (const EdgeIterator & *target*) const [inline]

Test the equality of EdgeIterator.

Parameters

in	*target*	EdgeIterator

Returns

True if both EdgeIterators are in the same mesh and have the same eit_.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.10 Graph< V, E >::EdgeIterator Class Reference

Iterator class for edges. A forward iterator.

```
#include <Graph.hpp>
```

Inherits equality_comparable< EdgeIterator >.

Public Types

- typedef Edge value_type
- typedef Edge ∗ pointer
- typedef Edge & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- EdgeIterator ()
- Edge operator∗ () const
- EdgeIterator & operator++ ()
- bool operator== (const EdgeIterator &target) const

Friends

- class **Graph**

4.10.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::EdgeIterator

Iterator class for edges. A forward iterator.

4.10.2 Member Typedef Documentation

4.10.2.1 template<typename V, typename E> typedef std::ptrdiff_t Graph< V, E >::EdgeIterator::difference_type

Difference between iterators

4.10.2.2 template<typename V, typename E> typedef std::input_iterator_tag Graph< V, E >::EdgeIterator::iterator_category

Iterator category.

4.10.2.3 template<typename V, typename E> typedef Edge∗ Graph< V, E >::EdgeIterator::pointer

Type of pointers to elements.

4.10.2.4 template<typename V, typename E> typedef Edge& Graph< V, E >::EdgeIterator::reference

Type of references to elements.

4.10.2.5 template<typename V, typename E> typedef Edge Graph< V, E >::EdgeIterator::value_type

Element type.

4.10.3 Constructor & Destructor Documentation

4.10.3.1 template<typename V, typename E> Graph< V, E >::EdgeIterator::EdgeIterator ( ) [inline]

Construct an invalid EdgeIterator.

4.10.4 Member Function Documentation

4.10.4.1 template<typename V, typename E> Edge Graph< V, E >::EdgeIterator::operator* ( ) const [inline]

Dereference the edge iterator

Precondition

> *node1_idx_* < *nodes.size()*
> *node2_pos_* < *nodes*[node1_idx_].link_edge.size()

Returns

> Edge connecting the nodes index *node1_idx_* and *nodes*[node1_idx_].link_edge[node2_pos_]

Complexity: O(1).

4.10.4.2 template<typename V, typename E> EdgeIterator& Graph< V, E >::EdgeIterator::operator++ ( ) [inline]

Increase the edge iterator Increase the iterator to the next edge in *link_edge* of node(*node1_idx*). If it is the last one, then point to the first edge of next node. To deal with the duplicated edges stored in the *link_edges*. (Both edge(i,j) and edge(j,i) are stored) Only count the edge(i,j) if i < j, skip the ones that j < i.

Returns

> the EdgeIterator advanced to next position, or be end.

Complexity: O(1).

4.10.4.3 template<typename V, typename E> bool Graph< V, E >::EdgeIterator::operator== (const EdgeIterator & *target*) const [inline]

Test the equality of EdgeIterator.

Parameters

in	*target*	EdgeIterator

Returns

> True if both EdgeIterators are in the same graph and connecting the same two nodes.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.11 FixedConstraint Struct Reference

Public Member Functions

- **FixedConstraint** (const vector< Point > &v)
- void operator() (MeshType &m, double t)

Public Attributes

- vector< Point > **cpoints**

4.11.1 Detailed Description

Fixed Constraint where you can specify some points to be static

4.11.2 Member Function Documentation

4.11.2.1 void FixedConstraint::operator() (MeshType & *m*, double *t*) [inline]

Fixed Constraint Setter

Parameters

in	*g*	Valid mesh.
in	*t*	Valid time.

Postcondition

The velocity of Points in *cpoints* are 0.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.12 Graph< V, E > Class Template Reference

A template for 3D undirected graphs.

```
#include <Graph.hpp>
```

Classes

- class Edge

  Class representing the graph's edges.
- class EdgeIterator

  Iterator class for edges. A forward iterator.
- class IncidentIterator

  Iterator class for edges incident to a node. A forward iterator.
- class IncidentIterator

  Iterator class for edges incident to a node. A forward iterator.
- class Node

  Class representing the graph's nodes.
- class NodeIterator

  Iterator class for nodes. A forward iterator.

Public Types

- typedef Graph graph_type
- typedef V **node_value_type**
- typedef E **edge_value_type**
- typedef Node node_type
- typedef Edge edge_type
- typedef unsigned size_type
- typedef NodeIterator node_iterator
- typedef EdgeIterator edge_iterator
- typedef IncidentIterator incident_iterator

Public Member Functions

- Graph ()
- ∼Graph ()=default
- size_type size () const
- void clear ()
- size_type num_nodes () const
- Node add_node (const Point &position, const node_value_type &v=node_value_type())
- bool has_node (const Node &n) const
- Node node (size_type i) const
- size_type remove_node (const Node &n)
- node_iterator remove_node (node_iterator n_it)
- size_type num_edges () const
- Edge add_edge (const Node &a, const Node &b)
- bool has_edge (const Node &a, const Node &b) const
- Edge edge (size_type index) const
- size_type remove_edge (const Edge &e)
- size_type remove_edge (const Node &a, const Node &b)
- edge_iterator remove_edge (edge_iterator e_it)
- node_iterator node_begin () const
- node_iterator node_end () const
- edge_iterator edge_begin () const
- edge_iterator edge_end () const

4.12.1 Detailed Description

template<typename V, typename E>class Graph< V, E >

A template for 3D undirected graphs.

Users can add and retrieve nodes and edges. Edges are unique (there is at most one edge between any pair of distinct nodes).

4.12.2 Member Typedef Documentation

4.12.2.1 template<typename V, typename E> typedef EdgeIterator Graph< V, E >::edge_iterator

Synonym for EdgeIterator

4.12.2.2 template<typename V, typename E> typedef Edge Graph< V, E >::edge_type

Synonym for Edge (following STL conventions).

4.12.2.3 template<typename V, typename E> typedef Graph Graph< V, E >::graph_type

Type of this graph.

4.12.2.4 template<typename V, typename E> typedef IncidentIterator Graph< V, E >::incident_iterator

Synonym for IncidentIterator

4.12.2.5 template<typename V, typename E> typedef NodeIterator Graph< V, E >::node_iterator

Synonym for NodeIterator

4.12.2.6 template<typename V, typename E> typedef Node Graph< V, E >::node_type

Synonym for Node (following STL conventions).

4.12.2.7 template<typename V, typename E> typedef unsigned Graph< V, E >::size_type

Type of indexes and sizes. Return type of Graph::Node::index(), Graph::num_nodes(), Graph::num_edges(), and argument type of Graph::node(size_type)

4.12.3 Constructor & Destructor Documentation

4.12.3.1 template<typename V, typename E> Graph< V, E >::Graph ( ) [inline]

Construct an empty graph.

4.12.3.2 template<typename V, typename E> Graph< V, E >::~Graph ( ) [default]

Default destructor

4.12.4 Member Function Documentation

4.12.4.1 template<typename V, typename E> Edge Graph< V, E >::add_edge (const Node & *a*, const Node & *b*) [inline]

Add an edge to the graph, or return the current edge if it already exists.

