Cloth simulation

Generated by Doxygen 1.8.9.1

Thu Jan 28 2016 00:29:57

Contents

Chapter 1

Project Overview

Author

Elias Zischg
Daly Chea
Gerhard Aigner

1.1 Introduction

Here goes the introduction of the project

1.2 Installation

1.2.1 Minimal Requirements

- Linux based OS
- Memory min. 4 MB RAM
- CPU min. speed 1 GHz with 2 cores e.g. i5-2540 or better
- approx 1 MB free hard-disk space

1.2.2 Dependencies

- OpenGL and utility libraries (package libglu***-dev e.g. libglu-mesa-dev; depends on graphics hardware)
- OpenGL utilities toolkit (package freeglut3-dev)
- libPNG – PNG file handling (package libpng12-dev)
- Eigen – math utilities for vector and matrix arithmetics (package libeigen3-dev)
- C++ compiler with C++ 11 support e.g. GNU GCC g++ V4.7 or better

1.2.3 Compiling

Go to the sub-folder *build of the project folder and run* `make all`.
`If you want to compile directly on the console use the following options:`

- `-std=c++11 or -std=c++0x (use the C++ 11 dialect)`
- `-lGL (the OpenGL library)`
- `-lGLU (OpenGL utility library)`
- `-lglut (the GL utility Toolkit - make sure 'freeglut' is installed, some extensions not present in the GLUT API are used)`
- `-lpthread (or -lthread; manage threads)`
- `-lpng (library to handle PNG files)`

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 std Namespace Reference

Classes

- class MipMap
- class Scene
- class Simulation

Functions

- bool vertexInTriangle (const Eigen::Vector3d &P, const Eigen::Vector3d &A, const Eigen::Vector3d &B, const Eigen::Vector3d &C, const float epsilon, float &dist, Eigen::Vector3d &N)
- bool baryVertexInTriangle (const Eigen::Vector3d &P, const Eigen::Vector3d &A, const Eigen::Vector3d &B, const Eigen::Vector3d &C)
- void collisionDetectionAndResponse (vector< Mass > &points, size_t offsP, size_t lenP, GLfloat object_↩ mesh[ ], size_t offsO, size_t lenO)
- void fwd_euler (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- Eigen::Vector3d f_int (Mass &pt, vector< Spring > &springs)
- Eigen::Vector3d gravity ()
- void symplectic (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void leapfrog (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void midpoint (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void force (vector< Mass > &points, vector< Spring > &springs, bool interaction)

Variables

- const size_t model3dVertices = MODEL(Vertices)

  number of vertices
- GLfloat * model3dPositions = MODEL(Positions)

  all vertex positions
- GLfloat * model3dTexels = MODEL(Texels)

  all texture coordinates
- GLfloat * model3dNormals = MODEL(Normals)

  all normals of the face surfaces
- const size_t model3dObjectsWithMass = MODEL(ObjectsWithMass)

  number of objects to apply to the mass-spring simulation
- const size_t * model3dMasses = MODEL(Masses)

array of 3D-objects with mass

- const size_t * model3dMassFwdOffs = MODEL(MassFwdOffs)

  offset in the points array
- const size_t * model3dMassVertices = MODEL(MassVertices)

  number of vertices per object with mass
- const size_t * model3dMassRevOffs = MODEL(MassRevOffs)

  offsets for the model3dRevIndex array
- const size_t * model3dMassRevOffsOrig = MODEL(MassRevOffsOrig)

  offset to the first vertex of an object with mass
- const size_t ** model3dFwdIndex = MODEL(FwdIndex)

  indices of the mass-points in the positions array
- const size_t * model3dFwdIndexLength = MODEL(FwdIndexLength)

  number of indices for each mass-point
- const size_t * model3dRevIndex = MODEL(RevIndex)

  index of a vertex in the mass-points array
- const size_t model3dObjects = MODEL(Objects)

  number of 3D-objects
- const size_t * model3dObjectOffset = MODEL(ObjectOffset)

  offset to the first vertex of a 3D-object
- const size_t * model3dObjectLength = MODEL(ObjectLength)

  number of vertices for each 3D-object
- const char ** model3dObjectNames = (const char**) MODEL(ObjectNames)

  names of the objects (for identification)
- const char * model3dTextureFilePath = "textures/textures_all.rgb"

  path to the texture-image file
- bool leapFrogInitialized = false
- double t = 0
- double floorLevel = -1.
- double repulsiveSpringConst = -50.

5.1.1 Function Documentation

5.1.1.1 bool std::baryVertexInTriangle (const Eigen::Vector3d & *P*, const Eigen::Vector3d & *A*, const Eigen::Vector3d & *B*, const Eigen::Vector3d & *C*)

5.1.1.2 void std::collisionDetectionAndResponse (vector< Mass > & *points*, size_t *offsP*, size_t *lenP*, GLfloat *object_mesh[]*, size_t *offsO*, size_t *lenO*)

Checks if there is a collision with mass spring points and object mesh. Adds penalty forces for collisions.

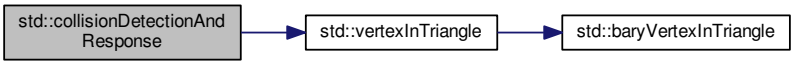
This algorithm can't deal with self collision, so assure the object_mesh doesn't contain vertices owned by points

Parameters

points	array of mass points
offsP	offset in the points array
lenP	number of mass points
object_mesh	vertices array with all triangles (except them present in points)
offsO	offset in terms of vertices

lenO	number of vertices to compute

Here is the call graph for this function:



5.1.1.3 Eigen::Vector3d std::f_int (Mass & *pt*, vector< Spring > & *springs*)

Compute internal forces of a point.

This is a subroutine for the forward Euler method.

Parameters

pt	the actual point
springs	vector of springs

Returns

sum of forces of all springs connected to point pt

5.1.1.4 void std::force (vector< Mass > & *points*, vector< Spring > & *springs*, bool *interaction*)

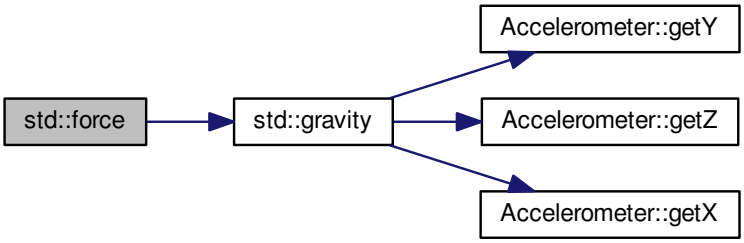
Calculate force for all points at time t according to x(t) stored in the points.

This is a subroutine for the midpoint method.

Parameters

points	all mass-points
springs	all springs
interaction	true = apply external forces other than gravity

Here is the call graph for this function:



5.1.1.5 void std::fwd_euler (double *dt*, vector< Mass > & *points*, vector< Spring > & *springs*, bool *interaction*)

Euler Method - Implementation.

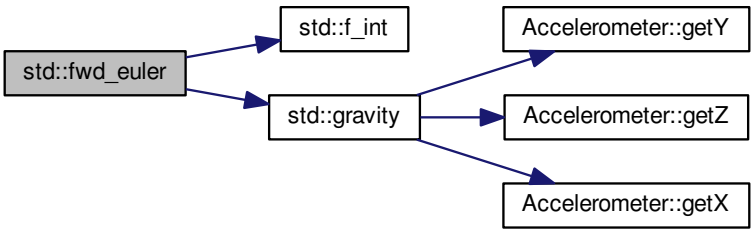
This function calculates the position and velocity for the next time step.

```
# Compute position at t+h      x(t+h) = x(t) + h*v(t)
# Compute forces at t          f(t)   = f_int(t) + f_ext(t)
# Compute acceleration at t    a(t)   = 1/m (f(t) - gamma*v(t))
# Compute velocity at t+h      v(t+h) = v(t) + h*a(t)
```

Parameters

dt	time step
points	all mass-points
springs	all springs
interaction	true = apply external forces other than gravity

Here is the call graph for this function:



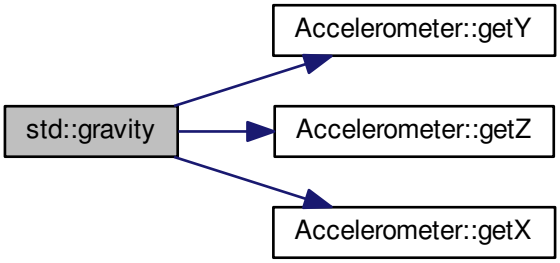
5.1.1.6 Eigen::Vector3d std::gravity ( )

The gravity. Either it is determined from an acceleration sensor or if not detected a constant in Z axis.

Returns

the gravity

Here is the call graph for this function:



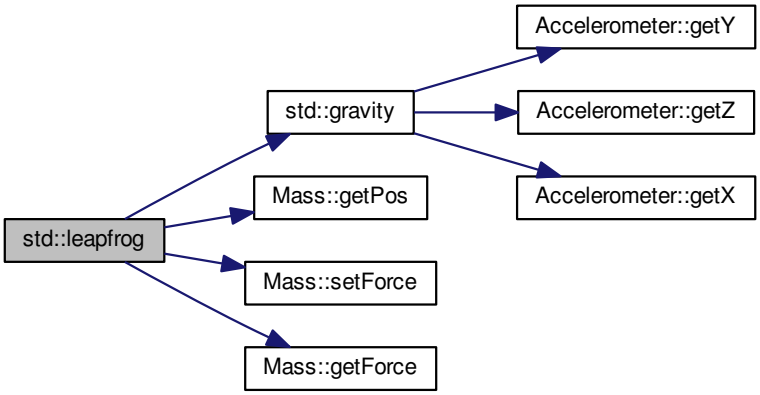
5.1.1.7 void std::leapfrog (double *dt*, vector< Mass > & *points*, vector< Spring > & *springs*, bool *interaction*)

Leapfrog method.

Parameters

dt	time step
points	all mass-points
springs	all springs
interaction	true = apply external forces other than gravity

Here is the call graph for this function:



5.1.1.8 void std::midpoint (double *dt*, vector< Mass > & *points*, vector< Spring > & *springs*, bool *interaction*)

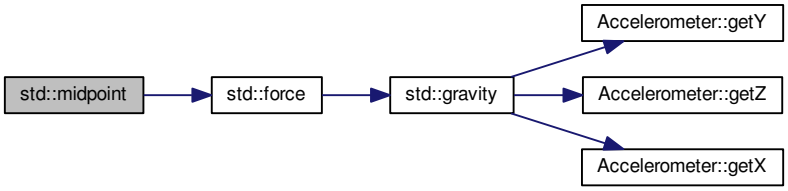
Midpoint method.

```
Formula:
  a(t) = (f(t) - gamma*v(t))/m
  v_half(t+h/2) = v(t) + h/2*a(t)
  x_half(t+h/2) = x(t) + h/2*v(t)
  a_half(t+h/2) = (f_half(t+h/2) - gamma*v_half(t+h/2))/m
  x(t+h) = x(t) + h*v_half(t+h/2)
  v(t+h) = v(t) + h*a_half(t+h/2)
```

Parameters

dt	time step
points	all mass-points
springs	all springs
interaction	true = apply external forces other than gravity

Here is the call graph for this function:



5.1.1.9 void std::symplectic (double *dt*, vector< Mass > & *points*, vector< Spring > & *springs*, bool *interaction*)

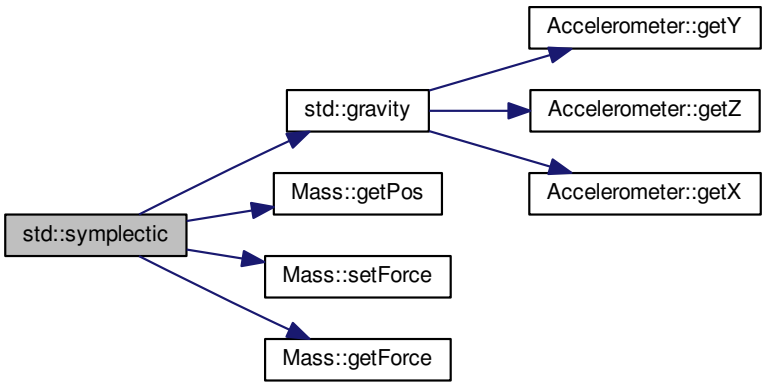
Symplectic euler method.

```
Algorithm:
# Compute position at t+h      x(t+h) = x(t) + h*v(t)
# Compute forces at t          f(t)   = f_int(t) + f_ext(t)
# Compute acceleration at t    a(t)   = 1/m (f(t) - gamma*v(t))
# Compute velocity at t+h      v(t+h) = v(t) + h*a(t)
```

Parameters

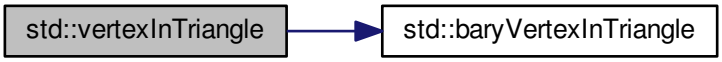
dt	time step
points	all mass-points
springs	all springs
interaction	true = apply external forces other than gravity

Here is the call graph for this function:



5.1.1.10 bool std::vertexInTriangle (const Eigen::Vector3d & *P,* const Eigen::Vector3d & *A,* const Eigen::Vector3d & *B,* const Eigen::Vector3d & *C,* const float *epsilon,* float & *dist,* Eigen::Vector3d & *N*)

Here is the call graph for this function:



5.1.2 Variable Documentation

5.1.2.1 double std::floorLevel = -1.

