ShapeOp

0.1.0

Generated by Doxygen 1.8.9.1

Thu Apr 2 2015 03:43:27

Contents

1 Main Page **1**

1.1 Introduction . . . 1

1.2 Compilation and Installation . . . 1

1.3 Documentation about the Libary and the Different Applications . . . 2

2 C++/OpenGL Viewer **3**

2.1 Compilation . . . 3

3 ShapeOp for Grasshopper(Rhino) **5**

3.1 Setup . . . 5

3.2 Run . . . 6

4 libShapeOp **7**

4.1 C++ libShapeOp . . . 7

4.2 C API . . . 7

4.3 Automatic Bindings with SWIG . . . 7

4.4 Coding Rules . . . 8

5 A Tutorial on Adding an Orientation Constraint to ShapeOp **9**

5.1 Mean-centering . . . 9

5.2 Projection . . . 9

5.3 C++ Implementation . . . 9

5.4 C API: . . . 11

5.5 Grasshopper: . . . 11

6 Namespace Index **13**

6.1 Namespace List . . . 13

7 Hierarchical Index **15**

7.1 Class Hierarchy . . . 15

8 Class Index **17**

8.1 Class List . . . 17

9 File Index **19**

9.1 File List . . . 19

10 Namespace Documentation **21**

10.1 ShapeOp Namespace Reference . . . 21

10.1.1 Detailed Description . . . 23

11 Class Documentation **25**

11.1 ShapeOp::AngleConstraint Class Reference . . . 25

11.1.1 Detailed Description . . . 25

11.1.2 Constructor & Destructor Documentation . . . 26

11.1.2.1 AngleConstraint . . . 26

11.2 ShapeOp::AreaConstraint Class Reference . . . 27

11.2.1 Detailed Description . . . 27

11.2.2 Constructor & Destructor Documentation . . . 28

11.2.2.1 AreaConstraint . . . 28

11.3 ShapeOp::BendingConstraint Class Reference . . . 28

11.3.1 Detailed Description . . . 29

11.3.2 Constructor & Destructor Documentation . . . 29

11.3.2.1 BendingConstraint . . . 29

11.4 ShapeOp::CGSolver Class Reference . . . 29

11.4.1 Detailed Description . . . 30

11.5 ShapeOp::CircleConstraint Class Reference . . . 30

11.5.1 Detailed Description . . . 30

11.5.2 Constructor & Destructor Documentation . . . 30

11.5.2.1 CircleConstraint . . . 30

11.6 ShapeOp::ClosenessConstraint Class Reference . . . 31

11.6.1 Detailed Description . . . 31

11.6.2 Constructor & Destructor Documentation . . . 31

11.6.2.1 ClosenessConstraint . . . 31

11.7 ShapeOp::Constraint Class Reference . . . 32

11.7.1 Detailed Description . . . 33

11.7.2 Constructor & Destructor Documentation . . . 33

11.7.2.1 Constraint . . . 33

11.7.3 Member Function Documentation . . . 33

11.7.3.1 addConstraint . . . 33

11.7.3.2 project . . . 34

11.7.3.3 shapeConstraintFactory . . . 35

11.8 ShapeOp::EdgeStrainConstraint Class Reference . . . 36

11.8.1 Detailed Description . . . 36

11.8.2 Constructor & Destructor Documentation . . . 37

11.8.2.1 EdgeStrainConstraint . . . 37
11.9 ShapeOp::Force Class Reference . . . 37
11.9.1 Detailed Description . . . 37
11.10 ShapeOp::GravityForce Class Reference . . . 37
11.10.1 Detailed Description . . . 38
11.11 ShapeOp::LineConstraint Class Reference . . . 38
11.11.1 Detailed Description . . . 38
11.11.2 Constructor & Destructor Documentation . . . 39
11.11.2.1 LineConstraint . . . 39
11.12 ShapeOp::LSSolver Class Reference . . . 39
11.12.1 Detailed Description . . . 39
11.13 ShapeOp::MINRESSolver Class Reference . . . 39
11.13.1 Detailed Description . . . 40
11.14 ShapeOp::ParallelogramConstraint Class Reference . . . 40
11.14.1 Detailed Description . . . 41
11.14.2 Constructor & Destructor Documentation . . . 41
11.14.2.1 ParallelogramConstraint . . . 41
11.15 ShapeOp::PlaneConstraint Class Reference . . . 41
11.15.1 Detailed Description . . . 41
11.15.2 Constructor & Destructor Documentation . . . 42
11.15.2.1 PlaneConstraint . . . 42
11.16 ShapeOp::RectangleConstraint Class Reference . . . 42
11.16.1 Detailed Description . . . 42
11.16.2 Constructor & Destructor Documentation . . . 42
11.16.2.1 RectangleConstraint . . . 42
11.17 ShapeOpSolver Struct Reference . . . 43
11.17.1 Detailed Description . . . 43
11.18 ShapeOp::SimilarityConstraint Class Reference . . . 43
11.18.1 Detailed Description . . . 44
11.18.2 Constructor & Destructor Documentation . . . 44
11.18.2.1 SimilarityConstraint . . . 44
11.18.3 Member Function Documentation . . . 44
11.18.3.1 setShapes . . . 44
11.19 ShapeOp::SimplicialLDLTSolver Class Reference . . . 44
11.19.1 Detailed Description . . . 45
11.20 ShapeOp::Solver Class Reference . . . 45
11.20.1 Detailed Description . . . 45
11.20.2 Member Function Documentation . . . 46
11.20.2.1 initialize . . . 46
11.20.2.2 solve . . . 46

11.21 ShapeOp::SORSolver Class Reference . . . 46
11.21.1 Detailed Description . . . 46
11.21.2 Constructor & Destructor Documentation . . . 47
11.21.2.1 SORSolver . . . 47
11.22 ShapeOp::SphereConstraint Class Reference . . . 48
11.22.1 Detailed Description . . . 48
11.22.2 Constructor & Destructor Documentation . . . 48
11.22.2.1 SphereConstraint . . . 48
11.23 ShapeOp::TetrahedronStrainConstraint Class Reference . . . 49
11.23.1 Detailed Description . . . 49
11.23.2 Constructor & Destructor Documentation . . . 49
11.23.2.1 TetrahedronStrainConstraint . . . 49
11.24 ShapeOp::TriangleStrainConstraint Class Reference . . . 50
11.24.1 Detailed Description . . . 50
11.24.2 Constructor & Destructor Documentation . . . 50
11.24.2.1 TriangleStrainConstraint . . . 50
11.25 ShapeOp::UniformLaplacianConstraint Class Reference . . . 51
11.25.1 Detailed Description . . . 51
11.25.2 Constructor & Destructor Documentation . . . 51
11.25.2.1 UniformLaplacianConstraint . . . 51
11.26 ShapeOp::VertexForce Class Reference . . . 51
11.26.1 Detailed Description . . . 52
11.27 ShapeOp::VolumeConstraint Class Reference . . . 52
11.27.1 Detailed Description . . . 53
11.27.2 Constructor & Destructor Documentation . . . 53
11.27.2.1 VolumeConstraint . . . 53

12 File Documentation **55**

12.1 libShapeOp/api/API.h File Reference . . . 55
12.1.1 Detailed Description . . . 56
12.1.2 Enumeration Type Documentation . . . 56
12.1.2.1 shapeop_err . . . 56
12.1.3 Function Documentation . . . 56
12.1.3.1 shapeop_addConstraint . . . 56
12.1.3.2 shapeop_addGravityForce . . . 58
12.1.3.3 shapeop_editConstraint . . . 58
12.1.3.4 shapeop_init . . . 59
12.1.3.5 shapeop_initDynamic . . . 59
12.1.3.6 shapeop_solve . . . 60
12.2 libShapeOp/src/Common.h File Reference . . . 60

12.2.1 Detailed Description . . . 60
12.3 libShapeOp/src/Constraint.h File Reference . . . 60
12.3.1 Detailed Description . . . 61
12.3.2 Macro Definition Documentation . . . 61
12.3.2.1 M_PI . . . 61
12.4 libShapeOp/src/Force.h File Reference . . . 61
12.4.1 Detailed Description . . . 62
12.5 libShapeOp/src/LSSolver.h File Reference . . . 62
12.5.1 Detailed Description . . . 62
12.6 libShapeOp/src/Solver.h File Reference . . . 63
12.6.1 Detailed Description . . . 63
12.7 libShapeOp/src/Types.h File Reference . . . 63
12.7.1 Detailed Description . . . 64

Bibliography **65**

Index **67**

Chapter 1

Main Page

1.1 Introduction

This library introduces a unified optimization framework for static and dynamic geometry processing based on [1] and [2]. This library comes from an effort to make research reproducible and to transfer knowledge and technology.

This project was a joint collaboration:

- ShapeOp Solver: *Sofien Bouaziz*, *Mario Deuss*, *Bailin Deng*
- Rhino/Kangaroo: *Anders Holden Deleuran*, *Mario Deuss*, *Bailin Deng*, *Daniel Piker*
- Project Management: *Mario Deuss*, *Mark Pauly*

We would also like to thank all our research collaborators without which this project could not have been possible:

- *Sebastian* Martin
- *Tiantian* Liu
- *Ladislav* Kavan
- *Yuliy* Schwartzburg
- *Thibaut* Weise

1.2 Compilation and Installation

Getting the library.

Download the ShapeOp source code from http://shapeop.org/.

Compilation under Unix and Mac.

Install CMake (http://www.cmake.org) and Eigen (http://eigen.tuxfamily.org/) using your favorite package manager. Then in the main repository type:

- mkdir build

- cd build
- cmake ..
- make

After building the library, you can generate the documentation by typing:

- cd doc
- doxygen DoxyShapeOp

To generate the documentation first install Doxygen (http://www.doxygen.org/). You will also need to have Bibtex and Perl available to get the bibliography compiled.

Compilation under Windows.

Download and install CMake (http://www.cmake.org) and Eigen (http://eigen.tuxfamily.org/).

- configure the sources with cmake
- specify the eigen include directory
- generate a visual studio solution
- open the visual studio solution and compile.

More details on the different compilation options.

For more details on the different compilation options see libShapeOp.

1.3 Documentation about the Libary and the Different Applications

- libShapeOp
- ShapeOp for Grasshopper(Rhino)
- C++/OpenGL Viewer
- A Tutorial on Adding an Orientation Constraint to ShapeOp

Chapter 2

C++/OpenGL Viewer

2.1 Compilation

This viewer enables rendering and interaction with a 3d scene using ShapeOp. To compile, install

- glew (http://glew.sourceforge.net/)
- glfw (http://www.glfw.org/)
- opengp (https://github.com/OpenGP/OpenGP)

When running cmake, specify the following additional argument:

cmake -DOPENGL_VIEWER=TRUE

Chapter 3

ShapeOp for Grasshopper(Rhino)

3.1 Setup

The implementation has the following assembly dependencies which will need to be installed on your system:

1. Rhino/Grasshopper:
   - Rhino Version 5 SR11 64-bit
   - Grasshopper 0.9.0076
   - Microsoft .NET Framework 4.5.1

2. Libraries:
   - ghpython.gha (http://www.food4rhino.com/project/ghpython)
   - vcomp120.dll (http://www.microsoft.com/en-us/download/details.aspx?id=40784)

3. Installing ghpython.gha:
   - Move the files to the Grasshopper Libraries folder "%appdata%\Grasshopper\Libraries".
   - Unblock them "Right-click the files -> Properties -> Unblock -> Ok".

4. Adding Grasshopper Libraries folder to RhinoPython: To import libraries placed in the Grasshopper libraries folder in a RhinoPython script you have to add its folder path to the list of directories which RhinoPython can access. You can do this in the script itself, or, hardcode it via the Rhino Python script editor (you only have to do this once):
   - In Rhino type in the command "EditPythonScript".
   - In this Python editor go "Tools -> Options -> Files".
   - Here you will see an overview of the directories which are currently referenced.
   - Add a reference to the Grasshopper Libraries folder (it may be hidden, of so unhide it).