Precondition

a and *b* are distinct valid nodes of this graph

Returns

an Edge object e with e.node1() == *a* and e.node2() == *b*

Postcondition

> has_edge(*a*, *b*) == true
> If old has_edge(*a*, *b*), new num_edges() == old num_edges(). Else, new num_edges() == old num_edges() + 1.

Can invalidate edge indexes – in other words, old edge(*i*) might not equal new edge(*i*). Must not invalidate outstanding Edge objects.

Complexity: No more than O(num_nodes() + num_edges()), hopefully less

4.12.4.2 template<typename V, typename E> Node Graph< V, E >::add_node (const Point & *position,* const node_value_type & *v* = `node_value_type ()`) `[inline]`

Add a node to the graph, returning the added node.

Parameters

`in`	*position*	The new node's position

Postcondition

> new size() == old size() + 1
> result_node.index() == old size()

Complexity: O(1) amortized operations.

4.12.4.3 template<typename V, typename E> void Graph< V, E >::clear ( ) `[inline]`

Remove all nodes and edges from this graph.

Postcondition

> num_nodes() == 0 && num_edges() == 0

Invalidates all outstanding Node and Edge objects.

4.12.4.4 template<typename V, typename E> Edge Graph< V, E >::edge (size_type *index*) const `[inline]`

Return the edge with index *i*.

Precondition

> 0 <= *i* < num_edges()

Complexity: No more than O(num_nodes() + num_edges()), hopefully less

4.12.4.5 template<typename V, typename E> edge_iterator Graph< V, E >::edge_begin ( ) const `[inline]`

Obtain the begin iterator of edge iterator

Returns

> the first edge in the first non-empty link_edge. The begin iterator will equal to end iterator if there is no any edge in this graph.

Complexity: O(num_nodes()).

4.12.4.6 template<typename V, typename E> **edge_iterator Graph**< V, E >::edge_end ( ) const [inline]

Obtain the end of edge iterator

Returns

the end iterator, which is defined as *node1_idx_* = *nodes.size()* and *node2_pos_* = 0.

Complexity: O(1).

4.12.4.7 template<typename V, typename E> bool **Graph**< V, E >::has_edge (const **Node** & *a*, const **Node** & *b*) const [inline]

Test whether two nodes are connected by an edge.

Precondition

a and *b* are valid nodes of this graph

Returns

true if, for some *i*, edge(*i*) connects *a* and *b*.

Complexity: No more than O(num_nodes() + num_edges()), hopefully less

4.12.4.8 template<typename V, typename E> bool **Graph**< V, E >::has_node (const **Node** & *n*) const [inline]

Determine if this Node belongs to this Graph

Returns

True if *n* is currently a Node of this Graph

Complexity: O(1).

4.12.4.9 template<typename V, typename E> **Node Graph**< V, E >::node (**size_type** *i*) const [inline]

Return the node with index *i*.

Precondition

0 <= *i* < num_nodes()

Postcondition

result_node.index() == i

Complexity: O(1).

4.12.4.10 template<typename V, typename E> **node_iterator Graph**< V, E >::node_begin ( ) const [inline]

Obtain a node_iterator pointing to the start of the graph's nodes.

Returns

a node_iterator at the beginning position of the graph's nodes, it could be invalid if there is no node in the graph.

Complexity: O(1).

4.12.4.11 template<typename V, typename E> node_iterator Graph< V, E >::node_end ( ) const [inline]

Obtain a node_iterator representing the end of the graph's nodes.

Returns

a node_iterator with index = *nodes.size()*

Complexity: O(1).

4.12.4.12 template<typename V, typename E> size_type Graph< V, E >::num_edges ( ) const [inline]

Return the total number of edges in the graph.

Complexity: No more than O(num_nodes() + num_edges()), hopefully less

4.12.4.13 template<typename V, typename E> size_type Graph< V, E >::num_nodes ( ) const [inline]

Synonym for size().

4.12.4.14 template<typename V, typename E> size_type Graph< V, E >::remove_edge (const Edge & *e*) [inline]

Remove an edge in the graph.

Parameters

in	*Edge*	The Edge we want to remove

Returns

0 if *e* is removed. return num_edge() if the edge is not in this graph.

Postcondition

new num_edge() == old num_edge() - 1 if edge was in this graph and removed. new num_edge() == old num_edge() if *e* is not in this graph. all former created Edge objects and edge iterators may be invalidated after an edge is removed.

Complexity: O(num_nodes()).

4.12.4.15 template<typename V, typename E> size_type Graph< V, E >::remove_edge (const Node & *a*, const Node & *b*) [inline]

Remove an edge in the graph.

Parameters

in	*Node*	The nodes *a* and *b* connecting the edge we want to remove

Precondition

a and *b* are in the same graph.

Returns

0 if the edge(*a*,*b*) is removed. return num_edge() if the edge is not in the graph.

Postcondition

> new num_edge() == old num_edge() - 1 if edge(*a*,*b*) was in this graph and removed. new num_edge() == old num_edge() if the edge(*a*,*b*) is not in this graph. all former created Edge objects and edge iterators may be invalidated after an edge is removed.

Complexity: O(num_nodes()).

4.12.4.16 template<typename V, typename E> edge_iterator Graph< V, E >::remove_edge (edge_iterator *e_it*) [inline]

Remove an edge in the graph.

Parameters

in	*EdgeIterator*	The iterator *e_it* pointing to the edge we want to remove

Precondition

> e_it can be dereferenced.

Returns

> the next edge of *e_it* if ∗e_it is removed. return *e_it* if the edge is not in the graph.

Postcondition

> new num_edge() == old num_edge() - 1 if ∗e_it was in this graph and removed. new num_edge() == old num_edge() if ∗e_it is not in this graph. all former created Edge objects and edge iterators may be invalidated after an edge is removed.

Complexity: O(num_nodes()).

4.12.4.17 template<typename V, typename E> size_type Graph< V, E >::remove_node (const Node & *n*) [inline]

Remove a node in the graph.

Parameters

in	*Node*	The node we want to remove

Returns

> the index of removed node if *n* was in this graph and removed. return the size() if *n* is not in this graph;

Postcondition

> new size() == old size() - 1 if *n* is removed. new size() == old size() if *n* is not in this graph. all former created Node objects and node iterators may be invalidated after a node is removed.

Complexity: O(num_nodes()$^\wedge$2).

4.12.4.18 template<typename V, typename E> node_iterator Graph< V, E >::remove_node (node_iterator *n_it*) [inline]

Remove a node in the graph.

Parameters

in	*NodeIterator*	The iterator pointing to the node we want to remove.

Precondition

> *n_it* can be dereferenced.

Returns

> the iterator pointing to the next node of removed node if $*$*n_it* was in graph and removed. return end() if *n_it* is not pointing to a node in this graph;

Postcondition

> new size() == old size() - 1 if $*$*n_it* is removed. new size() == old size() if $*$*n_it* is not in this graph. all former created Node objects and node iterators may be invalidated after a node is removed.

Complexity: O(num_nodes()^2).

4.12.4.19 template<typename V, typename E> size_type Graph< V, E >::size ( ) const [inline]

Return the number of nodes in the graph.

Complexity: O(1).

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.13 GravityForce Struct Reference

Public Member Functions

- GravityForce (double g=grav)
- Point operator() (Node n, double t)

Public Attributes

- double **gravity_**

4.13.1 Detailed Description

Gravity Force Functor that returns the Gravity Force

4.13.2 Constructor & Destructor Documentation

4.13.2.1 GravityForce::GravityForce (double *g* = grav) [inline]

GravityForce Constructor.