5.1.2.2 bool std::leapFrogInitialized = false

5.1.2.3 const size_t ∗∗ std::model3dFwdIndex = MODEL(FwdIndex)

indices of the mass-points in the positions array

5.1.2.4 const size_t ∗ std::model3dFwdIndexLength = MODEL(FwdIndexLength)

number of indices for each mass-point

5.1.2.5 const size_t ∗ std::model3dMasses = MODEL(Masses)

array of 3D-objects with mass

5.1.2.6 const size_t ∗ std::model3dMassFwdOffs = MODEL(MassFwdOffs)

offset in the points array

5.1.2.7 const size_t ∗ std::model3dMassRevOffs = MODEL(MassRevOffs)

offsets for the model3dRevIndex array

5.1.2.8 const size_t ∗ std::model3dMassRevOffsOrig = MODEL(MassRevOffsOrig)

offset to the first vertex of an object with mass

5.1.2.9 const size_t ∗ std::model3dMassVertices = MODEL(MassVertices)

number of vertices per object with mass

5.1.2.10 GLfloat ∗ std::model3dNormals = MODEL(Normals)

all normals of the face surfaces

5.1.2.11 const size_t * std::model3dObjectLength = MODEL(ObjectLength)

number of vertices for each 3D-object

5.1.2.12 const char ** std::model3dObjectNames = (const char**) MODEL(ObjectNames)

names of the objects (for identification)

5.1.2.13 const size_t * std::model3dObjectOffset = MODEL(ObjectOffset)

offset to the first vertex of a 3D-object

5.1.2.14 const size_t std::model3dObjects = MODEL(Objects)

number of 3D-objects

5.1.2.15 const size_t std::model3dObjectsWithMass = MODEL(ObjectsWithMass)

number of objects to apply to the mass-spring simulation

5.1.2.16 GLfloat * std::model3dPositions = MODEL(Positions)

all vertex positions

5.1.2.17 const size_t * std::model3dRevIndex = MODEL(RevIndex)

index of a vertex in the mass-points array

5.1.2.18 GLfloat * std::model3dTexels = MODEL(Texels)

all texture coordinates

5.1.2.19 const char * std::model3dTextureFilePath = "textures/textures_all.rgb"

path to the texture-image file

5.1.2.20 const size_t std::model3dVertices = MODEL(Vertices)

number of vertices

5.1.2.21 double std::repulsiveSpringConst = -50.

5.1.2.22 double std::t = 0

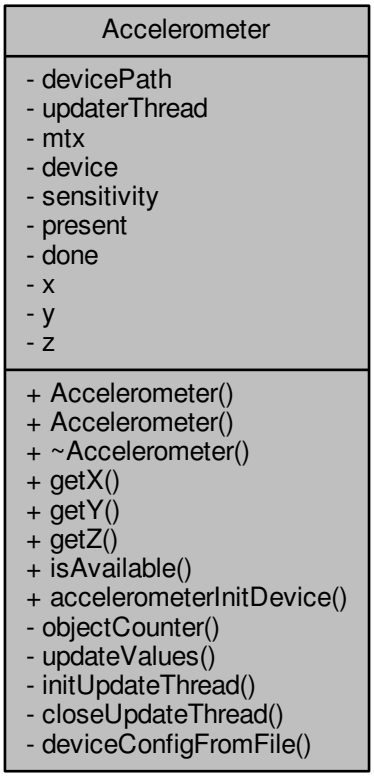
Chapter 6

Class Documentation

6.1 Accelerometer Class Reference

`#include <Accelerometer.h>`

Collaboration diagram for Accelerometer:



Public Member Functions

- Accelerometer ()
- Accelerometer (int argc, char ∗argv[ ])
- ∼Accelerometer ()
- double getX ()
- double getY ()
- double getZ ()

Static Public Member Functions

- static bool isAvailable ()
- static void accelerometerInitDevice (int argc, char ∗argv[ ])

Static Private Member Functions

- static int objectCounter (int count)
- static void updateValues ()
- static void initUpdateThread ()
- static void closeUpdateThread ()
- static void deviceConfigFromFile (std::string configPath)

Static Private Attributes

- static std::string devicePath
- static std::thread updaterThread
- static std::mutex mtx
- static std::ifstream device
- static double sensitivity = 1.0
- static bool present = false
- static bool done = false
- static double x = 0.0
- static double y = 0.0
- static double z = 0.0

6.1.1 Detailed Description

This class handles the connection to the accelerometer device of a notebook.

Actually it supports the accelerometer of type lis3lv02d. Other devices are unsupported, but they may work with proper settings in the configuration file.

Configuration file:
The file follows the specifications of a windows INI file. Here is an example setup:

```
1 [accelerometer] ; this section must be present
2 device = "/sys/devices/platform/lis3lv02d/position" ; must be an absolute path to the unix device
3 sensitivity = 0.001 ; multiplier to scale the accelerometer output to fit it to gravity force
```

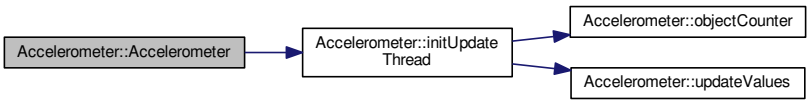
6.1.2 Constructor & Destructor Documentation

6.1.2.1 Accelerometer::Accelerometer ( )

Construct an accelerometer device handle.

This constructor fails if no device has been initialized previously by using the constructor which takes command line arguments.

Here is the call graph for this function:



6.1.2.2 Accelerometer::Accelerometer (int *argc*, char * *argv[]*)

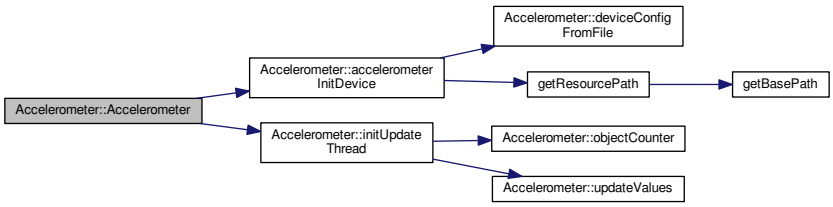
Construct an accelerometer device handle.

This constructor initializes a helping thread which determines periodically data from the device, if it is called the first time.

Parameters

argc	command line arguments counter from `main()`
argv	arguments vector

Here is the call graph for this function:

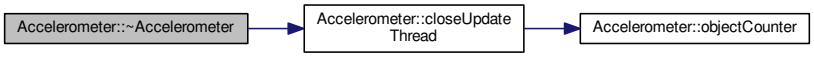


6.1.2.3 Accelerometer::~Accelerometer ( )

Destroy the device handle.

If the last handle is destroyed, also the connection to the device is closed and the appertaining thread is destroyed.

Here is the call graph for this function:



6.1.3 Member Function Documentation

6.1.3.1 void Accelerometer::accelerometerInitDevice (int *argc*, char * *argv[]*) [static]

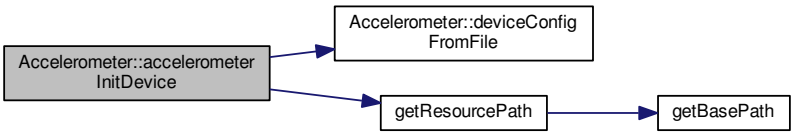
Initialize the device.

This method is called by the constructor of the Accelerometer class.

Parameters

argc	arguments counter
argv	arguments vector

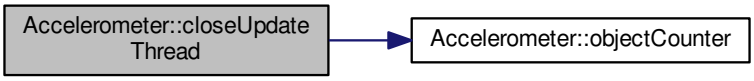
Here is the call graph for this function:



6.1.3.2 void Accelerometer::closeUpdateThread ( ) [static], [private]

Close the updater thread if the last instance of Accelerometer is destroyed. This method is called by the destructor of the Accelerometer class.

Here is the call graph for this function:



6.1.3.3 void Accelerometer::deviceConfigFromFile (std::string *configPath*) [static], [private]

Read configuration for accelerometer device from a config file.

Parameters

configPath	path to the config file

6.1.3.4 double Accelerometer::getX ( )

Get the force into X direction.

Returns

the force determined from the device.

6.1.3.5 double Accelerometer::getY ( )

Get the force into Y direction.

Returns

the force determined from the device.

6.1.3.6 double Accelerometer::getZ ( )

Get the force into Z direction.

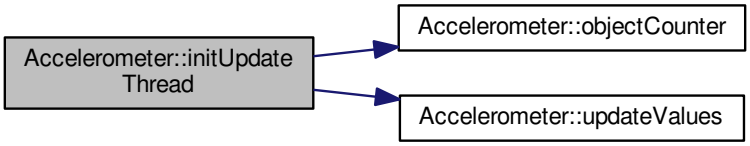
Returns

the force determined from the device.

6.1.3.7 void Accelerometer::initUpdateThread ( ) [static], [private]

Initialize the updater thread, which reads periodically the state of the accelerometer. This method is called each times an Accelerometer object is created.

Here is the call graph for this function:



6.1.3.8 bool Accelerometer::isAvailable ( ) [static]

Checks if the device is present.

Return values

true	if the device is online
false	otherwise

6.1.3.9 int Accelerometer::objectCounter (int *count*) [static],[private]

Count the number of open accelerometers

Parameters

count	1 count up, -1 count down or 0 to retrieve without change

Returns

the counter after change

6.1.3.10 void Accelerometer::updateValues ( ) [static],[private]

Updater function delivered to the device-handling thread.

6.1.4 Member Data Documentation

6.1.4.1 std::ifstream Accelerometer::device [static],[private]

6.1.4.2 std::string Accelerometer::devicePath [static],[private]

6.1.4.3 bool Accelerometer::done = false [static],[private]

6.1.4.4 std::mutex Accelerometer::mtx [static],[private]

6.1.4.5 bool Accelerometer::present = false [static],[private]

6.1.4.6 double Accelerometer::sensitivity = 1.0 [static],[private]

6.1.4.7 std::thread Accelerometer::updaterThread [static],[private]

6.1.4.8 double Accelerometer::x = 0.0 [static],[private]

6.1.4.9 double Accelerometer::y = 0.0 [static],[private]

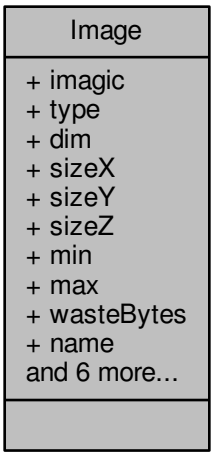
6.1.4.10 double Accelerometer::z = 0.0 [static],[private]

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

- src/Accelerometer.h
- src/Accelerometer.cpp

6.2 Image Struct Reference

Collaboration diagram for Image:



Public Attributes

- unsigned short imagic
- unsigned short type
- unsigned short dim
- unsigned short sizeX
- unsigned short sizeY
- unsigned short sizeZ
- unsigned int min
- unsigned int max
- unsigned int wasteBytes
- char name [80]
- unsigned int colorMap
- FILE ∗ file
- unsigned char ∗ tmp [5]
- unsigned int rleEnd
- unsigned int ∗ rowStart
- unsigned int ∗ rowSize

6.2.1 Member Data Documentation

6.2.1.1 unsigned int Image::colorMap

6.2.1.2 unsigned short Image::dim

6.2.1.3 FILE∗ Image::file

6.2.1.4 unsigned short Image::imagic

6.2.1.5 unsigned int Image::max

6.2.1.6 unsigned int Image::min

6.2.1.7 char Image::name[80]