5. Installing ShapeOp.dll and vcomp120.dll: You can either place these files in the default Windows folder (C↩:\Windows) or the Grasshopper Libraries folder (remember to unblock the files). In the latter case you will have to add the Grasshopper Libraries folder path to the Window path:

- Navigate to "Control Panel\System and Security\System".
- Click "Advanced System Settings".
- Click "Environment Variables".
- Append the Grasshopper Libraries folder path to the Path variable (in the System variables list).
- Restart Rhino.

3.2 Run

Open ShapeOpGHPython.ghx inside Rhino's Grasshopper.

Chapter 4

libShapeOp

4.1 C++ libShapeOp

libShapeOp is a lightweight C++ library for static and dynamic geometry processing. The main library is located in libShapeOp/src. The library can also be used as header only if **SHAPEOP_HEADER_ONLY** is defined. OpenMP can be activated by passing **-DOPENMP=TRUE** as options to CMake. The library can be compiled in **double** or in **float** by changing the ShapeOpScalar typedef located in Common.h. An example on how to use libShapeOp is provided (libShapeOp/bindings/c++/runme.cpp).

4.2 C API

A C API is provided along with the C++ library. The API is located in libShapeOp/api. This C API is useful when libShapeOp needs to be used from another programming language (see Automatic Bindings with SWIG for example). An example on how to use the API is provided (libShapeOp/bindings/c/runme.c).

4.3 Automatic Bindings with SWIG

SWIG (http://www.swig.org/) is a tool to automatically create binding between programs written in C and C++ and a variety of programming languages such as Javascript, Perl, and Python. The code generated by SWIG is compatible with both commercial and non-commercial projects. To install SWIG Unix or Mac use your favorite package manager. For Windows download the binary from the SWIG website. The CMake file builds Python, Java, and C# bindings as an example.

Python.

To generate the Python bindings you will need to pass **-DSWIG_PYTHON=TRUE** as options to CMake or set this cached variable in a CMake Gui. If CMake does not find your python installation automatically, you can set the **PYTHON_LIBRARY** and **PYTHON_INCLUDE_DIR** cached variable as before to your python version installed (to e.g. on OSX10.10 with the preinstalled python only: /System/Library/Frameworks/Python.framework/Headers respectively /usr/lib/libpython2.7.dylib or on Windows default installer from python.org to C:/Python27/python.exe respectively C:/Python27/include). Make sure you install a 64bit version of Python if you are building ShapeOp in 64bit (default), 32bit otherwise. A runme.py file is also provided (libShapeOp/bindings/python/runme.py). It shows how the Python bindings can be used to call libShapeOp. The runme.py file can be executed from the build directory by typing (when building with Visual Studio, first copy the file _shapeopPython.pyd from the Release subfolder into the folder of runme.py)

- python runme.py

For building bindings to python 3 add the flag -py3 as indicated in the comments in bindings/python/CMakeLists.txt, install python 3 and adapt the library and include_dir in cmake.

Java.

To generate the Java bindings you will need to pass **-DSWIG_JAVA=TRUE** as options to CMake. A runme.java file is also provided (libShapeOp/bindings/java/runme.java). It shows how the Java bindings can be used to call libShapeOp. The runme.java file can be executed from the build directory by typing

- javac ∗.java
- java runme

C#.

To generate the C# bindings you will need to pass **-DSWIG_CSHARP=TRUE** as options to CMake. A runme.cs file is also provided (libShapeOp/bindings/csharp/runme.cs). It shows how the C# bindings can be used to call libShapeOp. Under Unix first install Mono (http://www.mono-project.com) using the package manager. The runme.cs file can then can be executed from the build directory by typing

- dmcs -out:runme ∗.cs
- mono runme

4.4 Coding Rules

We are using astyle (http://astyle.sourceforge.net/) to enforce a coding style for the library. To install astyle under Unix or Mac use your favorite package manager. For Windows download the binary from the astyle website. The style file libShapeOp.astyle can be found in the libShapeOp/astyle folder along with a bash script run_astyle.sh to run astyle with libShapeOp coding style.

Chapter 5

A Tutorial on Adding an Orientation Constraint to ShapeOp

This tutorial will require to rebuild the library. See the ShapeOp Documentation for detailed instructions and requirements Compilation and Installation.

A constraint projection of a given shape is defined as the shape that satisfies the constraint and is closest to the given shape in the least-squares sense. In this tutorial we will implement a orientation constraint. A orientation constraint acts on a set of 3 dimensional point and is satisfied if those points all lie on a plane with a prescribed orientation.

5.1 Mean-centering

Since this constraint only concerns the relative position of points, we can first subtract the mean of all points, project them onto the closest shape satisfying the constraint, and then formulate the global step of ShapeOp with respect to the mean. The global step solves a linear system in the least-square sense. The computation of the mean of a set of given points is also linear, and can therefore be implemented in the linear system.

5.2 Projection

The orientation of a plane can be defined by a normal n. After subtracting the mean the least-square fitting plane with normal n contains the origin. The projection of a given point x onto that plane is given by $p(x) = x - n(n \cdot x)$, where $n \cdot x$ denotes the dot-product of n and x.

5.3 C++ Implementation

All constraints of ShapeOp are implemented in Constraint.h and Constaint.cpp. We therefore add the orientation constraint to those files. In the header file we add the declaration of our class inheriting from the abstract class ShapeOp::Constraint:

```
class SHAPEOP_API OrientationConstraint : public Constraint {
private:
  Vector3 normal_;
  mutable Matrix3X input;
};
```

The private member input is used in the projection function. It ensures that memory is allocated at the construction of the OrientationConstraint object. This has a big impact execution speed, since ShapeOp uses parallelization, while memory allocation is inherently sequential.

The three functions following the constructor are inherited from the parent ShapeOp::Constraint class:

```
public:
  OrientationConstraint(const std::vector<int> &idI,
                        Scalar weight,
                        const Matrix3X &positions,
                        Vector3 normal = Vector3(0.0,0.0,1.0));
  virtual ~OrientationConstraint() {}
  virtual void project(const Matrix3X & positions, Matrix3X &projections) const override
      final;
  virtual void addConstraint(std::vector<Triplet> &triplets, int &idO) const override final;
  void setOrientation(const Vector3 &normal);
};
```

The last function allows to set a new orientation, which is specific to this constraint. In Constraint.cpp we add the implementation of our OrientationConstraint class. Lets start with the constructor:

```
SHAPEOP_INLINE OrientationConstraint::OrientationConstraint(const std::vector<int> &idI,
                                                            Scalar weight,
                                                            const Matrix3X &positions,
                                                            Vector3 normal /*= Vector3(0.0,0.0,1.0)*/):
  Constraint(idI,weight){
  assert(idI.size() >= 1);
  setOrientation(normal);

  //Allocate memory for intermediate storage during projection
  input = Matrix3X::Zero(3, idI.size());
}
```

The macro SHAPEOP_INLINE allows to inline a function and if applied to all build the whole library inline as a header-only library. The arguments to the constructor are documented in the header file. The setOrientation function is very short:

```
SHAPEOP_INLINE void OrientationConstraint::setOrientation(const
      Vector3 &normal){
  normal_ = normal;
  normal_.normalize();
}
```

We now add the projection function:

```
SHAPEOP_INLINE void OrientationConstraint::project(const Matrix3X & positions,
      Matrix3X &projections) const {

  //Copy the constrained positions to input
  for (int i = 0; i < static_cast<int>(idI_.size()); ++i) input.col(i) = positions.col(idI_[i]);

  //Compute and subtract mean
  Vector3 mean_vector = input.rowwise().mean();
  input.colwise() -= mean_vector;

  //Project each position onto the plane through zero defined by normal_
  for (int i = 0; i < static_cast<int>(idI_.size()); ++i){
    projections.col(idO_+i) = (input.col(i) - normal_*(normal_.dot(input.col(i))))*weight_;
  }
}
```

ShapeOp uses the Eigen library for linear algebra. Please refer to their documentation for details. Note that the Matrix projections is a global Matrix for all projections in a problem. Each constraint therefore stores an integer id0_ to know where in the global layout of constraints he is. id0_ is stored in the function addConstraint which the solver calls on each constraint so that he adds himself to the linear system used for the global solve in ShapeOp:

```
SHAPEOP_INLINE void OrientationConstraint::addConstraint(std::vector<Triplet> &triplets, int
      &idO) const {

  //Store this constraints position in the global linear system
  idO_ = idO;

  //Precompute coefficients for mean-centering
  int n_idx = static_cast<int>(idI_.size());
  double coef1 = (1.0 - 1.0 / n_idx) * weight_;
  double coef2 = -weight_ / n_idx;

  //Add triplets to the sparse global linear system.
```

```
  for (int i = 0; i < n_idx; ++i) {
    for (int j = 0; j < n_idx; ++j)
      //Add the coefficent for mean-centering to the sparse linear system at column id0 and row idI_[j].
      triplets.push_back(Triplet(id0, idI_[j], (i == j ? coef1 : coef2)));
    id0++;
  }
}
```

This function sets up part of the global linear system matrix so that if multiplied with the vector of x-coordinates it yields the mean-centered x-coordinate for the corresponding points. The same holds for y- and z-coordinates. The right-hand side of the system is the Matrix projections we saw in the projection constraint. ShapeOp then solves the system - in a least-square sense if it is over-constrained. All code we have added so far yields a working version of our orientation constraint in the C++ part of our ShapeOp library. To support the factory pattern, we add the following line before the return in the static function Constraint::shapeConstraintFactory:

```
if (constraintType.compare("Orientation") == 0){ if (n <  1) { return c; } return
      std::make_shared<OrientationConstraint>(idI, weight, positions); }
```

5.4 C API:

Most interlingual-bindings require C interfaces. ShapeOp has a C interface in the files API.h and API.cpp. Our orientation constraint is already included by the last line of code we added to the factory. To enable editing the orientation normal, we add the following code to the implementation of the function shapeop_editConstraint in A↩ PI.cpp:

```
if (strcmp(constraintType, "Orientation") == 0) {

  //Trying to cast the constraint given by the id.
  auto c = std::dynamic_pointer_cast<ShapeOp::OrientationConstraint>(op->s->getConstraint(constraint_id));
  if (!c) { return SO_UNMATCHING_CONSTRAINT_ID; }

  //Ensure correct number of scalars provided.
  if (nb_scl != 3) { return SO_INVALID_ARGUMENT_LENGTH; }

  //Wrap the C-style array of Scalars into a Vector3 and setting the new orientation
  Eigen::Map<const ShapeOp::Vector3> n(scalars, 3, 1);
  c->setOrientation(n);

  return SO_SUCCESS;
}
```

Now the library needs to be rebuilt. To do so, follow the compilation documentation of ShapeOp Compilation and Installation.

5.5 Grasshopper:

The Grasshopper integration of ShapeOp uses Ironpythons c-types to call our library. First make sure the library loads properly by following the documentation at ShapeOp for Grasshopper(Rhino). Copy the freshly built library to the folder Rhino is loading it from. After loading one of our examples files and running it successfully, the only thing we need to add to use our constraint is an additional option in the Value List component which feeds into the SOCSig Component. To do so, double-click the Value List component and add the following line:

```
Orientation           = "Orientation"
```

This string is then forwarded without further changes to the SOCSig component, which bundles all parameters for a set of constraints into Constraint-Signatures. The SOSolver component uses the string when adding a new constraint with a call to the dll, where the C API will internally call ShapeOp::Constraint::shapeConstraintFactory where the string ultimately gets resolved.