Parameters

in	*g*	Gravity in m/s^2.

4.13.3 Member Function Documentation

4.13.3.1 Point GravityForce::operator() (Node *n*, double *t*) [inline]

Calculates Gravity Force

Parameters

in	*n*	Valid node.
in	*t*	Valid time.

Returns

> Point object that represents the gravity force calculated as m∗g.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.14 HoldConstraint Struct Reference

Public Member Functions

- HoldConstraint ()
- void operator() (MeshType &m, double t)

Public Attributes

- bool **hold**
- bool **pressed**

4.14.1 Detailed Description

Hold Constraint to stop objects moving when mouse button pressing on them

4.14.2 Constructor & Destructor Documentation

4.14.2.1 HoldConstraint::HoldConstraint ( ) [inline]

Default constructor of Hold Constraint. default setting of *hold* and *pressed* are false.

4.14.3 Member Function Documentation

4.14.3.1 void HoldConstraint::operator() (MeshType & *m*, double *t*) [inline]

Fixed Constraint Setter

Parameters

in	*g*	Valid mesh.
in	*t*	Valid time.

Postcondition

> The velocity of all points of this object are 0 when *hold* == true.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.15 HPlaneConstraint Struct Reference

Public Member Functions

- HPlaneConstraint (double z_constraint)
- void operator() (MeshType &m, double t)

Public Attributes

- double **z_constraint_**

4.15.1 Detailed Description

Horizontal Plane Constraint that models an impassable plane.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 HPlaneConstraint::HPlaneConstraint (double *z_constraint*) [inline]

HPlaneConstraint Constructor.

Parameters

in	*z_constraint*	Sets the z-coordinate to define the horizontal plane.

4.15.3 Member Function Documentation

4.15.3.1 void HPlaneConstraint::operator() (MeshType & *m*, double *t*) [inline]

Horizontal Constraint Setter

Parameters

in	*g*	Valid mesh.
in	*t*	Valid time.

Postcondition

> The velocity of *z_constraint_* is 0 and is set to the closest point to the horizontal plane as defined by *z_constraint_*.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.16 Mesh< N, E, T >::IncidentIterator Class Reference

Inherits equality_comparable< IncidentIterator >.

Public Types

- typedef Edge value_type
- typedef Edge ∗ pointer
- typedef Edge & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- IncidentIterator ()
- Edge operator∗ () const
- IncidentIterator & operator++ ()
- bool operator== (const IncidentIterator &target) const

Friends

- class **Mesh**

4.16.1 Member Typedef Documentation

4.16.1.1 template<typename N , typename E , typename T > typedef std::ptrdiff_t Mesh< N, E, T >::IncidentIterator::difference_type

Difference between iterators

4.16.1.2 template<typename N , typename E , typename T > typedef std::input_iterator_tag Mesh< N, E, T >::IncidentIterator::iterator_category

Iterator category.

4.16.1.3 template<typename N , typename E , typename T > typedef Edge∗ Mesh< N, E, T >::IncidentIterator::pointer

Type of pointers to elements.

4.16.1.4 template<typename N , typename E , typename T > typedef Edge& Mesh< N, E, T >::IncidentIterator::reference

Type of references to elements.

4.16.1.5 template<typename N , typename E , typename T > typedef Edge Mesh< N, E, T >::IncidentIterator::value_type

Element type.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 template<typename N , typename E , typename T > Mesh< N, E, T >::IncidentIterator::IncidentIterator ( ) [inline]

Construct an invalid IncidentIterator.

4.16.3 Member Function Documentation

4.16.3.1 template<typename N , typename E , typename T > Edge Mesh< N, E, T >::IncidentIterator::operator* ( ) const [inline]

Dereference the incident iterator

Precondition

NodeIterator != node.edge_end().

Returns

the Edge connecting nodes *node1_idx_* and *graph_->nodes*[node1_idx_].link_edge[node2_pos_]

Complexity: O(1).

4.16.3.2 template<typename N , typename E , typename T > IncidentIterator& Mesh< N, E, T >::IncidentIterator::operator++ ( ) [inline]

Increase the incident iterator.

Postcondition

Increase *node2_pos_* to the next index in the link_edge of current node.
*node2_pos_* will not increase if incident iterator equals to the end iterator.

Returns

the advanced IncidentIterator, may be valid or invalid position.

Complexity: O(1).

4.16.3.3 template<typename N , typename E , typename T > bool Mesh< N, E, T >::IncidentIterator::operator== (const IncidentIterator & *target*) const [inline]

Compare the equality of IncidentIterator.

Returns

True if the two IncidentIterators are in the same graph, have the same index of two sides of nodes.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.17 Graph< V, E >::IncidentIterator Class Reference

Iterator class for edges incident to a node. A forward iterator.

`#include <tet_mesh.hpp>`

Inherits equality_comparable< IncidentIterator >.

Public Types

- typedef Edge value_type
- typedef Edge * pointer
- typedef Edge & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- IncidentIterator ()
- Edge operator* () const
- IncidentIterator & operator++ ()
- bool operator== (const IncidentIterator &target) const

Friends

- class **Graph**

4.17.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::IncidentIterator

Iterator class for edges incident to a node. A forward iterator.

4.17.2 Member Typedef Documentation

4.17.2.1 template<typename V, typename E> typedef std::ptrdiff_t Graph< V, E >::IncidentIterator::difference_type

Difference between iterators

4.17.2.2 template<typename V, typename E> typedef std::input_iterator_tag Graph< V, E >::IncidentIterator::iterator_category

Iterator category.

4.17.2.3 template<typename V, typename E> typedef Edge* Graph< V, E >::IncidentIterator::pointer

Type of pointers to elements.

4.17.2.4 template<typename V, typename E> typedef Edge& Graph< V, E >::IncidentIterator::reference

Type of references to elements.

4.17.2.5 template<typename V, typename E> typedef Edge Graph< V, E >::IncidentIterator::value_type

Element type.

4.17.3 Constructor & Destructor Documentation

4.17.3.1 template<typename V, typename E> Graph< V, E >::IncidentIterator::IncidentIterator ( ) [inline]

Construct an invalid IncidentIterator.

4.17.4 Member Function Documentation

4.17.4.1 template<typename V, typename E> Edge Graph< V, E >::IncidentIterator::operator* ( ) const [inline]

Dereference the incident iterator

Precondition

> NodeIterator != node.edge_end().

Returns

> the Edge connecting nodes *node1_idx_* and *graph_->nodes*[node1_idx_].link_edge[node2_pos_]

Complexity: O(1).

4.17.4.2 template<typename V, typename E> IncidentIterator& Graph< V, E >::IncidentIterator::operator++ ( ) [inline]

Increase the incident iterator.

Postcondition

> Increase *node2_pos_* to the next index in the link_edge of current node.
> *node2_pos_* will not increase if incident iterator equals to the end iterator.

Returns

> the advanced IncidentIterator, may be valid or invalid position.

Complexity: O(1).

4.17.4.3 template<typename V, typename E> bool Graph< V, E >::IncidentIterator::operator== (const IncidentIterator & *target*) const [inline]

Compare the equality of IncidentIterator.

Returns

> True if the two IncidentIterators are in the same graph, have the same index of two sides of nodes.