6.2.1.8 unsigned int Image::rleEnd

6.2.1.9 unsigned int* Image::rowSize

6.2.1.10 unsigned int* Image::rowStart

6.2.1.11 unsigned short Image::sizeX

6.2.1.12 unsigned short Image::sizeY

6.2.1.13 unsigned short Image::sizeZ

6.2.1.14 unsigned char* Image::tmp[5]

6.2.1.15 unsigned short Image::type

6.2.1.16 unsigned int Image::wasteBytes

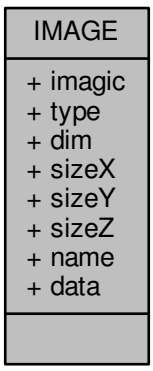
The documentation for this struct was generated from the following file:

- src/SGIimage.c

6.3 IMAGE Struct Reference

`#include <SGIimage.h>`

Collaboration diagram for IMAGE:



Public Attributes

- unsigned short imagic
- unsigned short type
- unsigned short dim
- unsigned short sizeX
- unsigned short sizeY
- unsigned short sizeZ
- char name [128]
- unsigned char * data

6.3.1 Member Data Documentation

6.3.1.1 unsigned char* IMAGE::data

6.3.1.2 unsigned short IMAGE::dim

6.3.1.3 unsigned short IMAGE::imagic

6.3.1.4 char IMAGE::name[128]

6.3.1.5 unsigned short IMAGE::sizeX

6.3.1.6 unsigned short IMAGE::sizeY

6.3.1.7 unsigned short IMAGE::sizeZ

6.3.1.8 unsigned short IMAGE::type

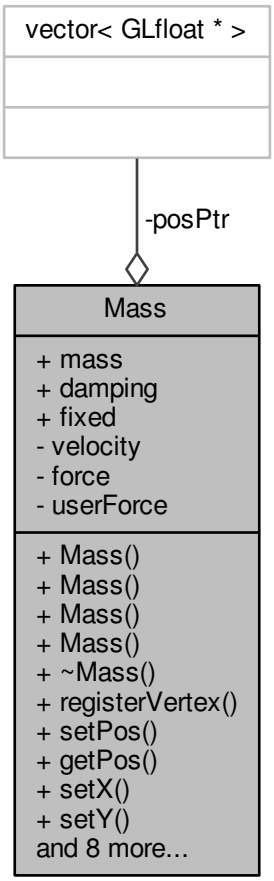
The documentation for this struct was generated from the following file:

- src/SGIimage.h

6.4 Mass Class Reference

```
#include <Mass.h>
```

Collaboration diagram for Mass:



Public Member Functions

- Mass (std::vector< GLfloat * > posPtr=vector< GLfloat * >(), double mass=0.0, double damp=0.0)
- Mass (double mass, double damp=0.0)
- Mass (const Mass &m)
- Mass (Mass &&)
- virtual ~Mass ()
- void registerVertex (GLfloat *ptr)
- void setPos (Eigen::Vector3d p)
- Eigen::Vector3d getPos ()
- void setX (double x)
- void setY (double y)
- void setZ (double z)
- void setVel (Eigen::Vector3d v)
- Eigen::Vector3d getVel ()
- void setForce (Eigen::Vector3d f)
- Eigen::Vector3d getForce ()
- void addForce (Eigen::Vector3d f)
- void setUserForce (Eigen::Vector3d f)
- Eigen::Vector3d getUserForce ()

Public Attributes

- double mass
- double damping
- bool fixed

Private Attributes

- vector< GLfloat * > posPtr
- Eigen::Vector3d velocity
- Eigen::Vector3d force
- Eigen::Vector3d userForce

6.4.1 Detailed Description

This class contains all properties of a mass-point which are necessary to run a physically based simulation of a mass-spring system.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Mass::Mass (std::vector< GLfloat * > *posPtr* = `vector<GLfloat*>()`, double *mass* = `0.0`, double *damp* = `0.0`)

Default constructor.

Parameters

posPtr	vector of pointers to the vertices in the positions array
mass	the mass of the mass-point
damp	the damping factor of the mass-point (to simulate friction and air resistance)

6.4.2.2 Mass::Mass (double *mass*, double *damp* = `0.0`)

Constructor which takes at least the mass argument.

Parameters

mass	the mass of the mass-point
damp	the damping factor of the mass-point (to simulate friction and air resistance)

6.4.2.3 Mass::Mass (const Mass & *m*)

Copy constructor

Parameters

m	source object

6.4.2.4 Mass::Mass (Mass && *m*)

Move Constructor

Parameters

m	

6.4.2.5 Mass::~Mass ( ) [virtual]

Default destructor.

6.4.3 Member Function Documentation

6.4.3.1 void Mass::addForce (Eigen::Vector3d *f*)

Add a force to the stored value.

Parameters

f	force vector

6.4.3.2 Eigen::Vector3d Mass::getForce ( )

Get the force.

Returns

force vector

6.4.3.3 Eigen::Vector3d Mass::getPos ( )

Get position vector of mass.

Don't write to the elements of the returned vector. This will lead to inconsistencies in the vertices array. To set a single value use setX(), setY() and setZ() instead.

Returns

the position vector or if no vertices are registered the 0 vector.

6.4.3.4 Eigen::Vector3d Mass::getUserForce ( )

Get external forces (usually without gravity).

Returns

force vector

6.4.3.5 Eigen::Vector3d Mass::getVel ( )

Get the velocity.

Returns

velocity vector

6.4.3.6 void Mass::registerVertex (GLfloat * *ptr*)

Register the pointer to a vertex as position.

Parameters

ptr	pointer targeting the x-coordinate of the vertex position. The y- and z-coordinates must be in consecutive order in the array.

6.4.3.7 void Mass::setForce (Eigen::Vector3d *f*)

Set the force.

This vector is reserved for the solver. Use `setUserForce()` to apply edxternal forces.

Parameters

f	force vector.

6.4.3.8 void Mass::setPos (Eigen::Vector3d *pos*)

Set position of the mass object.

This setter spreads the position to all registered vertices.

Parameters

pos	the position vector

6.4.3.9 void Mass::setUserForce (Eigen::Vector3d *f*)

Set external forces.

Use this setter to apply all external forces except the gravity, the solver applies it separately.

Parameters

f	force vector

6.4.3.10 void Mass::setVel (Eigen::Vector3d *v*)

Set the velocity.

Parameters

v	velocity vector

6.4.3.11 void Mass::setX (double *x*)

Set x-coordinate of the mass object.

This setter spreads the x-coordinate to all registered vertices. Don't use getPos.x to write a coordinate, it will mess up the vertices array.

Parameters

x	the coordinate

6.4.3.12 void Mass::setY (double *y*)

Set y-coordinate of the mass object.

This setter spreads the y-coordinate to all registered vertices. Don't use getPos.y to write a coordinate, it will mess up the vertices array.

Parameters

y	the coordinate

6.4.3.13 void Mass::setZ (double *z*)

Set z-coordinate of the mass object.

This setter spreads the z-coordinate to all registered vertices. Don't use getPos.z to write a coordinate, it will mess up the vertices array.

Parameters

z	the coordinate

6.4.4 Member Data Documentation

6.4.4.1 double Mass::damping

6.4.4.2 bool Mass::fixed

6.4.4.3 Eigen::Vector3d Mass::force [private]

6.4.4.4 double Mass::mass

6.4.4.5 vector<GLfloat*> Mass::posPtr [private]

6.4.4.6 Eigen::Vector3d Mass::userForce [private]

6.4.4.7 Eigen::Vector3d Mass::velocity [private]

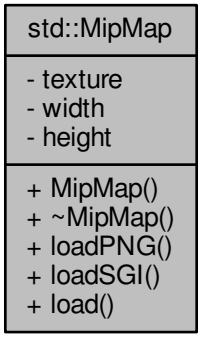
The documentation for this class was generated from the following files:

- src/Mass.h
- src/Mass.cpp

6.5 std::MipMap Class Reference

`#include <MipMap.h>`

Collaboration diagram for std::MipMap:



Public Member Functions

- MipMap ()
- virtual ~MipMap ()
- void loadPNG (const string fileName)

  load texture from PNG
- void loadSGI (const string fileName)

  load texture from PNG
- void load (const string fileName)

  load a texture with a selected import filter

Private Attributes

- GLuint texture = 0
- size_t width = 0
- size_t height = 0

6.5.1 Detailed Description

This class is a container for a texture mipmap.

It also handles the loading of an image file.
Supported file formats:
PNG images with three colors and no alpha channel SGI RGB images with uncompressed data and without alpha channel.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 std::MipMap::MipMap ( )

Default constructor.

6.5.2.2 std::MipMap::~MipMap ( ) [virtual]

Default destructor.

6.5.3 Member Function Documentation

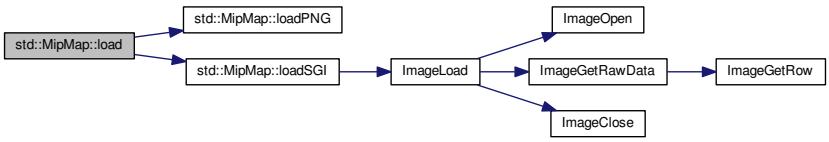
6.5.3.1 void std::MipMap::load (const string *fileName*)

load a texture with a selected import filter

Parameters

fileName	the image file to be loaded

Here is the call graph for this function:



6.5.3.2 void std::MipMap::loadPNG (const string *fileName*)

load texture from PNG

loads a png file into an opengl mipmap object, using cstdio , libpng, and opengl. This function is almost identical with the example code on https://en.wikibooks.org/wiki/OpenGL_Programming/↩ Intermediate/Textures

Parameters

fileName	the png file to be loaded

6.5.3.3 void std::MipMap::loadSGI (const string *fileName*)

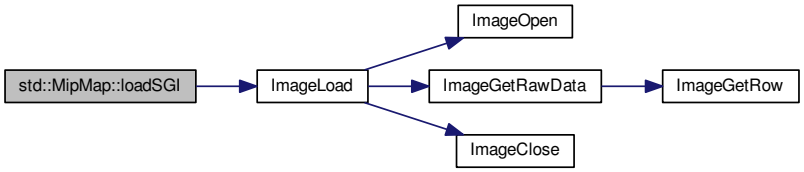
load texture from PNG

loads a RGB file into an opengl mipmap object, using cstdio , libpng, and opengl. This function is almost identical with the example code on https://en.wikibooks.org/wiki/OpenGL_Programming/↩ Intermediate/Textures

Parameters

fileName	the SGI-rgb file to be loaded

Here is the call graph for this function:



6.5.4 Member Data Documentation

6.5.4.1 size_t std::MipMap::height = 0 [private]

6.5.4.2 GLuint std::MipMap::texture = 0 [private]

6.5.4.3 size_t std::MipMap::width = 0 [private]

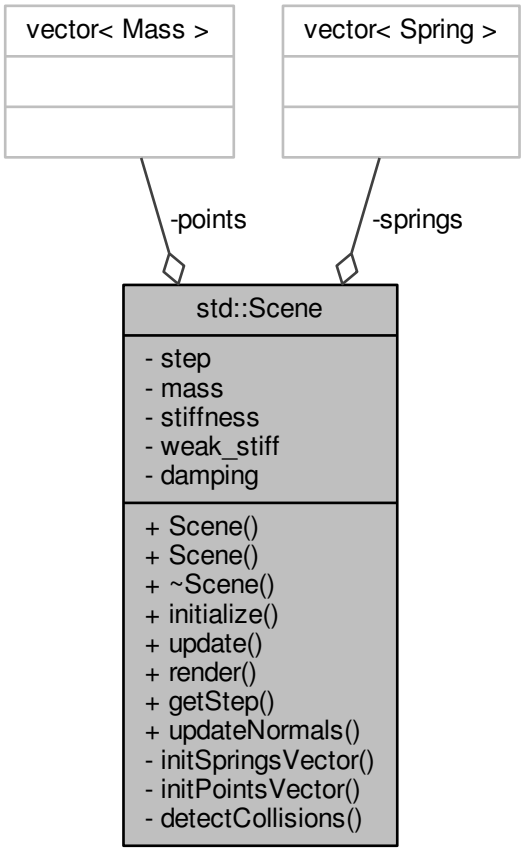
The documentation for this class was generated from the following files:

- src/MipMap.h
- src/MipMap.cpp

6.6 std::Scene Class Reference

```
#include <Scene.h>
```

Collaboration diagram for std::Scene:



Public Member Functions

- Scene ()
- Scene (int argc, char ∗argv[ ])
- virtual ∼Scene ()
- void initialize ()
- void update ()
- void render ()
- double getStep () const
- void updateNormals ()

Private Member Functions

- void initSpringsVector (size_t i)
- void initPointsVector (size_t i, size_t ∗n)
- void detectCollisions ()

Private Attributes

- double step
- double mass
- double stiffness
- double weak_stiff
- double damping
- vector< Mass > points
- vector< Spring > springs

6.6.1 Detailed Description

This is a class which handles all tasks to set up, simulate and render a scene with a physical model.