Chapter 6

Namespace Index

6.1 Namespace List

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

ShapeOp
Namespace of the ShapeOp library 21

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

ShapeOp::Constraint . . . 32
ShapeOp::AngleConstraint . . . 25
ShapeOp::AreaConstraint . . . 27
ShapeOp::BendingConstraint . . . 28
ShapeOp::CircleConstraint . . . 30
ShapeOp::ClosenessConstraint . . . 31
ShapeOp::EdgeStrainConstraint . . . 36
ShapeOp::LineConstraint . . . 38
ShapeOp::ParallelogramConstraint . . . 40
ShapeOp::PlaneConstraint . . . 41
ShapeOp::RectangleConstraint . . . 42
ShapeOp::SimilarityConstraint . . . 43
ShapeOp::SphereConstraint . . . 48
ShapeOp::TetrahedronStrainConstraint . . . 49
ShapeOp::TriangleStrainConstraint . . . 50
ShapeOp::UniformLaplacianConstraint . . . 51
ShapeOp::VolumeConstraint . . . 52
ShapeOp::Force . . . 37
ShapeOp::GravityForce . . . 37
ShapeOp::VertexForce . . . 51
ShapeOp::LSSolver . . . 39
ShapeOp::CGSolver . . . 29
ShapeOp::MINRESSolver . . . 39
ShapeOp::SimplicialLDLTSolver . . . 44
ShapeOp::SORSolver . . . 46
ShapeOpSolver . . . 43
ShapeOp::Solver . . . 45

Chapter 8

Class Index

8.1 Class List

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

ShapeOp::AngleConstraint
Angle range constraint. Constrains the angle between the two lines formed by three points to a range. See [3] for more details . . . 25
ShapeOp::AreaConstraint
Area constraint. Limits the area of a triangle to a range. See [2] for more details . . . 27
ShapeOp::BendingConstraint
Bending constraint. Limits the bending between two neighboring triangles. See [2] for more details . . . 28
ShapeOp::CGSolver
Sparse linear system solver based on CG. This class implements a sparse linear system solver based on the CG algorithm from Eigen . . . 29
ShapeOp::CircleConstraint
Circle constraint. Constrains a set of vertices to lie on the same circle. See [1] for more details 30
ShapeOp::ClosenessConstraint
Closeness constraint. Constrains a vertex to a position in space. Projects onto a given rest position . . . 31
ShapeOp::Constraint
Base class of any constraints. This class defines the interface of a ShapeOp constraint . . . 32
ShapeOp::EdgeStrainConstraint
Edge strain constraint. Constrains the distance between two points to a range. See [2] for more details . . . 36
ShapeOp::Force
Base class of any forces. This class defines interface of a ShapeOp force . . . 37
ShapeOp::GravityForce
This class defines a constant force for all vertices . . . 37
ShapeOp::LineConstraint
Line constraint. Constrains a set of vertices to lie on the same line. See [1] for more details . . 38
ShapeOp::LSSolver
Base class of any sparse linear system solver. This class defines the main functionalities of the ShapeOp sparse linear system solvers (Ax = b) . . . 39
ShapeOp::MINRESSolver
Sparse linear system solver based on MINRES. This class implements a sparse linear system solver based on the MINRES algorithm from Eigen . . . 39
ShapeOp::ParallelogramConstraint
ParallelogramConstraint constraint. Constrains the four vertices of a quad to be a parallelogram. See [1] for more details . . . 40
ShapeOp::PlaneConstraint
Plane constraint. Constrains a set of vertices to lie on the same plane. See [1] for more details 41

ShapeOp::RectangleConstraint
Rectangle constraint. Constrains the four vertices of a quad to be a rectangle. See [1] for more details . . . 42
ShapeOpSolver
C structure that containts the C++ ShapeOp solver . . . 43
ShapeOp::SimilarityConstraint
Similarity or Rigid constraint. Perserves the relative location of a set of vertices. See [1] for more details . . . 43
ShapeOp::SimplicialLDLTSolver
Sparse linear system solver based on Cholesky. This class implements a sparse linear system solver based on the Cholesky LDL^T algorithm from Eigen . . . 44
ShapeOp::Solver
ShapeOp Solver. This class implements the main ShapeOp solver based on [1] and [2] . . . 45
ShapeOp::SORSolver
Sparse linear system solver based on successive over-relaxation (SOR) . . . 46
ShapeOp::SphereConstraint
Sphere constraint. Constrains a set of vertices to lie on a sphere. See [1] for more details . . . 48
ShapeOp::TetrahedronStrainConstraint
A mesh-independent tetrahedron strain-limiting constraint. See [2] for more details . . . 49
ShapeOp::TriangleStrainConstraint
A mesh-independent triangle strain-limiting constraint. See [2] for more details . . . 50
ShapeOp::UniformLaplacianConstraint
Uniform Laplacian Constraint. Minimizes the uniform laplacian respectively the uniform laplacian of displacements . . . 51
ShapeOp::VertexForce
This class defines a constant force for a unique vertex . . . 51
ShapeOp::VolumeConstraint
Volume constraint. Limits the volume of a tetrahedron to a range. See [2] for more details . . . 52

Chapter 9

File Index

9.1 File List

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

libShapeOp/api/API.h 55
libShapeOp/src/Common.h 60
libShapeOp/src/Constraint.h 60
libShapeOp/src/Force.h 61
libShapeOp/src/LSSolver.h 62
libShapeOp/src/Solver.h 63
libShapeOp/src/Types.h 63

Chapter 10

Namespace Documentation

10.1 ShapeOp Namespace Reference

Namespace of the ShapeOp library.

Classes

- class AngleConstraint

  Angle range constraint. Constrains the angle between the two lines formed by three points to a range. See [3] for more details.
- class AreaConstraint

  Area constraint. Limits the area of a triangle to a range. See [2] for more details.
- class BendingConstraint

  Bending constraint. Limits the bending between two neighboring triangles. See [2] for more details.
- class CGSolver

  Sparse linear system solver based on CG. This class implements a sparse linear system solver based on the CG algorithm from Eigen.
- class CircleConstraint

  Circle constraint. Constrains a set of vertices to lie on the same circle. See [1] for more details.
- class ClosenessConstraint

  Closeness constraint. Constrains a vertex to a position in space. Projects onto a given rest position.
- class Constraint

  Base class of any constraints. This class defines the interface of a ShapeOp constraint.
- class EdgeStrainConstraint

  Edge strain constraint. Constrains the distance between two points to a range. See [2] for more details.
- class Force

  Base class of any forces. This class defines interface of a ShapeOp force.
- class GravityForce

  This class defines a constant force for all vertices.
- class LineConstraint

  Line constraint. Constrains a set of vertices to lie on the same line. See [1] for more details.
- class LSSolver

  Base class of any sparse linear system solver. This class defines the main functionalities of the ShapeOp sparse linear system solvers (Ax = b).
- class MINRESSolver

  Sparse linear system solver based on MINRES. This class implements a sparse linear system solver based on the MINRES algorithm from Eigen.
- class ParallelogramConstraint

ParallelogramConstraint constraint. Constrains the four vertices of a quad to be a parallelogram. See [1] for more details.

- class PlaneConstraint

  Plane constraint. Constrains a set of vertices to lie on the same plane. See [1] for more details.
- class RectangleConstraint

  Rectangle constraint. Constrains the four vertices of a quad to be a rectangle. See [1] for more details.
- class SimilarityConstraint

  Similarity or Rigid constraint. Perserves the relative location of a set of vertices. See [1] for more details.
- class SimplicialLDLTSolver

  Sparse linear system solver based on Cholesky. This class implements a sparse linear system solver based on the Cholesky LDL^T algorithm from Eigen.
- class Solver

  ShapeOp Solver. This class implements the main ShapeOp solver based on [1] and [2].
- class SORSolver

  Sparse linear system solver based on successive over-relaxation (SOR).
- class SphereConstraint

  Sphere constraint. Constrains a set of vertices to lie on a sphere. See [1] for more details.
- class TetrahedronStrainConstraint

  A mesh-independent tetrahedron strain-limiting constraint. See [2] for more details.
- class TriangleStrainConstraint

  A mesh-independent triangle strain-limiting constraint. See [2] for more details.
- class UniformLaplacianConstraint

  Uniform Laplacian Constraint. Minimizes the uniform laplacian respectively the uniform laplacian of displacements.
- class VertexForce

  This class defines a constant force for a unique vertex.
- class VolumeConstraint

  Volume constraint. Limits the volume of a tetrahedron to a range. See [2] for more details.

Typedefs

- typedef ShapeOpScalar Scalar

  A scalar type, double or float, as defined in ShapeOpScalar in Common.h.
- template<int Rows, int Cols, int Options = (Eigen::ColMajor | SHAPEOP_ALIGNMENT)>
  using MatrixT = Eigen::Matrix< Scalar, Rows, Cols, Options >

  A typedef of the dense matrix of Eigen.
- typedef MatrixT< 2, 1 > Vector2

  A 2d column vector.
- typedef MatrixT< 2, 2 > Matrix22

  A 2 by 2 matrix.
- typedef MatrixT< 2, 3 > Matrix23

  A 2 by 3 matrix.
- typedef MatrixT< 3, 1 > Vector3

  A 3d column vector.
- typedef MatrixT< 3, 2 > Matrix32

  A 3 by 2 matrix.
- typedef MatrixT< 3, 3 > Matrix33

  A 3 by 3 matrix.
- typedef MatrixT< 3, 4 > Matrix34

  A 3 by 4 matrix.
- typedef MatrixT< 4, 1 > Vector4

  A 4d column vector.

- typedef MatrixT< 4, 4 > Matrix44

  A 4 by 4 matrix.
- typedef MatrixT< 3, Eigen::Dynamic > Matrix3X

  A 3 by n matrix.
- typedef MatrixT< Eigen::Dynamic, 3 > MatrixX3

  A n by 3 matrix.
- typedef MatrixT< Eigen::Dynamic, 1 > VectorX

  A nd column vector.
- typedef MatrixT< Eigen::Dynamic, Eigen::Dynamic > MatrixXX

  A n by m matrix.
- template<int Options = Eigen::ColMajor>
  using SparseMatrixT = Eigen::SparseMatrix< Scalar, Options >

  A typedef of the sparse matrix of Eigen.
- typedef SparseMatrixT SparseMatrix

  The default sparse matrix of Eigen.
- typedef Eigen::Triplet< Scalar > Triplet

  A triplet, used in the sparse triplet representation for matrices.

Functions

- SHAPEOP_INLINE Scalar clamp (Scalar v, Scalar vMin, Scalar vMax)

  Clamps v to lie between vMin and vMax.

10.1.1 Detailed Description

Namespace of the ShapeOp library.

Chapter 11

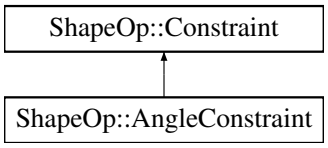
Class Documentation

11.1 ShapeOp::AngleConstraint Class Reference

Angle range constraint. Constrains the angle between the two lines formed by three points to a range. See [3] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::AngleConstraint:



Public Member Functions

- AngleConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar min↩ Angle=0.0, Scalar maxAngle=M_PI)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the angle constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setMinAngle (Scalar minAngle)

  Set a new minimum angle (in radian).
- void setMaxAngle (Scalar maxAngle)

  Set a new maximum angle (in radian).

Additional Inherited Members

11.1.1 Detailed Description

Angle range constraint. Constrains the angle between the two lines formed by three points to a range. See [3] for more details.

11.1.2 Constructor & Destructor Documentation

11.1.2.1 SHAPEOP_INLINE ShapeOp::AngleConstraint::AngleConstraint (const std::vector< int > & *idl*, Scalar *weight*, const Matrix3X & *positions*, Scalar *minAngle* = 0.0, Scalar *maxAngle* = M_PI)

Constraint constructor.