The documentation for this class was generated from the following files:

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp
- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.18 Graph< V, E >::IncidentIterator Class Reference

Iterator class for edges incident to a node. A forward iterator.

```
#include <tet_mesh.hpp>
```

Inherits equality_comparable< IncidentIterator >.

Public Types

- typedef Edge value_type
- typedef Edge * pointer
- typedef Edge & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- IncidentIterator ()
- Edge operator* () const
- IncidentIterator & operator++ ()
- bool operator== (const IncidentIterator &target) const

Friends

- class **Graph**

4.18.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::IncidentIterator

Iterator class for edges incident to a node. A forward iterator.

4.18.2 Member Typedef Documentation

4.18.2.1 template<typename V, typename E> typedef std::ptrdiff_t Graph< V, E >::IncidentIterator::difference_type

Difference between iterators

4.18.2.2 template<typename V, typename E> typedef std::input_iterator_tag Graph< V, E >::IncidentIterator::iterator_category

Iterator category.

4.18.2.3 template<typename V, typename E> typedef Edge* Graph< V, E >::IncidentIterator::pointer

Type of pointers to elements.

4.18.2.4 template<typename V, typename E> typedef Edge& Graph< V, E >::IncidentIterator::reference

Type of references to elements.

4.18.2.5 template<typename V, typename E> typedef Edge Graph< V, E >::IncidentIterator::value_type

Element type.

4.18.3 Constructor & Destructor Documentation

4.18.3.1 template<typename V, typename E> Graph< V, E >::IncidentIterator::IncidentIterator ( ) [inline]

Construct an invalid IncidentIterator.

4.18.4 Member Function Documentation

4.18.4.1 template<typename V, typename E> Edge Graph< V, E >::IncidentIterator::operator∗ ( ) const [inline]

Dereference the incident iterator

Precondition

NodeIterator != node.edge_end().

Returns

the Edge connecting nodes *node1_idx_* and *graph_->nodes*[node1_idx_].link_edge[node2_pos_]

Complexity: O(1).

4.18.4.2 template<typename V, typename E> IncidentIterator& Graph< V, E >::IncidentIterator::operator++ ( ) [inline]

Increase the incident iterator.

Postcondition

Increase *node2_pos_* to the next index in the link_edge of current node.
*node2_pos_* will not increase if incident iterator equals to the end iterator.

Returns

the advanced IncidentIterator, may be valid or invalid position.

Complexity: O(1).

4.18.4.3 template<typename V, typename E> bool Graph< V, E >::IncidentIterator::operator== (const IncidentIterator & *target*) const [inline]

Compare the equality of IncidentIterator.

Returns

True if the two IncidentIterators are in the same graph, have the same index of two sides of nodes.

The documentation for this class was generated from the following files:

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp
- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.19 Listener_Wind Struct Reference

Inherits ViewerCallback.

Public Member Functions

- **Listener_Wind** (WindForce &w, double increment)
- void **operator()** (const SDL_Event &event)

Public Attributes

- WindForce & **wind**
- Point **pre_level**
- double **increment_**

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.20 makePatterns Struct Reference

Public Member Functions

- **makePatterns** (const float &longestPath)
- CS207::Color **operator()** (const MeshType::Node &n) const

Public Attributes

- float **longestPath_** = 1.0

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.21 Mesh$<$ N, E, T $>$ Class Template Reference

A template for 3D tetrahedral meshes.

```
#include <tet_mesh.hpp>
```

Classes

- class Edge
- class EdgeIterator

  Iterator class for edges. A forward iterator.
- class IncidentIterator
- class Node
- class NodeIterator

  Iterator class for nodes. A forward iterator.
- class Tetrahedral
- class TetrahedralIterator

  Iterator class for Tetrahedrals. A forward iterator.

Public Types

- typedef N node_value_type
- typedef E **edge_value_type**
- typedef T **tet_value_type**
- typedef unsigned **size_type**
- typedef Graph< node_value_type, internal_edge > **g_real_type**
- typedef Graph < internal_tetrahedral, bool > **g_tet_type**
- typedef Node **node_type**
- typedef Edge **edge_type**
- typedef NodeIterator node_iterator
- typedef EdgeIterator edge_iterator
- typedef IncidentIterator incident_iterator
- typedef TetrahedralIterator tet_iterator

Public Member Functions

- size_type num_nodes () const
- size_type num_edges () const
- size_type num_tetrahedral () const
- node_iterator node_begin () const
- node_iterator node_end () const
- edge_iterator edge_begin () const
- edge_iterator edge_end () const
- tet_iterator tetrahedral_begin () const
- tet_iterator tetrahedral_end () const
- Node node (size_type i) const
- Tetrahedral tetrahedral (size_type i) const
- Node add_node (const Point &p, const node_value_type &node_value)
- Tetrahedral add_tetrahedral (const Node &n0, const Node &n1, const Node &n2, const Node &n3, const tet_value_type &tet_value=tet_value_type())

4.21.1 Detailed Description

template<typename N, typename E, typename T>class Mesh< N, E, T >

A template for 3D tetrahedral meshes.

Users can add tetrahedrals and retrieve nodes, edges, and tetrahedrals.

4.21.2 Member Typedef Documentation

4.21.2.1 template<typename N , typename E , typename T > typedef EdgeIterator Mesh< N, E, T >::edge_iterator

Synonym for EdgeIterator

4.21.2.2 template<typename N , typename E , typename T > typedef IncidentIterator Mesh< N, E, T >::incident_iterator

Synonym for IncidentIterator

4.21.2.3 template<typename N , typename E , typename T > typedef NodeIterator Mesh< N, E, T >::node_iterator

Synonym for NodeIterator

4.21.2.4 template<typename N , typename E , typename T > typedef N Mesh< N, E, T >::node_value_type

Type of indexes and sizes. Return type of Mesh::num_nodes().

4.21.2.5 template<typename N , typename E , typename T > typedef TetrahedralIterator Mesh< N, E, T >::tet_iterator

Synonym for TetrahedralIterator

4.21.3 Member Function Documentation

4.21.3.1 template<typename N , typename E , typename T > Node Mesh< N, E, T >::add_node (const Point & *p,* const node_value_type & *node_value*) `[inline]`

Add a node to the mesh, returning the added node.

Parameters

`in`	*p*	The new node's position node_value user defined node_value

Postcondition

new g_real_.size() == old g_real_.size() + 1
result_node.index() == old g_real_.size()

Complexity: O(1) amortized operations.

4.21.3.2 template<typename N , typename E , typename T > Tetrahedral Mesh< N, E, T >::add_tetrahedral (const Node & *n0,* const Node & *n1,* const Node & *n2,* const Node & *n3,* const tet_value_type & *tet_value* = `tet_value_type()`) `[inline]`

Add a tetrahedral to the mesh, return the added tetrahedral.

Parameters

`in`	*n0,n1,n2,n3*	The new tetrahedral's four nodes tet_value: User defined tet_value

Returns

a Tetrahedral object tet with tet.node(0) is min(*n0*, *n1*, *n2*,) tet.node(1) is the 2nd smallest among (*n0*, *n1*, *n2*), tet.node(2) is the 3rd smallest among (*n0*, *n1*, *n2*), and tet.node(3) is max(*n0*, *n1*, *n2*,)

Precondition

n0, *n1*, *n2*, *n3* are valid Mesh::Node
Tetrahedral compsed of *n0*, *n1*, *n2*, *n3*

Postcondition

new g_tet_.size() == old g_tet_.size() + 1 result_tet.index() == old g_tet_.size() Complexity: same as g_real↩_.add_node()

4.21.3.3 template<typename N , typename E , typename T > edge_iterator Mesh< N, E, T >::edge_begin ( ) const [inline]

Obtain the begin iterator of edge iterator

Returns

the first edge iterator with *eit_* = g_real_.edge_begin(), it could be invalid if there is no edge in the mesh.