Usually one scene object instance exists program-wide.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 std::Scene::Scene ( )

Default constructor

6.6.2.2 std::Scene::Scene (int *argc*, char * *argv[]*)

Constructor which initializes the objectives with values given by command line parameters.

Parameters

argc	arguments counter supplied by the main function
argv	arguments vector of length `argc`

6.6.2.3 std::Scene::~Scene ( ) [virtual]

Default destructor.

6.6.3 Member Function Documentation

6.6.3.1 void std::Scene::detectCollisions ( ) [private]

Call the collision detection algorithm for each 3D-object with mass.

Here is the call graph for this function:

6.6.3.2 double std::Scene::getStep ( ) const

Get the time step for the simulation.

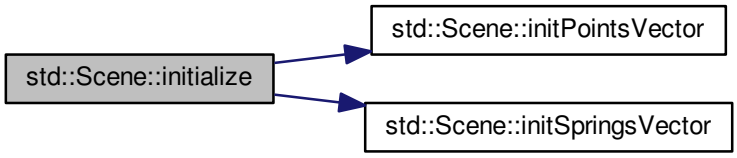
Returns

step in seconds

6.6.3.3 void std::Scene::initialize ( )

Initialize the scene for rendering and physical simulation.

Here is the call graph for this function:



6.6.3.4 void std::Scene::initPointsVector (size_t *i*, size_t * *n*) [private]

Initialize all mass-points.

This method connects all mass-points processed by the physics simulation to the vertices in the ...Positions vector for the renderer. The vector of points must be filled with the correct number of Mass objects initialized with default values.

Parameters

i	the i-th 3d-object
n	a pointer to a counter value, it is incremented with each initialized mass-point. it must be initialized with 0 before the first call of this method.

6.6.3.5 void std::Scene::initSpringsVector (size_t *i*) [private]

Initialize structural (strong) and bending (weak) springs. No diagonal springs, because of triangular grid

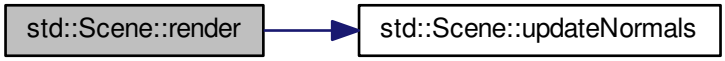
Parameters

i	the i-th 3D-object to process

6.6.3.6 void std::Scene::render ( )

Render the scene.

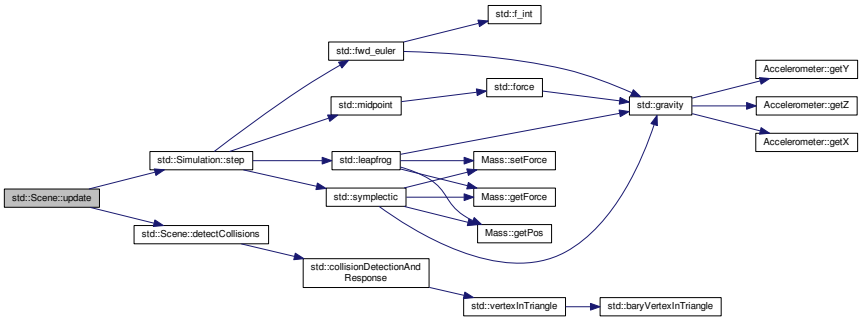
Here is the call graph for this function:



6.6.3.7 void std::Scene::update ( )

Update the simulation step and collision detection.

Here is the call graph for this function:



6.6.3.8 void std::Scene::updateNormals ( )

Recalculate all normals for all faces modified by the simulaton step.

6.6.4 Member Data Documentation

6.6.4.1 double std::Scene::damping [private]

6.6.4.2 double std::Scene::mass [private]

6.6.4.3 vector<Mass> std::Scene::points [private]

6.6.4.4 vector<Spring> std::Scene::springs [private]

6.6.4.5 double std::Scene::step [private]

6.6.4.6 double std::Scene::stiffness [private]

6.6.4.7 double std::Scene::weak_stiff [private]

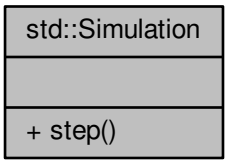
The documentation for this class was generated from the following files:

- src/Scene.h
- src/Scene.cpp

6.7 std::Simulation Class Reference

`#include <Simulation.h>`

Collaboration diagram for std::Simulation:



Public Types

- enum Method { EULER, SYMPLECTIC, LEAPFROG, MIDPOINT }

Static Public Member Functions

- static void step (double dt, Method method, vector< Mass > &points, vector< Spring > &springs)

6.7.1 Detailed Description

Simulation class. Solver for mass-spring systems.

This class provides some numerical solvers. They are programmed during a assignment of the pro-seminar physically based simulations preceding to this project.

6.7.2 Member Enumeration Documentation

6.7.2.1 enum std::Simulation::Method

Enumerator

EULER

SYMPLECTIC

LEAPFROG

MIDPOINT

6.7.3 Member Function Documentation

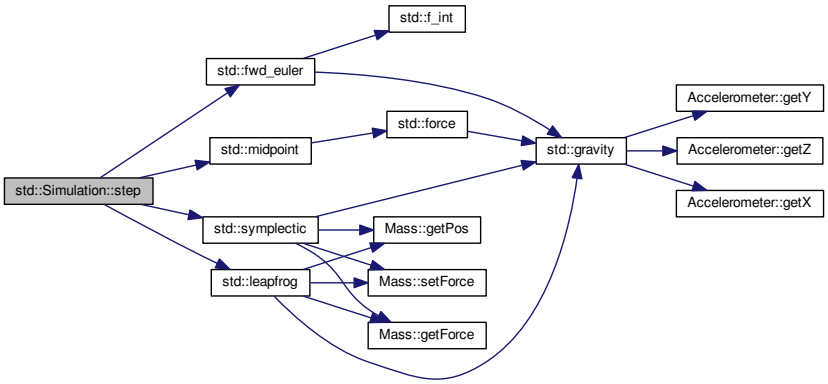
6.7.3.1 void std::Simulation::step (double *dt,* Method *method,* vector< Mass > & *points,* vector< Spring > & *springs*) [static]

Execute one step of the simulation.

Parameters

dt	time delay between two steps
method	select the method to use; The value can be EULER, SYMPLECTIC, LEAPFROG or MIDPOINT.
points	all mass-points
springs	all springs

Here is the call graph for this function:



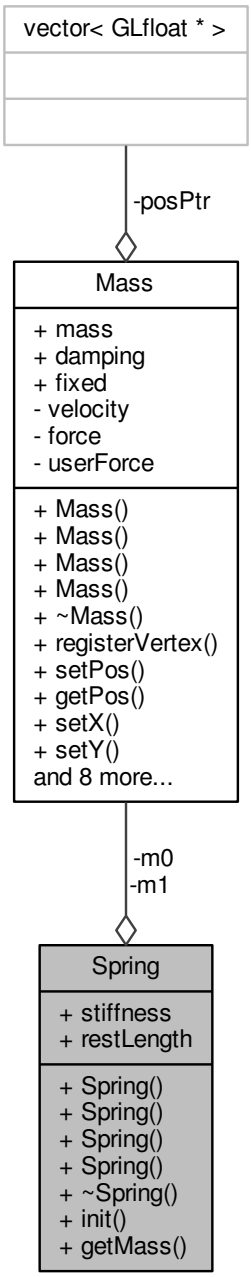
The documentation for this class was generated from the following files:

- src/Simulation.h
- src/Simulation.cpp

6.8 Spring Class Reference

```
#include <Spring.h>
```

Collaboration diagram for Spring:



Public Member Functions

- Spring (double stiff=0.0, double restLen=0.0)
- Spring (Mass *mass0, Mass *mass1, double stiff=0.0, double restLen=0.0)
- Spring (Spring &s)
- Spring (Spring &&s)
- virtual ~Spring (void)

- void init (Mass *mass0, Mass *mass1)
- Mass * getMass (int i)

Public Attributes

- double stiffness
- double restLength

Private Attributes

- Mass * m0
- Mass * m1

6.8.1 Detailed Description

This class contains all properties of a spring which are necessary to run a physically based simulation of a mass-spring system.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Spring::Spring (double *stiff* = 0.0, double *restLen* = 0.0)

Default constructor.

The end-points are left uninitialized, so after calling this constructor a call to `init()` is necessary.

Parameters

stiff	stiffness factor of the spring
restLen	the rest length = length of the spring without any forces taking effect to it

6.8.2.2 Spring::Spring (Mass * *mass0*, Mass * *mass1*, double *stiff* = 0.0, double *restLen* = 0.0)

Constructor which initializes the endpoints of the spring with two mass-points

Parameters

mass0	mass-point on the first end of the spring
mass1	mass-point on the second end
stiff	stiffness factor of the spring
restLen	the rest length = length of the spring without any forces taking effect to it

6.8.2.3 Spring::Spring (Spring & *s*)

Copy Constructor

Parameters

s	source spring

6.8.2.4 Spring::Spring (Spring && *s*)

Move constructor

Parameters

s	source spring

6.8.2.5 Spring::~Spring (void) [virtual]

Default destructor.

6.8.3 Member Function Documentation

6.8.3.1 Mass * Spring::getMass (int *i*)

Get a mass.

Parameters

i	0 or 1 to select the mass

Returns

the selected mass

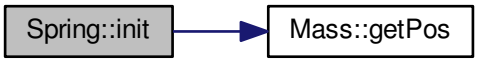
6.8.3.2 void Spring::init (Mass * *mass0*, Mass * *mass1*)

Initialize spring with two given masses and set the rest length to the distance of the two points.

Parameters

mass0	mass at end-point
mass1	mass at end-point

Here is the call graph for this function:



6.8.4 Member Data Documentation

6.8.4.1 Mass* Spring::m0 [private]

6.8.4.2 Mass * Spring::m1 [private]

6.8.4.3 double Spring::restLength

6.8.4.4 double Spring::stiffness

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

- src/Spring.h

- src/Spring.cpp

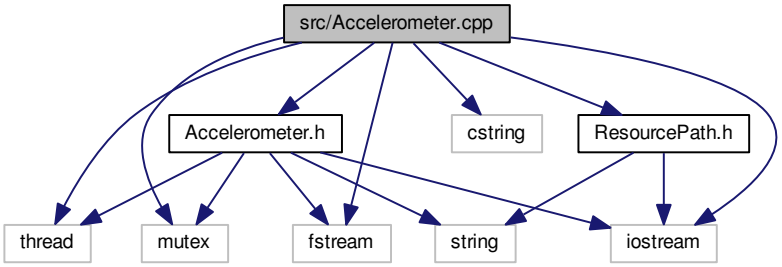
Chapter 7

File Documentation

7.1 src/Accelerometer.cpp File Reference

```
#include <thread>
#include <mutex>
#include <iostream>
#include <fstream>
#include <cstring>
#include "ResourcePath.h"
#include "Accelerometer.h"
```

Include dependency graph for Accelerometer.cpp:



Macros

- #define _PRIV_LINE_LEN 100

7.1.1 Macro Definition Documentation

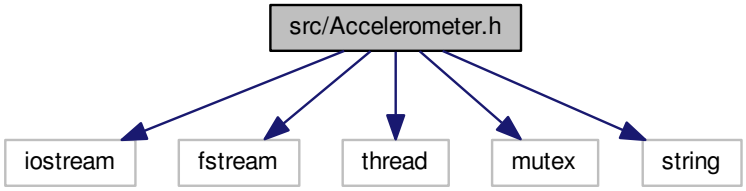
7.1.1.1 #define _PRIV_LINE_LEN 100

7.2 src/Accelerometer.h File Reference

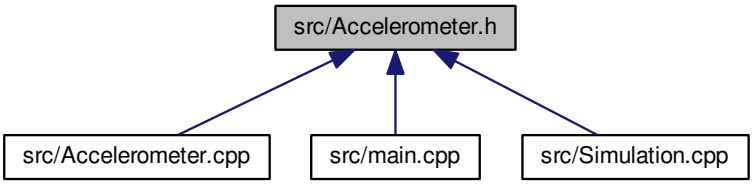
```
#include <iostream>
```

```
#include <fstream>
#include <thread>
#include <mutex>
#include <string>
```

Include dependency graph for Accelerometer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Accelerometer

Macros

- #define CFG_FILENAME "accelerometer.config"