Parameters

idI	are three indices of the vertices [v0,v1,v2]. The constrained angle is the one spanned between v1-v0 and v2-v0:
	`| id0 |`
	`| / | |`
	`| / | |`
	`| id1 id2 |`

weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
minAngle	The minimum angle (in radian, between 0 and PI).
maxAngle	The maximum angle (in radian, between 0 and PI).

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

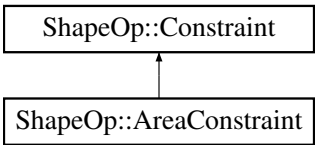
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.2 ShapeOp::AreaConstraint Class Reference

Area constraint. Limits the area of a triangle to a range. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::AreaConstraint:



Public Member Functions

- AreaConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar rangeMin=1.0, Scalar rangeMax=1.0)

  Constraint constructor. The target area is the area spanned by three vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the area [rangeMin∗ target_area,rangeMax∗ target_area].
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setRangeMin (Scalar rMin)

  Set a new range minimum.
- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.2.1 Detailed Description

Area constraint. Limits the area of a triangle to a range. See [2] for more details.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 SHAPEOP_INLINE ShapeOp::AreaConstraint::AreaConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = 1.0, Scalar *rangeMax* = 1.0)

Constraint constructor. The target area is the area spanned by three vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the area [rangeMin∗ target_area,rangeMax∗ target_area].

Parameters

idI	are three indices of the vertices forming the constrained triangle
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The factor to determine the minimal area: rangeMin∗target_area.
rangeMax	The factor to determine the maximal area: rangeMax∗target_area.

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

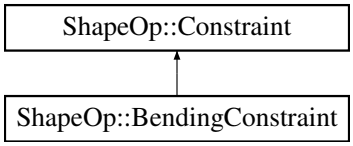
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.3 ShapeOp::BendingConstraint Class Reference

Bending constraint. Limits the bending between two neighboring triangles. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::BendingConstraint:



Public Member Functions

- BendingConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar rangeMin=1.0, Scalar rangeMax=1.0)

  Constraint constructor. The target bend is set to the bend between the triangles spanned by four vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the bend [rangeMin∗target_bend,rangeMax∗target_bend] The bending constraint applies to two neighboring triangles sharing an edge.

- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.

- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

- void setRangeMin (Scalar rMin)

  Set a new range minimum.

- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.3.1 Detailed Description

Bending constraint. Limits the bending between two neighboring triangles. See [2] for more details.

11.3.2 Constructor & Destructor Documentation

11.3.2.1 SHAPEOP_INLINE ShapeOp::BendingConstraint::BendingConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = 1.0, Scalar *rangeMax* = 1.0)

Constraint constructor. The target bend is set to the bend between the triangles spanned by four vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the bend [rangeMin∗target_bend,rangeMax∗target_bend] The bending constraint applies to two neighboring triangles sharing an edge.

Parameters

idI	are four indices of the vertices of the two triangles ordered as follows:
	
	`|  id2  |`
	`|  /  \  |`
	`|  id0---id1  |`
	`|  \  /  |`
	`|  id3  |`

idI	are four indices of the vertices of the tetrahedron
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The factor to determine the minimal bend: rangeMin∗target_bend.
rangeMax	The factor to determine the maximal bend: rangeMax∗target_bend.

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

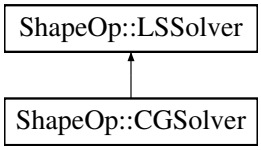
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.4 ShapeOp::CGSolver Class Reference

Sparse linear system solver based on CG. This class implements a sparse linear system solver based on the CG algorithm from Eigen.

`#include <LSSolver.h>`

Inheritance diagram for ShapeOp::CGSolver:



Public Member Functions

- virtual void initialize (const SparseMatrix &A, unsigned int iteration) overridefinal

Initialize PCG.

- virtual VectorX solve (const VectorX &b, const VectorX &x0) const overridefinal

  Solve the linear system by applying CG.

- virtual Eigen::ComputationInfo info () const overridefinal

  Reports whether previous computation was successful.

11.4.1 Detailed Description

Sparse linear system solver based on CG. This class implements a sparse linear system solver based on the CG algorithm from Eigen.

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

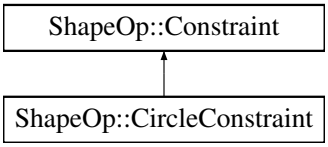
- libShapeOp/src/LSSolver.h
- libShapeOp/src/LSSolver.cpp

11.5 ShapeOp::CircleConstraint Class Reference

Circle constraint. Constrains a set of vertices to lie on the same circle. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::CircleConstraint:



Public Member Functions

- CircleConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor.

- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.

- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.5.1 Detailed Description

Circle constraint. Constrains a set of vertices to lie on the same circle. See [1] for more details.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 SHAPEOP_INLINE ShapeOp::CircleConstraint::CircleConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a circle.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

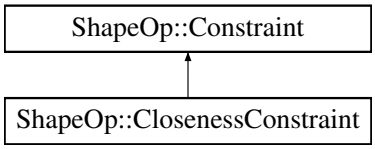
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.6 ShapeOp::ClosenessConstraint Class Reference

Closeness constraint. Constrains a vertex to a position in space. Projects onto a given rest position.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::ClosenessConstraint:



Public Member Functions

- ClosenessConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor. The target is the position of vertex idI in the initial positions given in the constructor.
- virtual void project (const Matrix3X &, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setPosition (const Vector3 &position)

  Set a new closeness position.
- Vector3 getPosition () const

  Get the closeness position.

Additional Inherited Members

11.6.1 Detailed Description

Closeness constraint. Constrains a vertex to a position in space. Projects onto a given rest position.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 SHAPEOP_INLINE ShapeOp::ClosenessConstraint::ClosenessConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor. The target is the position of vertex idI in the initial positions given in the constructor.

Parameters

idI	contains the one index of the vertex that wants to be close to a given target.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

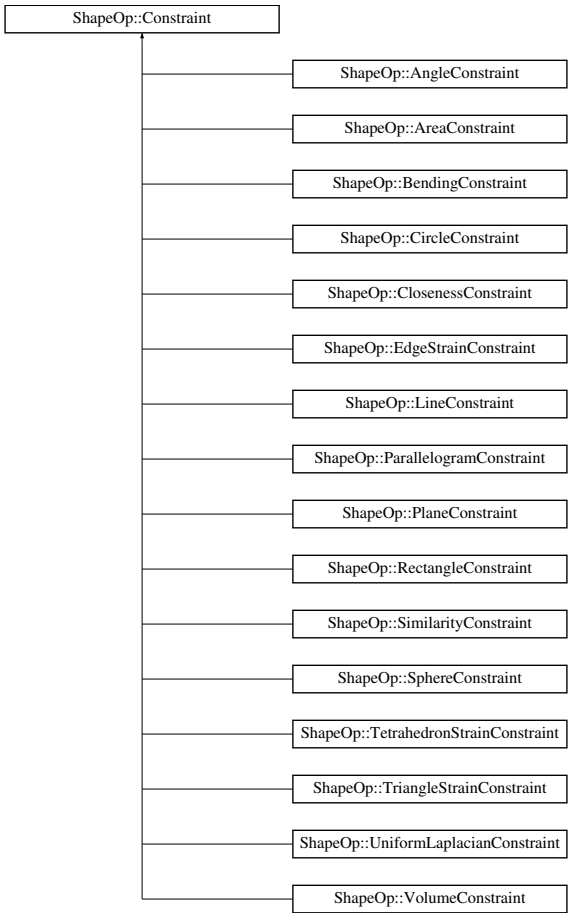
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.7 ShapeOp::Constraint Class Reference

Base class of any constraints. This class defines the interface of a ShapeOp constraint.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::Constraint:



Public Member Functions

- Constraint (const std::vector< int > &idI, Scalar weight)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const =0

  Find the closest, in the least-squre sense, configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const =0

Add the constraint to the linear system.

- std::size_t nIndices () const

  Number of indices of vertices involved in the constraint.

Static Public Member Functions

- static std::shared_ptr< Constraint > shapeConstraintFactory (const std::string &ConstraintType, const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Creates a constraint from a string type, a number of indices, a weight and the initial point positions.

Protected Attributes

- std::vector< int > idI_

  ids of the vertices involved in this constraint.

- Scalar weight_

  weight for the constraint.

- int idO_

  location of this constraint in the linear system.

11.7.1 Detailed Description

Base class of any constraints. This class defines the interface of a ShapeOp constraint.

11.7.2 Constructor & Destructor Documentation

11.7.2.1 SHAPEOP_INLINE ShapeOp::Constraint::Constraint (const std::vector< int > & *idI*, Scalar *weight*)

Constraint constructor.

Parameters

idI	A vector of indices of the vertices to be constrained.
weight	The weight of the constraint to be added relative to the other constraints.

11.7.3 Member Function Documentation

11.7.3.1 virtual void ShapeOp::Constraint::addConstraint (std::vector< Triplet > & *triplets*, int & *idO*) const [pure virtual]

Add the constraint to the linear system.

Parameters

out	*triplets*	A vector of triplets each representing an entry in a sparse matrix.
in,out	*idO*	In: The first row index of the constraint in the sparse matrix. Out: The last row index plus one.

Implemented in ShapeOp::AngleConstraint, ShapeOp::UniformLaplacianConstraint, ShapeOp::ParallelogramConstraint, ShapeOp::RectangleConstraint, ShapeOp::SimilarityConstraint, ShapeOp::SphereConstraint, ShapeOp::CircleConstraint, ShapeOp::PlaneConstraint, ShapeOp::LineConstraint, ShapeOp::ClosenessConstraint, ShapeOp::BendingConstraint, ShapeOp::VolumeConstraint, ShapeOp::AreaConstraint, ShapeOp::TetrahedronStrainConstraint, ShapeOp::TriangleStrainConstraint, and ShapeOp::EdgeStrainConstraint.

11.7.3.2 virtual void ShapeOp::Constraint::project (const Matrix3X & *positions,* Matrix3X & *projections*) const [pure virtual]

Find the closest, in the least-squre sense, configuration from the input positions that satisfy the constraint.

Parameters

positions	The positions of all the n vertices stacked in a 3 by n matrix.
projections	The projections of the vertices involved in the constraint.

Implemented in ShapeOp::AngleConstraint, ShapeOp::UniformLaplacianConstraint, ShapeOp::Parallelogram↩Constraint, ShapeOp::RectangleConstraint, ShapeOp::SimilarityConstraint, ShapeOp::SphereConstraint, Shape↩Op::CircleConstraint, ShapeOp::PlaneConstraint, ShapeOp::LineConstraint, ShapeOp::ClosenessConstraint, ShapeOp::BendingConstraint, ShapeOp::VolumeConstraint, ShapeOp::AreaConstraint, ShapeOp::Tetrahedron↩StrainConstraint, ShapeOp::TriangleStrainConstraint, and ShapeOp::EdgeStrainConstraint.

11.7.3.3 SHAPEOP_INLINE std::shared_ptr< Constraint > ShapeOp::Constraint::shapeConstraintFactory (const std::string & *ConstraintType,* const std::vector< int > & *idl,* Scalar *weight,* const Matrix3X & *positions*) [static]

Creates a constraint from a string type, a number of indices, a weight and the initial point positions.