Complexity: same as g_real_.edge_begin(), probably O(num_nodes()).

4.21.3.4 template<typename N , typename E , typename T > edge_iterator Mesh< N, E, T >::edge_end ( ) const [inline]

Obtain the end of edge iterator

Returns

the end edge iterator, set eit_ = g_real_.edge_end()

Complexity: same as g_real_.edge_end(), probably O(1).

4.21.3.5 template<typename N , typename E , typename T > Node Mesh< N, E, T >::node (size_type *i*) const [inline]

Return the total number of nodes in the mesh. Complexity: same as g_real_.num_nodes(), probably O(1)

4.21.3.6 template<typename N , typename E , typename T > node_iterator Mesh< N, E, T >::node_begin ( ) const [inline]

Obtain a node_iterator pointing to the start of the mesh's nodes.

Returns

a node_iterator with *nit_* = *g_real_.node_begin()*, it could be invalid if there is no node in the mesh.

Complexity: same as g_real.node_begin(), probably O(1).

4.21.3.7 template<typename N , typename E , typename T > node_iterator Mesh< N, E, T >::node_end ( ) const [inline]

Obtain a node_iterator representing the end of the mesh's nodes.

Returns

a node_iterator with *nit_* = *g_real_.node_end()*

Complexity: same as g_real_.node_end(), probably O(1).

4.21.3.8 template<typename N , typename E , typename T > size_type Mesh< N, E, T >::num_edges ( ) const [inline]

Return the number of edges in the mesh.

4.21.3.9 template<typename N , typename E , typename T > size_type Mesh< N, E, T >::num_nodes ( ) const [inline]

Return the number of nodes in the mesh.

4.21.3.10 template<typename N , typename E , typename T > size_type Mesh< N, E, T >::num_tetrahedral ( ) const [inline]

Return the number of tetrahedrals in the mesh.

4.21.3.11 template<typename N , typename E , typename T > Tetrahedral Mesh< N, E, T >::tetrahedral (size_type *i*) const [inline]

Return the total number of tetrahedrals in the mesh. Complexity: same as g_tet_.size(), probably O(1)

4.21.3.12 template<typename N , typename E , typename T > tet_iterator Mesh< N, E, T >::tetrahedral_begin ( ) const [inline]

Obtain a tet_iterator pointing to the start of the mesh's tetrahedral.

Returns

> a tet_iterator at the beginning position of the mesh's tetrahedrals, it could be invalid if there is no tetrahedral in this mesh.

Complexity: same as g_tet_.node_begin(), probably O(1).

4.21.3.13 template<typename N , typename E , typename T > tet_iterator Mesh< N, E, T >::tetrahedral_end ( ) const [inline]

Obtain a tet_iterator representing the end of the mesh's tetrahedral.

Returns

> a tet_iterator with index = *num_tetrahedrals()*

Complexity: same as g_tet_.node_end(), probably O(1).

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.22 MouseLeftClickCallback Class Reference

Inherits ViewerCallback.

Public Member Functions

- MouseLeftClickCallback (MeshType &m, ViewerType &v, DragForce &df, HoldConstraint &hc)
- void operator() (const SDL_Event &event)

4.22.1 Constructor & Destructor Documentation

4.22.1.1 MouseLeftClickCallback::MouseLeftClickCallback (MeshType & *m*, ViewerType & *v*, DragForce & *df*, HoldConstraint & *hc*) [inline]

Constructor of MouseLeftClickCallback

Parameters

in	*m*	the mesh that will be listened
in	*v*	the viewer that will interact
in	*df*	the drag force impact on the mesh *m*
in	*hc*	the hold constraint constrains the mesh *m*

4.22.2 Member Function Documentation

4.22.2.1 void MouseLeftClickCallback::operator() (const SDL_Event & *event*) [inline]

Functor executed when mouse button event happens. Monitor the pressing and releasing of mouse left key.

Parameters

in	*SDL_EVENT*	The event that occurred.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.23 MouseLeftDragCallback Class Reference

Inherits ViewerCallback.

Public Member Functions

- MouseLeftDragCallback (ViewerType &v, DragForce &df, HoldConstraint &hc)
- void operator() (const SDL_Event &event)

4.23.1 Constructor & Destructor Documentation

4.23.1.1 MouseLeftDragCallback::MouseLeftDragCallback (ViewerType & *v*, DragForce & *df*, HoldConstraint & *hc*) [inline]

Constructor of MouseLeftDragCallback

Parameters

in	*v*	the viewer that will interact
in	*df*	the drag force impact on the mesh *m*
in	*hc*	the hold constraint constrains the mesh *m*

4.23.2 Member Function Documentation

4.23.2.1 void MouseLeftDragCallback::operator() (const SDL_Event & *event*) [inline]

Functor executed when mouse motion event happens. Monitor the mouse motion when mouse left key is pressed.

Parameters

in	*SDL_EVENT*	The event that occurred.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.24 Graph< V, E >::Node Class Reference

Class representing the graph's nodes.

`#include <Graph.hpp>`

Inherits totally_ordered< Node >.

Public Member Functions

- Node ()
- Point & position ()
- const Point & position () const
- size_type index () const
- bool operator== (const Node &n) const
- bool operator< (const Node &n) const
- node_value_type & value ()
- const node_value_type & value () const
- size_type degree () const
- incident_iterator edge_begin () const
- incident_iterator edge_end () const

Friends

- class **Graph**
- std::ostream & **operator<<** (std::ostream &stream, const Node &n)

4.24.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::Node

Class representing the graph's nodes.

Node objects are used to access information about the Graph's nodes.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 template<typename V, typename E> Graph< V, E >::Node::Node ( ) [inline]

Construct an invalid node.

Valid nodes are obtained from the Graph class, but it is occasionally useful to declare an invalid node, and assign a valid node to it later. For example:

```
Graph::node_type x;
if (...should pick the first node...)
  x = graph.node(0);
else
  x = some other node using a complicated calculation
do_something(x);
```

4.24.3 Member Function Documentation

4.24.3.1 template<typename V, typename E> **size_type Graph**< V, E >::Node::degree ( ) const [inline]

Obtain the number of incidents edges connected to this node

Returns

the number of edges connected to this node s.t. 0 <= degree() < num_nodes()

Complexity: O(1).

4.24.3.2 template<typename V, typename E> **incident_iterator Graph**< V, E >::Node::edge_begin ( ) const [inline]

Obtain an incident iterator pointing to the first incident edge of this node.

Returns

an incident_iterator pointing to the first edge connecting to this node.

Complexity: O(1).

4.24.3.3 template<typename V, typename E> **incident_iterator Graph**< V, E >::Node::edge_end ( ) const [inline]

Obtain an incident iterator represents the end of incident iterator.