7.2.1 Macro Definition Documentation

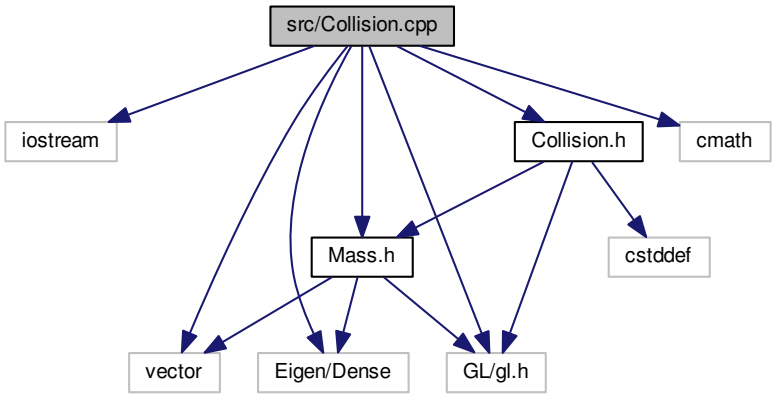
7.2.1.1 #define CFG_FILENAME "accelerometer.config"

This definition contains the hard-coded path to the device-config-file. TODO. do it in a generic way - some changes may be necessary to maintain multiple devices

7.3 src/Collision.cpp File Reference

```
#include <iostream>
#include <vector>
#include <cmath>
#include <Eigen/Dense>
#include <GL/gl.h>
#include "Collision.h"
#include "Mass.h"
```

Include dependency graph for Collision.cpp:



Namespaces

- std

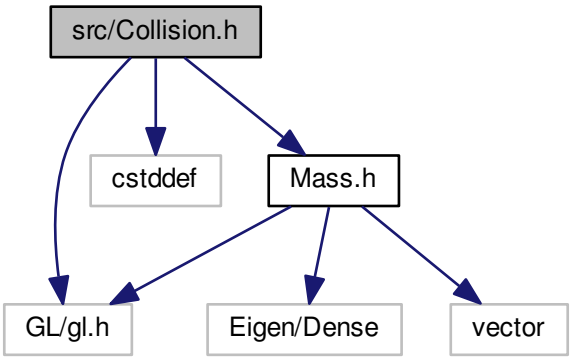
Functions

- bool std::vertexInTriangle (const Eigen::Vector3d &P, const Eigen::Vector3d &A, const Eigen::Vector3d &B, const Eigen::Vector3d &C, const float epsilon, float &dist, Eigen::Vector3d &N)
- bool std::baryVertexInTriangle (const Eigen::Vector3d &P, const Eigen::Vector3d &A, const Eigen::Vector3d &B, const Eigen::Vector3d &C)
- void std::collisionDetectionAndResponse (vector< Mass > &points, size_t offsP, size_t lenP, GLfloat object↩_mesh[ ], size_t offsO, size_t lenO)

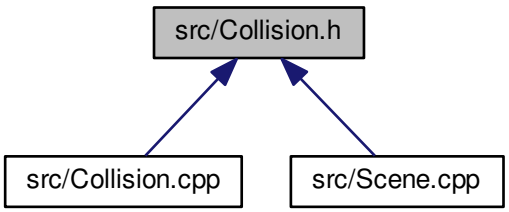
7.4 src/Collision.h File Reference

```
#include <GL/gl.h>
#include <cstddef>
#include "Mass.h"
```

Include dependency graph for Collision.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- std

Functions

- void std::collisionDetectionAndResponse (vector< Mass > &points, size_t offsP, size_t lenP, GLfloat object↩ _mesh[ ], size_t offsO, size_t lenO)

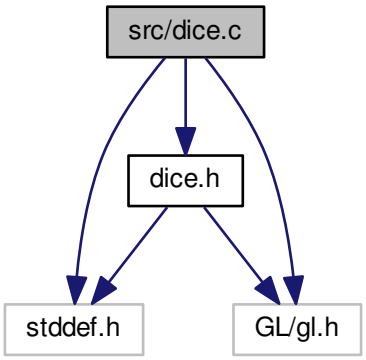
7.4.1 Detailed Description

This is a header file to include the functions needed for the collision detection algorithm.

7.5 src/dice.c File Reference

```
#include <stddef.h>
#include <GL/gl.h>
#include "dice.h"
```

Include dependency graph for dice.c:



Variables

- const size_t diceVertices = 78
- GLfloat dicePositions [234]
- GLfloat diceTexels [156]
- GLfloat diceNormals [234]
- const size_t diceObjectsWithMass = 2
- const size_t diceMasses [2]
- const size_t diceMassFwdOffs [2]
- const size_t diceMassVertices [2]
- const size_t diceMassRevOffs [2]
- const size_t diceMassRevOffsOrig [2]
- const size_t diceFwdIndexI [72]
- const size_t * diceFwdIndex [16]
- const size_t diceFwdIndexLength [16]
- const size_t diceRevIndex [72]
- const size_t diceObjects = 3
- const size_t diceObjectOffset [3]
- const size_t diceObjectLength [3]
- const char diceObjectNamesString [36]
- const char * diceObjectNames [3]
- const char diceTextureFilePath [34] = "/not_implemented_yet/edit/by.hand"

7.5.1 Detailed Description

This is a C-source file (.c) for the models "dice_Plane", "dice_RollingDie", "dice_Die" Don't edit! This is an auto-generated file by blender2oGL. Modifications are not permanent.

7.5.2 Variable Documentation

7.5.2.1 const size_t* diceFwdIndex[16]

Initial value:

```
= {
    &diceFwdIndexI[0],
    &diceFwdIndexI[5],
    &diceFwdIndexI[9],
    &diceFwdIndexI[14],
    &diceFwdIndexI[18],
    &diceFwdIndexI[23],
    &diceFwdIndexI[27],
    &diceFwdIndexI[32],
    &diceFwdIndexI[36],
    &diceFwdIndexI[41],
    &diceFwdIndexI[45],
    &diceFwdIndexI[50],
    &diceFwdIndexI[54],
    &diceFwdIndexI[59],
    &diceFwdIndexI[63],
    &diceFwdIndexI[68],
}
```

7.5.2.2 const size_t diceFwdIndexI[72]

Initial value:

```
= {
    6, 9, 18, 21, 37,
    7, 23, 24, 27,
    8, 10, 29, 30, 33,
    11, 35, 38, 40,
    12, 15, 19, 36, 39,
    17, 20, 22, 25,
    14, 16, 26, 28, 31,
    13, 32, 34, 41,
    42, 45, 54, 57, 73,
    43, 59, 60, 63,
    44, 46, 65, 66, 69,
    47, 71, 74, 76,
    48, 51, 55, 72, 75,
    53, 56, 58, 61,
    50, 52, 62, 64, 67,
    49, 68, 70, 77,
}
```

7.5.2.3 const size_t diceFwdIndexLength[16]

Initial value:

```
= {
    5,
    4,
    5,
    4,
    5,
    4,
    5,
    4,
    5,
    4,
    5,
    4,
    5,
    4,
    5,
    4,
}
```

7.5.2.4 const size_t diceMasses[2]

Initial value:

```
= {
    8,
    8,
}
```

7.5.2.5 const size_t diceMassFwdOffs[2]

Initial value:

```
= {
    0,
    8,
}
```

7.5.2.6 const size_t diceMassRevOffs[2]

Initial value:

```
= {
    0,
    36,
}
```

7.5.2.7 const size_t diceMassRevOffsOrig[2]

Initial value:

```
= {
    6,
    42,
}
```

7.5.2.8 const size_t diceMassVertices[2]

Initial value:

```
= {
    36,
    36,
}
```

7.5.2.9 GLfloat diceNormals[234]

7.5.2.10 const size_t diceObjectLength[3]

Initial value:

```
= {
    6,
    36,
    36,
}
```

7.5.2.11 const char* diceObjectNames[3]

Initial value:

```
= {
    &diceObjectNamesString[0],
    &diceObjectNamesString[11],
    &diceObjectNamesString[27],
}
```

7.5.2.12 const char diceObjectNamesString[36]

Initial value:

```
= {
    "dice_Plane\0"
    "dice_RollingDie\0"
    "dice_Die\0"
}
```

7.5.2.13 const size_t diceObjectOffset[3]

Initial value:

```
= {
    0,
    6,
    42,
}
```

7.5.2.14 const size_t diceObjects = 3

7.5.2.15 const size_t diceObjectsWithMass = 2

7.5.2.16 GLfloat dicePositions[234]

7.5.2.17 const size_t diceRevIndex[72]

Initial value:

```
= {
    0, 1, 2,
    0, 2, 3,
    4, 7, 6,
    4, 6, 5,
    0, 4, 5,
    0, 5, 1,
    1, 5, 6,
    1, 6, 2,
    2, 6, 7,
    2, 7, 3,
    4, 0, 3,
    4, 3, 7,
    8, 9, 10,
    8, 10, 11,
    12, 15, 14,
    12, 14, 13,
    8, 12, 13,
    8, 13, 9,
    9, 13, 14,
    9, 14, 10,
    10, 14, 15,
    10, 15, 11,
    12, 8, 11,
    12, 11, 15,
}
```

7.5.2.18 GLfloat diceTexels[156]

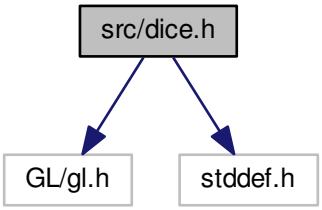
7.5.2.19 const char diceTextureFilePath[34] = "/not_implemented_yet/edit/by.hand"

7.5.2.20 const size_t diceVertices = 78

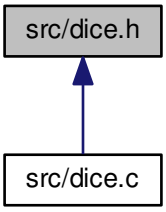
7.6 src/dice.h File Reference

```
#include <GL/gl.h>
#include <stddef.h>
```

Include dependency graph for dice.h:



This graph shows which files directly or indirectly include this file:



Variables

- const size_t diceVertices
- GLfloat dicePositions [234]
- GLfloat diceTexels [156]
- GLfloat diceNormals [234]
- const size_t diceObjectsWithMass
- const size_t diceMasses [2]
- const size_t diceMassFwdOffs [2]
- const size_t diceMassVertices [2]
- const size_t diceMassRevOffs [2]

- const size_t diceMassRevOffsOrig [2]
- const size_t diceFwdIndexI [72]
- const size_t * diceFwdIndex [16]
- const size_t diceFwdIndexLength [16]
- const size_t diceRevIndex [72]
- const size_t diceObjects
- const size_t diceObjectOffset [3]
- const size_t diceObjectLength [3]
- const char diceObjectNamesString [36]
- const char * diceObjectNames [3]
- const char diceTextureFilePath [34]

7.6.1 Detailed Description

This is a C-header file (.h) for the models "dice_Plane", "dice_RollingDie", "dice_Die" Don't edit! This is an auto-generated file by blender2oGL. Modifications are not permanent.