Parameters

ConstraintType	One of the following: • "EdgeStrain", see ShapeOp::EdgeStrainConstraint • "TriangleStrain", see ShapeOp::TriangleStrainConstraint • "TetrahedronStrain", see ShapeOp::TetrahedronStrainConstraint • "Area", see ShapeOp::AreaConstraint • "Volume", see ShapeOp::VolumeConstraint • "Bending", see ShapeOp::BendingConstraint • "Closeness", see ShapeOp::ClosenessConstraint • "Line", see ShapeOp::LineConstraint • "Plane", see ShapeOp::PlaneConstraint • "Circle", see ShapeOp::CircleConstraint • "Sphere", see ShapeOp::SphereConstraint • "Similarity", see ShapeOp::SimilarityConstraint with scaling • "Rigid", see ShapeOp::SimilarityConstraint without scaling • "Rectangle", see ShapeOp::RectangleConstraint • "Parallelogram", see ShapeOp::ParallelogramConstraint • "Laplacian", see ShapeOp::UniformLaplacianConstraint • "LaplacianDisplacement", see ShapeOp::UniformLaplacianConstraint of deformation

idI	A vector of indices of the vertices to be constrained.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

Returns

A std::shared pointer to the Constraint, which is empty/null if failed.

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

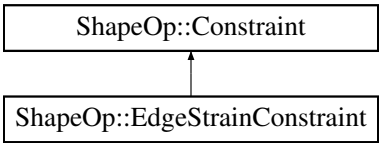
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.8 ShapeOp::EdgeStrainConstraint Class Reference

Edge strain constraint. Constrains the distance between two points to a range. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::EdgeStrainConstraint:



Public Member Functions

- EdgeStrainConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar rangeMin=1.0, Scalar rangeMax=1.0)

  Constraint constructor. The target length is set to the distance of the two vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the distance (equivalent to edge length) [rangeMin∗distance,rangeMax∗distance].
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setEdgeLength (Scalar length)

  Set a new edge length.
- void setRangeMin (Scalar rMin)

  Set a new range minimum.
- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.8.1 Detailed Description

Edge strain constraint. Constrains the distance between two points to a range. See [2] for more details.

11.8.2 Constructor & Destructor Documentation

11.8.2.1 SHAPEOP_INLINE ShapeOp::EdgeStrainConstraint::EdgeStrainConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = 1.0, Scalar *rangeMax* = 1.0)

Constraint constructor. The target length is set to the distance of the two vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the distance (equivalent to edge length) [rangeMin∗distance,rangeMax∗distance].

Parameters

idI	are two indices of the vertices of the edge.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The factor to determine the minimal distance from the target length: rangeMin∗distance
rangeMax	The factor to determine the maximal distance from the target length: rangeMax∗distance

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

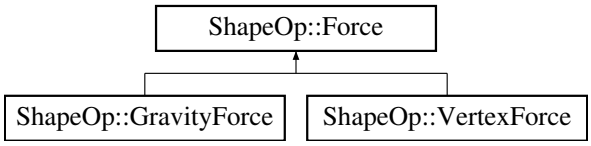
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.9 ShapeOp::Force Class Reference

Base class of any forces. This class defines interface of a ShapeOp force.

`#include <Force.h>`

Inheritance diagram for ShapeOp::Force:



Public Member Functions

- virtual Vector3 get (const Matrix3X &positions, int id) const =0

  Get force vector.

11.9.1 Detailed Description

Base class of any forces. This class defines interface of a ShapeOp force.

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

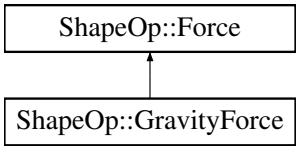
- libShapeOp/src/Force.h

11.10 ShapeOp::GravityForce Class Reference

This class defines a constant force for all vertices.

`#include <Force.h>`

Inheritance diagram for ShapeOp::GravityForce:



Public Member Functions

- GravityForce (const Vector3 &f)

  Constructor taking the gravity vector as parameter.
- virtual Vector3 get (const Matrix3X &, int) const overridefinal

  Get gravity vector.

11.10.1 Detailed Description

This class defines a constant force for all vertices.

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

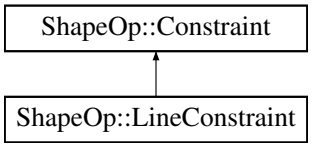
- libShapeOp/src/Force.h
- libShapeOp/src/Force.cpp

11.11 ShapeOp::LineConstraint Class Reference

Line constraint. Constrains a set of vertices to lie on the same line. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::LineConstraint:



Public Member Functions

- LineConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.11.1 Detailed Description

Line constraint. Constrains a set of vertices to lie on the same line. See [1] for more details.

11.11.2 Constructor & Destructor Documentation

11.11.2.1 SHAPEOP_INLINE ShapeOp::LineConstraint::LineConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a line.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

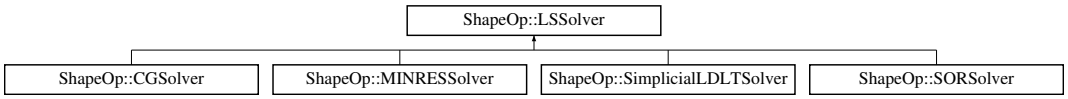
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.12 ShapeOp::LSSolver Class Reference

Base class of any sparse linear system solver. This class defines the main functionalities of the ShapeOp sparse linear system solvers (Ax = b).

`#include <LSSolver.h>`

Inheritance diagram for ShapeOp::LSSolver:



Public Member Functions

- virtual void initialize (const SparseMatrix &A, unsigned int iteration=1)=0

  Initialize the linear system solver using the sparse matrix A.
- virtual VectorX solve (const VectorX &b, const VectorX &x0) const =0

  Solve the linear system Ax = b.
- virtual Eigen::ComputationInfo info () const =0

  Reports whether previous computation was successful.

11.12.1 Detailed Description

Base class of any sparse linear system solver. This class defines the main functionalities of the ShapeOp sparse linear system solvers (Ax = b).

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

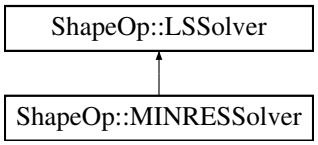
- libShapeOp/src/LSSolver.h

11.13 ShapeOp::MINRESSolver Class Reference

Sparse linear system solver based on MINRES. This class implements a sparse linear system solver based on the MINRES algorithm from Eigen.

`#include <LSSolver.h>`

Inheritance diagram for ShapeOp::MINRESSolver:



Public Member Functions

- virtual void initialize (const SparseMatrix &A, unsigned int iteration) overridefinal

  Initialize MINRES.
- virtual VectorX solve (const VectorX &b, const VectorX &x0) const overridefinal

  Solve the linear system by applying MINRES.
- virtual Eigen::ComputationInfo info () const overridefinal

  Reports whether previous computation was successful.

11.13.1 Detailed Description

Sparse linear system solver based on MINRES. This class implements a sparse linear system solver based on the MINRES algorithm from Eigen.

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

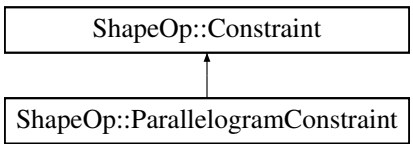
- libShapeOp/src/LSSolver.h
- libShapeOp/src/LSSolver.cpp

11.14 ShapeOp::ParallelogramConstraint Class Reference

ParallelogramConstraint constraint. Constrains the four vertices of a quad to be a parallelogram. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::ParallelogramConstraint:



Public Member Functions

- ParallelogramConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.14.1 Detailed Description

ParallelogramConstraint constraint. Constrains the four vertices of a quad to be a parallelogram. See [1] for more details.

11.14.2 Constructor & Destructor Documentation

11.14.2.1 SHAPEOP_INLINE ShapeOp::ParallelogramConstraint::ParallelogramConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a sphere.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

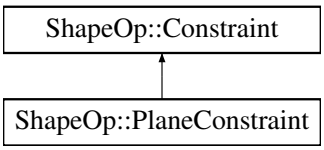
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.15 ShapeOp::PlaneConstraint Class Reference

Plane constraint. Constrains a set of vertices to lie on the same plane. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::PlaneConstraint:



Public Member Functions

- PlaneConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.15.1 Detailed Description

Plane constraint. Constrains a set of vertices to lie on the same plane. See [1] for more details.

11.15.2 Constructor & Destructor Documentation

11.15.2.1 SHAPEOP_INLINE ShapeOp::PlaneConstraint::PlaneConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a plane.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

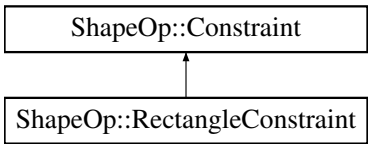
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.16 ShapeOp::RectangleConstraint Class Reference

Rectangle constraint. Constrains the four vertices of a quad to be a rectangle. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::RectangleConstraint:



Public Member Functions

- RectangleConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.16.1 Detailed Description

Rectangle constraint. Constrains the four vertices of a quad to be a rectangle. See [1] for more details.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 SHAPEOP_INLINE ShapeOp::RectangleConstraint::RectangleConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a sphere.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.17 ShapeOpSolver Struct Reference

C structure that containts the C++ ShapeOp solver.

Public Attributes

- std::shared_ptr< ShapeOp::Solver > s

  A shared pointer to a C++ ShapeOp solver.

11.17.1 Detailed Description

C structure that containts the C++ ShapeOp solver.

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

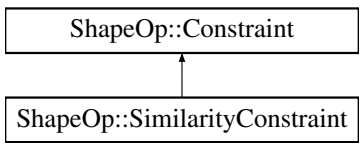
- libShapeOp/api/API.cpp

11.18 ShapeOp::SimilarityConstraint Class Reference

Similarity or Rigid constraint. Perserves the relative location of a set of vertices. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::SimilarityConstraint:



Public Member Functions

- SimilarityConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, bool scaling=true, bool rotate=true, bool flip=true)

  Constraint constructor. By default the shape to match is given by the initial positions of the vertices involved.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setShapes (const std::vector< Matrix3X > &shapes)

  A set of new shapes to match to. The SimilarityConstraint will choose and project onto the closest.

Additional Inherited Members

11.18.1 Detailed Description

Similarity or Rigid constraint. Perserves the relative location of a set of vertices. See [1] for more details.

11.18.2 Constructor & Destructor Documentation

11.18.2.1 SHAPEOP_INLINE ShapeOp::SimilarityConstraint::SimilarityConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, bool *scaling* = `true`, bool *rotate* = `true`, bool *flip* = `true`)

Constraint constructor. By default the shape to match is given by the initial positions of the vertices involved.

Parameters

idI	A vector of indices of the vertices to be constrained.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
scaling	If true, the matching is only up to scale, if false not.
rotate	If true the constraint minimizes over all possible rotations of the matchings.
flip	If true the constraint minimizes by also flipping the matchings.

11.18.3 Member Function Documentation

11.18.3.1 SHAPEOP_INLINE void ShapeOp::SimilarityConstraint::setShapes (const std::vector< Matrix3X > & *shapes*)

A set of new shapes to match to. The SimilarityConstraint will choose and project onto the closest.

Parameters

shapes	a vector of shapes. Each shape consists of a 3∗nIndices() Matrix representing a set of points.

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

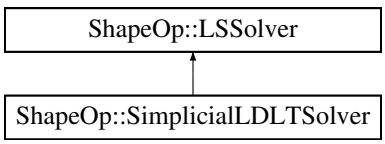
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.19 ShapeOp::SimplicialLDLTSolver Class Reference

Sparse linear system solver based on Cholesky. This class implements a sparse linear system solver based on the Cholesky LDL^T algorithm from Eigen.

`#include <LSSolver.h>`

Inheritance diagram for ShapeOp::SimplicialLDLTSolver:



Public Member Functions

- virtual void initialize (const SparseMatrix &A, unsigned int iteration) overridefinal

Prefactorize the sparse matrix (A = LDL^T).