Returns

an incident_iterator with idx_ equal to the number of incident edges. If there is no edge connecting to this node, this iterator will equal to the begin iterator. i.e. *graph_*->*nodes*[idx_].link_edge.size() == 0

Complexity: O(1).

4.24.3.4 template<typename V, typename E> **size_type Graph**< V, E >::Node::index ( ) const [inline]

Return this node's index, a number in the range [0, graph_size).

4.24.3.5 template<typename V, typename E> bool **Graph**< V, E >::Node::operator< (const **Node** & *n*) const [inline]

Test whether this node is less than *x* in the global order.

This ordering function is useful for STL containers such as std::map<>. It need not have any geometric meaning.

The node ordering relation must obey trichotomy: For any two nodes x and y, exactly one of x == y, x < y, and y < x is true.

4.24.3.6 template<typename V, typename E> bool **Graph**< V, E >::Node::operator== (const **Node** & *n*) const [inline]

Test whether this node and *x* are equal.

Equal nodes have the same graph and the same index.

4.24.3.7 template<typename V, typename E> Point& Graph< V, E >::Node::position ( ) [inline]

Return this node's position modifiable.

4.24.3.8 template<typename V, typename E> const Point& Graph< V, E >::Node::position ( ) const [inline]

Return this node's position.

4.24.3.9 template<typename V, typename E> node_value_type& Graph< V, E >::Node::value ( ) [inline]

Obtain the user defined type V stored in this node.

Returns

the *node_value* as a reference

Complexity: O(1).

4.24.3.10 template<typename V, typename E> const node_value_type& Graph< V, E >::Node::value ( ) const [inline]

Obtain the user defined type V stored in this node.

Returns

the *node_value* as a const reference

Complexity: O(1).

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.25 Mesh< N, E, T >::Node Class Reference

Inherits totally_ordered< Node >.

Public Member Functions

- Point & position ()
- const Point & position () const
- size_type index () const
- node_value_type & value ()
- const node_value_type & value () const
- vector< Tetrahedral > nodeAdjTetrahedral () const
- bool operator== (const Node &x) const
- bool operator< (const Node &n) const
- incident_iterator edge_begin () const
- incident_iterator edge_end () const

Friends

- class **Mesh**

4.25.1 Member Function Documentation

4.25.1.1 template<typename N , typename E , typename T > incident_iterator Mesh< N, E, T >::Node::edge_begin ( ) const [inline]

Obtain an incident iterator pointing to the first incident edge of this node.

Returns

an incident_iterator pointing to the first edge connecting to this node. Complexity: O(1).

4.25.1.2 template<typename N , typename E , typename T > incident_iterator Mesh< N, E, T >::Node::edge_end ( ) const [inline]

Obtain an incident iterator represents the end of incident iterator.

Returns

an incident_iterator with idx_ equal to the number of incident edges. Complexity: O(1).

4.25.1.3 template<typename N , typename E , typename T > size_type Mesh< N, E, T >::Node::index ( ) const [inline]

Return this node's index, a number in the range [0, g_real.graph_size).

Returns

The node's index i, s.t. 0 <= i < g_real.num_nodes() Complexity O(1)

4.25.1.4 template<typename N , typename E , typename T > vector<Tetrahedral> Mesh< N, E, T >::Node::nodeAdjTetrahedral ( ) const [inline]

Return a vector of tetrahedrals adjacent to the Node

Precondition

Valid Node.

Postcondition

return 0 <= result_vector.size() <= num_tetrahedrals()

Returns

vector containing Tetrahedrals

Complexity: O(g_real_.node(node_uid_).degree())

4.25.1.5 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Node::operator< (const Node & *n*) const [inline]

Test whether this node is less than *x* in the global order. This ordering function is useful for STL containers such as std::map<>. It need not have any geometric meaning. Complexity O(1)

4.25.1.6 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Node::operator== (const Node & *x*) const [inline]

Test whether this node and *x* are equal.

Parameters

in		a x is a node

Returns

True if this node has the same mesh pointer and uid; otherwise False.

Complexity O(1)

4.25.1.7 template<typename N , typename E , typename T > Point& Mesh< N, E, T >::Node::position ( ) [inline]

Return this node's position.

Returns

The node's Point object Complexity O(1)

4.25.1.8 template<typename N , typename E , typename T > const Point& Mesh< N, E, T >::Node::position ( ) const [inline]

Return this node's position as a constant.

Returns

The node's Point object Complexity O(1)

4.25.1.9 template<typename N , typename E , typename T > node_value_type& Mesh< N, E, T >::Node::value ( ) [inline]

Get this node's value (modifiable).

Returns

This node's node_value_type value as a reference. Complexity O(1)

4.25.1.10 template<typename N , typename E , typename T > const node_value_type& Mesh< N, E, T >::Node::value ( ) const [inline]

Get this node's value (non-modifiable).

Returns

This node's node_value_type value as a constant. Complexity O(1)

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.26 NodeData Struct Reference

Public Attributes

- Point **velocity**
- double **mass**
- double **color**

4.26.1 Detailed Description

Custom structure of data to store with Nodes

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.27 Mesh< N, E, T >::NodeIterator Class Reference

Iterator class for nodes. A forward iterator.

`#include <tet_mesh.hpp>`

Inherits equality_comparable< NodeIterator >.

Public Types

- typedef Node value_type
- typedef Node ∗ pointer
- typedef Node & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- NodeIterator ()
- Node operator∗ () const
- NodeIterator & operator++ ()
- bool operator== (const NodeIterator &target) const

Friends

- class **Mesh**

4.27.1 Detailed Description

template<typename N, typename E, typename T>class Mesh< N, E, T >::NodeIterator

Iterator class for nodes. A forward iterator.

4.27.2 Member Typedef Documentation

4.27.2.1 template<typename N , typename E , typename T > typedef std::ptrdiff_t Mesh< N, E, T >::NodeIterator::difference_type

Difference between iterators

4.27.2.2 template<typename N , typename E , typename T > typedef std::input_iterator_tag Mesh< N, E, T >::NodeIterator::iterator_category

Iterator category.

4.27.2.3 template<typename N , typename E , typename T > typedef Node∗ Mesh< N, E, T >::NodeIterator::pointer

Type of pointers to elements.

4.27.2.4 template<typename N , typename E , typename T > typedef Node& Mesh< N, E, T >::NodeIterator::reference

Type of references to elements.

4.27.2.5 template<typename N , typename E , typename T > typedef Node Mesh< N, E, T >::NodeIterator::value_type

Element type.

4.27.3 Constructor & Destructor Documentation

4.27.3.1 template<typename N , typename E , typename T > Mesh< N, E, T >::NodeIterator::NodeIterator ( ) [inline]

Construct an invalid NodeIterator.

4.27.4 Member Function Documentation

4.27.4.1 template<typename N , typename E , typename T > Node Mesh< N, E, T >::NodeIterator::operator∗ ( ) const [inline]

Obtain the abstract node this iterator pointing.

Returns

Node corresponding to the node in *g_real_*.

Complexity: same as g_real_type::NodeIterator operator∗(), probably O(1).

4.27.4.2 template<typename N , typename E , typename T > NodeIterator& Mesh< N, E, T >::NodeIterator::operator++ ( ) [inline]

Increment NodeIterator and return the next position.

Postcondition

the *nit_* increase by 1, may point to an invalid position.

Returns

the modified NodeIterator.

Complexity: same as g_real_type::NodeIterator operator++(), probably O(1).