7.6.2 Variable Documentation

7.6.2.1 const size_t* diceFwdIndex[16]

7.6.2.2 const size_t diceFwdIndexI[72]

7.6.2.3 const size_t diceFwdIndexLength[16]

7.6.2.4 const size_t diceMasses[2]

7.6.2.5 const size_t diceMassFwdOffs[2]

7.6.2.6 const size_t diceMassRevOffs[2]

7.6.2.7 const size_t diceMassRevOffsOrig[2]

7.6.2.8 const size_t diceMassVertices[2]

7.6.2.9 GLfloat diceNormals[234]

7.6.2.10 const size_t diceObjectLength[3]

7.6.2.11 const char* diceObjectNames[3]

7.6.2.12 const char diceObjectNamesString[36]

7.6.2.13 const size_t diceObjectOffset[3]

7.6.2.14 const size_t diceObjects

7.6.2.15 const size_t diceObjectsWithMass

7.6.2.16 GLfloat dicePositions[234]

7.6.2.17 const size_t diceRevIndex[72]

7.6.2.18 GLfloat diceTexels[156]

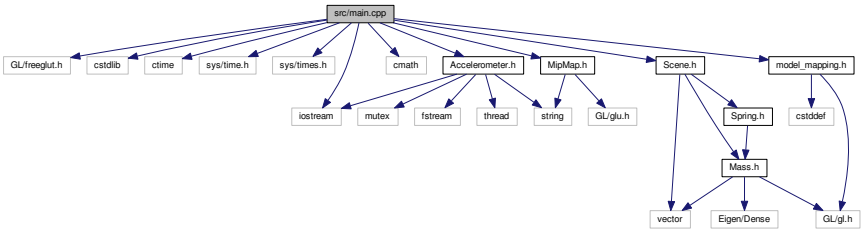
7.6.2.19 const char diceTextureFilePath[34]

7.6.2.20 const size_t diceVertices

7.7 src/main.cpp File Reference

```
#include <GL/freeglut.h>
#include <cstdlib>
#include <ctime>
#include <sys/time.h>
#include <sys/times.h>
#include <iostream>
#include <cmath>
#include "Accelerometer.h"
#include "Scene.h"
#include "MipMap.h"
#include "model_mapping.h"
```

Include dependency graph for main.cpp:



Functions

- ulong getTime ()
- void initialize ()
- void motionCallback (int, int)
- void reshapeCallback (int w, int h)
- void updateSimulation ()
- void displayCallback ()
- void keyboardCallback (unsigned char key, int, int)
- void mouseCallback (int button, int state, int x, int y)
- void idleCallback ()
- int main (int argc, char ∗argv[ ])

Variables

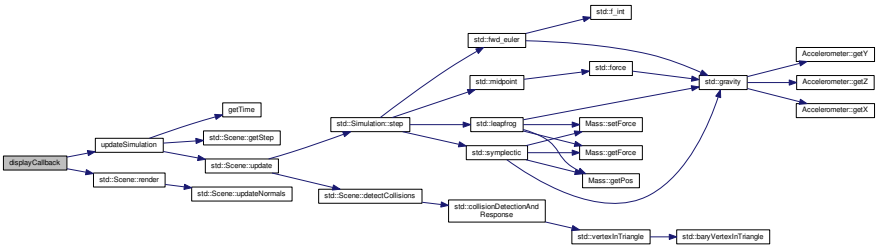
- Scene ∗ scene =nullptr
- static const double MAX_UPDATE_TIME = 0.01

7.7.1 Function Documentation

7.7.1.1 void displayCallback ( )

Display callback function. Called each times the display needs to be redrawn.

Here is the call graph for this function:



7.7.1.2 ulong getTime ( )

Get the system time. This function requires ∗NIX dependent system functions.

Returns

the time of now

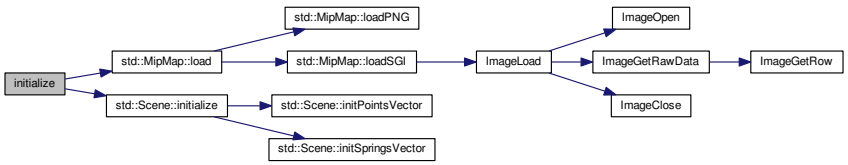
7.7.1.3 void idleCallback ( )

Idle function. Executed if OpenGL is in an idle state.

7.7.1.4 void initialize ( )

Initialize the OpenGL system and the scene.

Here is the call graph for this function:



7.7.1.5 void keyboardCallback (unsigned char *key*, int , int)

Keyboard callback function. Processes the input events from the keyboard.

Parameters

key	key mapped to an event

7.7.1.6 int main (int *argc*, char ∗ *argv[ ]*)

The main function.

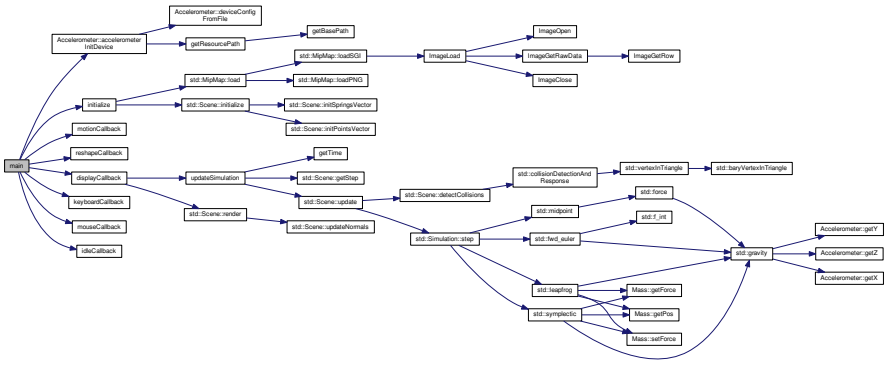
Parameters

argc	arguments counter.
argv	arguments vector. All arguments are passed on to `glutInit()` and `Scene()`

Returns

process return value `EXIT_SUCCESS` or `EXIT_FAILURE`

Here is the call graph for this function:



7.7.1.7 void motionCallback (int , int)

Mouse Motion callback function Processes the input events from the Mouse.

actually unused

7.7.1.8 void mouseCallback (int *button*, int *state*, int *x*, int *y*)

Mouse button callback function. Processes the input events from the Mouse.

Parameters

button	
state	
x	
y	

7.7.1.9 void reshapeCallback (int *w*, int *h*)

Window reshape callback function.

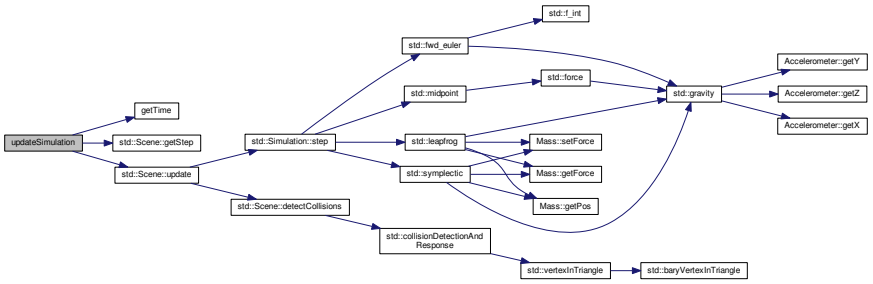
Parameters

w	window width after size change
h	window height after size change

7.7.1.10 void updateSimulation ( )

Update the simulation up to the present time.

Here is the call graph for this function:



7.7.2 Variable Documentation

7.7.2.1 const double MAX_UPDATE_TIME = 0.01 [static]

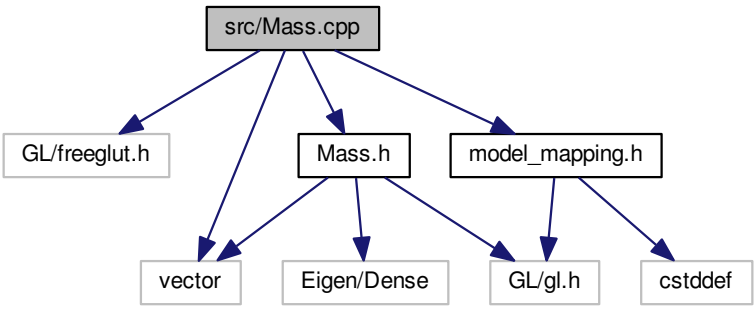
Maximum time allowed between updates when running in real-time mode (prevents performing too many calculations when running slower than real time)

7.7.2.2 Scene* scene =nullptr

7.8 src/Mass.cpp File Reference

```
#include <GL/freeglut.h>
#include <vector>
#include "Mass.h"
#include "model_mapping.h"
```

Include dependency graph for Mass.cpp:



Macros

- #define _DEBUG_MSG_C(X)
- #define _DEBUG_MSG(X)

7.8.1 Macro Definition Documentation

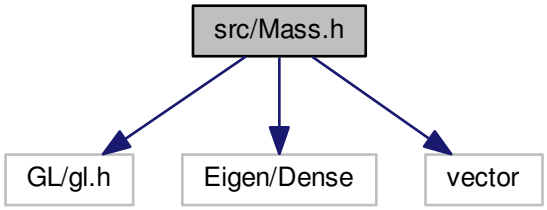
7.8.1.1 #define _DEBUG_MSG(X)

7.8.1.2 #define _DEBUG_MSG_C(X)

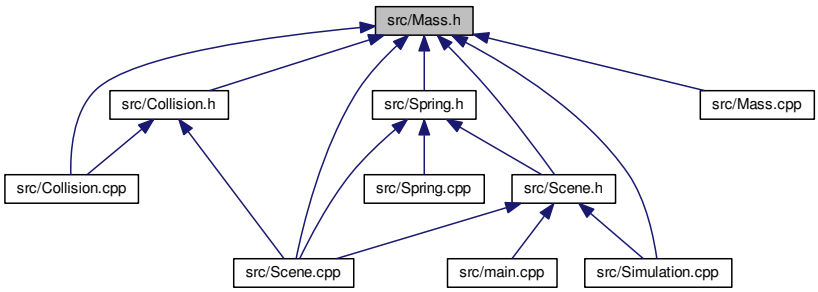
7.9 src/Mass.h File Reference

```
#include <GL/gl.h>
#include <Eigen/Dense>
#include <vector>
```

Include dependency graph for Mass.h:



This graph shows which files directly or indirectly include this file:



Classes

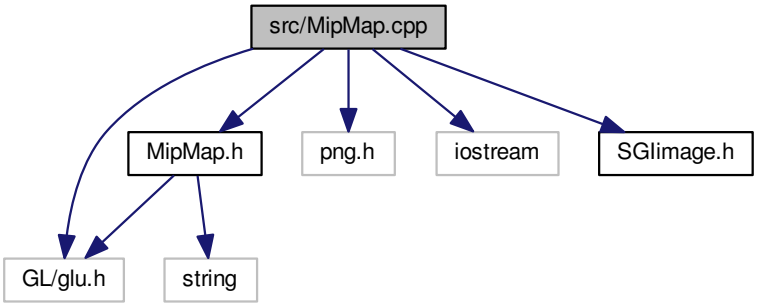
- class Mass

7.10 src/MipMap.cpp File Reference

```
#include "MipMap.h"
```

```
#include <GL/glu.h>
#include <png.h>
#include <iostream>
#include "SGIimage.h"
```

Include dependency graph for MipMap.cpp:



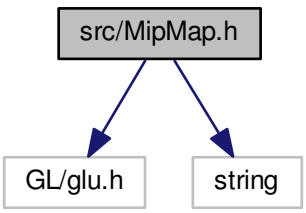
Namespaces

- std

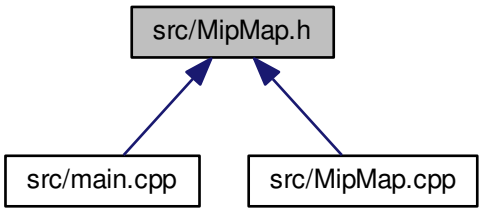
7.11 src/MipMap.h File Reference

```
#include <GL/glu.h>
#include <string>
```

Include dependency graph for MipMap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class std::MipMap

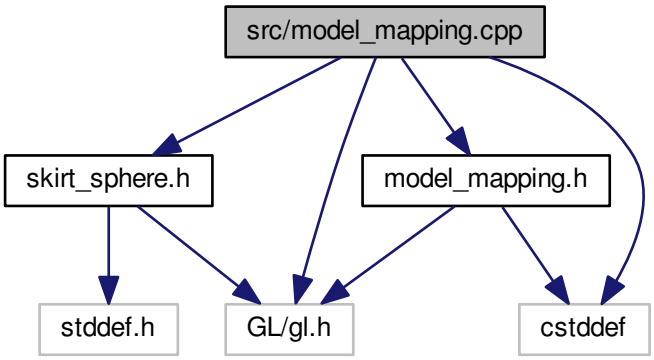
Namespaces

- std

7.12 src/model_mapping.cpp File Reference

```
#include <GL/gl.h>
#include <cstddef>
#include "model_mapping.h"
#include "skirt_sphere.h"
```

Include dependency graph for model_mapping.cpp:



Namespaces

- std

Macros

- #define MODEL(X) skirt_sphere##X

Variables

- const size_t std::model3dVertices = MODEL(Vertices)

  number of vertices
- GLfloat * std::model3dPositions = MODEL(Positions)

  all vertex positions
- GLfloat * std::model3dTexels = MODEL(Texels)

  all texture coordinates
- GLfloat * std::model3dNormals = MODEL(Normals)

  all normals of the face surfaces
- const size_t std::model3dObjectsWithMass = MODEL(ObjectsWithMass)

  number of objects to apply to the mass-spring simulation
- const size_t * std::model3dMasses = MODEL(Masses)

  array of 3D-objects with mass
- const size_t * std::model3dMassFwdOffs = MODEL(MassFwdOffs)

  offset in the points array
- const size_t * std::model3dMassVertices = MODEL(MassVertices)

  number of vertices per object with mass
- const size_t * std::model3dMassRevOffs = MODEL(MassRevOffs)

  offsets for the model3dRevIndex array
- const size_t * std::model3dMassRevOffsOrig = MODEL(MassRevOffsOrig)

  offset to the first vertex of an object with mass
- const size_t ** std::model3dFwdIndex = MODEL(FwdIndex)

  indices of the mass-points in the positions array
- const size_t * std::model3dFwdIndexLength = MODEL(FwdIndexLength)

  number of indices for each mass-point
- const size_t * std::model3dRevIndex = MODEL(RevIndex)

  index of a vertex in the mass-points array
- const size_t std::model3dObjects = MODEL(Objects)

  number of 3D-objects
- const size_t * std::model3dObjectOffset = MODEL(ObjectOffset)

  offset to the first vertex of a 3D-object
- const size_t * std::model3dObjectLength = MODEL(ObjectLength)

  number of vertices for each 3D-object
- const char ** std::model3dObjectNames = (const char**) MODEL(ObjectNames)

  names of the objects (for identification)
- const char * std::model3dTextureFilePath = "textures/textures_all.rgb"

  path to the texture-image file

7.12.1 Detailed Description

This file includes the vertex, texture and normals coordinates and some 3D object topology informations.