- virtual VectorX solve (const VectorX &b, const VectorX &x0) const overridefinal

  Solve the linear system by applying twice backsubstitution.

- virtual Eigen::ComputationInfo info () const overridefinal

  Reports whether previous computation was successful.

11.19.1 Detailed Description

Sparse linear system solver based on Cholesky. This class implements a sparse linear system solver based on the Cholesky LDL^T algorithm from Eigen.

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

- libShapeOp/src/LSSolver.h
- libShapeOp/src/LSSolver.cpp

11.20 ShapeOp::Solver Class Reference

ShapeOp Solver. This class implements the main ShapeOp solver based on [1] and [2].

`#include <Solver.h>`

Public Member Functions

- int addConstraint (const std::shared_ptr< Constraint > &c)

  Add a constraint to the solver and get back its id.

- std::shared_ptr< Constraint > & getConstraint (int id)

  Get a constraint using its id.

- int addForces (const std::shared_ptr< Force > &f)

  Add a force to the solver and get back its id.

- std::shared_ptr< Force > & getForce (int id)

  Get a force using its id.

- void setPoints (const Matrix3X &p)

  Set the points.

- void setTimeStep (Scalar timestep)

  Set the timestep for the dynamics.

- void setDamping (Scalar damping)

  Set the velocity damping for the dynamics.

- const Matrix3X & getPoints ()

  Get the points.

- bool initialize (bool dynamic=false, Scalar masses=1.0, Scalar damping=1.0, Scalar timestep=1.0)

  Initialize the ShapeOp linear system and the different parameters.

- bool solve (unsigned int iteration)

  Solve the constraint problem by projecting and merging.

11.20.1 Detailed Description

ShapeOp Solver. This class implements the main ShapeOp solver based on [1] and [2].

11.20.2 Member Function Documentation

11.20.2.1 SHAPEOP_INLINE bool ShapeOp::Solver::initialize (bool *dynamic* = `false`, Scalar *masses* = `1.0`, Scalar *damping* = `1.0`, Scalar *timestep* = `1.0`)

Initialize the ShapeOp linear system and the different parameters.

Returns

true if successfull

11.20.2.2 SHAPEOP_INLINE bool ShapeOp::Solver::solve (unsigned int *iteration*)

Solve the constraint problem by projecting and merging.

Returns

true if successfull

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

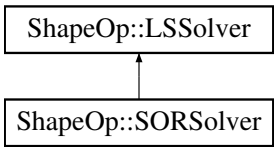
- libShapeOp/src/Solver.h
- libShapeOp/src/Solver.cpp

11.21 ShapeOp::SORSolver Class Reference

Sparse linear system solver based on successive over-relaxation (SOR).

`#include <LSSolver.h>`

Inheritance diagram for ShapeOp::SORSolver:



Public Member Functions

- SORSolver (ShapeOp::Scalar relaxation=1.6)
  Solver Constructor.
- virtual void initialize (const SparseMatrix &A, unsigned int iteration) overridefinal
  Initialize SOR.
- virtual VectorX solve (const VectorX &b, const VectorX &x0) const overridefinal
  Solve the linear system by applying SOR.
- virtual Eigen::ComputationInfo info () const overridefinal
  Reports whether previous computation was successful.

11.21.1 Detailed Description

Sparse linear system solver based on successive over-relaxation (SOR).

11.21.2 Constructor & Destructor Documentation

11.21.2.1 SHAPEOP_INLINE ShapeOp::SORSolver::SORSolver (ShapeOp::Scalar *relaxation* = 1.6)

Solver Constructor.

Parameters

relaxation	The relaxation factor.

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

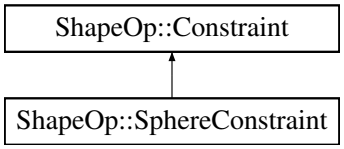
- libShapeOp/src/LSSolver.h
- libShapeOp/src/LSSolver.cpp

11.22 ShapeOp::SphereConstraint Class Reference

Sphere constraint. Constrains a set of vertices to lie on a sphere. See [1] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::SphereConstraint:



Public Member Functions

- SphereConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions)
  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal
  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal
  Add the constraint to the linear system.

Additional Inherited Members

11.22.1 Detailed Description

Sphere constraint. Constrains a set of vertices to lie on a sphere. See [1] for more details.

11.22.2 Constructor & Destructor Documentation

11.22.2.1 SHAPEOP_INLINE ShapeOp::SphereConstraint::SphereConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*)

Constraint constructor.

Parameters

idI	Indices of vertices to be projection onto on a sphere.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.

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

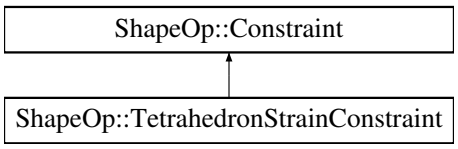
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.23 ShapeOp::TetrahedronStrainConstraint Class Reference

A mesh-independent tetrahedron strain-limiting constraint. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::TetrahedronStrainConstraint:



Public Member Functions

- TetrahedronStrainConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar rangeMin=1.0, Scalar rangeMax=1.0)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setRangeMin (Scalar rMin)

  Set a new range minimum.
- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.23.1 Detailed Description

A mesh-independent tetrahedron strain-limiting constraint. See [2] for more details.

11.23.2 Constructor & Destructor Documentation

11.23.2.1 SHAPEOP_INLINE ShapeOp::TetrahedronStrainConstraint::TetrahedronStrainConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = `1.0`, Scalar *rangeMax* = `1.0`)

Constraint constructor.

Parameters

idI	are four indices of the vertices of the tetrahedron
idI	are three indices of the vertices of the triangle
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The minimal strain.
rangeMax	The maximal strain.

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

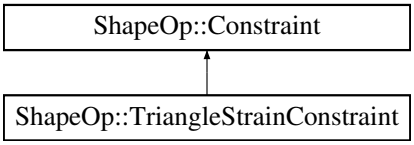
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.24 ShapeOp::TriangleStrainConstraint Class Reference

A mesh-independent triangle strain-limiting constraint. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::TriangleStrainConstraint:



Public Member Functions

- TriangleStrainConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar rangeMin=1.0, Scalar rangeMax=1.0)

  Constraint constructor.
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setRangeMin (Scalar rMin)

  Set a new range minimum.
- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.24.1 Detailed Description

A mesh-independent triangle strain-limiting constraint. See [2] for more details.

11.24.2 Constructor & Destructor Documentation

11.24.2.1 SHAPEOP_INLINE ShapeOp::TriangleStrainConstraint::TriangleStrainConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = `1.0`, Scalar *rangeMax* = `1.0`)

Constraint constructor.

Parameters

idI	are three indices of the vertices of the triangle
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The minimal strain.
rangeMax	The maximal strain.

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

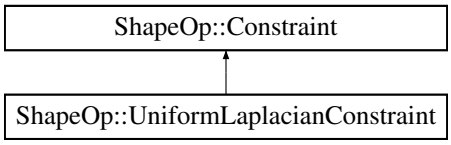
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.25 ShapeOp::UniformLaplacianConstraint Class Reference

Uniform Laplacian Constraint. Minimizes the uniform laplacian respectively the uniform laplacian of displacements.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::UniformLaplacianConstraint:



Public Member Functions

- UniformLaplacianConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, bool displacement_lap)

  Constraint constructor.
- virtual void project (const Matrix3X &, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.
- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.

Additional Inherited Members

11.25.1 Detailed Description

Uniform Laplacian Constraint. Minimizes the uniform laplacian respectively the uniform laplacian of displacements.

11.25.2 Constructor & Destructor Documentation

11.25.2.1 SHAPEOP_INLINE ShapeOp::UniformLaplacianConstraint::UniformLaplacianConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, bool *displacement_lap*)

Constraint constructor.

Parameters

idI	The first index corresponds to the central vertex, while the remaining ones correspond to the one-ring neighborhood.
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
displacement_lap	If false the laplacian is minimized, if true, the laplacian of deformations w.r.t. initial positions.

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

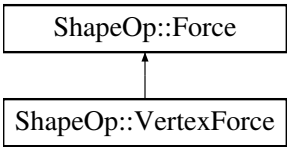
- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

11.26 ShapeOp::VertexForce Class Reference

This class defines a constant force for a unique vertex.

`#include <Force.h>`

Inheritance diagram for ShapeOp::VertexForce:



Public Member Functions

- VertexForce (const Vector3 &f=Vector3::Zero(), int id=-1)

  Constructor taking the force and the vertex id as parameters.
- virtual Vector3 get (const Matrix3X &, int id) const overridefinal

  Get force vector.
- void setId (int id)

  Set a new vertex id.
- void setForce (const Vector3 &f)

  Set a new force.

11.26.1 Detailed Description

This class defines a constant force for a unique vertex.

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

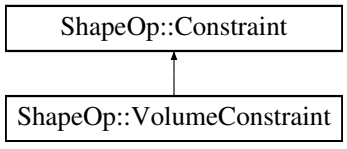
- libShapeOp/src/Force.h
- libShapeOp/src/Force.cpp

11.27 ShapeOp::VolumeConstraint Class Reference

Volume constraint. Limits the volume of a tetrahedron to a range. See [2] for more details.

`#include <Constraint.h>`

Inheritance diagram for ShapeOp::VolumeConstraint:



Public Member Functions

- VolumeConstraint (const std::vector< int > &idI, Scalar weight, const Matrix3X &positions, Scalar range↩ Min=1.0, Scalar rangeMax=1.0)

  Constraint constructor. The target volume is the volume spanned by four vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the volume [rangeMin∗target_↩ volume,rangeMax∗target_volume].
- virtual void project (const Matrix3X &positions, Matrix3X &projections) const overridefinal

  Find the closest configuration from the input positions that satisfy the constraint.

- virtual void addConstraint (std::vector< Triplet > &triplets, int &idO) const overridefinal

  Add the constraint to the linear system.
- void setRangeMin (Scalar rMin)

  Set a new range minimum.
- void setRangeMax (Scalar rMax)

  Set a new range maximum.

Additional Inherited Members

11.27.1 Detailed Description

Volume constraint. Limits the volume of a tetrahedron to a range. See [2] for more details.

11.27.2 Constructor & Destructor Documentation

11.27.2.1 SHAPEOP_INLINE ShapeOp::VolumeConstraint::VolumeConstraint (const std::vector< int > & *idI*, Scalar *weight*, const Matrix3X & *positions*, Scalar *rangeMin* = 1.0, Scalar *rangeMax* = 1.0)

Constraint constructor. The target volume is the volume spanned by four vertices in the parameter positions. The parameters rangeMin and rangeMax can be used to specify a target range for the volume [rangeMin∗target_↩ volume,rangeMax∗target_volume].

Parameters

idI	are four indices of the vertices of the tetrahedron
weight	The weight of the constraint to be added relative to the other constraints.
positions	The positions of all the n vertices stacked in a 3 by n matrix.
rangeMin	The factor to determine the minimal volume: rangeMin∗target_volume.
rangeMax	The factor to determine the maximal volume: rangeMax∗target_volume.

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

- libShapeOp/src/Constraint.h
- libShapeOp/src/Constraint.cpp

Chapter 12

File Documentation

12.1 libShapeOp/api/API.h File Reference

```
#include "Common.h"
```

Typedefs

- typedef struct ShapeOpSolver ShapeOpSolver

  C structure that containts the C++ ShapeOp::Solver.
- typedef enum shapeop_err shapeop_err

  ShapeOp Success and Error type. This list might be extended. To simply test for errors, use =! SO_SUCCESS.

Enumerations

- enum shapeop_err { **SO_SUCCESS** = 0, SO_INVALID_CONSTRAINT_TYPE = 1, SO_INVALID_ARGUMENT_LENGTH = 2, SO_UNMATCHING_CONSTRAINT_ID = 3 }

  ShapeOp Success and Error type. This list might be extended. To simply test for errors, use =! SO_SUCCESS.