4.27.4.3 template<typename N , typename E , typename T > bool Mesh< N, E, T >::NodeIterator::operator== (const NodeIterator & *target*) const [inline]

Test the equality of NodeIterator:

Returns

true if the two NodeIterators belong to the same mesh, have the same *nit_*.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.28 Graph< V, E >::NodeIterator Class Reference

Iterator class for nodes. A forward iterator.

`#include <Graph.hpp>`

Inherits equality_comparable< NodeIterator >.

Public Types

- typedef Node value_type
- typedef Node ∗ pointer
- typedef Node & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- NodeIterator ()
- Node operator∗ () const
- NodeIterator & operator++ ()
- bool operator== (const NodeIterator &target) const

Friends

- class **Graph**

4.28.1 Detailed Description

template<typename V, typename E>class Graph< V, E >::NodeIterator

Iterator class for nodes. A forward iterator.

4.28.2 Member Typedef Documentation

4.28.2.1 template<typename V, typename E> typedef std::ptrdiff_t Graph< V, E >::NodeIterator::difference_type

Difference between iterators

4.28.2.2 template<typename V, typename E> typedef std::input_iterator_tag Graph< V, E >::NodeIterator::iterator_category

Iterator category.

4.28.2.3 template<typename V, typename E> typedef **Node*** **Graph**< V, E >::**NodeIterator::pointer**

Type of pointers to elements.

4.28.2.4 template<typename V, typename E> typedef **Node&** **Graph**< V, E >::**NodeIterator::reference**

Type of references to elements.

4.28.2.5 template<typename V, typename E> typedef **Node** **Graph**< V, E >::**NodeIterator::value_type**

Element type.

4.28.3 Constructor & Destructor Documentation

4.28.3.1 template<typename V, typename E> **Graph**< V, E >::NodeIterator::NodeIterator ( ) `[inline]`

Construct an invalid NodeIterator.

4.28.4 Member Function Documentation

4.28.4.1 template<typename V, typename E> **Node** **Graph**< V, E >::NodeIterator::operator* ( ) const `[inline]`

Obtain the abstract node this iterator pointing.

Precondition

> *node_idx_* < num_nodes()

Returns

> Node with this graph's pointer and index of this node

Complexity: O(1).

4.28.4.2 template<typename V, typename E> **NodeIterator&** **Graph**< V, E >::NodeIterator::operator++ ( ) `[inline]`

Increment NodeIterator and return the next position.

Postcondition

> the increase by 1, may point to an invalid position.

Returns

> the modified NodeIterator.

Complexity: O(1).

4.28.4.3 template<typename V, typename E> bool **Graph**< V, E >::NodeIterator::operator== (const **NodeIterator** & *target*) const `[inline]`

Test the equality of NodeIterator:

Returns

true if the two NodeIterators belong to the same graph, same node and pointing to the same index.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp

4.29 NullConstraint Struct Reference

Public Member Functions

- void operator() (MeshType &m, double t)

4.29.1 Detailed Description

Null Constraint. No real constraint in this functor

4.29.2 Member Function Documentation

4.29.2.1 void NullConstraint::operator() (**MeshType** & *m*, double *t*) [inline]

Null Constraint Setter

Parameters

in	*g*	Valid mesh.
in	*t*	Valid time.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.30 Mesh< N, E, T >::Tetrahedral Class Reference

Inherits totally_ordered< Edge >.

Public Member Functions

- Node node (size_type i) const
- Edge edge (size_type i, size_type j) const
- Edge edge (size_type i) const
- size_type index () const
- tet_value_type & value ()
- const tet_value_type & value () const
- vector< Tetrahedral > tetAdjTetrahedral () const
- double volume () const
- bool isSurface () const
- bool operator== (const Tetrahedral &t) const
- bool operator< (const Tetrahedral &t) const

Friends

- class **Mesh**

4.30.1 Member Function Documentation

4.30.1.1 template<typename N , typename E , typename T > Edge Mesh< N, E, T >::Tetrahedral::edge (size_type *i*, size_type *j*) const [inline]

Return one of the six tetrahedral's edges with uid with *i*.

Precondition

0 <= *i* < 3, 0 <= *j* < 3

Returns

Edge (node(i), node(j)) Complexity: O(1).

4.30.1.2 template<typename N , typename E , typename T > Edge Mesh< N, E, T >::Tetrahedral::edge (size_type *i*) const [inline]

Return one of the four tetrahedral's nodes with uid with *i*.

Precondition

0 <= *i* < 3

Postcondition

edge(0) is (node0, node1), edge(1) is (node0, node2), edge(2) is (node0, node3), edge(3) is (node1, node2), edge(4) is (node1, node3) edge(5) is (node2, node3) where node0.index() < node1.index() < node2.index() < node3.index()

Returns

Edge such that *i* is the ordering of listed in

Postcondition

Complexity: O(1).

4.30.1.3 template<typename N , typename E , typename T > size_type Mesh< N, E, T >::Tetrahedral::index ( ) const [inline]

Return this tetrahedral's uid, a number in the range [0, g_tet_.num_nodes()).

Returns

The tetrahedral's uid i, s.t. 0 <= i < g_tet_.num_nodes() Complexity O(1)

4.30.1.4 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Tetrahedral::isSurface ( ) const [inline]

Return true if tetrahedral is on the surface

Precondition

Valid Tetrahedral.

Returns

bool if this tetrahedral is on the surface Complexity: O(1)

4.30.1.5 template<typename N , typename E , typename T > Node Mesh< N, E, T >::Tetrahedral::node (size_type *i*) const [inline]

Return one of the four tetrahedral's nodes with uid with *i*.

Precondition

0 <= *i* < 4

Returns

Node with the smallest index among the four nodes of this tetrahedral Complexity: O(1).

4.30.1.6 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Tetrahedral::operator< (const Tetrahedral & *t*) const [inline]

Test whether this Tetrahedral is less than *x* in the global order. This ordering function is useful for STL containers such as std::map<>. It need not have any geometric meaning.

4.30.1.7 template<typename N , typename E , typename T > bool Mesh< N, E, T >::Tetrahedral::operator== (const Tetrahedral & *t*) const [inline]

Test whether this Tetrahedral and *x* are equal.

Parameters

in	*x*	Tetrahedral in a mesh

Returns

Equal edges are from the same mesh and have the same tetrahedral uids. Complexity: O(1).

4.30.1.8 template<typename N , typename E , typename T > vector<Tetrahedral> Mesh< N, E, T >::Tetrahedral::tetAdjTetrahedral ( ) const [inline]

Return a vector of Tetrahedrals adjacent to the Tetrahedral

Precondition

Valid Tetrahedral.

Postcondition

return 0 <= result_vector.size() <= 4

Returns

vector containing Tetrahedral

Complexity: O(g_tet_.node(tet_uid_).degree())

4.30.1.9 template<typename N , typename E , typename T > tet_value_type& Mesh< N, E, T >::Tetrahedral::value ( ) [inline]

Get this tetrahedral's value (modifiable).

Returns

This tetrahedral's node_value_type value as a reference. Complexity O(1)

4.30.1.10 template<typename N , typename E , typename T > const tet_value_type& Mesh< N, E, T >::Tetrahedral::value ( ) const [inline]

Get this tetrahedral's value (Non-modifiable).