It is a wrapper to make the code relying on these variables independent from varying names supplied by the header file generator tool `blender2oGL`. Use the following line to generate a customized header file:

```
blender2oGL ./path/to-your/file.obj -m -s -o -r -a file_objectWithMass -a file_otherObjectIfDesired
```

Edit the pointers in this file to fit the code to an exchanged header file. DON'T EDIT them in the generated .h file, they may be overwritten by the generator tool.

7.12.2 Macro Definition Documentation

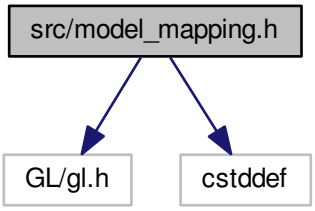
7.12.2.1 #define MODEL(*X*) skirt_sphere##X

define the prefix of the structures for mapping the variable names

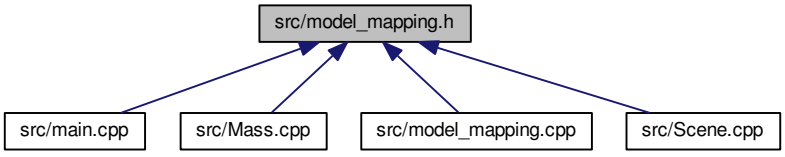
7.13 src/model_mapping.h File Reference

```
#include <GL/gl.h>
#include <cstddef>
```

Include dependency graph for model_mapping.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- std

7.13.1 Detailed Description

This file includes the vertex, texture and normals coordinates and some 3D object topology informations.

It is a wrapper to make the code relying on these variables independent from varying names supplied by the header file generator tool 'blender2oGL'

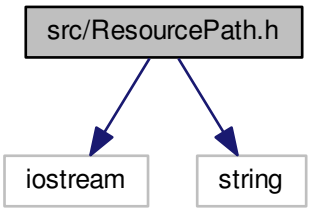
Edit the pointers in the 'model_mapping.cpp' file to fit the code to an exchanged header file. DON'T EDIT them in the generated .h file, they may be overwritten by the generator tool.

See 'model_mapping.cpp' for further details.

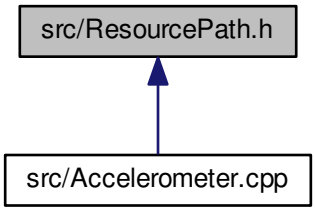
7.14 src/ResourcePath.h File Reference

```
#include <iostream>
#include <string>
```

Include dependency graph for ResourcePath.h:



This graph shows which files directly or indirectly include this file:



Functions

- string getBasePath (int, char *argv[]=nullptr)

  Get the path of the application folder.
- string getResourcePath (string filePath, string resourcesFolder="resources", int argc=0, char *argv[]=nullptr)

  Get path to resource file relative to the BasePath().

7.14.1 Function Documentation

7.14.1.1 string getBasePath (int , char * *argv[ ]* = nullptr)

Get the path of the application folder.

The path returned is an absolute path with trailing path separator. The first call of this function initializes the path constant and thus, it needs non-null values for argc and argv. If the initialization fails, the return value is an empty string.

Parameters

argv	the arguments list provided by the main() function on first call, omit or set to nullptr on succeeding calls

Returns

the base path or an empty string if not initialized.

7.14.1.2 string getResourcePath (string *filePath*, string *resourcesFolder* = "resources", int *argc* = 0, char * *argv[ ]* = nullptr)

Get path to resource file relative to the BasePath().

The path returned is an absolute path resolved from ../resources/filePath. If the path to a sub-folder is required use the path separator supported by the target system.

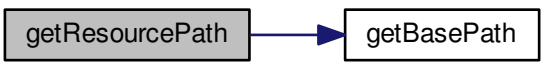
Parameters

filePath	the path to the requested file in the resources folder
resourcesFolder	name of the folder without path separators or empty string to address the parent folder, default is "resources"
argc	the arguments count of the main() function on first call
argv	the arguments list of the main() function on first call

Returns

the base path to the resource folder appended by filePath

Here is the call graph for this function:

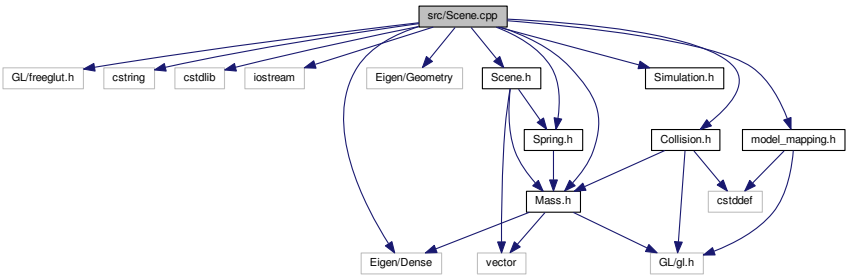


7.15 src/Scene.cpp File Reference

```
#include <GL/freeglut.h>
```

```
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include "Scene.h"
#include "Mass.h"
#include "Spring.h"
#include "Simulation.h"
#include "Collision.h"
#include "model_mapping.h"
```

Include dependency graph for Scene.cpp:



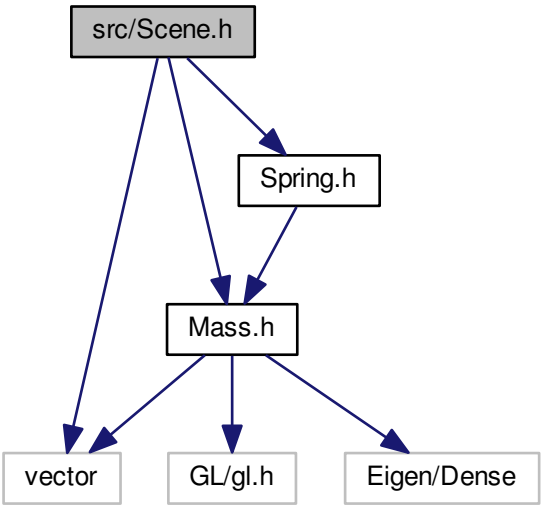
Namespaces

- std

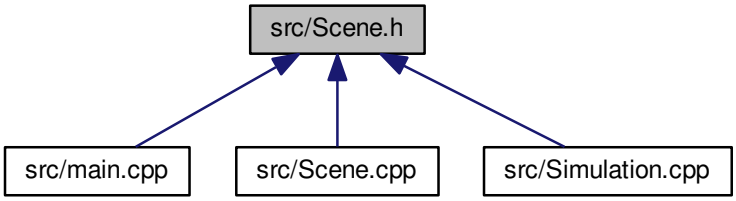
7.16 src/Scene.h File Reference

```
#include <vector>
#include "Mass.h"
#include "Spring.h"
```

Include dependency graph for Scene.h:



This graph shows which files directly or indirectly include this file:



Classes

- class std::Scene

Namespaces

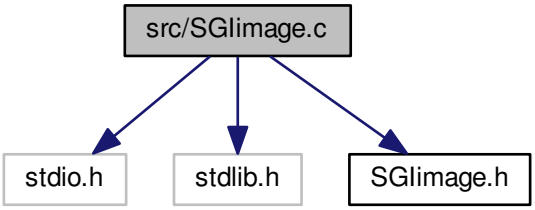
- std

7.17 src/SGlimage.c File Reference

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include "SGIimage.h"
```

Include dependency graph for SGIimage.c:



Classes

- struct Image

Macros

- #define IMAGIC 0x01da
- #define IMAGIC_SWAP 0xda01
- #define SWAP_SHORT_BYTES(x) ((((x) & 0xff) << 8) | (((x) & 0xff00) >> 8))
- #define SWAP_LONG_BYTES(x)

Functions

- static Image * ImageOpen (char *fileName)
- static void ImageClose (Image *image)
- static void ImageGetRow (Image *image, unsigned char *buf, int y, int z)
- static void ImageGetRawData (Image *image, unsigned char *data)
- IMAGE * ImageLoad (char *fileName)

7.17.1 Macro Definition Documentation

7.17.1.1 #define IMAGIC 0x01da

7.17.1.2 #define IMAGIC_SWAP 0xda01

7.17.1.3 #define SWAP_LONG_BYTES(*x*)

Value:

```
(((((x) & 0xff) << 24) | (((x) & 0xff00) << 8)) | \
((((x) & 0xff0000) >> 8) | (((x) & 0xff000000) >> 24)))
```

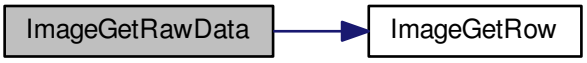
7.17.1.4 #define SWAP_SHORT_BYTES(*x*) ((((x) & 0xff) << 8) | (((x) & 0xff00) >> 8))

7.17.2 Function Documentation

7.17.2.1 static void ImageClose (Image * *image*) [static]

7.17.2.2 static void ImageGetRawData (Image * *image*, unsigned char * *data*) [static]

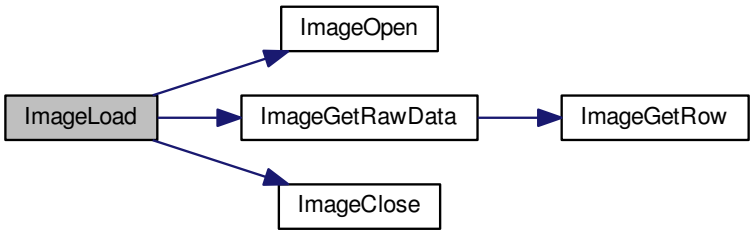
Here is the call graph for this function:



7.17.2.3 static void ImageGetRow (Image * *image*, unsigned char * *buf*, int *y*, int *z*) [static]

7.17.2.4 IMAGE* ImageLoad (char * *fileName*)

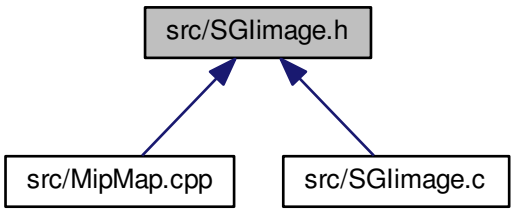
Here is the call graph for this function:



7.17.2.5 static Image* ImageOpen (char * *fileName*) [static]

7.18 src/SGIimage.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct IMAGE

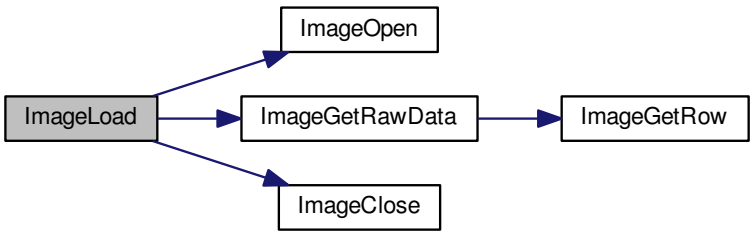
Functions

- IMAGE * ImageLoad (char *)

7.18.1 Function Documentation

7.18.1.1 IMAGE* ImageLoad (char *)

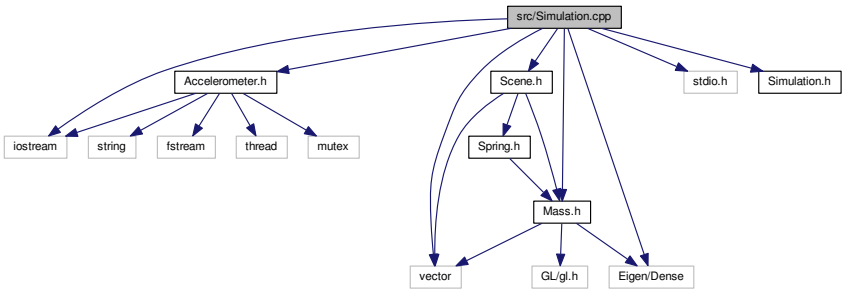
Here is the call graph for this function:



7.19 src/Simulation.cpp File Reference