Functions

- SHAPEOP_API ShapeOpSolver * shapeop_create ()

  Create the ShapeOp solver. For more details see ShapeOp::Solver.
- SHAPEOP_API void shapeop_delete (ShapeOpSolver *op)

  Delete the ShapeOp solver. For more details see ShapeOp::Solver.
- SHAPEOP_API int shapeop_init (ShapeOpSolver *op)

  Initialize the ShapeOp solver for static geometry processing. For more details see ShapeOp::Solver.
- SHAPEOP_API int shapeop_initDynamic (ShapeOpSolver *op, ShapeOpScalar masses, ShapeOpScalar damping, ShapeOpScalar timestep)

  Initialize the ShapeOp solver for dynamic geometry processing. For more details see ShapeOp::Solver.
- SHAPEOP_API int shapeop_solve (ShapeOpSolver *op, unsigned int iteration)

  Run the optimization. For more details see ShapeOp::Solver.
- SHAPEOP_API void shapeop_setPoints (ShapeOpSolver *op, ShapeOpScalar *points, int nb_points)

  Set the vertices to the ShapeOp solver. For more details see ShapeOp::Solver.
- SHAPEOP_API void shapeop_getPoints (ShapeOpSolver *op, ShapeOpScalar *points, int nb_points)

  Get the vertices back from the ShapeOp solver. For more details see ShapeOp::Solver.
- SHAPEOP_API void shapeop_setTimeStep (ShapeOpSolver *op, ShapeOpScalar timestep)

Set the timestep of the ShapeOp solver. For more details see ShapeOp::Solver.

- SHAPEOP_API void shapeop_setDamping (ShapeOpSolver *op, ShapeOpScalar damping)

  Set the damping of the ShapeOp solver. For more details see ShapeOp::Solver.
- SHAPEOP_API int shapeop_addConstraint (ShapeOpSolver *op, const char *constraintType, int *ids, int nb_ids, ShapeOpScalar weight)

  Add a constraint to the ShapeOp solver.
- SHAPEOP_API shapeop_err shapeop_editConstraint (ShapeOpSolver *op, const char *constraintType, int constraint_id, const ShapeOpScalar *scalars, int nb_scl)

  Add a constraint to the ShapeOp solver.
- SHAPEOP_API int shapeop_addGravityForce (ShapeOpSolver *op, ShapeOpScalar *force)

  Add a gravity force to the ShapeOp solver. For more details see ShapeOp::GravityForce.
- SHAPEOP_API int shapeop_addVertexForce (ShapeOpSolver *op, ShapeOpScalar *force, int id)

  Add a vertex force to the ShapeOp solver. For more details see ShapeOp::VertexForce.
- SHAPEOP_API void shapeop_editVertexForce (ShapeOpSolver *op, int force_id, ShapeOpScalar *force, int id)

  Edit a vertex force previously added to the ShapeOp solver. For more details see ShapeOp::VertexForce.

12.1.1 Detailed Description

This file implements a C API for the ShapeOp C++ library.

To use the library you need to:

1) Create the solver with shapeop_create

2) Set the vertices with shapeop_setPoints

3) Setup the constraints and forces with shapeop_addConstraint and shapeop_editConstraint

4) Initalize the solver with shapeop_init or shapeop_initDynamic

5) Optimize with shapeop_solve

6) Get back the vertices with shapeop_getPoints

7) Delete the solver with shapeop_delete

12.1.2 Enumeration Type Documentation

12.1.2.1 enum shapeop_err

ShapeOp Success and Error type. This list might be extended. To simply test for errors, use =! SO_SUCCESS.

Enumerator

SO_INVALID_CONSTRAINT_TYPE ShapeOp Success type indicating that no error happened.

SO_INVALID_ARGUMENT_LENGTH ShapeOp Error type indicating an invalid constraint type provided.

SO_UNMATCHING_CONSTRAINT_ID ShapeOp Error type indicating an invalid length of an array argument.

12.1.3 Function Documentation

12.1.3.1 SHAPEOP_API int shapeop_addConstraint (ShapeOpSolver * *op*, const char * *constraintType*, int * *ids*, int *nb_ids*, ShapeOpScalar *weight*)

Add a constraint to the ShapeOp solver.

Parameters

op	The ShapeOp Solver object
constraintType	A c-style string containing one of the following constraint types listed in the documentation of ShapeOp::Constraint::shapeConstraintFactory: • "EdgeStrain", takes 2 indices forming an edge. See ShapeOp::EdgeStrainConstraint • "TriangleStrain", takes 3 indices forming a triangle. See ShapeOp::TriangleStrainConstraint • "TetrahedronStrain", takes 4 indices forming a tetrahedron. See ShapeOp::TetrahedronStrainConstraint • "Area", takes 3 indices forming a triangle. See ShapeOp::AreaConstraint • "Volume", takes 4 indices forming a tetrahedron. See ShapeOp::VolumeConstraint • "Bending", takes 4 indices of two neighboring triangles sharing an edge. See ShapeOp::BendingConstraint • "Closeness", takes 1 index. See ShapeOp::ClosenessConstraint • "Line", takes 2 or more indices. See ShapeOp::LineConstraint • "Plane", takes 3 or more indices. See ShapeOp::PlaneConstraint • "Circle", takes 3 or more indices. See ShapeOp::CircleConstraint • "Sphere", takes 4 or more indices. See ShapeOp::SphereConstraint • "Similarity", takes 1 or more indices. See ShapeOp::SimilarityConstraint with scaling • "Rigid", takes 1 or more indices. See ShapeOp::SimilarityConstraint without scaling • "Rectangle", takes 4 indices. See ShapeOp::RectangleConstraint • "Parallelogram", takes 4 indices. See ShapeOp::ParallelogramConstraint • "Laplacian", takes 2 or more indices, center vertex first, then the one ring neighborhood. See ShapeOp::UniformLaplacianConstraint • "LaplacianDisplacement", takes 2 or more indices, center vertex first, then the one ring neighborhood. See ShapeOp::UniformLaplacianConstraint of deformation. • "Angle", takes 3 indices forming two consecutive edges. See ShapeOp::AngleConstraint.

ids	The array of indices of the points subject to the constraint. See documentation of the corresponding constraint in Constraint.h
nb_ids	The length or number of indices in ids.
weight	The weight of the constraint to be added relative to the other constraints in the ShapeOp↩ Solver.

Returns

constraint index in the solver. -1 if adding failed.

12.1.3.2 SHAPEOP_API int shapeop_addGravityForce (ShapeOpSolver * *op*, ShapeOpScalar * *force*)

Add a gravity force to the ShapeOp solver. For more details see ShapeOp::GravityForce.

Parameters

op	The ShapeOp Solver object
force	A c-style array of 3 ShapeOpScalar's specifying the force vector in each dimension

12.1.3.3 SHAPEOP_API shapeop_err shapeop_editConstraint (ShapeOpSolver * *op*, const char * *constraintType*, int *constraint_id*, const ShapeOpScalar * *scalars*, int *nb_scl*)

Add a constraint to the ShapeOp solver.

Parameters

op	The ShapeOp Solver object
constraintType	A c-style string containing one of the following constraint types • "EdgeStrain", see ShapeOp::EdgeStrainConstraint. The scalars have 3 entries: (1) the desired distance between the two constrained vertices; (2) rangeMin; (3) rangeMax. • "TriangleStrain", see ShapeOp::TriangleStrainConstraint. The scalars have 2 entries: (1) rangeMin; (2) rangeMax. • "TetrahedronStrain", see ShapeOp::TetrahedronStrainConstraint. The scalars have 2 entries: (1) rangeMin; (2) rangeMax. • "Area", see ShapeOp::AreaConstraint. The scalars have 2 entries: (1) rangeMin; (2) rangeMax. • "Volume", see ShapeOp::VolumeConstraint. The scalars have 2 entries: (1) rangeMin; (2) rangeMax. • "Bending", see ShapeOp::BendingConstraint. The scalars have 2 entries: (1) rangeMin; (2) rangeMax. • "Closeness", see ShapeOp::ClosenessConstraint. The scalars have 3 entries, which are the desire coordinates of constrained vertex. • "Similarity", see ShapeOp::SimilarityConstraint with scaling. The scalars have $3 * n * m$ entries, where m is the number of candidate shapes that the constrained vertices should be similar to, and n is the number of points in each candidate shape (the same as the number of constrained vertices). Each block of $3 * n$ scalars provides the point coordinates of a candidate shape. • "Rigid", see ShapeOp::SimilarityConstraint without scaling. Same as for "Similarity", see above. • "Angle", see ShapeOp::AngleConstraint. The scalars have 2 entries: (1) minAngle; (2) maxAngle.
constraint_id	The id of the constraint, which is returned in shapeop_addConstraint.
scalars	A c-style array of ShapeOpScalar's. See documentation of the corresponding constraint in Constraint.h
nb_scl	The length of the array scalars.

Returns

See shapeop_err

12.1.3.4 SHAPEOP_API int shapeop_init (ShapeOpSolver * *op*)

Initialize the ShapeOp solver for static geometry processing. For more details see ShapeOp::Solver.

Returns

0 on success, 1 otherwise

12.1.3.5 SHAPEOP_API int shapeop_initDynamic (ShapeOpSolver * *op*, ShapeOpScalar *masses*, ShapeOpScalar *damping*, ShapeOpScalar *timestep*)

Initialize the ShapeOp solver for dynamic geometry processing. For more details see ShapeOp::Solver.

Returns

0 on success, 1 otherwise

12.1.3.6 SHAPEOP_API int shapeop_solve (ShapeOpSolver * *op*, unsigned int *iteration*)

Run the optimization. For more details see ShapeOp::Solver.

Returns

0 on success, 1 otherwise

12.2 libShapeOp/src/Common.h File Reference

Macros

- #define SHAPEOP_SCALAR double

  Defines the scalar type used by the ShapeOp solver (float or double).
- #define SHAPEOP_API

  Defines the API prefix for the current platform.
- #define SHAPEOP_INLINE

  A macro to inline all code which produces a header-only library.

Typedefs

- typedef SHAPEOP_SCALAR ShapeOpScalar

  Defines the scalar type used by the ShapeOp solver (float or double).

12.2.1 Detailed Description

This file is used to define macros and typedefs used by the C++ code and the C API.

12.3 libShapeOp/src/Constraint.h File Reference

```
#include "Types.h"
#include <memory>
#include <cmath>
```

Classes

- class ShapeOp::Constraint

  Base class of any constraints. This class defines the interface of a ShapeOp constraint.
- class ShapeOp::EdgeStrainConstraint

  Edge strain constraint. Constrains the distance between two points to a range. See [2] for more details.
- class ShapeOp::TriangleStrainConstraint

  A mesh-independent triangle strain-limiting constraint. See [2] for more details.
- class ShapeOp::TetrahedronStrainConstraint

  A mesh-independent tetrahedron strain-limiting constraint. See [2] for more details.
- class ShapeOp::AreaConstraint

Area constraint. Limits the area of a triangle to a range. See [2] for more details.