Returns

This tetrahedral's node_value_type value as a reference. Complexity O(1)

4.30.1.11 template<typename N , typename E , typename T > double Mesh< N, E, T >::Tetrahedral::volume ( ) const [inline]

Return the volume of this tetrahedral

Precondition

Valid Tetrahedral.

Returns

Double volume of this tetrahedral. The volume will be positive if its sign is the same as the original volume. The volume will be negative if its sign is different as the original volume. Complexity: O(1)

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.31 TetrahedralData Struct Reference

Public Attributes

- double **initialVolume**

4.31.1 Detailed Description

Custom structure of data to store with Tetrahedral

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.32 Mesh< N, E, T >::TetrahedralIterator Class Reference

Iterator class for Tetrahedrals. A forward iterator.

Inherits equality_comparable< TetrahedralIterator >.

Public Types

- typedef Tetrahedral value_type
- typedef Tetrahedral ∗ pointer
- typedef Tetrahedral & reference
- typedef std::input_iterator_tag iterator_category
- typedef std::ptrdiff_t difference_type

Public Member Functions

- TetrahedralIterator ()
- Tetrahedral operator∗ () const
- TetrahedralIterator & operator++ ()
- bool operator== (const TetrahedralIterator &target) const

Friends

- class **Mesh**

4.32.1 Detailed Description

template<typename N, typename E, typename T>class Mesh< N, E, T >::TetrahedralIterator

Iterator class for Tetrahedrals. A forward iterator.

4.32.2 Member Typedef Documentation

4.32.2.1 template<typename N , typename E , typename T > typedef std::ptrdiff_t Mesh< N, E, T >::TetrahedralIterator::difference_type

Difference between iterators

4.32.2.2 template<typename N , typename E , typename T > typedef std::input_iterator_tag Mesh< N, E, T >::TetrahedralIterator::iterator_category

Iterator category.

4.32.2.3 template<typename N , typename E , typename T > typedef **Tetrahedral*** **Mesh**< N, E, T >::**TetrahedralIterator::pointer**

Type of pointers to elements.

4.32.2.4 template<typename N , typename E , typename T > typedef **Tetrahedral&** **Mesh**< N, E, T >::**TetrahedralIterator::reference**

Type of references to elements.

4.32.2.5 template<typename N , typename E , typename T > typedef **Tetrahedral** **Mesh**< N, E, T >::**TetrahedralIterator::value_type**

Element type.

4.32.3 Constructor & Destructor Documentation

4.32.3.1 template<typename N , typename E , typename T > **Mesh**< N, E, T >::TetrahedralIterator::TetrahedralIterator ( ) `[inline]`

Construct an invalid TetrahedralIterator.

4.32.4 Member Function Documentation

4.32.4.1 template<typename N , typename E , typename T > **Tetrahedral** **Mesh**< N, E, T >::TetrahedralIterator::operator* ( ) const `[inline]`

Obtain the abstract tetrahedral this iterator pointing.

Precondition

*tit_* < num_nodes()

Returns

Node with this mesh's pointer and index of this node

Complexity: g_tet_type::NodeIterator operator*(), probably O(1).

4.32.4.2 template<typename N , typename E , typename T > **TetrahedralIterator&** **Mesh**< N, E, T >::TetrahedralIterator::operator++ ( ) `[inline]`

Increment TetrahedralIterator and return the next position.

Postcondition

the *tet_* increase by 1, may point to an invalid position.

Returns

the modified TetrahedralIterator.

Complexity: g_tet_type::NodeIterator operator++(), probably O(1).

4.32.4.3 template<typename N , typename E , typename T > bool Mesh< N, E, T >::TetrahedralIterator::operator== (const TetrahedralIterator & *target*) const [inline]

Test the equality of TetrahedralIterator:

Returns

true if the two TetrahedralIterator belong to the same mesh, and have the same tit_.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp

4.33 twoColor Struct Reference

Public Member Functions

- CS207::Color **operator()** (const MeshType::Node &n) const

4.33.1 Detailed Description

Creates a functor to color the ball with 2 colors.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.34 VolumePenaltyForce Struct Reference

Public Member Functions

- VolumePenaltyForce (MeshType ∗m, double K)
- Point operator() (Node n, double t)

Public Attributes

- MeshType ∗ **m_**
- double **K_**

4.34.1 Detailed Description

Volume Penalty Force Functor that returns the Volume Penalty Force

4.34.2 Constructor & Destructor Documentation

4.34.2.1 VolumePenaltyForce::VolumePenaltyForce (MeshType ∗ *m*, double *K*) [inline]

VolumePenaltyForce Constructor.

Parameters

`in`	*m*	mesh pointer.
`in`	*K*	mass spring constant.

4.34.3 Member Function Documentation

4.34.3.1 Point VolumePenaltyForce::operator() (Node *n*, double *t*) `[inline]`

Calculates VolmePenalty Force

Parameters

`in`	*n*	Valid node.
`in`	*t*	Valid time.

Returns

Point object that represents the volume penalty force.

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

4.35 WindForce Struct Reference

Public Member Functions

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

Public Attributes

- Point **w**

4.35.1 Detailed Description

Wind Force Code written by Tian Lan and Xide Xia

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

- /Users/tianlan/Downloads/tetrahedral_mesh - doc/simulation.cpp

Chapter 5

File Documentation

5.1 /Users/tianlan/Downloads/tetrahedral_mesh - doc/Graph.hpp File Reference

An undirected graph type.

```
#include <vector>
#include <cassert>
#include <iostream>
#include "CS207/Util.hpp"
#include "CS207/Point.hpp"
```

Classes

- class Graph< V, E >

  A template for 3D undirected graphs.
- class Graph< V, E >::Node

  Class representing the graph's nodes.
- class Graph< V, E >::Edge

  Class representing the graph's edges.
- class Graph< V, E >::NodeIterator

  Iterator class for nodes. A forward iterator.
- class Graph< V, E >::EdgeIterator

  Iterator class for edges. A forward iterator.
- class Graph< V, E >::IncidentIterator

  Iterator class for edges incident to a node. A forward iterator.

5.1.1 Detailed Description

An undirected graph type.

5.2 /Users/tianlan/Downloads/tetrahedral_mesh - doc/tet_mesh.hpp File Reference

A Mesh is composed of nodes, edges, and tetrahedrals such that: – All tetrahedrals have four nodes and six edges. – All edges belong to at least one tetrahedral.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include "Graph.hpp"
#include "CS207/Util.hpp"
#include "CS207/Point.hpp"
```

Classes

- class Mesh< N, E, T >

    A template for 3D tetrahedral meshes.
- class Mesh< N, E, T >::Node
- class Mesh< N, E, T >::Edge
- class Mesh< N, E, T >::Tetrahedral
- class Mesh< N, E, T >::NodeIterator

    Iterator class for nodes. A forward iterator.
- class Mesh< N, E, T >::EdgeIterator

    Iterator class for edges. A forward iterator.
- class Mesh< N, E, T >::TetrahedralIterator

    Iterator class for Tetrahedrals. A forward iterator.
- class Mesh< N, E, T >::IncidentIterator

Variables

- const unsigned **NUM_TET_ADJ_TET** = 4

5.2.1 Detailed Description

A Mesh is composed of nodes, edges, and tetrahedrals such that: – All tetrahedrals have four nodes and six edges. – All edges belong to at least one tetrahedral.