```
#include <iostream>
#include <vector>
#include <stdio.h>
#include <Eigen/Dense>
#include "Mass.h"
#include "Scene.h"
#include "Accelerometer.h"
#include "Simulation.h"
```

Include dependency graph for Simulation.cpp:



Namespaces

- std

Functions

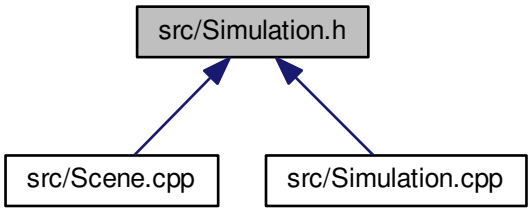
- void std::fwd_euler (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- Eigen::Vector3d std::f_int (Mass &pt, vector< Spring > &springs)
- Eigen::Vector3d std::gravity ()
- void std::symplectic (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void std::leapfrog (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void std::midpoint (double dt, vector< Mass > &points, vector< Spring > &springs, bool interaction)
- void std::force (vector< Mass > &points, vector< Spring > &springs, bool interaction)

Variables

- bool std::leapFrogInitialized = false
- double std::t = 0
- double std::floorLevel = -1.
- double std::repulsiveSpringConst = -50.

7.20 src/Simulation.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class std::Simulation

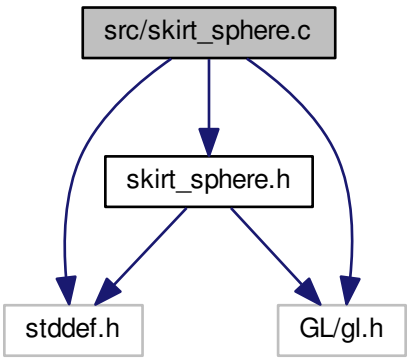
Namespaces

- std

7.21 src/skirt_sphere.c File Reference

```
#include <stddef.h>
#include <GL/gl.h>
#include "skirt_sphere.h"
```

Include dependency graph for skirt_sphere.c:



Variables

- const size_t skirt_sphereVertices = 4008
- GLfloat skirt_spherePositions [12024]
- GLfloat skirt_sphereTexels [8016]
- GLfloat skirt_sphereNormals [12024]
- const size_t skirt_sphereObjectsWithMass = 1
- const size_t skirt_sphereMasses [1]
- const size_t skirt_sphereMassFwdOffs [1]
- const size_t skirt_sphereMassVertices [1]
- const size_t skirt_sphereMassRevOffs [1]
- const size_t skirt_sphereMassRevOffsOrig [1]
- const size_t skirt_sphereFwdIndexI [1824]
- const size_t * skirt_sphereFwdIndex [353]
- const size_t skirt_sphereFwdIndexLength [353]
- const size_t skirt_sphereRevIndex [1824]
- const size_t skirt_sphereObjects = 3
- const size_t skirt_sphereObjectOffset [3]
- const size_t skirt_sphereObjectLength [3]
- const char skirt_sphereObjectNamesString [59]
- const char * skirt_sphereObjectNames [3]
- const char skirt_sphereTextureFilePath [34] = "/not_implemented_yet/edit/by.hand"

7.21.1 Detailed Description

This is a C-source file (.c) for the models "skirt_sphere_Cone", "skirt_sphere_Grid", "skirt_sphere_Icosphere" Don't edit! This is an auto-generated file by blender2oGL. Modifications are not permanent.

7.21.2 Variable Documentation

7.21.2.1 const size_t* skirt_sphereFwdIndex[353]

7.21.2.2 const size_t skirt_sphereFwdIndexI[1824]

7.21.2.3 const size_t skirt_sphereFwdIndexLength[353]

7.21.2.4 const size_t skirt_sphereMasses[1]

Initial value:

```
= {
    353,
}
```

7.21.2.5 const size_t skirt_sphereMassFwdOffs[1]

Initial value:

```
= {
    0,
}
```

7.21.2.6 const size_t skirt_sphereMassRevOffs[1]

Initial value:

```
= {
    0,
}
```

7.21.2.7 const size_t skirt_sphereMassRevOffsOrig[1]

Initial value:

```
= {
    0,
}
```

7.21.2.8 const size_t skirt_sphereMassVertices[1]

Initial value:

```
= {
    1824,
}
```

7.21.2.9 GLfloat skirt_sphereNormals[12024]

7.21.2.10 const size_t skirt_sphereObjectLength[3]

Initial value:

```
= {
    1824,
    1944,
    240,
}
```

7.21.2.11 const char* skirt_sphereObjectNames[3]

Initial value:

```
= {
    &skirt_sphereObjectNamesString[0],
    &skirt_sphereObjectNamesString[18],
    &skirt_sphereObjectNamesString[36],
}
```

7.21.2.12 const char skirt_sphereObjectNamesString[59]

Initial value:

```
= {
    "skirt_sphere_Cone\0"
    "skirt_sphere_Grid\0"
    "skirt_sphere_Icosphere\0"
}
```

7.21.2.13 const size_t skirt_sphereObjectOffset[3]

Initial value:

```
= {
    0,
    1824,
    3768,
}
```

7.21.2.14 const size_t skirt_sphereObjects = 3

7.21.2.15 const size_t skirt_sphereObjectsWithMass = 1

7.21.2.16 GLfloat skirt_spherePositions[12024]

7.21.2.17 const size_t skirt_sphereRevIndex[1824]

7.21.2.18 GLfloat skirt_sphereTexels[8016]

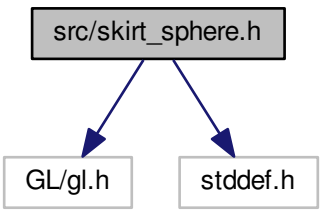
7.21.2.19 const char skirt_sphereTextureFilePath[34] = "/not_implemented_yet/edit/by.hand"

7.21.2.20 const size_t skirt_sphereVertices = 4008

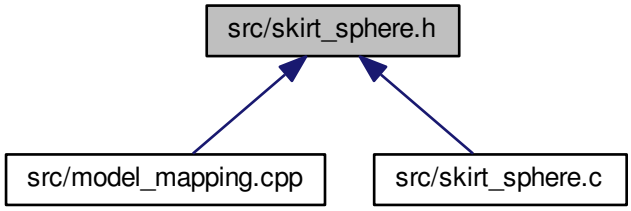
7.22 src/skirt_sphere.h File Reference

```
#include <GL/gl.h>
#include <stddef.h>
```

Include dependency graph for skirt_sphere.h:



This graph shows which files directly or indirectly include this file:



Variables

- const size_t skirt_sphereVertices
- GLfloat skirt_spherePositions [12024]
- GLfloat skirt_sphereTexels [8016]
- GLfloat skirt_sphereNormals [12024]
- const size_t skirt_sphereObjectsWithMass
- const size_t skirt_sphereMasses [1]
- const size_t skirt_sphereMassFwdOffs [1]
- const size_t skirt_sphereMassVertices [1]
- const size_t skirt_sphereMassRevOffs [1]
- const size_t skirt_sphereMassRevOffsOrig [1]
- const size_t skirt_sphereFwdIndexI [1824]
- const size_t * skirt_sphereFwdIndex [353]
- const size_t skirt_sphereFwdIndexLength [353]
- const size_t skirt_sphereRevIndex [1824]
- const size_t skirt_sphereObjects
- const size_t skirt_sphereObjectOffset [3]
- const size_t skirt_sphereObjectLength [3]
- const char skirt_sphereObjectNamesString [59]
- const char * skirt_sphereObjectNames [3]
- const char skirt_sphereTextureFilePath [34]

7.22.1 Detailed Description

This is a C-header file (.h) for the models "skirt_sphere_Cone", "skirt_sphere_Grid", "skirt_sphere_Icosphere" Don't edit! This is an auto-generated file by blender2oGL. Modifications are not permanent.

7.22.2 Variable Documentation

7.22.2.1 const size_t* skirt_sphereFwdIndex[353]

7.22.2.2 const size_t skirt_sphereFwdIndexI[1824]

7.22.2.3 const size_t skirt_sphereFwdIndexLength[353]

7.22.2.4 const size_t skirt_sphereMasses[1]

7.22.2.5 const size_t skirt_sphereMassFwdOffs[1]

7.22.2.6 const size_t skirt_sphereMassRevOffs[1]

7.22.2.7 const size_t skirt_sphereMassRevOffsOrig[1]

7.22.2.8 const size_t skirt_sphereMassVertices[1]

7.22.2.9 GLfloat skirt_sphereNormals[12024]

7.22.2.10 const size_t skirt_sphereObjectLength[3]

7.22.2.11 const char* skirt_sphereObjectNames[3]

7.22.2.12 const char skirt_sphereObjectNamesString[59]

7.22.2.13 const size_t skirt_sphereObjectOffset[3]

7.22.2.14 const size_t skirt_sphereObjects

7.22.2.15 const size_t skirt_sphereObjectsWithMass

7.22.2.16 GLfloat skirt_spherePositions[12024]

7.22.2.17 const size_t skirt_sphereRevIndex[1824]

7.22.2.18 GLfloat skirt_sphereTexels[8016]

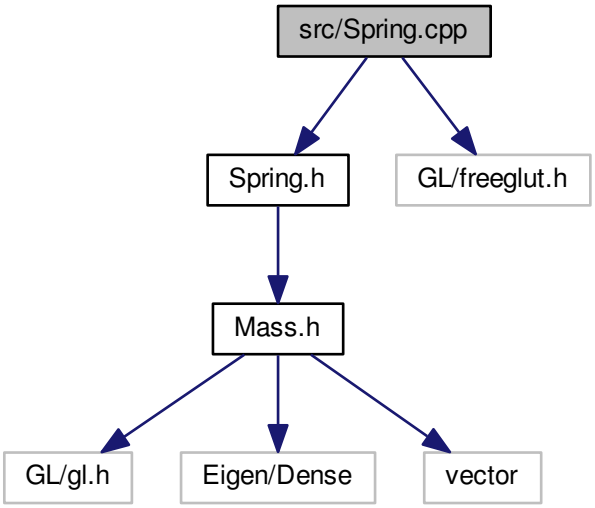
7.22.2.19 const char skirt_sphereTextureFilePath[34]

7.22.2.20 const size_t skirt_sphereVertices

7.23 src/Spring.cpp File Reference

```
#include "Spring.h"
#include <GL/freeglut.h>
```

Include dependency graph for Spring.cpp:



Macros

- #define _DEBUG_MSG_C(X)
- #define _DEBUG_MSG(X)

7.23.1 Macro Definition Documentation

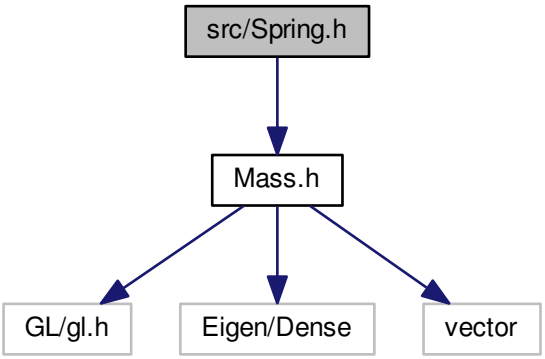
7.23.1.1 #define _DEBUG_MSG(*X*)

7.23.1.2 #define _DEBUG_MSG_C(*X*)

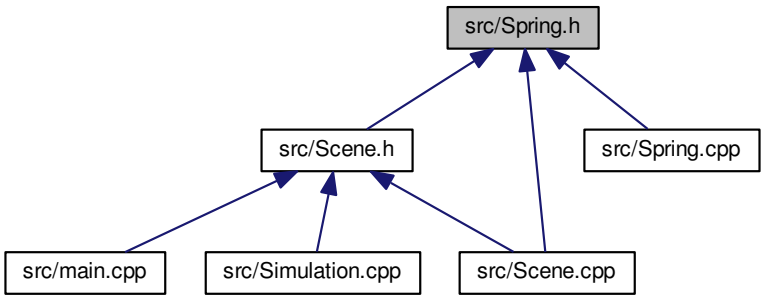
7.24 src/Spring.h File Reference

```
#include "Mass.h"
```

Include dependency graph for Spring.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Spring