- class ShapeOp::VolumeConstraint

  Volume constraint. Limits the volume of a tetrahedron to a range. See [2] for more details.
- class ShapeOp::BendingConstraint

  Bending constraint. Limits the bending between two neighboring triangles. See [2] for more details.
- class ShapeOp::ClosenessConstraint

  Closeness constraint. Constrains a vertex to a position in space. Projects onto a given rest position.
- class ShapeOp::LineConstraint

  Line constraint. Constrains a set of vertices to lie on the same line. See [1] for more details.
- class ShapeOp::PlaneConstraint

  Plane constraint. Constrains a set of vertices to lie on the same plane. See [1] for more details.
- class ShapeOp::CircleConstraint

  Circle constraint. Constrains a set of vertices to lie on the same circle. See [1] for more details.
- class ShapeOp::SphereConstraint

  Sphere constraint. Constrains a set of vertices to lie on a sphere. See [1] for more details.
- class ShapeOp::SimilarityConstraint

  Similarity or Rigid constraint. Perserves the relative location of a set of vertices. See [1] for more details.
- class ShapeOp::RectangleConstraint

  Rectangle constraint. Constrains the four vertices of a quad to be a rectangle. See [1] for more details.
- class ShapeOp::ParallelogramConstraint

  ParallelogramConstraint constraint. Constrains the four vertices of a quad to be a parallelogram. See [1] for more details.
- class ShapeOp::UniformLaplacianConstraint

  Uniform Laplacian Constraint. Minimizes the uniform laplacian respectively the uniform laplacian of displacements.
- class ShapeOp::AngleConstraint

  Angle range constraint. Constrains the angle between the two lines formed by three points to a range. See [3] for more details.

Namespaces

- ShapeOp

  Namespace of the ShapeOp library.

Macros

- #define M_PI 3.14159265358979323846

12.3.1 Detailed Description

This file containts all the constraints of the ShapeOp libary.

12.3.2 Macro Definition Documentation

12.3.2.1 #define M_PI 3.14159265358979323846

Defining pi.

12.4 libShapeOp/src/Force.h File Reference

```
#include "Types.h"
```

Classes

- class ShapeOp::Force

  Base class of any forces. This class defines interface of a ShapeOp force.
- class ShapeOp::GravityForce

  This class defines a constant force for all vertices.
- class ShapeOp::VertexForce

  This class defines a constant force for a unique vertex.

Namespaces

- ShapeOp

  Namespace of the ShapeOp library.

12.4.1 Detailed Description

This file contains all the forces (in fact defined as accelerations) of the ShapeOp library.

12.5 libShapeOp/src/LSSolver.h File Reference

```
#include "Types.h"
#include <iostream>
#include <unsupported/Eigen/src/IterativeSolvers/MINRES.h>
```

Classes

- class ShapeOp::LSSolver

  Base class of any sparse linear system solver. This class defines the main functionalities of the ShapeOp sparse linear system solvers (Ax = b).
- class ShapeOp::SimplicialLDLTSolver

  Sparse linear system solver based on Cholesky. This class implements a sparse linear system solver based on the Cholesky LDL^T algorithm from Eigen.
- class ShapeOp::CGSolver

  Sparse linear system solver based on CG. This class implements a sparse linear system solver based on the CG algorithm from Eigen.
- class ShapeOp::MINRESSolver

  Sparse linear system solver based on MINRES. This class implements a sparse linear system solver based on the MINRES algorithm from Eigen.
- class ShapeOp::SORSolver

  Sparse linear system solver based on successive over-relaxation (SOR).

Namespaces

- ShapeOp

  Namespace of the ShapeOp library.

12.5.1 Detailed Description

This file contains all the linear system solvers of the ShapeOp library.

12.6 libShapeOp/src/Solver.h File Reference

```
#include <vector>
#include <memory>
#include "Types.h"
```

Classes

- class ShapeOp::Solver

  ShapeOp Solver. This class implements the main ShapeOp solver based on [1] and [2].

Namespaces

- ShapeOp

  Namespace of the ShapeOp library.

12.6.1 Detailed Description

This file contains the main ShapeOp solver.

12.7 libShapeOp/src/Types.h File Reference

```
#include "Common.h"
#include <Eigen/Dense>
#include <Eigen/Sparse>
```

Namespaces

- ShapeOp

  Namespace of the ShapeOp library.

Macros

- #define SHAPEOP_ALIGNMENT Eigen::AutoAlign

  Defines Eigen Alignment type.

Typedefs

- typedef ShapeOpScalar ShapeOp::Scalar

  A scalar type, double or float, as defined in ShapeOpScalar in Common.h.
- template<int Rows, int Cols, int Options = (Eigen::ColMajor | SHAPEOP_ALIGNMENT)>
  using ShapeOp::MatrixT = Eigen::Matrix< Scalar, Rows, Cols, Options >

  A typedef of the dense matrix of Eigen.
- typedef MatrixT< 2, 1 > ShapeOp::Vector2

  A 2d column vector.
- typedef MatrixT< 2, 2 > ShapeOp::Matrix22

  A 2 by 2 matrix.

- typedef MatrixT< 2, 3 > ShapeOp::Matrix23

  A 2 by 3 matrix.
- typedef MatrixT< 3, 1 > ShapeOp::Vector3

  A 3d column vector.
- typedef MatrixT< 3, 2 > ShapeOp::Matrix32

  A 3 by 2 matrix.
- typedef MatrixT< 3, 3 > ShapeOp::Matrix33

  A 3 by 3 matrix.
- typedef MatrixT< 3, 4 > ShapeOp::Matrix34

  A 3 by 4 matrix.
- typedef MatrixT< 4, 1 > ShapeOp::Vector4

  A 4d column vector.
- typedef MatrixT< 4, 4 > ShapeOp::Matrix44

  A 4 by 4 matrix.
- typedef MatrixT< 3, Eigen::Dynamic > ShapeOp::Matrix3X

  A 3 by n matrix.
- typedef MatrixT< Eigen::Dynamic, 3 > ShapeOp::MatrixX3

  A n by 3 matrix.
- typedef MatrixT< Eigen::Dynamic, 1 > ShapeOp::VectorX

  A nd column vector.
- typedef MatrixT< Eigen::Dynamic, Eigen::Dynamic > ShapeOp::MatrixXX

  A n by m matrix.
- template<int Options = Eigen::ColMajor>
  using ShapeOp::SparseMatrixT = Eigen::SparseMatrix< Scalar, Options >

  A typedef of the sparse matrix of Eigen.
- typedef SparseMatrixT ShapeOp::SparseMatrix

  The default sparse matrix of Eigen.
- typedef Eigen::Triplet< Scalar > ShapeOp::Triplet

  A triplet, used in the sparse triplet representation for matrices.

12.7.1 Detailed Description

This file redefines EIGEN types using the scalar type ShapeOpScalar defined in Common.h.

Bibliography

[1] Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. Shape-up: Shaping discrete geometry with projections. *Comp. Graph. Forum*, 2012. (pdf). 1, 17, 18, 21, 22, 30, 38, 40, 41, 42, 43, 44, 45, 48, 61, 63

[2] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 2014. (pdf). 1, 17, 18, 21, 22, 27, 28, 29, 36, 45, 49, 50, 52, 53, 60, 61, 63

[3] Bailin Deng, Sofien Bouaziz, Mario Deuss, Alexandre Kaspar, Yuliy Schwartzburg, and Mark Pauly. Interactive design exploration for constrained meshes. *Computer-Aided Design*, 61:13 – 23, 2015. (pdf). 17, 21, 25, 61

Index

API.h
    SO_INVALID_ARGUMENT_LENGTH, 56
    SO_INVALID_CONSTRAINT_TYPE, 56
    SO_UNMATCHING_CONSTRAINT_ID, 56
    shapeop_addConstraint, 56
    shapeop_addGravityForce, 58
    shapeop_editConstraint, 58
    shapeop_err, 56
    shapeop_init, 59
    shapeop_initDynamic, 59
    shapeop_solve, 60
addConstraint
    ShapeOp::Constraint, 33
AngleConstraint
    ShapeOp::AngleConstraint, 26
AreaConstraint
    ShapeOp::AreaConstraint, 28

BendingConstraint
    ShapeOp::BendingConstraint, 29

CircleConstraint
    ShapeOp::CircleConstraint, 30
ClosenessConstraint
    ShapeOp::ClosenessConstraint, 31
Constraint
    ShapeOp::Constraint, 33
Constraint.h
    M_PI, 61

EdgeStrainConstraint
    ShapeOp::EdgeStrainConstraint, 37

initialize
    ShapeOp::Solver, 46

libShapeOp/api/API.h, 55
libShapeOp/src/Common.h, 60
libShapeOp/src/Constraint.h, 60
libShapeOp/src/Force.h, 61
libShapeOp/src/LSSolver.h, 62
libShapeOp/src/Solver.h, 63
libShapeOp/src/Types.h, 63
LineConstraint
    ShapeOp::LineConstraint, 39

M_PI
    Constraint.h, 61

ParallelogramConstraint
    ShapeOp::ParallelogramConstraint, 41
PlaneConstraint
    ShapeOp::PlaneConstraint, 42
project
    ShapeOp::Constraint, 33

RectangleConstraint
    ShapeOp::RectangleConstraint, 42

SO_INVALID_ARGUMENT_LENGTH
    API.h, 56
SO_INVALID_CONSTRAINT_TYPE
    API.h, 56
SO_UNMATCHING_CONSTRAINT_ID
    API.h, 56
SORSolver
    ShapeOp::SORSolver, 47
setShapes
    ShapeOp::SimilarityConstraint, 44
shapeConstraintFactory
    ShapeOp::Constraint, 35
ShapeOp, 21
ShapeOp::AngleConstraint, 25
    AngleConstraint, 26
ShapeOp::AreaConstraint, 27
    AreaConstraint, 28
ShapeOp::BendingConstraint, 28
    BendingConstraint, 29
ShapeOp::CGSolver, 29
ShapeOp::CircleConstraint, 30
    CircleConstraint, 30
ShapeOp::ClosenessConstraint, 31
    ClosenessConstraint, 31
ShapeOp::Constraint, 32
    addConstraint, 33
    Constraint, 33
    project, 33
    shapeConstraintFactory, 35
ShapeOp::EdgeStrainConstraint, 36
    EdgeStrainConstraint, 37
ShapeOp::Force, 37
ShapeOp::GravityForce, 37
ShapeOp::LSSolver, 39
ShapeOp::LineConstraint, 38
    LineConstraint, 39
ShapeOp::MINRESSolver, 39
ShapeOp::ParallelogramConstraint, 40
    ParallelogramConstraint, 41
ShapeOp::PlaneConstraint, 41
    PlaneConstraint, 42
ShapeOp::RectangleConstraint, 42

RectangleConstraint, 42
ShapeOp::SORSolver, 46
SORSolver, 47
ShapeOp::SimilarityConstraint, 43
setShapes, 44
SimilarityConstraint, 44
ShapeOp::SimplicialLDLTSolver, 44
ShapeOp::Solver, 45
initialize, 46
solve, 46
ShapeOp::SphereConstraint, 48
SphereConstraint, 48
ShapeOp::TetrahedronStrainConstraint, 49
TetrahedronStrainConstraint, 49
ShapeOp::TriangleStrainConstraint, 50
TriangleStrainConstraint, 50
ShapeOp::UniformLaplacianConstraint, 51
UniformLaplacianConstraint, 51
ShapeOp::VertexForce, 51
ShapeOp::VolumeConstraint, 52
VolumeConstraint, 53
ShapeOpSolver, 43
shapeop_addConstraint
API.h, 56
shapeop_addGravityForce
API.h, 58
shapeop_editConstraint
API.h, 58
shapeop_err
API.h, 56
shapeop_init
API.h, 59
shapeop_initDynamic
API.h, 59
shapeop_solve
API.h, 60
SimilarityConstraint
ShapeOp::SimilarityConstraint, 44
solve
ShapeOp::Solver, 46
SphereConstraint
ShapeOp::SphereConstraint, 48

TetrahedronStrainConstraint
ShapeOp::TetrahedronStrainConstraint, 49
TriangleStrainConstraint
ShapeOp::TriangleStrainConstraint, 50

UniformLaplacianConstraint
ShapeOp::UniformLaplacianConstraint, 51

VolumeConstraint
ShapeOp::VolumeConstraint, 53